Ατομική Διπλωματική Εργασία

ΙΕΡΑΡΧΙΚΗ ΕΝΙΣΧΥΤΙΚΗ ΜΑΘΗΣΗ ΣΕ ΠΡΟΒΛΗΜΑΤΑ ΕΝΟΣ ΠΡΑΚΤΟΡΑ ΚΑΙ ΠΟΛΛΑΠΛΩΝ ΠΡΑΚΤΟΡΩΝ

Ιωάννης Λάμπρου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιεραρχική Ενισχυτική Μάθηση σε προβλήματα
Ενός Πράκτορα και Πολλαπλών Πρακτόρων

Ιωάννης Λάμπρου

Επιβλέπων Καθηγητής
Δρ. Χρίστος Χριστοδούλου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επόπτη μου, Δρ. Χρίστο Χριστοδούλου, που με εμπιστεύθηκε για την εκπόνηση αυτής της διπλωματικής εργασίας και για την εξαιρετική συνεργασία και την βοήθεια που μου πρόσφερε καθοδηγώντας με καθ' όλη την διάρκεια της εργασίας. Επίσης θα ήθελα να ευχαριστήσω τον διδακτορικό φοιτητή Βασίλη Βασιλειάδη για τη συνεχή υποστήριξη και την πολύτιμη βοήθεια που μου πρόσφερε.

Τέλος θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου, η οποία πάντα μου πρόσφερε υποστήριξη και με ενθάρρυνε να προχωρώ μπροστά.

Περίληψη

Η διπλωματική αυτή εργασία έχει σαν στόχο να εξερευνήσει κατά πόσο οι Ιεραρχικές Μεθόδοι επιταχύνουν την μάθηση και βοηθούν στην αντιμετώπιση προβλημάτων με μεγάλο σύνολο καταστάσεων-ενεργειών (state-action space), τόσο σε προβλήματα Ενισχυτικής Μάθησης Ενός Πράκτορα (ΕΜΕΠ) όσο και σε προβλήματα Ενισχυτικής Μάθησης Πολλαπλών Πρακτόρων (ΕΜΠΠ). Το πλαίσιο που χρησιμοποιήθηκε είναι το «Taxi Problem» (ΤΡ) και το «Εκτεταμένο Taxi Problem» (ΕΡΤ) σε προβλήματα ΕΜΕΠ και το «Multi Agent Taxi Problem» (ΜΑΤΡ), το οποίο είναι μια επέκταση του απλού ΤΡ, σε προβλήματα ΕΜΠΠ. Αρχικά δοκιμάσαμε κάποιους αλγόριθμους σε προβλήματα ΕΜΕΠ και συνεχίσαμε δοκιμάζοντας τους σε προβλήματα ΕΜΠΠ. Στα προβλήματα ΕΜΕΠ δοκιμάσαμε τους αλγόριθμους Χρονικών Διαφορών (ΧΔ) Q-Learning και SARSA, χρησιμοποιώντας την ε-greedy πολιτική και την πολιτική Boltzmann, τον αλγόριθμο Policy Hill-Climbing (PHC), ο οποίος είναι μια επέκταση του αλγόριθμου Q-Learning, τον αλγόριθμο WoLF-Policy Hill-Climbing (WoLF-PHC), ο οποίος είναι μια απλή επέκταση του αλγόριθμου PHC, τον Ιεραρχικό αλγόριθμο MAXQ-Q και τους Ιεραρχικούς αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC οι οποίοι είναι μια Ιεραρχική επέκταση των αλγόριθμων PHC και WoLF-PHC αντίστοιχα, και τους οποίους αποφασίσαμε να υλοποιήσουμε, αφού εξ όσον γνωρίζουμε δεν υπάρχουν μέχρι τώρα στην βιβλιογραφία. Από τα αποτελέσματα που πήραμε παρατηρήσαμε ότι οι πράκτορες εκπαιδεύονται ικανοποιητικά με όλους τους αλγόριθμους. Επίσης παρατηρήσαμε ότι με την χρησιμοποίηση των Ιεραρχικών αλγορίθμων MAXQ-Q, MAXQ-PHC και MAXQ-WoLF-PHC η μάθηση επιταχύνεται σημαντικά. Στα προβλήματα ΕΜΠΠ δοκιμάσαμε τον αλγόριθμο ΧΔ Q-Learning, χρησιμοποιώντας την ε-greedy πολιτική και τους αλγόριθμους PHC και WoLF-PHC, όπου συγκρίνοντας τα αποτελέσματα τους είδαμε ότι όταν οι πράκτορες χρησιμοποιούν τον αλγόριθμο Q-Learning δεν εκπαιδεύονται ικανοποιητικά σε σχέση με όταν χρησιμοποιούν τους άλλους δύο αλγόριθμους. Με την εφαρμογή των Ιεραρχικών αλγορίθμων MAXQ-PHC και MAXQ-WoLF-PHC παρατηρήσαμε ότι η μάθηση επιταχύνεται σημαντικά σε σχέση με τους αλγόριθμους PHC και WoLF-PHC.

Περιεχόμενα

Λίστα Ακρωνυμίων

ΔΠ	Δυναμικός Προγραμματισμός
ΧΔ	Χρονική Διαφορά
ΕΜ	Ενισχυτική Μάθηση
ΕΜΕΠ	Ενισχυτική Μάθηση Ενός Πράκτορα
ΕΜΠΠ	Ενισχυτική Μάθηση Πολλαπλών Πρακτόρων
ΙΕΜ	Ιεραρχική Ενισχυτική Μάθηση
ΜΔΑ	Μαρκοβιανή Διαδικασία Απόφασης
TP	Taxi Problem
ETP	Extended Taxi Problem
MATP	Multi Agent Taxi Problem

Κεφάλαιο 1

Εισαγωγή

Η Ενισχυτική Μάθηση (ΕΜ) υπήρξε μια ενεργή ερευνητική περιοχή της Τεχνητής Νοημοσύνης για πολλά χρόνια. Βασίζεται στην ιδέα του ότι η τάση επιλογής μιας ενέργειας πρέπει να ενδυναμώνεται (ενισχύεται) αν παράγει ευνοϊκά αποτελέσματα και να αποδυναμώνεται αν παράγει δυσμενή αποτελέσματα. Γι' αυτό το λόγο μπορεί να χαρακτηριστεί επίσης ως «μάθηση μετ' εμπειρίας», αφού ο εκπαιδευόμενος δεν ξέρει από πριν ποιες ενέργειες να επιλέξει, αλλά αντιθέτως πρέπει να ανακαλύψει ποιες ενέργειες οδηγούν στη μεγαλύτερη αμοιβή, με το να τις δοκιμάσει.

Σε αρκετά προβλήματα ΕΜ υπάρχει ένα μεγάλο σύνολο καταστάσεων-ενεργειών (state-action space). Σε γενικές γραμμές είναι δύσκολο να καθοριστεί το κατάλληλο σύνολο καταστάσεων και ενεργειών σε όλα τα προβλήματα ΕΜ του πραγματικού κόσμου. Τις περισσότερες φορές απαιτείται αρκετά μεγάλη χωρητικότητα μνήμης για να καλύψει όλες τις ενδεχόμενες σχετικές καταστάσεις και ενέργειες που μπορεί να επιλεγούν, καθώς μπορεί να υπάρχει μια ευρεία ποικιλία καταστάσεων και ενεργειών που μπορούν να επιλεγούν για κάθε κατάσταση. Επίσης σε κάποια προβλήματα χρειάζεται απεριόριστη μνήμη, για παράδειγμα όταν έχουμε συνεχής σύνολα καταστάσεων-ενεργειών. Κατά συνέπεια υπάρχει ένα αρκετά μεγάλο πρόβλημα χώρου κατά την προσπάθεια να διερευνηθούν όλες οι πιθανές ενέργειες από όλες τις πιθανές καταστάσεις. Ακόμη, υπάρχει και αρκετό πρόβλημα χρόνου αφού όσο μεγαλώνει το σύνολο καταστάσεων-ενεργειών ενός προβλήματος, τόσο περισσότερη εκπαίδευση χρειάζεται ο πράκτορας μέχρι να επέλθει η μάθηση. Για την επίλυση αυτού του προβλήματος χρησιμοποιούνται διάφορες μεθόδοι όπως οι Ιεραρχικές Μεθόδοι και οι Τεχνικές Προσέγγισης Συναρτήσεων.

Έτσι στόχος αυτής της διπλωματικής εργασίας είναι να παρουσιάσει μια ανασκόπηση της έρευνας που έχει γίνει στο πεδίο των Ιεραρχικών Μεθόδων και να δούμε κατά πόσο

οι Ιεραρχικές Μεθόδοι επιταχύνουν την μάθηση και επιλύουν ως ένα βαθμό αυτό το πρόβλημα, τόσο σε προβλήματα ΕΜΕΠ όσο και σε προβλήματα ΕΜΠΠ.

Το πλαίσιο που θα χρησιμοποιηθεί είναι το ΤΡ και το ΕΤΡ σε προβλήματα ΕΜΕΠ και το ΜΑΤΡ, το οποίο είναι μια επέκταση του απλού ΤΡ, σε προβλήματα ΕΜΠΠ. Αρχικά θα υλοποιηθεί το σύστημα που θα προσομοιώνει τους αλγόριθμους ΕΜΕΠ και στη συνέχεια το σύστημα που θα προσομοιώνει τους αλγόριθμους ΕΜΠΠ.

Στο σύστημα ΕΜΕΠ, αρχικά θα υλοποιηθούν οι δύο αλγόριθμοι ΧΔ, Q-Learning και SARSA, χρησιμοποιώντας την ε-greedy πολιτική και την πολιτική Boltzmann. Στη συνέχεια θα υλοποιηθούν οι αλγόριθμοι PHC και WoLF-PHC, οι οποίοι αν και είναι αλγόριθμοι οι οποίοι χρησιμοποιούνται σε περιβάλλοντα ΕΜΠΠ θα είναι ενδιαφέρων να δούμε την συμπεριφορά τους σε grid world προβλήματα ΕΜΕΠ, και θα συγκριθεί η απόδοση τους με την απόδοση του αλγόριθμου Q-Learning. Ακολούθως θα υλοποιηθεί ο Ιεραρχικός αλγόριθμος MAXQ-Q και θα συγκρίνουμε τα αποτελέσματα του με τον αλγόριθμο Q-Learning. Έπειτα, αν οι αλγόριθμοι PHC και WoLF-PHC έχουν ικανοποιητικά αποτελέσματα, θα επιχειρήσουμε να τους υλοποιήσουμε Ιεραρχικά, χρησιμοποιώντας την MAXQ διάσπαση, επεκτείνοντας τον αλγόριθμο MAXQ-Q. Τότε θα συγκρίνουμε τα αποτελέσματα αυτών των Ιεραρχικών αλγορίθμων με τους αλγόριθμους PHC και WoLF-PHC για να δούμε κατά πόσο επιταχύνεται η μάθηση, με την χρήση τους.

Ακολούθως θα ασχοληθούμε με το σύστημα ΕΜΠΠ. Αρχικά θα υλοποιηθεί ο αλγόριθμος ΧΔ Q-Learning, χρησιμοποιώντας την ε-greedy πολιτική και την πολιτική Boltzmann, για να εξετάσουμε την συμπεριφορά του σε περιβάλλοντα ΕΜΠΠ. Στη συνέχεια θα υλοποιηθούν οι αλγόριθμοι PHC και WoLF-PHC και θα συγκριθεί η απόδοση τους με την απόδοση του αλγόριθμου Q-Learning. Ακολούθως θα υλοποιηθεί ο Ιεραρχικός αλγόριθμος MAXQ-Q, ο οποίος θα επεκταθεί για να υλοποιηθούν οι αλγόριθμοι MAXQ-PHC και MAXQ-WoLF-PHC. Τέλος θα συγκρίνουμε τα αποτελέσματα αυτών των αλγορίθμων με τα αποτελέσματα των αλγορίθμων PHC και WoLF-PHC για να δούμε κατά πόσο επιταχύνεται η μάθηση, με την χρήση τους.

Το σύστημα θα υλοποιηθεί με ευέλικτο και εξελίξιμο τρόπο, υποστηρίζοντας την εύκολη και γρήγορη προσθήκη νέων αλγορίθμων ΕΜ, νέων πολιτικών και νέων περιβαλλόντων. Τα αποτελέσματα θα ελεγχθούν και θα συζητηθούν, και στη συνέχεια θα εντοπιστούν πιθανοί τρόποι επέκτασης του συστήματος.

Κεφάλαιο 2

Γνωσιολογικό Υπόβαθρο

2.1 Στοιχεία Ενισχυτικής Μάθησης

Η Ενισχυτική Μάθηση (Kaelbling, Moore και Littman, 1996) είναι μάθηση του πώς να συμπεριφέρομαι σε μια δεδομένη κατάσταση. Μπορεί να περιγραφεί ως «μάθηση μετ' εμπειρίας», αφού δεν λέει κανείς στον εκπαιδευόμενο ποιες ενέργειες να επιλέξει, αλλά πρέπει να ανακαλύψει αυτές που δίνουν την πιο μεγάλη αμοιβή με το να τις δοκιμάσει. Αυτή η μάθηση μετ' εμπειρίας μπορεί να διαρκέσει καθ' όλη τη διάρκεια ζωής του εκπαιδευομένου.

Ο εκπαιδευόμενος ή αυτός που παίρνει αποφάσεις, όπως π.χ. ένας παίκτης ή ένα ρομπότ, ορίζεται ως πράκτορας. Μια ενέργεια ορίζει τι μπορεί να κάνει ο πράκτορας σε μια δεδομένη κατάσταση, όπως για παράδειγμα τι κινήσεις να παίξει πάνω στην σκακιέρα. Ένα επεισόδιο μπορεί να οριστεί ως μία σειρά από ενέργειες που εκτελούνται από τον πράκτορα μέχρι να επιτύχει τον στόχο του. Ο στόχος είναι η

μεγιστοποίηση της αναμενόμενης ολικής αμοιβής μετά από κάποια χρονική περίοδο, όπως για παράδειγμα, να κερδίσω την μέγιστη αμοιβή μετά από 100 επιλογές ενεργειών. Ο πράκτορας δεν μπορεί να αλλάξει ή να επηρεάσει τον στόχο του. Για παράδειγμα, στο σκάκι ο στόχος κάποιου παίκτη είναι να κάνει ματ στον αντίπαλό του και έτσι να κερδίσει το παιγνίδι. Η επιτυχία καθορίζεται από το πόσο καλά ή κακά παίζει ο αντίπαλος.

Κάθε ενέργεια έχει κάποια αναμενόμενη αμοιβή αν επιλεχθεί, η οποία ονομάζεται ως αξία της ενέργειας αυτής. Υπάρχουν πολλοί τρόποι για να εκτιμηθεί η αξία μιας ενέργειας. Στην Ενισχυτική Μάθηση ο σκοπός είναι να μάθουμε να εκτιμούμε ή να προβλέπουμε την αξία κάποιας ενέργειας με ακρίβεια. Η μάθηση μπορεί να περιγραφεί ως μη-συσχετιζόμενη (non-associative), που ορίζεται ως μάθηση του πως να συμπεριφερόμαστε σε μια δεδομένη κατάσταση, ή συσχετιζόμενη κατά την οποία υπάρχει μια απαίτηση για συσχέτιση διαφορετικών ενεργειών με διαφορετικές καταστάσεις. Επιλογή-ενέργειας ορίζεται ως το πρόβλημα της επιλογής μιας ενέργειας που είναι κατάλληλη για την παρούσα κατάσταση.

Ένα από τα κυριότερα χαρακτηριστικά της Ενισχυτικής Μάθησης είναι το δίλημμα του πότε να εκμεταλλευτούμε την υπάρχουσα γνώση των ενεργειών που θα δώσουν μία άμεση αμοιβή και πότε να εξερευνήσουμε νέες ενέργειες, οι οποίες μπορεί να δώσουν κάποια αμοιβή. Η άπληστη ενέργεια είναι η ενέργεια της οποίας η εκτιμημένη αξία είναι η μεγαλύτερη κάποια δεδομένη στιγμή. Η επιλογή της άπληστης ενέργειας ισοδυναμεί με εκμετάλλευση υπάρχουσας γνώσης. Η επιλογή μιας μη-άπληστης ενέργειας ισοδυναμεί με εξερεύνηση νέων ενεργειών. Η εκμετάλλευση θα μεγιστοποιήσει την άμεση αμοιβή, αλλά η εξερεύνηση μπορεί να παραγάγει τη μεγαλύτερη ολική αμοιβή μακροπρόθεσμα.

Υπάρχουν πολλές μέθοδοι που ασχολούνται με το πρόβλημα της επιλογής ενέργειας. Η άπληστη επιλογή εκμεταλλεύεται υπάρχουσα γνώση και πάντοτε επιλέγει την ενέργεια με τη μεγαλύτερη αξία. Η σχεδόν-άπληστη επιλογή εκτελεί την άπληστη επιλογή στην πλειονότητα των περιπτώσεων, αλλά θα επιλέξει μια μη-άπληστη ενέργεια τυχαία. Μια παραλλαγή αυτού είναι η softmax επιλογή ενέργειας (Sutton και Barto, 1998), όπου

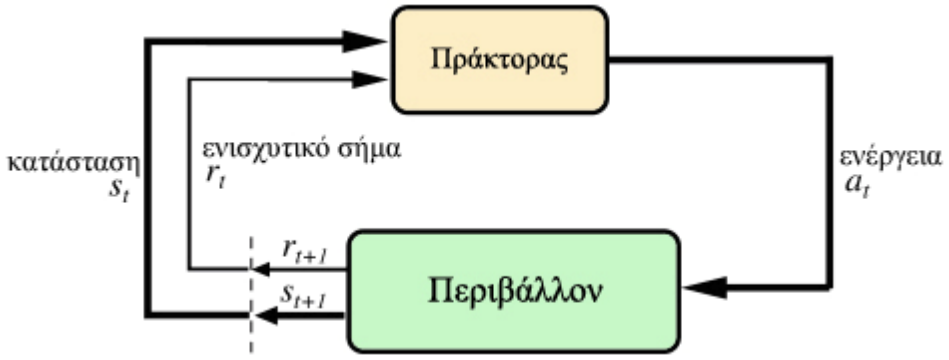
όλες οι ενέργειες κατατάσσονται ανάλογα με την εκτιμημένη τους αξία. Η πιο συνηθισμένη softmax μέθοδος χρησιμοποιεί μια κατανομή Boltzmann. Διαλέγει μια ενέργεια *α*, τη χρονική στιγμή *t* με πιθανότητα:

$$p(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}} \qquad (2.1)$$

όπου τ είναι μια θετική παράμετρος που ονομάζεται θερμοκρασία, $Q_t(a)$ είναι η εκτιμημένη αξία της ενέργειας *α* τη χρονική στιγμή *t* και *n* είναι ο αριθμός των διαθέσιμων ενεργειών.

Τα πιο πάνω περιγράφουν μεθόδους ενέργειας-αξίας για επιλογή ενέργειας, όπου διατηρείται μια εκτίμηση της αξίας κάποιας συγκεκριμένης ενέργειας. Εναλλακτικά, οι μεθόδοι σύγκρισης για επιλογή ενέργειας διατηρούν μια ολική εκτίμηση μιας ενέργειας σε σύγκριση με άλλες ενέργειες. Η επιλογή της μεθόδου εξαρτάται από το πρόβλημα. Όπου οι αληθινές αξίες των ενεργειών αλλάζουν λίγο κατά τη διάρκεια του χρόνου, δεν υπάρχει ανάγκη για περαιτέρω εξερεύνηση από το να δοκιμαστεί κάθε ενέργεια μια φορά. Όπου οι αληθινές αξίες των ενεργειών αλλάζουν κατά τη διάρκεια του χρόνου για κάθε κατάσταση, θα χρειαζόταν περισσότερη εξερεύνηση για να βρεθεί η βέλτιστη ενέργεια κάποια στιγμή.

Το πλαίσιο Ενισχυτικής Μάθησης για ένα πράκτορα συνοψίζεται στο Σχήμα 2.1. Το περιβάλλον είναι οτιδήποτε εκτός από τον πράκτορα. Το όριο μεταξύ του πράκτορα και του περιβάλλοντος μπορεί να είναι ασαφές. Για παράδειγμα, τα αισθητήρια όργανα κάποιου ρομπότ θα θεωρούνταν ως μέρος του περιβάλλοντος. Ο πράκτορας δεν ορίζεται ανάλογα με το όριο του φυσικού του σώματος, αλλά με οτιδήποτε δεν μπορεί να αλλάξει τυχαία. Με άλλα λόγια ο πράκτορας ορίζεται ανάλογα με το όριο του ελέγχου του και όχι με αυτό της γνώσης του.



Σχήμα 2.1: Ένα μοντέλο για ενισχυτική μάθηση ενός πράκτορα

Η αλληλεπίδραση ενός πράκτορα με το περιβάλλον στην ενισχυτική μάθηση, υλοποιημένη στο πλαίσιο Μαρκοβιανής Διαδικασίας Απόφασης. Ένα Περιβάλλον περιέχει οτιδήποτε εξωτερικό του πράκτορα. Μία Κατάσταση είναι μια περίληψη προηγούμενης συμπεριφοράς η οποία χρειάζεται για να καθορίσει μελλοντική συμπεριφορά. Ο Πράκτορας είναι ο εκπαιδευόμενος, για παράδειγμα ο παίκτης ή το Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ). Μία Ενέργεια είναι κάτι που μπορεί να κάνει ο πράκτορας, π.χ. μια κίνηση στη σκακιέρα, η επιλογή ενός μοχλού κτλ. Υπάρχει ένα Ενισχυτικό Σήμα για να αξιολογήσει την τρέχουσα ενέργεια, αλλά δεν λέει στον εκπαιδευόμενο ποια είναι η καλύτερη ενέργεια για να επιλέξει.

Η διεπιφάνεια πράκτορα-περιβάλλοντος αποτελείται από: 1) τη διαδικασία επιλογής μιας ενέργειας από τον πράκτορα, 2) το περιβάλλον που ενισχύει την επιλεχθείσα ενέργεια, 3) ενημέρωση του περιβάλλοντος και 4) παρουσίαση της νέας κατάστασης στον πράκτορα. Ο πράκτορας και το περιβάλλον αλληλεπιδρούν σε μια σειρά από χρονικά βήματα, τα οποία μπορεί να είναι διακριτά ή συνεχή. Η κατάσταση είναι η αναπαράσταση του περιβάλλοντος. Παρέχει τη βάση πάνω στην οποία γίνονται οι επιλογές, αλλά δεν χρειάζεται να ενημερώσει τον πράκτορα για οτιδήποτε σχετικό με το περιβάλλον. Για παράδειγμα, σε ένα παιγνίδι με χαρτιά ο πράκτορας δεν ξέρει τα επόμενα στην τράπουλα πριν να διαμοιραστούν.

Αν η κατάσταση κρατά σχετικές πληροφορίες πάνω σε προηγούμενες καταστάσεις, λέγεται ότι είναι Markov (Bellman, 1957). Έχοντας την ιδιότητα Markov σημαίνει ότι ο πράκτορας μπορεί να προβλέπει την επόμενη κατάσταση και την αναμενόμενη αμοιβή από την τρέχουσα κατάσταση. Στην Ενισχυτική Μάθηση αυτό μπορεί να μην ισχύει πάντοτε, αλλά είναι αρκετό το ότι η κατάσταση είναι μία καλή προσέγγιση. Μια

εργασία (task) Ενισχυτικής Μάθησης, η οποία ικανοποιεί την ιδιότητα Markov ονομάζεται Μαρκοβιανή Διαδικασία Απόφασης (Markov Decision Process - MDP) (ΜΔΑ). Αν τα σύνολα καταστάσεων και ενεργειών είναι πεπερασμένα, τότε η εργασία Ενισχυτικής Μάθησης είναι μια πεπερασμένη ΜΔΑ. Το πλαίσιο ΜΔΑ παρέχει ένα μοντέλο για Ενισχυτική Μάθηση Ενός Πράκτορα, όπως φαίνεται στο Σχήμα 2.1.

Η πολιτική είναι μια συσχέτιση από μια δοθείσα κατάσταση σε μια ενέργεια. Είναι ένας στοχαστικός κανόνας με τον οποίο ένας πράκτορας επιλέγει μια ενέργεια σαν συνάρτηση της κατάστασης του (επιλογή ενέργειας). Ο εκπαιδευόμενος ή πράκτορας ψάχνει για την βέλτιστη πολιτική στην οποία το αναμενόμενο αποτέλεσμα είναι το μεγαλύτερο. Μπορεί να υπάρχουν περισσότερες από μια βέλτιστες πολιτικές. Ο βασικός αλγόριθμος Ενισχυτικής Μάθησης δίνεται στο Σχήμα 2.2.

```
Δημιουργία αρχικού συνόλου καταστάσεων
Εκτίμηση της αξίας κάθε κατάσταση
Για κάθε επεισόδιο
        Αρχικοποίηση τρέχουσας κατάστασης
        Για κάθε βήμα t του επεισοδίου
                Επιλογή μιας ενέργειας
                Εφαρμογή της ενέργειας
                Παρατήρηση ανταμοιβής
                Ενημέρωση της εκτιμημένης αξίας της κατάστασης
                Ανάκτηση επόμενης κατάστασης
        Μέχρι το t να είναι τερματικό
Μέχρι το τέλος του επεισοδίου
```

Σχήμα 2.2: Ο βασικός αλγόριθμος ενισχυτικής μάθησης – μάθησης μετ' εμπειρίας

Το πρώτο βήμα είναι να καθοριστεί η κατάσταση από την οποία θα αρχίσουμε. Σε κάθε βήμα, ο πράκτορας επιλέγει μια ενέργεια βασισμένος πάνω στην τρέχουσα κατάσταση. Μετά εφαρμόζει την ενέργεια και λαμβάνει μια ανταμοιβή. Οι ανταμοιβές είναι άμεσες και δίνονται κατ' ευθείαν από το περιβάλλον. Το τι είναι καλό μακροπρόθεσμα καθορίζεται από την αξία της κατάστασης. Γενικά μιλώντας, η αξία της κατάστασης είναι η ολική ανταμοιβή που μπορεί να περιμένει ο πράκτορας αρχίζοντας από αυτή την κατάσταση. Οι αξίες των καταστάσεων ενημερώνονται από παρατηρήσεις που κάνει ο πράκτορας καθ' όλη τη διάρκεια ζωής του, και γι' αυτό το λόγο ο πράκτορας μαθαίνει μέσω εμπειρίας. Τέλος, η επόμενη κατάσταση επιλέγεται.

Η ενέργεια του πράκτορα αμείβεται σε κάθε βήμα με ένα ενισχυτικό σήμα από τη συνάρτηση αμοιβής. Η συνάρτηση αμοιβής είναι η διαδικασία που παράγει ένα άμεσο ενισχυτικό σήμα σε κάθε ενέργεια που επιλέγει ο πράκτορας από κάθε κατάσταση. Αυτή η συνάρτηση δεν βρίσκεται μέσα στα όρια ελέγχου του πράκτορα, οπότε υπάρχει έξω απ' αυτόν (στο περιβάλλον). Ο πράκτορας αμείβεται ανάλογα με το στόχο του. Το ενισχυτικό σήμα ορίζει τον στόχο του πράκτορα, οπότε η αμοιβή πρέπει να δείχνει τι πρέπει να εκπληρωθεί, όχι πως να το κατορθώσει.

Στην Ενισχυτική Μάθηση ο στόχος του πράκτορα είναι να μεγιστοποιήσει τις αμοιβές του μακροπρόθεσμα. Αυτό σημαίνει ότι πρέπει να έχει την ικανότητα να αναπαριστά την αξία μελλοντικών αμοιβών τώρα. Χρησιμοποιεί τον ρυθμό έκπτωσης για να το κάνει αυτό. Ένας ρυθμός έκπτωσης ίσος με μηδέν σημαίνει ότι ο πράκτορας είναι «μυωπικός», που αυτό σημαίνει ότι επιλέγει μία ενέργεια στο χρόνο t για να μεγιστοποιήσει την αμοιβή στο χρόνο t+1. Όσο ο ρυθμός έκπτωσης πλησιάζει το 1, ο πράκτορας γίνεται πιο «διορατικός», που αυτό σημαίνει ότι επιλέγει μία ενέργεια στο χρόνο t για να μεγιστοποιήσει μελλοντικές αμοιβές σε κάποιο μετέπειτα χρονικό βήμα t+n, όπου n>1. Ο ρυθμός έκπτωσης μπορεί να ιδωθεί ως η σχετική αξία των καθυστερημένων, έναντι των άμεσων αμοιβών. Το αναμενόμενο αποτέλεσμα των μελλοντικών αμοιβών που έχουν υποστεί έκπτωση *Rt* φαίνεται από την Εξίσωση 2.2:

$$R(t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ \ldots = \sum_{\kappa=0}^{T} \gamma^k \, r_{t+k+1} \qquad (2.2)$$

όπου γ, $0 \leq \gamma \leq 1$, είναι ο ρυθμός έκπτωσης, *r* είναι η αμοιβή σε κάθε χρονική στιγμή και *T* είναι η τελική χρονική στιγμή ή το άπειρο. Το αναμενόμενο αποτέλεσμα των μελλοντικών αμοιβών χρησιμοποιείται στην Συνάρτηση Αξίας για να καταστήσει ικανό τον πράκτορα να εκτιμά ή να προβλέπει ποια ενέργεια να πάρει. Υπάρχουν δύο τύποι Συναρτήσεων Αξίας:

1. $V^{\pi}(s,a)$ είναι η συνάρτηση κατάστασης-αξίας (state-value function) για μια πολιτική π, που αυτό σημαίνει ότι δίνει το αναμενόμενο αποτέλεσμα όταν ξεκινά από μια κατάσταση s ενόσω ακολουθεί μια δοθείσα πολιτική π.

2. $Q^{\pi}(s,a)$ είναι η συνάρτηση ενέργειας-αξίας (action-value function) για μια πολιτική π, που αυτό σημαίνει ότι είναι η αξία της επιλογής μιας ενέργειας α στην κατάσταση s ενόσω ακολουθεί μια δοθείσα πολιτική π.

Η εξίσωση κάθε συνάρτησης εξαρτάται από τον αλγόριθμο που εφαρμόζεται κάθε φορά. Πιο κάτω θα παρουσιάσουμε κάποιες μεθόδους ΕΜ, και θα δώσουμε τις εξισώσεις για τους πιο κύριους αλγόριθμους που χρησιμοποιούνται.

2.2 Μέθοδοι Ενισχυτικής Μάθησης

2.2.1 Εισαγωγή

Οι μέθοδοι της ΕΜ ενημερώνουν την πολιτική του πράκτορα ως αποτέλεσμα της εμπειρίας του. Υπάρχουν τρεις κύριες κλάσεις μεθόδων: Δυναμικός Προγραμματισμός (ΔΠ) (Bellman, 1957), Monte Carlo (Michie και Chambers, 1968) και μάθηση Χρονικών Διαφορών (Temporal Difference) (Sutton και Barto, 1998). Η κάθε μία έχει τα δικά της υπέρ και κατά, εν τούτοις όλες έχουν τις βασικές διαδικασίες: αξιολόγηση πολιτικής (policy evaluation), που περιλαμβάνει εκτίμηση της Συνάρτησης Αξίας και συγκράτηση των αξιών των πραγματικών ή πιθανών καταστάσεων και βελτίωση πολιτικής (policy improvement), που ενημερώνει τη Συνάρτηση Αξίας και βελτιώνει την πολιτική του πράκτορα.

2.2.2 Μέθοδος Δυναμικού Προγραμματισμού

Στο ΔΠ (Bellman, 1957) ο στόχος είναι να υπολογιστούν οι βέλτιστες πολιτικές (επιλογή ενέργειας), δοθέντος ενός τέλειου μοντέλου του περιβάλλοντος σαν μία ΜΔΑ. Κάποιο μοντέλο μπορεί να οριστεί σαν μια αναπαράσταση του περιβάλλοντος, τέτοια ώστε από μια δεδομένη κατάσταση και ενέργεια ο εκπαιδευόμενος ή πράκτορας μπορεί να προβλέψει

την προκύπτουσα επόμενη κατάσταση και επόμενη ενέργεια. Ένα μοντέλο μπορεί να είναι στοχαστικό. Ένα στοχαστικό μοντέλο έχει διάφορες πιθανές καταστάσεις και ανταμοιβές, κάθε μία με κάποια πιθανότητα εμφάνισης. Η μέθοδος του ΔΠ χρησιμοποιεί Συναρτήσεις Αξίας για να οργανώσει το χώρο αναζήτησης για καλές πολιτικές. Η βασική ιδέα είναι η χρησιμοποίηση των εξισώσεων Bellman (1957) σαν κανόνες ενημέρωσης για προσέγγιση των επιθυμητών Συναρτήσεων Αξίας. Ο ΔΠ το κάνει αυτό μέσω 2 μεθόδων: αξιολόγηση πολιτικής και βελτίωση πολιτικής. Ο σκοπός της αξιολόγησης πολιτικής, διαφορετικά γνωστή ως το πρόβλημα της πρόβλεψης (prediction problem), είναι πως να υπολογίσει τη συνάρτηση κατάστασης-αξίας (V(s)). Στην ΕΜ είναι η επαναληπτική αποτίμηση πολιτικής, δηλαδή πως να παραχθεί μια διαδοχική προσέγγιση της συνάρτησης κατάστασης-αξίας, που είναι σημαντική. Η συνάρτηση κατάστασης-αξίας προσεγγίζεται με «bootstrapping», δηλαδή εκτίμηση των αξιών των καταστάσεων με βάση τις εκτιμήσεις των επόμενων καταστάσεων. Στη μέθοδο του ΔΠ αυτό σημαίνει η συγκράτηση εφεδρικών (backups) καταστάσεων για κάθε κατάσταση, το οποίο είναι υπολογιστικά ακριβό. Συνήθως η τερματική συνθήκη είναι όταν η διαφορά μεταξύ της τρέχουσας συνάρτησης κατάστασης-αξίας και της προηγούμενης είναι μικρή. Στη διαδικασία της βελτίωσης πολιτικής, ο σκοπός είναι ο προσδιορισμός του πως να αποκτηθεί μια καλύτερη πολιτική η οποία βελτιώνεται πάνω στην αρχική πολιτική, δηλαδή έχοντας μεγαλύτερο αναμενόμενο κέρδος. Η επανάληψη πολιτικής (policy iteration) συνδυάζει αξιολόγηση και βελτίωση με την περικοπή των δύο ξεχωριστών διαδικασιών για να γίνει υπολογιστικά φθηνότερη. Ο ΔΠ είναι καλά αναπτυγμένος, αφού υπάρχει από τα τέλη της δεκαετίας του 1950. Το κύριο μειονέκτημά του είναι ότι απαιτεί ένα πλήρες και ακριβές μοντέλο του περιβάλλοντος το οποίο δεν είναι πάντα διαθέσιμο. Επιπρόσθετα, η μέθοδος του ΔΠ ενεργεί πάνω σε όλο το σύνολο καταστάσεων και το σύνολο ενεργειών, η οποία παρόλο που μπορεί να βελτιωθεί με τεχνικές γενίκευσης, είναι ακόμη υπολογιστικά ακριβή σε σύγκριση με άλλες μεθόδους ΕΜ.

2.2.3 Μέθοδος Monte Carlo

Η μέθοδος Monte Carlo (Michie και Chambers, 1968), δεν απαιτεί την διατήρηση πλήρης γνώσης του περιβάλλοντος (model free). Μαθαίνει από προσομοιωμένη εμπειρία που βασίζεται σε επεισόδια, αντί σε διακριτά χρονικά διαστήματα κάποιας εργασίας. Όπως και με τη μέθοδο του ΔΠ, η μέθοδος Monte Carlo βασίζεται στην εκτίμηση της Συνάρτησης Αξίας. Στην μέθοδο αυτή, η αξιολόγηση πολιτικής υπολογίζει κατά μέσο όρο τα παρατηρημένα αποτελέσματα μετά από επισκέψεις σε κάποια δοθείσα κατάσταση και σταδιακά ο μέσος όρος για κάποια κατάσταση θα πρέπει να συγκλίνει, δηλαδή να μην αλλάξει. Η μέθοδος Monte Carlo δεν κάνει «bootstrap», όπως η μέθοδος του ΔΠ. Το κύριο πλεονέκτημα της μεθόδου Monte Carlo, σε σύγκριση με τη μέθοδο του ΔΠ, είναι ότι η μέθοδος Monte Carlo μαθαίνει κατ' ευθείαν από το περιβάλλον και έτσι δεν χρειάζεται ένα ακριβές μοντέλο του περιβάλλοντος. Το κάνει αυτό με δειγματοληψία ή προσέγγιση των καταστάσεων, σε σύγκριση με τη διατήρηση ενός πλήρους συνόλου καταστάσεων. Αυτό την κάνει υπολογιστικά λιγότερο απαιτητική από τη μέθοδο του ΔΠ. Για να μπορεί να συνεχίσει να εξερευνά νέες ενέργειες, σε σύγκριση με την εκμετάλλευση γνωστών ενεργειών οι οποίες αποφέρουν αμοιβές, η μέθοδος Monte Carlo χρησιμοποιεί είτε ένα αλγόριθμο εντός-πολιτικής (on-policy), ο οποίος αποτιμά ή βελτιώνει την πολιτική ενόσω την χρησιμοποιεί, ή ένα αλγόριθμο εκτός-πολιτικής (off-policy), ο οποίος διαχωρίζει την πολιτική σε μια πολιτική εκτίμησης για βελτίωση και μια πολιτική συμπεριφοράς για να εκτελέσει τη βελτίωση.

2.2.4 Μέθοδος Χρονικών Διαφορών

Η μέθοδος ΧΔ (Sutton και Barto, 1998), έχει τα πλεονεκτήματα και της μεθόδου του ΔΠ και της μεθόδου Monte Carlo, αφού δεν χρειάζεται να διατηρεί πλήρη γνώση του περιβάλλοντος (άρα συμπεριφέρεται όπως τη μέθοδο Monte Carlo), και βασίζει τις εκτιμήσεις σε προηγούμενες μαθημένες εμπειρίες (όπως η μέθοδος του ΔΠ). Η αξιολόγηση πολιτικής χρησιμοποιεί εμπειρία για να ενημερώνει τις εκτιμήσεις της Συνάρτησης Αξίας όπως φαίνεται στην Εξίσωση 2.3 (Sutton and Barto, 1998):

$$V(S_t) = V(S_t) + \alpha \cdot [r_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)] \qquad (2.3)$$

όπου $V(S_t)$ η συνάρτηση κατάστασης-αξίας της κατάστασης S για τη χρονική στιγμή *t*, *α* είναι μια παράμετρος που συχνά αναφέρεται ως ρυθμός μάθησης, r_{t+1} είναι αμοιβή που λήφθηκε τη χρονική στιγμή *t+1*, *γ* είναι ο ρυθμός έκπτωσης και $V(S_{t+1})$ είναι η συνάρτηση κατάστασης-αξίας της κατάστασης *S* για τη χρονική στιγμή *t+1*. Η αξία της προηγούμενης κατάστασης ενημερώνεται με βάση τη νέα κατάσταση, δηλαδή γίνεται bootstrapping και χρειάζεται να περιμένει μόνο μέχρι το τέλος μιας χρονικής στιγμής για να ενημερώσει τη Συνάρτηση Αξίας, ενώ η μέθοδος Monte Carlo χρειάζεται να περιμένει μέχρι το τέλος ενός επεισοδίου.

Υπάρχουν διάφορες μέθοδοι Χρονικής Διαφοράς. Ο εκτός-πολιτικής αλγόριθμος Q-learning (Watkins, 1989) προσεγγίζει την καταλληλότερη συνάρτηση ενέργειας-αξίας (*Q(s,α)*), χωρίς να χρησιμοποιεί την πολιτική. Συγκεκριμένα, μαθαίνει την άπληστη πολιτική ενόσω χρησιμοποιεί μια πολιτική που εξερευνά, δηλαδή επιλέγει μη-άπληστες ενέργειες. Η μέθοδος «ενέργειας-κριτικής» (actor-critic method) είναι μια εντός-πολιτικής μέθοδος Χρονικής Διαφοράς, αφού ο κριτής πρέπει να μάθει το ενισχυτικό σήμα. Πριν την ανάπτυξη του αλγορίθμου Q-learning, η μέθοδος ενέργειας-κριτικής ήταν η μέθοδος που προτιμούσαν αρκετοί ερευνητές του τομέα της ενισχυτικής μάθησης. Η μέθοδος ΧΔ είναι η πιο διαδεδομένη μέθοδος ενισχυτικής μάθησης, αφού είναι απλή, λειτουργεί βήμα με βήμα και είναι υπολογιστικά λιγότερο ακριβή από άλλες μεθόδους ΕΜ.

Η βάση όλων των μεθόδων ΧΔ είναι πως να κατανέμουν αμοιβή (credit) σε ενέργειες, οι οποίες παρήγαγαν την τελική ανταμοιβή. Αυτό ορίζεται ως το πρόβλημα της Χρονικά καθυστερημένης Απόδοσης ανταμοιβών (Temporal Credit Assignment problem). Στις μεθόδους Χρονικών Διαφορών, το σφάλμα *δt* υπολογίζεται ανάλογα με την Εξίσωση 2.4 και έπειτα ανατίθεται στις κατάλληλες καταστάσεις:

$$\delta_t = r_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t) \qquad (2.4)$$

όπου ο χρόνος *t* είναι η τρέχουσα χρονική στιγμή, r_{t+1} είναι το ενισχυτικό σήμα της χρονικής στιγμής *t+1*, *γ* είναι ο ρυθμός έκπτωσης των μελλοντικών αμοιβών, $V(S_{t+1})$ είναι η συνάρτηση κατάστασης-αξίας της χρονικής στιγμής t+1 και $V(S_t)$ η συνάρτηση κατάστασης-αξίας της χρονικής στιγμής t. Ο βασικός μηχανισμός για χρονική απόδοση ευθύνης στην ΕΜ είναι το ίχνος επιλεξιμότητας (eligibility trace) *λ*. Η βασική ιδέα είναι ότι όταν συμβαίνει ένα σφάλμα, μόνο στις επιλέξιμες καταστάσεις αποδίδεται ευθύνη γι' αυτό. Σε προηγούμενες καταστάσεις δίνεται λιγότερη ευθύνη για το σφάλμα. Ένα ίχνος επιλεξιμότητας μπορεί να αυξάνεται κάθε φορά που γίνεται επίσκεψη μιας κατάστασης και μετά να εξαφανίζεται βαθμιαία όταν η κατάσταση δεν επισκέπτεται. Η Εξίσωση 2.5 υπολογίζει το ίχνος επιλεξιμότητας για κάποια κατάσταση, $e_t(s)$.Αν η κατάσταση έχει επισκεφθεί, τότε το ίχνος επιλεξιμότητας της αυξάνεται κατά 1, αλλιώς εξασθενεί με την πάροδο του χρόνου:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s), & s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1, & s = s_t \end{cases} \qquad (2.5)$$

όπου γ είναι ο ρυθμός έκπτωσης, λ είναι η παράμετρος εξαφάνισης ίχνους με τιμές που κυμαίνονται μεταξύ 0 (που αντιπροσωπεύει bootstrapping) και 1 (που αντιπροσωπεύει nonbootstrapping), $e_t(s)$ είναι το ίχνος επιλεξιμότητας της κατάστασης s τη χρονική στιγμή *t*. Η Εξίσωση 2.6 υπολογίζει την αλλαγή στην συνάρτηση κατάστασης-αξίας V(s) για τις πρόσφατα επισκεφθείσες καταστάσεις:

$$\Delta V(s) = a\, \delta_t\, e_t(s) \qquad (2.6)$$

όπου α είναι ο ρυθμός μάθησης, δ_t είναι το σφάλμα χρονικών διαφορών που δίνεται από την Εξίσωση 2.4 και $e_t(s)$ είναι το ίχνος επιλεξιμότητας της κατάστασης *s* τη χρονική στιγμή *t* που δίνεται από την Εξίσωση 2.5. Η Εξίσωση 2.3 είναι η Συνάρτηση Αξίας για μια ειδική περίπτωση του αλγόριθμου TD(λ), τη TD(0), όπου μόνο μια κατάσταση που προηγείται της τρέχουσας, αλλάζει από το σφάλμα χρονικών διαφορών, σε αντίθεση με την περίπτωση TD(λ), που επιλέγει όλες τις επιλέξιμες καταστάσεις για να αλλάξουν από το σφάλμα χρονικών διαφορών.

2.2.4.1 Q-Learning

Ο αλγόριθμος Q-Learning (Watkins, 1989) λαμβάνει υπόψη του μεταβάσεις από ζεύγος κατάστασης-ενέργειας σε ζεύγος κατάστασης-ενέργειας και θεωρείται ως μια από τις πιο σημαντικές ανακαλύψεις στον τομέα της ΕΜ, αφού χρησιμοποιεί μια μέθοδο ΧΔ εκτός-πολιτικής. Μια μέθοδος εκτός-πολιτικής μπορεί να μάθει διαφορετικές πολιτικές για τη συμπεριφορά και την εκτίμηση. Συγκεκριμένα, σε αυτή την περίπτωση, η συνάρτηση ενέργειας-αξίας, *Q*, προσεγγίζει άμεσα την *Q**, που είναι η βέλτιστη συνάρτηση ενέργειας-αξίας ανεξάρτητα από την πολιτική που ακολουθείται. Η Εξίσωση 2.7 χρησιμοποιείται για την εκτίμηση της αναμενόμενης αξίας του ζεύγους κατάστασης-ενέργειας <S_t,a_t>:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \quad (2.7)$$

όπου $Q(s_t, a_t)$ είναι η συνάρτηση ενέργειας-αξίας του ζεύγους κατάστασης-ενέργειας <S_t,a_t> για τη χρονική στιγμή *t*, *α* είναι ο ρυθμός μάθησης, r_{t+1} είναι η αμοιβή που λήφθηκε τη χρονική στιγμή *t+1*, *γ* είναι ο ρυθμός έκπτωσης και $\max_a Q(s_{t+1}, a)$ είναι η συνάρτηση ενέργειας-αξίας του ζεύγους κατάστασης-ενέργειας <S_{t+1},a_t> για τη χρονική στιγμή *t+1*, όπου α είναι η ενέργεια στην οποία αντιστοιχεί η μέγιστη τιμή της συνάρτησης *Q*. Η αποθήκευση των τιμών γίνεται σε look-up tables.

2.2.4.2 SARSA

Ο αλγόριθμος SARSA (State Action Reward State Action) αρχικά παρουσιάστηκε σαν "τροποποιημένος Q-Learning" (Rummery and Niranjan, 1994). Λαμβάνει υπόψη του μεταβάσεις από ζεύγος κατάστασης-ενέργειας σε ζεύγος κατάστασης-ενέργειας χρησιμοποιώντας μια μέθοδο ΧΔ εντός-πολιτικής. Σε μία μέθοδο εντός-πολιτικής πρέπει να εκτιμηθεί η συνάρτηση ενέργειας-αξίας, $Q^{\pi}(s,\alpha)$, για την τρέχουσα πολιτική, και για όλες τις καταστάσεις και ενέργειες. Συγκεκριμένα, υπολογίζεται συνεχώς η συνάρτηση ενέργειας-αξίας Q^{π} για την τρέχουσα πολιτική, και ταυτόχρονα αλλάζει η

πολιτική προς την απληστία σύμφωνα με τη συνάρτηση ενέργειας-αξίας Q^{π}. Η Εξίσωση 2.8 χρησιμοποιείται για την εκτίμηση της αναμενόμενης αξίας του ζεύγους κατάστασης-ενέργειας <St,at>:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \quad (2.8)$$

όπου $Q(s_t, a_t)$ είναι η συνάρτηση ενέργειας-αξίας του ζεύγους κατάστασης-ενέργειας <St,at> για τη χρονική στιγμή *t*, *α* είναι ο ρυθμός μάθησης, r_{t+1} είναι η αμοιβή που λήφθηκε τη χρονική στιγμή *t+1*, *γ* είναι ο ρυθμός έκπτωσης και $Q(s_{t+1}, a_{t+1})$ είναι η συνάρτηση ενέργειας-αξίας του ζεύγους κατάστασης-ενέργειας <S_{t+1},a_{t+1}> για τη χρονική στιγμή *t+1*. Για να υπολογιστεί αυτή η εξίσωση πρέπει για χρόνο *t* και *t+1* να γίνει πέρασμα μέσα από τη μετάβαση: κατάσταση, ενέργεια, αμοιβή, κατάσταση, ενέργεια (State, Action, Reward, State, Action), και σύμφωνα με αυτό ο αλγόριθμος πήρε το όνομα SARSA (Sutton 1996).

2.2.4.3 Policy Hill-Climbing (PHC)

Μια απλή επέκταση του αλγόριθμου Q-Learning είναι ο αλγόριθμος Policy Hill-Climbing (PHC) (Bowling και Veloso, 2001), που εφαρμόζεται στα παίγνια μικτής στρατηγικής. Ο PHC στην ουσία κάνει hill-climbing στον χώρο των μικτών στρατηγικών. Οι Q τιμές διατηρούνται όπως και στον Q-Learning. Επιπρόσθετα, ο αλγόριθμος διατηρεί την τρέχουσα μικτή πολιτική. Η πολιτική βελτιώνεται με την αύξηση της πιθανότητας ότι θα επιλέξει την ενέργεια με την υψηλότερη τιμή, σύμφωνα με ένα ρυθμό μάθησης, $\delta \in (0,1]$. Όταν δ=1, ο αλγόριθμος είναι ισοδύναμος με τον Q-Learning, μιας και με κάθε βήμα η πολιτική κινείται στην άπληστη επιλογή, επιλέγοντας με πιθανότητα 1 την ενέργεια με την υψηλότερη αμοιβή.

Αυτή η τεχνική είναι ορθολογιστική και συγκλίνει σε μια βέλτιστη πολιτική αν οι άλλοι παίχτες παίζουν σταθερές πολιτικές. Η απόδειξη προέρχεται από την απόδειξη του αλγόριθμου Q-Learning, που εγγυείται ότι οι *Q* τιμές θα συγκλίνουν στις βέλτιστες *Q**

με μια εφαρμόσιμη πολιτική που εκτελεί και εξερεύνηση. Παρομοίως, το π θα συγκλίνει σε μια πολιτική που είναι άπληστη σύμφωνα με την Q, η οποία συγκλίνει στην Q^*.

Ακολουθεί στο Σχήμα 2.3 ο ψευδοκώδικας του αλγόριθμου αυτού.

---

1. Έστω α και δ ρυθμοί μάθησης. Αρχικοποίησε,

$$Q(s,a) \leftarrow 0,\ \pi(s,a) \leftarrow \frac{1}{|A_i|}$$

2. Επανέλαβε,

   (α) Από την κατάσταση S, επέλεξε ενέργεια α με πιθανότητα π(s,α), με κάποια εξερεύνηση

   (β) Παρατήρησε την ανταμοιβή r και επόμενη κατάσταση s` και υπολόγισε

$$Q(s,\alpha) = (1-\alpha)Q(s,\alpha) + \alpha\,(r + \gamma \max_{a'} Q(s',a'))$$

   (γ) Ενημέρωσε το π(s,a) με

$$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta, \text{ αν } a = argmax_{a'} Q(s,a') \\ |A_i| - 1\ , \text{αλλιώς} \end{cases}$$

Σχήμα 2.3: Αλγόριθμος Policy Hill-Climbing (PHC) για παίχτη i

Σύμφωνα με τον αλγόριθμο διατηρούμε τιμές Q τις οποίες αρχικοποιούμε με 0 και πιθανότητες για κάθε ζεύγος κατάστασης-ενέργειας τις οποίες αρχικοποιούμε με 1/Πλήθος ενεργειών του συγκεκριμένου παίχτη. Ακολούθως επαναλαμβάνεται η εξής διαδικασία: Από την κατάσταση s επιλέγουμε την ενέργεια a με πιθανότητα $\pi(s,\alpha)$, κάνοντας όμως και χρήση της πιθανότητας για εξερεύνηση. Εφόσον πάρουμε την αμοιβή και την επόμενη κατάσταση ανανεώνουμε την Q τιμή της κατάστασης s με ενέργεια α σύμφωνα με τον τύπο στο σχήμα, καθώς και την πιθανότητα $\pi(s,\alpha)$, όπως δείχνει ο αλγόριθμος στο σχήμα.

2.2.4.4 WoLF Policy Hill-Climbing (WoLF- PHC)

Ο αλγόριθμος WoLF Policy Hill-Climbing (Bowling και Veloso, 2001) είναι μια επέκταση του αλγόριθμου PHC. Οι σημαντικές διαφοροποιήσεις του σε σχέση με τον PHC είναι η χρήση μεταβλητού ρυθμού μάθησης και η αρχή WoLF.

Η βασική ιδέα είναι να μεταβάλουμε τον ρυθμό μάθησης με τρόπο ώστε να ενθαρρύνουμε την σύγκλιση χωρίς να θυσιάζουμε τον ορθολογισμό. Η αρχή του WoLF δηλώνει «Μάθε γρήγορα ενόσω χάνεις και αργά ενόσω νικάς» (“Win Or Lose Fast Policy Hill-Climbing”). Η μέθοδος, για να διευκρινίζουμε πότε νικά ο πράκτορας και πότε όχι, συγκρίνει την προβλεπόμενη αμοιβή της τρέχουσας πολιτικής με αυτή της μέσης πολιτικής στο χρόνο. Αυτή η αρχή βοηθά στη σύγκλιση, με το να δίνει περισσότερο χρόνο στους άλλους παίχτες να προσαρμοστούν στις αλλαγές της στρατηγικής του παίχτη, που στην αρχή παρουσιάζονται βοηθητικές, ενώ επιτρέπουν στον παίχτη να προσαρμοστεί γρηγορότερα στις στρατηγικές των άλλων παιχτών όταν είναι επιβλαβής.

Ακολουθεί στο Σχήμα 2.4 ο ψευδοκώδικας του αλγόριθμου αυτού.

---

1. Έστω α και δl > δw ρυθμοί μάθησης. Αρχικοποίησε,

$$Q(s,a) \leftarrow 0,\ \pi(s,a) \leftarrow \frac{1}{|A_i|},\ C(s) \leftarrow 0$$

2. Επανέλαβε,

(α) Από την κατάσταση S, επέλεξε ενέργεια α με πιθανότητα π(s,α), με κάποια εξερεύνηση

(β) Παρατήρησε την ανταμοιβή r και επόμενη κατάσταση s` και υπολόγισε

$$Q(s,\alpha) = (1-\alpha)Q(s,\alpha) + \alpha\,(r + \gamma \max_{\alpha'} Q(s',\alpha'))$$

(γ) Ενημέρωσε την αναμενόμενη πιθανότητα της μέσης πολιτικής $\bar{\pi}$

$C(s) \leftarrow C(s)+1$

$$\forall\, \alpha' \in Ai \quad \bar{\pi}(s,\alpha') \leftarrow \bar{\pi}(s,\alpha') + \frac{1}{C(s)}\,(\pi(s,\alpha') - \bar{\pi}(s,\alpha'))$$

(δ) Ενημέρωσε $\pi(s,\alpha)$ βάση

$$\pi(s,\alpha) = \pi(s,\alpha) + \begin{cases} \delta, & \text{αν } \alpha = argmax_{a'} Q(s,a') \\ \dfrac{-\delta}{|A_i|-1}, & \text{αλλιώς} \end{cases}$$

Όπου,

$$\delta = \begin{cases} \delta w, \text{αν } \sum_a \pi(s,\alpha)Q(s,a) > \sum_a \bar{\pi}(s,\alpha)Q(s,a) \\ \delta l, \text{αλλιώς} \end{cases}$$

Σχήμα 2.4: Αλγόριθμος WoLF-Policy Hill-Climbing (WoLF-PHC) για παίχτη i

Ο αλγόριθμος απαιτεί 2 ρυθμούς μάθησης *δl* > *δw*. Όταν ο παίχτης νικά χρησιμοποιείται το *δw* και στην αντίθετη περίπτωση το *δl*. Αυτό εντοπίζεται συγκρίνοντας την αναμενόμενη τιμή χρησιμοποιώντας τις τρέχουσες Q τιμές ακολουθώντας την τρέχουσα πολιτική π στην τρέχουσα κατάσταση με αυτό ακολουθώντας την μέση πολιτική $\bar{\pi}$. Αν η αναμενόμενη τιμή από την τρέχουσα πολιτική είναι μικρότερη (ο παίχτης χάνει), ο μεγαλύτερος ρυθμός μάθησης χρησιμοποιείται.

2.3 Αντιμετώπιση μεγάλων συνόλων κατάστασης-ενέργειας

2.3.1 Εισαγωγή

Για μερικά προβλήματα δεν είναι εφικτό να υπάρχει ένας πίνακας με εγγραφές για κάθε κατάσταση (*V(s)*), ή ζεύγος κατάστασης-ενέργειας (*Q(s,α)*). Οι λόγοι μπορεί να είναι οι

εξής: 1) χρειάζεται απεριόριστη μνήμη, για παράδειγμα όταν έχουμε συνεχείς σύνολα καταστάσεων ενεργειών, 2) χρειάζεται περισσότερος υπολογιστικός χρόνος και 3) τα δεδομένα γίνονται θορυβώδη (noisy). Μπορεί να είναι αδύνατο να διατηρείται πλήρης περιγραφή για όλες τις πιθανές καταστάσεις, ή εναλλακτικά κάποιος μπορεί να έχει μόνο μερικώς παρατηρημένη πληροφορία (partially observable information) για τις καταστάσεις, η οποία μπορεί να τροποποιείται μέσω της μάθησης κατά τη διάρκεια του χρόνου.

2.3.2 Ιεραρχικές μεθόδοι

Οι ιεραρχικές μέθοδοι επικεντρώνονται στη διαδοχική και χρονική διάσταση μιας εργασίας και χρησιμοποιούνται σε προβλήματα τα οποία είναι δομημένα, δηλαδή έχουν κάποια δομή και μπορούν να διασπαστούν σε πιο απλές υποδομές. Προσπαθούν αντί να μαθαίνουν μια πολιτική από την αρχή, να ξαναχρησιμοποιούν ήδη υπάρχουσες πολιτικές για πιο απλές υποδομές (sub-task), κάτι το οποίο έχει αρκετά πλεονεκτήματα όπως πιο γρήγορη μάθηση, μάθηση με πιο λίγες δοκιμές και βελτίωση της εξερεύνησης (Dietterich, 2000). Ένα ανοιχτό πρόβλημα στην Ιεραρχική Ενισχυτική Μάθηση (IEM) είναι η εύρεση της ιεραρχικής δομής. Υπάρχουν μέθοδοι όπου η διάσπαση του προβλήματος σε μια ιεραρχία από υποδομές αφήνεται στον χρήστη, όπως για παράδειγμα στη MAXQ διάσπαση (Dietterich, 2000), και μέθοδοι όπου μαθαίνουν την ιεραρχία από μόνες τους, όπως για παράδειγμα η μέθοδος HEXQ (Hengst, 2002). Άλλο ένα καλό παράδειγμα ιεραρχικής μεθόδου είναι οι επιλογές (options) (Suton et al., 1999), οι οποίες είναι χρονικά αφηρημένες γενικεύσεις αρχέγονων καταστάσεων, υψηλότερου επιπέδου, και μπορούν να οριστούν ως πολιτικές για επιλογή ενέργειας κατά τη διάρκεια μιας χρονικής περιόδου.

2.3.3 Τεχνικές Προσέγγισης Συναρτήσεων

Η μέθοδος της προσέγγισης συναρτήσεων (Sutton και Barto, 1998), που χρησιμοποιείται με ΕΜ, πρέπει να μπορεί να αντιμετωπίζει τα διακεκριμένα

χαρακτηριστικά της ΕΜ. Αυτά είναι: η καθυστερημένη ανταμοιβή, η χρονική απόδοση ευθυνών, το πρόβλημα της εξερεύνησης και εκμετάλλευσης, την πιθανότητα μερικώς παρατηρημένων καταστάσεων και μάθηση καθ' όλη τη διάρκεια ζωής του πράκτορα. Ο σκοπός της προσέγγισης συναρτήσεων για την ΕΜ είναι να διδάξει μια πολιτική για επιλογή κάποιας ενέργειας δεδομένης κάποιας κατάστασης. Η μάθηση στην ΕΜ γίνεται σε πραγματικό χρόνο, δηλαδή η συνάρτηση πρέπει να μπορεί να αλλάζει κατά τη διάρκεια του προβλήματος. Επιπρόσθετα, η συνάρτηση χρειάζεται να μαθαίνει αυξητικά. Για παράδειγμα, στην ΕΜ ο πράκτορας μαθαίνει την βέλτιστη πολιτική, ενώ αυτή αλλάζει. Ακόμη και αν η πολιτική μένει η ίδια, οι επιθυμητές τιμές της κατάστασης μπορεί να αλλάξουν.

2.4 Πολιτική

Πολιτική είναι ο τρόπος που καθορίζεται η επιλογή ενέργειας ή αλλιώς της πολιτικής που θα ακολουθείται. Συχνά χρησιμοποιούνται τρεις πολιτικές για επιλογή ενέργειας. Ο σκοπός αυτών των πολιτικών είναι η εξισορρόπηση μεταξύ της εκμετάλλευσης προηγούμενης γνώσης και της εξερεύνησης νέων ενεργειών. Αυτές είναι 1) ε-greedy, κατά την οποία τις περισσότερες φορές επιλέγεται η ενέργεια με την μεγαλύτερη εκτιμημένη ανταμοιβή, η οποία ονομάζεται ως η πιο άπληστη ενέργεια. Πάντα, μετά από κάποιο χρονικό διάστημα, επιλέγεται τυχαία μια ενέργεια με μια μικρή πιθανότητα ε. Η ενέργεια επιλέγεται ομοιόμορφα, ανεξάρτητα από τις εκτιμήσεις ενέργειας-αξίας. Αυτή η μέθοδος εξασφαλίζει ότι αν εκτελεστούν πολλές επαναλήψεις, κάθε ενέργεια θα δοκιμαστεί αρκετές φορές, και συνεπώς εγγυείται ότι θα ανακαλυφθούν βέλτιστες ενέργειες. 2) ε-soft, η οποία είναι πολύ παρόμοια με την ε-greedy. Η καλύτερη ενέργεια επιλέγεται με πιθανότητα 1-ε και τον υπόλοιπο χρόνο επιλέγεται ομοιόμορφα μια τυχαία ενέργεια. Ένα μειονέκτημα των πολιτικών ε-greedy και ε-soft είναι ότι επιλέγουν τυχαίες ενέργειες ομοιόμορφα. Για παράδειγμα η χειρότερη ενέργεια και η δεύτερη καλύτερη ενέργεια μπορούν να επιλεγούν με την ίδια πιθανότητα. 3) Softmax, η οποία δεν έχει το μειονέκτημα των δύο προηγούμενων πολιτικών, αφού ταξινομεί κάθε ενέργεια ανάλογα με την εκτίμηση ενέργειας-αξίας που έχει. Μια τυχαία ενέργεια επιλέγεται έχοντας υπόψη το βάρος που σχετίστηκε με κάθε ενέργεια, κάτι το οποίο σημαίνει ότι υπάρχει πολύ μικρή πιθανότητα να

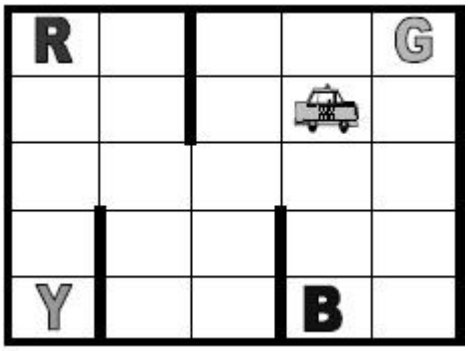
επιλεγούν οι χειρότερες ενέργειες. Η πιο συνηθισμένη softmax μέθοδος χρησιμοποιεί μια κατανομή Boltzmann η οποία εξηγήθηκε στο κεφάλαιο 2.1.

2.5 Μαρκοβιανές Διαδικασίες Απόφασης

Μια βασική υπόθεση αρκετής έρευνας στην ΕΜ είναι ότι η αλληλεπίδραση ενός πράκτορα και του περιβάλλοντος μπορεί να μοντελοποιηθεί ως μια Μαρκοβιανή Διαδικασία Απόφασης ΜΔΑ. Ανεξάρτητα από την ΕΜ και υιοθετώντας μια σχετικά πιο απλοϊκή εικόνα, μια ΜΔΑ είναι μια πλειάδα (S,A,p_T,p_R), όπου S είναι το σύνολο των καταστάσεων, A το σύνολο των ενεργειών, p_T είναι ένα μοντέλο μετάβασης που συλλαμβάνει την πιθανότητα $p_T(s->s')$ η ενέργεια α στην κατάσταση s τη χρονική στιγμή t, θα οδηγήσει στην κατάσταση s' τη χρονική στιγμή $t+1$, και p_R είναι ένα μοντέλο αμοιβής το οποίο συλλαμβάνει την πιθανότητα $p_R(s->r)$ της λήψης της αμοιβής r μετά την εκτέλεση της ενέργειας α στην κατάσταση s. Η επίλυση των ΜΔΑ αποτελείται από την εύρεση μιας πολιτικής $\pi: S->A$, η οποία συσχετίζει τις καταστάσεις με τις ενέργειες και μεγιστοποιεί τις μελλοντικές αμοιβές r, έχοντας κάνει έκπτωση σ' αυτές με το ρυθμό έκπτωσης γ, κατά τη διάρκεια του χρόνου t.

2.6 Taxi Problem

Το Taxi Problem (TP) έχει δημιουργηθεί για την επίδειξη της MAXQ ιεραρχικής διάσπασης (decomposition) από τον Dietterich (2000) και από τότε χρησιμοποιείται ευρέως στον τομέα των ιεραρχικών μεθόδων. Στο TP έχουμε ένα 5x5 πλαίσιο όπως φαίνεται στο Σχήμα 2.5.



Σχήμα 2.5: Το πλαίσιο του προβλήματος του ταξί

όπου υπάρχουν τέσσερεις ειδικά σχεδιασμένες τοποθεσίες οι οποίες απεικονίζονται ως R(ed), B(lue), G(reen) και Y(ellow), ένα ταξί και ένας επιβάτης. Ο επιβάτης μπορεί να βρίσκεται σε μια από αυτές τις τέσσερεις τοποθεσίες είτε μέσα στο ταξί.

Σκοπός αυτού του προβλήματος είναι να μάθει, το ταξί (δηλαδή ο πράκτορας), που βρίσκεται ο επιβάτης, να μετακινηθεί στην τοποθεσία αυτή και να τον παραλάβει, και στη συνέχεια να τον μεταφέρει στον προορισμό που επιθυμεί με όσο το δυνατό λιγότερα βήματα.

Το ταξί ξεκίνα από μια τυχαία θέση σε αυτό το πλαίσιο, καθώς επίσης και η αρχική τοποθεσία του επιβάτη όπως και ο τελικός προορισμός του, που καθορίζονται τυχαία σε μια από αυτές τις τέσσερεις ειδικά σχεδιασμένες τοποθεσίες. Επίσης είναι δυνατόν η αρχική τοποθεσία του επιβάτη να είναι μέσα στο ταξί.

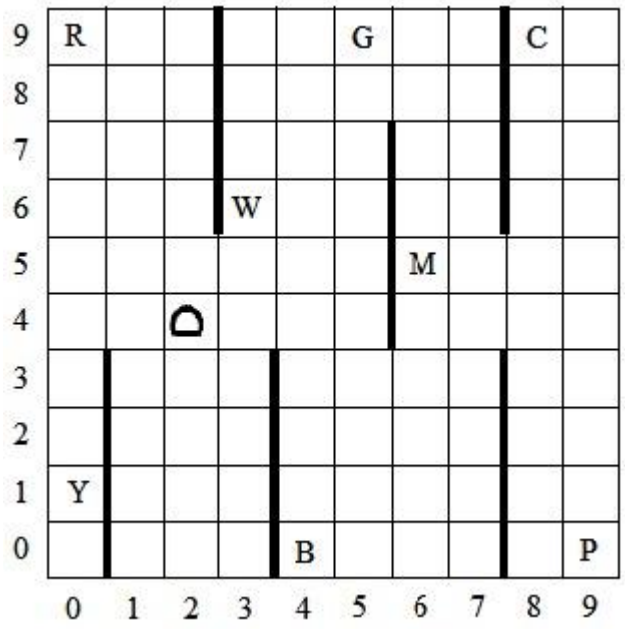
Σε κάθε βήμα, το ταξί μπορεί να εκτελέσει μια ενέργεια από τις ακόλουθες έξι: να κινηθεί α) ένα τετράγωνο βόρια, β) ένα τετράγωνο νότια, γ) ένα τετράγωνο ανατολικά, και δ) ένα τετράγωνο δυτικά, ε) να παραλάβει τον επιβάτη και ζ) να αφήσει τον επιβάτη. Αν κτυπήσει σε τοίχο ή αν προσπαθήσει να βγει εκτός των ορίων του πλαισίου η θέση του δεν αλλάζει. Για μια επιτυχημένη μεταφορά παίρνει σαν αμοιβή 20. Αν το ταξί εκτελέσει την ενέργεια «παρέλαβε τον επιβάτη» σε τοποθεσία όπου δεν βρίσκεται ο επιβάτης, ή αν ήδη μεταφέρεται ο επιβάτης τότε παίρνει σαν αμοιβή -10. Επίσης αν εκτελέσει την ενέργεια «άφησε τον επιβάτη» σε λανθασμένο προορισμό, ή χωρίς να μεταφέρει κάποιο επιβάτη και πάλι η αμοιβή που παίρνει είναι -10. Για οποιαδήποτε

άλλη ενέργεια η αμοιβή είναι -1. Η κάθε δοκιμή τελειώνει όταν γίνει επιτυχώς η μεταφορά του επιβάτη, από το ταξί, στον προορισμό που επιθυμεί.

Το TP μπορεί να διατυπωθεί ως μια επεισοδιακή ΜΔΑ με τρεις μεταβλητές κατάστασης: την τοποθεσία που βρίσκεται το ταξί (0-24), την τοποθεσία του επιβάτη (0-4, όπου το 0 σημαίνει ότι ο επιβάτης βρίσκεται μέσα στο ταξί) και την τοποθεσία του προορισμού (1-4). Άρα σύνολο θα έχουμε 25x5x4=500 καταστάσεις.

2.7 Εκτεταμένο Taxi Problem

Είναι μια επέκταση του απλού Taxi Problem (Dietterich, 2000) και έχει δημιουργηθεί από τους Diuk, Cohen και Littman (2008) για να γίνει επιδείξουν πώς ο αντικειμενοστρεφής αλγόριθμος DOORMAX (Diuk, Cohen και Littman, 2008) συμπεριφέρεται σε προβλήματα με μεγάλο σύνολο κατάστασης-ενέργειας. Στο ETP έχουμε ένα 10x10 πλαίσιο και οκτώ ειδικά σχεδιασμένες τοποθεσίες όπου μπορεί να βρίσκεται ή να επιθυμεί να μεταβεί ο επιβάτης όπως φαίνεται στο Σχήμα 2.6. Σε αυτό το πλαίσιο ισχύει ότι ισχύει και για το απλό TP όσον αφορά τις έξι ενέργειες που δικαιούται το ταξί να εκτελέσει σε κάθε βήμα, καθώς επίσης και ότι ισχύει για τις αμοιβές.

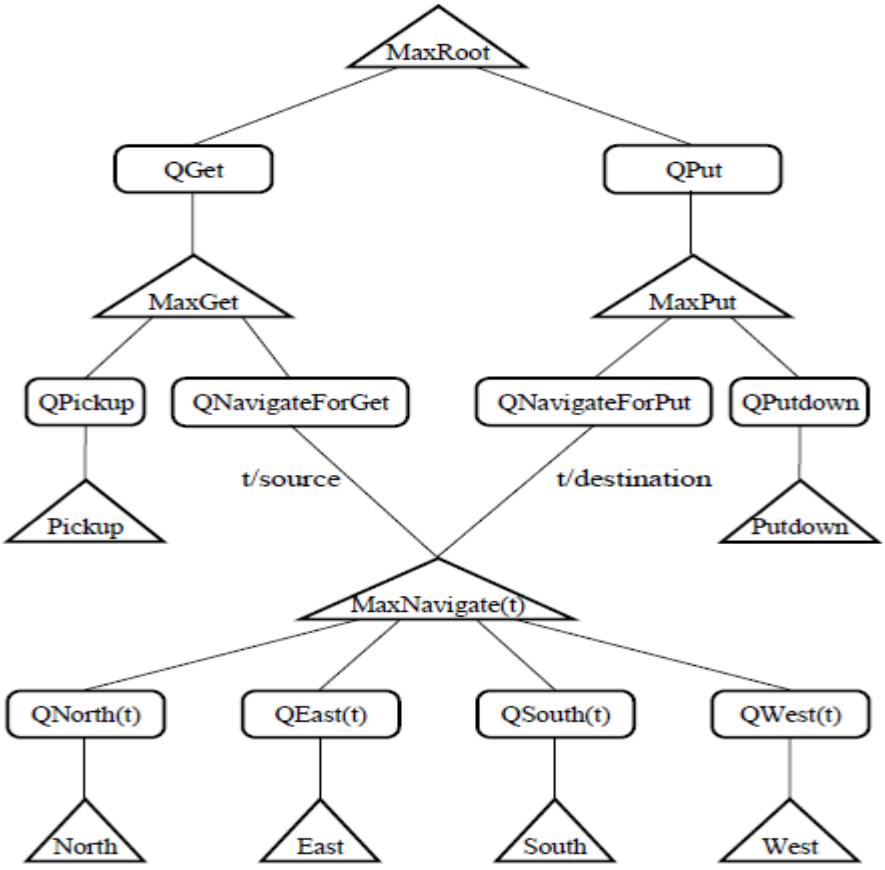


Σχήμα 2.6: Το εκτεταμένο πλαίσιο του προβλήματος του ταξί

όπου υπάρχουν οκτώ ειδικά σχεδιασμένες τοποθεσίες οι οποίες απεικονίζονται ως R, B, G, Y, W, M, P και C, ένα ταξί και ένας επιβάτης. Ο επιβάτης μπορεί να βρίσκεται σε μια από αυτές τις οκτώ τοποθεσίες είτε μέσα στο ταξί.

2.8 MAXQ διάσπαση για το πρόβλημα του ταξί

Η MAXQ διάσπαση (decomposition) για το πρόβλημα του ταξί (Dietterich, 2000) είναι μια ιεραρχική αναπαράσταση του προβλήματος, στην οποία χρησιμοποιείται ένας γράφος, όπως φαίνεται στο Σχήμα 2.7, ο οποίος περιέχει δύο είδη κόμβων : Max κόμβοι και Q κόμβοι. Οι Max κόμβοι χωρίς παιδιά υποδηλώνουν αρχέγονες ενέργειες και οι Max κόμβοι με παιδιά αναπαριστούν υποδομές (sub-task). Κάθε Q κόμβος υποδηλώνει μια ενέργεια η οποία μπορεί να εκτελεστεί για να επιτύχει η υποδομή του γονέα του.



Σχήμα 2.7: MAXQ γράφος για το πρόβλημα του ταξί (Dietterich, 2000)

όπου υπάρχουν Max κόμβοι και Q κόμβοι. Οι Max κόμβοι χωρίς παιδιά υποδηλώνουν αρχέγονες ενέργειες και οι Max κόμβοι με παιδιά αναπαριστούν υποδομές (sub-task). Κάθε Q κόμβος υποδηλώνει μια ενέργεια η οποία μπορεί να εκτελεστεί για να επιτύχει η υποδομή του γονέα του.

2.9 Προηγούμενη εργασία

Είναι πολύ δύσκολο να καθοριστεί το κατάλληλο σύνολο καταστάσεων και ενεργειών σε όλα τα προβλήματα ΕΜ του πραγματικού κόσμου. Τις περισσότερες φορές απαιτείται αρκετά μεγάλη χωρητικότητα μνήμης για να καλύψει όλες τις ενδεχόμενες σχετικές καταστάσεις και ενέργειες που μπορεί να επιλεγούν καθώς μπορεί να υπάρχει μια ευρεία ποικιλία καταστάσεων, και ενεργειών που μπορούν να επιλεγούν για κάθε κατάσταση. Κατά συνέπεια υπάρχει ένα συνδυαστικό πρόβλημα έκρηξης

(combinatorial explosion problem) κατά την προσπάθεια να διερευνηθούν όλες οι πιθανές ενέργειες από όλες τις πιθανές καταστάσεις.

Πρώτος ο Singh (1992) παρουσίασε ένα αλγόριθμο και μια αρχιτεκτονική αποτελούμενη από υπομονάδες, η οποία μαθαίνει την διάσπαση μιας σύνθετης Ακολουθιακής Εργασίας Απόφασης (ΑΕΑ), και επιτυγχάνει την μεταφορά της μάθησης ανταλλάζοντας τις λύσεις μεταξύ των υπομονάδων της. Τα αποτελέσματα έδειξαν ότι η επίλυση μιας σύνθετης ΑΕΑ είναι κατασκευασμένη από μια υπολογιστικά ανέξοδη μετατροπή των λύσεων των υπομονάδων της.

Στη συνέχεια οι Dayan και Hinton (1993) έδειξαν πως μπορεί να δημιουργηθεί μια διευθυντική ιεραρχία Q-Learning (Watkins, 1989) όπου στα υψηλά επίπεδα οι διευθυντές μαθαίνουν πώς να κατανέμουν τις εργασίες στους υποδιευθυντές, οι οποίοι με την σειρά τους μαθαίνουν πώς να τους ικανοποιούν.

Η Kaelbling (1993) παρουσίασε τον αλγόριθμο HDG (Hierarchical Distance to Goal) ο οποίος χρησιμοποιεί μια ιεραρχική διάσπαση του συνόλου των καταστάσεων για να κάνει την μάθηση πιο αποδοτική.

Οι Wiering και Schmidhuber (1996), σχεδίασαν τον ιεραρχικό αλγόριθμο HQ-Learning, ο οποίος είναι μια ιεραρχική επέκταση του αλγόριθμου Q-Learning (Watkins, 1989), ο οποίος λύνει Μερικώς Παρατηρημένες Μαρκοβιανές Διαδικασίες Απόφασης (ΜΠΜΔΑ). Ο HQ-Learning βασίζεται σε μια διατεταγμένη ακολουθία από υπο-πράκτορες, όπου ο καθένας μαθαίνει να αναγνωρίζει και να επιλύει μια Μαρκοβιανή υποδομή από ολόκληρη την δομή. Κάθε πράκτορας μαθαίνει α) ένα κατάλληλο υποστόχο και β) μια Μαρκοβιανή πολιτική, έχοντας κάποιο συγκεκριμένο υποστόχο. Τέλος κατάφεραν να δείξουν ότι: α) αυτός ο αλγόριθμος μπορεί να επιλύσει προβλήματα που ο απλός αλγόριθμος Q-Learning δεν μπορεί να επιλύσει, β) μπορεί να επιλύσει μερικώς παρατηρημένους λαβύρινθους με περισσότερες καταστάσεις από αυτές που χρησιμοποιούνται στις περισσότερες προηγούμενες εργασίες με ΜΠΜΔΑ και γ) μπορεί να επιλύσει γρήγορα, πολύπλοκες δομές που χρειάζονται υπολογισμούς του περιβάλλοντος για να ελευθερώσουν ένα μπλοκαρισμένο μονοπάτι προς τον στόχο.

Ο Parr (1998) ανέπτυξε μια νέα προσέγγιση για πολιτικές ιεραρχικών δομών ΜΔΑ που τις ονόμασε HAMs (Hierarchies of Abstract Machines). Οι HAMs, εκμεταλλεύονται τη θεωρία της ημι-ΜΔΑ (όπου το χρονικό διάστημα μεταξύ μίας απόφασης και της επόμενης της καθορίζεται από μία τυχαία μεταβλητή), αλλά η έμφαση δίνεται στην απλοποίηση των περίπλοκων ΜΔΑ περιορίζοντας την κλάση των εφικτών πολιτικών, και όχι στην επέκταση των επιλογών ενέργειας. Αν και εισαγάγουν ιεραρχικά, χρονικά αφηρημένες ενέργειες, δεν διασπούν τις ΜΔΑ σε ανεξάρτητα υποπροβλήματα. Ωστόσο, συμπληρώνουν ένα νέο είδος διάσπασης, που μπορεί να διασπάσει κατά μέρος την ΜΔΑ σε ανεξάρτητα κομμάτια. Αυτή η μέθοδος διάσπασης κατασκευάζει ένα σύνολο πολιτικών για τις διάφορες περιοχές της ΜΔΑ και τις συνδυάζει για να βρει μια γενική λύση.

Η μάθηση, ο σχεδιασμός και η αναπαράσταση της γνώσης σε διάφορα επίπεδα χρονικής αφαιρετικότητας είναι μακροχρόνιες προκλήσεις στον τομέα της Τεχνητής Νοημοσύνης. Οι Sutton, Precup και Singh (1999) έδειξαν πως αυτές οι προκλήσεις μπορούν να διευθετηθούν μέσα στο πλαίσιο εργασίας της ΕΜ και των Μαρκοβιανών Διαδικασιών Απόφασης (ΜΔΑ). Επέκτειναν την συνηθισμένη έννοια της ενέργειας στο πλαίσιο εργασίας για να εισαγάγουν επιλογές-κλειστού-βρόγχου πολιτικές (options-closed-loop policies) για την εκτέλεση μιας ενέργειας μετά από μια χρονική περίοδο. Γενικά έδειξαν ότι οι επιλογές ενεργοποιούν την χρονική αφαιρετικότητα της γνώσης και της ενέργειας για να εισαχθούν στο πλαίσιο εργασίας της ΕΜ με φυσικό και γενικό τρόπο. Πιο συγκεκριμένα έδειξαν ότι οι επιλογές μπορεί να χρησιμοποιηθούν εναλλακτικά με τις αρχέγονες ενέργειες σε μεθόδους σχεδιασμού, όπως δυναμικός προγραμματισμός, και σε μεθόδους μάθησης, όπως Q-Learning.

Ο Dietterich (2000) παρουσίασε μια νέα προσέγγιση στην ιεραρχική ενισχυτική μάθηση βασισμένη στη MAXQ διάσπαση η οποία έχει και διαδικαστική σημασιολογία (σαν ιεραρχική υπο-ρουτίνα) και δηλωτική σημασιολογία (σαν αναπαράσταση της συνάρτησης αξίας της ιεραρχικής πολιτικής). Για την ιεραρχική αναπαράσταση του προβλήματος χρησιμοποιείται ένας γράφος ο οποίος περιέχει δύο είδη κόμβων : Max κόμβοι και Q κόμβοι. Οι Max κόμβοι χωρίς παιδιά υποδηλώνουν αρχέγονες ενέργειες

και οι Max κόμβοι με παιδιά αναπαριστούν υποδομές (sub-task). Κάθε Q κόμβος υποδηλώνει μια ενέργεια η οποία μπορεί να εκτελεστεί για να επιτύχει η υποδομή του γονέα του. Ο Dietterich έδειξε ότι ο MAXQ γράφος μπορεί να αναπαραστήσει την Συνάρτηση Αξίας οποιασδήποτε ιεραρχικής πολιτικής η οποία υλοποιείται από τον γράφο. Η πιο σημαντική πτυχή της μεθόδου MAXQ είναι ο διαχωρισμός μεταξύ της ανεξάρτητης από το περιβάλλον πολιτικής και της συνάρτησης αξίας (οι οποίες αναπαριστώνται από τους Max κόμβους) και της εξαρτημένης από το περιβάλλον συνάρτησης αξίας (η οποία αναπαριστάται από τους Q κόμβους). Αυτό επιτρέπει τη μάθηση της συνάρτηση αξίας των υποδομών ανεξάρτητα από το περιβάλλον τους, και αυτό ενισχύει την επαναχρησιμοποίηση των υποδομών και κάνει πιο εύκολη την αφαιρετικότητα καταστάσεων μεταξύ των υποδομών. Επίσης έδειξε ότι ο αλγόριθμος MAXQ-Q, που είναι βασισμένος στον αλγόριθμο Q-Learning, έχει πολύ καλύτερη απόδοση από τον μη ιεραρχικό αλγόριθμο Q-Learning. Το πρόβλημα που αντιμετωπίζει η μέθοδος MAXQ είναι το γεγονός ότι η ιεραρχική δομή του προβλήματος δεν γίνεται αυτόματα, αλλά πρέπει να γίνει από τον προγραμματιστή. Αυτό μερικές φορές μπορεί να θεωρηθεί σαν πλεονέκτημα επειδή όταν η διάσπαση γίνει από τον προγραμματιστή, τότε είναι πολύ πιθανό να έχει γίνει ορθά, ενώ αν ο αλγόριθμος εντοπίζει αυτόματα την κατάλληλη δομή υπάρχει πιθανότητα να μην βρεθεί η καταλληλότερη.

Οι Andre και Russell (2001) επέκτειναν την μέθοδο HAM (Parr, 1998) δημιουργώντας τη μέθοδο programmable HAM (PHAM) η οποία επιτρέπει στον χρήστη να περιορίσει τις πολιτικές οι οποίες λαμβάνονται υπόψη κατά τη διάρκεια τη διαδικασία της μάθησης. Έδειξαν ότι τα προγράμματα πρακτόρων γραμμένα με αυτή τη μέθοδο είναι σύντομα καθώς επίσης και ιεραρχικά (αποτελούμενα από υπομονάδες). Αυτό διευκολύνει την αφαιρετικότητα καταστάσεων και την μεταφορά των δεξιοτήτων που έχει μάθει ο πράκτορας. Η μέθοδος αυτή περιλαμβάνει σταθερά χαρακτηριστικά, όπως οι παραμετροποιημένες υπορουτίνες, προσωρινές διακοπές, αναστολές και μεταβλητές μνήμης. Επίσης επιτρέπει μη καθορισμένες επιλογές στο πρόγραμμα του πράκτορα.

Ο Hengst (2002) παρουσίασε τον αλγόριθμο HEXQ ο οποίος προσπαθεί να εντοπίσει αυτόματα την ιεραρχική δομή και να επιλύσει τις μικρότερες διασυνδεδεμένες ΜΔΑ πιο αποδοτικά με το να βρίσκει και να εκμεταλλεύεται επαναλαμβανόμενες υποδομές

στο περιβάλλον. Έχει σχεδιαστεί για να εντοπίζει αυτόματα την αφαιρετικότητα καταστάσεων καθώς επίσης και την χρονική αφαιρετικότητα. Ακόμη βρίσκει κατάλληλους υπο-στόχους (sub-goals) και δημιουργεί την ιεραρχική αναπαράσταση για την επίλυση της γενικής ΜΔΑ.

Οι Ghavamzadeh και Mahadevan (2004) ασχολήθηκαν με την επικοινωνία μεταξύ αυτόνομων πρακτόρων και πρότειναν ένα νέο αλγόριθμο συνεταιρισμού που τον ονόμασαν COM-Cooperative HRL. Αυτός ο αλγόριθμος εκτός από το να μαθαίνει τις δεξιότητες συντονισμού στο επίπεδο των υποδομών, περιλαμβάνει και τις αποφάσεις που λαμβάνονται και αφορούν την επικοινωνία μεταξύ των πρακτόρων. Χρησιμοποιώντας αυτό τον αλγόριθμο οι πράκτορες μαθαίνουν μια πολιτική για να ισορροπηθεί η επικοινωνία που χρειάζεται, για το κατάλληλο κόστος επικοινωνίας και συντονισμού.

Οι Marthi,Latham, Russell και Guestrin (2005) περίγραψαν μια γλώσσα για μερικώς προσδιορισμένες πολιτικές, σε πλαίσια όπου δουλεύουν μαζί πολλοί υποπράκτορες, για να μεγιστοποιήσουν μια κοινή συνάρτηση αμοιβής. Αυτή η γλώσσα επεκτείνει την γλώσσα ALisp με κατασκευαστές για συγχρονισμό και δυναμική ανάθεση υποπρακτόρων στη δομή. Κατά τη διάρκεια της μάθησης, οι υποπράκτορες μαθαίνουν μια κατανεμημένη αναπαράσταση της συνάρτησης Q για αυτή την μερική πολιτική και στη συνέχεια, κατά τη διάρκεια εκτέλεσης, συντονίζονται για να εντοπίσουν την καλύτερη κοινή ενέργεια για κάθε βήμα.

Οι Mehta και Tadepalli (2005) διερεύνησαν την διευκόλυνση της ανταλλαγής των συναρτήσεων αξίας των υποδομών, μεταξύ πολλαπλών πρακτόρων και έδειξαν ότι όταν, αυτή η προσέγγιση, συνδυαστεί με κατάλληλες πληροφορίες συντονισμού, μπορεί να αυξήσει σημαντικά την ταχύτητα της μάθησης και να την κάνει πιο επεκτάσιμη ανάλογα με τον αριθμό των πρακτόρων. Επέκτειναν το MAXQ πλαίσιο δημιουργώντας το multi-agent shared hierarchy (MASH) πλαίσιο, το οποίο επιτρέπει την επιλεκτική ανταλλαγή των συναρτήσεων αξίας των υποδομών μεταξύ των πρακτόρων, και ανάπτυξαν ένα αλγόριθμο, για αυτό το πλαίσιο, ο οποίος είναι βασισμένος σε μοντέλα μέσου όρου αμοιβών.

Μια νέα μέθοδος της IEM που ονομάζεται OMQ παρουσιάστηκε από τους Shen, Liu και Gu (2006) και ενσωματώνει τις Επιλογές (Suton et al., 1999) στην ιεραρχική μέθοδο MAXQ (Dietterich, 2000). Η μέθοδος MAXQ χρησιμοποιείται σαν το βασικό πλαίσιο εργασίας για τον εμπειρικό σχεδιασμό ιεραρχιών και οι Επιλογές χρησιμοποιούνται για την αυτόματη κατασκευή ιεραρχιών. Σύγκριναν την απόδοση της μεθόδου OMQ σε σχέση με την απόδοση των Επιλογών και της MAXQ και κατέληξαν στο συμπέρασμα ότι η μέθοδος OMQ είναι η πιο πρακτική σε μερικώς παρατηρημένα περιβάλλοντα από τις άλλες 2 μεθόδους.

Μια από τις πιο αποτελεσματικές μεθόδους στην IEM είναι η MAXQ μέθοδος (Dietterich, 2000), η οποία όμως είναι υπολογιστικά ακριβή και έχει βαθιά ιεραρχία, κάτι το οποίο την καθιστά μη πρακτική για κάποιες εφαρμογές. Οι Mirzazadeh, Behsaz, και Beigy (2007) παρουσίασαν ένα νέο αλγόριθμο ο οποίος βελτιώνει τον MAXQ-Q (Dietterich, 2000) και μειώνει την υπολογιστική πολυπλοκότητα δείχνοντας ότι ο χρόνος εκτέλεσης του είναι πολύ πιο μικρός από τον χρόνο εκτέλεσης του MAXQ-Q. Αυτός ο αλγόριθμος μαθαίνει την συνάρτηση αξίας αντί να την υπολογίζει με πλήρη έρευνα από όλα τα μονοπάτια του γράφου. Αυτό γίνεται με το να μαθαίνει και να αποθηκεύει την συνάρτηση αξίας των αρχέγονων καθώς και των σύνθετων κόμβων (και όχι μόνο των αρχέγονων κόμβων όπως γίνεται στον MAXQ-Q).

Οι Mehta, Ray, Tadepalli και Dietterich (2008) παρουσίασαν τον αλγόριθμο HI-MAT (Hierarchy Induction via Models And Trajectories) ο οποίος βρίσκει τις ιεραρχίες σε εργασίες που στηρίζονται στην MAXQ μέθοδο, εφαρμόζοντας μοντέλα Bayesian δικτύου σε μια επιτυχημένη τροχιά από την αρχική εργασία. Έδειξαν ότι ο HI-MAT κατασκευάζει συμπαγείς ιεραρχίες οι οποίες είναι συγκρίσιμες με τις ιεραρχίες οι οποίες δημιουργούνται από τον χρήστη και διευκολύνεται η επιτάχυνση της μάθησης όταν μεταφέρεται σε μια εργασία στόχο. Ο HI-MAT βρίσκει τις υποεργασίες αναλύοντας τις τυχαίες και χρονικές σχέσεις μεταξύ των ενεργειών σε αυτή την τροχιά. Κάτω από τις κατάλληλες υποθέσεις βρίσκει ιεραρχίες οι οποίες είναι συνεπής με την τροχιά η οποία παρατηρήθηκε και έχουν συμπαγείς πίνακες συνάρτησης αξίας που βοηθούν στην ασφαλή αφαιρετικότητα καταστάσεων.

Οι πιο πάνω μεθόδοι στοχεύουν να βρουν πολιτικές οι οποίες μεγιστοποιούν το εκπτωτική ολική αμοιβή μακροπρόθεσμα. Από την άλλη πλευρά, έχει αποδειχθεί ότι για μια μεγάλη κατηγορία των συνεχόμενων εργασιών είναι πιο κατάλληλο το κριτήριο του βέλτιστου μέσου όρου αμοιβών. Έτσι οι Ghavamzadeh και Mahadevan (2007) ανέπτυξαν ένα πλαίσιο εργασίας βασισμένο στο βέλτιστο μέσο όρο αμοιβών και διερεύνησαν δύο φορμαλισμούς της IEM που αντιστοιχούν σε δύο έννοιες της βελτιστοποίησης στην IEM: ιεραρχική και αναδρομική βελτιστοποίησης. Χρησιμοποιώντας αυτό το πλαίσιο παρουσίασαν τους αλγόριθμους: HAR (Hierarchically optimal Average Reward), ο οποίος βρίσκει την ιεραρχική πολιτική από τις πολιτικές που καθορίζονται από την ιεραρχική διάσπαση η οποία μεγιστοποιεί το ολικό κέρδος, και RAR (Recursively optimal Average Reward), ο οποίος μεταχειρίζεται τις σύνθετες υποδομές σαν προβλήματα συνεχόμενου μέσου όρου αμοιβών, όπου ο στόχος σε κάθε υποδομή είναι η μεγιστοποίηση του κέρδους έχοντας τις πολιτικές του παιδιού του.

Πάρα πολλή έρευνα, για την επίλυση ΜΔΑ χρησιμοποιώντας ΕΜ, στο παρελθόν, βασίστηκε στον συνδυασμό μεθόδων εκτίμησης παραμέτρων και αρχιτεκτονικών προσέγγισης συναρτήσεων για την αναπαράσταση της συνάρτησης αξίας. Τα τελευταία χρόνια υπάρχει μεγάλο ενδιαφέρον σε ένα ευρύτερο πλαίσιο το οποίο συνδυάζει την ανακάλυψη της αναπαράστασης και τον έλεγχο της μάθησης, όπου οι συναρτήσεις αξίας προσεγγίζονται χρησιμοποιώντας ένα γραμμικό συνδυασμό μιας συνάρτησης βάσης, εξαρτημένης από την εργασία η οποία μαθαίνεται κατά την διάρκεια επίλυσης μιας ξεχωριστής ΜΔΑ. Οι Osentoski και Mahadevan (2010) παρουσίασαν μια νέα προσέγγιση για την αυτόματη κατασκευή της συνάρτησης βάσης για IEM και έδειξαν ότι η μάθηση έχει σημαντική επιτάχυνση σε σχέση με τις αρχιτεκτονικές προσέγγισης συναρτήσεων.

Κεφάλαιο 3

Περιγραφή Συστήματος

3.1 Εισαγωγή

Όπως έχω προαναφέρει, ο στόχος αυτής της διπλωματικής εργασίας είναι η εξερεύνηση μεθόδων ΕΜ σε προβλήματα με μεγάλο σύνολο καταστάσεων-ενεργειών, και πιο συγκεκριμένα Ιεραρχικών μεθόδων ΕΜ. Έτσι αναπτύχθηκε ένα πλαίσιο εργασίας για την αξιολόγηση των πρακτόρων ΕΜ, χρησιμοποιώντας την γλώσσα προγραμματισμού Java, η οποία μας προσφέρει αντικειμενοστρέφεια και αρκετή επίδοση. Το κύριο μας κίνητρο ήταν η ανάπτυξη ενός γενικού και εύρωστου πλαισίου που υποστηρίζει την ανάπτυξη και την αξιολόγηση των πρακτόρων, οι οποίοι εκπαιδεύονται με ΕΜ. Αυτή η διπλωματική εργασία καταπιάνεται με το «Taxi Problem». Έτσι, χρειαζόμασταν ένα σύστημα που θα μπορούσε να υποστηρίξει οποιαδήποτε grid world προβλήματα, αφού στο μέλλον θα μπορούσε να επεκταθεί αυτή η εργασία, εξετάζοντας τους αλγόριθμους σε περισσότερα προβλήματα. Ακόμη ασχοληθήκαμε με Ενισχυτική Μάθηση Ενός Πράκτορα (ΕΜΕΠ), καθώς επίσης και με Ενισχυτική Μάθηση Πολλαπλών Πρακτόρων (ΕΜΠΠ). Έτσι το σύστημα θα έπρεπε να σχεδιαστεί με τέτοιο τρόπο, ώστε να μπορεί να λειτουργεί ορθά και στις δύο περιπτώσεις.

Επίσης, το σύστημα θα έπρεπε να σχεδιαστεί με τέτοιο τρόπο, έτσι ώστε οποιοσδήποτε θα ήθελε να το χρησιμοποιήσει για να φτιάξει τις δικές του επεκτάσεις (περιβάλλοντα,

grid world, πράκτορες, νέους αλγόριθμους), να μην χρειάζεται να ξέρει πως λειτουργεί το σύστημα, αλλά να πρέπει μόνο να επεκτείνει συγκεκριμένες κλάσεις, γράφοντας τον δικό του κώδικα, χωρίς να ανησυχεί για τα άλλα συστατικά του συστήματος.

Εν κατακλείδι, το σύστημα αποτελείται από δύο επιμέρους συστήματα: α) σύστημα ΕΜΕΠ, που ασχολείται αποκλειστικά με την εκπαίδευση ενός πράκτορα και β) σύστημα ΕΜΠΠ, που ασχολείται με την εκπαίδευση πολλαπλών πρακτόρων, το οποίο όμως μπορεί να χρησιμοποιηθεί και για την εκπαίδευση ενός πράκτορα.

3.2 Ενισχυτική Μάθηση Ενός Πράκτορα

3.2.1 Αλγόριθμοι που υλοποιήθηκαν

3.2.1.1 Αλγόριθμος Q-Learning Ενός Πράκτορα

Στον αλγόριθμο Q-Learning αρχικά έχουμε την φάση αρχικοποιήσεων στην οποία αρχικοποιούνται οι ενέργειες, τα ενισχυτικά σήματα και η κατάσταση του περιβάλλοντος με κάποιες αρχικές τιμές. Στην συνέχεια, για κάθε επεισόδιο έχουμε την εξεύρεση της αρχικής κατάστασης. Ακολούθως, για κάθε βήμα του επεισοδίου, ο πράκτορας επιλέγει μια ενέργεια με βάση την κατάσταση του περιβάλλοντος και την πολιτική που ακολουθεί. Στη συνέχεια, το περιβάλλον λαμβάνει την ενέργεια του πράκτορα, διαμορφώνει την κατάσταση του και την επιστρέφει στον πράκτορα, δίνοντας του και το κατάλληλο ενισχυτικό σήμα, που αντιστοιχεί στην αμοιβή. Τέλος, ο πράκτορας ενημερώνει την Συνάρτηση ενέργειας-αξίας.

Ακολουθεί στο Σχήμα 3.1 ο ψευδοκώδικας του αλγόριθμου αυτού:

Αρχικοποίηση των τιμών της συνάρτησης ενέργειας-αξίας Q(s,a) αυθαίρετα

Επανέλαβε (για κάθε επεισόδιο):

Αρχικοποίησε την αρχική κατάσταση s

Επανέλαβε (για κάθε βήμα του επεισοδίου):

Επέλεξε μια ενέργεια α από την κατάσταση s χρησιμοποιώντας μια πολιτική η οποία απορρέει από την Q (π.χ ε-greedy)

Εκτέλεσε την ενέργεια α, παρατήρησε την αμοιβή r και την επόμενη κατάσταση s'

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

s←s';

Μέχρι το s να είναι τερματικό

Σχήμα 3.1: Αλγόριθμος Q-Learning Ενός Πράκτορα

Σύμφωνα με τον αλγόριθμο διατηρούμε τιμές *Q* τις οποίες αρχικοποιούμε αυθαίρετα. Ακολούθως επαναλαμβάνεται η εξής διαδικασία: Από την κατάσταση *s* επιλέγουμε την ενέργεια α σύμφωνα με κάποια πολιτική. Εφόσον πάρουμε την αμοιβή και την επόμενη κατάσταση ανανεώνουμε την *Q* τιμή της κατάστασης *s* με ενέργεια *α* σύμφωνα με τον τύπο στο σχήμα.

3.2.1.2 Αλγόριθμος SARSA Ενός Πράκτορα

Ο αλγόριθμος SARSA είναι παρόμοιος με τον αλγόριθμο Q-Learning. Η διαφορά τους έγκειται στον τρόπο με τον οποίο γίνεται η ενημέρωση της συνάρτησης ενέργειας-αξίας από τον πράκτορα. Συγκεκριμένα σε αυτό τον αλγόριθμο έχουμε αρχικά την φάση αρχικοποιήσεων στην οποία αρχικοποιούνται οι ενέργειες, τα ενισχυτικά σήματα και η κατάσταση του περιβάλλοντος με κάποιες αρχικές τιμές. Στην συνέχεια, για κάθε επεισόδιο έχουμε την εξεύρεση της αρχικής κατάστασης και την επιλογή μιας ενέργειας από τον πράκτορα με βάση την αρχική κατάσταση του περιβάλλοντος και την πολιτική που ακολουθεί. Ακολούθως, για κάθε βήμα του επεισοδίου, το περιβάλλον λαμβάνει την ενέργεια του πράκτορα, διαμορφώνει την κατάσταση του και την επιστρέφει στον πράκτορα, δίνοντας του και το κατάλληλο ενισχυτικό σήμα, που αντιστοιχεί στην αμοιβή. Τότε, ο πράκτορας επιλέγει την επόμενη ενέργεια με βάση την επόμενη

κατάσταση του περιβάλλοντος και την πολιτική που ακολουθεί. Τέλος, ο πράκτορας ενημερώνει την Συνάρτηση ενέργειας-αξίας.

Ακολουθεί στο Σχήμα 3.2 ο ψευδοκώδικας του αλγόριθμου αυτού:

Αρχικοποίηση των τιμών της συνάρτησης ενέργειας-αξίας Q(s,a) αυθαίρετα
Επανέλαβε (για κάθε επεισόδιο):
Αρχικοποίησε την αρχική κατάσταση s
Επέλεξε μια ενέργεια α από την κατάσταση s χρησιμοποιώντας μια πολιτική
η οποία απορρέει από την Q (π.χ ε-greedy)
Επανέλαβε (για κάθε βήμα του επεισοδίου):
Εκτέλεσε την ενέργεια α , παρατήρησε την αμοιβή r και την επόμενη κατάσταση s'
Επέλεξε μια ενέργεια α' από την κατάσταση s' χρησιμοποιώντας μια πολιτική
η οποία απορρέει από την Q (π.χ ε-greedy)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

s←s'; α←α';
Μέχρι το s να είναι τερματικό

Σχήμα 3.2: Αλγόριθμος SARSA Ενός Πράκτορα

Σύμφωνα με τον αλγόριθμο διατηρούμε τιμές *Q* τις οποίες αρχικοποιούμε αυθαίρετα. Ακολούθως επαναλαμβάνεται η εξής διαδικασία: Από την κατάσταση *s* επιλέγουμε την ενέργεια α σύμφωνα με κάποια πολιτική. Εφόσον πάρουμε την αμοιβή και την επόμενη κατάσταση ανανεώνουμε την *Q* τιμή της κατάστασης *s* με ενέργεια *α* σύμφωνα με τον τύπο στο σχήμα.

3.2.1.3 Αλγόριθμος PHC Ενός Πράκτορα

Ο αλγόριθμος Policy Hill-Climbing (PHC) είναι μια απλή επέκταση του αλγόριθμου Q-Learning, που εφαρμόζεται στα παίγνια μικτής στρατηγικής. Ο PHC στην ουσία κάνει hill-climbing στον χώρο των μικτών στρατηγικών. Οι *Q* τιμές διατηρούνται όπως και στον Q-Learning. Επιπρόσθετα, ο αλγόριθμος διατηρεί την τρέχουσα μικτή πολιτική. Η

πολιτική βελτιώνεται με την αύξηση της πιθανότητας ότι θα επιλέξει την ενέργεια με την υψηλότερη τιμή, σύμφωνα με ένα ρυθμό μάθησης $\delta \in (0,1]$. Όταν $\delta=1$, ο αλγόριθμος είναι ισοδύναμος με τον Q-Learning, μιας και με κάθε βήμα η πολιτική κινείται στην άπληστη επιλογή, επιλέγοντας με πιθανότητα 1 την ενέργεια με την υψηλότερη αμοιβή.

Ακολουθεί στο Σχήμα 3.3 ο ψευδοκώδικας του αλγόριθμου αυτού.

---

1. Έστω α και δ ρυθμοί μάθησης. Αρχικοποίησε,

$$Q(s,a) \leftarrow 0,\ \pi(s,a) \leftarrow \frac{1}{|A_i|}$$

2. Επανέλαβε,

(α) Από την κατάσταση S, επέλεξε ενέργεια α με πιθανότητα π(s,α), με κάποια εξερεύνηση

(β) Παρατήρησε την ανταμοιβή r και επόμενη κατάσταση s` και υπολόγισε

$$Q(s,\alpha) = (1-\alpha)Q(s,\alpha) + \alpha\,(r + \gamma \max_{\alpha'} Q(s',\alpha'))$$

(γ) Ενημέρωσε το π(s,a) με

$$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta, & \text{αν } \alpha = argmax_{a'} Q(s,a') \\ |A_i| - 1, & \text{αλλιώς} \end{cases}$$

Σχήμα 3.3: Αλγόριθμος Policy Hill-Climbing (PHC) για παίχτη i

Σύμφωνα με τον αλγόριθμο διατηρούμε τιμές Q τις οποίες αρχικοποιούμε με 0 και πιθανότητες για κάθε ζεύγος κατάστασης-ενέργειας τις οποίες αρχικοποιούμε με 1/Πλήθος ενεργειών του συγκεκριμένου παίχτη. Ακολούθως επαναλαμβάνεται η εξής διαδικασία: Από την κατάσταση s επιλέγουμε την ενέργεια α με πιθανότητα $\pi(s,\alpha)$, κάνοντας όμως και χρήση της πιθανότητας για εξερεύνηση. Εφόσον πάρουμε την αμοιβή και την επόμενη κατάσταση ανανεώνουμε την Q τιμή της κατάστασης s με ενέργεια α σύμφωνα με τον τύπο στο σχήμα, καθώς και την πιθανότητα $\pi(s,\alpha)$, όπως δείχνει ο αλγόριθμος στο σχήμα.

3.2.1.4 Αλγόριθμος WoLF-PHC Ενός Πράκτορα

Ο αλγόριθμος WoLF Policy Hill-Climbing είναι μια επέκταση του αλγόριθμου PHC. Οι σημαντικές διαφοροποιήσεις του σε σχέση με τον PHC είναι η χρήση μεταβλητού ρυθμού μάθησης και η αρχή WoLF. Η αρχή του WoLF δηλώνει «Μάθε γρήγορα ενόσω χάνεις και αργά ενόσω νικάς» (“Win Or Lose Fast Policy Hill-Climbing”). Η μέθοδος, για να διευκρινίζουμε πότε νικά ο πράκτορας και πότε όχι, συγκρίνει την προβλεπόμενη αμοιβή της τρέχουσας πολιτικής με αυτή της μέσης πολιτικής στο χρόνο.

Ακολουθεί στο Σχήμα 3.4 ο ψευδοκώδικας του αλγόριθμου αυτού.

1. Έστω α και δl > δw ρυθμοί μάθησης. Αρχικοποίησε,

$$Q(s,a) \leftarrow 0,\ \pi(s,a) \leftarrow \frac{1}{|A_i|},\ C(s) \leftarrow 0$$

2. Επανέλαβε,

(α) Από την κατάσταση S, επέλεξε ενέργεια α με πιθανότητα π(s,α), με κάποια εξερεύνηση

(β) Παρατήρησε την ανταμοιβή r και επόμενη κατάσταση s` και υπολόγισε

$$Q(s,\alpha) = (1-\alpha)Q(s,\alpha) + \alpha\,(r + \gamma \max_{a'} Q(s',a'))$$

(γ) Ενημέρωσε την αναμενόμενη πιθανότητα της μέσης πολιτικής $\bar{\pi}$

$$C(s) \leftarrow C(s)+1$$

$$\forall\, a' \in Ai\quad \bar{\pi}(s,a') \leftarrow \bar{\pi}(s,a') + \frac{1}{C(s)}\,(\pi(s,a') - \bar{\pi}(s,a'))$$

(δ) Ενημέρωσε $\pi(s,\alpha)$ βάση

$$\pi(s,\alpha) = \pi(s,\alpha) + \begin{cases} \delta, & \text{αν } \alpha = argmax_a, Q(s,a') \\ \dfrac{-\delta}{|A_i|-1}, & \text{αλλιώς} \end{cases}$$

Όπου,

$$\delta = \begin{cases} \delta w, \text{αν } \sum_a \pi(s,\alpha)Q(s,a) > \sum_a \bar{\pi}(s,\alpha)Q(s,a) \\ \delta l, \text{αλλιώς} \end{cases}$$

Σχήμα 3.4: Αλγόριθμος WoLF-Policy Hill-Climbing (WoLF-PHC) για παίχτη i

Ο αλγόριθμος απαιτεί 2 ρυθμούς μάθησης *δl* > *δw*. Όταν ο παίχτης νικά χρησιμοποιείται το *δw* και στην αντίθετη περίπτωση το *δl*. Αυτό εντοπίζεται συγκρίνοντας την αναμενόμενη τιμή χρησιμοποιώντας τις τρέχουσες *Q* τιμές ακολουθώντας την τρέχουσα πολιτική *π* στην τρέχουσα κατάσταση με αυτό ακολουθώντας την μέση πολιτική $\bar{\pi}$. Αν η αναμενόμενη τιμή από την τρέχουσα πολιτική είναι μικρότερη (ο παίχτης χάνει), ο μεγαλύτερος ρυθμός μάθησης χρησιμοποιείται.

3.2.1.5 Αλγόριθμος MAXQ-Q Ενός Πράκτορα

Ο αλγόριθμος MAXQ-Q χρησιμοποιεί την Ιεραρχική μέθοδο MAXQ, η οποία διασπά το πρόβλημα σε μικρότερες ΜΔΑ (Μ) και είναι μια από τις πιο αποτελεσματικές μεθόδους στην ΙΕΜ. Επίσης είναι βασισμένος στον αλγόριθμο ΧΔ Q-Learning. Για την ιεραρχική αναπαράσταση του προβλήματος χρησιμοποιείται ένας γράφος ο οποίος περιέχει δύο είδη κόμβων : Max κόμβοι και Q κόμβοι. Οι Max κόμβοι χωρίς παιδιά υποδηλώνουν αρχέγονες ενέργειες και οι Max κόμβοι με παιδιά αναπαριστούν υποδομές (sub-task). Κάθε Q κόμβος υποδηλώνει μια ενέργεια η οποία μπορεί να εκτελεστεί για να επιτύχει η υποδομή του γονέα του.

Κάθε αρχέγονος Max κόμβος έχει την δική του Συνάρτηση Αξίας, ενώ η τιμή της Συνάρτηση Αξίας στους σύνθετους Max κόμβους υπολογίζεται με την αναδρομική συνάρτηση EvaluateMaxNode(i,s) του σχήματος 3.6.

Κάθε Q κόμβος έχει δύο Συναρτήσεις Ολοκλήρωσης (completion function): α) την «εξωτερική» συνάρτηση ολοκλήρωσης $C(i,s,a)$ και β) την «εσωτερική» συνάρτηση ολοκλήρωσης $\hat{C}(i,s,a)$.

Η «εξωτερική» συνάρτηση ολοκλήρωσης $C(i,s,a)$ υπολογίζει την αναμενόμενη αμοιβή για την ολοκλήρωση της δομής Mi, μετά την εκτέλεση της ενέργειας a στην κατάσταση s, ακολουθώντας την πολιτική για την δομή Mi. Στη συνέχεια χρησιμοποιείται από την δομή «πατέρας» της Mi για να υπολογίσει την τιμή $V(i,s)$, που είναι η αναμενόμενη αμοιβή για την εκτέλεση της ενέργειας i ξεκινώντας από την κατάσταση s.
Η «εσωτερική» συνάρτηση ολοκλήρωσης $\hat{C}(i,s,a)$ είναι μια συνάρτηση ολοκλήρωσης η οποία χρησιμοποιείται μόνο «μέσα» στον κόμβο i, προκειμένου να ανακαλύψει την τοπικά βέλτιστη πολιτική για τη δομή Mi.

Ακολουθεί στο Σχήμα 3.5 ο ψευδοκώδικας του αλγόριθμου αυτού.

```
function MAXQ-Q(MaxNode i, State s)

1       let seq = () be the sequence of states visited while executing i
2       if  i is a primitive MaxNode
3               execute i. receive r. and observe result state s'
4               V_{t+1}(i,s) := (1 − α_t(i)) · V_t(i,s) + α_t(i) · r_t
5               push s onto the beginning of seq
6       else
7               let count = 0
8               while Ti(s) is false do
9                       choose an action a according to the current exploration policy πx(i, s)
10                      let childSeq = MAXQ-Q(a, s), where childSeq is the sequence of states
                        visited  while executing action a. (in reverse order)
11                      observe result state s'
12                      let a* = argmax_{a'} [C̃_t(i,s',a') + V_t(a',s')]
13                      let N = 1
14                      for each s in childSeq do
15                          C̃_{t+1}(i,s,a) := (1 − α_t(i)) · C̃_t(i,s,a) + α_t(i) · γ^N [R̃_i(s') + C̃_t(i,s',a*) + V_t(a*,s)]
16                          C_{t+1}(i,s,a) := (1 − α_t(i)) · C_t(i,s,a) + α_t(i) · γ^N [C_t(i,s',a*) + V_t(a*,s')]
17                          N := N + 1
18                      end // for
19                      append childSeq onto the front of seq
20                      s := s'
21                      end // while
22              end // else
23      return seq
        end MAXQ-Q
```

Σχήμα 3.5: Αλγόριθμος MAXQ-Q Ενός Πράκτορα

Ο αλγόριθμος είναι ένας αναδρομικός αλγόριθμος. Η εκτέλεση μιας αναδρομικής κλήσης τερματίζει είτε όταν ο Max κόμβος που βρισκόμαστε είναι αρχέγονος κόμβος, είτε όταν βρισκόμαστε σε τερματική κατάσταση. Χρησιμοποιείται μια λίστα seq η οποία αποθηκεύει την ακολουθία με τις καταστάσεις οι οποίες έχουν επισκεφτεί κατά την εκτέλεση του Max κόμβου *i*. Επίσης χρησιμοποιείται η λίστα *childSeq* στην οποία αποθηκεύεται η λίστα *seq* του κάθε Max κόμβου *i* όταν τερματίσει μια αναδρομική κλήση. Κατά την εκτέλεση του αλγόριθμου, αν ο Max κόμβος δεν είναι αρχέγονος και η κατάσταση που βρισκόμαστε δεν είναι τερματική, τότε γίνεται επιλογή ενέργειας σύμφωνα με μια πολιτική και γίνεται ξανά αναδρομική κλήση του αλγόριθμου σύμφωνα με την νέα ενέργεια που επιλέγηκε. Αν ο Max κόμβος είναι αρχέγονος τότε γίνεται εκτέλεση της ενέργειας και παρατηρείται η αμοιβή και η επόμενη κατάσταση. Στη συνέχεια ανανεώνεται η τιμή της συνάρτησης αξίας *V* της κατάστασης *s* σύμφωνα με τον τύπο στο σχήμα. Μετά τον τερματισμό μιας αναδρομικής κλήσης γίνεται παρατήρηση της επόμενης κατάστασης *s'* και γίνεται εύρεση της άπληστης ενέργειας *α** για την κατάσταση *s'* σύμφωνα με τον τύπο στο σχήμα. Στη συνέχεια, για κάθε κατάσταση *s* που βρίσκεται στη λίστα *childSeq* γίνεται ανανέωση των δύο συναρτήσεων ολοκλήρωσης (completion function) *C* και *Ĉ* σύμφωνα με τους τύπους στο σχήμα, όπου *R̂i(s')* είναι ψευδοαμοιβή για την κατάσταση *s'*.

Ακολουθεί στο Σχήμα 3.6 ο ψευδοκώδικας της συνάρτησης EvaluateMaxNode(i,s).

```
function EvaluateMaxNode(i, s)

    if i is a primitive Max node
        return ⟨V_t(i, s), i⟩
    else
        for each j ∈ A_i,
            let ⟨V_t(j, s), a_j⟩ = EvaluateMaxNode(j, s)
        let j^hg = argmax_j V_t(j, s) + C_t(i, s, j)
        return ⟨V_t(j^hg, s), a_{j^hg}⟩
end // EvaluateMaxNode
```

Σχήμα 3.6: Συνάρτηση EvaluateMaxNode(i,s)

Η συνάρτηση αυτή είναι αναδρομική και κάνει ένα πλήρη έλεγχο από όλα τα μονοπάτια του MAXQ γράφου, χρησιμοποιώντας προθεματική διάσχιση, ξεκινώντας από τον κόμβο *i* και καταλήγοντας σε κόμβους «φύλλα». Η εκτέλεση μιας αναδρομικής κλήσης τερματίζει όταν ο Max κόμβος που βρισκόμαστε είναι αρχέγονος κόμβος, επιστρέφοντας την τιμή της Συνάρτησης Αξίας, $V_t(i,s)$, του κόμβου αυτού, καθώς επίσης και την ενέργεια *i* η οποία οδηγεί σε αυτή την τιμή. Αν δεν βρισκόμαστε σε αρχέγονο κόμβο, τότε γίνεται αναδρομική κλήση της συνάρτησης EvaluateMaxNode(i,s) για κάθε ενέργεια η οποία μπορεί να επιλεγεί από τον συγκεκριμένο κόμβο. Όταν γίνει ο έλεγχος για όλες τις ενέργειες που μπορούν να επιλεγούν από ένα Max κόμβο, επιλέγεται η ενέργεια η οποία μεγιστοποιεί την τιμή της $V_t(i,s)$ σύμφωνα με τον τύπο στο σχήμα.

3.2.1.6 Αλγόριθμος MAXQ-PHC Ενός Πράκτορα

Ο αλγόριθμος MAXQ-PHC είναι μια Ιεραρχική υλοποίηση του αλγόριθμου PHC χρησιμοποιώντας την MAXQ Ιεραρχική διάσπαση. Είναι παρόμοιος με τον αλγόριθμο MAXQ-Q, με την διαφορά ότι η επιλογή της κάθε ενέργειας γίνεται με βάση κάποιων πιθανοτήτων.

Ακολουθεί στο Σχήμα 3.7 ο ψευδοκώδικας του αλγόριθμου αυτού.

function MAXQ-PHC(MaxNode i, State s)

1 let seq = () be the sequence of states visited while executing i
2 if i is a primitive MaxNode
3 execute i. receive r. and observe result state s'
4 $V_{t+1}(i,s) := (1-\alpha_t(i)) \cdot V_t(i,s) + \alpha_t(i) \cdot r_t$
5 push s onto the beginning of seq
6 else
7 let count = 0
8 while Ti(s) is false do
9 from state s select action a, with probability π(s,a) with some exploration
10 update π(s,a) and constrain it to a legal probability distribution,
11 $\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}$
12 let childSeq = MAXQ-Q(a, s), where childSeq is the sequence of states visited while executing action a. (in reverse order)
13 observe result state s'
14 $\textbf{let } a^* = \text{argmax}_{a'} [\tilde{C}_t(i,s',a') + V_t(a',s')]$
15 let N = 1
16 for each s in childSeq do
17 $\tilde{C}_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot \tilde{C}_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s)]$
18 $C_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [C_t(i,s',a^*) + V_t(a^*,s')]$
19 N := N + 1
20 end // for
21 append childSeq onto the front of seq
22 s := s'
23 end // while
24 end // else
25 return seq
end MAXQ- PHC

Σχήμα 3.7: Αλγόριθμος MAXQ-PHC Ενός Πράκτορα

Σύμφωνα με τον αλγόριθμο διατηρούμε πιθανότητες για κάθε ζεύγος κατάστασης-ενέργειας τις οποίες αρχικοποιούμε με 1/Πλήθος ενεργειών. Ο αλγόριθμος εξηγήθηκε στο σχήμα 3.5. Η διαφορά είναι ότι στον αλγόριθμο MAXQ-PHC η επιλογή ενέργειας γίνεται με βάση κάποιων πιθανοτήτων (γραμμή 9), και μετά από κάθε επιλογή ενέργειας γίνεται ανανέωση της πιθανότητας *π(s,α)*, όπως δείχνει ο αλγόριθμος στο σχήμα (γραμμή 10 και 11), όπου *δ* είναι ο ρυθμός μάθησης.

3.2.1.7 Αλγόριθμος MAXQ-WoLF-PHC Ενός Πράκτορα

Ο αλγόριθμος MAXQ-WoLF-PHC είναι μια επέκταση του αλγόριθμου MAXQ-PHC και μια Ιεραρχική υλοποίηση του αλγόριθμου WoLF-PHC χρησιμοποιώντας την MAXQ Ιεραρχική διάσπαση. Συγκεκριμένα ο αλγόριθμος απαιτεί 2 ρυθμούς μάθησης $\delta l > \delta w$. Όταν ο παίχτης νικά, χρησιμοποιείται το δw και στην αντίθετη περίπτωση το δl. Αυτό εντοπίζεται συγκρίνοντας την αναμενόμενη τιμή, χρησιμοποιώντας τις τρέχουσες Q τιμές, ακολουθώντας την τρέχουσα πολιτική π, στην τρέχουσα κατάσταση με αυτό, ακολουθώντας την μέση πολιτική $\bar{\pi}$.Αν η αναμενόμενη τιμή από την τρέχουσα πολιτική είναι μικρότερη (ο παίχτης χάνει), ο μεγαλύτερος ρυθμός μάθησης χρησιμοποιείται.

Ακολουθεί στο Σχήμα 3.8 ο ψευδοκώδικας του αλγόριθμου αυτού.

---

function MAXQ- WoLF -PHC (MaxNode i, State s)

1 let seq = () be the sequence of states visited while executing i
2 if i is a primitive MaxNode
3 execute i. receive r. and observe result state s'
4 $V_{t+1}(i,s) := (1-\alpha_t(i)) \cdot V_t(i,s) + \alpha_t(i) \cdot r_t$
5 push s onto the beginning of seq
6 else
7 let count = 0
8 while Ti(s) is false do
9 from state s select action a, with probability π(s,a) with some exploration
10 update estimate of average policy, $\bar{\pi}$
$C(s) \leftarrow C(s)+1$
$\forall\, \alpha' \in Ai \quad \bar{\pi}(s,\alpha') \leftarrow \bar{\pi}(s,\alpha') + \frac{1}{C(s)} (\pi(s,\alpha') - \bar{\pi}(s,\alpha'))$
11 update π(s,a) and constrain it to a legal probability distribution,
12 $\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}$
where
$\delta = \begin{cases} \delta_w & \text{if } \sum_a \pi(s,a)Q(s,a) > \sum_a \bar{\pi}(s,a)Q(s,a) \\ \delta_l & \text{otherwise} \end{cases}$
13 let childSeq = MAXQ-Q(a, s), where childSeq is the sequence of states visited while executing action a. (in reverse order)
14 observe result state s'
15 $\textbf{let } a^* = \text{argmax}_{a'} [\tilde{C}_t(i,s',a') + V_t(a',s')]$
16 let N = 1
17 for each s in childSeq do
18 $\tilde{C}_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot \tilde{C}_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s)]$
19 $C_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [C_t(i,s',a^*) + V_t(a^*,s')]$
20 N := N + 1
21 end // for
22 append childSeq onto the front of seq
23 s := s'
24 end // while
25 end // else
26 return seq
end MAXQ- WoLF -PHC

Σχήμα 3.8: Αλγόριθμος MAXQ- WoLF -PHC

Σύμφωνα με τον αλγόριθμο διατηρούμε πιθανότητες και τιμή μέσης πιθανότητας για κάθε ζεύγος κατάστασης-ενέργειας τις οποίες αρχικοποιούμε με 1/Πλήθος ενεργειών. Επίσης διατηρούμε και ένα μετρητή για κάθε κατάσταση ο οποίος αρχικοποιείται με 0 και αυξάνεται κατά 1, κάθε φορά που επισκεπτόμαστε την κατάσταση αυτή. Ο αλγόριθμος εξηγήθηκε στο σχήμα 3.5. Η διαφορά είναι ότι στον αλγόριθμο MAXQ-WoLF -PHC η επιλογή ενέργειας γίνεται με βάση κάποιων πιθανοτήτων (γραμμή 9), και μετά από κάθε επιλογή ενέργειας γίνεται ανανέωση της μέσης πιθανότητας $\bar{\pi}$ και της πιθανότητας *π(s,α)*, όπως δείχνει ο αλγόριθμος στο σχήμα (γραμμή 10 - 12).

3.2.2 Συστατικά Συστήματος

Το σύστημα αποτελείται από τα εξής κύρια συστατικά:

1. Simulation
2. Environment,
3. SA_Gridworld,
4. Graph,
5. MaxQ,
6. Q_Learning,
7. Sarsa,
8. Phc,
9. Wolf,
10. MaxQ_Q,
11. MaxQ_PHC,
12. MaxQ_WoLF_PHC

Η κλάση Simulation είναι το κύριο συστατικό του συστήματος μας. Υλοποιήθηκε σαν Αφαιρετική Κλάση (Abstract Class), για να αναγκάζει την επέκταση της και έτσι επιτρέπει τη χρήση πολυμορφισμού. Μια κλάση που επεκτείνει την Simulation, μπορεί είτε να αφορά προσομοίωση αλγορίθμων ενός πράκτορα, είτε προσομοίωση αλγορίθμων πολλαπλών πρακτόρων. Στο σύστημα υλοποιήσαμε την κλάση SA_Simulation (Single Agent Simulation) η οποία κληρονομεί από την κλάση Simulation. Η SA_Simulation αλληλεπιδρά με τον χρήστη δίνοντας του την ευκαιρία να επιλέξει τον αλγόριθμο που θέλει να προσομοιώσει. Συλλέγει της πληροφορίες και τις στέλνει σε ένα αντικείμενο της κλάσης RunTrial, το οποίο είναι υπεύθυνο για να κάνει αυτή την προσομοίωση. Επίσης επικοινωνεί με ένα αντικείμενο της κλάσης PlotGraph το οποίο είναι υπεύθυνο για την δημιουργία των γραφικών παραστάσεων.

Η κλάση Environment είναι το κύριο συστατικό για το περιβάλλον που θα χρησιμοποιηθεί. Υλοποιήθηκε σαν Αφαιρετική Κλάση, για να αναγκάζει την επέκταση της από οποιοδήποτε περιβάλλον. Επικοινωνεί με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να

μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει τη νέα κατάσταση. Στο σύστημα που υλοποιήσαμε δημιουργήσαμε μια κλάση SA_Taxi_Problem (Single Agent Taxi_Problem) η οποία κληρονομεί από την κλάση Environment και επικοινωνεί με την κλάση SA_Gridworld, αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ο επιβάτης.

Η κλάση SA_Gridworld (Single Agent Gridworld) αντιπροσωπεύει το πλαίσιο του προβλήματος, αφού διαβάζοντας ένα αντιπροσωπευτικό αρχείο το δημιουργεί. Επικοινωνεί με την κλάση Wall, αφού καθορίζει που υπάρχουν εμπόδια στο πλαίσιο. Επίσης σε αυτή την κλάση υπάρχουν οι επιτρεπτές τοποθεσίες για τον επιβάτη και οι καταστάσεις του προβλήματος.

Η κλάση Graph είναι το κύριο συστατικό του συστήματος που αντιπροσωπεύει την ιεραρχική αναπαράσταση του προβλήματος μας. Έχει υλοποιηθεί σαν Αφαιρετική Κλάση για να αναγκάζει την επέκταση της και έτσι επιτρέπει τη χρήση πολυμορφισμού. Μια κλάση που επεκτείνει την Graph μπορεί να αναπαραστήσει ιεραρχικά οποιοδήποτε άλλο πρόβλημα. Σε αυτό το σύστημα έχει υλοποιηθεί η κλάση SA_Graph (Single Agent Graph), η οποία υλοποιεί τον γράφο για το TP για ένα πράκτορα. Επικοινωνεί με τις κλάσεις Node, MaxNode, MaxNodePHC, MaxNodeWoLFPHC και Qnode, αφού χρειάζεται να δημιουργήσει ένα γράφο με Max και Q κόμβους. Η κλάση MaxNode κληρονομεί από την κλάση Node, η κλάση MaxNodePHC κληρονομεί από την κλάση MaxNode και η κλάση MaxNodeWoLFPHC κληρονομεί από την κλάση MaxNodePHC. Θα εξηγηθεί στην συνέχεια που χρησιμοποιούνται αυτές οι κλάσεις. Επίσης επικοινωνεί με την κλάση SA_Gridworld, αφού χρειάζεται να γνωρίζει τις καταστάσεις του προβλήματος.

Η κλάση MaxQ είναι το κύριο συστατικό για το περιβάλλον που θα χρησιμοποιηθεί για τους Ιεραρχικούς αλγόριθμους. Κληρονομεί από την κλάση Graph αφού χρειάζεται να ξέρει την ιεραρχική αναπαράσταση του προβλήματος. Επίσης επικοινωνεί με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει

τη νέα κατάσταση. Ακόμη επικοινωνεί με την κλάση SA_Gridworld, αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ο επιβάτης.

Η κλαση Q_Learning είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος Q-Learning. Κληρονομεί από την κλάση SA_Taxi_Problem, αφού το πρόβλημα που είχαμε να υλοποιήσουμε αφορούσε το Taxi Problem. Επικοινωνεί με την κλάση SA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος.

Η κλαση Sarsa είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος Sarsa. Κληρονομεί από την κλάση SA_Taxi_Problem, αφού το πρόβλημα που είχαμε να υλοποιήσουμε αφορούσε το Taxi Problem. Επικοινωνεί με την κλάση SA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος.

Η κλαση Phc είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος PHC (Policy Hill-Climbing). Κληρονομεί από την κλάση SA_Taxi_Problem, αφού το πρόβλημα που είχαμε να υλοποιήσουμε αφορούσε το Taxi Problem. Επικοινωνεί με την κλάση SA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος.

Η κλαση Wolf είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος WoLF-PHC (WoLF Policy Hill-Climbing). Κληρονομεί από την κλάση Phc, αφού ο αλγόριθμος WoLF-PHC είναι μια επέκταση του αλγόριθμου PHC. Επικοινωνεί με την κλάση SA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος.

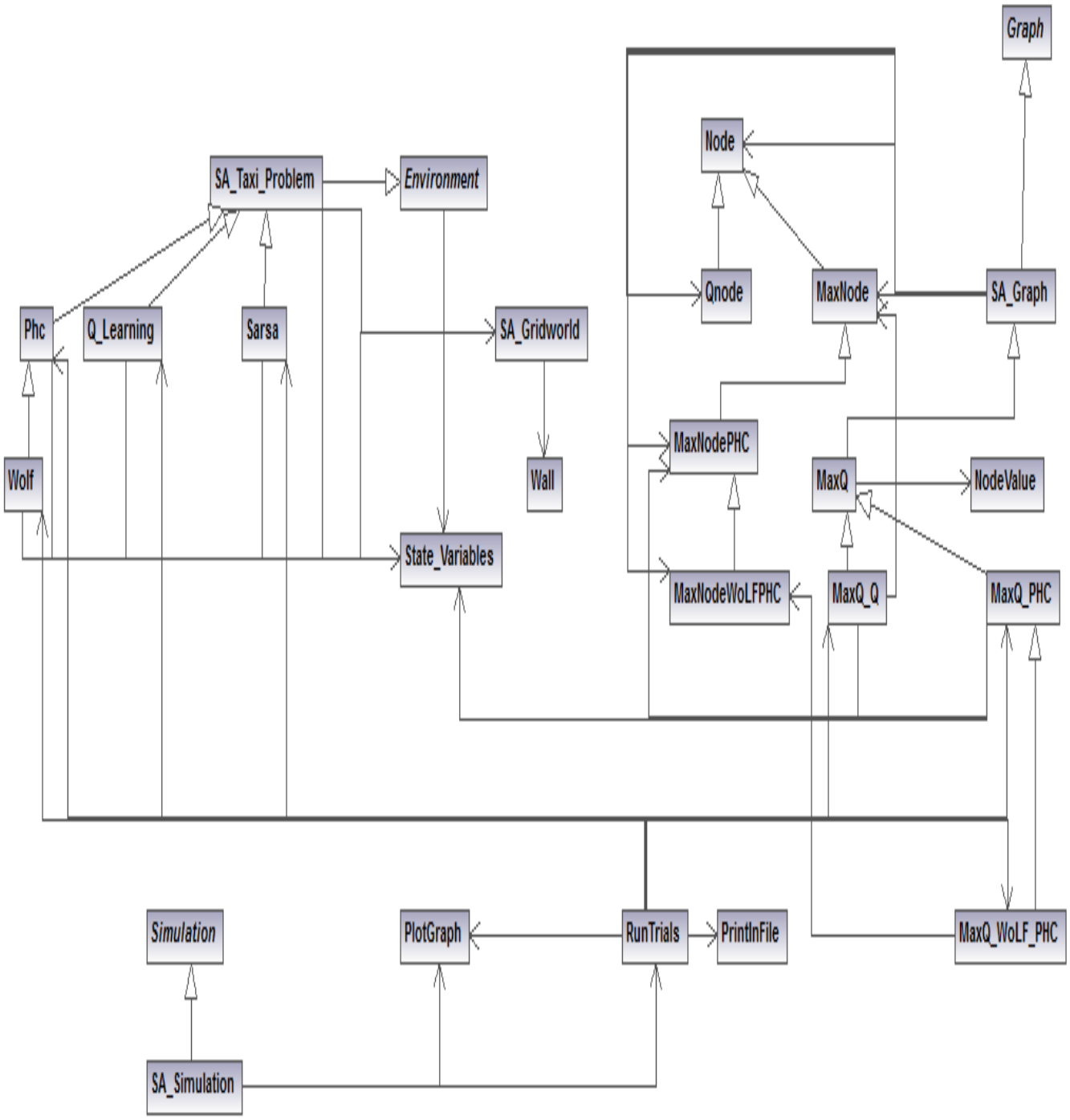
Η κλαση MaxQ_Q είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος MaxQ-Q. Κληρονομεί από την κλάση MaxQ, η οποία αντιπροσωπεύει την ιεραρχική

αναπαράσταση του προβλήματος μας. Επικοινωνεί με την κλάση SA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Επίσης επικοινωνεί με τις κλάσεις MaxNode και Qnode αφού χρησιμοποιεί τους MaxNode και Qnode του γράφου για την επίλυση του προβλήματος.

Η κλαση MaxQ_PHC είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος MaxQ-PHC. Κληρονομεί από την κλάση MaxQ, η οποία αντιπροσωπεύει την ιεραρχική αναπαράσταση του προβλήματος μας. Επικοινωνεί με την κλάση SA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Επίσης επικοινωνεί με τις κλάσεις MaxNodePHC και Qnode αφού χρησιμοποιεί τους MaxNodePHC και Qnode του γράφου για την επίλυση του προβλήματος.

Η κλαση MaxQ_WoLF_PHC είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος MaxQ-WoLF-PHC. Κληρονομεί από την κλάση MaxQ, η οποία αντιπροσωπεύει την ιεραρχική αναπαράσταση του προβλήματος μας. Επικοινωνεί με την κλάση SA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Επίσης επικοινωνεί με τις κλάσεις MaxNodeWoLFPHC και Qnode αφού χρησιμοποιεί τους MaxNodeWoLFPHC και Qnode του γράφου για την επίλυση του προβλήματος.

Το Σχήμα 3.9 παρουσιάζει τη διασύνδεση των κλάσεων αυτού του επιμέρους συστήματος, δείχνοντας πιο παραστατικά όλα όσα εξηγήθηκαν πιο πάνω. Από το σχήμα αυτό παρατηρούμε ότι το σύστημα μπορεί να επεκταθεί εύκολα.



Σχήμα 3.9: Διασύνδεση Κλάσεων Συστήματος ΕΜΕΠ

Το βέλος ⟶ σημαίνει ότι η κλάση (ή αντικείμενά της) που βρίσκονται στα αριστερά του βέλους, είτε χρησιμοποιεί στιγμιότυπα αντικειμένων της κλάσης που βρίσκεται στα δεξιά του βέλους, είτε καλεί στατικές μεθόδους της. Το βέλος ⟶▷ σημαίνει ότι η κλάση που βρίσκεται αριστερά απ'αυτό κληρονομεί χαρακτηριστικά από την κλάση που βρίσκεται στα δεξιά του.

3.3 Ενισχυτική Μάθηση Πολλαπλών Πρακτόρων

3.3.1 Αλγόριθμοι που υλοποιήθηκαν

3.3.1.1 Αλγόριθμος Q-Learning για Πολλαπλούς Πράκτορες

Στον αλγόριθμο Q-Learning αρχικά έχουμε την φάση αρχικοποιήσεων στην οποία αρχικοποιούνται οι ενέργειες, τα ενισχυτικά σήματα και η κατάσταση του περιβάλλοντος με κάποιες αρχικές τιμές. Στην συνέχεια, για κάθε επεισόδιο έχουμε την εξεύρεση της αρχικής κατάστασης. Ακολούθως, για κάθε βήμα του επεισοδίου, κάθε πράκτορας επιλέγει μια ενέργεια με βάση την κατάσταση του περιβάλλοντος και την πολιτική που ακολουθεί. Στη συνέχεια, το περιβάλλον λαμβάνει τις ενέργειες των πρακτόρων, διαμορφώνει την κατάσταση του και την επιστρέφει στους πράκτορες, δίνοντας τους και το αντίστοιχο ενισχυτικό σήμα για τον κάθε πράκτορα, που αντιστοιχεί στην αμοιβή. Τέλος, ο κάθε πράκτορας ενημερώνει την Συνάρτηση ενέργειας-αξίας του.

Ακολουθεί στο Σχήμα 3.10 ο ψευδοκώδικας του αλγόριθμου αυτού:

Αρχικοποίηση των τιμών της συνάρτησης ενέργειας-αξίας Q(s,a) αυθαίρετα

Επανέλαβε (για κάθε επεισόδιο):

Αρχικοποίησε την αρχική κατάσταση s

Επανέλαβε (για κάθε βήμα του επεισοδίου):

Επανέλαβε (για κάθε πράκτορα):

Επέλεξε μια ενέργεια α από την κατάσταση s χρησιμοποιώντας μια πολιτική η οποία απορρέει από την Q (π.χ ε-greedy)

Επανέλαβε (για κάθε πράκτορα):

Εκτέλεσε την ενέργεια α

Επανέλαβε (για κάθε πράκτορα):

Παρατήρησε την αμοιβή r

Παρατήρησε την επόμενη κατάσταση s'

Επανέλαβε (για κάθε πράκτορα):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

s←s';

Μέχρι να κάνεις x επιτυχημένες μεταφορές

Σχήμα 3.10: Αλγόριθμος Q-Learning για Πολλαπλούς Πράκτορες

Σύμφωνα με τον αλγόριθμο διατηρούμε τιμές *Q* τις οποίες αρχικοποιούμε αυθαίρετα. Ακολούθως επαναλαμβάνεται η εξής διαδικασία: Από την κατάσταση *s* επιλέγουμε για κάθε πράκτορα την ενέργεια *α* σύμφωνα με κάποια πολιτική. Στη συνέχεια, αφού εκτελεστεί η ενέργεια του κάθε πράκτορα, παίρνουμε την αμοιβή για τον κάθε πράκτορα. Εφόσον πάρουμε την αμοιβή, παρατηρούμε την επόμενη κατάσταση ανανεώνουμε την *Q* τιμή, για κάθε πράκτορα, της κατάστασης *s* με ενέργεια *α* σύμφωνα με τον τύπο στο σχήμα. Ο αλγόριθμος τελειώνει όταν γίνουν *x* επιτυχημένες μεταφορές, όπου το *x* καθορίζεται από τον χρήστη.

3.3.1.2 Αλγόριθμος PHC για Πολλαπλούς Πράκτορες

Ο αλγόριθμος PHC για Πολλαπλούς Πράκτορες έχει υλοποιηθεί με το ίδιο σκεπτικό που υλοποιήθηκε και ο αλγόριθμος PHC Ενός Πράκτορα και έχει εξηγηθεί στο κεφάλαιο 3.2.1.3 και στο Σχήμα 3.3.

3.3.1.3 Αλγόριθμος WoLF-PHC για Πολλαπλούς Πράκτορες

Ο αλγόριθμος WoLF-PHC για Πολλαπλούς Πράκτορες έχει υλοποιηθεί με το ίδιο σκεπτικό που υλοποιήθηκε και ο αλγόριθμος WoLF-PHC Ενός Πράκτορα και έχει εξηγηθεί στο κεφάλαιο 3.2.1.4 και στο Σχήμα 3.4.

3.3.1.4 Αλγόριθμος MAXQ-Q για Πολλαπλούς Πράκτορες

Ο αλγόριθμος MAXQ-Q ο οποίος έχει υλοποιηθεί στο επιμέρους σύστημα ενός πράκτορα είναι ένας αναδρομικός αλγόριθμος. Για να μπορέσει να επεκταθεί, ούτως ώστε να μπορεί να υποστηρίζει περισσότερους από ένα πράκτορες, χρειαζόταν να μετατραπεί από αναδρομικός σε επαναληπτικός. Σε αυτό τον αλγόριθμο ο κάθε πράκτορας έχει τον δικό του γράφο, ο οποίος αντιστοιχεί στην Ιεραρχική αναπαράσταση του προβλήματος, και έτσι ο κάθε πράκτορας έχει την δική του Συνάρτηση Αξίας και τις δικές του Συναρτήσεις Ολοκλήρωσης. Αρχικά γίνεται επιλογή ενεργειών από ένα πράκτορα, μέχρι να καταλήξει σε αρχέγονο κόμβο στον γράφο. Τότε συνεχίζει ο επόμενος πράκτορας με την ίδια διαδικασία. Η διαδικασία αυτή συνεχίζεται μέχρι να καταλήξουν όλοι οι πράκτορες σε αρχέγονο κόμβο. Τότε το περιβάλλον λαμβάνει τις ενέργειες των πρακτόρων, διαμορφώνει την κατάσταση του και την επιστρέφει στους πράκτορες, δίνοντας τους και το αντίστοιχο ενισχυτικό σήμα για τον κάθε πράκτορα, που αντιστοιχεί στην αμοιβή. Τέλος, ο κάθε πράκτορας ενημερώνει την Συνάρτηση Αξίας του.

Ακολουθεί στο Σχήμα 3.11 ο ψευδοκώδικας του αλγόριθμου αυτού.

```
1  for each agent do
2       push MaxRoot into agent stack
3       let seq = () be the sequence of states visited while executing i
4       push seq onto agent seqStack
5  do
6       for each agent do
7            maxNode= get the top of agent stack
8            while  i is not  a primitive MaxNode
9                 let seq = () be the sequence of states visited while executing i
10                push seq onto agent seqStack
11                if  Ti(s) is false
12                     choose an action a according to the current exploration
                       policy πx(i, s)
13                     push action onto agent stack
                  else
14                     pop(agent stack)
15                     maxNode= get the top of agent stack
16                     let a* = argmax_a' [C̃_t(i, s', a') + V_t(a', s')]
17                     for each s in seqStack do
18                       C̃_t+1(i, s, a) := (1 − α_t(i)) · C̃_t(i, s, a) + α_t(i) · γ^N [R̃_i(s') + C̃_t(i, s', a*) + V_t(a*, s)]
19                       C_t+1(i, s, a) := (1 − α_t(i)) · C_t(i, s, a) + α_t(i) · γ^N [C_t(i, s', a*) + V_t(a*, s')]
20                        N := N + 1
21                     end //for
22                     s := s'
                  end //else
23                maxNode = get the top of agent stack
24           end //while
25      end //for
26      for each agent do
27           execute i
28      for each agent do
29           receive r
30       observe result state s'
31      for each agent do
32           V_t+1(i, s) := (1 − α_t(i)) · V_t(i, s) + α_t(i) · r_t
33           push s onto the beginning of the top seq in agent seqStack
34           pop(agent stack)
35           maxNode= get the top of agent stack
36           let a* = argmax_a' [C̃_t(i, s', a') + V_t(a', s')]
37           for each s in seqStack do
38             C̃_t+1(i, s, a) := (1 − α_t(i)) · C̃_t(i, s, a) + α_t(i) · γ^N [R̃_i(s') + C̃_t(i, s', a*) + V_t(a*, s)]
39             C_t+1(i, s, a) := (1 − α_t(i)) · C_t(i, s, a) + α_t(i) · γ^N [C_t(i, s', a*) + V_t(a*, s')]
40             N := N + 1
41           end //for
42      end/ /for
43      s := s'
44      until you make x successful delivery
```

Σχήμα 3.11: Αλγόριθμος MAXQ-Q για Πολλαπλούς Πράκτορες

Σε αυτό τον αλγόριθμο χρησιμοποιούνται δύο στοίβες για κάθε πράκτορα. Η στοίβα *stack* χρησιμοποιείται για να κρατά το μονοπάτι με τους κόμβους τους οποίους έχει επισκεφθεί ο πράκτορας, ούτως ώστε να είναι δυνατή η διέλευση του μέσα στον γράφο. Η στοίβα *seqStack* αποθηκεύει την ακολουθία με τις καταστάσεις οι οποίες έχουν επισκεφτεί κατά την εκτέλεση του Max κόμβου *i*. Κατά την εκτέλεση του αλγόριθμου (από ένα πράκτορα), αν ο Max κόμβος δεν είναι αρχέγονος και η κατάσταση που βρισκόμαστε δεν είναι τερματική, τότε γίνεται επιλογή ενέργειας σύμφωνα με μια πολιτική. Αν η κατάσταση που βρισκόμαστε είναι τερματική, γίνεται εύρεση της άπληστης ενέργειας α^* για την κατάσταση s' σύμφωνα με τον τύπο στο σχήμα. Στη συνέχεια, για κάθε κατάσταση s που βρίσκεται στη στοίβα *seqStack* γίνεται ανανέωση των δύο συναρτήσεων ολοκλήρωσης (completion function) C και $\hat{C}$ σύμφωνα με τους τύπους στο σχήμα, όπου $\hat{R}i(s')$ είναι ψευδοαμοιβή για την κατάσταση s'. Αν ο Max κόμβος είναι αρχέγονος τότε σταματά η εκτέλεση για τον συγκεκριμένο πράκτορα και ξεκινά την εκτέλεση της πιο πάνω διαδικασίας ο επόμενος πράκτορας. Όταν εκτελεστεί αυτή η διαδικασία από όλους τους πράκτορες, το περιβάλλον λαμβάνει τις ενέργειες των πρακτόρων, διαμορφώνει την κατάσταση του και την επιστρέφει στους πράκτορες, δίνοντας τους και το αντίστοιχο ενισχυτικό σήμα για τον κάθε πράκτορα, που αντιστοιχεί στην αμοιβή. Τέλος, ο κάθε πράκτορας ενημερώνει την Συνάρτηση Αξίας του. Ο αλγόριθμος τελειώνει όταν γίνουν x επιτυχημένες μεταφορές, όπου το x καθορίζεται από τον χρήστη.

3.3.1.5 Αλγόριθμος MAXQ-PHC για Πολλαπλούς Πράκτορες

Ο αλγόριθμος MAXQ-PHC, για Πολλαπλούς Πράκτορες, είναι μια Ιεραρχική υλοποίηση του αλγόριθμου PHC χρησιμοποιώντας την MAXQ Ιεραρχική διάσπαση. Είναι παρόμοιος με τον αλγόριθμο MAXQ-Q, για Πολλαπλούς Πράκτορες, με την διαφορά ότι η επιλογή της κάθε ενέργειας γίνεται με βάση κάποιων πιθανοτήτων.

Ακολουθεί στο Σχήμα 3.12 ο ψευδοκώδικας του αλγόριθμου αυτού.

---

1 for each agent do
2 push MaxRoot into agent stack
3 let seq = () be the sequence of states visited while executing i
4 push seq onto agent seqStack
5 do
6 for each agent do
7 maxNode= get the top of agent stack
8 while i is not a primitive MaxNode
9 let seq = () be the sequence of states visited while executing i
10 push seq onto agent seqStack
11 if Ti(s) is false
12 from state s select action a, with probability π(s,a) with some exploration
13 update π(s,a) and constrain it to a legal probability distribution,
14 $$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}$$
15 push action onto agent stack
else
16 pop(agent stack)
17 maxNode= get the top of agent stack
18 $\textbf{let } a^* = \text{argmax}_{a'} [\tilde{C}_t(i,s',a') + V_t(a',s')]$
19 for each s in childSeq do
20 $\tilde{C}_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot \tilde{C}_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s)]$
21 $C_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [C_t(i,s',a^*) + V_t(a^*,s')]$
22 N := N + 1
23 end //for
24 s := s'
end //else
25 maxNode = get the top of agent stack
26 end //while
27 end //for
28 for each agent do
29 execute i
30 for each agent do
31 receive r
32 observe result state s'
33 for each agent do
34 $V_{t+1}(i,s) := (1-\alpha_t(i)) \cdot V_t(i,s) + \alpha_t(i) \cdot r_t$
35 push s onto the beginning of the top seq in agent seqStack
36 pop(agent stack)
37 maxNode= get the top of agent stack
38 $\textbf{let } a^* = \text{argmax}_{a'} [\tilde{C}_t(i,s',a') + V_t(a',s')]$
39 for each s in childSeq do
40 $\tilde{C}_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot \tilde{C}_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s)]$
41 $C_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [C_t(i,s',a^*) + V_t(a^*,s')]$
42 N := N + 1
43 end //for
44 end/ /for
45 s := s'
46 until you make x successful delivery

Σχήμα 3.12: Αλγόριθμος MAXQ-PHC για Πολλαπλούς Πράκτορες

Σύμφωνα με τον αλγόριθμο διατηρούμε πιθανότητες για κάθε ζεύγος κατάστασης-ενέργειας τις οποίες αρχικοποιούμε με 1/Πλήθος ενεργειών. Ο αλγόριθμος εξηγήθηκε στο σχήμα 3.11. Η διαφορά είναι ότι στον αλγόριθμο MAXQ-PHC η επιλογή ενέργειας γίνεται με βάση κάποιων πιθανοτήτων (γραμμή 12), και μετά από κάθε επιλογή ενέργειας γίνεται ανανέωση της πιθανότητας *π(s,α)*, όπως δείχνει ο αλγόριθμος στο σχήμα (γραμμή 13 και 14), όπου *δ* είναι ο ρυθμός μάθησης.

3.3.1.6 Αλγόριθμος MAXQ-WoLF-PHC για Πολλαπλούς Πράκτορες

Ο αλγόριθμος MAXQ-WoLF-PHC είναι μια επέκταση του αλγόριθμου MAXQ-PHC και μια Ιεραρχική υλοποίηση του αλγόριθμου WoLF-PHC χρησιμοποιώντας την MAXQ Ιεραρχική διάσπαση. Συγκεκριμένα ο αλγόριθμος απαιτεί 2 ρυθμούς μάθησης $\delta l > \delta w$. Όταν ο παίχτης νικά, χρησιμοποιείται το δw και στην αντίθετη περίπτωση το δl. Αυτό εντοπίζεται συγκρίνοντας την αναμενόμενη τιμή, χρησιμοποιώντας τις τρέχουσες *Q* τιμές, ακολουθώντας την τρέχουσα πολιτική *π*, στην τρέχουσα κατάσταση με αυτό, ακολουθώντας την μέση πολιτική $\bar{\pi}$. Αν η αναμενόμενη τιμή από την τρέχουσα πολιτική είναι μικρότερη (ο παίχτης χάνει), ο μεγαλύτερος ρυθμός μάθησης χρησιμοποιείται.

Ακολουθεί στο Σχήμα 3.13 ο ψευδοκώδικας του αλγόριθμου αυτού.

---

1 for each agent do
2 push MaxRoot into agent stack
3 let seq = () be the sequence of states visited while executing i
4 push seq onto agent seqStack
5 do
6 for each agent do
7 maxNode= get the top of agent stack
8 while i is not a primitive MaxNode
9 let seq = () be the sequence of states visited while executing i
10 push seq onto agent seqStack
11 if Ti(s) is false
12 from state s select action a, with probability π(s,a) with some exploration
13 update estimate of average policy,
14 $C(s) \leftarrow C(s)+1$

$\forall\, a' \in Ai \quad \bar{\pi}(s,a') \leftarrow \bar{\pi}(s,a') + \frac{1}{C(s)} (\pi(s,a') - \bar{\pi}(s,a'))$
15 update π(s,a) and constrain it to a legal probability distribution,
16 $$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}$$

where

$$\delta = \begin{cases} \delta_w & \text{if } \sum_a \pi(s,a)Q(s,a) > \sum_a \bar{\pi}(s,a)Q(s,a) \\ \delta_l & \text{otherwise} \end{cases}$$
17 push action onto agent stack
18 else
19 pop(agent stack)
20 maxNode= get the top of agent stack
21 $\textbf{let } a^* = \text{argmax}_{a'} [\tilde{C}_t(i,s',a') + V_t(a',s')]$
22 for each s in childSeq do
23 $\tilde{C}_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot \tilde{C}_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s)]$
24 $C_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [C_t(i,s',a^*) + V_t(a^*,s')]$
N := N + 1
25 end //for
26 s := s'
27 end //else
maxNode = get the top of agent stack
28 end //while
29 end //for
30 for each agent do
31 execute i
32 for each agent do
33 receive r
34 observe result state s'
35 for each agent do
36 $V_{t+1}(i,s) := (1-\alpha_t(i)) \cdot V_t(i,s) + \alpha_t(i) \cdot r_t$
37 push s onto the beginning of the top seq in agent seqStack
38 pop(agent stack)
39 maxNode= get the top of agent stack
40 $\textbf{let } a^* = \text{argmax}_{a'} [\tilde{C}_t(i,s',a') + V_t(a',s')]$
41 for each s in childSeq do
42 $\tilde{C}_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot \tilde{C}_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s)]$
43 $C_{t+1}(i,s,a) := (1-\alpha_t(i)) \cdot C_t(i,s,a) + \alpha_t(i) \cdot \gamma^N [C_t(i,s',a^*) + V_t(a^*,s')]$
44 N := N + 1
45 end //for
46 end/ /for
47 s := s'
48 until you make x successful delivery

Σχήμα 3.13: Αλγόριθμος MAXQ- WoLF –PHC για Πολλαπλούς Πράκτορες

Σύμφωνα με τον αλγόριθμο διατηρούμε πιθανότητες και τιμή μέσης πιθανότητας για κάθε ζεύγος κατάστασης-ενέργειας τις οποίες αρχικοποιούμε με 1/Πλήθος ενεργειών. Επίσης διατηρούμε και ένα μετρητή για κάθε κατάσταση ο οποίος αρχικοποιείται με 0 και αυξάνεται κατά 1, κάθε φορά που επισκεπτόμαστε την κατάσταση αυτή. Ο αλγόριθμος εξηγήθηκε στο σχήμα 3.11. Η διαφορά είναι ότι στον αλγόριθμο MAXQ-WoLF-PHC η επιλογή ενέργειας γίνεται με βάση κάποιων πιθανοτήτων (γραμμή 12), και μετά από κάθε επιλογή ενέργειας γίνεται ανανέωση της μέσης πιθανότητας $\bar{\pi}$ και της πιθανότητας *π(s,α)*, όπως δείχνει ο αλγόριθμος στο σχήμα (γραμμή 13 - 16).

3.3.2 Συστατικά Συστήματος

Το σύστημα αποτελείται από τα εξής κύρια συστατικά:

1. Simulation
2. MA_Environment,
3. MA_Gridworld,
4. Graph,
5. MA_MaxQ,
4. MAQ_Learning,
5. MA_Phc,
6. MA_Wolf,
7. MA_MAXQ-Q,
8. MA_MAXQ_PHC,
9. MA_MAXQ_WoLF_PHC,
10. Agent,
11. Q_Controller,
12. PHC_ Controller,
13. WP_ Controller,
14. MAXQ_ Controller

Η κλάση Simulation είναι το κύριο συστατικό του συστήματος μας. Υλοποιήθηκε σαν Αφαιρετική Κλάση (Abstract Class), για να αναγκάζει την επέκταση της και έτσι επιτρέπει τη χρήση πολυμορφισμού. Μια κλάση που επεκτείνει την Simulation, μπορεί

είτε να αφορά προσομοίωση αλγορίθμων ενός πράκτορα, είτε προσομοίωση αλγορίθμων πολλαπλών πρακτόρων. Στο σύστημα υλοποιήσαμε την κλάση MA_Simulation (Multi Agent Simulation) η οποία κληρονομεί από την κλάση Simulation. Η MA_Simulation αλληλεπιδρά με τον χρήστη δίνοντας του την ευκαιρία να επιλέξει τον αλγόριθμο που θέλει να προσομοιώσει. Συλλέγει της πληροφορίες και τις στέλνει σε ένα αντικείμενο της κλάσης MA_RunTrial (Multi Agent RunTrial), το οποίο είναι υπεύθυνο για να κάνει αυτή την προσομοίωση. Επίσης επικοινωνεί με ένα αντικείμενο της κλάσης MA_PlotGraph το οποίο είναι υπεύθυνο για την δημιουργία των γραφικών παραστάσεων.

Η κλάση MA_Environment είναι το κύριο συστατικό για το περιβάλλον που θα χρησιμοποιηθεί. Υλοποιήθηκε σαν Αφαιρετική Κλάση, για να αναγκάζει την επέκταση της από οποιοδήποτε περιβάλλον. Επικοινωνεί με την κλάση MA_State_Variables (Multi Agent State_Variables), αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει τη νέα κατάσταση. Στο σύστημα που υλοποιήσαμε δημιουργήσαμε μια κλάση MA_Taxi_Problem (Multi Agent Taxi_Problem) η οποία κληρονομεί από την κλάση Environment και επικοινωνεί με την κλάση MA_Gridworld (Multi Agent Gridworld), αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ο επιβάτης.

Η κλάση MA_Gridworld (Multi Agent Gridworld) αντιπροσωπεύει το πλαίσιο του προβλήματος, αφού διαβάζοντας ένα αντιπροσωπευτικό αρχείο το δημιουργεί. Επικοινωνεί με την κλάση Wall, αφού καθορίζει που υπάρχουν εμπόδια στο πλαίσιο. Επίσης σε αυτή την κλάση υπάρχουν οι επιτρεπτές τοποθεσίες για τον επιβάτη και οι καταστάσεις του προβλήματος.

Η κλάση Graph είναι το κύριο συστατικό του συστήματος που αντιπροσωπεύει την ιεραρχική αναπαράσταση του προβλήματος μας. Έχει υλοποιηθεί σαν Αφαιρετική Κλάση για να αναγκάζει την επέκταση της και έτσι επιτρέπει τη χρήση πολυμορφισμού. Μια κλάση που επεκτείνει την Graph μπορεί να αναπαραστήσει ιεραρχικά οποιοδήποτε

άλλο πρόβλημα. Σε αυτό το σύστημα έχει υλοποιηθεί η κλάση MA_Graph (Multi Agent Graph), η οποία υλοποιεί τον γράφο για το TP για πολλαπλούς πράκτορα. Επικοινωνεί με τις κλάσεις Node, MaxNode, MaxNodePHC, MaxNodeWoLFPHC και Qnode, αφού χρειάζεται να δημιουργήσει ένα γράφο με Max και Q κόμβους. Η κλάση MaxNode κληρονομεί από την κλάση Node, η κλάση MaxNodePHC κληρονομεί από την κλάση MaxNode και η κλάση MaxNodeWoLFPHC κληρονομεί από την κλάση MaxNodePHC. Θα εξηγηθεί στην συνέχεια που χρησιμοποιούνται αυτές οι κλάσεις. Επίσης επικοινωνεί με την κλάση MA_Gridworld, αφού χρειάζεται να γνωρίζει τις καταστάσεις του προβλήματος.

Η κλάση MA_MaxQ (Multi Agent MaxQ) είναι το κύριο συστατικό για το περιβάλλον που θα χρησιμοποιηθεί για τους Ιεραρχικούς αλγόριθμους ΕΜΠΠ. Κληρονομεί από την κλάση MA_Graph αφού χρειάζεται να ξέρει την ιεραρχική αναπαράσταση του προβλήματος. Επίσης επικοινωνεί με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει τη νέα κατάσταση. Ακόμη επικοινωνεί με την κλάση MA_Gridworld, αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ο επιβάτης.

Η κλάση MAQ_Learning (Multi Agent Q_Learning) είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος Q-Learning για πολλαπλούς πράκτορες. Κληρονομεί από την κλάση MA_Taxi_Problem, αφού το πρόβλημα που είχαμε να υλοποιήσουμε αφορούσε το Taxi Problem. Επικοινωνεί με την κλάση MA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Αφού ο αλγόριθμος αφορά πολλαπλούς πράκτορες, η κλάση MAQ_Learning διατηρεί αντικείμενα της κλάσης Q_Agent, η οποία αντιπροσωπεύει τον Q πράκτορα. Επίσης επικοινωνεί με την κλάση Q_Controller η οποία είναι υπεύθυνη για τις ενέργειες αυτού του πράκτορα. Έτσι είναι σε θέση ανά πάσα στιγμή να γνωρίζει οτιδήποτε έχει σχέση με τον κάθε πράκτορα.

Η κλάση MA_ Phc (Multi Agent Phc) είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος PHC (Policy Hill-Climbing) για πολλαπλούς πράκτορες. Κληρονομεί από την κλάση MA_Taxi_Problem, αφού το πρόβλημα που είχαμε να υλοποιήσουμε αφορούσε το Taxi Problem. Επικοινωνεί με την κλάση MA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Αφού και σε αυτό τον αλγόριθμο υπάρχουν πολλαπλοί πράκτορες η κλάση MA_ Phc διατηρεί αντικείμενα της κλάσης PHC_Agent, η οποία αντιπροσωπεύει τον PHC πράκτορα. Επίσης επικοινωνεί με την κλάση PHC_Controller η οποία είναι υπεύθυνη για τις ενέργειες αυτού του πράκτορα. Έτσι είναι σε θέση ανά πάσα στιγμή να γνωρίζει οτιδήποτε έχει σχέση με τον κάθε πράκτορα.

Η κλάση MA_Wolf (Multi Agent Wolf) είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος WoLF-PHC (WoLF Policy Hill-Climbing) για πολλαπλούς πράκτορες. Κληρονομεί από την κλάση MA_Taxi_Problem, αφού το πρόβλημα που είχαμε να υλοποιήσουμε αφορούσε το Taxi Problem. Επικοινωνεί με την κλάση MA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Αφού και σε αυτό τον αλγόριθμο υπάρχουν πολλαπλοί πράκτορες η κλάση MA_ Wolf διατηρεί αντικείμενα της κλάσης WP_Agent, η οποία αντιπροσωπεύει τον WoLF-PHC πράκτορα. Επίσης επικοινωνεί με την κλάση WP_Controller η οποία είναι υπεύθυνη για τις ενέργειες αυτού του πράκτορα. Έτσι είναι σε θέση ανά πάσα στιγμή να γνωρίζει οτιδήποτε έχει σχέση με τον κάθε πράκτορα.

Η κλαση MA_MaxQ_Q είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος MaxQ-Q για πολλαπλούς πράκτορες. Κληρονομεί από την κλάση MA_MaxQ, η οποία αντιπροσωπεύει την ιεραρχική αναπαράσταση του προβλήματος μας. Επικοινωνεί με την κλάση MA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Επίσης επικοινωνεί με τις κλάσεις MaxNode και Qnode αφού χρησιμοποιεί τους MaxNode και Qnode του γράφου για την επίλυση του

προβλήματος. Αφού και σε αυτό τον αλγόριθμο υπάρχουν πολλαπλοί πράκτορες η κλάση MA_ MaxQ_Q διατηρεί αντικείμενα της κλάσης MaxQ_Agent, η οποία αντιπροσωπεύει τον MaxQ πράκτορα. Επίσης επικοινωνεί με την κλάση MaxQ_Controller η οποία είναι υπεύθυνη για τις ενέργειες αυτού του πράκτορα. Έτσι είναι σε θέση ανά πάσα στιγμή να γνωρίζει οτιδήποτε έχει σχέση με τον κάθε πράκτορα.

Η κλαση MA_MaxQ_PHC είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος MaxQ-PHC για πολλαπλούς πράκτορες. Κληρονομεί από την κλάση MA_MaxQ, η οποία αντιπροσωπεύει την ιεραρχική αναπαράσταση του προβλήματος μας. Επικοινωνεί με την κλάση MA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Επίσης επικοινωνεί με τις κλάσεις MaxNodePHC και Qnode αφού χρησιμοποιεί τους MaxNodePHC και Qnode του γράφου για την επίλυση του προβλήματος. Αφού και σε αυτό τον αλγόριθμο υπάρχουν πολλαπλοί πράκτορες η κλάση MA_MaxQ_ PHC διατηρεί αντικείμενα της κλάσης MaxQ_Agent, η οποία αντιπροσωπεύει τον MaxQ πράκτορα. Επίσης επικοινωνεί με την κλάση MaxQ_Controller η οποία είναι υπεύθυνη για τις ενέργειες αυτού του πράκτορα. Έτσι είναι σε θέση ανά πάσα στιγμή να γνωρίζει οτιδήποτε έχει σχέση με τον κάθε πράκτορα.

Η κλαση MA_MaxQ_WoLF_PHC είναι η κλάση στην οποία υλοποιήθηκε ο αλγόριθμος MaxQ-WoLF-PHC. Κληρονομεί από την κλάση MA_MaxQ, η οποία αντιπροσωπεύει την ιεραρχική αναπαράσταση του προβλήματος μας. Επικοινωνεί με την κλάση MA_Gridworld για να δημιουργήσει το πλαίσιο του προβλήματος και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος. Επίσης επικοινωνεί με τις κλάσεις MaxNodeWoLFPHC και Qnode αφού χρησιμοποιεί τους MaxNodeWoLFPHC και Qnode του γράφου για την επίλυση του προβλήματος. Αφού και σε αυτό τον αλγόριθμο υπάρχουν πολλαπλοί πράκτορες η κλάση MA_MaxQ_ WoLF_PHC διατηρεί αντικείμενα της κλάσης MaxQ_Agent, η οποία αντιπροσωπεύει τον MaxQ πράκτορα. Επίσης επικοινωνεί με την κλάση MaxQ_Controller η οποία είναι υπεύθυνη για τις

ενέργειες αυτού του πράκτορα. Έτσι είναι σε θέση ανά πάσα στιγμή να γνωρίζει οτιδήποτε έχει σχέση με τον κάθε πράκτορα.

Η κλάση Agent, είναι και αυτή Αφαιρετική, αφού από μόνη της δεν σημαίνει τίποτα και έτσι αναγκάζει τον προγραμματιστή να την επεκτείνει υλοποιώντας τους δικούς του πράκτορες. Οι πράκτορες που υλοποιήθηκαν περιγράφονται από τις κλάσεις Q_Agent, PHC_Agent,WP_Agent (WoLF_PHC Agent) και MaxQ_Agent ο οποίος αντιπροσωπεύει τους πράκτορες MAXQ-Q, MAXQ-PHC και MAXQ-WoLF-PHC.

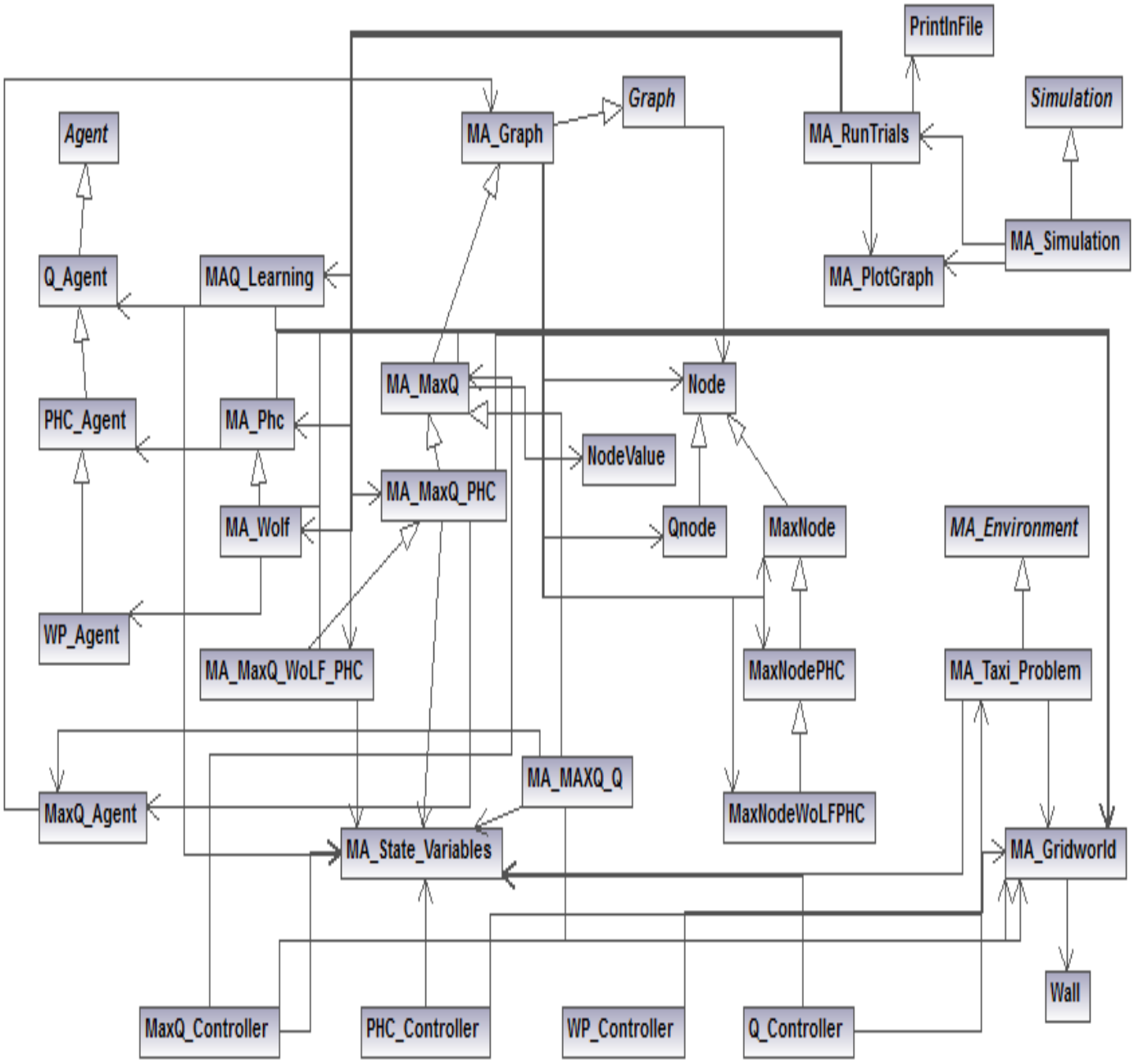
Η κλάση Q_Controller, είναι η κλάση όπου είναι υπεύθυνη για την επιλογή ενέργειας από τους Q πράκτορες, για την εκτέλεση των ενεργειών τους, την παρατήρηση της αμοιβής και της επόμενης κατάστασης. Έτσι επικοινωνεί με την κλάση Q_Agent, αφού χρειάζεται πληροφορίες για τον Q πράκτορα και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει τη νέα κατάσταση. Επίσης επικοινωνεί με την κλάση MA_Taxi_Problem, αφού αυτή η κλάση αντιπροσωπεύει το περιβάλλον και είναι εδώ που γίνεται η επιλογή ενέργειας από κάθε Q πράκτορα. Ακόμη επικοινωνεί με την κλάση MA_Gridworld, αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ένας επιβάτης.

Η κλάση PHC_Controller, είναι η κλάση όπου είναι υπεύθυνη για την επιλογή ενέργειας από τους PHC πράκτορες, για την εκτέλεση των ενεργειών τους, την παρατήρηση της αμοιβής και της επόμενης κατάστασης. Έτσι επικοινωνεί με την κλάση PHC_Agent, αφού χρειάζεται πληροφορίες για τον PHC πράκτορα και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει τη νέα κατάσταση. Επίσης επικοινωνεί με την κλάση MA_Taxi_Problem, αφού αυτή η κλάση αντιπροσωπεύει το περιβάλλον. Ακόμη επικοινωνεί με την κλάση MA_Gridworld, αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ένας επιβάτης.

Η κλάση WP_Controller (WoLF-PHC Controller), είναι η κλάση όπου είναι υπεύθυνη για την επιλογή ενέργειας από τους WoLF-PHC πράκτορες, για την εκτέλεση των ενεργειών τους, την παρατήρηση της αμοιβής και της επόμενης κατάστασης. Έτσι επικοινωνεί με την κλάση WP_Agent, αφού χρειάζεται πληροφορίες για τον WoLF-PHC πράκτορα και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει τη νέα κατάσταση. Επίσης επικοινωνεί με την κλάση MA_Taxi_Problem, αφού αυτή η κλάση αντιπροσωπεύει το περιβάλλον. Ακόμη επικοινωνεί με την κλάση MA_Gridworld, αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ένας επιβάτης.

Η κλάση MaxQ_Controller, είναι η κλάση όπου είναι υπεύθυνη για την εκτέλεση των ενεργειών, την παρατήρηση της αμοιβής και της επόμενης κατάστασης για τους MaxQ πράκτορες. Έτσι επικοινωνεί με την κλάση MaxQ_Agent, αφού χρειάζεται πληροφορίες για τον MaxQ πράκτορα και με την κλάση MA_State_Variables, αφού χρειάζεται ένα τρόπο κωδικοποίησης των καταστάσεων του περιβάλλοντος, για να μπορεί να ερμηνεύσει κάποια ενέργεια και να διαμορφώσει τη νέα κατάσταση. Επίσης επικοινωνεί με την κλάση MA_Gridworld, αφού χρειάζεται να γνωρίζει που υπάρχουν εμπόδια στο πλαίσιο, τα όρια του πλαισίου και τις επιτρεπτές τοποθεσίες που μπορεί να βρίσκεται ή να θέλει να μεταφερθεί ένας επιβάτης.

Το Σχήμα 3.14 παρουσιάζει τη διασύνδεση των κλάσεων αυτού του επιμέρους συστήματος, δείχνοντας πιο παραστατικά όλα όσα εξηγήθηκαν πιο πάνω. Από το σχήμα αυτό παρατηρούμε ότι το σύστημα μπορεί να επεκταθεί εύκολα.



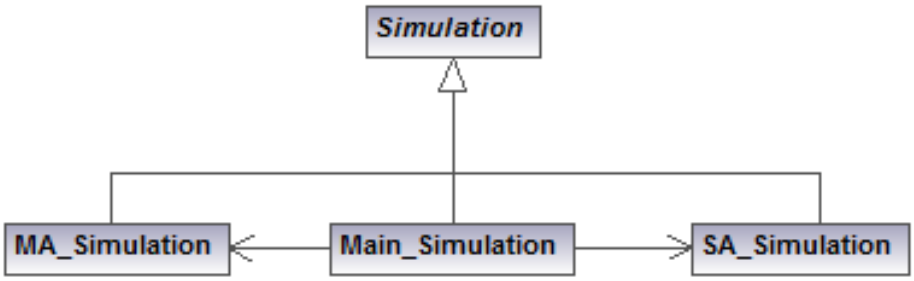
Σχήμα 3.14: Διασύνδεση Κλάσεων Συστήματος ΕΜΠΠ

Το βέλος ⟶ σημαίνει ότι η κλάση (ή αντικείμενά της) που βρίσκονται στα αριστερά του βέλους, είτε χρησιμοποιεί στιγμιότυπα αντικειμένων της κλάσης που βρίσκεται στα δεξιά του βέλους, είτε καλεί στατικές μεθόδους της. Το βέλος ⟶▷ σημαίνει ότι η κλάση που βρίσκεται αριστερά απ'αυτό κληρονομεί χαρακτηριστικά από την κλάση που βρίσκεται στα δεξιά του.

3.4 Γενικό Σύστημα

Σε αυτό το σύστημα γίνεται η σύνδεση των δύο συστημάτων που έχουν προαναφερθεί (Σύστημα ΕΜΕΠ και Σύστημα ΕΜΠΠ). Αποτελείται από το κύριο συστατικό Main_Simulation το οποίο κληρονομεί από την αφαιρετική κλάση Simulation. Η κλάση Main_Simulation αλληλεπιδρά με τον χρήστη δίνοντας του την επιλογή να επιλέξει αν θέλει να προσομοιώσει αλγόριθμους ΕΜΕΠ ή ΕΜΠΠ. Ανάλογα με την επιλογή του χρήστη, επικοινωνεί με τις κλάσεις SA_Simulation και MA_Simulation αντίστοιχα.

Το Σχήμα 3.15 παρουσιάζει τη διασύνδεση των κλάσεων του συστήματος αυτού, δείχνοντας πιο παραστατικά όλα όσα εξηγήθηκαν πιο πάνω.



Σχήμα 3.15: Διασύνδεση Κλάσεων Γενικού Συστήματος

Το βέλος ⟶ σημαίνει ότι η κλάση (ή αντικείμενά της) που βρίσκονται στα αριστερά του βέλους, είτε χρησιμοποιεί στιγμιότυπα αντικειμένων της κλάσης που βρίσκεται στα δεξιά του βέλους, είτε καλεί στατικές μεθόδους της. Το βέλος ⟶▷ σημαίνει ότι η κλάση που βρίσκεται αριστερά απ'αυτό κληρονομεί χαρακτηριστικά από την κλάση που βρίσκεται στα δεξιά του.

Κεφάλαιο 4

Αποτελέσματα και Συζήτηση

4.1 Εισαγωγή

Εφαρμόζοντας τους αλγόριθμους ΕΜΕΠ που υλοποιήθηκαν, στο απλό TP και στο ETP, καθώς και τους αλγόριθμους ΕΜΠΠ στο MATP πήραμε κάποια αποτελέσματα που μας δείχνουν πόσο καλά εκπαιδεύεται ο πράκτορας, αλλά και πόσο χρόνο χρειάζεται για να εκπαιδευτεί, χρησιμοποιώντας τον κάθε αλγόριθμο. Έτσι θέτουμε σαν κριτήριο για το πόσο αποδοτικός είναι ο κάθε αλγόριθμος, στα συγκεκριμένα προβλήματα, το πόσο γρήγορα εκπαιδεύεται ο πράκτορας και πόσο καλή είναι αυτή η εκπαίδευση.

4.2 Ενισχυτική Μάθηση Ενός Πράκτορα

4.2.1 Εισαγωγή

Αρχικά υλοποιήθηκαν δύο αλγόριθμοι ΧΔ, ο εκτός-πολιτικής αλγόριθμος Q-Learning και ο εντός-πολιτικής αλγόριθμος SARSA. Στη συνέχεια αποφασίσαμε να δοκιμάσουμε τον αλγόριθμο PHC, ο οποίος είναι μια απλή επέκταση του αλγόριθμου Q-Learning, και τον αλγόριθμο WoLF-PHC, ο οποίος είναι μια επέκταση του αλγόριθμου PHC. Αυτοί οι αλγόριθμοι εφαρμόζονται στα παίγνια μικτής στρατηγικής, και παρά το γεγονός ότι

χρησιμοποιούνται σαν αλγόριθμοι ΕΜΠΠ σκεφτήκαμε ότι θα ήταν ενδιαφέρον να δούμε πως συμπεριφέρονται σε αυτού τους είδους προβλήματα, τα οποία χρησιμοποιούν μόνο ένα πράκτορα. Έπειτα ασχοληθήκαμε με αλγόριθμους ΙΕΜ. Πιο συγκεκριμένα ασχοληθήκαμε με την ιεραρχική διάσπαση MAXQ και υλοποιήθηκε ο αλγόριθμος MAXQ-Q, ο οποίος είναι μια ιεραρχική επέκταση του αλγόριθμου Q-Learning. Από τα αποτελέσματα παρατηρήσαμε ότι ο αλγόριθμος MAXQ-Q έχει πολύ καλύτερη απόδοση από τον μη ιεραρχικό αλγόριθμο Q-Learning. Στη συνέχεια, αφού μελετήσαμε αρκετές προηγούμενες εργασίες που αφορούσαν αλγόριθμους ΙΕΜ προσέξαμε ότι κανείς δεν είχε δοκιμάσει να υλοποιήσει ιεραρχικά τους αλγόριθμους PHC και WoLF-PHC. Έτσι, αφού προηγουμένως είχαμε παρατηρήσει ικανοποιητικά αποτελέσματα όσον αφορά αυτούς τους δύο αλγόριθμους, αποφασίσαμε να τους υλοποιήσουμε Ιεραρχικά χρησιμοποιώντας την MAXQ διάσπαση.

Όπως έχω προαναφέρει το TP και ETP είναι επεισοδιακά. Σε κάθε επεισόδιο μετράμε τον αριθμό των βημάτων που κάνει ο πράκτορας μέχρι να επιτύχει τον στόχο του. Αποφασίσαμε να κάνουμε ένα αριθμό από δοκιμές για τον κάθε αλγόριθμο, όπου σε κάθε δοκιμή ο αλγόριθμος θα τρέχει για ένα αριθμό επεισοδίων. Στο τέλος παίρνουμε τον μέσο όρο των βημάτων που κάνει ο πράκτορας, ούτως ώστε να εξαχθούν καλύτερα και πιο αντιπροσωπευτικά αποτελέσματα.

Ακολουθούν τα αποτελέσματα από την εφαρμογή αυτών τον αλγορίθμων στο TP και στο ETP.

4.2.2 Αποτελέσματα εφαρμογής των αλγόριθμων στο απλό και στο εκτεταμένο Taxi Problem

Ο αριθμός των δοκιμών στο απλό TP επιλέχθηκε να είναι 50, αφού όσο πιο μεγάλος είναι ο αριθμός των δοκιμών τόσο πιο ακριβή αποτελέσματα μπορούμε να πάρουμε, και αριθμός των επεισοδίων 3000, αφού η μάθηση συμβαίνει γρήγορα και δεν χρειάζονται περισσότερα επεισόδια.

Ο αριθμός των δοκιμών στο ETP επιλέχθηκε να είναι 30, λόγω του ότι χρειάζεται περισσότερος χρόνος για να τρέξει αφού το σύνολο καταστάσεων-ενεργειών αυξάνεται σημαντικά, και αριθμός των επεισοδίων 5000, αφού λόγο του μεγαλύτερου συνόλου καταστάσεων-ενεργειών σε σχέση με το απλό TP η μάθηση συμβαίνει πιο αργά και έτσι χρειάζονται περισσότερα επεισόδια.

Οι αμοιβές για κάθε κίνηση του πράκτορα έχουν καθοριστεί σύμφωνα με το κεφάλαιο 2.6 όπου περιγράφεται το TP. Οι ψευδοαμοιβές, για τους αλγόριθμους MAXQ-Q, MAXQ-PHC και MAXQ-WoLF-PHC μετά από κάποιες δοκιμές αποφασίστηκε να είναι 0 για κάθε τερματική κατάσταση και -100 για κάθε άλλη κατάσταση.

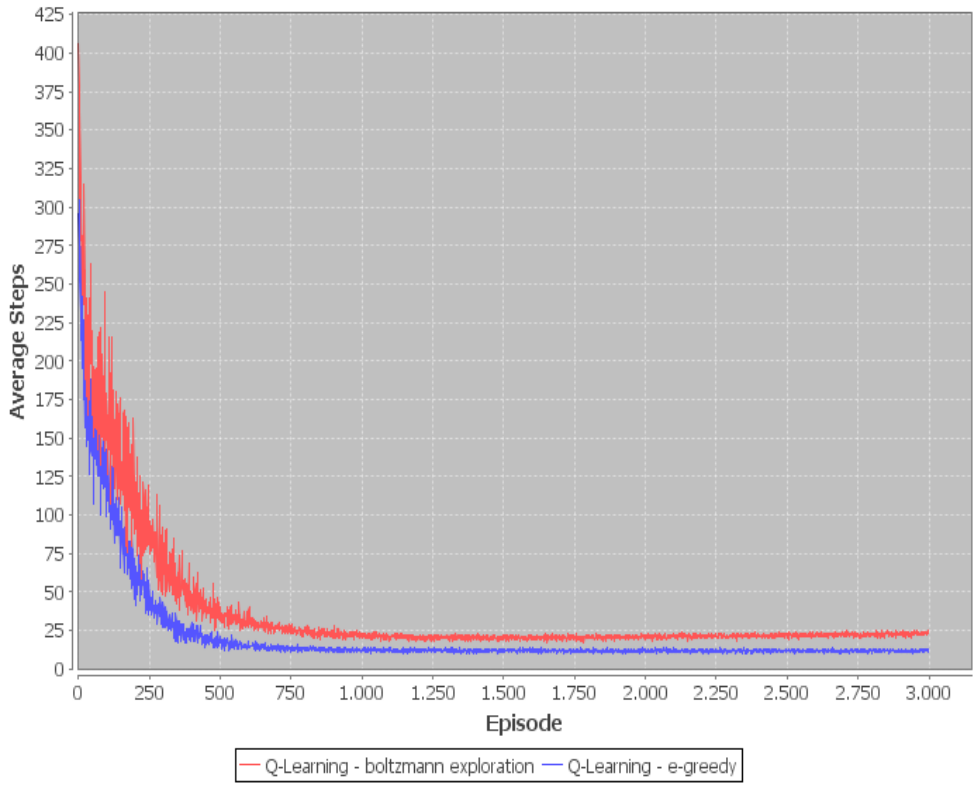
Στα πιο κάτω αποτελέσματα ακλουθούμε τους εξής συμβολισμούς για τις παραμέτρους: Συμβολίζουμε τον ρυθμό μάθησης με *α* και τον ρυθμό έκπτωσης με *γ*. Όσον αφορά την πολιτική ε-greedy συμβολίζουμε το έψιλον με *ε*, και στην πολιτική Boltzmann την αρχική θερμοκρασία με *t* και τον ρυθμό μείωσης της θερμοκρασίας με *cr* (cooling rate). Στους αλγόριθμους PHC και MAXQ-PHC συμβολίζουμε το δέλτα με *δ* και στους αλγόριθμους WoLF-PHC και MAXQ-WoLF-PHC συμβολίζουμε το δέλτα winning με *δw* και το δέλτα losing με *δl*.

4.2.2.1 Εφαρμογή αλγόριθμου Q-Learning Ενός Πράκτορα

Αυτός ο αλγόριθμος έχει υλοποιηθεί ούτως ώστε να μπορεί να χρησιμοποιεί είτε την πολιτική ε-greedy είτε την πολιτική Boltzmann. Μετά από κάποιες δοκιμές αποφασίστηκε όπως χρησιμοποιηθούν οι ακόλουθες τιμές για τις παραμέτρους: ε=0.1, στην περίπτωση της ε-greedy πολιτικής, και στην περίπτωση της πολιτικής Boltzmann t=80 και cr=0.98. Επίσης θέσαμε α=0.3 και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές.

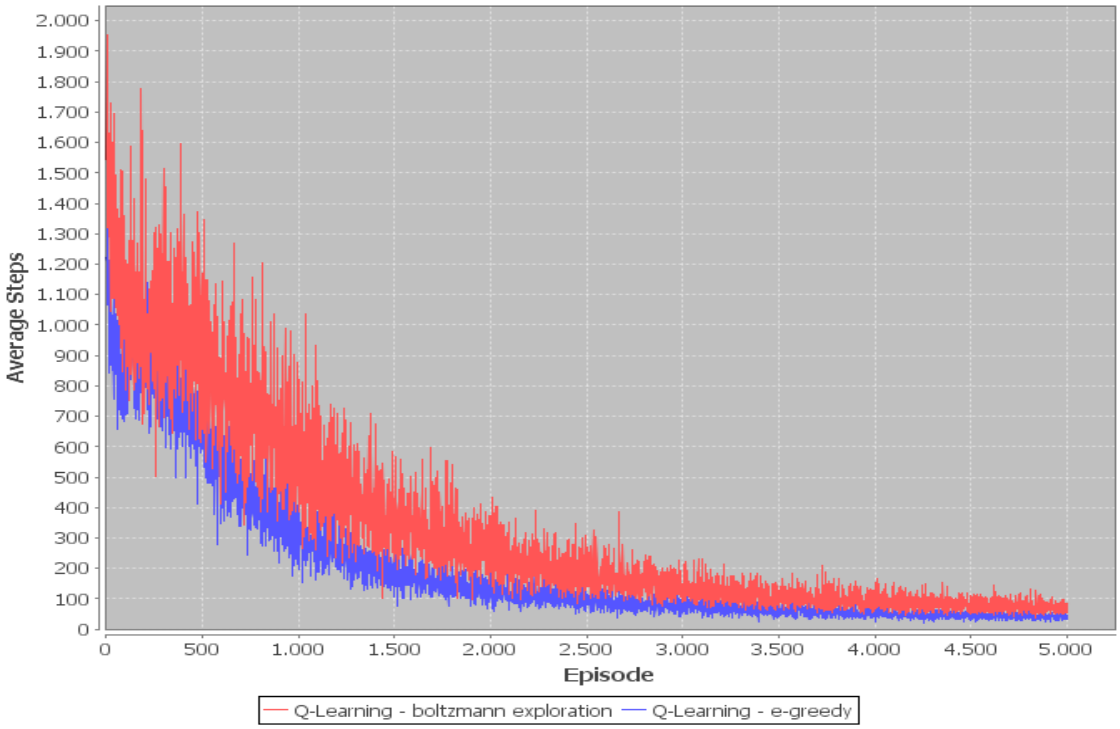
Το Σχήμα 4.1 και το Σχήμα 4.2 δείχνουν το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκε ο πράκτορας για να επιτύχει τον στόχο του, στο TP και στο ETP αντίστοιχα.

Παρατηρούμε ότι ο πράκτορας εκπαιδεύεται με επιτυχία τόσο όταν χρησιμοποιεί την ε-greedy πολιτική, καθώς και όταν χρησιμοποιεί την πολιτική Boltzmann. Όμως, βλέπουμε ότι η σύγκλιση επέρχεται πιο γρήγορα και σε μικρότερο αριθμό βημάτων όταν χρησιμοποιείται η ε-greedy πολιτική. Αυτό σημαίνει ότι η επιλογή των ενεργειών του πράκτορα όταν χρησιμοποιεί την ε-greedy πολιτική είναι πιο σταθερή, σε σχέση με την αντίστοιχη περίπτωση όπου χρησιμοποιεί την πολιτική Boltzmann, αφού στην ε-greedy πολιτική τείνει να επιλέγει την πιο άπληστη ενέργεια χωρίς να χρησιμοποιεί τον παράγοντα της θερμοκρασίας, που εμπεριέχεται στην πολιτική Boltzmann και προσφέρει κάποια τυχαιότητα στην επιλογή των ενεργειών. Έτσι συμπεραίνουμε ότι η εφαρμογή του αλγόριθμου Q-Learning στο πρόβλημα του TP ενός πράκτορα είναι καλύτερα να χρησιμοποιεί την ε-greedy πολιτική.



Σχήμα 4.1: Μέσος όρος βημάτων χρησιμοποιώντας τον αλγόριθμο Q-Learning Ενός Πράκτορα με ε-greedy και Boltzmann πολιτική για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy και την Boltzmann πολιτική αντίστοιχα σε κάθε περίπτωση. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3, γ=0.95, t=80 και cr=0.98. Ο πράκτορας εκπαιδεύεται με επιτυχία και με τις δύο πολιτικές, όμως η σύγκλιση επέρχεται πιο γρήγορα και σε μικρότερο αριθμό βημάτων όταν χρησιμοποιείται η ε-greedy πολιτική.



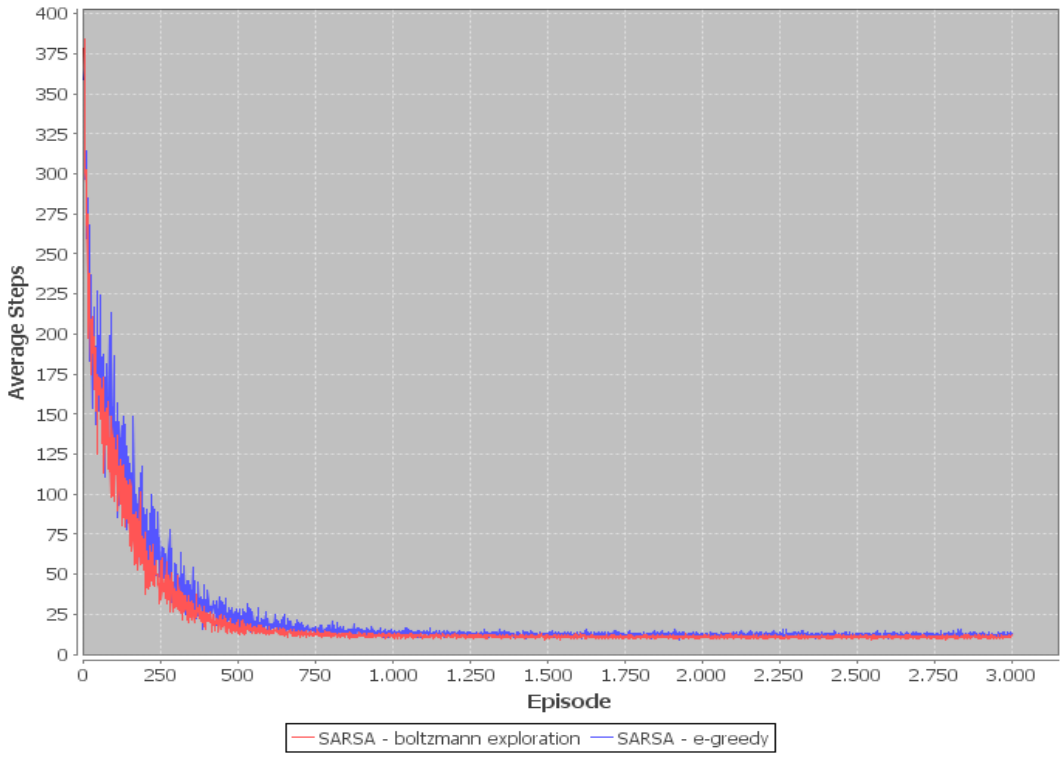
Σχήμα 4.2: Μέσος όρος βημάτων χρησιμοποιώντας τον αλγόριθμο Q-Learning Ενός Πράκτορα με ε-greedy και Boltzmann πολιτική για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy και την Boltzmann πολιτική αντίστοιχα σε κάθε περίπτωση. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3, γ=0.95 , t=80 και cr=0.98. Ο πράκτορας εκπαιδεύεται με επιτυχία και με τις δύο πολιτικές, όμως η σύγκλιση επέρχεται πιο γρήγορα και σε μικρότερο αριθμό βημάτων όταν χρησιμοποιείται η ε-greedy πολιτική.

4.2.2.2 Εφαρμογή αλγόριθμου SARSA Ενός Πράκτορα

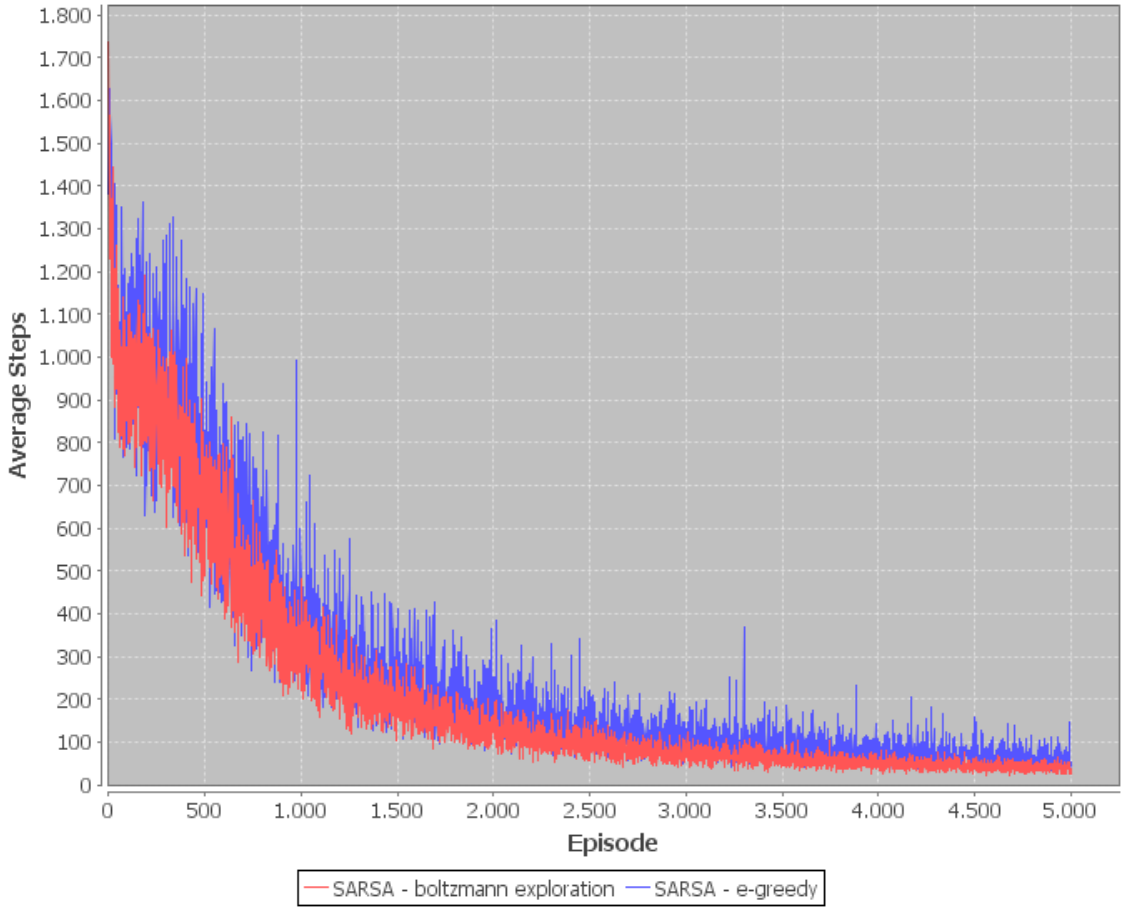
Όπως και ο προηγούμενος αλγόριθμος, έτσι και αυτός έχει υλοποιηθεί ούτως ώστε να μπορεί να χρησιμοποιεί είτε την πολιτική ε-greedy είτε την πολιτική Boltzmann. Μετά από κάποιες δοκιμές αποφασίστηκε όπως χρησιμοποιηθούν οι ακόλουθες τιμές για τις παραμέτρους: ε=0.1, στην περίπτωση της ε-greedy πολιτικής, και στην περίπτωση της πολιτικής Boltzmann t=80 και cr=0.98. Επίσης θέσαμε α=0.3 και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές.

Το Σχήμα 4.3 και το Σχήμα 4.4 δείχνουν το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκε ο πράκτορας για να επιτύχει τον στόχο του, στο TP και στο ETP αντίστοιχα. Παρατηρούμε ότι ο πράκτορας εκπαιδεύεται με επιτυχία τόσο όταν χρησιμοποιεί την ε-greedy πολιτική, καθώς και όταν χρησιμοποιεί την πολιτική Boltzmann . Βλέπουμε επίσης, ότι στην περίπτωση όπου χρησιμοποιείται η πολιτική Boltzmann η σύγκλιση επέρχεται ελαφρώς πιο γρήγορα, σε σχέση με την περίπτωση όπου χρησιμοποιείται η ε-greedy πολιτική. Ακόμη, στην περίπτωση του ETP παρατηρούμε ότι η γραφική παράσταση που αφορά την εφαρμογή του αλγόριθμου χρησιμοποιώντας την πολιτική Boltzmann είναι πιο απαλή από την γραφική παράσταση που αφορά την ε-greedy πολιτική. Από αυτό συμπεραίνουμε ότι η εφαρμογή του αλγόριθμου SARSA σε grid world προβλήματα ενός πράκτορα είναι καλύτερα να χρησιμοποιεί την πολιτική Boltzmann, ειδικά όταν το σύνολο καταστάσεων ενεργειών είναι μεγάλο.



Σχήμα 4.3: Μέσος όρος βημάτων χρησιμοποιώντας τον αλγόριθμο SARSA Ενός Πράκτορα με ε-greedy και Boltzmann πολιτική για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy και την Boltzmann πολιτική αντίστοιχα σε κάθε περίπτωση. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3 , γ=0.95 , t=80 και cr=0.98. Ο πράκτορας εκπαιδεύεται με επιτυχία και με τις δύο πολιτικές με ελαφρώς πιο γρήγορη σύγκλιση όταν χρησιμοποιείται η πολιτική Boltzmann.



Σχήμα 4.4: Μέσος όρος βημάτων χρησιμοποιώντας τον αλγόριθμο SARSA Ενός Πράκτορα με ε-greedy και Boltzmann πολιτική για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy και την Boltzmann πολιτική αντίστοιχα σε κάθε περίπτωση. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3 , γ=0.95 , t=80 και cr=0.98. Ο πράκτορας εκπαιδεύεται με επιτυχία και με τις δύο πολιτικές με ελαφρώς πιο γρήγορη σύγκλιση όταν χρησιμοποιείται η πολιτική Boltzmann και με πιο απαλή γραφική παράσταση.

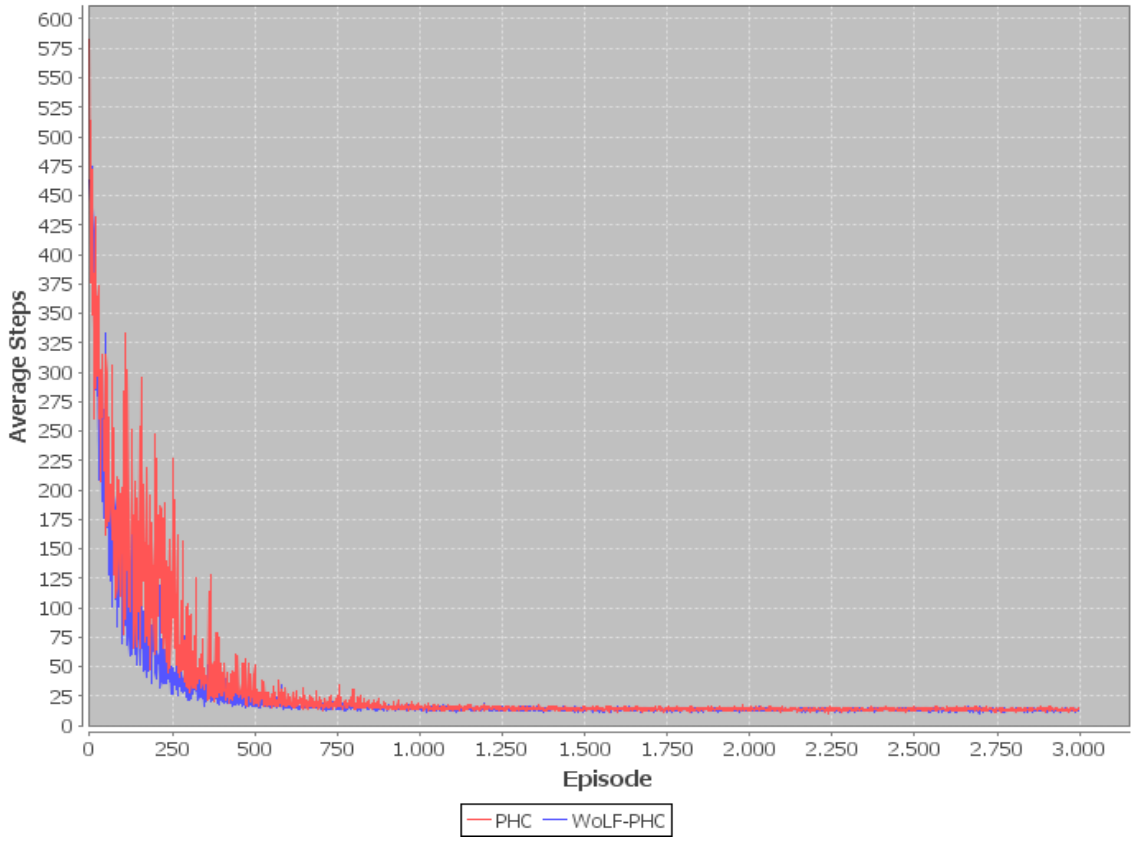
4.2.2.3 Εφαρμογή αλγορίθμων PHC και WoLF-PHC Ενός Πράκτορα

Παρά το γεγονός ότι αυτοί οι αλγόριθμοι χρησιμοποιούνται σαν αλγόριθμοι ΕΜΠΠ σκεφτήκαμε ότι θα ήταν ενδιαφέρων να δούμε πως συμπεριφέρονται και σε περιβάλλοντα ενός πράκτορα. Αρχικά υλοποιήθηκε ο αλγόριθμος PHC ο οποίος είναι

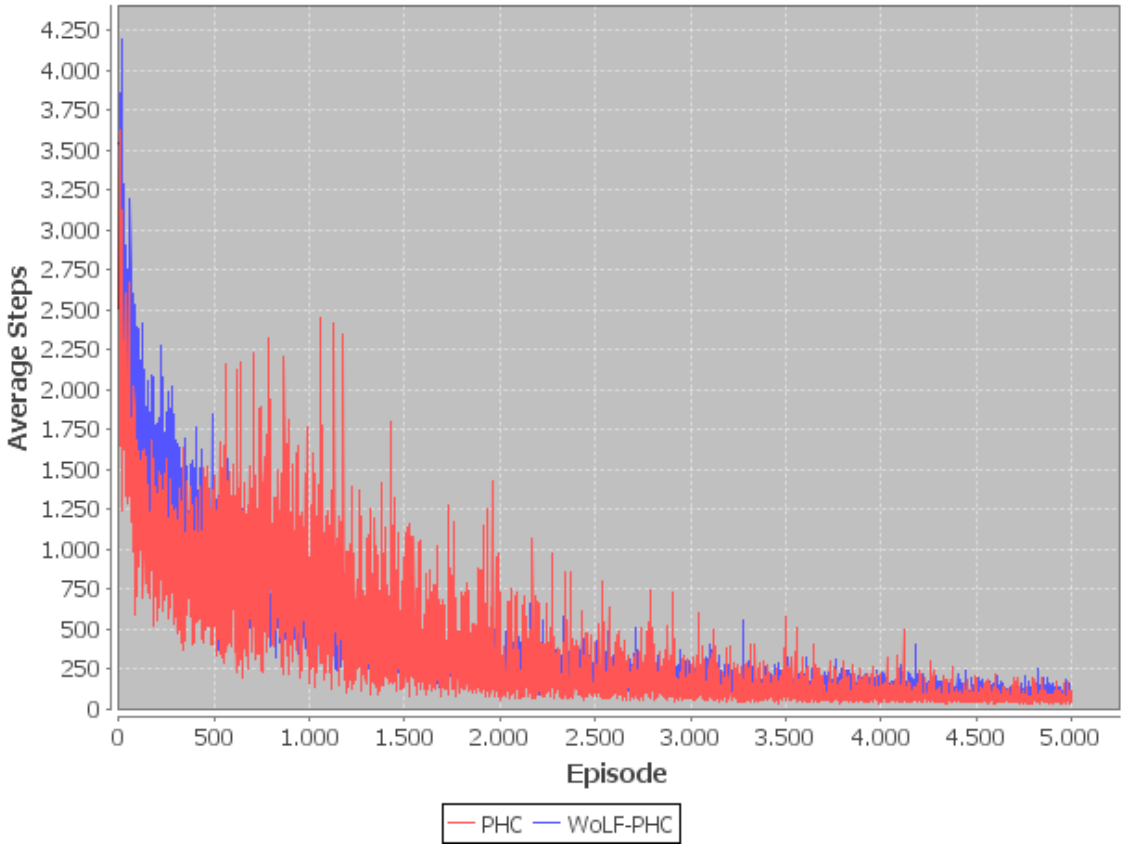
μια απλή επέκταση του αλγόριθμου Q-Learning και στην συνέχεια ο αλγόριθμος WoLF-PHC σαν επέκταση του προηγούμενου.

Μετά από κάποιες δοκιμές αποφασίστηκε όπως χρησιμοποιηθούν οι ακόλουθες τιμές για τις παραμέτρους: ε=0.1 και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές, και στους δύο αλγόριθμους. Επίσης, στον αλγόριθμο PHC καθορίσαμε α=0.3 και δ=0.2, ενώ στον αλγόριθμο WoLF-PHC καθορίσαμε α=0.5, δw=0.05 και δl=0.2, για την εφαρμογή του στο TP, και α=0.3, δw=0.01 και δl=0.04 για την εφαρμογή του στο ETP, αφού σύμφωνα με τον αλγόριθμο πρέπει δl> δw.

Το Σχήμα 4.5 και το Σχήμα 4.6 δείχνουν το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκε ο πράκτορας για να επιτύχει τον στόχο του, στο TP και στο ETP αντίστοιχα. Παρατηρούμε ότι ο πράκτορας εκπαιδεύεται με επιτυχία, παρά το γεγονός ότι χρησιμοποιήθηκαν δύο αλγόριθμοι μικτής στρατηγικής που εφαρμόζονται σε περιβάλλοντα πολλαπλών πρακτόρων. Επίσης συγκρίνοντας τους δύο αλγόριθμους παρατηρούμε ότι η γραφική παράσταση με τον μέσο αριθμό βημάτων για τον αλγόριθμο WoLF-PHC είναι πιο απαλή από την γραφική παράσταση του αλγόριθμου PHC, κάτι το οποίο φαίνεται πολύ καλύτερα κοιτάζοντας τα αποτελέσματα από την εφαρμογή των αλγορίθμων στο ETP. Αυτό ήταν αναμενόμενο αφού ο αλγόριθμος WoLF-PHC είναι μια επέκταση του αλγόριθμου PHC η οποία έγινε για να βελτιωθεί ο αλγόριθμος ούτως ώστε να ενθαρρύνεται η σύγκλιση.



Σχήμα 4.5: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους PHC και WoLF-PHC Ενός Πράκτορα για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP
Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3,γ=0.95 και δ=0.2 στον αλγόριθμο PHC, και ε=0.1, α=0.5,γ=0.95, δw=0.05 και δl=0.2 στον στον αλγόριθμο WoLF-PHC. Ο πράκτορας εκπαιδεύεται με επιτυχία και με τους δύο αλγόριθμους, με λίγο καλύτερη απόδοση όταν χρησιμοποιείται ο αλγόριθμος WoLF-PHC, αφού η γραφική παράσταση του αλγόριθμου WoLF-PHC είναι πιο απαλή από την γραφική παράσταση του αλγόριθμου PHC.



Σχήμα 4.6: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους PHC και WoLF-PHC Ενός Πράκτορα για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο ETP

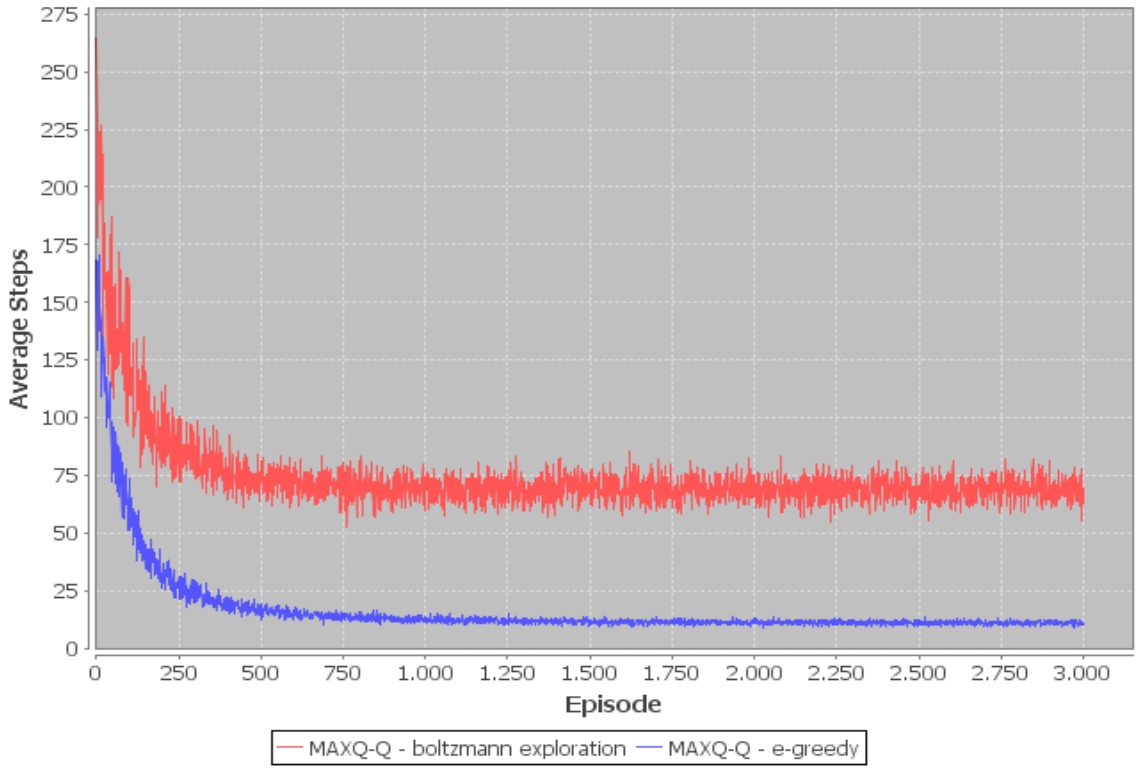
Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3,γ=0.95 και δ=0.2 στον στον αλγόριθμο PHC, και ε=0.1, α=0.3,γ=0.95, δw=0.01 και δl=0.04 στον στον αλγόριθμο WoLF-PHC. Ο πράκτορας εκπαιδεύεται με επιτυχία και με τους δύο αλγόριθμους, με λίγο καλύτερη απόδοση όταν χρησιμοποιείται ο αλγόριθμος WoLF-PHC, αφού η γραφική παράσταση του αλγόριθμου WoLF-PHC είναι πιο απαλή από την γραφική παράσταση του αλγόριθμου PHC.

4.2.2.4 Εφαρμογή αλγόριθμου MAXQ-Q Ενός Πράκτορα

Συνεχίσαμε με την υλοποίηση του αλγόριθμου MAXQ-Q,ο οποίος είναι μια ιεραρχική υλοποίηση του αλγόριθμου Q-Learning χρησιμοποιώντας την Ιεραρχική διάσπαση MAXQ. Όπως και ο Q-Learning, έτσι και ο MAXQ-Q έχει υλοποιηθεί ούτως ώστε να μπορεί να χρησιμοποιεί είτε την πολιτική ε-greedy είτε την πολιτική Boltzmann για να μπορέσουμε στη συνέχεια να συγκρίνουμε την απόδοση τους. Μετά από κάποιες

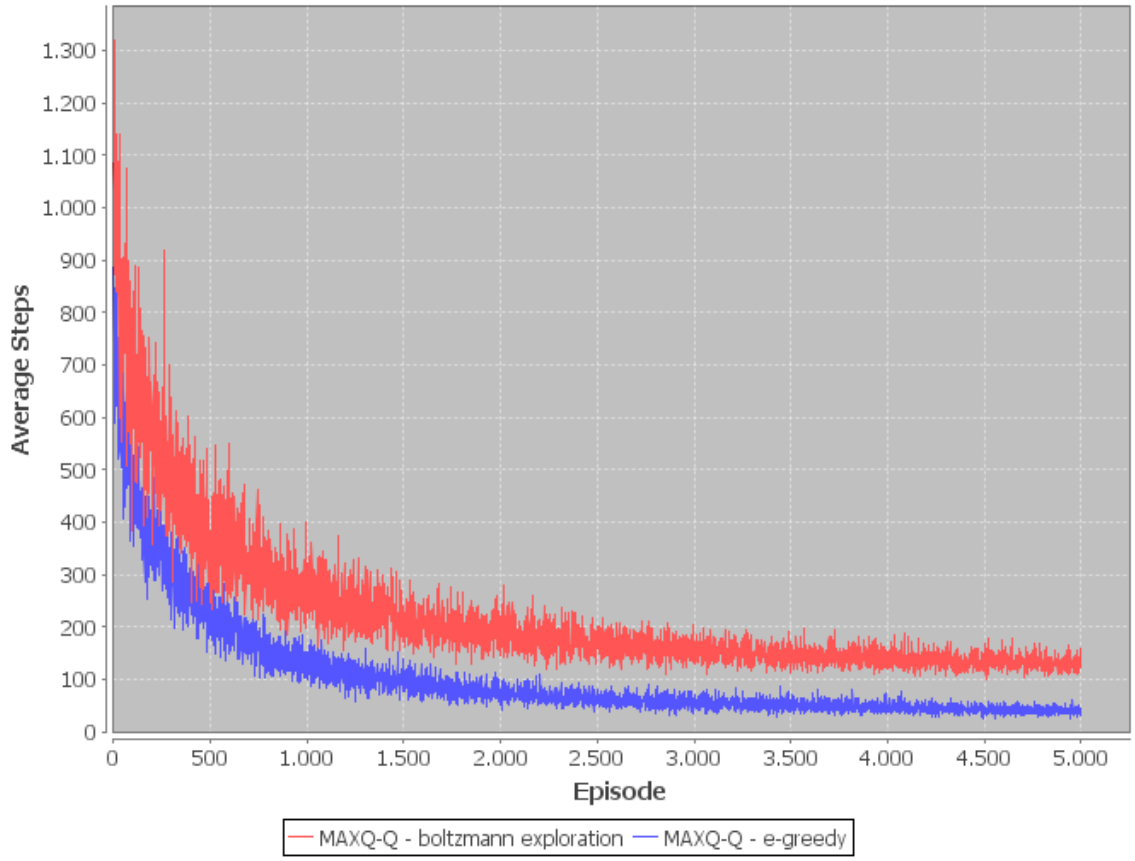
δοκιμές αποφασίστηκε όπως χρησιμοποιηθούν οι ακόλουθες τιμές για τις παραμέτρους: ε=0.1, στην περίπτωση της ε-greedy πολιτικής, και στην περίπτωση της πολιτικής Boltzmann t=80. Επίσης θέσαμε α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές. Ακόμη, στην περίπτωση της πολιτικής Boltzmann θέσαμε τους ακόλουθους ρυθμούς μείωσης της θερμοκρασίας για κάθε Max κόμβο: MaxRoot=0.9996, MaxPut=0.9996, MaxGet=0.9939 και MaxNavigatee=0.9939, οι οποίοι έχουν χρησιμοποιηθεί από τον Dietterich (2000) στην παρουσίαση του αλγόριθμου MAXQ-Q.

Το Σχήμα 4.7 και το Σχήμα 4.8 δείχνουν το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκε ο πράκτορας για να επιτύχει τον στόχο του, στο TP και στο ETP αντίστοιχα. Παρατηρούμε ότι ο πράκτορας εκπαιδεύεται με επιτυχία τόσο όταν χρησιμοποιεί την ε-greedy πολιτική, καθώς και όταν χρησιμοποιεί την πολιτική Boltzmann . Όμως, βλέπουμε ότι η σύγκλιση επέρχεται σε πολύ μικρότερο αριθμό βημάτων όταν χρησιμοποιείται η ε-greedy πολιτική.



Σχήμα 4.7: Μέσος όρος βημάτων χρησιμοποιώντας τον αλγόριθμο MAXQ-Q Ενός Πράκτορα με ε-greedy και Boltzmann πολιτική για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy και την Boltzmann πολιτική αντίστοιχα σε κάθε περίπτωση. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, t=80, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές. Ο ρυθμός μείωσης της θερμοκρασίας για κάθε Max κόμβο είναι για τον MaxRoot=0.9996, τον MaxPut=0.9996, τον MaxGet=0.9939 και τον MaxNavigatee=0.9939.Ο πράκτορας εκπαιδεύεται με επιτυχία και με τις δύο πολιτικές, όμως η σύγκλιση επέρχεται πιο γρήγορα και σε μικρότερο αριθμό βημάτων όταν χρησιμοποιείται η ε-greedy πολιτική.



Σχήμα 4.8: Μέσος όρος βημάτων χρησιμοποιώντας τον αλγόριθμο MAXQ-Q Ενός Πράκτορα με ε-greedy και Boltzmann πολιτική για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy και την Boltzmann πολιτική αντίστοιχα σε κάθε περίπτωση. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, t=80, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές. Ο ρυθμός μείωσης της θερμοκρασίας για κάθε Max κόμβο είναι για τον MaxRoot=0.9996, τον MaxPut=0.9996, τον MaxGet=0.9939 και τον MaxNavigatee=0.9939. Ο πράκτορας εκπαιδεύεται με επιτυχία και με τις δύο πολιτικές, όμως η σύγκλιση επέρχεται πιο γρήγορα και σε μικρότερο αριθμό βημάτων όταν χρησιμοποιείται η ε-greedy πολιτική.

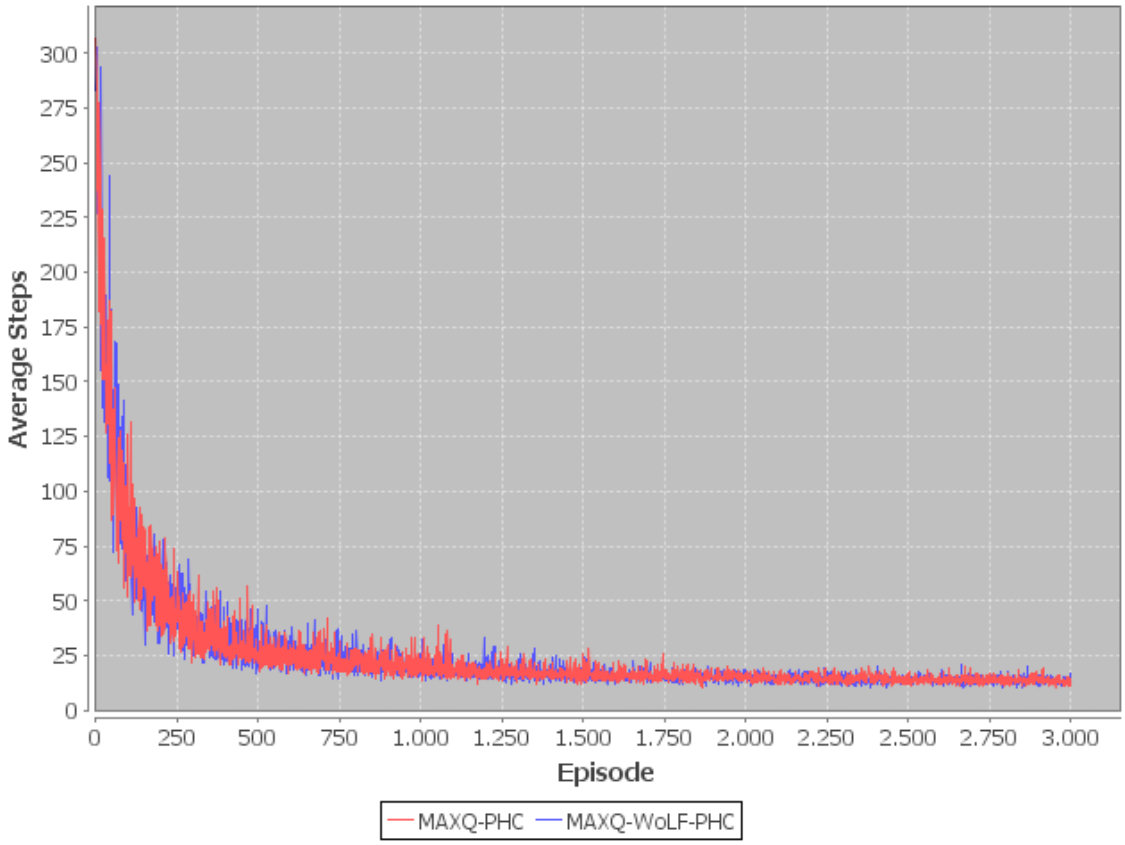
4.2.2.5 Εφαρμογή αλγορίθμων MAXQ-PHC και MAXQ-WoLF-PHC Ενός Πράκτορα

Στη συνέχεια, έχοντας μελετήσει αρκετές προηγούμενες εργασίες που αφορούσαν αλγόριθμους IEM προσέξαμε ότι κανείς δεν είχε δοκιμάσει να υλοποιήσει Ιεραρχικά τους αλγόριθμους PHC και WoLF-PHC. Έτσι, αφού είχαμε ήδη παρατηρήσει αρκετά

καλά αποτελέσματα όσον αφορά αυτούς τους δύο αλγόριθμους, αποφασίσαμε να τους υλοποιήσουμε Ιεραρχικά χρησιμοποιώντας την MAXQ διάσπαση.

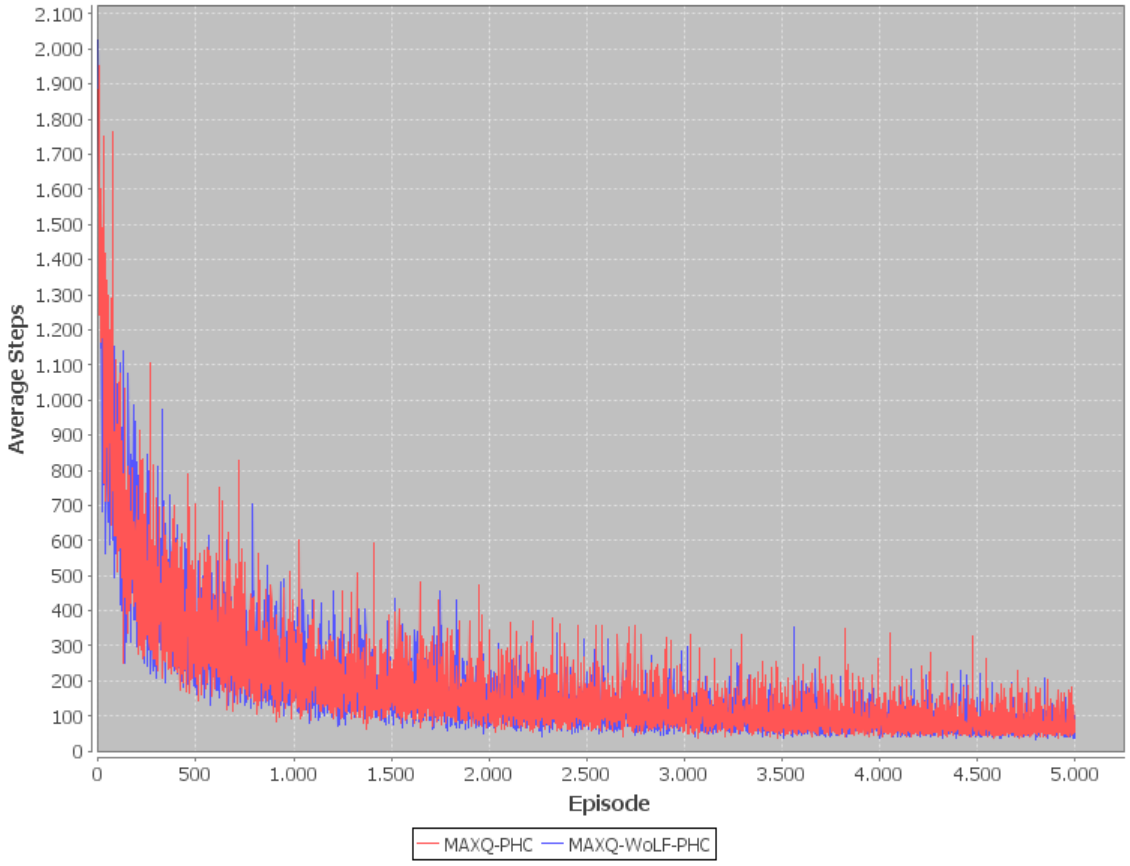
Μετά από κάποιες δοκιμές αποφασίστηκε όπως χρησιμοποιηθούν οι ακόλουθες τιμές για τις παραμέτρους: ε=0.1 και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές, και στους δύο αλγόριθμους. Επίσης, στον αλγόριθμο MAXQ-PHC καθορίσαμε α=0.5 για κάθε Max κόμβο του γράφου και δ=0.2, ενώ στον αλγόριθμο MAXQ-WoLF-PHC καθορίσαμε α=0.5 για κάθε Max κόμβο, και δw=0.05 και δl=0.2 στο TP και δw=0.1 και δl=0.4 στο ETP, αφού σύμφωνα με τον αλγόριθμο πρέπει δl> δw.

Το Σχήμα 4.9 και το Σχήμα 4.10 δείχνουν το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκε ο πράκτορας για να επιτύχει τον στόχο του, στο TP και στο ETP αντίστοιχα. Παρατηρούμε ότι ο πράκτορας εκπαιδεύεται με επιτυχία, παρά το γεγονός ότι χρησιμοποιήθηκαν δύο αλγόριθμοι μικτής στρατηγικής που εφαρμόζονται σε περιβάλλοντα πολλαπλών πρακτόρων. Επίσης βλέπουμε ότι η σύγκλιση επέρχεται την ίδια χρονική στιγμή και στους δύο αλγόριθμους.



Σχήμα 4.9: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC Ενός Πράκτορα για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95 και δ=0.2 στον αλγόριθμο MAXQ-PHC, και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.05 και δl=0.2 στον αλγόριθμο MAXQ-WoLF-PHC. Ο πράκτορας εκπαιδεύεται με επιτυχία, και με τους δύο αλγόριθμους παρά το γεγονός ότι είναι δύο αλγόριθμοι μικτής στρατηγικής που εφαρμόζονται σε περιβάλλοντα πολλαπλών πρακτόρων. Επίσης η σύγκλιση επέρχεται την ίδια χρονική στιγμή και στους δύο αλγόριθμους.



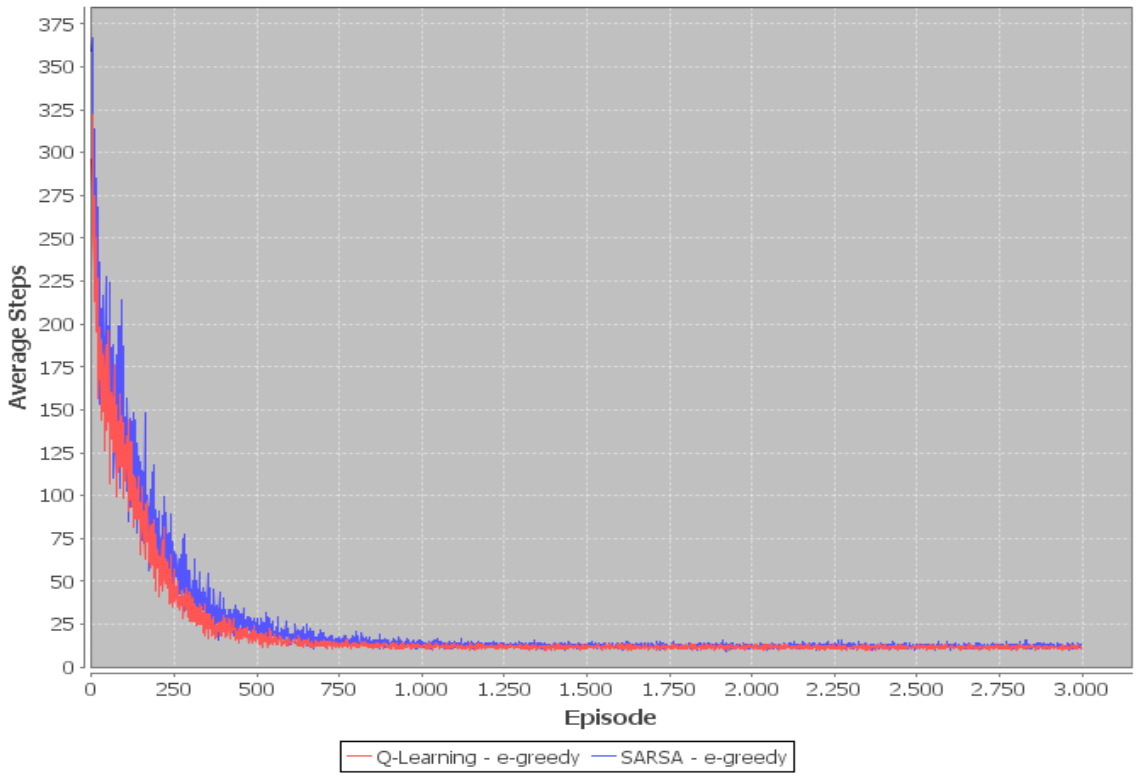
Σχήμα 4.10: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC Ενός Πράκτορα για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95 και δ=0.2 στον αλγόριθμο MAXQ-PHC, και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.1 και δl=0.4 στον αλγόριθμο MAXQ-WoLF-PHC. Ο πράκτορας εκπαιδεύεται με επιτυχία, και με τους δύο αλγόριθμους παρά το γεγονός ότι είναι δύο αλγόριθμοι μικτής στρατηγικής που εφαρμόζονται σε περιβάλλοντα πολλαπλών πρακτόρων. Επίσης η σύγκλιση επέρχεται την ίδια χρονική στιγμή και στους δύο αλγόριθμους.

4.2.2.6 Σύγκριση αλγορίθμων Q-Learning και SARSA Ενός Πράκτορα

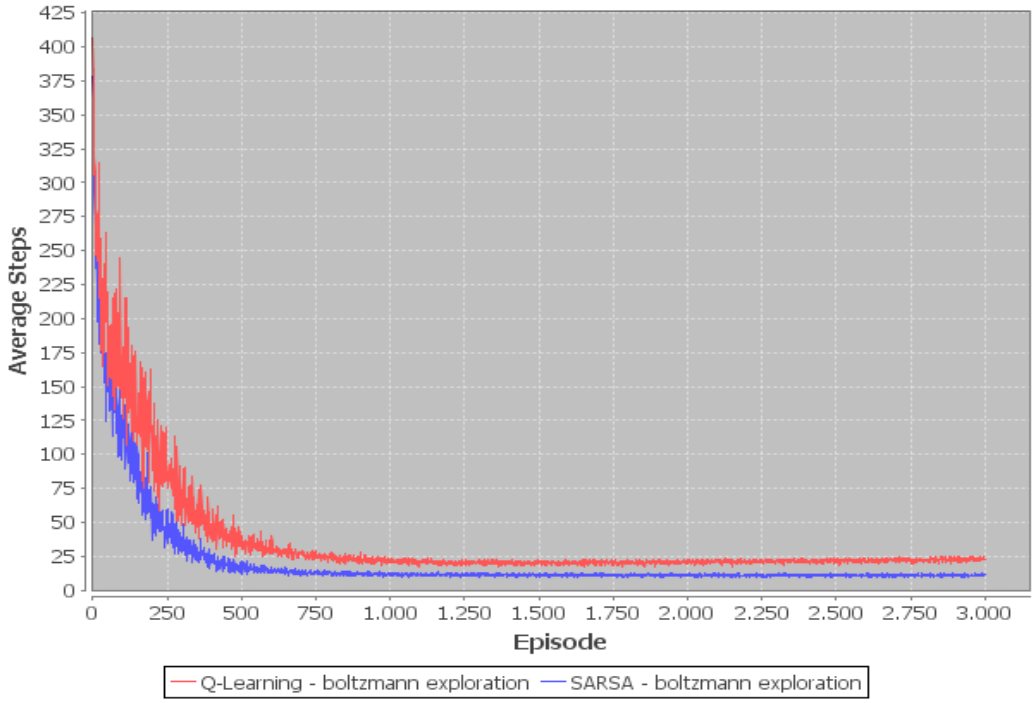
Στη συνέχεια συγκρίναμε τα αποτελέσματα που πήραμε από τους δύο αλγόριθμους ΧΔ. Το Σχήμα 4.11 δείχνει τα αποτελέσματα από τους δύο αλγόριθμους έχοντας χρησιμοποιήσει την ε-greedy πολιτική, και το Σχήμα 4.12 τα αποτελέσματα χρησιμοποιώντας την πολιτική Boltzmann για το απλό ΤΡ, ενώ τα Σχήματα 4.13 και

4.14 δείχνουν τα αποτελέσματα χρησιμοποιώντας την ε-greedy και την Boltzmann πολιτική για το ETP αντίστοιχα. Στην περίπτωση που χρησιμοποιείται η ε-greedy πολιτική παρατηρούμε ότι η σύγκλιση στον αλγόριθμο Q-Learning επέρχεται ελαφρώς πιο γρήγορα από ότι στον αλγόριθμο SARSA. Επίσης κοιτάζοντας τα αποτελέσματα με την εφαρμογή των αλγορίθμων στο ETP, βλέπουμε ότι η γραφική παράσταση που αφορά τον αλγόριθμο Q-Learning είναι πιο απαλή από την γραφική παράσταση που αφορά τον αλγόριθμο SARSA. Όταν χρησιμοποιείται η πολιτική Boltzmann παρατηρούμε ότι ο αλγόριθμος SARSA συγκλίνει πιο γρήγορα και σε μικρότερο αριθμό βημάτων από τον αλγόριθμο Q-Learning. Έτσι συμπεραίνουμε ότι στο πρόβλημα του TP ενός πράκτορα είναι καλύτερα να εφαρμόζεται ο αλγόριθμος Q-Learning χρησιμοποιώντας την ε-greedy πολιτική και ο αλγόριθμος SARSA χρησιμοποιώντας την πολιτική Boltzmann.



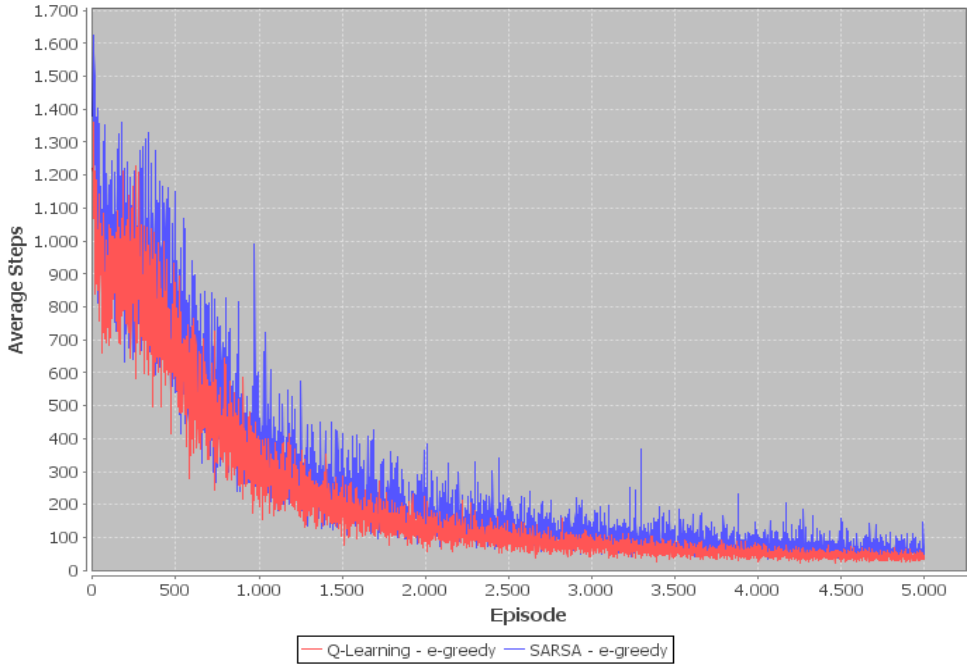
Σχήμα 4.11: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και SARSA Ενός Πράκτορα με ε-greedy πολιτική για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy πολιτική. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3 και γ=0.95. Η σύγκλιση επέρχεται ελαφρώς πιο γρήγορα όταν χρησιμοποιείται ο αλγόριθμος Q-Learning.



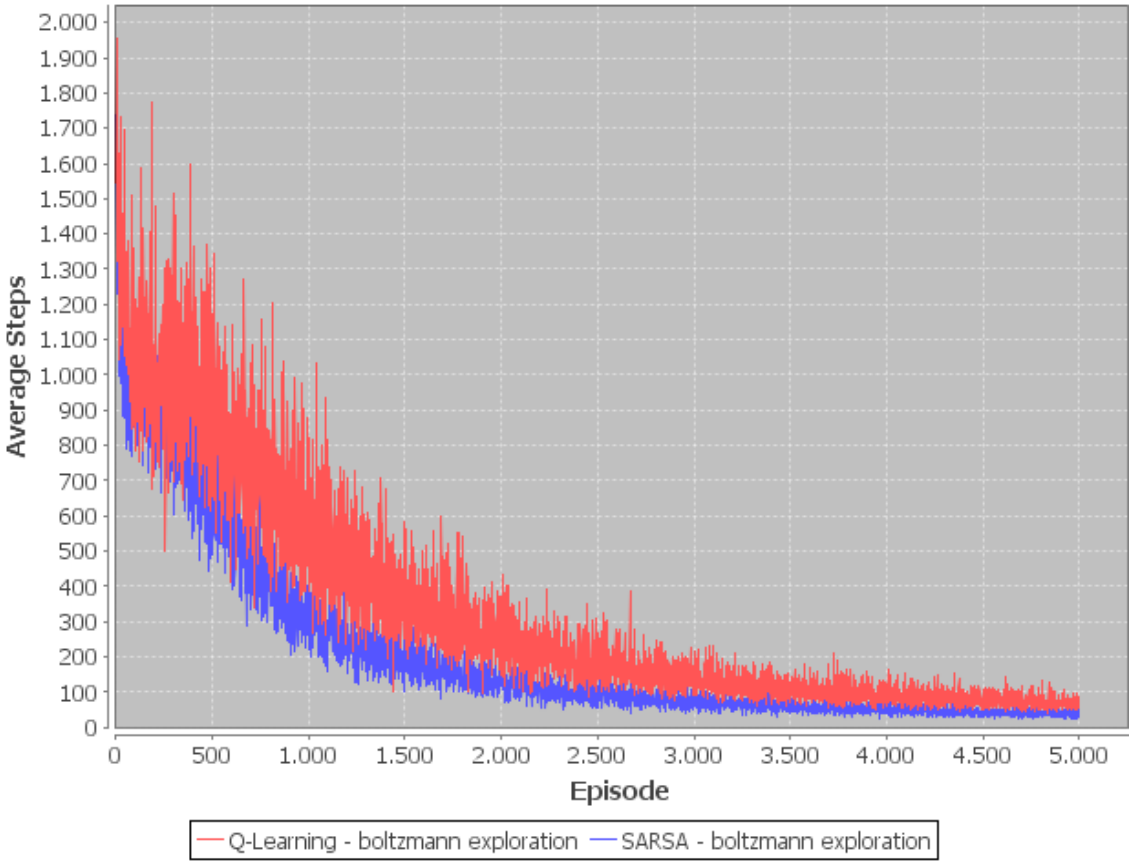
Σχήμα 4.12: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και SARSA Ενός Πράκτορα με πολιτική Boltzmann για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την πολιτική Boltzmann. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι α=0.3, γ=0.95 , t=80 και cr=0.98. Η σύγκλιση επέρχεται πιο γρήγορα όταν χρησιμοποιείται ο αλγόριθμος SARSA και σε μικρότερο αριθμό βημάτων.



Σχήμα 4.13: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και SARSA Ενός Πράκτορα με ε-greedy πολιτική για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy πολιτική. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3 και γ=0.95. Η σύγκλιση επέρχεται ελαφρώς πιο γρήγορα όταν χρησιμοποιείται ο αλγόριθμος Q-Learning και η γραφική παράσταση που αφορά την εφαρμογή του αλγόριθμου Q-Learning είναι πιο απαλή από την γραφική παράσταση που αφορά τον αλγόριθμο SARSA.



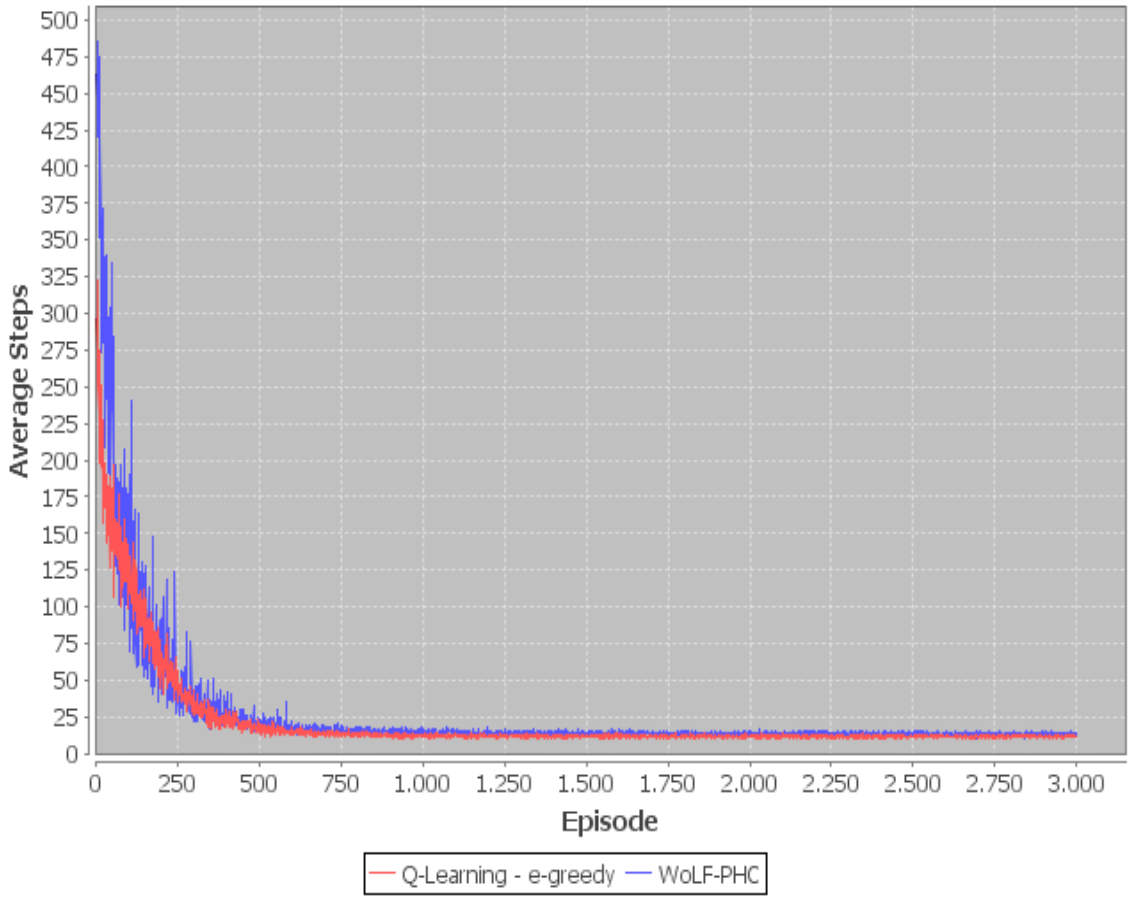
Σχήμα 4.14: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και SARSA Ενός Πράκτορα με πολιτική Boltzmann για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την πολιτική Boltzmann. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι α=0.3, γ=0.95 , t=80 και cr=0.98. Η σύγκλιση επέρχεται πιο γρήγορα όταν χρησιμοποιείται ο αλγόριθμος SARSA και σε μικρότερο αριθμό βημάτων.

4.2.2.7 Σύγκριση αλγορίθμων Q-Learning και WoLF-PHC Ενός Πράκτορα

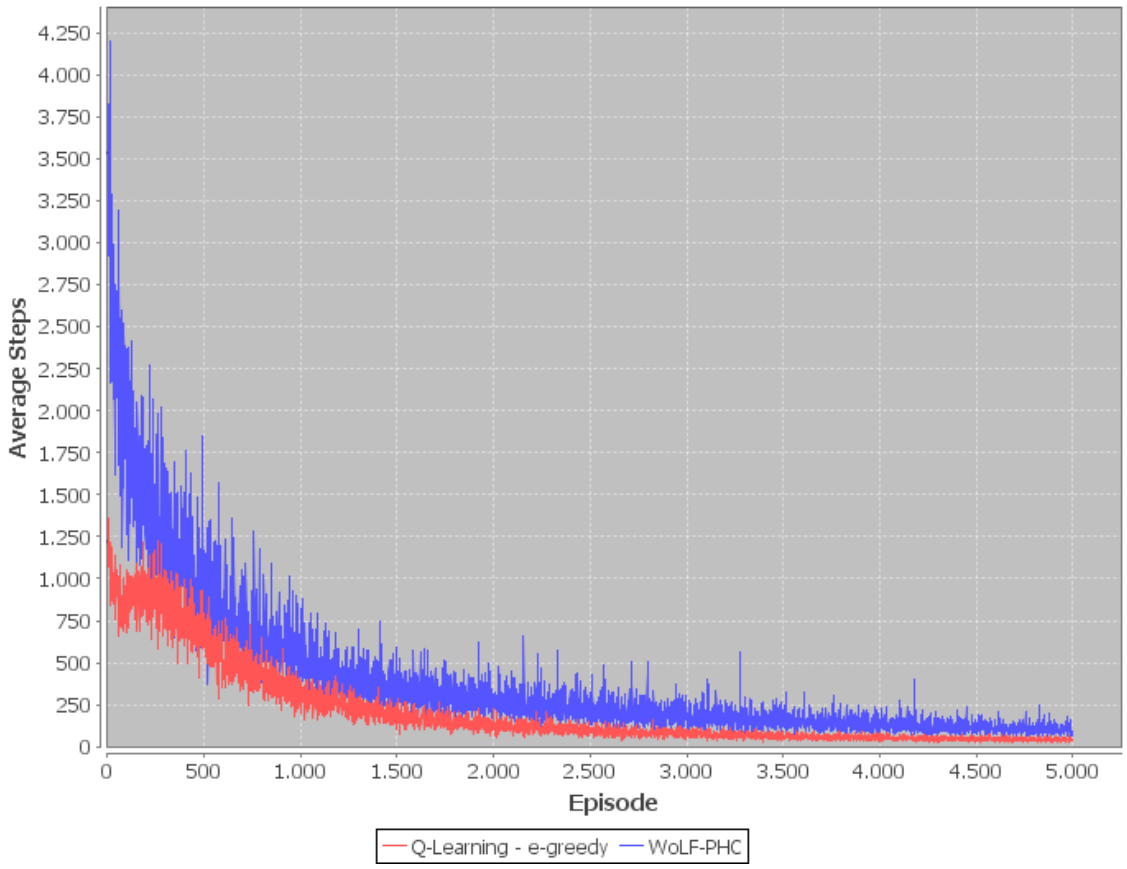
Στη συνέχεια συγκρίναμε τα αποτελέσματα που πήραμε από τους αλγόριθμους Q-Learning χρησιμοποιώντας την ε-greedy πολιτική, αφού είχε καλύτερα αποτελέσματα παρά την πολιτική Boltzmann, και WoLF-PHC. Στο Σχήμα 4.15 φαίνονται τα αποτελέσματα με την εφαρμογή των δύο αλγορίθμων στο απλό TP και στο Σχήμα 4.16 με την εφαρμογή τους στο ETP. Παρατηρούμε ότι στα αρχικά επεισόδια ο μέσος αριθμός βημάτων του πράκτορα, χρησιμοποιώντας τον αλγόριθμο WoLF-PHC είναι

αρκετά μεγαλύτερος από τον μέσο αριθμό βημάτων που χρειάζεται χρησιμοποιώντας τον αλγόριθμο Q-Learning. Αυτό ήταν αναμενόμενο αφού όπως έχουμε αναφέρει προηγουμένως ο αλγόριθμος WoLF-PHC χρησιμοποιείται σε περιβάλλοντα ΕΜΠΠ και χρειάζεται εκπαίδευση μέχρι να προσαρμοστεί στο περιβάλλον ΕΜΕΠ. Το αξιοσημείωτο είναι ότι οι δύο αλγόριθμοι συγκλίνουν την ίδια χρονική στιγμή και περίπου στον ίδιο μέσο αριθμό βημάτων όταν εφαρμόζονται στο TP.



Σχήμα 4.15: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning με ε-greedy πολιτική και WoLF-PHC Ενός Πράκτορα για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3 και γ=0.95 στον αλγόριθμο Q-Learning, και ε=0.1, α=0.5,γ=0.95, δl=0.05 και δl=0.2 στον αλγόριθμο WoLF-PHC. Στα αρχικά επεισόδια ο μέσος αριθμός βημάτων του πράκτορα, χρησιμοποιώντας τον αλγόριθμο WoLF-PHC είναι αρκετά μεγαλύτερος από τον μέσο αριθμό βημάτων που χρειάζεται χρησιμοποιώντας τον αλγόριθμο Q-Learning. Στη συνέχεια οι δύο αλγόριθμοι συγκλίνουν την ίδια χρονική στιγμή και περίπου στον ίδιο μέσο αριθμό βημάτων.



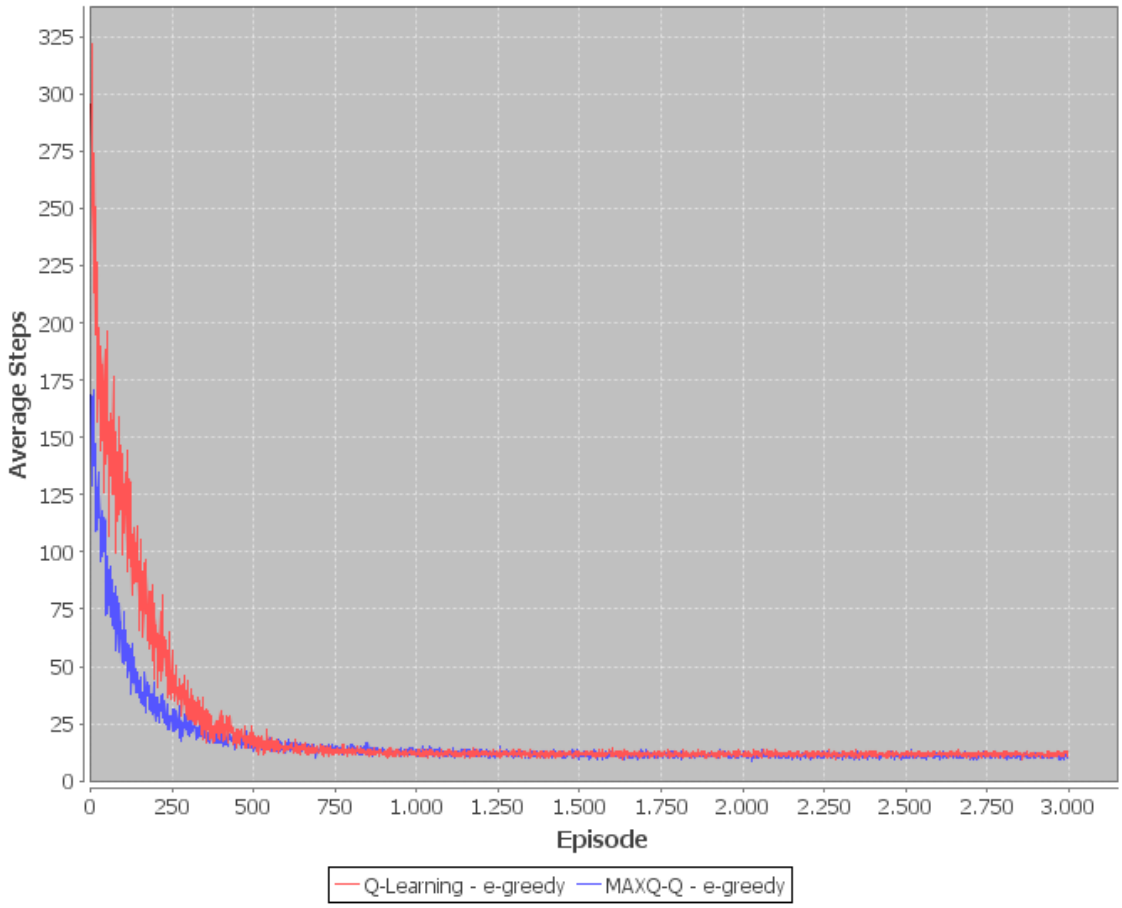
Σχήμα 4.16: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning με ε-greedy πολιτική και WoLF-PHC Ενός Πράκτορα για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3 και γ=0.95 στον αλγόριθμο Q-Learning, και ε=0.1, α=0.3,γ=0.95, δl=0.01 και δl=0.04 στον αλγόριθμο WoLF-PHC. Στα αρχικά επεισόδια ο μέσος αριθμός βημάτων του πράκτορα, χρησιμοποιώντας τον αλγόριθμο WoLF-PHC είναι αρκετά μεγαλύτερος από τον μέσο αριθμό βημάτων που χρειάζεται χρησιμοποιώντας τον αλγόριθμο Q-Learning. Στη συνέχεια οι δύο αλγόριθμοι συγκλίνουν, με τον αλγόριθμο Q-Learning να συγκλίνει σε μικρότερο αριθμό βημάτων από τον αλγόριθμο WoLF-PHC.

4.2.2.8 Σύγκριση αλγορίθμων Q-Learning και MAXQ-Q Ενός Πράκτορα

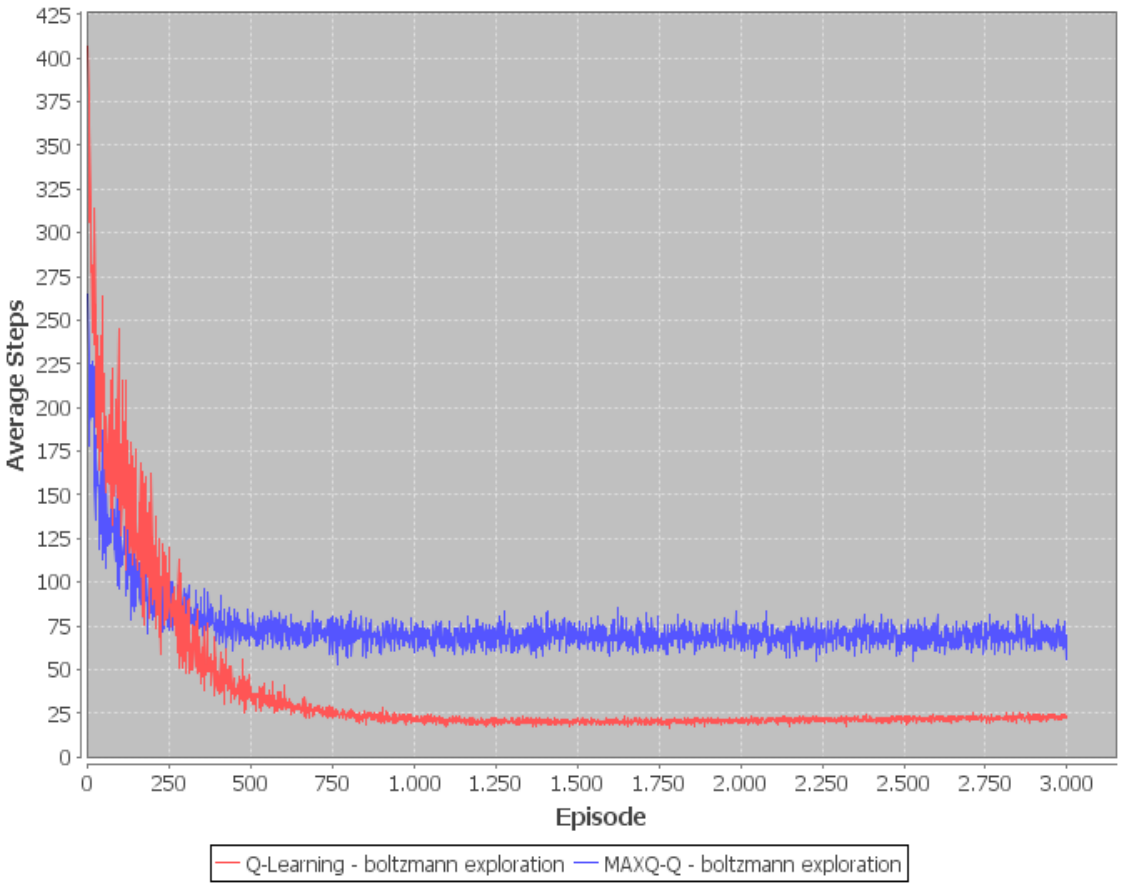
Ακολούθως συγκρίναμε τα αποτελέσματα που πήραμε από τους αλγόριθμους Q-Learning και MAXQ-Q. Το Σχήμα 4.17 δείχνει τα αποτελέσματα από τους δύο αλγόριθμους έχοντας χρησιμοποιήσει την ε-greedy πολιτική, και το Σχήμα 4.18 τα

αποτελέσματα χρησιμοποιώντας την πολιτική Boltzmann για το απλό TP, ενώ τα Σχήματα 4.19 και 4.20 δείχνουν τα αποτελέσματα χρησιμοποιώντας την ε-greedy και την Boltzmann πολιτική για το ETP αντίστοιχα. Παρατηρούμε ότι όταν χρησιμοποιείται η ε-greedy πολιτική οι δύο αλγόριθμοι συγκλίνουν στον ίδιο μέσο αριθμό βημάτων όμως η σύγκλιση στον αλγόριθμο MAXQ-Q επέρχεται πολύ πιο γρήγορα. Επίσης βλέπουμε ότι ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι αρκετά μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος Q-Learning παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-Q. Αυτό φαίνεται αρκετά καθαρά από την εφαρμογή τους στο ETP όπου το σύνολο καταστάσεων-ενεργειών είναι αρκετά μεγαλύτερο. Έτσι παρατηρούμε ότι με την χρησιμοποίηση του MAXQ-Q η μάθηση επιταχύνεται κάτι το οποίο ήταν αναμενόμενο αφού οι Ιεραρχικοί αλγόριθμοι χρησιμοποιούνται για να επιταχύνουν την μάθηση. Όταν χρησιμοποιείται η πολιτική Boltzmann παρατηρούμε ότι ο αλγόριθμος Q-Learning συγκλίνει σε μικρότερο μέσο αριθμό βημάτων από τον αλγόριθμο MAXQ-Q, αλλά και πάλι βλέπουμε ότι η μάθηση επιταχύνεται με την χρησιμοποίηση του αλγόριθμου MAXQ-Q αφού ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι αρκετά μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος Q-Learning παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-Q.



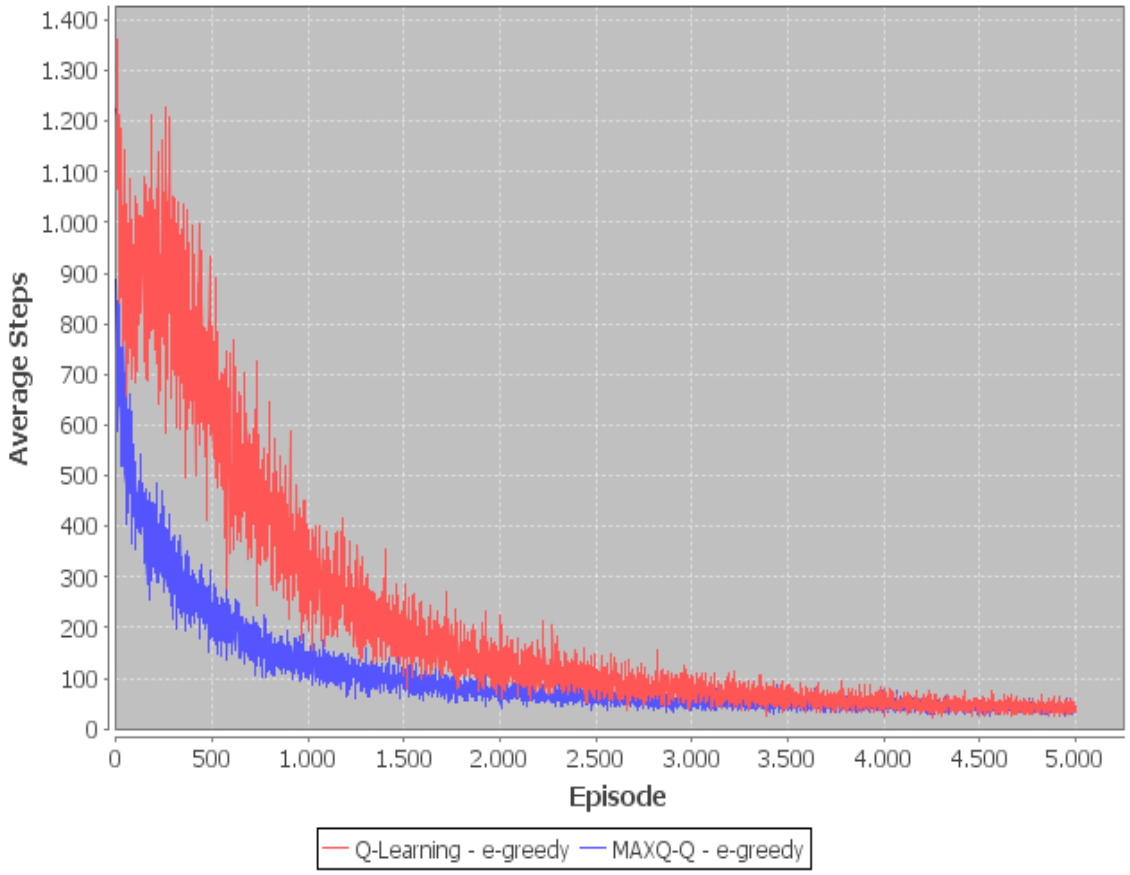
Σχήμα 4.17: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και MAXQ-Q Ενός Πράκτορα με ε-greedy πολιτική για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες τους σύμφωνα με την ε-greedy πολιτική. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3 και γ=0.95 για τον αλγόριθμο Q-Learning και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95 για τον αλγόριθμο MAXQ-Q. Οι δύο αλγόριθμοι συγκλίνουν στον ίδιο μέσο αριθμό βημάτων όμως η σύγκλιση στον αλγόριθμο MAXQ-Q επέρχεται πολύ πιο γρήγορα. Επίσης ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι αρκετά μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος Q-Learning παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-Q.



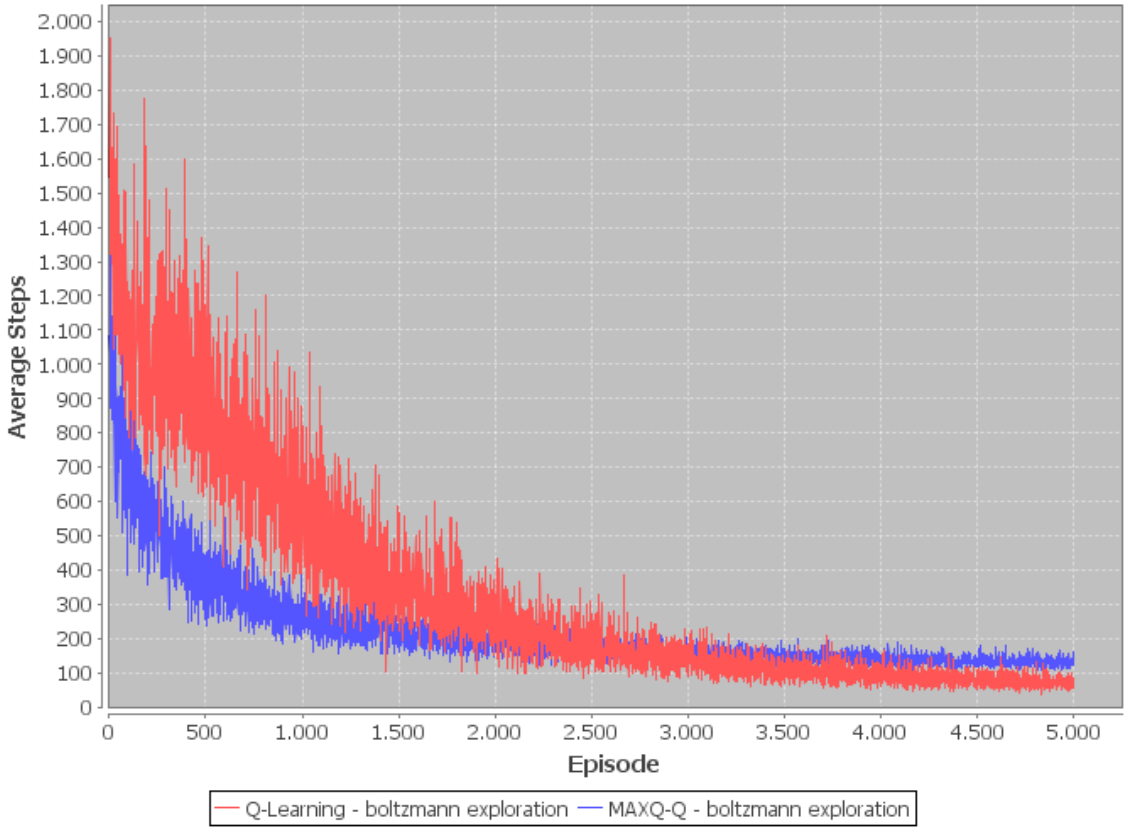
Σχήμα 4.18: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και MAXQ-Q Ενός Πράκτορα με πολιτική Boltzmann για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την πολιτική Boltzmann. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν για τον αλγόριθμο Q-Learning είναι α=0.3, γ=0.95, t=80, και cr=0.98. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν για τον αλγόριθμο MAXQ-Q είναι t=80, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95. Ο ρυθμός μείωσης της θερμοκρασίας για κάθε Max κόμβο είναι για τον MaxRoot=0.9996, τον MaxPut=0.9996, τον MaxGet=0.9939 και τον MaxNavigatee=0.9939. Ο αλγόριθμος Q-Learning συγκλίνει σε μικρότερο μέσο αριθμό βημάτων από τον αλγόριθμο MAXQ-Q. Όμως ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος Q-Learning παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-Q.



Σχήμα 4.19: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και MAXQ-Q Ενός Πράκτορα με ε-greedy πολιτική για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy πολιτική. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3 και γ=0.95 για τον αλγόριθμο Q-Learning και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95 για τον αλγόριθμο MAXQ-Q. Οι δύο αλγόριθμοι συγκλίνουν στον ίδιο μέσο αριθμό βημάτων όμως η σύγκλιση στον αλγόριθμο MAXQ-Q επέρχεται πολύ πιο γρήγορα. Επίσης ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι αρκετά μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος Q-Learning παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-Q.

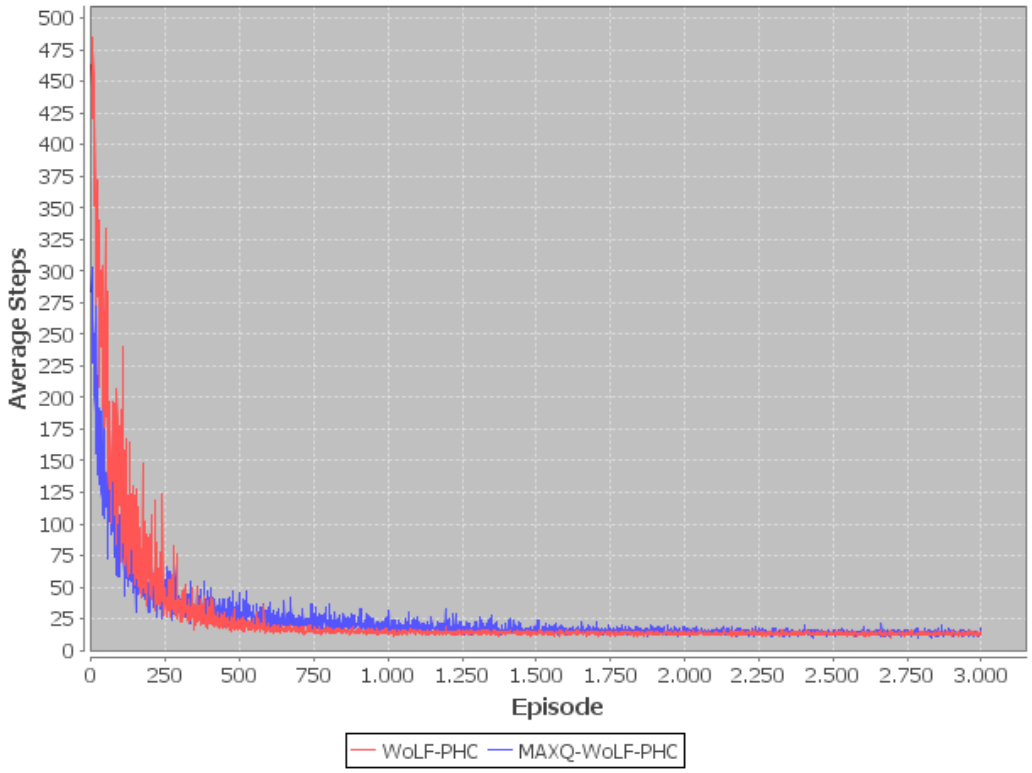


Σχήμα 4.20: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και MAXQ-Q Ενός Πράκτορα με πολιτική Boltzmann για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP
Σε αυτή την προσομοίωση ο πράκτορας επιλέγει τις ενέργειες του σύμφωνα με την πολιτική Boltzmann. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν για τον αλγόριθμο Q-Learning είναι α=0.3, γ=0.95, t=80, και cr=0.98. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν για τον αλγόριθμο MAXQ-Q είναι α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95. Ο ρυθμός μείωσης της θερμοκρασίας για κάθε Max κόμβο είναι για τον MaxRoot=0.9996, τον MaxPut=0.9996, τον MaxGet=0.9939 και τον MaxNavigatee=0.9939. Ο αλγόριθμος Q-Learning συγκλίνει σε ελαφρώς μικρότερο μέσο αριθμό βημάτων από τον αλγόριθμο MAXQ-Q. Όμως ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος Q-Learning παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-Q.

4.2.2.9 Σύγκριση αλγορίθμων WoLF-PHC και MAXQ-WoLF-PHC Ενός Πράκτορα

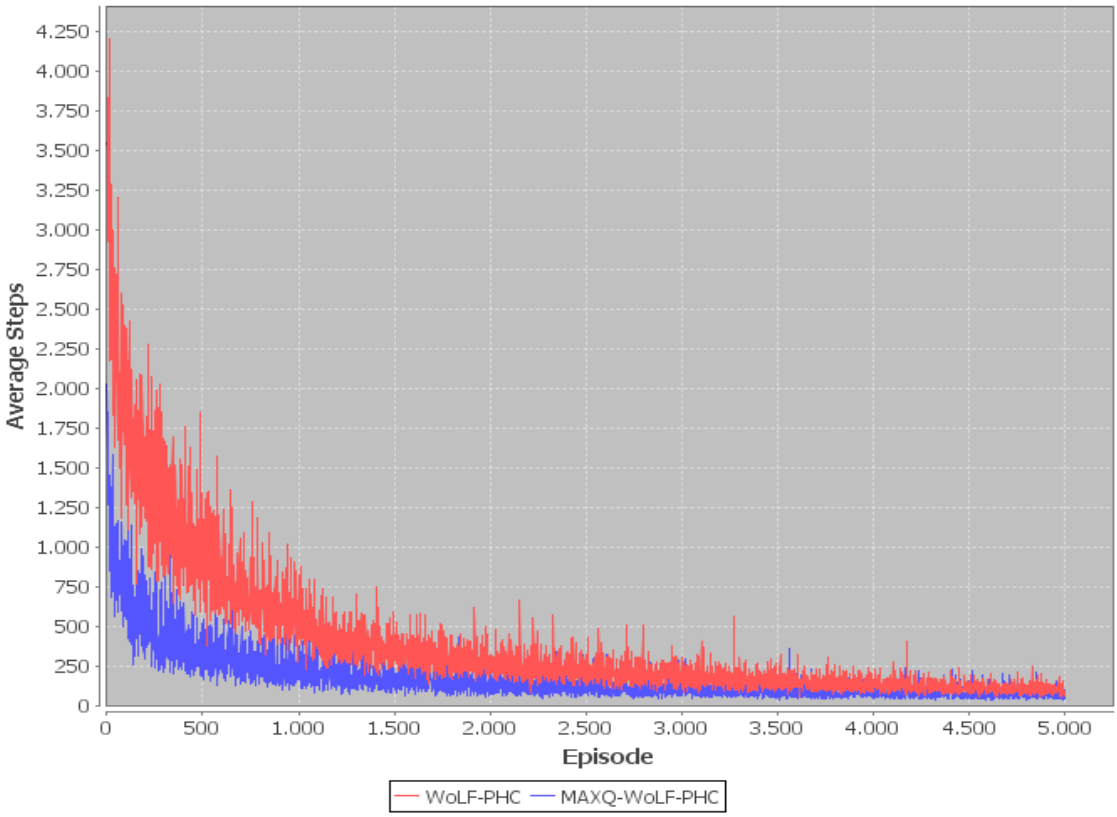
Σε αυτό το σημείο συγκρίναμε τα αποτελέσματα που πήραμε με την εφαρμογή των αλγορίθμων WoLF-PHC και MAXQ-WoLF-PHC. Το Σχήμα 4.21 δείχνει τα αποτελέσματα με την εφαρμογή τους στο απλό TP και το Σχήμα 4.22 με την εφαρμογή τους στο ETP. Παρατηρούμε ότι οι δύο αλγόριθμοι συγκλίνουν στον ίδιο αριθμό

βημάτων. Βλέπουμε όμως, ότι ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος WoLF-PHC παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-WoLF-PHC. Αυτό φαίνεται αρκετά καθαρά από την εφαρμογή τους στο ETP όπου το σύνολο καταστάσεων-ενεργειών είναι αρκετά μεγαλύτερο. Επίσης από την εφαρμογή τους στο ETP παρατηρούμε ότι η σύγκλιση στον αλγόριθμο MAXQ-WoLF-PHC επέρχεται πιο γρήγορα παρά στον αλγόριθμο WoLF-PHC. Έτσι παρατηρούμε ότι με την χρησιμοποίηση του MAXQ-WoLF-PHC η μάθηση επιταχύνεται κάτι το οποίο ήταν αναμενόμενο αφού οι Ιεραρχικοί αλγόριθμοι χρησιμοποιούνται για να επιταχύνουν την μάθηση.



Σχήμα 4.21: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους WoLF-PHC και MAXQ-WoLF-PHC Ενός Πράκτορα για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.5,γ=0.95, δw=0.1 και δl=0.4 στον αλγόριθμο WoLF-PHC, και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.02 και δl=0.5 στον αλγόριθμο MAXQ-WoLF-PHC. Οι δύο αλγόριθμοι συγκλίνουν στον ίδιο μέσο αριθμό βημάτων όμως η σύγκλιση στον αλγόριθμο MAXQ-WoLF-PHC επέρχεται πολύ πιο γρήγορα. Επίσης ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι αρκετά μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος WoLF-PHC παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-WoLF-PHC.



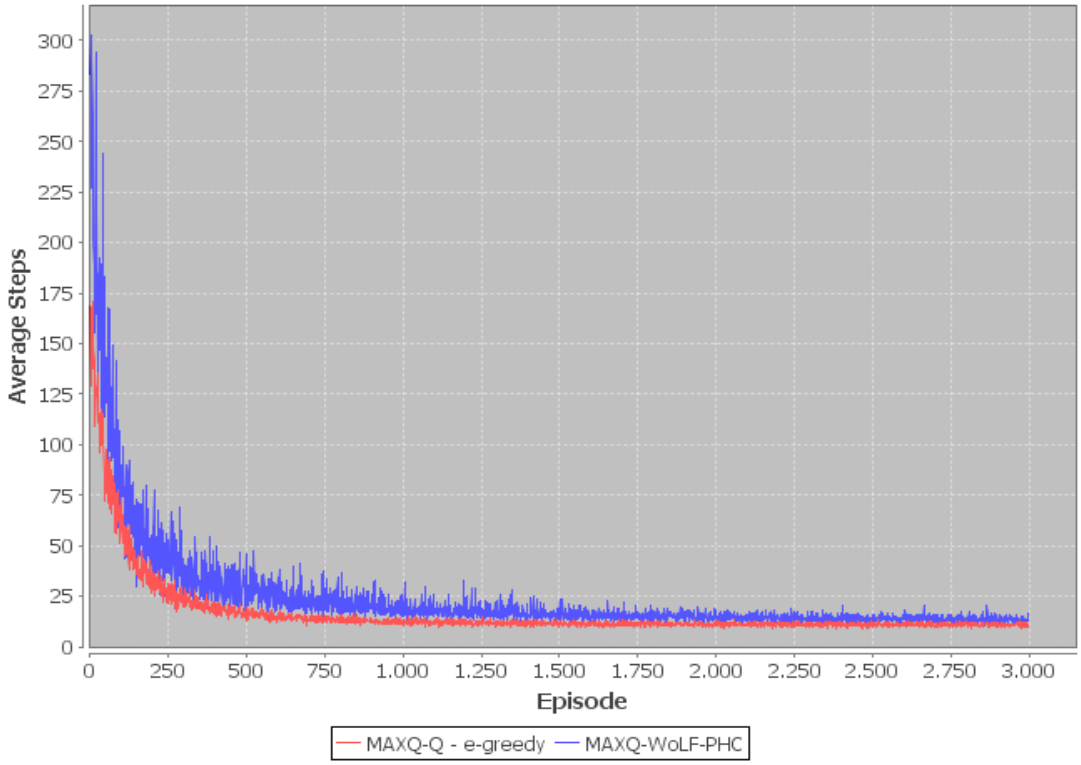
Σχήμα 4.22: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους WoLF-PHC και MAXQ-WoLF-PHC Ενός Πράκτορα για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3,γ=0.95, δw=0.01 και δl=0.04 στον στον αλγόριθμο WoLF-PHC, και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.1 και δl=0.4 στον αλγόριθμο MAXQ-WoLF-PHC. Οι δύο αλγόριθμοι συγκλίνουν στον ίδιο μέσο αριθμό βημάτων όμως η σύγκλιση στον αλγόριθμο MAXQ-WoLF-PHC επέρχεται πιο γρήγορα. Επίσης ο αριθμός των βημάτων που χρειάζεται ο πράκτορας στα αρχικά επεισόδια, πριν την σύγκλιση, είναι μεγαλύτερος όταν χρησιμοποιείται ο αλγόριθμος WoLF-PHC παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-WoLF-PHC.

4.2.2.10 Σύγκριση αλγορίθμων MAXQ-Q και MAXQ-WoLF-PHC Ενός Πράκτορα

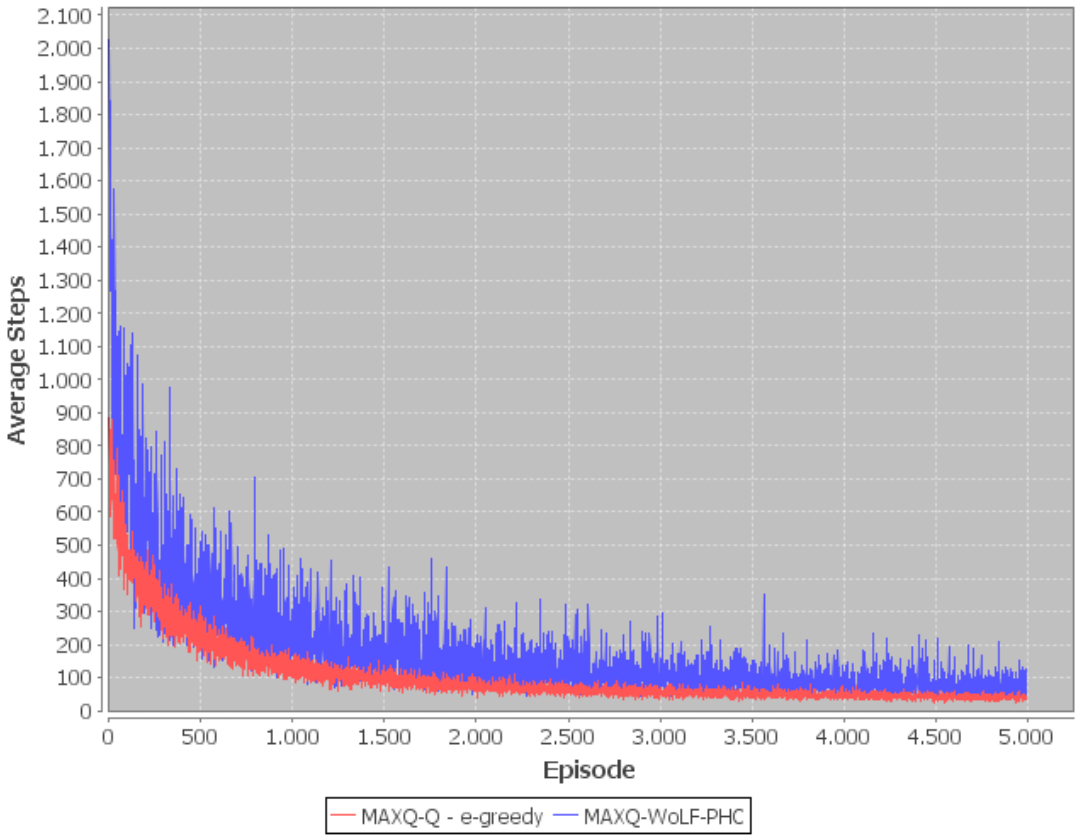
Έπειτα συγκρίναμε τα αποτελέσματα που πήραμε με την εφαρμογή των αλγορίθμων MAXQ-Q χρησιμοποιώντας την ε-greedy πολιτική και MAXQ-WoLF-PHC. Το Σχήμα 4.23 δείχνει τα αποτελέσματα με την εφαρμογή τους στο απλό TP και το Σχήμα 4.24 με την εφαρμογή τους στο ETP. Παρατηρούμε ότι στα αρχικά επεισόδια ο μέσος αριθμός

βημάτων του πράκτορα, χρησιμοποιώντας τον αλγόριθμο MAXQ-WoLF-PHC είναι αρκετά μεγαλύτερος από τον μέσο αριθμό βημάτων που χρειάζεται χρησιμοποιώντας τον αλγόριθμο MAXQ-Q. Αυτό ήταν αναμενόμενο αφού ο αλγόριθμος MAXQ-WoLF-PHC χρησιμοποιείται σε περιβάλλοντα ΕΜΠΠ και χρειάζεται εκπαίδευση μέχρι να προσαρμοστεί στο περιβάλλον ΕΜΕΠ. Επίσης κοιτάζοντας τις γραφικές παραστάσεις που αφορούν την προσομοίωση των δύο αλγορίθμων στο ETP παρατηρούμε ότι η γραφική παράσταση που αφορά τον αλγόριθμο MAXQ-Q είναι πιο απαλή από την γραφική παράσταση που αφορά τον αλγόριθμο MAXQ-WoLF-PHC. Έτσι καταλήγουμε στο συμπέρασμα ότι στο πρόβλημα του ETP ενός πράκτορα, όπου υπάρχει μεγάλο συνόλο καταστάσεων-ενεργειών είναι καλύτερα να χρησιμοποιείται ο αλγόριθμος MAXQ-Q, χρησιμοποιώντας την ε-greedy πολιτική.



Σχήμα 4.23: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους MAXQ-Q με ε-greedy πολιτική και MAXQ-WoLF-PHC Ενός Πράκτορα για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση ο πράκτορας στον αλγόριθμο MAXQ-Q επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy πολιτική. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95 για τον αλγόριθμο MAXQ-Q και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.05 και δl=0.2 στον αλγόριθμο MAXQ-WoLF-PHC. Στα αρχικά επεισόδια ο μέσος αριθμός βημάτων του πράκτορα, χρησιμοποιώντας τον αλγόριθμο MAXQ-WoLF-PHC είναι μεγαλύτερος από τον μέσο αριθμό βημάτων που χρειάζεται χρησιμοποιώντας τον αλγόριθμο MAXQ-Q. Στη συνέχεια οι δύο αλγόριθμοι συγκλίνουν με τον αλγόριθμο MAXQ-Q να συγκλίνει σε ελαφρώς μικρότερο αριθμό βημάτων από τον αλγόριθμο MAXQ-WoLF-PHC.



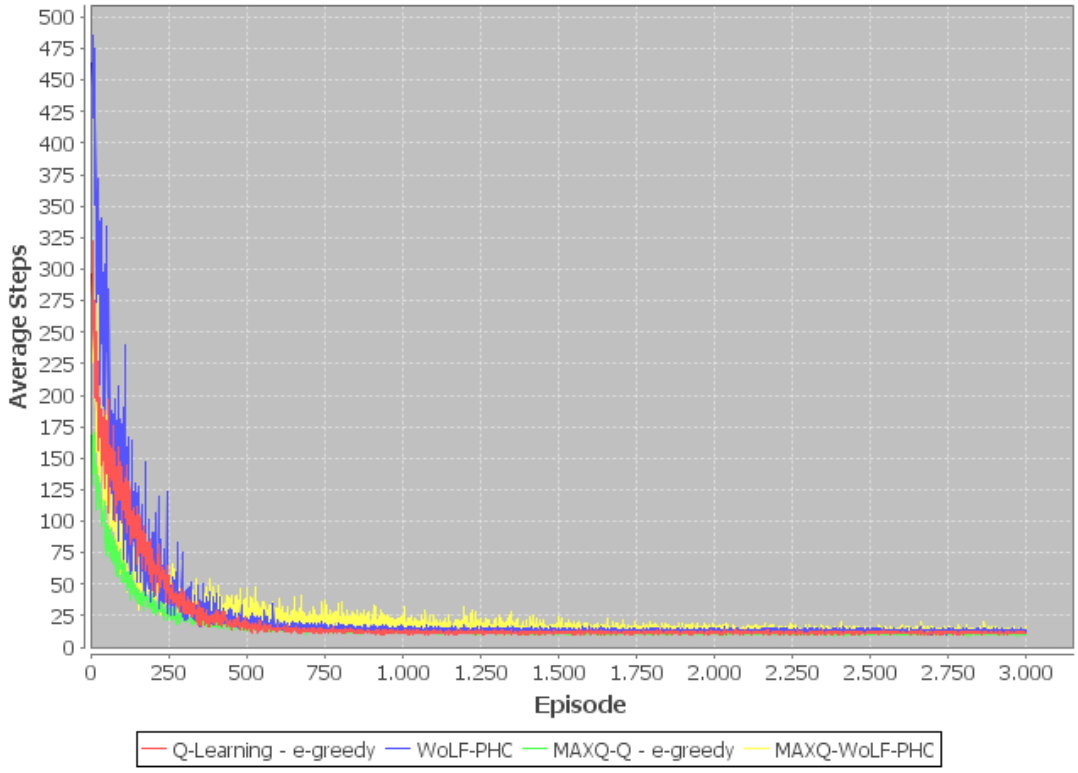
Σχήμα 4.24: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους MAXQ-Q με ε-greedy πολιτική και MAXQ-WoLF-PHC Ενός Πράκτορα για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση ο πράκτορας στον αλγόριθμο MAXQ-Q επιλέγει τις ενέργειες του σύμφωνα με την ε-greedy πολιτική. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95 για τον αλγόριθμο MAXQ-Q και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.1 και δl=0.4 στον αλγόριθμο MAXQ-WoLF-PHC. Στα αρχικά επεισόδια ο μέσος αριθμός βημάτων του πράκτορα, χρησιμοποιώντας τον αλγόριθμο MAXQ-WoLF-PHC είναι μεγαλύτερος από τον μέσο αριθμό βημάτων που χρειάζεται χρησιμοποιώντας τον αλγόριθμο MAXQ-Q. Στη συνέχεια οι δύο αλγόριθμοι συγκλίνουν με τον αλγόριθμο MAXQ-Q να συγκλίνει σε μικρότερο αριθμό βημάτων από τον αλγόριθμο MAXQ-WoLF-PHC. Επίσης η γραφική παράσταση που αφορά την εφαρμογή του αλγόριθμου MAXQ-Q είναι πιο απαλή από την γραφική παράσταση που αφορά την εφαρμογή του αλγόριθμου MAXQ-WoLF-PHC.

4.2.2.11 Σύγκριση αλγορίθμων Q-Learning, WoLF-PHC, MAXQ-Q και MAXQ-WoLF-PHC Ενός Πράκτορα

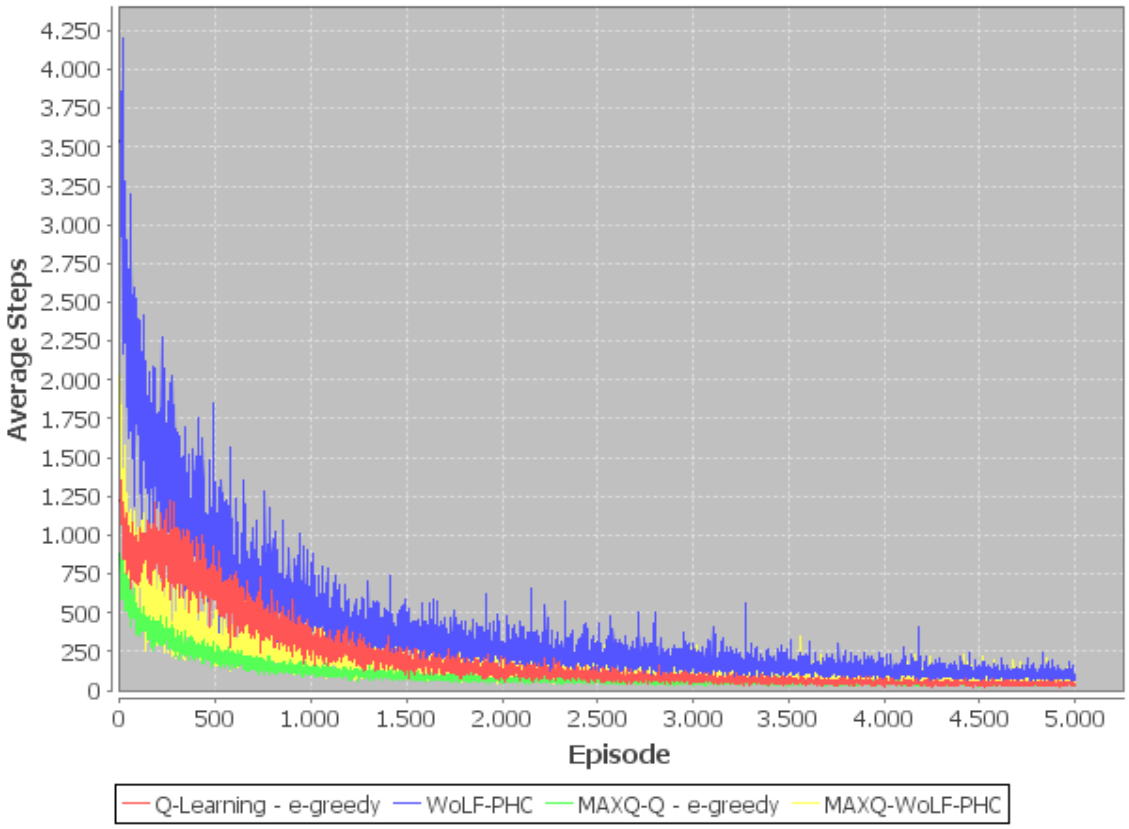
Τέλος συγκρίναμε τα αποτελέσματα που πήραμε από τους αλγόριθμους Q-Learning, MAXQ-Q, WoLF-PHC και MAXQ-WoLF-PHC. Πήραμε τα αποτελέσματα των αλγόριθμων Q-Learning και MAXQ-Q χρησιμοποιώντας την ε-greedy πολιτική, αφού

όπως είδαμε, χρησιμοποιώντας αυτή την πολιτική είχαν καλύτερα αποτελέσματα παρά από όταν χρησιμοποιούσαν την πολιτική Boltzmann. Το Σχήμα 4.25 δείχνει τα αποτελέσματα με την εφαρμογή τους στο απλό TP και το Σχήμα 4.26 με την εφαρμογή τους στο ETP. Παρατηρούμε ότι οι δύο Ιεραρχικοί αλγόριθμοι, MAXQ-Q και MAXQ-WoLF-PHC, έχουν καλύτερη απόδοση από τους άλλους δύο αλγόριθμους, αφού η μάθηση γίνεται πιο γρήγορα όταν χρησιμοποιούνται αυτοί οι αλγόριθμοι, κάτι το οποίο ήταν αναμενόμενο αφού σκοπός των Ιεραρχικών μεθόδων είναι να επιταχύνουν την μάθηση. Επίσης βλέπουμε ότι ο αλγόριθμος MAXQ-Q έχει την καλύτερη απόδοση από όλους τους αλγόριθμους.



Σχήμα 4.25: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning χρησιμοποιώντας την ε-greedy πολιτική, WoLF-PHC, MAXQ-Q χρησιμοποιώντας την ε-greedy πολιτική και MAXQ-WoLF-PHC Ενός Πράκτορα για 50 δοκιμές και 3000 επεισόδια ανά δοκιμή στο TP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3 και γ=0.95 στον αλγόριθμο Q-Learning, και ε=0.1, α=0.5,γ=0.95, δl=0,05 και δl=0.2 στον αλγόριθμο WoLF-PHC. Για τον αλγόριθμο MAXQ-Q χρησιμοποιήθηκαν οι τιμές ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95 και για τον αλγόριθμο MAXQ-WoLF-PHC οι τιμές ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.1 και δl=0.4. Στα αρχικά επεισόδια, πριν την σύγκλιση ο μέσος αριθμός βημάτων που χρειάζονται οι δύο Ιεραρχικοί αλγόριθμοι MAXQ-Q και MAXQ-WoLF-PHC είναι μικρότερος από τον μέσο αριθμό βημάτων που χρειάζονται οι άλλοι δύο αλγόριθμοι, με τον αλγόριθμο MAXQ-Q να χρειάζεται τον μικρότερο μέσο αριθμό βημάτων από όλους τους αλγόριθμους. Στη συνέχεια συγκλίνουν στον ίδιο μέσο αριθμό βημάτων, με την σύγκλιση να επέρχεται πιο γρήγορα στον αλγόριθμο MAXQ-Q.



Σχήμα 4.26: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning χρησιμοποιώντας την ε-greedy πολιτική, WoLF-PHC, MAXQ-Q χρησιμοποιώντας την ε-greedy πολιτική και MAXQ-WoLF-PHC Ενός Πράκτορα για 30 δοκιμές και 5000 επεισόδια ανά δοκιμή στο ETP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3 και γ=0.95 στον αλγόριθμο Q-Learning, και ε=0.1, α=0.3,γ=0.95, δl=0.01 και δl=0.04 στον αλγόριθμο WoLF-PHC. Για τον αλγόριθμο MAXQ-Q χρησιμοποιήθηκαν οι τιμές ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου και γ=0.95 και για τον αλγόριθμο MAXQ-WoLF-PHC οι τιμές ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.1 και δl=0.4. Στα αρχικά επεισόδια, πριν την σύγκλιση ο μέσος αριθμός βημάτων που χρειάζονται οι δύο Ιεραρχικοί αλγόριθμοι MAXQ-Q και MAXQ-WoLF-PHC είναι μικρότερος από τον μέσο αριθμό βημάτων που χρειάζονται οι άλλοι δύο αλγόριθμοι, με τον αλγόριθμο MAXQ-Q να χρειάζεται τον μικρότερο μέσο αριθμό βημάτων από όλους τους αλγόριθμους. Στη συνέχεια συγκλίνουν περίπου στον ίδιο μέσο αριθμό βημάτων, με την σύγκλιση να επέρχεται πιο γρήγορα στους αλγόριθμος MAXQ-Q και MAXQ-WoLF-PHC.

4.3 Ενισχυτική Μάθηση Πολλαπλών Πρακτόρων

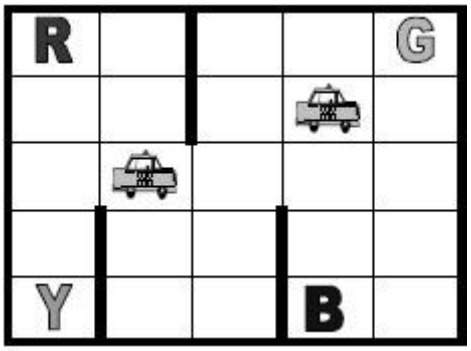
4.3.1 Εισαγωγή

Αρχικά υλοποιήθηκε ο εκτός-πολιτικής αλγόριθμος Q-Learning για να δούμε πως συμπεριφέρεται σε περιβάλλοντα πολλαπλών πρακτόρων. Στη συνέχεια δοκιμάσαμε τους αλγόριθμους PHC και WoLF-PHC, ο οποίοι είναι αλγόριθμοι που εφαρμόζονται σε περιβάλλοντα πολλαπλών πρακτόρων. Έπειτα ασχοληθήκαμε με αλγόριθμους IEM. Από τα αποτελέσματα του αλγόριθμου Q-Learning και από τη σύγκριση των αλγορίθμων Q-Learning και WoLF-PHC καταλήξαμε στο συμπέρασμα ότι ο αλγόριθμος MAXQ-Q δεν είναι τόσο ενδιαφέρον όσο οι αλγόριθμοι MAXQ-PHC και MAXQ-WoLF-PHC σε περιβάλλοντα πολλαπλών πρακτόρων. Έτσι συνεχίσαμε με την εφαρμογή των αλγόριθμων MAXQ-PHC και MAXQ-WoLF-PHC.

Με την εισαγωγή περισσότερων από ένα πρακτόρων το σύνολο καταστάσεων-ενεργειών που δημιουργείται αυξάνεται δραματικά, έτσι αποφασίσαμε να εφαρμόσουμε τους πιο πάνω αλγόριθμους στο απλό TP για πολλαπλούς πράκτορες το οποίο θα εξηγηθεί στη συνέχεια και όχι στο ETP αφού θα είχαμε ακόμα πιο μεγάλο σύνολο καταστάσεων-ενεργειών.

4.3.2 Taxi Problem για Πολλαπλούς Πράκτορες

Το Taxi Problem για Πολλαπλούς Πράκτορες (MATP-Multi Agent Taxi Problem) είναι μια επέκταση του απλού TP, την οποία έχουμε δημιουργήσει για την εφαρμογή των αλγορίθμων ΕΜΠΠ, και εξ όσον γνωρίζουμε δεν υπάρχει στην βιβλιογραφία, με τα συγκεκριμένα κριτήρια και υποθέσεις που χρησιμοποιούμε και εξηγούνται στην συνέχεια. Στο MATP έχουμε ένα 5x5 πλαίσιο όπως φαίνεται στο Σχήμα 4.27.



Σχήμα 4.27: Το πλαίσιο του προβλήματος του ταξί για Πολλαπλούς Πράκτορες

όπου υπάρχουν τέσσερεις ειδικά σχεδιασμένες τοποθεσίες οι οποίες απεικονίζονται ως R(ed), B(lue), G(reen) και Y(ellow), δύο ή περισσότερα ταξί και δύο ή περισσότεροι επιβάτες. Οι επιβάτες μπορούν να βρίσκονται σε μια από αυτές τις τέσσερεις τοποθεσίες είτε μέσα στο ταξί.

Σκοπός αυτού του προβλήματος είναι να μάθει, το κάθε ταξί (δηλαδή ο πράκτορας), που βρίσκεται ένας επιβάτης, να μετακινηθεί στην τοποθεσία αυτή και να τον παραλάβει, και στη συνέχεια να τον μεταφέρει στον προορισμό που επιθυμεί με όσο το δυνατό λιγότερα βήματα.

Τα ταξί ξεκινούν από μια τυχαία θέση σε αυτό το πλαίσιο, καθώς επίσης και η αρχική τοποθεσία των επιβατών όπως και ο τελικός προορισμός τους, που καθορίζονται τυχαία σε μια από αυτές τις τέσσερις ειδικά σχεδιασμένες τοποθεσίες.

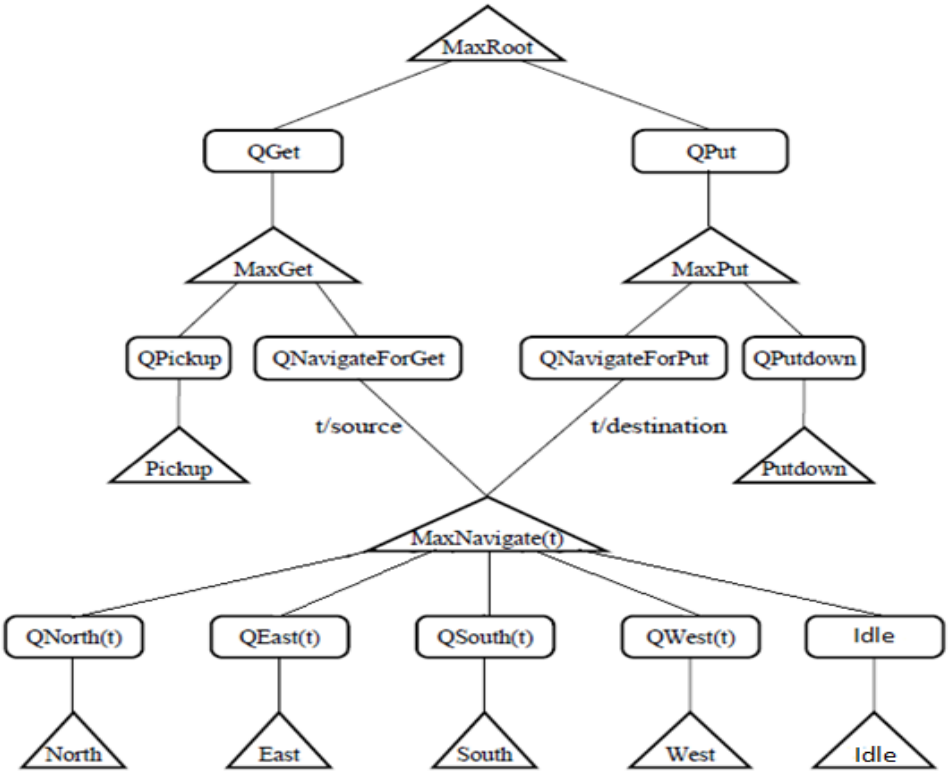
Σε κάθε βήμα, το ταξί μπορεί να εκτελέσει μια ενέργεια από τις ακόλουθες επτά: να κινηθεί α) ένα τετράγωνο βόρια, β) ένα τετράγωνο νότια, γ) ένα τετράγωνο ανατολικά, και δ) ένα τετράγωνο δυτικά, ε) να παραλάβει τον επιβάτη, ζ) να αφήσει τον επιβάτη και η) να μην κάνει καμία κίνηση. Αν κτυπήσει σε τοίχο ή αν προσπαθήσει να βγει εκτός των ορίων του πλαισίου η θέση του δεν αλλάζει. Το κάθε ταξί μπορεί να μεταφέρει μόνο 1 επιβάτη κάθε φορά. Για μια επιτυχημένη μεταφορά παίρνει σαν αμοιβή 20. Αν το ταξί εκτελέσει την ενέργεια «παρέλαβε τον επιβάτη» σε τοποθεσία όπου δεν βρίσκεται ο επιβάτης, ή αν ήδη μεταφέρεται ο επιβάτης τότε παίρνει σαν

αμοιβή -10. Επίσης αν εκτελέσει την ενέργεια «άφησε τον επιβάτη» σε λανθασμένο προορισμό, ή χωρίς να μεταφέρει κάποιο επιβάτη και πάλι η αμοιβή που παίρνει είναι -10. Για οποιαδήποτε άλλη ενέργεια η αμοιβή είναι -1. Αν δύο ή περισσότερα ταξί συγκρουστούν τότε παίρνουν μια επιπλέον αμοιβή -20. Σε αυτή την περίπτωση επιλέγεται τυχαία πιο ταξί θα παραμείνει στη συγκεκριμένη τοποθεσία και τα υπόλοιπα μεταφέρονται στην προηγούμενη τοποθεσία που βρίσκονταν, πριν από αυτή τους την κίνηση. Στην περίπτωση όπου ένα από τα ταξί που συγκρούστηκαν έχει επιλέξει μία από τις κινήσεις (ε), (ζ), ή (η), τότε σίγουρα αυτό το ταξί θα παραμείνει στην τοποθεσία της σύγκρουσης και τα υπόλοιπα θα επιστρέψουν στην προηγούμενη τους τοποθεσία. Η κάθε δοκιμή τελειώνει όταν γίνει ένας αριθμός επιτυχημένων μεταφορών, όπου αυτός ο αριθμός καθορίζεται από τον χρήστη.

Το MATP μπορεί να διατυπωθεί ως ένα παιχνίδι Markov με τρεις μεταβλητές κατάστασης: την τοποθεσία που βρίσκεται το κάθε ταξί (0-24), την τοποθεσία του κάθε επιβάτη (0-4, όπου το 0 σημαίνει ότι ο επιβάτης βρίσκεται μέσα στο ταξί) και την τοποθεσία του προορισμού για τον κάθε επιβάτη(1-4). Άρα σύνολο θα έχουμε 25^{n} x 5^{k} x 4^{k} καταστάσεις, όπου n είναι ο αριθμός των ταξί και k ο αριθμός των επιβατών.

4.3.3 MAXQ διάσπαση για το πρόβλημα του ταξί για Πολλαπλούς Πράκτορες

Η MAXQ διάσπαση για το MATP είναι μια επέκταση της MAXQ διάσπασης για το απλό TP η οποία εξηγήθηκε στο κεφάλαιο 2.8. Σε αυτή την επέκταση έχουμε προσθέσει την ενέργεια Idle, η οποία αντιπροσωπεύει την ενέργεια «μην κάνεις καμία κίνηση», όπου έχει την επιλογή να επιλέξει ο πράκτορας στο MATP. Στο Σχήμα 4.28 φαίνεται ο εκτεταμένος γράφος για το MATP.



Σχήμα 4.28: Εκτεταμένος MAXQ γράφος για το πρόβλημα του ταξί για Πολλαπλούς Πράκτορες

όπου υπάρχουν Max κόμβοι και Q κόμβοι. Οι Max κόμβοι χωρίς παιδιά υποδηλώνουν αρχέγονες ενέργειες και οι Max κόμβοι με παιδιά αναπαριστούν υποδομές (sub-task). Κάθε Q κόμβος υποδηλώνει μια ενέργεια η οποία μπορεί να εκτελεστεί για να επιτύχει η υποδομή του γονέα του.

4.3.4 Αποτελέσματα εφαρμογής των αλγόριθμων στο Taxi Problem για Πολλαπλούς Πράκτορες

Όπως έχω προαναφέρει το MATP είναι επεισοδιακό. Σε κάθε επεισόδιο οι πράκτορες κάνουν τα βήματα τους παράλληλα., έτσι μετράμε τον αριθμό των βημάτων που κάνουν οι πράκτορες μέχρι να γίνει ένας αριθμός επιτυχημένων μεταφορών. Αποφασίσαμε να κάνουμε ένα αριθμό από δοκιμές για τον κάθε αλγόριθμο, όπου σε κάθε δοκιμή ο αλγόριθμος θα τρέχει για ένα αριθμό επεισοδίων. Στο τέλος παίρνουμε τον μέσο όρο των βημάτων που κάνει ο πράκτορας, ούτως ώστε να εξαχθούν καλύτερα και πιο αντιπροσωπευτικά αποτελέσματα.

Επιλέχθηκε να χρησιμοποιήσουμε 2 πράκτορες και 2 επιβάτες, αφού το σύνολο καταστάσεων-ενεργειών μεγαλώνει πάρα πολύ αν χρησιμοποιήσουμε μεγαλύτερο αριθμό, κάτι το οποίο θα μας κοστίσει σε χρόνο και χώρο. Σε χρόνο επειδή θα χρειάζονται ακόμα περισσότερα επεισόδια μέχρι να επέλθει η μάθηση και σε χώρο γιατί θα χρειαζόμαστε ακόμα περισσότερη μνήμη. Με αυτά τα δεδομένα έχουμε 25^2 x 5^2 x $4^2 = 250000$ καταστάσεις που είναι ήδη ένας αρκετά μεγάλος αριθμός καταστάσεων. Ο αριθμός των επεισοδίων επιλέχθηκε να είναι 1000000, αφού λόγο του μεγάλου συνόλου καταστάσεων-ενεργειών χρειάζονται αρκετά επεισόδια μέχρι να εκπαιδευτούν οι πράκτορες. Ο αριθμός των δοκιμών επιλέχθηκε να είναι 10, αφού λόγο του μεγάλου συνόλου καταστάσεων-ενεργειών και του μεγάλου αριθμού επεισοδίων χρειάζεται αρκετά μεγάλο χρονικό διάστημα μέχρι να τελειώσει η κάθε δοκιμή. Επίσης επιλέξαμε, το κάθε επεισόδιο να τελειώνει μετά από 2 επιτυχημένες μεταφορές.

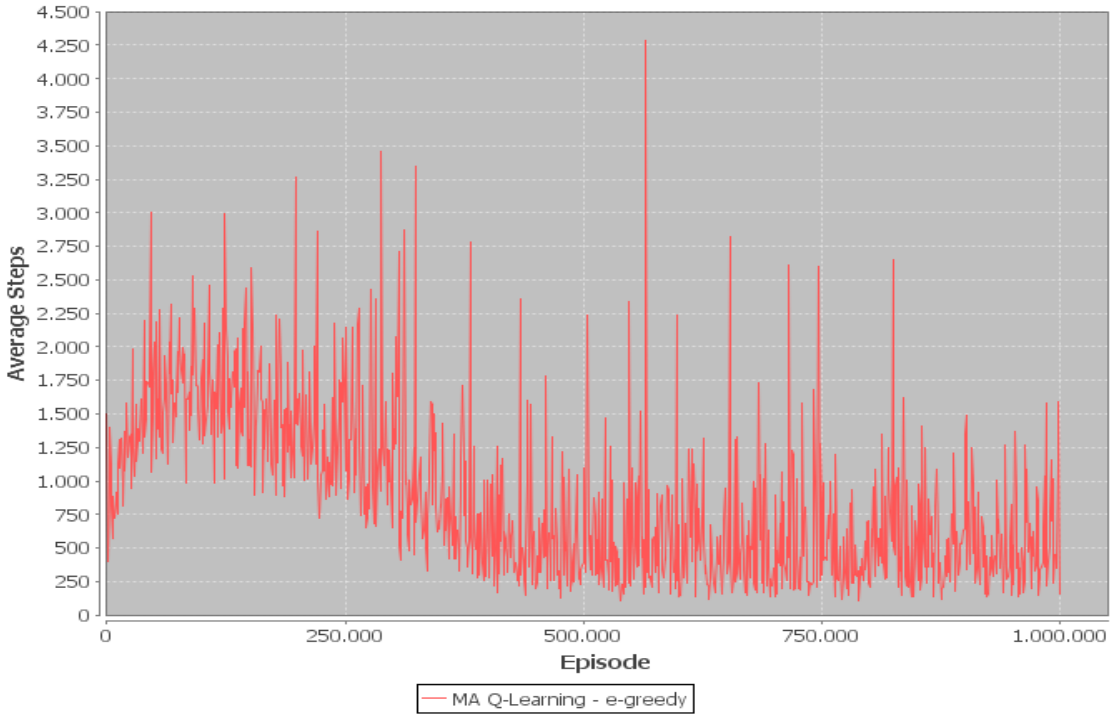
Οι αμοιβές για κάθε κίνηση του πράκτορα έχουν καθοριστεί σύμφωνα με το κεφάλαιο 4.2.2 όπου περιγράφεται το MATP. Οι ψευδοαμοιβές, για τους αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC μετά από κάποιες δοκιμές αποφασίστηκε να είναι 0 για κάθε τερματική κατάσταση και -100 για κάθε άλλη κατάσταση. Επίσης ακολουθείται ο συμβολισμός του κεφαλαίου 4.1.2 για τις παραμέτρους.

Ακολουθούν τα αποτελέσματα από την εφαρμογή αυτών τον αλγορίθμων στο MATP.

4.3.4.1 Εφαρμογή αλγόριθμου Q-Learning Πολλαπλών Πρακτόρων

Στο MATP αποφασίσαμε να δοκιμάσουμε τον αλγόριθμο Q-Learning Πολλαπλών Πρακτόρων χρησιμοποιώντας την ε-greedy πολιτική. Μετά από κάποιες δοκιμές καθορίσαμε τις τιμές ε=0.1, α=0.3 και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές.

Το Σχήμα 4.29 δείχνει το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκε ο πράκτορας για να επιτύχει τον στόχο του στο MATP. Παρατηρούμε ότι, αρχικά οι πράκτορες δυσκολεύονται πολύ να επιτύχουν τον στόχο τους, αφού ο μέσος αριθμός βημάτων που χρειάζονται αυξάνεται. Στη συνέχεια βλέπουμε να βελτιώνεται η απόδοση τους, χωρίς όμως αρκετά ικανοποιητικά αποτελέσματα, αφού όπως φαίνεται στην γραφική παράσταση υπάρχουν αρκετά σκαμπανεβάσματα, κάτι το οποίο σημαίνει ότι δεν έχει γίνει αρκετά καλή εκπαίδευση. Αυτό ήταν αναμενόμενο αφού ο αλγόριθμος Q-Learning είναι αλγόριθμος που αφορά προβλήματα ΕΜΕΠ. Έτσι συμπεραίνουμε ότι η εφαρμογή του αλγόριθμου Q-Learning στο πρόβλημα του MATP δεν είναι ιδιαίτερα χρήσιμη.



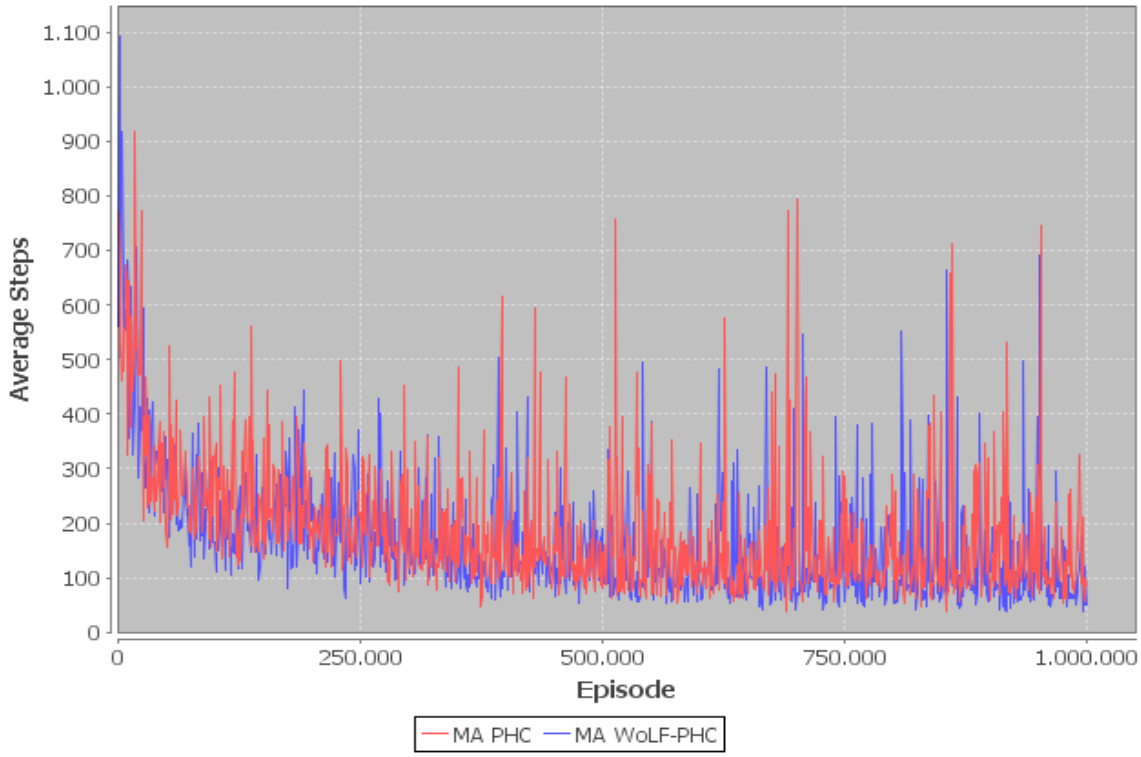
Σχήμα 4.29: Μέσος όρος βημάτων χρησιμοποιώντας τον αλγόριθμο Q-Learning Πολλαπλών Πρακτόρων με ε-greedy και Boltzmann πολιτική για 10 δοκιμές και 1000000 επεισόδια ανά δοκιμή στο MATP

Σε αυτή την προσομοίωση οι πράκτορες επιλέγουν τις ενέργειες τους σύμφωνα με την ε-greedy. Οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1 , α=0.3 και γ=0.95. Οι πράκτορες δεν εκπαιδεύονται με ιδιαίτερη επιτυχία αφού αρχικά ο μέσος αριθμός βημάτων που χρειάζονται για να επιτύχουν τον στόχο τους αυξάνεται, αλλά ακόμα και στη συνέχεια, όπου ο μέσος αριθμός βημάτων που χρειάζονται μειώνεται, υπάρχουν αρκετά σκαμπανεβάσματα στην γραφική παράσταση, κάτι το οποίο σημαίνει ότι δεν έχει γίνει αρκετά καλή εκπαίδευση.

4.3.4.2 Εφαρμογή αλγορίθμων PHC και WoLF-PHC για Πολλαπλούς Πράκτορες

Στη συνέχεια δοκιμάσαμε τους αλγόριθμους PHC και WoLF-PHC για πολλαπλούς πράκτορες. Μετά από κάποιες δοκιμές αποφασίστηκε όπως χρησιμοποιηθούν οι ακόλουθες τιμές για τις παραμέτρους: ε=0.1 και γ=0.95, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές, και στους δύο αλγόριθμους. Επίσης, στον αλγόριθμο PHC καθορίσαμε α=0.3 και δ=0.2, ενώ στον αλγόριθμο WoLF-PHC καθορίσαμε α=0.3, δw=0.05 και δl=0.2, αφού σύμφωνα με τον αλγόριθμο πρέπει δl> δw.

Το Σχήμα 4.30 δείχνει το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκαν οι πράκτορες για να επιτύχουν τον στόχο τους, στο MATP. Παρατηρούμε ότι οι πράκτορες εκπαιδεύονται με επιτυχία, έχοντας παρόμοια συμπεριφορά, και στους δύο αλγόριθμους.



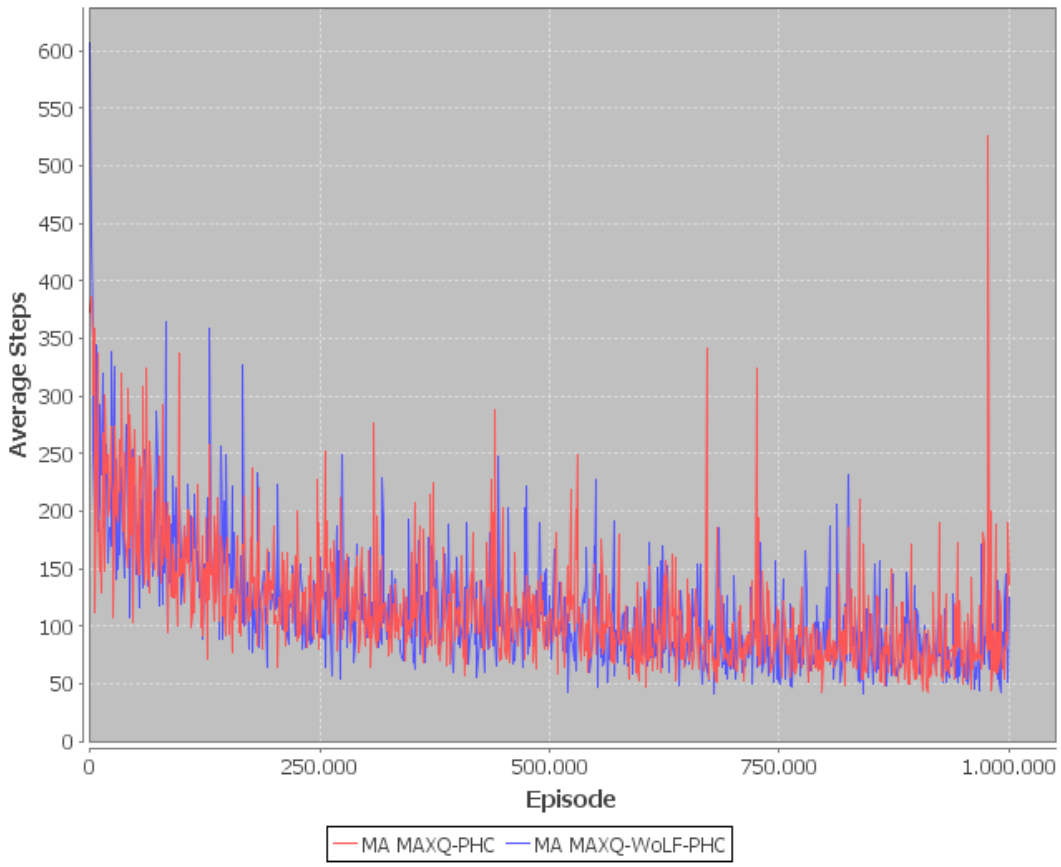
Σχήμα 4.30: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους PHC και WoLF-PHC Πολλαπλών Πρακτόρων για 10 δοκιμές και 1000000 επεισόδια ανά δοκιμή στο MATP
Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3,γ=0.95 και δ=0.2 στον αλγόριθμο PHC, και ε=0.1, α=0.3,γ=0.95, δw=0.05 και δl=0.2 στον στον αλγόριθμο WoLF-PHC. Οι πράκτορες εκπαιδεύονται με επιτυχία και με τους δύο αλγόριθμους, με λίγο καλύτερη απόδοση όταν χρησιμοποιείται ο αλγόριθμος WoLF-PHC, αφού η γραφική παράσταση του αλγόριθμου WoLF-PHC είναι ελαφρώς πιο απαλή από την γραφική παράσταση του αλγόριθμου PHC.

4.3.4.3 Εφαρμογή αλγορίθμων MAXQ-PHC και MAXQ-WoLF-PHC για Πολλαπλούς Πράκτορες

Στη συνέχεια υλοποιήσουμε Ιεραρχικά, χρησιμοποιώντας την MAXQ διάσπαση για Πολλαπλούς Πράκτορες, τους αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC.

Μετά από κάποιες δοκιμές αποφασίστηκε όπως χρησιμοποιηθούν οι ακόλουθες τιμές για τις παραμέτρους: ε=0.1 και γ=0.9, για τον λόγο ότι θέλαμε να υπάρχει μεγάλη βαρύτητα στην επιλογή ενεργειών που δίνουν μακροπρόθεσμα μεγαλύτερες αμοιβές, και στους δύο αλγόριθμους. Επίσης, στον αλγόριθμο MAXQ-PHC καθορίσαμε α=0.5 για κάθε Max κόμβο του γράφου και δ=0.2, ενώ στον αλγόριθμο MAXQ-WoLF-PHC καθορίσαμε α=0.5 για κάθε Max κόμβο, δw=0.05 και δl=0.2, αφού σύμφωνα με τον αλγόριθμο πρέπει δl> δw.

Το Σχήμα 4.31 δείχνει το μέσο αριθμό βημάτων ανά επεισόδιο, που χρειάστηκαν οι πράκτορες για να επιτύχουν τον στόχο τους στο MATP. Παρατηρούμε ότι οι πράκτορες εκπαιδεύονται με επιτυχία, έχοντας παρόμοια συμπεριφορά, και στους δύο αλγόριθμους.

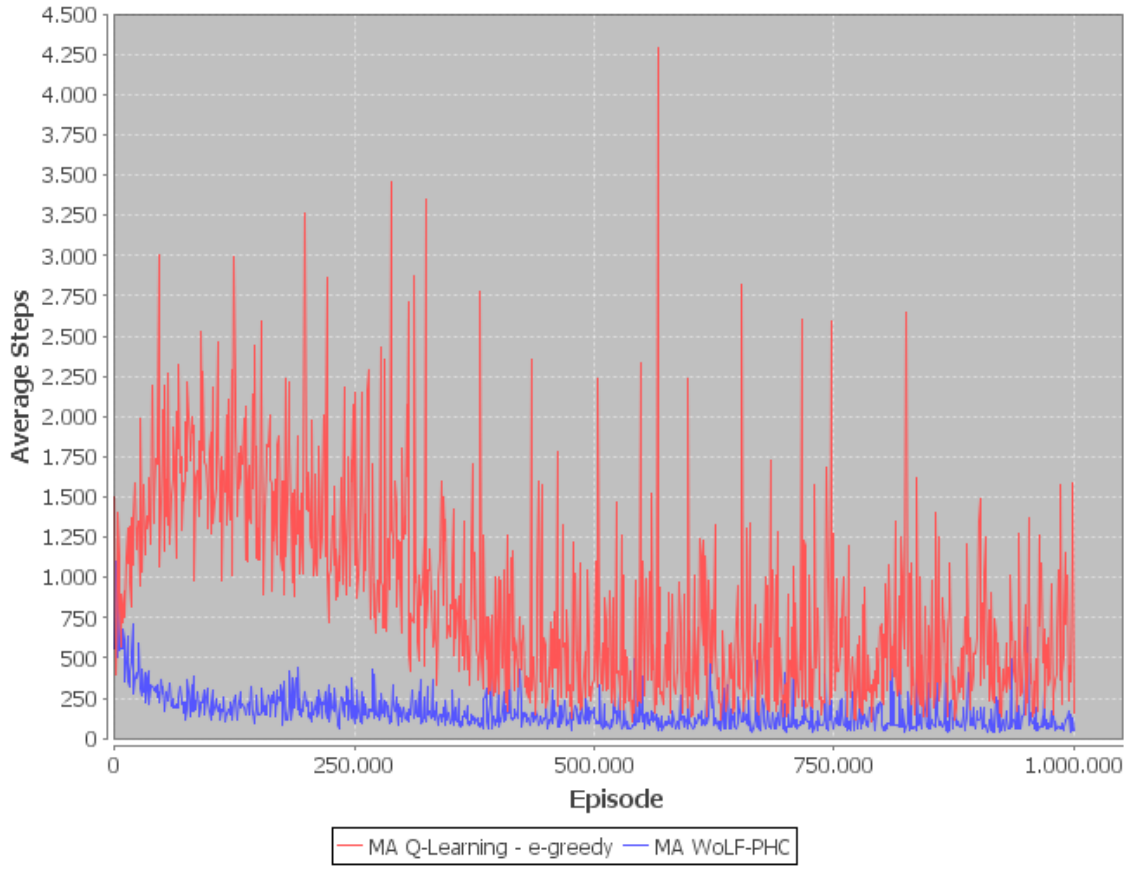


Σχήμα 4.31: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC Πολλαπλών Πρακτόρων για 10 δοκιμές και 1000000 επεισόδια ανά δοκιμή στο MATP
Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου, γ=0.9 και δ=0.2 στον αλγόριθμο MAXQ-PHC, και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου, γ=0.9, δw=0.05 και δl=0.2 στον αλγόριθμο MAXQ-WoLF-PHC. Ο πράκτορας εκπαιδεύεται με επιτυχία, και με τους δύο αλγόριθμους έχοντας παρόμοια συμπεριφορά.

4.3.4.4 Σύγκριση αλγορίθμων Q-Learning και WoLF-PHC Πολλαπλών Πρακτόρων

Στη συνέχεια συγκρίναμε τα αποτελέσματα που πήραμε από τους αλγόριθμους Q-Learning και WoLF-PHC. Στο Σχήμα 4.32 φαίνονται τα αποτελέσματα με την εφαρμογή των δύο αλγορίθμων στο MATP. Παρατηρούμε ότι ο αλγόριθμος WoLF-PHC έχει πάρα πολύ καλύτερη απόδοση από τον αλγόριθμο Q-Learning, αφού είναι

φανερό ότι υπάρχει σύγκλιση του αλγόριθμου WoLF-PHC σε πολύ μικρότερο μέσο αριθμό βημάτων από τον αλγόριθμο Q-Learning, με τον οποίο οι πράκτορες χρειάζονται αρκετά περισσότερα βήματα μέχρι να επιτύχουν τον στόχο τους. Αυτό ήταν αναμενόμενο, αφού ο αλγόριθμος Q-Learning είναι αλγόριθμος που αφορά προβλήματα ΕΜΕΠ.

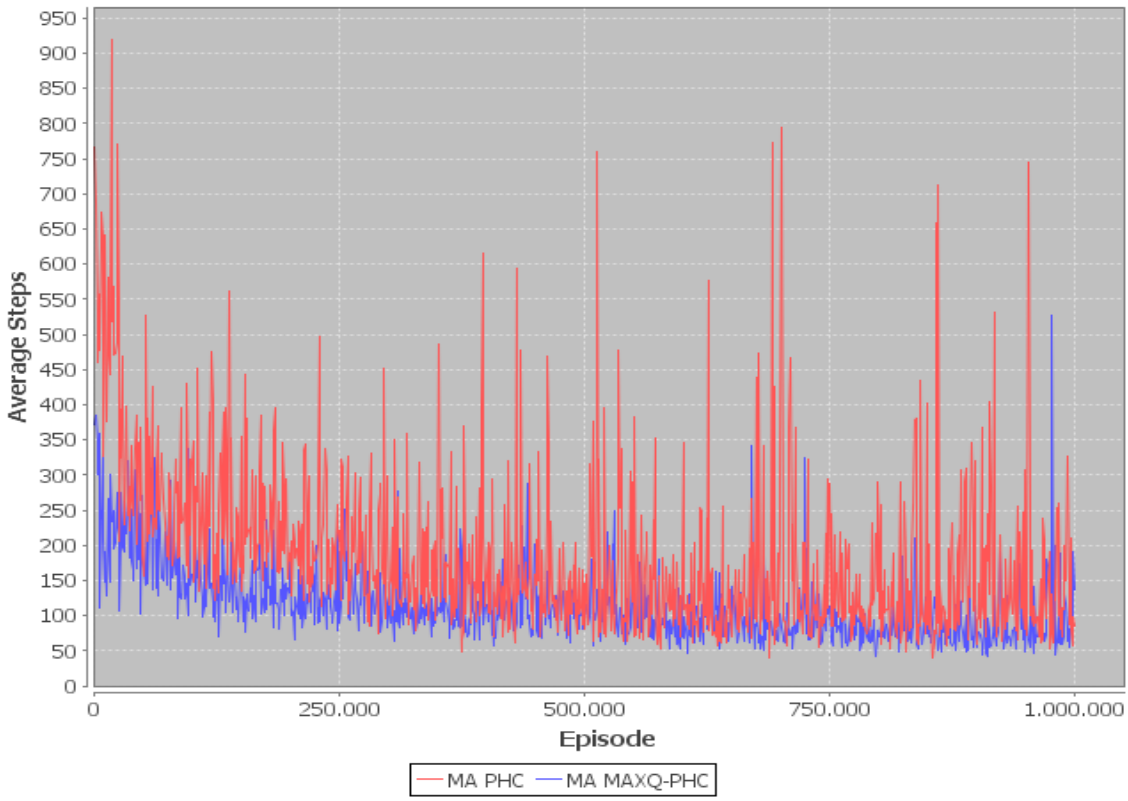


Σχήμα 4.32: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning με ε-greedy πολιτική και WoLF-PHC Πολλαπλών Πρακτόρων για 10 δοκιμές και 1000000 επεισόδια ανά δοκιμή στο MATP

Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3 και γ=0.95 στον αλγόριθμο Q-Learning, και ε=0.1, α=0.3, γ=0.95, δl=0.05 και δl=0.2 στον αλγόριθμο WoLF-PHC. Οι πράκτορες εκπαιδεύονται πολύ καλύτερα χρησιμοποιώντας τον αλγόριθμο WoLF-PHC, αφού χρησιμοποιώντας τον αλγόριθμο WoLF-PHC χρειάζονται πολύ λιγότερα βήματα για να επιτύχουν τον στόχο τους σε σχέση με την χρησιμοποίηση του αλγόριθμου Q-Learning.

4.3.4.5 Σύγκριση αλγορίθμων PHC και MAXQ-PHC Πολλαπλών Πρακτόρων

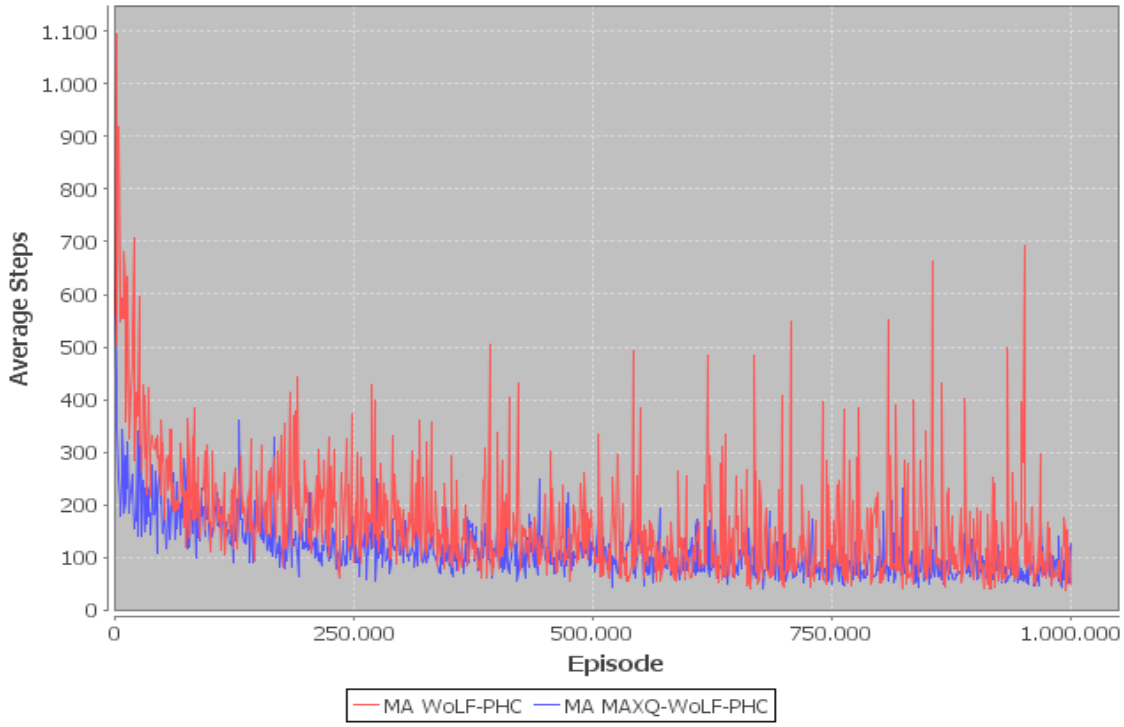
Σε αυτό το σημείο συγκρίναμε τα αποτελέσματα που πήραμε με την εφαρμογή των αλγορίθμων PHC και MAXQ-PHC. Το Σχήμα 4.33 δείχνει τα αποτελέσματα με την εφαρμογή τους στο MATP. Παρατηρούμε ότι με την εφαρμογή του Ιεραρχικού αλγόριθμου MAXQ-PHC η απόδοση του αλγόριθμου βελτιώνεται σε σχέση με τον αλγόριθμο PHC. Βλέπουμε ότι ο μέσος αριθμός των βημάτων που χρειάζονται οι πράκτορες για να επιτύχουν τον στόχο τους είναι πολύ μικρότερος όταν χρησιμοποιείται ο αλγόριθμος MAXQ-PHC, και επίσης η σύγκλιση επέρχεται πιο γρήγορα. Έτσι καταλήγουμε στο συμπέρασμα ότι με την επέκταση του αλγόριθμου PHC στον Ιεραρχικό αλγόριθμο MAXQ-PHC επιτύχαμε την επιτάχυνση της μάθησης, κάτι το οποίο ήταν αναμενόμενο, αφού οι Ιεραρχικοί αλγόριθμοι σκοπεύουν στην επιτάχυνση της μάθησης.



Σχήμα 4.33: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους PHC και MAXQ-PHC Πολλαπλών Πρακτόρων για 10 δοκιμές και 1000000 επεισόδια ανά δοκιμή στο MATP
Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3,γ=0.95 και δ=0.2 στον αλγόριθμο PHC και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου, γ=0.9 και δ=0.2 στον αλγόριθμο MAXQ-PHC. Χρησιμοποιώντας τον αλγόριθμο MAXQ-PHC η μάθηση επιταχύνεται, αφού οι πράκτορες χρειάζονται λιγότερα βήματα για να επιτύχουν τον στόχο τους, σε σχέση με την χρησιμοποίηση του αλγόριθμου PHC.

4.3.4.6 Σύγκριση αλγορίθμων WoLF-PHC και MAXQ-WoLF-PHC Πολλαπλών Πρακτόρων

Έπειτα συγκρίναμε τα αποτελέσματα που πήραμε με την εφαρμογή των αλγορίθμων WoLF-PHC και MAXQ-WoLF-PHC. Το Σχήμα 4.34 δείχνει τα αποτελέσματα με την εφαρμογή τους στο MATP. Παρατηρούμε ότι με την εφαρμογή του Ιεραρχικού αλγόριθμου MAXQ-WoLF-PHC η απόδοση του αλγόριθμου βελτιώνεται σε σχέση με τον αλγόριθμο WoLF-PHC. Βλέπουμε ότι ο μέσος αριθμός των βημάτων που χρειάζονται οι πράκτορες για να επιτύχουν τον στόχο τους είναι πολύ μικρότερος όταν χρησιμοποιείται ο αλγόριθμος MAXQ-WoLF-PHC, και επίσης η σύγκλιση επέρχεται πιο γρήγορα. Έτσι καταλήγουμε στο συμπέρασμα ότι με την επέκταση του αλγόριθμου WoLF-PHC στον Ιεραρχικό αλγόριθμο MAXQ-WoLF-PHC επιτύχαμε την επιτάχυνση της μάθησης, κάτι το οποίο ήταν αναμενόμενο, αφού οι Ιεραρχικοί αλγόριθμοι σκοπεύουν στην επιτάχυνση της μάθησης.



Σχήμα 4.34: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους WoLF-PHC και MAXQ-WoLF-PHC Πολλαπλών Πρακτόρων για 10 δοκιμές και 1000000 επεισόδια ανά δοκιμή στο MATP
Σε αυτή την προσομοίωση οι τιμές των παραμέτρων που χρησιμοποιήθηκαν είναι ε=0.1, α=0.3,γ=0.95, δw=0.05 και δl=0.2 στον αλγόριθμο WoLF-PHC, και ε=0.1, α=0.5 για κάθε Max κόμβο του γράφου,γ=0.95, δw=0.05 και δl=0.2 στον αλγόριθμο MAXQ-WoLF-PHC. Χρησιμοποιώντας τον αλγόριθμο MAXQ-WoLF-PHC η μάθηση επιταχύνεται, αφού οι πράκτορες χρειάζονται λιγότερα βήματα για να επιτύχουν τον στόχο τους, σε σχέση με την χρησιμοποίηση του αλγόριθμου WoLF-PHC.

4.4 Σύνοψη Αποτελεσμάτων

Τελειώνοντας, συνοψίζουμε τα αποτελέσματα απαντώντας έτσι, στο ερώτημα που θέσαμε στην αρχή αυτού του κεφαλαίου, κατά πόσο αυτοί οι αλγόριθμοι είναι αποδοτικοί στα συγκεκριμένα προβλήματα. Στους δύο πίνακες που ακολουθούν έχουμε κατατάξει τους αλγόριθμους σε σειρά καλύτερης απόδοσης. Στον Πίνακα 4.1 παρουσιάζεται η σειρά κατάταξης των αλγόριθμων ΕΜΕΠ, όσον αφορά την εφαρμογή τους στο TP και στο ETP, όπου 1 είναι ο αποδοτικότερος αλγόριθμος και 10 ο λιγότερο αποδοτικός. Στον Πίνακα 4.2 παρουσιάζεται η σειρά κατάταξης των αλγόριθμων ΕΜΠΠ, όσον αφορά την εφαρμογή τους στο TP, όπου 1 είναι ο αποδοτικότερος αλγόριθμος και 5 ο λιγότερο αποδοτικός. Θεωρίσαμε ότι μια καλή μετρική είναι ο «μεσαίος αριθμός» (median) των βημάτων που χρειάζεται κάθε πράκτορας για να επιτύχει τον στόχο του σε κάθε αλγόριθμο, χωρίς να είμαστε απόλυτα βέβαιοι. Για πιο έγκυρα αποτελέσματα θα ήταν καλό να δοκιμαστούν οι αλγόριθμοι σε κάποιο περιβάλλον δοκιμής (test environment), μετά την εκπαίδευση των πρακτόρων, και κατά τη διάρκεια αυτής της δοκιμής να μην γίνεται επιπλεόν εκπαίδευση και εξερεύνηση.

Σύμφωνα με τον Πίνακα 4.1 βλέπουμε ότι πιο αποδοτικοί αλγόριθμοι ΕΜΕΠ για εφαρμογή τους στο TP είναι ο αλγόριθμος SARSA χρησιμοποιώντας την πολιτική Boltzmann και ο αλγόριθμος MAXQ-Q χρησιμοποιώντας την ε-greedy πολιτική με «μεσαίο αριθμό» ίσο με 11.46 και 11.86 αντίστοιχα. Ακολουθούν οι αλγόριθμοι Q-Learning και SARSA χρησιμοποιώντας την ε-greedy πολιτική, με «μεσαίο αριθμό» ίσο με 12.1 και 12.9 αντίστοιχα και στη συνέχεια οι αλγόριθμοι WoLF-PHC, με «μεσαίο αριθμό» ίσο με 14.35 και PHC, με «μεσαίο αριθμό» ίσο με 14.74. Μετά έχουμε τους αλγόριθμους MAXQ-PHC, με «μεσαίο αριθμό» ίσο με 16.64 και MAXQ-WoLF-PHC, με «μεσαίο αριθμό» ίσο με 16.88. Τέλος έχουμε τους αλγόριθμους Q-Learning και MAXQ-Q χρησιμοποιώντας την πολιτική Boltzmann, με «μεσαίο αριθμό» ίσο με 21.88 και 70.22 αντίστοιχα. Στο ETP παρατηρούμε ότι πιο αποδοτικός είναι ο αλγόριθμος MAXQ-Q χρησιμοποιώντας την ε-greedy πολιτική με «μεσαίο αριθμό» ίσο με 50.416 και ακολουθεί ο αλγόριθμος SARSA χρησιμοποιώντας την πολιτική Boltzmann με «μεσαίο αριθμό» ίσο με 59.383. Στη συνέχεια έχουμε τον αλγόριθμο Q-Learning χρησιμοποιώντας την ε-greedy πολιτική, με «μεσαίο αριθμό» ίσο με 61.283 και

ακολουθούν οι αλγόριθμοι MAXQ-WoLF-PHC, SARSA χρησιμοποιώντας την ε-greedy πολιτική και PHC, «μεσαίο αριθμό» ίσο με 85.2, 86.56 και 94.267 αντίστοιχα. Τέλος έχουμε τους αλγόριθμους Q-Learning και MAXQ-Q χρησιμοποιώντας την πολιτική Boltzmann, και τον αλγόριθμο WoLF-PHC, με «μεσαίο αριθμό» ίσο με 109.03, 146.93 και 151.33 αντίστοιχα.

Παρατηρούμε ότι η σειρά κατάταξης των αλγορίθμων στα δύο προβλήματα ΕΜΕΠ διαφέρει. Βλέπουμε ότι αποδοτικότερος αλγόριθμος στο TP φαίνεται να είναι ο αλγόριθμος SARSA χρησιμοποιώντας την πολιτική Boltzmann, ενώ στο ETP παρατηρούμε ότι αποδοτικότερος είναι ο αλγόριθμος MAXQ-Q χρησιμοποιώντας την ε-greedy πολιτική. Αυτό συμβαίνει λόγω του ότι στο ETP έχουμε μεγαλύτερο σύνολο καταστάσεων-ενεργειών και επομένως ο πράκτορας χρειάζεται περισσότερη εκπαίδευση. Γνωρίζοντας ότι στόχος των Ιεραρχικών μεθόδων ΕΜ είναι να επιταχύνουν την μάθηση, δικαιολογημένα ο αλγόριθμος MAXQ-Q είναι ο πιο αποδοτικός αλγόριθμος στο ETP. Βλέπουμε επίσης, πώς και στα δύο προβλήματα, οι αλγόριθμοι Q-Learning και MAXQ-Q έχουν καλύτερα αποτελέσματα όταν χρησιμοποιούν την ε-greedy πολιτική, ενώ ο αλγόριθμος SARSA έχει καλύτερα αποτελέσματα όταν χρησιμοποιεί την πολιτική Boltzmann. Αυτό ίσως να συμβαίνει λόγω του ότι ο αλγόριθμος Q-Learning είναι ένας εκτός πολιτικής αλγόριθμος και ο αλγόριθμος MAXQ-Q είναι μια Ιεραρχική επέκταση του, ενώ ο αλγόριθμος SARSA είναι ένας εντός πολιτικής αλγόριθμος. Επομένως, στους αλγόριθμους Q-Learning και MAXQ-Q, η ανανέωση των *Q* τιμών κάθε κατάστασης, γίνεται βάση της ενέργειας που μεγιστοποιεί την επόμενη κατάσταση, δηλαδή της άπληστης ενέργειας, και όχι της ενέργειας που θα επιλεχθεί για να εκτελεστεί στην επόμενη κατάσταση. Επομένως αν η ενέργεια που θα επιλεχθεί για εκτέλεση στην επόμενη κατάσταση είναι η άπληστη ενέργεια, τότε η ενέργεια που χρησιμοποιείται για την ανανέωση των *Q* τιμών και η επόμενη ενέργεια για εκτέλεση θα είναι οι ίδιες. Σε αντίθεση στον αλγόριθμο SARSA, η ανανέωση των *Q* τιμών κάθε κατάστασης, γίνεται βάση της επόμενης ενέργειας και ίσως να είναι καλύτερα, σε αυτή την περίπτωση, η επιλογή της επόμενης ενέργειας να γίνεται βάση κάποιων πιθανοτήτων. Συνεχίζοντας βλέπουμε ότι οι αλγόριθμοι PHC και WoLF-PHC βρίσκονται στις μεσαίες θέσεις της κατάταξης στο TP, κάτι που σημαίνει ότι, αν και είναι αλγόριθμοι που χρησιμοποιούνται σε προβλήματα ΕΜΠΠ, δεν έχουν

άσχημα αποτελέσματα στο TP, το οποίο είναι ένα πρόβλημα ΕΜΕΠ. Αυτό ίσως να έγκειται στο γεγονός ότι αυτοί οι αλγόριθμοι είναι επέκταση του αλγόριθμου Q-Leanrning. Επίσης ο αλγόριθμος WoLF-PHC είναι ελαφρώς καλύτερος από τον αλγόριθμο PHC, κάτι το οποίο ήταν αναμενόμενο, αφού ο αλγόριθμος WoLF-PHC είναι μια επέκταση του αλγόριθμου PHC η οποία έγινε για να βελτιωθεί ο αλγόριθμος ούτως ώστε να ενθαρρύνεται η σύγκλιση. Κάτι το οποίο δεν ήταν αναμενόμενο είναι ότι στο ETP ο αλγόριθμος PHC φαίνεται να είναι πιο αποδοτικός από τον αλγόριθμο WoLF-PHC. Αυτό δεν το αναμέναμε, αφού σύμφωνα με τα αποτελέσματα που βλέπουμε στην γραφική παράσταση του Σχήματος 4.6, ο αλγόριθμος WoLF-PHC φαίνεται να έχει καλύτερη συμπεριφορά από τον αλγόριθμο PHC. Έτσι, βλέπουμε ότι ίσως να μην έχουμε χρησιμοποιήσει την πιο κατάλληλη μετρική για αυτό το πρόβλημα. Τέλος, βλέπουμε ότι οι Ιεραρχικοί αλγόριθμοι MAXQ-PHC και MAXQ-WoLF-PHC είναι πιο αποδοτικοί στο ETP παρά στο TP, κάτι το οποίο αναμέναμε αφού βοηθούν στην επιτάχυνση της μάθησης, και έτσι ο πράκτορας μαθαίνει πιο γρήγορα χρησιμοποιώντας αυτούς τους αλγόριθμους στο ETP, όπου έχουμε πιο μεγάλο σύνολο καταστάσεων-ενεργειών από ότι στο TP.

Στον Πίνακα 4.2 παρατηρούμε ότι πιο αποδοτικοί αλγόριθμοι ΕΜΠΠ για εφαρμογή τους στο TP είναι οι αλγόριθμοι MAXQ-WoLF-PHC και MAXQ-PHC με «μεσαίο αριθμό» ίσο με 102 και 103 αντίστοιχα. Ακολουθούν οι αλγόριθμοι WoLF-PHC, με «μεσαίο αριθμό» ίσο με 133.1 και PHC, με «μεσαίο αριθμό» ίσο με 153.4. Τέλος έχουμε τον αλγόριθμο Q-Learning χρησιμοποιώντας την ε-greedy πολιτική, με «μεσαίο αριθμό» ίσο με 705.4. Αυτά τα αποτελέσματα ήταν αναμενόμενα, αφού σε αυτό το πρόβλημα έχουμε ένα αρκετά μεγάλο σύνολο καταστάσεων-ενεργειων, και επομένως η χρήση Ιεραρχικών αλγορίθμων που χρησιμοποιούνται σε προβλήματα ΕΜΠΠ, επιταχύνει την μάθηση. Έτσι, δικαιολογημένα οι αλγόριθμοι MAXQ-WoLF-PHC και MAXQ-PHC έχουν την καλύτερη απόδοση. Επίσης, δικαιολογημένα ο αλγόριθμος Q-Learning έχει την χειρότερη απόδοση, αφού είναι ένας αλγόριθμος ο οποίος χρησιμοποιείται σε προβλήματα ΕΜΕΠ. Τελειώνοντας, όπως αναμέναμε, οι αλγόριθμοι PHC και WoLF-PHC έχουν καλύτερη απόδοση από τον αλγόριθμο Q-Learning, αφού αυτοί οι αλγόριθμοι είναι δύο αλγόριθμοι οι οποίοι χρησιμοποιούνται σε προβλήματα ΕΜΠΠ. Ακόμη ο αλγόριθμος WoLF-PHC είναι αποδοτικότερος από τον αλγόριθμο

PHC, κάτι το οποίο ήταν επίσης αναμενόμενο, αφού όπως έχω προαναφέρει ο αλγόριθμος WoLF-PHC επεκτείνει τον αλγόριθμο PHC με σκοπό να τον βελτιώσει.

ΕΝΙΣΧΥΤΙΚΗ ΜΑΘΗΣΗ ΕΝΟΣ ΠΡΑΚΤΟΡΑ				
	TAXI PROBLEM		ΕΚΤΕΤΑΜΕΝΟ TAXI PROBLEM	
ΑΛΓΟΡΘΜΟΙ	ΣΕΙΡΑ ΚΑΤΑΤΑΞΗΣ	ΜΕΣΑΙΟΣ (MEDIAN)	ΣΕΙΡΑ ΚΑΤΑΤΑΞΗΣ	ΜΕΣΑΙΟΣ (MEDIAN)
SARSA (Boltzmann exploration)	1	11.46	2	59.383
MAXQ-Q (ε-greedy)	2	11.86	1	50.416
Q-Learning (ε-greedy)	3	12.1	3	61.283
SARSA (ε-greedy)	4	12.9	5	86.56
WoLF-PHC	5	14.35	10	151.33
PHC	6	14.74	7	94.267
MAXQ-PHC	7	16.64	6	90.417
MAXQ-WoLF-PHC	8	16.88	4	85.2
Q-Learning (Boltzmann exploration)	9	21.88	8	109.03
MAXQ-Q (Boltzmann exploration)	10	70.22	9	146.93

Πίνακας 4.1: Απόδοση αλγορίθμων ΕΜΕΠ
Η απόδοση του κάθε αλγόριθμου καθορίζεται από τον «μεσαίο αριθμό» (median) των βημάτων που χρειάζεται κάθε πράκτορας για να επιτύχει τον στόχο του σε κάθε αλγόριθμο. Στην σειρά κατάταξης έχουμε με 1 τον αποδοτικότερο αλγόριθμο και με 10 τον λιγότερο αποδοτικό.

ΕΝΙΣΧΥΤΙΚΗ ΜΑΘΗΣΗ ΠΟΛΛΑΠΛΩΝ ΠΡΑΚΤΟΡΩΝ		
	TAXI PROBLEM	
ΑΛΓΟΡΘΜΟΙ	ΣΕΙΡΑ ΚΑΤΑΤΑΞΗΣ	ΜΕΣΑΙΟΣ (MEDIAN)
MAXQ-WoLF-PHC	1	102
MAXQ-PHC	2	103
WoLF-PHC	3	133.1
PHC	4	153.4
Q-Learning (ε-greedy)	5	705.4

Πίνακας 4.2: Απόδοση αλγορίθμων ΕΜΠΠ
Η απόδοση του κάθε αλγόριθμου καθορίζεται από τον «μεσαίο αριθμό» (median) των βημάτων που χρειάζεται κάθε πράκτορας για να επιτύχει τον στόχο του σε κάθε αλγόριθμο. Στην σειρά κατάταξης έχουμε με 1 τον αποδοτικότερο αλγόριθμο και με 5 τον λιγότερο αποδοτικό.

Κεφάλαιο 5

Συμπεράσματα και Μελλοντική Εργασία

5.1 Σύνοψη

Η ΕΜ υπήρξε μια ενεργή ερευνητική περιοχή της Τεχνητής Νοημοσύνης για πολλά χρόνια (Sutton και Barto, 1998). Βασίζεται στην ιδέα του ότι η τάση επιλογής μιας ενέργειας πρέπει να ενδυναμώνεται (ενισχύεται) αν παράγει ευνοϊκά αποτελέσματα και να αποδυναμώνεται αν παράγει δυσμενή αποτελέσματα. Γι' αυτό το λόγο μπορεί να χαρακτηριστεί επίσης ως «μάθηση μετ' εμπειρίας», αφού ο εκπαιδευόμενος δεν ξέρει από πριν ποιες ενέργειες να επιλέξει, αλλά αντιθέτως πρέπει να ανακαλύψει ποιες ενέργειες οδηγούν στη μεγαλύτερη αμοιβή, με το να τις δοκιμάσει.

Σε αρκετά προβλήματα ΕΜ υπάρχει ένα μεγάλο σύνολο καταστάσεων-ενεργειών (state-action space). Σε γενικές γραμμές είναι δύσκολο να καθοριστεί το κατάλληλο σύνολο καταστάσεων και ενεργειών σε όλα τα προβλήματα ΕΜ του πραγματικού κόσμου. Τις περισσότερες φορές απαιτείται αρκετά μεγάλη χωρητικότητα μνήμης για να καλύψει όλες τις ενδεχόμενες σχετικές καταστάσεις και ενέργειες που μπορεί να επιλεγούν, καθώς μπορεί να υπάρχει μια ευρεία ποικιλία καταστάσεων και ενεργειών που μπορούν να επιλεγούν για κάθε κατάσταση. Επίσης σε κάποια προβλήματα χρειάζεται απεριόριστη μνήμη, για παράδειγμα όταν έχουμε συνεχής σύνολα καταστάσεων-ενεργειών. Κατά συνέπεια υπάρχει ένα αρκετά μεγάλο πρόβλημα χώρου κατά την

προσπάθεια να διερευνηθούν όλες οι πιθανές ενέργειες από όλες τις πιθανές καταστάσεις. Ακόμη, υπάρχει και αρκετό πρόβλημα χρόνου αφού όσο μεγαλώνει το σύνολο καταστάσεων-ενεργειών ενός προβλήματος, τόσο περισσότερη εκπαίδευση χρειάζεται ο πράκτορας μέχρι να επέλθει η μάθηση. Για την επίλυση αυτού του προβλήματος χρησιμοποιούνται διάφορες μεθόδοι όπως οι Ιεραρχικές Μεθόδοι και οι Τεχνικές Προσέγγισης Συναρτήσεων.

Στόχος αυτής της διπλωματικής εργασίας ήταν να δούμε κατά πόσο οι Ιεραρχικές Μεθόδοι επιταχύνουν την μάθηση και επιλύουν ως ένα βαθμό αυτό το πρόβλημα, τόσο σε προβλήματα ΕΜΕΠ όσο και σε προβλήματα ΕΜΠΠ. Το πλαίσιο που χρησιμοποιήθηκε είναι το «Taxi Problem» και το «Εκτεταμένο Taxi Problem» σε προβλήματα ΕΜΕΠ και το «Multi Agent Taxi Problem», το οποίο είναι μια επέκταση του απλού ΤΡ, σε προβλήματα ΕΜΠΠ. Κάναμε αρχή εξετάζοντας την απόδοση κάποιων αλγορίθμων σε προβλήματα ΕΜΕΠ και συνεχίσαμε εξετάζοντας την απόδοση τους σε προβλήματα ΕΜΠΠ.

Στην ΕΜΕΠ, αρχικά εξετάσαμε τους δύο αλγόριθμους ΧΔ Q-Learning και SARSA, χρησιμοποιώντας την ε-greedy πολιτική και την πολιτική Boltzmann. Παρατηρήσαμε ότι με την εφαρμογή του αλγόριθμου Q-Learning ο πράκτορας εκπαιδευόταν καλύτερα χρησιμοποιώντας την ε-greedy πολιτική, ενώ με την εφαρμογή του αλγόριθμου SARSA ο πράκτορας εκπαιδευόταν καλύτερα χρησιμοποιώντας την πολιτική Boltzmann.

Στη συνέχεια εξετάστηκαν οι αλγόριθμοι PHC και WoLF-PHC, που αν και είναι αλγόριθμοι οι οποίοι χρησιμοποιούνται σε περιβάλλοντα ΕΜΠΠ, έδειξαν ότι έχουν αρκετά καλή απόδοση και σε αυτού του είδους τα προβλήματα, αφού ο πράκτορας εκπαιδεύεται με επιτυχία. Επίσης παρατηρήσαμε ότι όταν ο πράκτορας χρησιμοποιεί τον αλγόριθμο WoLF-PHC τυγχάνει καλύτερης εκπαίδευσης και η μάθηση επέρχεται πιο γρήγορα, παρά όταν χρησιμοποιεί τον αλγόριθμο PHC.

Έπειτα ασχοληθήκαμε με Ιεραρχικές Μεθόδους ΕΜ και ποιο συγκεκριμένα με την Ιεραρχική διάσπαση MAXQ. Αρχικά υλοποιήθηκε ο αλγόριθμος MAXQ-Q στον οποίο ο πράκτορας μπορεί να χρησιμοποιεί είτε την ε-greedy πολιτική είτε την πολιτική

Boltzmann. Παρατηρήσαμε ότι όταν ο πράκτορας χρησιμοποιεί την ε-greedy πολιτική η σύγκλιση επέρχεται σε πολύ μικρότερο αριθμό βημάτων, παρά όταν χρησιμοποιείται η πολιτική Boltzmann. Επίσης συγκρίνοντας τα αποτελέσματα από την εφαρμογή του αλγόριθμου MAXQ-Q με τα αποτελέσματα από την εφαρμογή του αλγόριθμου Q-Learning παρατηρήσαμε ότι έχει γίνει σημαντική επιτάχυνση της μάθησης με την εφαρμογή του αλγόριθμου MAXQ-Q, ειδικά στην προσομοίωση του ETP.

Όταν είδαμε όλα αυτά τα αποτελέσματα, σκεφτήκαμε να υλοποιήσουμε Ιεραρχικά τους αλγόριθμους PHC και WoLF-PHC χρησιμοποιώντας την MAXQ διάσπαση, κάτι το οποίο δεν έχει παρατηρηθεί στη βιβλιογραφία. Έτσι πρωτοτυπήσαμε υλοποιώντας τους αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC αντίστοιχα, οι οποίοι έδειξαν ότι έχουν αρκετά καλή απόδοση. Όταν ο πράκτορας χρησιμοποιεί τον αλγόριθμο MAXQ-WoLF-PHC η μάθηση επέρχεται πιο γρήγορα, παρά όταν χρησιμοποιείται ο αλγόριθμος MAXQ-PHC. Ακόμη, συγκρίνοντας τα αποτελέσματα από την εφαρμογή του αλγόριθμου WoLF-PHC με τα αποτελέσματα από την εφαρμογή του αλγόριθμου MAXQ-WoLF-PHC παρατηρήσαμε ότι η μάθηση έχει επιταχυνθεί σημαντικά με την εφαρμογή του αλγόριθμου MAXQ-WoLF-PHC.

Ακολούθως εφαρμόσαμε τους αλγόριθμους σε περιβάλλοντα ΕΜΠΠ. Ξεκινήσαμε με την εφαρμογή του αλγόριθμου Q-Learning. Αφού όπως είδαμε στην ΕΜΕΠ, ο αλγόριθμος Q-Learning έχει καλύτερα αποτελέσματα χρησιμοποιώντας την ε-greedy πολιτική, έγινε εφαρμογή του, χρησιμοποιώντας αυτή την πολιτική. Παρατηρήσαμε όμως ότι δεν έχει αρκετά καλή απόδοση σε περιβάλλοντα ΕΜΠΠ.

Στη συνέχεια δοκιμάσαμε τους αλγόριθμους PHC και WoLF-PHC οι οποίοι σε σύγκριση με τον αλγόριθμο Q-Learning έχουν πολύ καλύτερη απόδοση. Έτσι, αποφασίσαμε ότι δεν έχει τόσο ενδιαφέρον η εφαρμογή του αλγόριθμου MAXQ-Q σε περιβάλλοντα ΕΜΠΠ αφού είναι μια Ιεραρχική επέκταση του αλγόριθμου Q-Learning, που όπως είδαμε δεν έχει καλή απόδοση.

Έτσι πρωτοτυπήσαμε εφαρμόζοντας τους αλγορίθμους MAXQ-PHC και MAXQ-WoLF-PHC σε περιβάλλοντα ΕΜΠΠ. Παρατηρήσαμε ότι οι πράκτορες εκπαιδεύονται

αρκετά καλά, και επίσης η μάθηση επιταχύνεται αρκετά σε σχέση με τους αλγόριθμους PHC και WoLF-PHC.

5.2 Συμπεράσματα

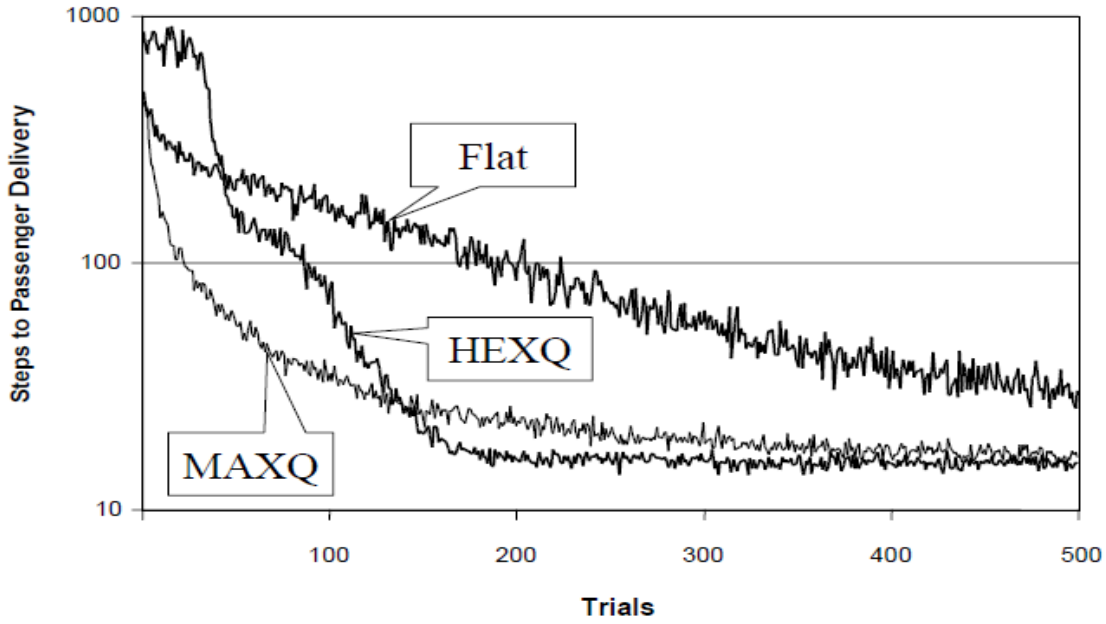
Απαντώντας το ερώτημα που θέσαμε στην αρχή, καταλήξαμε στο συμπέρασμα ότι όντως οι Ιεραρχικές Μεθόδοι επιταχύνουν την μάθηση και έτσι επιλύουν ως ένα βαθμό το πρόβλημα που δημιουργείται όταν έχουμε μεγάλο σύνολο καταστάσεων-ενεργειών, τόσο σε προβλήματα ΕΜΕΠ, όσο και σε προβλήματα ΕΜΠΠ. Αυτό μπορεί να επιτευχθεί με την χρησιμοποίηση της Ιεραρχικής διάσπασης MAXQ. Με την χρησιμοποίηση αυτής της Ιεραρχικής διάσπασης καταφέραμε να πρωτοτυπήσουμε δημιουργώντας δύο Ιεραρχικούς αλγόριθμους, τους αλγόριθμους MAXQ-PHC και MAXQ-WoLF-PHC, οι οποίοι επιτυγχάνουν να επιταχύνουν την μάθηση τόσο σε προβλήματα ΕΜΕΠ, όσο και σε προβλήματα ΕΜΠΠ.

5.3 Σύγκριση με άλλες μελέτες

Αυτή η έρευνα αποτελεί μια μελέτη αλγορίθμων ΕΜ, και κυρίως αλγορίθμων ΙΕΜ, τόσο στο πεδίο ενός πράκτορα αλλά και στο πεδίο πολλαπλών πρακτόρων χρησιμοποιώντας σαν περιβάλλον το TP, το ETP και το MATP. Προηγούμενες μελέτες από άλλους ερευνητές επικεντρώθηκαν κυρίως στην ανάπτυξη Ιεραρχικών αλγορίθμων οι οποίοι προσπαθούν να αντιμετωπίσουν το πρόβλημα του μεγάλου συνόλου καταστάσεων-ενεργειών που αντιμετωπίζει η ΕΜ.

Μια έρευνα που θα μπορούσαμε να την συγκρίνουμε με την δική μας, και αφορά περιβάλλοντα ενός πράκτορα, είναι αυτή του Hengst (2002). Σε αυτή την έρευνα ο Hengst παρουσίασε τον αλγόριθμο HEXQ ο οποίος προσπαθεί να εντοπίσει αυτόματα την ιεραρχική δομή του προβλήματος. Ο αριθμός των δοκιμών που χρησιμοποίησαν είναι 100 και ο αριθμός των επεισοδίων είναι 500. Σε κάθε επεισόδιο κατέγραφαν τον αριθμό βημάτων που χρειάστηκε ο πράκτορας για να επιτύχει τον στόχο του,

βρίσκοντας στο τέλος το μέσο αριθμό βημάτων που χρειάστηκε για κάθε επεισόδιο, όπως ακριβώς γίνεται και σε αυτή την εργασία. Εμείς όμως, σε αυτή την εργασία, έχουμε χρησιμοποιήσει 50 δοκιμές και 3000 επεισόδια όμως πάλι μπορεί να γίνει η σύγκριση. Παρατηρούμε ότι η σύγκλιση στον αλγόριθμο HEXQ επέρχεται μετά από περίπου 200 επεισόδια, αλλά αρχικά χρειάζεται πάρα πολλά βήματα, περίπου 900, αφού πρέπει πρώτα να βρει την Ιεραρχία. Συγκρίνοντας με τους Ιεραρχικούς αλγόριθμους που υλοποιήσαμε εμείς (MAXQ-Q, MAXQ-PHC και MAXQ-WoLF-PHC) η σύγκλιση επέρχεται μετά από περίπου 500 επεισόδια (Σχήμα 4.7 και Σχήμα 4.9). Έτσι βλέπουμε ότι η σύγκλιση στον αλγόριθμο HEXQ επέρχεται περίπου 2.5 φορές πιο γρήγορα από τους δικούς μας αλγόριθμους. Όμως παρατηρούμε ότι αρχικά οι δικοί μας αλγόριθμοι χρειάζονται πολύ λιγότερα βήματα, περίπου 175 ο αλγόριθμος MAXQ-Q (Σχήμα 4.7) και περίπου 300 οι αλγόριθμοι MAXQ-PHC και MAXQ-WoLF-PHC (Σχήμα 4.9). Έτσι βλέπουμε ότι αρχικά ο αλγόριθμος MAXQ-Q χρειάζεται περίπου 5 φορές λιγότερα βήματα και οι αλγόριθμοι MAXQ-PHC και MAXQ-WoLF-PHC περίπου 3 φορές λιγότερα βήματα από τον αλγόριθμο HEXQ. Το Σχήμα 5.1 είναι παρμένο από το άρθρο «Discovering Hierarchy in Reinforcement Learning with HEXQ» (Hengst, 2002) και δείχνει το μέσο αριθμό βημάτων που χρειάζεται ο πράκτορας για να επιτύχει τον στόχο του χρησιμοποιώντας τους αλγόριθμους Q-Learning, MAXQ και HEXQ. Επίσης αφού ο αλγόριθμος HEXQ βρίσκει την Ιεραρχική δομή από μόνος του, μπορεί να χρησιμοποιηθεί σε οποιοδήποτε άλλο πρόβλημα, ενώ για να χρησιμοποιηθούν οι δικοί μας αλγόριθμοι, θα πρέπει πρώτα να υλοποιήσουμε την Ιεραρχική δομή για το συγκεκριμένο πρόβλημα.



Σχήμα 5.1: Μέσος όρος βημάτων χρησιμοποιώντας τους αλγόριθμους Q-Learning και MAXQ και HEXQ για 100 δοκιμές και 500 επεισόδια ανά δοκιμή στο TP (Hengst, 2002)

Η σύγκλιση επέρχεται πιο γρήγορα στον αλγόριθμο HEXQ, μετά από περίπου 200 επεισόδια, ενώ αρχικά χρειάζεται αρκετά βήματα, περίπου 900, μέχρι να μάθει την Ιεραρχία.

5.4 Συνεισφορά

Σε αυτή τη διπλωματική εργασία δοκιμάσαμε τον αλγόριθμο MAXQ-Q, όχι μόνο στο αρχικό πρόβλημα (TP), για το οποίο προτάθηκε, αλλά και σε ένα πρόβλημα το οποίο είναι επέκταση του αρχικού (ETP), και έχει προταθεί για την υλοποίηση του αλγόριθμου DOORMAX (Diuk, Cohen και Littman, 2008), έτσι ώστε να παρατηρήσουμε την συμπεριφορά του αλγόριθμου σε ένα πρόβλημα με μεγαλύτερο σύνολο καταστάσεων-ενεργειών.

Επίσης αποφασίσαμε να δοκιμάσουμε τον αλγόριθμο Q-Learning που χρησιμοποιείται σε προβλήματα ΕΜΕΠ, και σε προβλήματα ΕΜΠΠ για να δούμε πως συμπεριφέρεται σε αυτά τα προβλήματα.

Ακόμη δοκιμάσαμε τους αλγόριθμους PHC και WoLF-PHC, οι οποίοι είναι αλγόριθμοι που χρησιμοποιούνται σε προβλήματα ΕΜΠΠ, και σε προβλήματα ΕΜΕΠ για να δούμε πως συμπεριφέρονται σε αυτά τα προβλήματα. Δεν έχει παρατηρηθεί στην βιβλιογραφία υλοποίηση τους σε προβλήματα ΕΜΕΠ, αν και σύμφωνα με τα αποτελέσματα που πήραμε έδειξαν ότι θα μπορούσαν να χρησιμοποιηθούν στο TP και ETP.

Τέλος, καταφέραμε να πρωτοτυπήσουμε αφού οι Ιεραρχικοί αλγόριθμοι MAXQ-PHC και MAXQ-WoLF-PHC που υλοποιήσαμε, τόσο για προβλήματα ΕΜΕΠ, όσο και για προβλήματα ΕΜΠΠ, εξ όσον γνωρίζουμε δεν έχουν υλοποιηθεί μέχρι στιγμής από κάποιον άλλο. Με αυτούς τους αλγόριθμους επιτύχαμε να αυξήσουμε τον ρυθμό μάθησης, και στα προβλήματα ΕΜΕΠ αλλά και στα προβλήματα ΕΜΠΠ, κάτι το οποίο είναι αρκετά σημαντικό, αφού φαίνεται ότι αυτοί οι αλγόριθμοι μπορούν να χρησιμοποιηθούν και στις δύο περιπτώσεις.

5.5 Μελλοντική εργασία

Αυτή η διπλωματική εργασία θα μπορούσε να επεκταθεί με πολλούς τρόπους. Θα μπορούσε να χρησιμοποιηθεί και κάποιο άλλο πλαίσιο εκτός από το TP, όπως για παράδειγμα ο λαβύρινθος του Parr (Parr, 1998), ο οποίος έχει δοκιμαστεί από τον Dietterich (2000) χρησιμοποιώντας τον αλγόριθμο MAXQ-Q. Έτσι θα μπορεί να γίνει μια πιο γενική αξιολόγηση όσον αφορά την απόδοση αυτών των αλγορίθμων σε αυτού του είδους προβλήματα.

Ο τρόπος με τον οποίο γίνεται η εύρεση της τιμής της Συνάρτησης Αξίας στους σύνθετους κόμβους του MAXQ γράφου χρησιμοποιώντας την συνάρτηση EvaluateMaxNode(i,s) (σχήμα 3.6) είναι υπολογιστικά ακριβός, αφού κάνει ένα πλήρη

έλεγχο από όλα τα μονοπάτια του γράφου, χρησιμοποιώντας προθεματική διάσχιση (best-first search). Θα μπορούσε να βελτιωθεί έτσι ώστε όταν αλλάξει η κατάσταση του περιβάλλοντος, να επανε3ετασθούν μόνο οι κόμβοι των οποίων οι τιμές έχουν υποστεί αλλαγές, λόγω αυτής της αλλαγής της κατάστασης του περιβάλλοντος.

Στην ΕΜΠΠ θα μπορούσε να χρησιμοποιηθούν περισσότεροι από δύο πράκτορες. Θα μπορούσε επίσης σε μια προσομοίωση ο κάθε πράκτορας να χρησιμοποιεί διαφορετικό αλγόριθμο. Με αυτό τον τρόπο θα δούμε πως θα επηρεαστεί η συμπεριφορά του κάθε πράκτορα και κατά πόσο θα μπορεί να συνεργαστεί με ένα πράκτορα που λειτουργεί διαφορετικά.

Στους αλγόριθμους που χρησιμοποιήσαμε, δεν υπάρχει καμία επικοινωνία μεταξύ των πρακτόρων και επομένως καθόλου ανταλλαγή γνώσης και εμπειρίας. Θα μπορούσε να υλοποιηθεί κάποιος αλγόριθμος ο οποίος να έχει την ικανότητα ανταλλαγής γνώσης και εμπειρίας, όπως είναι ο αλγόριθμος WoLF-PSP (Hwang et al, 2007), ο οποίος είναι μια επέκταση του αλγόριθμου WoLF-PHC. Έτσι θα μπορούσε εύκολα, στη συνέχεια, να επεκταθεί Ιεραρχικά, με μια επέκταση του αλγόριθμου MAXQ-WoLF-PHC.

Οι αλγόριθμοι Q-Learning, PHC και WoLF-PHC είναι εκτός πολιτικής αλγόριθμοι. Θα μπορούσαν να υλοποιηθούν κάποιοι αλγόριθμοι εντός πολιτικής, όπως SARSA-PHC και SARSA-WoLF-PHC, οι οποίοι θα συνδυάζουν τον αλγόριθμο SARSA και τους αλγόριθμους PHC και WoLF-PHC αντίστοιχα. Στη συνέχεια θα μπορούσαν να επεκταθούν ιεραρχικά χρησιμοποιώνατας την MAXQ διάσπαση, υλοποιώντας έτσι τους αλγόριθμους MAXQ-SARSA, MAXQ-SARSA-PHC και MAXQ-SARSA-WoLF-PHC.

Αναφορές

Andre D. and Russell S. J. (2001). Programmable reinforcement learning agents. *In Proceedings of Advances in Neural Information Processing Systems 13*, pp. 1019–1025. MIT Press.

Bellman R. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics,6,* pp. 679-684.

Bowling M. and Veloso M. (2001).Rational and Convergent Learning in Stochastic Games. In: *Proceedings of the 17th international joint conference on Artificial intelligence, 2*, pp.1021-1026. San Francisco: Morgan Kaufmann.

Dayan P. and Hinton G. (1993). Feudal reinforcement learning. *NIPS, 5*, pp. 271–278. San Francisco:Morgan Kaufmann.

Dietterich T. G. (2000), Hierarchical reinforcement learning with the MAXQ value function decomposition, *Journal of Artificial Intelligence Research, 13*, pp. 227–303.

Diuk C., Cohen A. and Littman M. L. (2008). An object-oriented representation for efficient reinforcement learning. In: *Proceedings of the 25th international conference on Machine learning*, pp. 240-247. New York: ACM.

Ghavamzadeh M. and Mahadevan S. (2007). Hierarchical Average Reward Reinforcement Learning. pp. 2629-2669. MIT Press.

Ghavamzadeh M. and Mahadevan S. (2004). Learning to communicate and act using hierarchical reinforcement learning. In:*Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 3* , pp. 1114-1121. Washington: IEEE Computer Society.

Hengst B. (2002). Discovering Hierarchy in Reinforcement Learning with HEXQ, In: *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 243-250. San Francisco: Morgan Kaufmann.

Hwang K. S., Lin C. J., Wu C. J. and Lo C. Y. (2007). Cooperation between multiple agents based on partially sharing policy. In:*Proceedings of the intelligent computing 3rd international conference on Advanced intelligent computing theories and applications*, pp. 422-432. Berlin: Springer-Verlag.

Kaelbling L. P. (1993). Hierarchical reinforcement learning: Preliminary results. *ICML-93*, pp. 167–173 San Francisco:Morgan Kaufmann.

Kaelbling L. P., Moore A. and Littman M. (1996) Reinforcement Learning: A survey, *Journal of Artificial Intelligence Research 4*, pp. 237-285.

Marthi B., Russell S., Latham D. and Guestrin C. (2005). Concurrent Hierarchical Reinforcement Learning. In:*Proceedings of the 19th international joint conference on Artificial intelligence,* pp. 779-785. San Francisco: Morgan Kaufmann Publishers Inc.

Mehta N., Ray S., Tadepalli P. and Dietterich T. (2008). Automatic discovery and transfer of MAXQ hierarchies. In: *Proceedings of the 25th international conference on Machine learning,* pp. 648-655. New York: ACM.

Mehta N., and Tadepalli P. (2005). Multi-Agent Shared Hierarchy Reinforcement Learning. ICML Rich Representations in Reinforcement Learning Workshop.

Michie D. and Chambers R. A. (1968). Boxes: An Experiment in Adaptive Control. In E. Dale and D. Michie (Eds.), *Machine Intelligence 2*, pp. 137-152. Edinburgh: Oliver and Boyd

Mirzazadeh F., Behsaz B. and Beigy H. (2005). A New Learning Algorithm for the MAXQ Hierarchical Reinforcement Learning Method. In: *Proceedings of International Conference on Information and Communication Technology* , pp. 105-108. Dhaka: IEEE Computer Society.

Osentoski S. and Mahadevan S. (2010). Basis function construction for hierarchical reinforcement learning. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, 1*, pp. 747-754. Richland: International Foundation for Autonomous Agents and Multiagent Systems.

Parr R. (1998), Hierarchical control and learning for Markov decision processes, Ph.D. dissertation, University of California at Berkeley.

Rummery G. A. and Niranjan M., On-line Q-learning using connectionist systems. Technical Report CUED/F-1NFENG/TR 166, Cambridge University Engineering Department, 1994.

Shen J., Liu H., and Gu G. Hierarchical Reinforcement Learning with OMQ. *In Proceedings of IEEE*. pp.584588.

Singh S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Mach. Learn., 8,* pp. 323–339.

Sutton R. S. (1996). Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding, In: *Proceedings of the 1995 conference on Advances in Neural Information Processing Systems, 8*, pp. 1038-1044. Cambridge, MA: MIT Press.

Sutton R. S. and Barto A. G. (1998). Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press,
Available: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html.

Sutton R., Precup D. and Singh S. (1999). Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Journal of Artificial Intelligence*, *112*, pp. 181-211.

Watkins C. J. C. H. (1989). Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, England.

Wiering M. and Schmidhuber J. (1996). HQ-Learning: Discovering Markovian Subgoals For Non-Markovian Reinforcement Learning. Technical Report IDSIA-95-96,Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale.

Γενική βιβλιογραφία

Βασιλειάδης Β. Ενισχυτική Μάθηση Πολλαπλών Πρακτόρων όπως εφαρμόζεται σε Παίγνια Γενικού Αθροίσματος. Ατομική Διπλωματική Εργασία 2007, Τμήμα Πληροφορικής ,Πανεπιστήμιο Κύπρου.

Καραολής Ι. Προσέγγιση Κυπριακού προβλήματος με Ενισχυτική Μάθηση Πολλαπλών Πρακτόρων όπως εφαρμόζεται σε Παίγνια Γενικού Αθροίσματος. Ατομική Διπλωματική Εργασία 2010, Τμήμα Πληροφορικής ,Πανεπιστήμιο Κύπρου.

Dr. Chris Christodoulou, EPL 442 - Informational Learning Systems notes, Department of Computer Science, University of Cyprus

Scholarpedia. Reinforcement learning from http://www.scholarpedia.org/article/Reinforcement_learning.

Παράρτημα Α

Πηγαίος Κώδικας

Simulation.java

```
/**
 * Afairetiki klasi gia tin prosomiwsi
 *
 * @author Giannis
 *
 */
public abstract class Simulation {
	/**
	 * methodos i opoia kani tin prosomiosi
	 */
	abstract void simulation();
}
```

Main_Simulation.java

```
/**
 * einai i kiria klasi tou sistimatos
 *
 * @author Giannis
 *
 */
public class Main_Simulation extends Simulation {

	/**
	 * methodos i opoia kani tin prosomiosi
	 */
	public void simulation() {
		int choice = 0;
		int count = 0;
		int nextMove = 0;
		while (true) { // atermon vrogxos skopima
			System.out
					.println("Do you want Single Agent or
Multi Agent simulation?:");
			do {
				if (count > 0)
					System.out
							.println("Your choice should
be one of the following:");
				System.out.println("\t0: Single Agent");
				System.out.println("\t1: Multi Agent");
				System.out.println();
```

```
                        count++;
                        choice = StdIn.readInt();
                  } while (choice < 0 || choice > 1);

                  count = 0;
                  if (choice == 0) {
                        SA_Simulation s = new SA_Simulation();
                        s.simulation();
                  } else {
                        MA_Simulation s = new MA_Simulation();
                        s.simulation();
                  }

                  do {
                        System.out.println("Please choose the next
move:");
                        System.out.println("\t0: Continue with
simulation");
                        System.out.println("\t1: Exit");
                        nextMove = StdIn.readInt();
                  } while (nextMove < 0 || nextMove > 1);

                  if (nextMove == 1) {
                        System.out.println("Exiting..");
                        System.exit(-1);
                  }
            }
      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            Main_Simulation s=new Main_Simulation();
            s.simulation();
      }

}
```

SA_Simulation.java

```
/**
 * klasis i opoia aposkopei stin prosomiwsi ton single agent
 * algorithmon
 * @author Giannis
 *
 */
public class SA_Simulation extends Simulation {

      /**
       * methodos i opoia ektiponi tis grafikes gia tous algorithmous
       * pou exoun ektelesti
       *
```

```
	 * @param table
	 */
	public void plotGraphs(boolean[] table) {
		System.out.println("Make your choice between the executed
algorithms:");
		System.out
				.println("Write your choice in one line
separated with space..");
		for (int i = 0; i < table.length; i++) {
			if (table[i]) {
				switch (i) {
				case 0:
					System.out.println("\t0: Q-Learning");
					break;
				case 1:
					System.out.println("\t1: SARSA");
					break;
				case 2:
					System.out.println("\t2: PHC");
					break;
				case 3:
					System.out.println("\t3: WoLF-PHC");
					break;
				case 4:
					System.out.println("\t4: MAXQ-Q");
					break;
				case 5:
					System.out.println("\t5: MAXQ-PHC");
					break;
				case 6:
					System.out.println("\t6: MAXQ-WoLF-
PHC");
					break;
				default:
					System.err.println("Invalid number for
choice..Exiting..");
					System.exit(-1);
				}
			}
		}
		boolean[] algorithms = new boolean[7];// 7 einai o
arithmos ton

	// algorithmon
		for (int i = 0; i < table.length; i++)
			algorithms[i] = false;
		StdIn.readLine();
		String line = StdIn.readLine();
		String[] splitline = line.split(" ");

		for (int i = 0; i < splitline.length; i++)
			algorithms[Integer.parseInt(splitline[i])] = true;

		System.out.println("Plot in the same graph or in different
graphs?");
		System.out.println("\tPress 0 for the same or 1 for
different");
		boolean flag = StdIn.readBoolean();
```

```
            PlotGraph.makeGraph(algorithms, flag);

      }

      /**
       * methodos stin opoia ginete i epilogi tis epomenis energias
       * apo ton xristi
       * @return
       */
      public int chooseNextMove() {
            int nextMove;
            do {
                  System.out.println("Please choose the next move:");
                  System.out.println("\t0: Choose another Single Agent
algorithm");
                  System.out.println("\t1: Plot graphs");
                  System.out
                              .println("\t2: Choose between Single
Agent and Multi Agent Algorithms");
                  System.out.println("\t3: Exit");
                  nextMove = StdIn.readInt();
            } while (nextMove < 0 || nextMove > 3);
            return nextMove;
      }

      /**
       * methodos i opoia kani tin prosomiosi
       */
      public void simulation() {
            int choice = 0;
            int count = 0;
            int nextMove = 0;
            boolean[] table = new boolean[7];// 7 einai o arithmos ton
algorithmon
            for (int i = 0; i < table.length; i++)
                  table[i] = false;
            while (true) { // atermon vrogxos skopima
                  System.out.println("Please choose an algorithm:");
                  do {
                        if (count > 0)
                              System.out
                                          .println("Your choice should
be one of the following:");
                        System.out.println("\t0: Q-Learning");
                        System.out.println("\t1: SARSA");
                        System.out.println("\t2: PHC");
                        System.out.println("\t3: WoLF-PHC");
                        System.out.println("\t4: MAXQ-Q");
                        System.out.println("\t5: MAXQ-PHC");
                        System.out.println("\t6: MAXQ-WoLF-PHC");
                        System.out.println();
                        count++;
                        choice = StdIn.readInt();
                  } while (choice < 0 || choice > 6);

                  table[choice] = true;
                  count = 0;
                  RunTrials r = new RunTrials();
```

```
                    r.calculate(choice);

                    nextMove = chooseNextMove();

                    if (nextMove == 1) {
                          do {
                                plotGraphs(table);
                                nextMove = chooseNextMove();
                          } while (nextMove == 1);
                    }
                    if (nextMove == 3) {
                          System.out.println("Exiting..");
                          System.exit(-1);
                    }
                    if (nextMove == 2) {
                          break;
                    }

              }

        }

        /**
         * @param args
         */
        public static void main(String[] args) {
              // TODO Auto-generated method stub
              SA_Simulation s = new SA_Simulation();
              s.simulation();
        }

}
```

SA_Gridworld.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * klasi pou antiprosopevi ena geniko gridworld
 *
 * @author Giannis
 *
 */
public class SA_Gridworld {

      public static int size_x; // grammes tou grid
      public static int size_y; // stiles tou grid
      public static ArrayList<Wall> wall = new ArrayList<Wall>();;
      public static int numOfStartPosition; // kai mesa sto ta3i sto
taxi problem
      public static int numOfGoals;
      public static int states;
```

```
	public static int actions;
	private static int[] pasLoc; // passenger location
	private static int[] destLoc; // destination location

	// rewards gia kathe action
	public static int wall_hit;
	public static int wrong_put;
	public static int wrong_get;
	public static int success;
	public static int simple_move;

	// statheres gia actions
	public static final int north = 0;
	public static final int south = 1;
	public static final int east = 2;
	public static final int west = 3;
	public static final int pickup = 4;
	public static final int putdown = 5;

	public static void setStates() {

		states = size_x * size_y * numOfStartPosition *
numOfGoals;
	}

	/**
	 * methodos i opia epistrefi ton arithmo ton grammwn (x) tou
grid
	 *
	 * @return the size_x
	 */
	public static int getSize_x() {
		return size_x;
	}

	/**
	 * methodos i opia epistrefi ton arithmo ton stilon (y) tou grid
	 *
	 * @return the size_y
	 */
	public static int getSize_y() {
		return size_y;
	}

	/**
	 * methodos i opoia epistrefi ton pinaka me tis sintetagmenes
twn wall
	 *
	 * @return the wall
	 */
	public ArrayList<Wall> getWall() {
		return wall;
	}

	/**
	 * /** methodos i opoia epistrefi ton arithmo ton arxikon
thesewn pou
	 * iparxoun sto grid
```

```
	 *
	 * @return the numOfStartPosition
	 */
	public static int getNumOfStartPosition() {
		return numOfStartPosition;
	}

	/**
	 * methodos i opoia epistrefi ton arithmon ton telikon thesewn
pou iparxoun
	 * sto grid
	 *
	 * @return the numOfGoals
	 */
	public static int getNumOfGoals() {
		return numOfGoals;
	}

	/**
	 * methodos i opoia dimiourgi ena wall meta3i ton dio cells
	 *
	 * @param cell1
	 * @param cell2
	 */
	public static void setWall(int cell1, int cell2) {
		if (((Math.abs(cell1 - cell2) == 1) || ((Math.abs(cell1 -
cell2) == getSize_y())))
				&& (cell1 >= 0)
				&& (cell1 < ((getSize_x() * getSize_y())))
				&& (cell2 >= 0) && (cell2 < ((getSize_x() *
getSize_y())))) {
			Wall temp = new Wall(cell1, cell2);
			wall.add(temp);
		} else {
			System.err
					.println("You cannot enter a wall
between these cells\nExiting..");
			System.exit(-1);
		}

	}

	/**
	 * methodos i opoia filaei ti thesi tou passenger
	 *
	 * @param i
	 * @param location
	 */
	public static void setPasLoc(int i, int location) {
		pasLoc[i] = location;
	}

	/**
	 * methodos i opoia epistrefi tin thesi tou passenger simfwna me
tin
	 * metavliti i
	 *
	 * @param i
```

```
	 * @return
	 */
	public static int getPasLoc(int i) {
		return pasLoc[i];
	}

	/**
	 * methodos i opoia anatheti filaei to destination
	 *
	 * @param i
	 * @param location
	 */
	public static void setDestLoc(int i, int location) {
		destLoc[i] = location;
	}

	/**
	 * methodos i opoia epistrefi to destination
	 *
	 * @param i
	 * @return
	 */
	public static int getDestLoc(int i) {
		return destLoc[i];
	}

	/**
	 * methodos i opoia elegxi an iparxi wall stin katefthinsi opou
	 * theli na metakinithi o agent epistrefi true an iparxi kai
	 * false an den iparxi
	 *
	 * @param cur_pos
	 * @param next_pos
	 * @return
	 */
	public static boolean checkForWall(int cur_pos, int next_pos) {
		for (int i = 0; i < wall.size(); i++) {
			if (((cur_pos == wall.get(i).cell1) && (next_pos ==
wall.get(i).cell2))
				|| ((cur_pos == wall.get(i).cell2) &&
(next_pos == wall
					.get(i).cell1)))
				return true;
		}
		return false;
	}

	/**
	 * methodos i opoia dimiourga to gridworld
	 * @param filename
	 */
	public static void makeGridworld(String filename) {
		BufferedReader in = null;
		String[] splitline;

		try {
			in = new BufferedReader(new FileReader(filename));
		} catch (FileNotFoundException e) {
```

```
		e.printStackTrace();
	}
	String line = null;

	// grammes tou gridworld
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	size_x = Integer.parseInt(splitline[1]);

	// stiles tou gridworld
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	size_y = Integer.parseInt(splitline[1]);

	// actions
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	actions = Integer.parseInt(splitline[1]);

	// number of start position
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	numOfStartPosition = Integer.parseInt(splitline[1]);
	pasLoc = new int[numOfStartPosition];

	// start positions
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	String[] temp = splitline[1].split(",");
	for (int i = 0; i < temp.length; i++)
		setPasLoc(i, Integer.parseInt(temp[i]));

	// number of destinations
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
```

```
                splitline = line.split("=");
                numOfGoals = Integer.parseInt(splitline[1]);
                destLoc = new int[numOfGoals];

                // destinations
                try {
                      line = in.readLine();
                } catch (IOException e) {
                      e.printStackTrace();
                }
                splitline = line.split("=");
                temp = splitline[1].split(",");
                for (int i = 0; i < temp.length; i++)
                      setDestLoc(i, Integer.parseInt(temp[i]));

                // walls
                try {
                      line = in.readLine();
                } catch (IOException e) {
                      e.printStackTrace();
                }
                splitline = line.split("=");
                temp = splitline[1].split(",");
                String[] tmp;
                for (int i = 0; i < temp.length; i++) {
                      tmp = temp[i].split("-");
                      setWall(Integer.parseInt(tmp[0]),
Integer.parseInt(tmp[1]));
                }

                // reward for wall hit
                try {
                      line = in.readLine();
                } catch (IOException e) {
                      e.printStackTrace();
                }
                splitline = line.split("=");
                wall_hit = Integer.parseInt(splitline[1]);

                // reward for simple move
                try {
                      line = in.readLine();
                } catch (IOException e) {
                      e.printStackTrace();
                }
                splitline = line.split("=");
                simple_move = Integer.parseInt(splitline[1]);

                // reward for wrong get
                try {
                      line = in.readLine();
                } catch (IOException e) {
                      e.printStackTrace();
                }
                splitline = line.split("=");
                wrong_get = Integer.parseInt(splitline[1]);

                // reward for wrong put
```

```
			try {
				line = in.readLine();
			} catch (IOException e) {
				e.printStackTrace();
			}
			splitline = line.split("=");
			wrong_put = Integer.parseInt(splitline[1]);

			// reward for succes
			try {
				line = in.readLine();
			} catch (IOException e) {
				e.printStackTrace();
			}
			splitline = line.split("=");
			success = Integer.parseInt(splitline[1]);

			try {
				line = in.readLine();
			} catch (IOException e) {
				e.printStackTrace();
			}

			setStates();

		}

		public static void main(String[] args) {
			SA_Gridworld.makeGridworld("file.txt");
		}
}
```

Environment.java

```
import java.util.ArrayList;
import java.util.Random;

/**
 * abstract class for envrironment
 *
 * @author Giannis
 *
 */
public abstract class Environment {
	public static Random r = new Random();
	private double[][] Q;
	private ArrayList<State_Variables>[] episodes;
	private int[] episodeSteps;
	private double[] timePerEpisode;
	private double discount_factor;
	private double learning_rate;
	private double lr0 = 0.8;
	private double epsilon0 = 0.5;
	private double epsilon;
```

```
	private double timeConst;
	private double temperature;// xrisimopoieite sto boltzmann
exploration
	private double coolingRate; // xrisimopoieite gia na miwnete to
temperature
	private double offset = 1; // xrisimopieite gia tin allagi
thermokrasias

	/******** methodoi set,get,add gia tis private metavlites *****/

	/**
	 * methodos i opoia dimiourga ton pinaka Q
	 */
	abstract void makeQ();

	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou Q
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	abstract void setQValue(int i, int j, double value);

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
tou Q
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	abstract double getQValue(int i, int j);

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou Q to
neo value
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	abstract void addQValue(int i, int j, double value);

	/**
	 * methodos i opoia dimiourga ena arrylist gia ta episodia
	 *
	 * @param max_episode
	 */
	abstract void makeNewEpisodes(int max_episode);

	/**
	 * methodos i opoia prostheti ena neo state sto episodio
	 *
	 * @param i
	 * @param s
	 */
	abstract void addEpisodeState(int i, State_Variables s);
```

```
        /**
         * methodos i opoia dimiourga ton pinaka episodeSteps
         *
         * @param max_episode
         */
        abstract void makeNewEpisodeSteps(int max_episode);

        /**
         * methodos i opoia dimiourgei ton pinaka episodeSteps
         *
         * @param max_episode
         */
        abstract void makeNewTimePerEpisode(int max_episode);

        /**
         * methodos i opoia dini timi sta steps tou episodiou i
         *
         * @param i
         * @param steps
         */
        abstract void setTimePerEpisode(int i, double time);

        /**
         * methodos i opoia epistrefi ton pinaka timePerEpisode
         *
         * @return
         */
        abstract double[] getTimePerEpisode();

        /**
         * methodos i opoia epistrefi to xrono pou xriastike to
episodiou i
         *
         * @param i
         * @return
         */
        abstract double getTimePerEpisode(int i);

        /**
         * methodos i opoia epistrefi ton pinaka episodeSteps
         *
         * @return
         */
        abstract int[] getEpisodeSteps();

        /**
         * methodos i opoia dini timi sta steps tou episodiou i
         *
         * @param i
         * @param steps
         */
        abstract void setEpisodeSteps(int i, int steps);

        /**
         * methodos i opoia epistrefi ta steps tou episodiou i
         *
         * @param i
```

```
     * @return
     */
    abstract int getEpisodeSteps(int i);

    /**
     * methodos i opoia af3ani kata 1 ta steps tou episodiou i
     *
     * @param i
     */
    abstract void addEpisodeSteps(int i);

    /**
     * methodos i opoia epistrefi ta state enos episodiou
     *
     * @param i
     * @return
     */
    abstract ArrayList<State_Variables> getEpisodeStates(int i);

    /**
     * methodos i opoia kathorizei to learning rate
     *
     * @param num
     */
    abstract void setLearningRate(double num);

    /**
     * methodos i opoia epistrefi to learning rate
     *
     * @return
     */
    abstract double getLearningRate();

    /**
     * methodos i opoia kathorizei to discount factor
     *
     * @param num
     */
    abstract void setDiscountFactor(double num);

    /**
     * methodos i opoia epistrefi to discount factor
     *
     * @return
     */
    abstract double getDiscountFactor();

    /**
     * methodos i opoia kathorizei to epsilon
     *
     * @param num
     */
    abstract void setEpsilon(double num);

    /**
     * methodos i opoia epistrefi to epsilon
     *
     * @return
```

```
     */
    abstract double getEpsilon();

    /**
     * methodos i opoia kathorizi to temperature
     *
     * @param temperature
     */
    abstract void setTemperature(double temperature);

    /**
     * methodos i opoia epistrefi to temperature
     *
     * @return
     */
    abstract double getTemperature();

    /**
     * methodos i opoia kathorizi to coolingRate
     *
     * @param coolingRate
     *            the coolingRate to set
     */
    abstract void setCoolingRate(double coolingRate);

    /**
     * methodos i opoia epistrefi to coolingRate
     *
     * @return the coolingRate
     */
    abstract double getCoolingRate();

    /*****************************************/

    /**
     * methodos i opoia arxikopoiei to Q
     */
    abstract void initQ(double num);

    /**
     * methodos i opoia ipologizi to index apo ena vector (vriski
tin katastasi
     * pou imaste sto perivallon,tin grammi tou Q table)
     *
     * @param s
     * @return
     */
    abstract int findIndex(State_Variables s);

    /**
     * methodos i opoia vriski tin thesi tou max action se ena state
     *
     * @param index
     * @return
     */
    abstract int findMaxQ(int index);

    /**
```

```
         * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
         * sigkekrimeni katastasi (state) simfona me tin policy e-greedy
         *
         * @param index
         * @return
         */
        abstract int chooseAction(int index);

        /******************Boltzmann Exploration *******************/

        /**
         * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
         * sigkekrimeni katastasi (state) simfona me tin Boltzmann
         * Exploration
         * @param index
         * @return
         */
        abstract int chooseActionWithBoltzmannExploration(int index);

        /**
         * methodos i opoia kanonikopoiei tis pithanottites enos pinaka
         *
         * @param index
         */
        abstract double[] normalizeProbability(double[] table);

        /**
         * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
         * ena pinaka pithanotitwn
         *
         * @param index
         * @return
         */
        abstract double findMinNegativeValue(double[] table);

        /**
         * methodos i opoia epilegi mia kinisi (action) vasi tis
         * pithanotitas p(s,a)
         * @param index
         * @return
         */
        abstract int chooseProbAction(double[] table);

        /**
         * methodos i opoia vriski pia thesi tou pinaka exi ti
         * megaliteri timi
         * @param table
         * @return
         */
        abstract int findMax(double[] table);



        /**
         * methodos i opoia allazi to temperature vasi kapiou cooling
rate
         */
        abstract void changeTemperature(int numStep);
```

```
	/**
	 * methodos i opoia allazti to epsilon0 se kathe iteration
	 *
	 * @param i
	 * @param max_iteration
	 */
	abstract void changeEpsilon0(int i, int max_episode);

	/**
	 * methodos i opoia allazi to epsilon se kathe step tou
episodiou
	 *
	 * @param episode
	 */
	abstract void changeEpsilon(int episode);

	/**
	 * methodos i opoia allazi to learning rate se kathe step tou
episodiou
	 *
	 * @param episode
	 */
	abstract void changeLearningRate(int episode);

	/**
	 * methodos i opoia allazi to timeConst gia kathe epsilon0
	 */
	abstract void changeTimeConst();

	/**
	 * methodos i opoia ekteli to action apo to state s, vriski tin
epomeni
	 * katastasi (state) kai epistrefi to reward
	 *
	 * @param s
	 * @param tempState
	 * @param action
	 * @return
	 */
	abstract int takeAction(State_Variables s, State_Variables
tempState,
			int action);

}
```

SA_Taxi_Problem.java

```
import java.util.ArrayList;
import java.util.Random;

/**
 * klasi i opoia epiliei to taxi problem
 */
public class SA_Taxi_Problem extends Environment{

	public static Random r = new Random();
	private double[][] Q;
	private ArrayList<State_Variables>[] episodes;
	private int[] episodeSteps;
	private double[] timePerEpisode;
	private double discount_factor;
	private double learning_rate;
	private double lr0 = 0.8;
	private double epsilon0 = 0.5;
	private double epsilon;
	private double timeConst;
	private double temperature;// xrisimopoieite sto boltzmann
exploration
	private double coolingRate; // xrisimopoieite gia na miwnete to
temperature
	private double offset = 1; // xrisimopieite gia tin allagi
thermokrasias

	/***** methodoi set,get,add gia tis private metavlites ********/

	/**
	 * methodos i opoia dimiourga ton pinaka Q
	 */
	public void makeQ() {
		Q = new double[SA_Gridworld.states][SA_Gridworld.actions];
	}

	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou Q
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setQValue(int i, int j, double value) {
		this.Q[i][j] = value;
	}

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou Q
	 * @param i
	 * @param j
	 * @return
	 */
```

```
public double getQValue(int i, int j) {
	return this.Q[i][j];
}

/**
 * methodos i opoia prostheti stin sigkekrimeni timi tou Q to
 * neo value
 * @param i
 * @param j
 * @param value
 */
public void addQValue(int i, int j, double value) {
	this.Q[i][j] += value;
}

/**
 * methodos i opoia dimiourga ena arrylist gia ta episodia
 *
 * @param max_episode
 */
public void makeNewEpisodes(int max_episode) {
	this.episodes = new ArrayList[max_episode];
	for (int i = 0; i < max_episode; i++) {
		this.episodes[i] = new ArrayList<State_Variables>();
	}
}

/**
 * methodos i opoia prostheti ena neo state sto episodio
 *
 * @param i
 * @param s
 */
public void addEpisodeState(int i, State_Variables s) {
	this.episodes[i].add(s);
}

/**
 * methodos i opoia dimiourga ton pinaka episodeSteps
 *
 * @param max_episode
 */
public void makeNewEpisodeSteps(int max_episode) {
	this.episodeSteps = new int[max_episode];
	// initialize episodeSteps
	for (int i = 0; i < max_episode; i++) {
		// this.episodeSteps[i] = 0;
		this.setEpisodeSteps(i, 0);
	}
}

/**
 * methodos i opoia dimiourgei ton pinaka episodeSteps
 *
 * @param max_episode
 */
public void makeNewTimePerEpisode(int max_episode) {
	this.timePerEpisode = new double[max_episode];
```

```
		// initialize episodeSteps
		for (int i = 0; i < max_episode; i++) {
			// this.episodeSteps[i] = 0;
			this.setTimePerEpisode(i, 0.0);
		}
	}

	/**
	 * methodos i opoia dini timi sta steps tou episodiou i
	 *
	 * @param i
	 * @param steps
	 */
	public void setTimePerEpisode(int i, double time) {
		this.timePerEpisode[i] = time;
	}

	/**
	 * methodos i opoia epistrefi ton pinaka timePerEpisode
	 *
	 * @return
	 */
	public double[] getTimePerEpisode() {
		return this.timePerEpisode;

	}

	/**
	 * methodos i opoia epistrefi to xrono pou xriastike to
	 * episodiou i
	 * @param i
	 * @return
	 */
	public double getTimePerEpisode(int i) {
		return this.timePerEpisode[i];
	}

	/**
	 * methodos i opoia epistrefi ton pinaka episodeSteps
	 *
	 * @return
	 */
	public int[] getEpisodeSteps() {
		return this.episodeSteps;

	}

	/**
	 * methodos i opoia dini timi sta steps tou episodiou i
	 *
	 * @param i
	 * @param steps
	 */
	public void setEpisodeSteps(int i, int steps) {
		this.episodeSteps[i] = steps;
	}
```

```
/**
 * methodos i opoia epistrefi ta steps tou episodiou i
 *
 * @param i
 * @return
 */
public int getEpisodeSteps(int i) {
	return this.episodeSteps[i];
}

/**
 * methodos i opoia af3ani kata 1 ta steps tou episodiou i
 *
 * @param i
 */
public void addEpisodeSteps(int i) {
	this.episodeSteps[i]++;
}

/**
 * methodos i opoia epistrefi ta state enos episodiou
 *
 * @param i
 * @return
 */
public ArrayList<State_Variables> getEpisodeStates(int i) {
	return this.episodes[i];
}

/**
 * methodos i opoia kathorizei to learning rate
 *
 * @param num
 */
public void setLearningRate(double num) {
	this.learning_rate = num;
}

/**
 * methodos i opoia epistrefi to learning rate
 *
 * @return
 */
public double getLearningRate() {
	return this.learning_rate;
}

/**
 * methodos i opoia kathorizei to discount factor
 *
 * @param num
 */
public void setDiscountFactor(double num) {
	this.discount_factor = num;
}

/**
 * methodos i opoia epistrefi to discount factor
```

```
 *
 * @return
 */
public double getDiscountFactor() {
	return this.discount_factor;
}

/**
 * methodos i opoia kathorizei to epsilon
 *
 * @param num
 */
public void setEpsilon(double num) {
	this.epsilon = num;
}

/**
 * methodos i opoia epistrefi to epsilon
 *
 * @return
 */
public double getEpsilon() {
	return this.epsilon;
}

/**
 * methodos i opoia kathorizi to temperature
 *
 * @param temperature
 */
public void setTemperature(double temperature) {
	this.temperature = temperature;
}




/**
 * methodos i opoia epistrefi to temperature
 *
 * @return
 */
public double getTemperature() {
	return temperature;
}

/**
 * methodos i opoia kathorizi to coolingRate
 *
 * @param coolingRate
 *            the coolingRate to set
 */
public void setCoolingRate(double coolingRate) {
	this.coolingRate = coolingRate;
}
```

```
	/**
	 * methodos i opoia epistrefi to coolingRate
	 *
	 * @return the coolingRate
	 */
	public double getCoolingRate() {
		return coolingRate;
	}
	/**
	 * methodos i opoia arxikopoiei to Q
	 */
	public void initQ(double num) {
		for (int i = 0; i < SA_Gridworld.states; i++) {
			for (int j = 0; j < SA_Gridworld.actions; j++) {
				// this.Q[i][j] = num;
				this.setQValue(i, j, num);
			}
		}
	}
	/**
	 * methodos i opoia ipologizi to index apo ena vector (vriski
	 * tin katastasi pou imaste sto perivallon,tin grammi tou Q
	 * table)
	 * @param s
	 * @return
	 */
	public int findIndex(State_Variables s) {
		return (s.taxi * SA_Gridworld.numOfStartPosition *
SA_Gridworld.numOfGoals
				+ s.passenger * SA_Gridworld.numOfGoals +
s.destination);
	}
	/**
	 * methodos i opoia vriski tin thesi tou max action se ena state
	 *
	 * @param index
	 * @return
	 */
	public int findMaxQ(int index) {
		double temp = this.getQValue(index, 0);
		int pos = 0;
		for (int i = 1; i < SA_Gridworld.actions; i++) {
			if (this.getQValue(index, i) > temp) {
				temp = this.getQValue(index, i);
				pos = i;
			}
		}
		return pos;
	}
```

```
	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
	 * sigkekrimeni katastasi (state) simfona me tin policy e-greedy
	 *
	 * @param index
	 * @return
	 */
	public int chooseAction(int index) {
		if (r.nextDouble() < this.epsilon)
			return (r.nextInt(SA_Gridworld.actions));
		return this.findMaxQ(index);
	}

	/************ Boltzmann Exploration *****************/

	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
	 * sigkekrimeni katastasi (state) simfona me tin Boltzmann
	 * Exploration
	 * @param index
	 * @return
	 */
	public int chooseActionWithBoltzmannExploration(int index) {
		double[] probTable = new double[SA_Gridworld.actions];
		double sum = 0.0;
		for (int i = 0; i < SA_Gridworld.actions; i++)
			sum += Math.exp((this.getQValue(index, i) /
this.getTemperature()));
		for (int i = 0; i < SA_Gridworld.actions; i++)
			probTable[i] = Math.exp((this.getQValue(index, i) /
this
					.getTemperature()))
					/ sum;

		probTable = this.normalizeProbability(probTable);
		return this.chooseProbAction(probTable);//
this.findMax(probTable);
	}


	/**
	 * methodos i opoia kanonikopoiei tis pithanottites enos pinaka
	 *
	 * @param index
	 */
	public double[] normalizeProbability(double[] table) {
		double minNegative = this.findMinNegativeValue(table);
		double increment = Math.abs(2 * minNegative);
		double sum = 0;
		for (int i = 0; i < table.length; i++)
			table[i] += increment;
		for (int i = 0; i < table.length; i++)
			sum += table[i];
		for (int i = 0; i < table.length; i++)
			table[i] /= sum;

		return table;
	}
```

```java
/**
 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
 * ena pinaka pithanotitwn
 *
 * @param index
 * @return
 */
public double findMinNegativeValue(double[] table) {
	double temp = 0;
	for (int i = 0; i < table.length; i++) {
		if (table[i] < temp) {
			temp = table[i];
		}
	}
	return temp;
}

/**
 * methodos i opoia epilegi mia kinisi (action) vasi tis)
 * pithanotitas p(s,a
 * @param index
 * @return
 */
public int chooseProbAction(double[] table) {
	double offset = 0;
	double rand = r.nextDouble();
	for (int i = 0; i < table.length; i++) {
		offset += table[i];
		if (offset > rand)
			return i;
	}

	 // en tha erti pote dame
	return r.nextInt(table.length);
}

/**
 * methodos i opoia vriski pia thesi tou pinaka exi ti
 * megaliteri timi
 * @param table
 * @return
 */
public int findMax(double[] table) {
	int maxPos = 0;
	double max = table[0];
	for (int i = 0; i < table.length; i++) {
		if (table[i] > max) {
			max = table[i];
			maxPos = i;
		}
	}
	return maxPos;

}
```

```
	/**
	 * methodos i opoia allazi to temperature vasi kapiou cooling
	 * rate
	 */
	public void changeTemperature(int numStep) {
		this.setTemperature(this.offset
				+ (this.temperature *
Math.pow(this.coolingRate, numStep)));
	}

	/*************************************************************/

	/**
	 * methodos i opoia allazti to epsilon0 se kathe iteration
	 *
	 * @param i
	 * @param max_iteration
	 */
	public void changeEpsilon0(int i, int max_episode) {
		this.epsilon0 = this.epsilon0 * Math.exp(-((double) i /
max_episode));
	}

	/**
	 * methodos i opoia allazi to epsilon se kathe step tou
	 * episodiou
	 * @param episode
	 */
	public void changeEpsilon(int episode) {
		this.epsilon = this.epsilon0
				* Math
						.exp(-((double)
this.getEpisodeSteps(episode) / this.timeConst));
	}
	/**
	 * methodos i opoia allazi to learning rate se kathe step tou
	 * episodiou
	 * @param episode
	 */
	public void changeLearningRate(int episode) {
		this.learning_rate = this.lr0
				* Math.exp(-((double)
this.getEpisodeSteps(episode) / 1000000));
		// this.learning_rate=this.lr0*((double)1-
((double)this.getEpisodeSteps(episode)/1000000));
	}
	/**
	 * methodos i opoia allazi to timeConst gia kathe epsilon0
	 */
	public void changeTimeConst() {
		this.timeConst = Math.log(0.01 / this.epsilon0);
	}
```

```
	/**
	 * methodos i opoia ekteli to action apo to state s, vriski tin
	 * epomeni katastasi (state) kai epistrefi to reward
	 *
	 * @param s
	 * @param tempState
	 * @param action
	 * @return
	 */
	public int takeAction(State_Variables s, State_Variables
tempState,
			int action) {

		int taxi = s.taxi;
		int passenger = s.passenger;
		int destination = s.destination;
		int reward = -1;
		switch (action) {
		// north (move up)
		case 0: // ektos orion tou grid
			if ((!((s.taxi - SA_Gridworld.size_x) < 0))
					&& (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi - SA_Gridworld.size_x))))) {
				taxi = s.taxi - SA_Gridworld.size_x;
				reward = SA_Gridworld.simple_move;
			} else {
				reward = SA_Gridworld.wall_hit;
			}

			break;
		// south (move down)
		case 1: // ektos orion tou grid
			if ((!((s.taxi + SA_Gridworld.size_y) >=
(SA_Gridworld.size_x * SA_Gridworld.size_y)))
					&& (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi + SA_Gridworld.size_y))))) {
				taxi = s.taxi + SA_Gridworld.size_y;
				reward = SA_Gridworld.simple_move;
			} else {
				reward = SA_Gridworld.wall_hit;
			}
			break;
		// east (move right)
		case 2: // hit the wall or ektos orion tou grid
			if ((!((s.taxi % SA_Gridworld.size_y) ==
(SA_Gridworld.size_y - 1)))
					&& (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi + 1))))) {
				taxi = s.taxi + 1;
				reward = SA_Gridworld.simple_move;
			} else {
				reward = SA_Gridworld.wall_hit;
			}
			break;
		// west (move left)
		case 3: // hit the wall or ektos orion tou grid
			if ((!((s.taxi % SA_Gridworld.size_y) == 0))
```

```
                                    && (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi - 1))))) {
                              taxi = s.taxi - 1;
                              reward = SA_Gridworld.simple_move;
                        } else {
                              reward = SA_Gridworld.wall_hit;
                        }
                        break;
                  // pickup the passenger
                  case 4: // location without a passenger or passenger
already in the taxi
                        if ((s.taxi != SA_Gridworld.getPasLoc(s.passenger))
                                    || (SA_Gridworld.getPasLoc(s.passenger)
== -1)) {
                              reward = SA_Gridworld.wrong_get;
                        } else {
                              passenger = 0;
                              reward = SA_Gridworld.simple_move;
                        }
                        break;
                  // put down the passenger
                  case 5: // wrong destination or passenger not in the taxi
                        if ((s.taxi !=
SA_Gridworld.getDestLoc(s.destination))
                                    || (SA_Gridworld.getPasLoc(s.passenger)
!= -1)) {
                              reward =SA_Gridworld.wrong_put;
                        } else { // Successful passenger delivery
                              passenger = s.destination + 1;
                              reward = SA_Gridworld.success;
                        }
                        break;
                  default:
                        System.err.println("Invalid number for
action..Exiting..");
                        System.exit(-1);
                  }
                  tempState.taxi = taxi;
                  tempState.passenger = passenger;
                  tempState.destination = destination;
                  return reward;
            }

            /**
             * @param args
             */
            public static void main(String[] args) {
                  // TODO Auto-generated method stub
            }

}
```

SA_Graph.java

```
import java.util.ArrayList;

/**
 * klasi i opoia antiprosopevi ton grafo
 *
 * @author giannis
 *
 */
public class SA_Graph extends Graph {

	ArrayList<Node> graph = new ArrayList<Node>();

	public final int MaxRoot = 0;
	public final int MaxGet = 1;
	public final int MaxPut = 2;
	public final int Pickup = 3;
	public final int Putdown = 4;
	public final int MaxNavigate = 5;
	public final int North = 6;
	public final int East = 7;
	public final int South = 8;
	public final int West = 9;

	public final int Qget = 10;
	public final int Qput = 11;
	public final int Qpickup = 12;
	public final int QNavigateForGet = 13;
	public final int QNavigateForPut = 14;
	public final int Qputdown = 15;
	public final int Qnorth = 16;
	public final int Qeast = 17;
	public final int Qsouth = 18;
	public final int Qwest = 19;

	public final int numMaxNodes = 10;
	public final int numQnodes = 10;

	/**
	 * methodos i opoia dimiourgi ton grafo
	 */
	public void makeGraph(int states,int num) {

		// add the MaxNodes exoume 4 composite kai 6 primitive
		for (int i = 0; i < this.numMaxNodes; i++)
			this.addMaxNode(SA_Gridworld.states,num);

		// add the Qnodes
		for (int i = 0; i < this.numQnodes; i++)
			this.addQnode(SA_Gridworld.states);

		// add childrens
		this.addChild(this.MaxRoot, this.Qget);
		this.addChild(this.MaxRoot, this.Qput);
```

```
            this.addChild(this.MaxGet, this.Qpickup);
            this.addChild(this.MaxGet, this.QNavigateForGet);
            this.addChild(this.MaxPut, this.QNavigateForPut);
            this.addChild(this.MaxPut, this.Qputdown);
            this.addChild(this.MaxNavigate, this.Qnorth);
            this.addChild(this.MaxNavigate, this.Qeast);
            this.addChild(this.MaxNavigate, this.Qsouth);
            this.addChild(this.MaxNavigate, this.Qwest);
            this.addChild(this.Qget, this.MaxGet);
            this.addChild(this.Qput, this.MaxPut);
            this.addChild(this.Qpickup, this.Pickup);
            this.addChild(this.QNavigateForGet, this.MaxNavigate);
            this.addChild(this.QNavigateForPut, this.MaxNavigate);
            this.addChild(this.Qputdown, this.Putdown);
            this.addChild(this.Qnorth, this.North);
            this.addChild(this.Qeast, this.East);
            this.addChild(this.Qsouth, this.South);
            this.addChild(this.Qwest, this.West);

            // add actions
            ((MaxNode)
this.graph.get(this.Pickup)).setAction(SA_Gridworld.pickup);
            ((MaxNode)
this.graph.get(this.Putdown)).setAction(SA_Gridworld.putdown);
            ((MaxNode)
this.graph.get(this.North)).setAction(SA_Gridworld.north);
            ((MaxNode)
this.graph.get(this.East)).setAction(SA_Gridworld.east);
            ((MaxNode)
this.graph.get(this.South)).setAction(SA_Gridworld.south);
            ((MaxNode)
this.graph.get(this.West)).setAction(SA_Gridworld.west);

      }

      /**
       * methodos i opoia prostheti ena MaxNode sto grafo
       *
       * @param i
       * @param num
       */
      public void addMaxNode(int i, int num) {
            switch (num) {
            case 0:// gia ton algorithmo MAXQ-Q
                  MaxNode tmp = new MaxNode(i);
                  this.graph.add(tmp);
                  break;
            case 1:// gia ton algorithmo MAXQ-PHC
                  MaxNodePHC tmp1 = new MaxNodePHC(i);
                  this.graph.add(tmp1);
                  break;
            case 2:// gia ton algorithmo MAXQ-WoLF-PHC
                  MaxNodeWoLFPHC tmp2 = new MaxNodeWoLFPHC(i);
                  this.graph.add(tmp2);
                  break;
            default:
                  System.err.println("Invalid number for
num..Exiting..");
```

```
                        System.exit(-1);
                }
        }

        /**
         * methodos i opoia prostheti ena Qnode sto grafo
         *
         * @param i
         * @param j
         */
        public void addQnode(int i) {
                Qnode tmp = new Qnode(i);
                this.graph.add(tmp);
        }

        /**
         * methodos i opoia prostheti ena paidi se ena komvo
         *
         * @param parent
         * @param child
         */
        public void addChild(int parent, int child) {
                this.graph.get(parent).children.add(child);
        }

        /**
         * @param args
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub

        }

}
```

Graph.java

```
import java.util.ArrayList;
/**
 * afairetiki klasi pou antiprosopevi ton grafo
 *
 * @author Giannis
 *
 */
public abstract class Graph {
        ArrayList<Node> graph = new ArrayList<Node>();

        public final int MaxRoot = 0;
        public final int MaxGet = 1;
        public final int MaxPut = 2;
        public final int Pickup = 3;
        public final int Putdown = 4;
        public final int MaxNavigate = 5;
        public final int North = 6;
        public final int East = 7;
```

```
        public final int South = 8;
        public final int West = 9;

        public final int Qget = 10;
        public final int Qput = 11;
        public final int Qpickup = 12;
        public final int QNavigateForGet = 13;
        public final int QNavigateForPut = 14;
        public final int Qputdown = 15;
        public final int Qnorth = 16;
        public final int Qeast = 17;
        public final int Qsouth = 18;
        public final int Qwest = 19;

        public final int numMaxNodes = 10;
        public final int numQnodes = 10;

        /**
         * methodos i opoia dimiourgi ton grafo
         */
        abstract void makeGraph(int states, int num);

        /**
         * methodos i opoia prostheti ena MaxNode sto grafo
         *
         * @param i
         * @param num
         */
        abstract void addMaxNode(int i, int num);

        /**
         * methodos i opoia prostheti ena Qnode sto grafo
         *
         * @param i
         * @param j
         */
        abstract void addQnode(int i);

        /**
         * methodos i opoia prostheti ena paidi se ena komvo
         *
         * @param parent
         * @param child
         */
        abstract void addChild(int parent, int child);
}
```

```
import java.util.ArrayList;
import java.util.Random;

/**
 * klasis i opoia ilopoiei to MaxQ Value function decomposition
 *
 * @author Giannis
 *
 */
public class MaxQ extends SA_Graph {


	public Random r = new Random();
	private int[] episodeSteps;
	private double[] timePerEpisode;
	private double discount_factor;
	private double epsilon;

	/**** methodoi set,get,add gia tis private metavlites ********/

	/**
	 * methodos i opoia dini timi sta steps tou episodiou i
	 *
	 * @param i
	 * @param steps
	 */
	public void setEpisodeSteps(int i, int steps) {
		this.episodeSteps[i] = steps;
	}

	/**
	 * methodos i opoia epistrefi ton pinaka episodeSteps
	 *
	 * @return
	 */
	public int[] getEpisodeSteps() {
		return this.episodeSteps;

	}

	/**
	 * methodos i opoia epistrefi ta steps tou episodiou i
	 *
	 * @param i
	 * @return
	 */
	public int getEpisodeSteps(int i) {
		return this.episodeSteps[i];
	}
```

```java
/**
 * methodos i opoia af3ani kata 1 ta steps tou episodiou i
 *
 * @param i
 */
public void addEpisodeSteps(int i) {
	this.episodeSteps[i]++;
}

/**
 * methodos i opoia dini timi sta steps tou episodiou i
 *
 * @param i
 * @param steps
 */
public void setTimePerEpisode(int i, double time) {
	this.timePerEpisode[i] = time;
}

/**
 * methodos i opoia epistrefi ton pinaka timePerEpisode
 *
 * @return
 */
public double[] getTimePerEpisode() {
	return this.timePerEpisode;

}

/**
 * methodos i opoia epistrefi to xrono pou xriastike to
 * episodiou i
 * @param i
 * @return
 */
public double getTimePerEpisode(int i) {
	return this.timePerEpisode[i];
}

/**
 * methodos i opoia kathorizei to discount factor
 *
 * @param num
 */
public void setDiscountFactor(double num) {
	this.discount_factor = num;
}

/**
 * methodos i opoia epistrefi to discount factor
 *
 * @return
 */
public double getDiscountFactor() {
	return this.discount_factor;
}
```

```
	/**
	 * methodos i opoia kathorizei to epsilon
	 *
	 * @param num
	 */
	public void setEpsilon(double num) {
		this.epsilon = num;
	}

	/**
	 * methodos i opoia epistrefi to epsilon
	 *
	 * @return
	 */
	public double getEpsilon() {
		return this.epsilon;
	}

	/*******************************************************/

	/**
	 * methodos i opoia dimiourgei ton pinaka episodeSteps
	 *
	 * @param max_episode
	 */
	public void makeNewEpisodeSteps(int max_episode) {
		this.episodeSteps = new int[max_episode];
		// initialize episodeSteps
		for (int i = 0; i < max_episode; i++) {
			this.setEpisodeSteps(i, 0);
		}
	}

	/**
	 * methodos i opoia dimiourgei ton pinaka episodeSteps
	 *
	 * @param max_episode
	 */
	public void makeNewTimePerEpisode(int max_episode) {
		this.timePerEpisode = new double[max_episode];
		// initialize episodeSteps
		for (int i = 0; i < max_episode; i++) {
			// this.episodeSteps[i] = 0;
			this.setTimePerEpisode(i, 0.0);
		}
	}



	/**
	 * methodos i opoia arxikopoiei ta actions se ola ta primitive
	 * MaxNode
	 */
	public void initActionsToPrimitive() {
		((MaxNode)
this.graph.get(this.Pickup)).setAction(SA_Gridworld.pickup);
		((MaxNode)
this.graph.get(this.Putdown)).setAction(SA_Gridworld.putdown);
```

```
            ((MaxNode)
this.graph.get(this.North)).setAction(SA_Gridworld.north);
            ((MaxNode)
this.graph.get(this.South)).setAction(SA_Gridworld.south);
            ((MaxNode)
this.graph.get(this.East)).setAction(SA_Gridworld.east);
            ((MaxNode)
this.graph.get(this.West)).setAction(SA_Gridworld.west);
      }

      /**
       * methodos i opoia ipologizi to index apo ena vector (vriski
       * tin katastasi pou imaste sto perivallon,tin grammi tou Q
       * table)
       * @param s
       * @return
       */
      public int findIndex(State_Variables s) {
            return (s.taxi * SA_Gridworld.numOfStartPosition *
SA_Gridworld.numOfGoals
                        + s.passenger * SA_Gridworld.numOfGoals +
s.destination);
      }

      /**
       * methodos i opoia vriski tin timi (V(i,s)) gia ena MaxNode pou
       * ine composite (not primitive) - Greedy Execution of the MAXQ
       * Graph (evaluateMaxNode(i,s))
       *
       * @param maxnode
       * @param index
       *            ( dixni tin katastasi (state) pou vriskomaste)
       * @return
       */
      public NodeValue evaluateMaxNode(int node, int index) {

            ArrayList<NodeValue> nodeValue = new
ArrayList<NodeValue>();

            int maxpos = 0;

            if (this.graph.get(node) instanceof Qnode) {
                  node = ((Qnode)
this.graph.get(node)).children.get(0);
            }

            if (((MaxNode) this.graph.get(node)).isPrimitive()) {
                  NodeValue tmp = new NodeValue(((MaxNode)
this.graph.get(node))
                              .getValueFunctForPrimitive(index),
node);
                  return tmp;
            } else {
                  for (int i = 0; i <
this.graph.get(node).children.size(); i++) {
                        NodeValue tmp = this.evaluateMaxNode(

      this.graph.get(node).children.get(i), index);
```

```
                        nodeValue.add(tmp);
                  }
                  maxpos = this.findMax(nodeValue);
                  nodeValue.get(maxpos).value += ((Qnode) this.graph
                              .get(((MaxNode)
this.graph.get(node)).children.get(maxpos)))
                              .getCvalue(index);

                  return nodeValue.get(maxpos);
            }

      }

      /**
       * methodos i opoia vriski pio node apo ena arraylist exi ti pio
       * megali timi (Value) V(j,s)) - xrisimopoieitai apo tin
       * evaluateMaxNode
       * @param nodeValue
       * @param node
       * @param index
       * @return
       */
      public int findMax(ArrayList<NodeValue> nodeValue) {
            double max = 0;
            int maxpos = 0;
            double temp = 0;
            if (nodeValue.isEmpty()) {
                  System.err.println("Error in evaluating Max
Node\nExiting..");
                  System.exit(-1);
            }
            max = nodeValue.get(0).value;
            for (int i = 1; i < nodeValue.size(); i++) {
                  temp = nodeValue.get(i).value;
                  if (temp > max) {
                        max = temp;
                        maxpos = i;
                  }
            }
            return maxpos;
      }

      /**
       * methodos i opoia elegxi an imaste se terminal state enos
       * subtask (MaxNode)
       *
       * @param node
       * @param s
       * @return
       */
      public boolean isTerminal(int node, State_Variables s) {
            switch (node) {
            case 0:// MaxRoot goal state
                  if ((SA_Gridworld.getPasLoc(s.passenger) ==
SA_Gridworld.getDestLoc(s.destination))
                              && (s.taxi ==
SA_Gridworld.getDestLoc(s.destination)))
                        return true;
```

```
                  return false;
            case 1:// MaxGet
                  if (SA_Gridworld.getPasLoc(s.passenger) == -1)
                        return true;
                  return false;
            case 2:// MaxPut
                  // goal state
                  if (((SA_Gridworld.getPasLoc(s.passenger) ==
SA_Gridworld.getDestLoc(s.destination)) && (s.taxi == SA_Gridworld
                              .getDestLoc(s.destination)))
                              || SA_Gridworld.getPasLoc(s.passenger)
!= -1)
                        // terminal state (to OR) (simeni oti o
passenger dn ine mesa
                        // sto
                        // taxi ara prepi na kamw get prwta)
                        return true;
                  return false;
            case 5:// MaxNavigate(t) goal state
                  if (((MaxNode) this.graph.get(node)).getDest() ==
s.taxi)
                        return true;
                  return false;
            default:
                  return false;
            }
      }

      /**
       * methodos i opoia epistrefi tin timi Value gia mia katastasi
       * enos node analoga me to an ine primitive i composite node
       *
       * @param maxNode
       * @param index
       * @return
       */
      public double findValue(int maxNode, int index) {
            if (!(this.graph.get(maxNode) instanceof Qnode))
                  if (((MaxNode)
this.graph.get(maxNode)).isPrimitive()) {

                        return (((MaxNode) this.graph.get(maxNode))

      .getValueFunctForPrimitive(index));
                  }
            NodeValue nValue = this.evaluateMaxNode(maxNode, index);

            return nValue.value;

      }
```

```
	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
	 * sigkekrimeni katastasi (state) simfona me tin policy e-greedy
	 *
	 * @param node
	 * @param index
	 * @return
	 */
	public int chooseAction(int node, int index) {
		if (r.nextDouble() < this.epsilon)
			return
(r.nextInt(this.graph.get(node).children.size()));
		return this.findMaxQvalue(node, index);
	}

	/****************** Boltzmann Exploration ******************/

	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
	 * sigkekrimeni katastasi (state) simfona me Boltzmann
	 * Exploration
	 * @param node
	 * @param index
	 * @return
	 */
	public int chooseActionWithBoltzmannExploration(int node, int
index) {
		double[] probTable = new
double[this.graph.get(node).children.size()];
		probTable = this.findQvalues(node, index);
		double sum = 0.0;
		for (int i = 0; i < probTable.length; i++)
			sum += Math.exp(probTable[i]
					/ ((MaxNode)
this.graph.get(node)).getTemperature());

		 for (int i = 0; i < probTable.length; i++)
			probTable[i] = Math.exp(probTable[i]
					/ ((MaxNode)
this.graph.get(node)).getTemperature())
					/ sum;


		probTable=this.normalizeProbability(probTable);

		return this.chooseProbAction(probTable);
	}

	/**
	 * methodos i opoia kanonikopoiei tis pithanottites enos pinaka
	 *
	 * @param index
	 */
	public double[] normalizeProbability(double[] table) {
		double minNegative = this.findMinNegativeValue(table);
		double increment = Math.abs(2 * minNegative);
		double sum = 0;
```

```
		for (int i = 0; i < table.length; i++)
			table[i]+= increment;
		for (int i = 0; i < table.length; i++)
			sum += table[i];
		for (int i = 0; i < table.length; i++)
			table[i]/=sum;

		return table;
	}

	/**
	 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
	 * ena pinaka pithanotitwn
	 *
	 * @param index
	 * @return
	 */
	public double findMinNegativeValue(double[] table) {
		double temp = 0;
		for (int i = 0; i < table.length; i++) {
			if (table[i] < temp) {
				temp = table[i];
			}
		}
		return temp;
	}

	/**
	 * methodos i opoia epilegi mia kinisi (action) vasi tis
	 * pithanotitas p(s,a)
	 * @param index
	 * @return
	 */
	public int chooseProbAction(double[] table) {
		double offset = 0;
		double rand = r.nextDouble();
		for (int i = 0; i < table.length; i++) {
			offset += table[i];
			if (offset > rand)
				return i;
		}
		//en tha erti potte dame
		return r.nextInt(table.length);

	}

	/**
	 * methodos i opoia vriski pia thesi tou pinaka exi ti
	 * megaliteri timi
	 * @param table
	 * @return
	 */
	public int findMax(double[] table) {
		int maxPos = 0;
		double max = table[0];
		for (int i = 1; i < table.length; i++) {
			if (table[i] > max) {
				max = table[i];
```

```
                        maxPos = i;
                  }
            }
            return maxPos;
      }

      /************ End of Boltzmann Exploration ***************/
      /**
       * methodos i opoia vriski pios Qnode exi ti megaliteri Expected
       * Discounted Cumulative Reward ( C(i,s,a) )
       *
       * @param node
       * @param index
       * @return
       */
      public int findMax(int node, int index) {
            double maxValue = ((Qnode)
this.graph.get(this.graph.get(node).children
                        .get(0))).getCvalue(index);
            int maxPos = 0;
            double temp = 0;
            for (int i = 1; i < this.graph.get(node).children.size();
i++) {
                  temp = ((Qnode) this.graph

      .get(this.graph.get(node).children.get(i)))
                              .getCvalue(index);
                  if (temp > maxValue) {
                        maxValue = temp;
                        maxPos = i;
                  }
            }
            return maxPos;
      }

      /**
       * methodos i opoia vriski pios Qnode apo ta children enos
       * Maxnode exi to max Q value
       *
       * @param maxNode
       * @param index
       * @return
       */
      public int findMaxQvalue(int maxNode, int index) {
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            table = this.findQvalues(maxNode, index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia vriski ta Q values gia kathe children enos
       * MaxNode
       * @param node
       * @param index
       * @return
       */
      public double[] findQvalues(int maxNode, int index) {
```

```
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            NodeValue nValue;
            for (int i = 0; i <
this.graph.get(maxNode).children.size(); i++) {
                  nValue =
this.evaluateMaxNode(this.graph.get(maxNode).children
                              .get(i), index);
                  table[i] = nValue.value;
            }

            for (int i = 0; i < table.length; i++)
                  table[i] += ((Qnode) this.graph

      .get(this.graph.get(maxNode).children.get(i)))
                              .getCvalue(index);

            return table;
      }

      /**
       * methodos i opoia vriski pios Qnode exi ti mikroteri Expected
       * Discounted Cumulative Reward ( C(i,s,a) )
       *
       * @param node
       * @param index
       * @return
       */
      public int findMin(int node, int index) {
            double minValue = ((Qnode)
this.graph.get(this.graph.get(node).children
                        .get(0))).getCvalue(index);
            int minPos = 0;
            double temp = 0;
            for (int i = 1; i < this.graph.get(node).children.size();
i++) {
                  temp = ((Qnode) this.graph

      .get(this.graph.get(node).children.get(i)))
                              .getCvalue(index);
                  if (temp < minValue) {
                        minValue = temp;
                        minPos = i;
                  }
            }
            return minPos;
      }

      /**
       * methodos i opoia arxikopoiei ta learning rates ton Max nodes
       */
      public void initLearningRate(double lr) {
            // set learning rate to Max nodes
            ((MaxNode)
this.graph.get(this.MaxRoot)).setLearningRate(lr);
            ((MaxNode)
this.graph.get(this.MaxGet)).setLearningRate(lr);
```

```
            ((MaxNode)
this.graph.get(this.MaxPut)).setLearningRate(lr);
            ((MaxNode)
this.graph.get(this.Pickup)).setLearningRate(lr);
            ((MaxNode)
this.graph.get(this.Putdown)).setLearningRate(lr);
            ((MaxNode)
this.graph.get(this.MaxNavigate)).setLearningRate(lr);
            ((MaxNode)
this.graph.get(this.North)).setLearningRate(lr);
            ((MaxNode) this.graph.get(this.East)).setLearningRate(lr);
            ((MaxNode)
this.graph.get(this.South)).setLearningRate(lr);
            ((MaxNode) this.graph.get(this.West)).setLearningRate(lr);


      }

      /**
       * methodos i opoia arxikopoiei tis thermokrasies stous MaxNodes
       */
      public void initTemperature(int t) {
            ((MaxNode)
this.graph.get(this.MaxRoot)).setTemperature(t);
            ((MaxNode) this.graph.get(this.MaxGet)).setTemperature(t);
            ((MaxNode) this.graph.get(this.MaxPut)).setTemperature(t);
            ((MaxNode)
this.graph.get(this.MaxNavigate)).setTemperature(t);
      }

      /**
       * methodos i opoia arxikopoiei ta cooling rates stous MaxNodes
       */
      public void initCoolingRates() {
            ((MaxNode)
this.graph.get(this.MaxRoot)).setCoolingRate(0.9996);
            ((MaxNode)
this.graph.get(this.MaxGet)).setCoolingRate(0.9939);
            ((MaxNode)
this.graph.get(this.MaxPut)).setCoolingRate(0.9996);
            ((MaxNode)
this.graph.get(this.MaxNavigate)).setCoolingRate(0.9879);
      }

      /**
       * methodos i opoia allazi ta learning rate ton Max Nodes
       *
       * @param episode
       * @param max_episode
       */
      public void changeLearningRate(int episode, int max_episode) {
            ((MaxNode)
this.graph.get(this.MaxRoot)).changeLearningRate(episode,
                        max_episode);
            ((MaxNode)
this.graph.get(this.MaxGet)).changeLearningRate(episode,
                        max_episode);
```

```
            ((MaxNode)
this.graph.get(this.MaxPut)).changeLearningRate(episode,
                        max_episode);
            ((MaxNode)
this.graph.get(this.Pickup)).changeLearningRate(episode,
                        max_episode);
            ((MaxNode)
this.graph.get(this.Putdown)).changeLearningRate(episode,
                        max_episode);
            ((MaxNode)
this.graph.get(this.MaxNavigate)).changeLearningRate(
                        episode, max_episode);
            ((MaxNode)
this.graph.get(this.North)).changeLearningRate(episode,
                        max_episode);
            ((MaxNode)
this.graph.get(this.East)).changeLearningRate(episode,
                        max_episode);
            ((MaxNode)
this.graph.get(this.South)).changeLearningRate(episode,
                        max_episode);
            ((MaxNode)
this.graph.get(this.West)).changeLearningRate(episode,
                        max_episode);
      }
      /**
       * methodos i opoia ekteli to action apo to state s, vriski tin
       * epomeni katastasi (state) kai epistrefi to reward
       *
       * @param s
       * @param tempState
       * @param action
       * @return
       */
      public int takeAction(State_Variables s, State_Variables
tempState,
                  int action) {

            int taxi = s.taxi;
            int passenger = s.passenger;
            int destination = s.destination;
            int reward = -1;
            switch (action) {
            // north (move up)
            case 0: // ektos orion tou grid
                  if ((!((s.taxi - SA_Gridworld.size_x) < 0))
                              && (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi - SA_Gridworld.size_x))))) {
                        taxi = s.taxi - SA_Gridworld.size_x;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }

                  break;
            // south (move down)
            case 1: // ektos orion tou grid
```

```
                  if ((!((s.taxi + SA_Gridworld.size_y) >=
(SA_Gridworld.size_x * SA_Gridworld.size_y)))
                              && (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi + SA_Gridworld.size_y))))) {
                        taxi = s.taxi + SA_Gridworld.size_y;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // east (move right)
            case 2: // hit the wall or ektos orion tou grid
                  if ((!((s.taxi % SA_Gridworld.size_y) ==
(SA_Gridworld.size_y - 1)))
                              && (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi + 1))))) {
                        taxi = s.taxi + 1;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // west (move left)
            case 3: // hit the wall or ektos orion tou grid
                  if ((!((s.taxi % SA_Gridworld.size_y) == 0))
                              && (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi - 1))))) {
                        taxi = s.taxi - 1;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // pickup the passenger
            case 4: // location without a passenger or passenger
already in the taxi
                  if ((s.taxi != SA_Gridworld.getPasLoc(s.passenger))
                              || (SA_Gridworld.getPasLoc(s.passenger)
== -1)) {
                        reward = SA_Gridworld.wrong_get;
                  } else {
                        passenger = 0;
                        reward = SA_Gridworld.simple_move;
                  }
                  break;
            // put down the passenger
            case 5: // wrong destination or passenger not in the taxi
                  if ((s.taxi !=
SA_Gridworld.getDestLoc(s.destination))
                              || (SA_Gridworld.getPasLoc(s.passenger)
!= -1)) {
                        reward =SA_Gridworld.wrong_put;
                  } else { // Successful passenger delivery
                        passenger = s.destination + 1;
                        reward = SA_Gridworld.success;
                  }
                  break;
            default:
```

```
                    System.err.println("Invalid number for
action..Exiting..");
                    System.exit(-1);
              }
              tempState.taxi = taxi;
              tempState.passenger = passenger;
              tempState.destination = destination;
              return reward;
        }
}
```

State_Variables.java

```
/**
 * klasi i opoia antiprosopevi ta state variables tou episodiou gia
 * single agent
 */
public class State_Variables {

      // Constructor 1
      State_Variables() {
            super();
      }

      // Constructor 2
      State_Variables(int size_x, int size_y, int numOfStartPosition,
                  int numOfGoals) {
            int tmp = size_x * size_y;
            this.taxi = SA_Taxi_Problem.r.nextInt(tmp);
            this.passenger =
SA_Taxi_Problem.r.nextInt(numOfStartPosition);
            this.destination = SA_Taxi_Problem.r.nextInt(numOfGoals);
      }

      // Constructor 3
      State_Variables(int taxi, int passenger, int destination) {
            this.taxi = taxi;
            this.passenger = passenger;
            this.destination = destination;
      }

      // Constructor 4
      State_Variables(State_Variables s) {
            this.taxi = s.taxi;
            this.passenger = s.passenger;
            this.destination = s.destination;
      }

      int taxi;
      int passenger;
      int destination;
}
```

```java
import org.jfree.data.xy.XYSeries;

/**
 * klasi i opoia einai ipefthini gia na kanei polla trials gia tous
 * algorithmous
 * @author Giannis
 *
 */
public class RunTrials {

	private int[][] episodeSteps;
	private double[] averageSteps;

	/********** set,get for private variables ***********/

	/**
	 * @param episodeSteps
	 *            the episodeSteps to set
	 */
	public void setEpisodeSteps(int[][] episodeSteps) {
		this.episodeSteps = episodeSteps;
	}

	/**
	 * methodos i opoia kathorizi ta steps se kapio episodio kapiou
	 * trial
	 * @param i
	 * @param j
	 * @param steps
	 */
	public void setEpisodeSteps(int trial, int[] episodeSteps) {
		for (int i = 0; i < episodeSteps.length; i++)
			this.episodeSteps[trial][i] = episodeSteps[i];
	}

	/**
	 * @return the episodeSteps
	 */
	public int[][] getEpisodeSteps() {
		return episodeSteps;
	}

	/**
	 * methodos i opoia epistrefi ta steps enos episode enos trial
	 *
	 * @param trial
	 * @param episode
	 * @return
	 */
	public int getEpisodeStep(int trial, int episode) {
		return this.episodeSteps[trial][episode];
	}
```

```java
/**
 * methodos i opoia kathorizi ta avarege step enos episode
 *
 * @param i
 * @param steps
 */
public void setAverageSteps(int episode, double avgSteps) {
	this.averageSteps[episode] = avgSteps;
}



/**
 * @return the averageSteps
 */
public double[] getAverageSteps() {
	return averageSteps;
}

/**
 * methodos i opoia epistrefi ta average steps enos trial
 *
 * @param trial
 * @return
 */
public double getAverageSteps(int trial) {
	return averageSteps[trial];
}

/*************************************************/

/**
 * methodos i opoia dimiourgi ton pinaka episodeSteps
 */
public void makeNewEpisodeSteps(int trials, int episodes) {
	this.episodeSteps = new int[trials][episodes];
}

/**
 * methodos i opoia dimiourgi ton pinaka averageSteps
 *
 * @param trials
 */
public void makeNewAverageSteps(int episodes) {
	this.averageSteps = new double[episodes];
}

/**
 * methodos i opoia vriski to average step gia ena episode
 *
 * @param trial
 * @param episodeSteps
 * @param trials
 * @return
 */
```

```
	public double findAvarage(int episode, int[][] episodeSteps, int
trials) {
		double sum = 0;
		for (int i = 0; i < trials; i++) {
			sum += episodeSteps[i][episode];
		}
		return sum / trials;
	}

	/**
	 * methodos stin opoia kathorizete to learning rate
	 *
	 * @return
	 */
	public double chooseLearningRate() {
		double lr;
		do {
			System.out
					.println("Please set the learning rate
(between 0 and 1)");
			lr = StdIn.readDouble();
		} while (lr < 0 || lr > 1);
		return lr;
	}

	/**
	 * methodos stin opoia kathorizete to discount factor
	 *
	 * @return
	 */
	public double chooseDiscountFactor() {
		double df;
		do {
			System.out
					.println("Please set the discount factor
(between 0 and 1)");
			df = StdIn.readDouble();
		} while (df < 0 || df > 1);
		return df;
	}

	/**
	 * methodos stin opoia kathorizete to epsilon
	 *
	 * @return
	 */
	public double chooseEpsilon() {
		double epsilon;
		do {
			System.out.println("Please set epsilon(between 0 and
1)");
			epsilon = StdIn.readDouble();
		} while (epsilon < 0 || epsilon > 1);
		return epsilon;
	}

	/**
	 * methodos stin opoia kathorizetai to temperature gia boltzmann
```

```
	 * exploration
	 */
	public double chooseTemperature() {
		double temperature = 0;
		System.out.println("Please set temperature");
		temperature = StdIn.readDouble();
		return temperature;
	}

	/**
	 * methodos stin opoia kathorizetai to coolingRate gia boltzmann
	 * exploration
	 */
	public double chooseCoolingRate() {
		double coolingRate = 0;
		do {
			System.out.println("Please set the cooling rate
(between 0-1)");
			coolingRate = StdIn.readDouble();
		} while (coolingRate < 0 || coolingRate > 1);
		return coolingRate;
	}

	/**
	 * methodos stin opoia ginete epilogi gia exploration meta3i e-
	 * greedy kai boltzmann
	 */
	public boolean chooseExploration() {
		boolean exploration;
		System.out
				.println("Please choose what exploration do
you want from the following:");
		System.out.println("\t0: Botzmann explotation");
		System.out.println("\t1: e-greedy");
		exploration = StdIn.readBoolean();
		return exploration;
	}

	/**
	 * methodos i opoia kani plot tin grafiki
	 *
	 * @param title
	 */
	public void plotGraph(String title,String xySeries) {
		// dimiourgia grafikis
		XYSeries steps = new XYSeries(xySeries);

		for (int i = 0; i < this.getAverageSteps().length; i++) {
			steps.add(i, this.getAverageSteps(i));
		}

		PlotGraph.createChart(steps, title, "Episode", "Average
Steps");
	}
```

```
	/**
	 * methodos i opoia kani polla trials gia tous diaforous
	 * algorithmous kai ipologizi ta average steps tous
	 *
	 * @param trials
	 * @param episodes
	 */
	public void calculate(int choice) {

		System.out
				.println("Please enter how many trials you
would like to run:");
		int trials = StdIn.readInt();
		System.out
				.println("Please enter how many episodes you
would like to run:");
		int episodes = StdIn.readInt();
		System.out.println("Please insert the name of the input
file:");
		String filename = StdIn.readString();
		this.makeNewEpisodeSteps(trials, episodes);
		this.makeNewAverageSteps(episodes);
		boolean exploration = true;
		double epsilon = 0;
		double temperature = 0;
		double coolingRate = 0;
		double lr = 0;
		double df = 0;
		double delta = 0;
		double deltaW = 0;
		double deltaL = 0;
		PrintInFile p = new PrintInFile();
		switch (choice) {
		case 0:// Q-Learning
			exploration = chooseExploration();
			if (exploration) {
				epsilon = chooseEpsilon();
			} else {
				temperature = chooseTemperature();
				coolingRate = chooseCoolingRate();
			}
			lr = chooseLearningRate();
			df = chooseDiscountFactor();
			Q_Learning ql = new Q_Learning();
			for (int i = 0; i < trials; i++) {
				System.out.println("trial=" + i);
				ql.calculate(episodes, exploration, epsilon,
temperature,
						coolingRate, filename, lr, df);
				this.setEpisodeSteps(i, ql.getEpisodeSteps());
			}
			for (int i = 0; i < episodes; i++) {
				this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
						trials));
			}
			if (exploration)
```

```
                        p.printResults("Q-Learning Average Results
with e-greedy.txt",
                                    this.getAverageSteps(), episodes);
                  else
                        p
                                    .printResults(
                                                "Q-Learning Average
Results with boltzmann exploration.txt",

      this.getAverageSteps(), episodes);
                  plotGraph("Q-Learning Average Steps Per Episode","Q-
Learning");
                  break;
            case 1:// Sarsa
                  exploration = chooseExploration();
                  if (exploration) {
                        epsilon = chooseEpsilon();
                  } else {
                        temperature = chooseTemperature();
                        coolingRate = chooseCoolingRate();
                  }
                  lr = chooseLearningRate();
                  df = chooseDiscountFactor();
                  Sarsa s = new Sarsa();
                  for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        s.calculate(episodes, exploration, epsilon,
temperature,
                                    coolingRate, filename, lr, df);
                        this.setEpisodeSteps(i, s.getEpisodeSteps());
                  }
                  for (int i = 0; i < episodes; i++) {
                        this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                    trials));
                  }
                  if (exploration)
                        p.printResults("Sarsa Average Results with e-
greedy.txt", this
                                    .getAverageSteps(), episodes);
                  else
                        p.printResults(
                                    "Sarsa Average Results with
boltzmann exploration.txt",
                                    this.getAverageSteps(), episodes);
                  plotGraph("Sarsa Average Steps Per
Episode","SARSA");
                  break;
            case 2:// PHC
                  lr = chooseLearningRate();
                  df = chooseDiscountFactor();
                  epsilon = chooseEpsilon();
                  do {
                        System.out.println("Please set delta (between
0 and 1)");
                        delta = StdIn.readDouble();
                  } while (delta < 0 || delta > 1);
                  Phc phc = new Phc();
```

```
                    for (int i = 0; i < trials; i++) {
                          System.out.println("trial=" + i);
                          phc.calculate(episodes, epsilon, lr, df,
delta, filename);
                          this.setEpisodeSteps(i,
phc.getEpisodeSteps());
                    }
                    for (int i = 0; i < episodes; i++) {
                          this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                      trials));
                    }
                    p.printResults("PHC Average Results.txt",
this.getAverageSteps(),
                                episodes);
                    plotGraph("PHC Average Steps Per Episode","PHC");
                    break;
              case 3:// WoLF-PHC
                    Wolf w = new Wolf();
                    lr = chooseLearningRate();
                    df = chooseDiscountFactor();
                    epsilon = chooseEpsilon();
                    do {
                          System.out
                                      .println("Please set delta winning
(between 0 and 1)");
                          deltaW = StdIn.readDouble();
                    } while (deltaW < 0 || deltaW > 1);

                    do {
                          System.out.println("Please set delta losing
(between 0 and 1)");
                          deltaL = StdIn.readDouble();
                    } while (deltaL < 0 || deltaL > 1);

                    for (int i = 0; i < trials; i++) {
                          System.out.println("trial=" + i);
                          w
                                      .calculate(episodes, epsilon, lr,
df, deltaW, deltaL,
                                                  filename);
                          this.setEpisodeSteps(i, w.getEpisodeSteps());
                    }
                    for (int i = 0; i < episodes; i++) {
                          this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                      trials));
                    }
                    p.printResults("WoLF-PHC Average Results.txt", this
                                .getAverageSteps(), episodes);
                    plotGraph("WoLF-PHC Average Steps Per
Episode","WoLF-PHC");
                    break;
              case 4:// MAXQ-Q
                    System.out
                                .println("Please choose what exploration
do you want from the following:");
                    System.out.println("\t0: Botzmann explotation");
```

```
                  System.out.println("\t1: e-greedy");
                  exploration = StdIn.readBoolean();
                  if (exploration)
                        epsilon = chooseEpsilon();
                  lr = chooseLearningRate();
                  df = chooseDiscountFactor();
                  MaxQ_Q maxq = new MaxQ_Q();
                  this.makeNewEpisodeSteps(trials, episodes);
                  this.makeNewAverageSteps(episodes);
                  for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        maxq.calculate(episodes, exploration, epsilon,
df, filename,lr);
                        this.setEpisodeSteps(i,
maxq.getEpisodeSteps());
                  }
                  for (int i = 0; i < episodes; i++) {
                        this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                    trials));
                  }
                  if (exploration)
                        p.printResults("MaxQ-Q Average Results with e-
greedy.txt", this
                                    .getAverageSteps(), episodes);
                  else
                        p
                                    .printResults(
                                                "MAXQ-Q Average
Results with boltzmann exploration.txt",

      this.getAverageSteps(), episodes);

                  plotGraph("MAXQ-Q Average Steps Per Episode","MAXQ-
Q");
                  break;
            case 5:// MAXQ-PHC
                  lr = chooseLearningRate();
                  df = chooseDiscountFactor();
                  epsilon = chooseEpsilon();
                  do {
                        System.out.println("Please set delta (between
0 and 1)");
                        delta = StdIn.readDouble();
                  } while (delta < 0 || delta > 1);

                  MaxQ_PHC m = new MaxQ_PHC();
                  this.makeNewEpisodeSteps(trials, episodes);
                  this.makeNewAverageSteps(episodes);
                  for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        m.calculate(episodes, epsilon, df, delta,
filename,lr);
                        this.setEpisodeSteps(i, m.getEpisodeSteps());
                  }
                  for (int i = 0; i < episodes; i++) {
                        this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
```

```
                                    trials));
                        }
                        p.printResults("MAXQ-PHC Average Results.txt", this
                                    .getAverageSteps(), episodes);
                        plotGraph("MAXQ-PHC Average Steps Per
Episode","MAXQ-PHC");
                        break;
                  case 6:// MAXQ-WoLF-PHC
                        lr = chooseLearningRate();
                        df = chooseDiscountFactor();
                        epsilon = chooseEpsilon();
                        do {
                              System.out
                                          .println("Please set delta winning
(between 0 and 1)");
                              deltaW = StdIn.readDouble();
                        } while (deltaW < 0 || deltaW > 1);

                        do {
                              System.out.println("Please set delta losing
(between 0 and 1)");
                              deltaL = StdIn.readDouble();
                        } while (deltaL < 0 || deltaL > 1);

                        MaxQ_WoLF_PHC mw = new MaxQ_WoLF_PHC();
                        this.makeNewEpisodeSteps(trials, episodes);
                        this.makeNewAverageSteps(episodes);
                        for (int i = 0; i < trials; i++) {
                              System.out.println("trial=" + i);
                              mw.calculate(episodes, epsilon, df, deltaW,
deltaL, filename,lr);
                              this.setEpisodeSteps(i, mw.getEpisodeSteps());
                        }
                        for (int i = 0; i < episodes; i++) {
                              this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                          trials));
                        }
                        p.printResults("MAXQ-WoLF-PHC Average Results.txt",
this
                                    .getAverageSteps(), episodes);
                        plotGraph("MAXQ-WoLF-PHC Average Steps Per
Episode","MAXQ-WoLF-PHC");
                        break;
                  default:
                        System.err.println("Invalid number for
choice..Exiting..");
                        System.exit(-1);
                  }
            }

            /**
             * @param args
             */
            public static void main(String[] args) {
                  // TODO Auto-generated method stub
            }
}
```

PlotGraph.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

/**
 * klasi ipefthini gia tin dimiourgia ton grafikon parastasewn
 *
 * @author Giannis
 *
 */
public class PlotGraph {

	/**
	 * methodos i opoia kani plot tin grafiki
	 *
	 * @param data
	 * @param title
	 * @param xAxis
	 * @param yAxis
	 */
	public static void createChart(ArrayList<XYSeries> data, String
title,
			String xAxis, String yAxis) {

		XYSeriesCollection dataset = new XYSeriesCollection();
		for (int i = 0; i < data.size(); i++) {
			dataset.addSeries(data.get(i));
		}

		JFreeChart chart = ChartFactory.createXYLineChart(title,
xAxis, yAxis,
				dataset, PlotOrientation.VERTICAL, true, true,
false);
		ChartFrame frame = new ChartFrame(title, chart);
		frame.setLocation(0, 0);
		frame.setVisible(true);
		frame.setSize(500, 400);

	}
```

```
	/**
	 * methodos i opoia dimiourga tin lista me tis times gia kathe
	 * grafiki
	 * @param algorithms
	 * @param flag
	 */
	public static void makeGraph(boolean[] algorithms, boolean flag)
{
		ArrayList<XYSeries> steps = new ArrayList<XYSeries>();
		int exploration = 0;
		for (int i = 0; i < algorithms.length; i++) {
			if (algorithms[i]) {
				switch (i) {
				case 0:
					do {
						System.out.println("Choose
exploration for Q-Learning");
						System.out.println("\t0: Botzmann
explotation");
						System.out.println("\t1: e-
greedy");
						System.out.println("\t2: both (if
they are exist)");
						exploration = StdIn.readInt();
					} while (exploration < 0 || exploration
> 2);
					if (exploration == 0) {
						readFromFile(
								"Q-Learning Average
Results with boltzmann exploration.txt",
								steps, "Q-Learning -
boltzmann exploration", i);
					} else if (exploration == 1) {
						readFromFile(
								"Q-Learning Average
Results with e-greedy.txt",
								steps, "Q-Learning -
e-greedy", i);
					} else {
						readFromFile(
								"Q-Learning Average
Results with boltzmann exploration.txt",
								steps, "Q-Learning -
boltzmann exploration", i);
						readFromFile(
								"Q-Learning Average
Results with e-greedy.txt",
								steps, "Q-Learning -
e-greedy", i);
					}
					break;
				case 1:
					do {
						System.out.println("Choose
exploration for Sarsa");
						System.out.println("\t0: Botzmann
explotation");
```

```
                                    System.out.println("\t1: e-
greedy");
                                    System.out.println("\t2: both (if
they are exist)");
                                    exploration = StdIn.readInt();
                              } while (exploration < 0 || exploration
> 2);
                              if (exploration == 0) {
                                    readFromFile(
                                                "Sarsa Average Results
with boltzmann exploration.txt",
                                                steps, "SARSA -
boltzmann exploration", i);
                              } else if (exploration == 1) {
                                    readFromFile("Sarsa Average
Results with e-greedy.txt",
                                                steps, "SARSA - e-
greedy", i);
                              } else {
                                    readFromFile(
                                                "Sarsa Average Results
with boltzmann exploration.txt",
                                                steps, "SARSA -
boltzmann exploration", i);
                                    readFromFile("Sarsa Average
Results with e-greedy.txt",
                                                steps, "SARSA - e-
greedy", i);
                              }
                              break;
                        case 2:
                              readFromFile("PHC Average Results.txt",
steps, "PHC", i);
                              ;
                              break;
                        case 3:
                              readFromFile("WoLF-PHC Average
Results.txt", steps,
                                          "WoLF-PHC", i);
                              break;
                        case 4:
                              do {
                                    System.out.println("Choose
exploration for MAXQ-Q");
                                    System.out.println("\t0: Botzmann
explotation");
                                    System.out.println("\t1: e-
greedy");
                                    System.out.println("\t2: both (if
they are exist)");
                                    exploration = StdIn.readInt();
                              } while (exploration < 0 || exploration
> 2);
                              if (exploration == 0) {
                                    readFromFile(
                                                "MAXQ-Q Average
Results with boltzmann exploration.txt",
```

```
                                                steps, "MAXQ-Q -
boltzmann exploration", i);
                              } else if (exploration == 1) {
                                    readFromFile(
                                                "MAXQ-Q Average
Results with e-greedy.txt",
                                                steps, "MAXQ-Q - e-
greedy", i);
                              } else {
                                    readFromFile(
                                                "MAXQ-Q Average
Results with boltzmann exploration.txt",
                                                steps, "MAXQ-Q -
boltzmann exploration", i);
                                    readFromFile(
                                                "MAXQ-Q Average
Results with e-greedy.txt",
                                                steps, "MAXQ-Q - e-
greedy", i);
                              }
                              break;
                        case 5:
                              readFromFile("MAXQ-PHC Average
Results.txt", steps,
                                          "MAXQ-PHC", i);
                              break;
                        case 6:
                              readFromFile("MAXQ-WoLF-PHC Average
Results.txt", steps,
                                          "MAXQ-WoLF-PHC", i);
                              break;
                        default:
                              System.err.println("Invalid
number..Exiting..");
                              System.exit(-1);
                        }
                  }
            }
            ArrayList<XYSeries> tmp = new ArrayList<XYSeries>();
            if (flag) {// different graphs
                  for (int i = 0; i < steps.size(); i++) {
                        tmp.add(steps.get(i));
                        PlotGraph.createChart(tmp, "Average Steps Per
Episode",
                                    "Episode", "Average Steps");
                        tmp.clear();
                  }
            } else {
                  PlotGraph.createChart(steps, "Average Steps Per
Episode",
                              "Episode", "Average Steps");
            }
      }
```

```
	/**
	 * methodos i opoia diavazi tis times gia tis grafikes apo to
	 * arxio
	 * @param filename
	 * @param steps
	 * @param str
	 * @param algorithm
	 */
	public static void readFromFile(String filename,
ArrayList<XYSeries> steps,
			String str, int algorithm) {
		XYSeries tmp = new XYSeries(str);

		BufferedReader in = null;
		String[] splitline;

		try {
			in = new BufferedReader(new FileReader(filename));
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
		String line = null;

		// grammes tou gridworld
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		int count = 0;
		while (line != null) {
			splitline = line.split("\t");
			tmp.add(count, Double.parseDouble(splitline[1]));
			try {
				line = in.readLine();
			} catch (IOException e) {
				e.printStackTrace();
			}
			count++;
		}

		steps.add(tmp);
	}

	public static void createChart(XYSeries data, String title,
String xAxis,
			String yAxis) {

		XYSeriesCollection dataset = new XYSeriesCollection();
		dataset.addSeries(data);

		JFreeChart chart = ChartFactory.createXYLineChart(title,
xAxis, yAxis,
				dataset, PlotOrientation.VERTICAL, true, true,
false);

		ChartFrame frame = new ChartFrame(title, chart);
```

```
		frame.setLocation(0, 0);
		frame.setVisible(true);
		frame.setSize(500, 400);

	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

Q_Learning.java

```
/**
 * klasi i opoia ilopoiei ton algorithmo Q-Learning gia to Taxi
 * Problem
 * @author Giannis
 *
 */
public class Q_Learning extends SA_Taxi_Problem {

	/**
	 * methodos i opia ekpedevi ton agent
	 */
	public void calculate(int max_episode, boolean exploration,
double epsilon,
			double temperature, double coolingRate, String
filename, double lr,
			double df) {
		SA_Gridworld.makeGridworld(filename);
		this.makeQ();
		double timer;
		/*
		 * if (!exploration) this.setCoolingRate(0.98); else
		 * this.setEpsilon(0.1); this.setLearningRate(0.5);
		 * this.setDiscountFactor(0.95);
		 */
		if (!exploration)
			this.setCoolingRate(coolingRate);
		else
			this.setEpsilon(epsilon);
		this.setLearningRate(lr);
		this.setDiscountFactor(df);

		State_Variables s;
		this.makeNewEpisodeSteps(max_episode);
		this.makeNewTimePerEpisode(max_episode);
		this.initQ(1);
		for (int i = 0; i < max_episode; i++) {
			if (!exploration)
```

```
                        this.setTemperature(50);
                  timer = (double) System.currentTimeMillis() / 1000;
                  // System.out.println("episode=" + i);
                  s = new State_Variables(SA_Gridworld.size_x,
SA_Gridworld.size_y,
                              SA_Gridworld.numOfStartPosition,
SA_Gridworld.numOfGoals);
                  this.q_learning(s, i, exploration);
                  this.setTimePerEpisode(i,
                              ((double) System.currentTimeMillis() /
1000 - timer));
            }

            // PrintInFile p = new PrintInFile();
            // p.printResults("Q-Learning_Results-twra.txt",
this.getEpisodeSteps(),
            // max_episode);
            // p.printResults("Q-Learning_clock time.txt",
this.getTimePerEpisode(),
            // max_episode);
            // p.printResults("Q-Learning_Results-Boltzmann.txt",
            // this.getEpisodeSteps(),
            // max_episode);
            // p.printResults("Q-Learning_clock time - Boltzmann.txt",
            // this.getTimePerEpisode(),
            // max_episode);
      }

      /**
       * methodos i opoia ilopoiei to taxi problem simfona me ton
       * algorithmo Q-Learning
       *
       * @param s
       * @param episode
       */
      public void q_learning(State_Variables s, int episode, boolean
exploration) {

            int index = this.findIndex(s);

            int action;
            if (exploration) {
                  action = this.chooseAction(index);
            } else {
                  action =
this.chooseActionWithBoltzmannExploration(index);

      this.changeTemperature(this.getEpisodeSteps(episode));
            }
            int reward = 0;
            int next_index;
            State_Variables tempState;
            // Successful passenger delivery
            while (reward != 20) {
                  tempState = new State_Variables();
                  reward = this.takeAction(s, tempState, action);
                  next_index = this.findIndex(tempState);
                  this.addQValue(index, action,
```

```
                                    ((double) this.getLearningRate() *
((double) reward
                                                + (double)
this.getDiscountFactor()
                                                * this.getQValue(next_index,
this

      .findMaxQ(next_index)) - this.getQValue(
                                                index, action))));

                        s = new State_Variables(tempState.taxi,
tempState.passenger,
                                    tempState.destination);
                        this.addEpisodeSteps(episode);

                        index = next_index;
                        if (exploration) {
                              action = this.chooseAction(index);
                        } else {
                              action =
this.chooseActionWithBoltzmannExploration(index);

      this.changeTemperature(this.getEpisodeSteps(episode));
                        }
                  }

      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            Q_Learning ql = new Q_Learning();

ql.calculate(500,true,0.1,80,0.98,"inputFile.txt",0.3,0.95);
            // TODO Auto-generated method stub

      }

}
```

Sarsa.java

```
/**
 * klasi i opoia ilopoiei ton algorithmo Sarsa gia to Taxi Problem
 *
 * @author Giannis
 *
 */
public class Sarsa extends SA_Taxi_Problem {
      /**
       * methodos i opia ekpedevi ton agent
       */
      public void calculate(int max_episode, boolean exploration,
double epsilon,
```

```
                  double temperature, double coolingRate, String
filename, double lr,
                  double df) {
            SA_Gridworld.makeGridworld(filename);
            this.makeQ();
            if (!exploration)
                  this.setCoolingRate(coolingRate);
            else
                  this.setEpsilon(epsilon);
            this.setLearningRate(lr);
            this.setDiscountFactor(df);
            State_Variables s;
            this.makeNewEpisodes(max_episode);
            this.makeNewEpisodeSteps(max_episode);
            this.initQ(1);
            for (int i = 0; i < max_episode; i++) {
            //    System.out.println("episode=" + i);
                  if (!exploration)
                        this.setTemperature(50);
                  s = new State_Variables(SA_Gridworld.size_x,
SA_Gridworld.size_y,
                              SA_Gridworld.numOfStartPosition,
SA_Gridworld.numOfGoals);
                  this.sarsa(s, i,exploration);
            }

            //PrintInFile p = new PrintInFile();
            // p.printResuls("Q-Learning_Results.txt",
this.getEpisodeSteps(),
            // max_episode);
      //    p
      //                .printResults("Sarsa_Results.txt",
this.getEpisodeSteps(),
      //                            max_episode);
      }

      /**
       * methodos i opoia ilopoiei to taxi problem simfona me ton
       * algorithmo Sarsa
       * @param s
       * @param episode
       */
      public void sarsa(State_Variables s, int episode,boolean
exploration) {

            int index = this.findIndex(s);
            int action;
            if (exploration) {
                  action = this.chooseAction(index);
            } else {
                  action =
this.chooseActionWithBoltzmannExploration(index);

      this.changeTemperature(this.getEpisodeSteps(episode));
            }
            int reward = 0;
            int next_index;
            int next_action;
```

```
            State_Variables tempState;
            // Successful passenger delivery
            while (reward != 20) {
                  tempState = new State_Variables();
                  reward = this.takeAction(s, tempState, action);
                  next_index = this.findIndex(tempState);
                  next_action = this.chooseAction(next_index);

                  this.addQValue(index, action,
                              ((double) this.getLearningRate() *
((double) reward
                                          + (double)
this.getDiscountFactor()
                                          * this.getQValue(next_index,
next_action) - this.getQValue(index, action))));

                  s = new State_Variables(tempState.taxi,
tempState.passenger,
                              tempState.destination);

                  this.addEpisodeSteps(episode);

                  index = next_index;
                  action = next_action;
            }
      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            Sarsa sarsa = new Sarsa();
      //    sarsa.calculate(5000);

      }

}
```

PHC.java

```
/**
 * klasi i opoia ilopoiei ton algorithmo PHC gia to Taxi Problem
 *
 * @author Giannis
 */
public class Phc extends SA_Taxi_Problem {
      private double[][] policyValue = new
double[SA_Gridworld.states][SA_Gridworld.actions];
      private double delta; // learning rate (to delta)

      /****************** methodoi get,set,add ******************/
      /**
```

```
	 * methodos i opia dini timi stin sigkekrimeni thesi tou
policyValue
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setPolicyValue(int i, int j, double value) {
		this.policyValue[i][j] = value;
	}

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou policyValue
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	public double getPolicyValue(int i, int j) {
		return this.policyValue[i][j];
	}

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou
	 * policyValue to neo value
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addPolicyValue(int i, int j, double value) {
		this.policyValue[i][j] += value;
	}

	/**
	 * methodos i opoia arxikopoiei to delta
	 *
	 * @param delta
	 */
	public void setDelta(double delta) {
		this.delta = delta;
	}

	/**
	 * methodos i opoia epistrefi to delta
	 *
	 * @return
	 */
	public double getDelta() {
		return delta;
	}

	/*****************************************************/
	/**
	 * methodos i opoia dimiourga ena pinaka policyValue
	 */
	public void makeNewPolicyValue() {
```

```
            policyValue = new
double[SA_Gridworld.states][SA_Gridworld.actions];
      }

      /**
       * methodos i opoia arxikopoiei to policyValue
       *
       * @param actions
       */
      public void initPolicyValue() {
            for (int i = 0; i < SA_Gridworld.states; i++) {
                  for (int j = 0; j < SA_Gridworld.actions; j++) {
                        this.setPolicyValue(i, j, ((double) 1 /
SA_Gridworld.actions));
                  }
            }
      }



      /**
       * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
       * sigkekrimeni katastasi (state) me pithanotita p(s,a) kai me
       * kapia anixnefsi
       * @param index
       * @return
       */
      public int chooseAction(int index) {
            if (r.nextDouble() < this.getEpsilon())
                  return (r.nextInt(SA_Gridworld.actions));
            return this.chooseProbAction(index);
      }

      /**
       * methodos i opoia epilegi mia kinisi (action) vasi tis
       * pithanotitas p(s,a)
       * @param index
       * @return
       */
      public int chooseProbAction(int index) {
            double offset = 0;
            double rand = r.nextDouble();
            for (int i = 0; i < SA_Gridworld.actions; i++) {
                  offset += this.getPolicyValue(index, i);
                  if (offset >= rand)
                        return i;
            }
            System.out.println("rand=" + rand);
            System.out.println("offset=" + offset);
            return 5; // en tha erti pote dame

      }

      /**
       * methodos i opoia kani update tin timi tis pithanotitas gia ti
       * sigkekrimeni katastasi
       *
       * @param index
```

```
	 * @param action
	 */
	public void updatePolicyValue(int index, int action) {
		// System.out.println("index="+index);
		// System.out.println("action="+action);
		//
System.out.println("this.findMaxQ(index)="+this.findMaxQ(index));
		if (action == this.findMaxQ(index))
			this.addPolicyValue(index, action, this.getDelta());
		else
			this.addPolicyValue(index, action,
					((-this.getDelta()) /
(SA_Gridworld.actions - 1)));

		// System.out.println("this.getDelta()="+this.getDelta());
		// System.out.println("-this.getDelta() / (this.actions -
1)="+(-this.getDelta()
		// / (this.actions - 1)));

	}

	/**
	 * methodos i opoia kanonikopoiei tis pithanottites tou
	 * policyValue meta to update
	 *
	 * @param index
	 */
	public void normalizePolicyValue(int index) {
		double minNegative =
this.findMinNegativePolicyValue(index);
		double temp = Math.abs(2 * minNegative);
		double sum = 0;
		for (int i = 0; i < SA_Gridworld.actions; i++)
			this.addPolicyValue(index, i, temp);
		for (int i = 0; i < SA_Gridworld.actions; i++)
			sum += this.getPolicyValue(index, i);
		for (int i = 0; i < SA_Gridworld.actions; i++)
			this
					.setPolicyValue(index, i,
							(this.getPolicyValue(index,
i) / sum));

	}

	/**
	 * methodos i opoia vriski tin thesi tis megaliteris
	 * pithanotitas se ena state
	 *
	 * @param index
	 * @return
	 */
	public int findMaxValue(int index) {
		double temp = this.getPolicyValue(index, 0);
		int pos = 0;
		for (int i = 1; i < SA_Gridworld.actions; i++) {
			if (this.getPolicyValue(index, i) > temp) {
				temp = this.getPolicyValue(index, i);
				pos = i;
```

```
                }
            }
            return pos;
        }

        /**
         * methodos i opoia vriski tin thesi tis mikroteris pithanotitas
         * se ena state tou policyValue
         *
         * @param index
         * @return
         */
        public int findMinValue(int index) {
            double temp = this.getPolicyValue(index, 0);
            int pos = 0;
            for (int i = 1; i < SA_Gridworld.actions; i++) {
                if (this.getPolicyValue(index, i) < temp) {
                    temp = this.getPolicyValue(index, i);
                    pos = i;
                }
            }
            return pos;
        }

        /**
         * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
         * ena state tou policyValue
         *
         * @param index
         * @return
         */
        public double findMinNegativePolicyValue(int index) {
            double temp = 0;
            for (int i = 0; i < SA_Gridworld.actions; i++) {
                if (this.getPolicyValue(index, i) < temp) {
                    temp = this.getPolicyValue(index, i);
                }
            }
            return temp;
        }

        /**
         * methodos i opia ekpedevi ton agent
         */
        public void calculate(int max_episode,double epsilon,double
lr,double df,double delta,String filename) {
            SA_Gridworld.makeGridworld(filename);
            this.makeQ();
            /*this.setEpsilon(0.1);
            this.setLearningRate(0.3);
            this.setDiscountFactor(0.95);
            this.setDelta(0.2);
            */
            this.setEpsilon(epsilon);
            this.setLearningRate(lr);
            this.setDiscountFactor(df);
            this.setDelta(delta);
            State_Variables s;
```

```
            this.makeNewEpisodes(max_episode);
            this.makeNewEpisodeSteps(max_episode);
            this.initQ(1);
            this.makeNewPolicyValue();
            this.initPolicyValue();
            for (int i = 0; i < max_episode; i++) {
            //    System.out.println("episode=" + i);
                  s = new State_Variables(SA_Gridworld.size_x,
SA_Gridworld.size_y,
                              SA_Gridworld.numOfStartPosition,
SA_Gridworld.numOfGoals);
                  this.phc(s, i);
            }

            //PrintInFile p = new PrintInFile();
            //p.printResults("PHC Results.txt",
this.getEpisodeSteps(), max_episode);
      }

      /**
       * methodos i opoia ilopoiei to taxi problem simfona me ton
       * algorithmo PHC
       * @param s
       * @param episode
       */
      public void phc(State_Variables s, int episode) {
            int index = this.findIndex(s);
            int action = this.chooseAction(index);
            int reward = 0;
            int next_index;
            State_Variables tempState;
            // Successful passenger delivery
            while (reward != 20) {
                  tempState = new State_Variables();
                  reward = this.takeAction(s, tempState, action);
                  next_index = this.findIndex(tempState);
                  this.setQValue(index, action,
                              (((double) 1 - this.getLearningRate())
                                          * this.getQValue(index,
action) + this

                                          .getLearningRate()
                                          * ((double) reward +
(double) this

      .getDiscountFactor()
                                                      *
this.getQValue(next_index, this

      .findMaxQ(next_index)))));

                  // update policyValue(s,a)
                  this.updatePolicyValue(index, action);

                  this.normalizePolicyValue(index);

                  s = new State_Variables(tempState.taxi,
tempState.passenger,
                              tempState.destination);
```

```
                this.addEpisodeSteps(episode);

                index = next_index;
                action = this.chooseAction(index);
            }
        }

        /**
         * @param args
         */
        public static void main(String[] args) {
            // TODO Auto-generated method stub
            Phc phc = new Phc();
            //phc.calculate(4000);

        }
}
```

WoLF-PHC.java

```
/**
 * klasi i opoia ilopoiei ton algorithmo WoLF-PHC gia to Taxi Problem
 *
 * @author Giannis
 *
 */
public class Wolf extends Phc {

        private double[][] avgPolicyValue; // average policy value
        private int[] C = new int[SA_Gridworld.states];
        private double deltaW; // learning rate for winning
        private double deltaL; // learning rate for losing

        /****************** methodoi get,set,add *********************/
        /**
         * methodos i opia dini timi stin sigkekrimeni thesi tou
         * avgPolicyValue
         * @param i
         * @param j
         * @param value
         */
        public void setAvgPolicyValue(int i, int j, double value) {
            this.avgPolicyValue[i][j] = value;
        }

        /**
         * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
         * tou avgPolicyValue
         *
         * @param i
         * @param j
         * @return
         */
```

```
public double getAvgPolicyValue(int i, int j) {
	return this.avgPolicyValue[i][j];
}

/**
 * methodos i opoia prostheti stin sigkekrimeni timi tou
 * avgPolicyValue to neo value
 *
 * @param i
 * @param j
 * @param value
 */
public void addAvgPolicyValue(int i, int j, double value) {
	this.avgPolicyValue[i][j] += value;
}

/**
 * methodos i opoia dini timi se mia thesi tou pinaka C
 *
 * @param state
 * @param num
 */
public void setC(int state, int num) {
	C[state] = num;
}

/**
 * methodos i opoia epistrefi mia timi tou pinaka C
 *
 * @return
 */
public int getC(int state) {
	return C[state];
}

/**
 * methodos i opoia prostheti 1 se mia thesi tou pinaka C
 *
 * @param state
 */
public void addOneToC(int state) {
	C[state]++;
}

/**
 * methodos i opoia arxikopoiei to deltaW
 *
 * @param deltaW
 */
public void setDeltaW(double deltaW) {
	this.deltaW = deltaW;
}

/**
 * methodos i opoia epistrefi to deltaW
 *
 * @return
 */
```

```
	public double getDeltaW() {
		return deltaW;
	}

	/**
	 * methodos i opoia arxikopoiei to deltaL
	 *
	 * @param deltaL
	 */
	public void setDeltaL(double deltaL) {
		this.deltaL = deltaL;
	}

	/**
	 * methodos i opoia epistrefi to deltaL
	 *
	 * @return
	 */
	public double getDeltaL() {
		return deltaL;
	}

	/*******************************************************/

	/**
	 * methodos i opoia dimiourga ena pinaka avgPolicyValue
	 */
	public void makeNewAvgPolicyValue() {
		avgPolicyValue = new
double[SA_Gridworld.states][SA_Gridworld.actions];
	}

	/**
	 * methodos i opoia dimiourga ena pinaka C
	 */
	public void makeNewC() {
		C = new int[SA_Gridworld.states];
	}

	/**
	 * methodos i opoia arxikopoiei to avgPolicyValue
	 *
	 * @param actions
	 */
	public void initAvgPolicyValue() {
		for (int i = 0; i < SA_Gridworld.states; i++) {
			for (int j = 0; j < SA_Gridworld.actions; j++) {
				this.setAvgPolicyValue(i, j, ((double) 1 /
SA_Gridworld.actions));
			}
		}
	}

	/**
	 * methodos i opoia arxikopoiei to C
	 *
	 * @param actions
	 */
```

```
	public void initC(int num) {
		for (int i = 0; i < 500; i++) {
			this.setC(i, 0);
		}
	}


	/**
	 * update estimate of average policy p
	 *
	 * @param index
	 * @param action
	 */
	public void updateAvgPolicyValue(int index, int action) {
		this.addOneToC(index);
		for (int i = 0; i < SA_Gridworld.actions; i++)
			this.addAvgPolicyValue(index, i,
					(((double) 1 / this.getC(index)) *
(this.getPolicyValue(
							index, i) -
this.getAvgPolicyValue(index, i))));
	}

	/**
	 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
	 * ena state tou avgPolicyValue
	 *
	 * @param index
	 * @return
	 */
	public double findMinNegativeAvgPolicyValue(int index) {
		double temp = 0;
		for (int i = 0; i < SA_Gridworld.actions; i++) {
			if (this.getAvgPolicyValue(index, i) < temp) {
				temp = this.getAvgPolicyValue(index, i);
			}
		}
		return temp;
	}

	/**
	 * methodos i opoia kanonikopoiei tis pithanottites tou
	 * avgPolicyValue meta to update
	 *
	 * @param index
	 */
	public void normalizeAvgPolicyValue(int index) {
		double minNegative =
this.findMinNegativeAvgPolicyValue(index);
		double temp = Math.abs(2 * minNegative);
		double sum = 0;
		for (int i = 0; i < SA_Gridworld.actions; i++)
			this.addAvgPolicyValue(index, i, temp);
		for (int i = 0; i < SA_Gridworld.actions; i++)
			sum += this.getAvgPolicyValue(index, i);
		for (int i = 0; i < SA_Gridworld.actions; i++)
			this.setAvgPolicyValue(index, i,
```

```
                              (this.getAvgPolicyValue(index, i) /
sum));

      }

      /**
       * methodos i opoia vriski pio delta (winning or losing) tha
       * xrisimopoihthei gia na ginoun update oi times tou policyValue
       *
       * @param index
       * @return
       */
      public double chooseDeltaForUpdate(int index) {
            double sumPolicy = 0;
            double sumAvgPolicy = 0;
            for (int i = 0; i < SA_Gridworld.actions; i++) {
                  sumPolicy += (this.getPolicyValue(index, i) *
this.getQValue(index,
                              i));
                  sumAvgPolicy += (this.getAvgPolicyValue(index, i) *
this.getQValue(
                              index, i));
            }
            if (sumPolicy > sumAvgPolicy)
                  return this.getDeltaW();
            else
                  return this.getDeltaL();
      }

      /**
       * methodos i opoia kani update tin timi tis pithanotitas gia ti
       * sigkekrimeni katastasi (simfona me to megalo to arthro)
       *
       * @param index
       * @param action
       */
      public void updatePolicyValue2(int index, int action) {
            double delta = this.chooseDeltaForUpdate(index);
            if (action != this.findMaxQ(index)) {
                  this.addPolicyValue(index, action,
Math.min(this.getPolicyValue(
                              index, action), (delta /
(SA_Gridworld.actions - 1))));
            } else {
                  double sum = 0;
                  for (int i = 0; i < SA_Gridworld.actions; i++) {
                        if (i == action)
                              continue;
                        sum += Math.min(this.getPolicyValue(index, i),
                                    (delta / (SA_Gridworld.actions -
1)));
                  }
                  this.addPolicyValue(index, action, sum);
            }
      }

      /**
       * methodos i opoia kani update tin timi tis pithanotitas gia ti
```

```
	 * sigkekrimeni katastasi
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue(int index, int action) {
		// System.out.println("index="+index);
		// System.out.println("action="+action);
		//
System.out.println("this.findMaxQ(index)="+this.findMaxQ(index));
		double delta = this.chooseDeltaForUpdate(index);
		if (action == this.findMaxQ(index))
			this.addPolicyValue(index, action, delta);
		else
			this.addPolicyValue(index, action,
					((-delta) / (SA_Gridworld.actions -
1)));

		// System.out.println("this.getDelta()="+this.getDelta());
		// System.out.println("-this.getDelta() / (this.actions -
1)="+(-this.getDelta()
		// / (this.actions - 1)));

	}

	/**
	 * methodos i opia ekpedevi ton agent
	 */
	public void calculate(int max_episode,double epsilon, double lr,
double df, double deltaW,
			double deltaL, String filename) {

		SA_Gridworld.makeGridworld(filename);
		this.makeQ();
		/*this.setEpsilon(0.1);
		this.setLearningRate(0.3);
		this.setDiscountFactor(0.95);
		this.setDeltaL(0.4);
		this.setDeltaW(0.1);
		*/
		this.setEpsilon(epsilon);
		this.setLearningRate(lr);
		this.setDiscountFactor(df);
		this.setDeltaL(deltaL);
		this.setDeltaW(deltaW);

		State_Variables s;
		this.makeNewEpisodes(max_episode);
		this.makeNewEpisodeSteps(max_episode);
		this.initQ(1);
		this.makeNewPolicyValue();
		this.initPolicyValue();
		this.makeNewAvgPolicyValue();
		this.initAvgPolicyValue();
		this.makeNewC();
		for (int i = 0; i < max_episode; i++) {
			// System.out.println("episode=" + i);
```

```
                  s = new State_Variables(SA_Gridworld.size_x,
SA_Gridworld.size_y,
                              SA_Gridworld.numOfStartPosition,
SA_Gridworld.numOfGoals);
                  this.wolf(s, i);
            }

            /*
             * PrintInFile p = new PrintInFile();
             * p.printResults("WoLF-PHC Results.txt",
this.getEpisodeSteps(),
             * max_episode);
             */
      }

      /**
       * methodos i opoia ilopoiei to taxi problem simfona me ton
       * algorithmo WoLF-PHC
       *
       * @param s
       * @param episode
       */
      public void wolf(State_Variables s, int episode) {
            int index = this.findIndex(s);
            int action = this.chooseAction(index);
            int reward = 0;
            int next_index;
            State_Variables tempState;
            // Successful passenger delivery
            while (reward != 20) {
                  tempState = new State_Variables();
                  reward = this.takeAction(s, tempState, action);
                  next_index = this.findIndex(tempState);
                  this.setQValue(index, action,
                              (((double) 1 - this.getLearningRate())
                                          * this.getQValue(index,
action) + this

                                          .getLearningRate()
                                          * ((double) reward +
(double) this

      .getDiscountFactor()
                                                      *
this.getQValue(next_index, this

      .findMaxQ(next_index)))));

                  // update avgPolicyValue(s,a)
                  this.updateAvgPolicyValue(index, action);

                  this.normalizeAvgPolicyValue(index);

                  // update policyValue(s,a)
                   this.updatePolicyValue(index, action);
                  //this.updatePolicyValue2(index, action);

                  this.normalizePolicyValue(index);
```

```
                    s = new State_Variables(tempState.taxi,
tempState.passenger,
                                tempState.destination);
                    this.addEpisodeSteps(episode);
                    index = next_index;
                    action = this.chooseAction(index);
              }

        }

        /**
         * @param args
         */
        public static void main(String[] args) {
              // TODO Auto-generated method stub
              Wolf wolf = new Wolf();
              //wolf.calculate(4000);
        }

}
```

MaxQ_Q.java

```
import java.util.ArrayList;

/**
 * klasi i opoia ilopoiei ton algorithmo MAXQ_Q
 *
 * @author Giannis
 *
 */
public class MaxQ_Q extends MaxQ {

        private int count;
        private int n;
        private State_Variables next_state;
        private int greedyAction;

        /********methodoi set,get,add gia tis private metavlites ******/

        /**
         * methodos i opoia anatheti timi sto count
         *
         * @param count
         *            the count to set
         */
        public void setCount(int count) {
              this.count = count;
        }

        /**
         * methodos i opoia epistrefi to count
         *
         * @return the count
         */
```

```
public int getCount() {
	return count;
}

/**
 * methodos i opoia anatheti timi sto n
 *
 * @param n
 */
public void setN(int n) {
	this.n = n;
}

/**
 * methodos i opoia epistrefi to n
 *
 * @return
 */
public int getN() {
	return n;
}

/**
 * methodos i opoia prostheti 1 sto n
 *
 * @return
 */
public void addOneToN() {
	this.n++;
}

/**
 * methodos i opoia kathorizi to greedy action
 *
 * @param greedyAction
 */
public void setGreedyAction(int greedyAction) {
	this.greedyAction = greedyAction;
}

/**
 * methodos i opoia epistrefi to greedy action
 *
 * @return
 */
public int getGreedyAction() {
	return greedyAction;
}

/**
 * methodos i opoia kathorizi to next state
 *
 * @param next_state
 *            the next_state to set
 */
public void setNext_state(State_Variables next_state) {
	this.next_state.destination = next_state.destination;
	this.next_state.passenger = next_state.passenger;
```

```
            this.next_state.taxi = next_state.taxi;
      }

      /**
       * methodos i opoia epistrefi to next state
       *
       * @return the next_state
       */
      public State_Variables getNext_state() {
            return next_state;
      }

      /*************************************************************/

      /**
       * methodos i opoia vriski ta Q values gia kathe children enos
       * MaxNode xrisimopoiontas to Cinside
       *
       * @param node
       * @param index
       * @return
       */
      public double[] findQvaluesWithCinside(int maxNode, int index) {
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            NodeValue nValue;
            for (int i = 0; i <
this.graph.get(maxNode).children.size(); i++) {
                  nValue =
this.evaluateMaxNode(this.graph.get(this.graph

      .get(maxNode).children.get(i)).children.get(0), index);
                  table[i] = nValue.value;
            }

            for (int i = 0; i < table.length; i++)
                  table[i] += ((Qnode) this.graph

      .get(this.graph.get(maxNode).children.get(i)))
                              .getCinsideValue(index);

            return table;
      }

      /**
       * methodos i opoia vriski ta Q values gia kathe children enos
       * MaxNode xrisimopoiontas to C
       *
       * @param node
       * @param index
       * @return
       */
      public double[] findQvaluesWithC(int maxNode, int index) {
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            NodeValue nValue;
            for (int i = 0; i <
this.graph.get(maxNode).children.size(); i++) {
```

```
                  nValue =
this.evaluateMaxNode(this.graph.get(this.graph

      .get(maxNode).children.get(i)).children.get(0), index);
                  table[i] = nValue.value;
            }

            for (int i = 0; i < table.length; i++)
                  table[i] += ((Qnode) this.graph

      .get(this.graph.get(maxNode).children.get(i)))
                              .getCvalue(index);

            return table;
      }

      /**
       * methodos i opoia vriski to greedy action gia ena Max node
       * xrisimopoiontas to Cinside
       *
       * @param node
       * @param index
       * @return
       */
      public int findGreedyAction(int maxNode, int index) {
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            table = this.findQvaluesWithCinside(maxNode, index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia vriski to greedy action gia ena Max node
       * xrisimopoiontas to C
       *
       * @param node
       * @param index
       * @return
       */
      public int findGreedyActionWithC(int maxNode, int index) {
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            table = this.findQvaluesWithC(maxNode, index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia kani copy ta nodes tou seq sto childSeq
       */
      public void setChildSeq(ArrayList<State_Variables> seq,
                  ArrayList<State_Variables> childSeq) {
            childSeq.clear();
            for (int i = 0; i < seq.size(); i++)
                  childSeq.add(seq.get(i));
      }
```

```
	/**
	 * methodos i opia ekpedevi ton agent
	 */
	public void calculate(int max_episode, boolean exploration,
double epsilon,
			double df, String filename,double lr) {
		SA_Gridworld.makeGridworld(filename);
		double timer;
		this.setEpsilon(epsilon);
		this.setDiscountFactor(df);
		this.graph.clear();
		this.makeGraph(SA_Gridworld.states,0);
		State_Variables s;
		// this.initTemperature(80);// gia boltzmann exploration
		this.initCoolingRates();// gia boltzmann exploration
		this.initLearningRate(lr);
		this.makeNewEpisodeSteps(max_episode);
		this.makeNewTimePerEpisode(max_episode);
		for (int i = 0; i < max_episode; i++) {
			timer = (double) System.currentTimeMillis() / 1000;
			// this.changeLearningRate(i, max_episode);
			this.initTemperature(50);// gia boltzmann
exploration
			// System.out.println("episode=" + i);
			// System.out.println("learning rate=" + ((MaxNode)
			// this.graph.get(this.MaxRoot)).getLearningRate());
			s = new State_Variables(SA_Gridworld.size_x,
SA_Gridworld.size_y,
					SA_Gridworld.numOfStartPosition,
SA_Gridworld.numOfGoals);
			this.next_state = new
State_Variables(SA_Gridworld.size_x,
					SA_Gridworld.size_y,
SA_Gridworld.numOfStartPosition,
					SA_Gridworld.numOfGoals);
			this.maxQ_Q(this.MaxRoot, s, i, exploration);
			this.setTimePerEpisode(i,
					((double) System.currentTimeMillis() /
1000 - timer));
		}

		//PrintInFile p = new PrintInFile();
		//p.printResults("MaxQ_Q-results twra .txt",
this.getEpisodeSteps(),
		//		max_episode);
		// p.printResults("MaxQ_Q-clock time.txt",
this.getTimePerEpisode(),
		// max_episode);

		// p.printResults("MaxQ_Q-results Boltzmann.txt",
		// this.getEpisodeSteps(),
		// max_episode);
		// p.printResults("MaxQ_Q-clock time Boltzmann.txt", this
		// .getTimePerEpisode(), max_episode);
	}
```

```
      /**
       * methodos i opoia ilopoiei ton MAXQ-Q algorithmo
       *
       * @param maxNode
       * @param s
       * @param episode
       * @return
       */
      public ArrayList<State_Variables> maxQ_Q(int maxNode,
State_Variables s,
                  int episode, boolean exploration) {

            ArrayList<State_Variables> seq = new
ArrayList<State_Variables>();
            int reward = 0;
            int index = 0;
            double lr = 0;
            // int count = 0;
            // int n = 0;

            // an einai Qnode tote pernoume to child tou (ton maxnode
pou exteli to
            // action)
            if (this.graph.get(maxNode) instanceof Qnode) {
                  // kathorizoume tin parametro t gia to
MaxNavigate(t)
                  if (maxNode == this.QNavigateForGet) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNode)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                    .getPasLoc(s.passenger));
                  } else if (maxNode == this.QNavigateForPut) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNode)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                    .getDestLoc(s.destination));
                  } else {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                  }
            }

            // i periptosi opou o maxNode einai primitive
            if (((MaxNode) this.graph.get(maxNode)).isPrimitive()) {
                  index = this.findIndex(s);
                  State_Variables nextState = new State_Variables();
                  reward = this.takeAction(s, nextState, ((MaxNode)
this.graph
                              .get(maxNode)).getAction());
                  this.setNext_state(nextState);
                  lr = ((MaxNode)
this.graph.get(maxNode)).getLearningRate();
                  ((MaxNode)
this.graph.get(maxNode)).setValueFunctForPrimitive(
                              index, (1 - lr)
```

```
                                              * ((MaxNode)
this.graph.get(maxNode))

      .getValueFunctForPrimitive(index)
                                              + (lr * reward));

                  // System.out.println("s.destination=" +
s.destination
                  // + "\ns.passenger=" + s.passenger + "\ns.taxi=" +
s.taxi);

                  // System.exit(-1);
                  State_Variables tmp = new State_Variables(s);
                  seq.add(0, tmp);
                  this.addEpisodeSteps(episode);
            } else {// i periptosi opou o maxnode einai composite (not
primitive)
                  count = 0;
                  int action = 0;
                  int next_index = 0;
                  while (!(this.isTerminal(maxNode, s))) {
                        ArrayList<State_Variables> childSeq = new
ArrayList<State_Variables>();
                        index = this.findIndex(s);

                        // choose action
                        if (exploration) {//e-greedy
                              action = ((MaxNode)
this.graph.get(maxNode)).children

      .get(this.chooseAction(maxNode, index));
                        } else {//boltzmann
                              action = ((MaxNode)
this.graph.get(maxNode)).children

      .get(this.chooseActionWithBoltzmannExploration(
                                                          maxNode,
index));

                              ((MaxNode)
this.graph.get(maxNode)).changeTemperature(this
                                              .getEpisodeSteps(episode));
                        }
                        // einai to childSeq= tou algorithmou
                        this.setChildSeq(this.maxQ_Q(action, s,
episode, exploration),
                                    childSeq);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNode)
this.graph.get(maxNode)).getLearningRate();

                        this
                                    .setGreedyAction(((MaxNode)
this.graph.get(maxNode)).children

      .get(this.findGreedyAction(maxNode, next_index)));
```

```
                        this.setN(1);
                        int tmp_index = 0;
                        for (int i = 0; i < childSeq.size(); i++) {
                              tmp_index =
this.findIndex(childSeq.get(i));

                              // set Cinside
                              ((Qnode) this.graph.get(action))
                                          .setCinsideValue(
                                                      tmp_index,
                                                      ((1 - lr)
                                                                  *
((Qnode) this.graph.get(action))

            .getCinsideValue(tmp_index) + (lr
                                                                  *
Math

            .pow(

                        this

                                    .getDiscountFactor(),

                        n) * (this

      .getPseudoreward(maxNode,

            this.next_state)
                                                                  +
((Qnode) this.graph.get(this

            .getGreedyAction()))

            .getCinsideValue(next_index) + this

      .findValue(this.getGreedyAction(),

            tmp_index)))));

                              // set C value
                              ((Qnode) this.graph.get(action))
                                          .setCvalue(
                                                      tmp_index,
                                                      ((1 - lr)
                                                                  *
((Qnode) this.graph.get(action))

            .getCvalue(tmp_index) + (lr
                                                                  *
Math

            .pow(

                        this

                                    .getDiscountFactor(),
```

```
                        n) * (((Qnode) this.graph

      .get(this.getGreedyAction()))

      .getCvalue(next_index) + this

      .findValue(this.getGreedyAction(),

            next_index)))));

                              this.addOneToN();
                        }
                        // append childSeq onto the front of seq
                        this.appendChildSeqOntoSeq(childSeq, seq);

                        s = new State_Variables(this.next_state.taxi,
                                    this.next_state.passenger,
this.next_state.destination);
                        index = next_index;

                  }
            }

            return seq;
      }

      /**
       * methodos i opoia epistrefi to pseudo-reward gia ena maxnode
       * otan vriskete se kapia katastasi
       *
       * @param maxnode
       * @param s
       * @return
       */
      public int getPseudoreward(int maxnode, State_Variables s) {
            if (this.isTerminal(maxnode, s))
                  return 0;
            else
                  return -100;
      }

      /**
       * methodos i opoia vazi to childSeq stin arxi tou seq
       *
       * @param childSeq
       * @param seq
       */
      public void appendChildSeqOntoSeq(ArrayList<State_Variables>
childSeq,
                  ArrayList<State_Variables> seq) {
            for (int i = childSeq.size() - 1; i >= 0; i--)
                  seq.add(0, childSeq.get(i));

            // for (int i = 0; i < childSeq.size(); i++)
            // seq.add(0, childSeq.get(i));
      }
```

```
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MaxQ_Q maxq = new MaxQ_Q();
		//maxq.calculate(5000);
	}

}
```

MaxQ_PHC.java

```
import java.util.ArrayList;
import java.util.Stack;

/**
 * klasi i opoia ilopoiei ton algorithmo MAXQ-PHC
 *
 * @author Giannis
 *
 */
public class MaxQ_PHC extends MaxQ {

	protected int count;
	protected int n;
	protected State_Variables next_state;
	protected int greedyAction;

	private double delta; // learning rate (to delta)

	/*************** methodoi set,get,add *********************/

	/**
	 * methodos i opoia arxikopoiei to delta
	 *
	 * @param delta
	 */
	public void setDelta(double delta) {
		this.delta = delta;
	}

	/**
	 * methodos i opoia epistrefi to delta
	 *
	 * @return
	 */
	public double getDelta() {
		return delta;
	}
```

```
/**
 * methodos i opoia anatheti timi sto count
 *
 * @param count
 *            the count to set
 */
public void setCount(int count) {
	this.count = count;
}
/**
 * methodos i opoia epistrefi to count
 *
 * @return the count
 */
public int getCount() {
	return count;
}
/**
 * methodos i opoia anatheti timi sto n
 *
 * @param n
 */
public void setN(int n) {
	this.n = n;
}
/**
 * methodos i opoia epistrefi to n
 *
 * @return
 */
public int getN() {
	return n;
}
/**
 * methodos i opoia prostheti 1 sto n
 *
 * @return
 */
public void addOneToN() {
	this.n++;
}
/**
 * methodos i opoia kathorizi to greedy action
 *
 * @param greedyAction
 */
public void setGreedyAction(int greedyAction) {
	this.greedyAction = greedyAction;
}

/**
 * methodos i opoia epistrefi to greedy action
 *
 * @return
 */
public int getGreedyAction() {
	return greedyAction;
}
```

```
/**
 * methodos i opoia kathorizi to next state
 *
 * @param next_state
 *            the next_state to set
 */
public void setNext_state(State_Variables next_state) {
	this.next_state.destination = next_state.destination;
	this.next_state.passenger = next_state.passenger;
	this.next_state.taxi = next_state.taxi;
}

/**
 * methodos i opoia epistrefi to next state
 *
 * @return the next_state
 */
public State_Variables getNext_state() {
	return next_state;
}

/*************************************************************/

/********************** PHC ***********************/

/**
 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
 * sigkekrimeni katastasi (state) me pithanotita p(s,a) kai me
 * kapia anixnefsi
 * @param index
 * @return
 */
public int chooseAction(int index, double[][] policyValue) {
	if (r.nextDouble() < this.getEpsilon())
		return (r.nextInt(policyValue[0].length));
	return this.chooseProbAction(index, policyValue);
}

/**
 * methodos i opoia epilegi mia kinisi (action) vasi tis
 * pithanotitas p(s,a)
 * @param index
 * @return
 */
public int chooseProbAction(int index, double[][] policyValue) {
	double offset = 0;
	double rand = r.nextDouble();
	for (int i = 0; i < SA_Gridworld.actions; i++) {
		offset += policyValue[index][i];
		if (offset >= rand)
			return i;
	}
	System.out.println("rand=" + rand);
	System.out.println("offset=" + offset);
	return 5; // en tha erti pote dame

}
```

```
	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue2(int index, int action, int
maxNode) {
		int length = ((MaxNodePHC) this.graph.get(maxNode))
				.getPolicyValueLength();

		if (action != this.findGreedyAction(maxNode, index))
			((MaxNodePHC)
this.graph.get(maxNode)).addPolicyValue(index,
					action, Math.min(((MaxNodePHC)
this.graph.get(maxNode))
							.getPolicyValue(index,
action),
							(this.getDelta() / (length -
1))));
		else {
			double sum = 0;
			for (int i = 0; i < length; i++) {
				if (i == action)
					continue;
				sum += Math.min(((MaxNodePHC)
this.graph.get(maxNode))
						.getPolicyValue(index, i),
						(this.getDelta() / (length - 1)));
			}
			((MaxNodePHC)
this.graph.get(maxNode)).addPolicyValue(index,
					action, sum);
		}
	}

	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue(int index, int action, int
maxNode) {
		// System.out.println("index="+index);
		// System.out.println("action="+action);
		//
System.out.println("this.findMaxQ(index)="+this.findMaxQ(index));
		if (action == this.findGreedyAction(maxNode, index))
			((MaxNodePHC)
this.graph.get(maxNode)).addPolicyValue(index,
					action, this.getDelta());
		else
			((MaxNodePHC)
this.graph.get(maxNode)).addPolicyValue(index,
```

```
                                    action, ((-this.getDelta()) /
(((MaxNodePHC) this.graph

      .get(maxNode)).getPolicyValueLength() - 1)));

            // System.out.println("this.getDelta()="+this.getDelta());
            // System.out.println("-this.getDelta() / (this.actions -
1)="+(-this.getDelta()
            // / (this.actions - 1)));

      }

      /**
       * methodos i opoia kanonikopoiei tis pithanottites tou
       * policyValue meta to update
       *
       * @param index
       */
      public void normalizePolicyValue(int index, int maxNode) {
            int length = ((MaxNodePHC) this.graph.get(maxNode))
                        .getPolicyValueLength();
            double minNegative =
this.findMinNegativePolicyValue(index, maxNode);
            double temp = Math.abs(2 * minNegative);
            double sum = 0;
            for (int i = 0; i < length; i++)
                  ((MaxNodePHC)
this.graph.get(maxNode)).addPolicyValue(index, i,
                                    temp);
            for (int i = 0; i < length; i++)
                  sum += ((MaxNodePHC)
this.graph.get(maxNode)).getPolicyValue(index,
                                    i);
            for (int i = 0; i < length; i++)
                  ((MaxNodePHC)
this.graph.get(maxNode)).setPolicyValue(index, i,
                                    (((MaxNodePHC)
this.graph.get(maxNode)).getPolicyValue(
                                                index, i) / sum));

      }

      /**
       * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
ena state
       * tou policyValue
       *
       * @param index
       * @return
       */
      public double findMinNegativePolicyValue(int index, int maxNode)
{
            double temp = 0;
            int length = ((MaxNodePHC) this.graph.get(maxNode))
                        .getPolicyValueLength();
            for (int i = 0; i < length; i++) {
                  if (((MaxNodePHC)
this.graph.get(maxNode)).getPolicyValue(index, i) < temp) {
```

```
                        temp = ((MaxNodePHC)
this.graph.get(maxNode)).getPolicyValue(
                                        index, i);
                  }
            }
            return temp;
      }

      /**
       * methodos i opoia dimiourgei k arxikopoiei tous pinakes
policyValues se
       * olous tous komvous tou grafou
       */
      public void makePolicyValues() {
            for (int i = 0; i < this.numMaxNodes; i++) {
                  ((MaxNodePHC) this.graph.get(i)).makeNewPolicyValue(
                              SA_Gridworld.states,
this.graph.get(i).children.size());
                  ((MaxNodePHC) this.graph.get(i)).initPolicyValue(
                              SA_Gridworld.states,
this.graph.get(i).children.size());
            }
      }

      /*************************************************************/

      /**
       * methodos i opoia vriski ta Q values gia kathe children enos
       * MaxNode xrisimopoiontas to Cinside
       *
       * @param node
       * @param index
       * @return
       */
      public double[] findQvaluesWithCinside(int maxNode, int index) {
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            NodeValue nValue;
            for (int i = 0; i <
this.graph.get(maxNode).children.size(); i++) {
                  nValue =
this.evaluateMaxNode(this.graph.get(this.graph

      .get(maxNode).children.get(i)).children.get(0), index);
                  table[i] = nValue.value;
            }

            for (int i = 0; i < table.length; i++)
                  table[i] += ((Qnode) this.graph

      .get(this.graph.get(maxNode).children.get(i)))
                              .getCinsideValue(index);

            return table;
      }
```

```
	/**
	 * methodos i opoia vriski ta Q values gia kathe children enos
	 * MaxNode xrisimopoiontas to C
	 *
	 * @param node
	 * @param index
	 * @return
	 */
	public double[] findQvaluesWithC(int maxNode, int index) {
		double[] table = new
double[this.graph.get(maxNode).children.size()];
		NodeValue nValue;
		for (int i = 0; i <
this.graph.get(maxNode).children.size(); i++) {
			nValue =
this.evaluateMaxNode(this.graph.get(this.graph

	.get(maxNode).children.get(i)).children.get(0), index);
			table[i] = nValue.value;
		}

		for (int i = 0; i < table.length; i++)
			table[i] += ((Qnode) this.graph

	.get(this.graph.get(maxNode).children.get(i)))
					.getCvalue(index);

		return table;
	}

	/**
	 * methodos i opoia vriski to greedy action gia ena Max node
	 * xrisimopoiontas to Cinside
	 *
	 * @param node
	 * @param index
	 * @return
	 */
	public int findGreedyAction(int maxNode, int index) {
		double[] table = new
double[this.graph.get(maxNode).children.size()];
		table = this.findQvaluesWithCinside(maxNode, index);
		return this.findMax(table);
	}
	/**
	 * methodos i opoia vriski to greedy action gia ena Max node
	 * xrisimopoiontas to C
	 *
	 * @param node
	 * @param index
	 * @return
	 */
	public int findGreedyActionWithC(int maxNode, int index) {
		double[] table = new
double[this.graph.get(maxNode).children.size()];
		table = this.findQvaluesWithC(maxNode, index);
		return this.findMax(table);
	}
```

```
	/**
	 * methodos i opoia kani copy ta nodes tou seq sto childSeq
	 */
	public void setChildSeq(ArrayList<State_Variables> seq,
			ArrayList<State_Variables> childSeq) {
		childSeq.clear();
		for (int i = 0; i < seq.size(); i++)
			childSeq.add(seq.get(i));
	}

	/**
	 * methodos i opia ekpedevi ton agent
	 */
	public void calculate(int max_episode, double epsilon, double
df,
			double delta, String filename,double lr) {
		SA_Gridworld.makeGridworld(filename);
		double timer;
		this.setEpsilon(epsilon);
		this.setDiscountFactor(df);
		this.setDelta(delta);
		this.graph.clear();
		this.makeGraph(SA_Gridworld.states, 1);
		this.makePolicyValues();
		State_Variables s;
		this.initLearningRate(lr);
		this.makeNewEpisodeSteps(max_episode);
		this.makeNewTimePerEpisode(max_episode);
		for (int i = 0; i < max_episode; i++) {
			timer = (double) System.currentTimeMillis() / 1000;
			// System.out.println("episode=" + i);
			s = new State_Variables(SA_Gridworld.size_x,
SA_Gridworld.size_y,
					SA_Gridworld.numOfStartPosition,
SA_Gridworld.numOfGoals);
			this.next_state = new
State_Variables(SA_Gridworld.size_x,
					SA_Gridworld.size_y,
SA_Gridworld.numOfStartPosition,
					SA_Gridworld.numOfGoals);
			this.maxQ_PHC(this.MaxRoot, s, i);
			//this.maxQ_PHC_iterative(this.MaxRoot, s, i);
			this.setTimePerEpisode(i,
					((double) System.currentTimeMillis() /
1000 - timer));
		}

		/*
		 * PrintInFile p = new PrintInFile();
		 * p.printResults("MaxQ_PHC-results.txt",
this.getEpisodeSteps(),
		 * max_episode); p.printResults("MaxQ_PHC-clock time.txt",
		 * this.getTimePerEpisode(), max_episode);
		 */
	}
```

```
      /**
       * methodos i opoia ilopoiei ton MAXQ-0 algorithmo
       *
       * @param maxNode
       * @param s
       * @param episode
       * @return
       */
      public ArrayList<State_Variables> maxQ_PHC(int maxNode,
State_Variables s,
                  int episode) {

            ArrayList<State_Variables> seq = new
ArrayList<State_Variables>();
            int reward = 0;
            int index = 0;
            double lr = 0;
            // int count = 0;
            // int n = 0;

            // an einai Qnode tote pernoume to child tou (ton maxnode
pou exteli to
            // action)
            if (this.graph.get(maxNode) instanceof Qnode) {
                  // kathorizoume tin parametro t gia to
MaxNavigate(t)
                  if (maxNode == this.QNavigateForGet) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNodePHC)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                    .getPasLoc(s.passenger));
                  } else if (maxNode == this.QNavigateForPut) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNodePHC)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                    .getDestLoc(s.destination));
                  } else {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                  }
            }

            // i periptosi opou o maxNode einai primitive
            if (((MaxNodePHC) this.graph.get(maxNode)).isPrimitive())
{
                  index = this.findIndex(s);
                  State_Variables nextState = new State_Variables();
                  reward = this.takeAction(s, nextState, ((MaxNodePHC)
this.graph
                              .get(maxNode)).getAction());
                  this.setNext_state(nextState);
                  lr = ((MaxNodePHC)
this.graph.get(maxNode)).getLearningRate();
                  ((MaxNodePHC)
this.graph.get(maxNode)).setValueFunctForPrimitive(
                              index, (1 - lr)
```

```
                                                * ((MaxNodePHC)
this.graph.get(maxNode))

      .getValueFunctForPrimitive(index)
                                                + (lr * reward));

                  // System.out.println("s.destination=" +
s.destination
                  // + "\ns.passenger=" + s.passenger + "\ns.taxi=" +
s.taxi);

                  // System.exit(-1);
                  State_Variables tmp = new State_Variables(s);
                  seq.add(0, tmp);
                  this.addEpisodeSteps(episode);
            } else {// i periptosi opou o maxnode einai composite (not
primitive)
                  count = 0;
                  int action = 0;
                  int childAction = 0;
                  int next_index = 0;
                  while (!(this.isTerminal(maxNode, s))) {
                        ArrayList<State_Variables> childSeq = new
ArrayList<State_Variables>();
                        index = this.findIndex(s);

                        // choose action
                        childAction = this.chooseAction(index,
((MaxNodePHC) this.graph
                                    .get(maxNode)).getPolicyValue());
                        action = ((MaxNodePHC)
this.graph.get(maxNode)).children
                                    .get(childAction);

                        this.updatePolicyValue(index, childAction,
maxNode);
                        this.normalizePolicyValue(index, maxNode);

                        // einai to childSeq= tou algorithmou
                        this.setChildSeq(this.maxQ_PHC(action, s,
episode), childSeq);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNodePHC)
this.graph.get(maxNode)).getLearningRate();

                        this
                                    .setGreedyAction(((MaxNodePHC)
this.graph.get(maxNode)).children

      .get(this.findGreedyAction(maxNode, next_index)));
                        this.setN(1);
                        int tmp_index = 0;
                        for (int i = 0; i < childSeq.size(); i++) {
                              tmp_index =
this.findIndex(childSeq.get(i));
```

```
                                // set Cinside
                                ((Qnode) this.graph.get(action))
                                        .setCinsideValue(
                                                tmp_index,
                                                ((1 - lr)
                                                                        *
((Qnode) this.graph.get(action))

                .getCinsideValue(tmp_index) + (lr
                                                                        *
Math

                .pow(

                                this

                                        .getDiscountFactor(),

                                n) * (this

        .getPseudoreward(maxNode,

                this.next_state)
                                                                        +
((Qnode) this.graph.get(this

                .getGreedyAction()))

                .getCinsideValue(next_index) + this

        .findValue(this.getGreedyAction(),

                tmp_index)))));

                                // set C value
                                ((Qnode) this.graph.get(action))
                                        .setCvalue(
                                                tmp_index,
                                                ((1 - lr)
                                                                        *
((Qnode) this.graph.get(action))

                .getCvalue(tmp_index) + (lr
                                                                        *
Math

                .pow(

                                this

                                        .getDiscountFactor(),

                                n) * (((Qnode) this.graph

        .get(this.getGreedyAction()))

        .getCvalue(next_index) + this
```

```
      .findValue(this.getGreedyAction(),

            next_index)))));

                              this.addOneToN();
                        }
                        // append childSeq onto the front of seq
                        this.appendChildSeqOntoSeq(childSeq, seq);

                        s = new State_Variables(this.next_state.taxi,
                                    this.next_state.passenger,
this.next_state.destination);
                        index = next_index;

                  }
            }

            return seq;
      }
      /**
       * methodos i opoia elegxei an o node pou imaste einai Max node
       * i Qnode an einai Qnode tote pernoume to child tou (ton
       * maxnode pou exteli to action)
       * @param maxNode
       * @param s
       */
      public int checkInstanceOf(int maxNode, State_Variables s) {
            if (this.graph.get(maxNode) instanceof Qnode) {
                  // kathorizoume tin parametro t gia to
MaxNavigate(t)
                  if (maxNode == this.QNavigateForGet) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNodePHC)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                    .getPasLoc(s.passenger));
                  } else if (maxNode == this.QNavigateForPut) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNodePHC)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                    .getDestLoc(s.destination));
                  } else {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                  }
            }
            return maxNode;
      }
```

```
	/**
	 * methodos i opoia allazi tin timi tou Completion function (C
	 * kai Cinside)
	 * @param seqStack
	 * @param index
	 * @param next_index
	 * @param action
	 * @param maxNode
	 * @param lr
	 */
	public void changeCompletionValue(
			Stack<ArrayList<State_Variables>> seqStack, int
index,
			int next_index, int action, int maxNode, double lr)
{
		this.setN(1);
		int tmp_index = 0;
		for (int i = 0; i < seqStack.peek().size(); i++) {
			tmp_index = this.findIndex(seqStack.peek().get(i));

			// set Cinside
			((Qnode)
this.graph.get(action)).setCinsideValue(tmp_index,
					((1 - lr)
							* ((Qnode)
this.graph.get(action))

	.getCinsideValue(tmp_index) + (lr
							*
Math.pow(this.getDiscountFactor(), n) * (this
							.getPseudoreward(maxNode,
this.next_state)
							+ ((Qnode)
this.graph.get(this.getGreedyAction()))

	.getCinsideValue(next_index) + this

	.findValue(this.getGreedyAction(), tmp_index)))));

			// set C value
			((Qnode) this.graph.get(action))
					.setCvalue(
							tmp_index,
							((1 - lr)
									* ((Qnode)
this.graph.get(action))

	.getCvalue(tmp_index) + (lr
									*
Math.pow(this.getDiscountFactor(), n) * (((Qnode) this.graph

	.get(this.getGreedyAction()))

	.getCvalue(next_index) + this.findValue(

	this.getGreedyAction(), next_index)))));

			this.addOneToN();
```

```
            }
            // append childSeq onto the front of seq
            if (!seqStack.isEmpty())
                  this.appendChildSeqOntoSeq(seqStack.pop(),
seqStack.peek());
      }

      /**
       * methodos i opoia ilopoiei ton MAXQ-PHC algorithmo (iterative)
       *
       * @param maxNode
       * @param s
       * @param episode
       * @return
       */
      public void maxQ_PHC_iterative(int node, State_Variables s, int
episode) {

            Stack<ArrayList<State_Variables>> seqStack = new
Stack<ArrayList<State_Variables>>();
            Stack<Integer> stack = new Stack<Integer>();
            stack.push(node);

            int reward = 0;
            int index = 0;
            double lr = 0;
            // int count = 0;
            // int n = 0;
            int maxNode;
            int action = 0;
            int childAction = 0;
            int next_index = 0;

            while (!stack.isEmpty()) {
                  ArrayList<State_Variables> seq = new
ArrayList<State_Variables>();
                  seqStack.push(seq);
                  maxNode = stack.peek();
                  // elegxos an imaste se Max node
                  maxNode = checkInstanceOf(maxNode, s);
                  // i periptosi opou o maxNode einai primitive
                  if (((MaxNodePHC)
this.graph.get(maxNode)).isPrimitive()) {
                        index = this.findIndex(s);
                        State_Variables nextState = new
State_Variables();
                        reward = this.takeAction(s, nextState,
((MaxNodePHC) this.graph
                                    .get(maxNode)).getAction());
                        this.setNext_state(nextState);
                        lr = ((MaxNodePHC)
this.graph.get(maxNode)).getLearningRate();
                        ((MaxNodePHC) this.graph.get(maxNode))
                                    .setValueFunctForPrimitive(index,
(1 - lr)
                                                * ((MaxNodePHC)
this.graph.get(maxNode))
```

```
      .getValueFunctForPrimitive(index)
                                                + (lr * reward));

                        // System.out.println("s.destination=" +
s.destination
                        // + "\ns.passenger=" + s.passenger +
"\ns.taxi=" + s.taxi);

                        // System.exit(-1);
                        State_Variables tmp = new State_Variables(s);
                        seqStack.peek().add(0, tmp);
                        this.addEpisodeSteps(episode);
                        stack.pop();
                        maxNode = stack.peek();

                        // elegxos an imaste se Max node
                        maxNode = checkInstanceOf(maxNode, s);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNodePHC)
this.graph.get(maxNode)).getLearningRate();
                        this
                                    .setGreedyAction(((MaxNodePHC)
this.graph.get(maxNode)).children

      .get(this.findGreedyAction(maxNode, next_index)));

                        changeCompletionValue(seqStack, index,
next_index, action,
                                    maxNode, lr);
                        s = new State_Variables(this.next_state.taxi,
                                    this.next_state.passenger,
this.next_state.destination);
                        index = next_index;

                  } else {
                        if (!(this.isTerminal(maxNode, s))) {
                              index = this.findIndex(s);

                              // choose action
                              childAction = this.chooseAction(index,
                                          ((MaxNodePHC)
this.graph.get(maxNode))

      .getPolicyValue());
                              action = ((MaxNodePHC)
this.graph.get(maxNode)).children
                                          .get(childAction);

                              this.updatePolicyValue2(index,
childAction, maxNode);
                              this.normalizePolicyValue(index,
maxNode);
                              stack.push(action);
                        } else {
                              stack.pop();
```

```
                                    if (stack.isEmpty())
                                          break;
                                    maxNode = stack.peek();

                                    // elegxos an imaste se Max node
                                    maxNode = checkInstanceOf(maxNode, s);
                                    // observe next state
                                    next_index =
this.findIndex(this.next_state);
                                    lr = ((MaxNodePHC)
this.graph.get(maxNode))
                                                .getLearningRate();

                                    this

      .setGreedyAction(((MaxNodePHC) this.graph

      .get(maxNode)).children.get(this

      .findGreedyAction(maxNode, next_index)));
                                    changeCompletionValue(seqStack, index,
next_index, action,
                                                maxNode, lr);
                                    s = new
State_Variables(this.next_state.taxi,
                                                this.next_state.passenger,

      this.next_state.destination);
                                    index = next_index;
                              }
                        }

            }

      }

      /**
       * methodos i opoia epistrefi to pseudo-reward gia ena maxnode
otan vriskete
       * se kapia katastasi
       *
       * @param maxnode
       * @param s
       * @return
       */
      public int getPseudoreward(int maxnode, State_Variables s) {
            if (this.isTerminal(maxnode, s))
                  return 0;
            else
                  return -100;
      }
```

```
	/**
	 * methodos i opoia vazi to childSeq stin arxi tou seq
	 *
	 * @param childSeq
	 * @param seq
	 */
	public void appendChildSeqOntoSeq(ArrayList<State_Variables>
childSeq,
			ArrayList<State_Variables> seq) {
		for (int i = childSeq.size() - 1; i >= 0; i--)
			seq.add(0, childSeq.get(i));
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MaxQ_PHC maxq = new MaxQ_PHC();
		// maxq.calculate(500);
	}

}
```

MaxQ_WoLF_PHC.java

```
import java.util.ArrayList;
import java.util.Stack;

/**
 * klasi i opoia ilopoiei ton algorithmo MAXQ-WoLF-PHC
 *
 * @author giannis
 *
 */
public class MaxQ_WoLF_PHC extends MaxQ_PHC {

	private double deltaW; // learning rate for winning
	private double deltaL; // learning rate for losing

	/***************** methodoi set get ******************/

	/**
	 * methodos i opoia arxikopoiei to deltaW
	 *
	 * @param deltaW
	 */
	public void setDeltaW(double deltaW) {
		this.deltaW = deltaW;
	}

	/**
	 * methodos i opoia epistrefi to deltaW
	 *
```

```
	 * @return
	 */
	public double getDeltaW() {
		return deltaW;
	}

	/**
	 * methodos i opoia arxikopoiei to deltaL
	 *
	 * @param deltaL
	 */
	public void setDeltaL(double deltaL) {
		this.deltaL = deltaL;
	}

	/**
	 * methodos i opoia epistrefi to deltaL
	 *
	 * @return
	 */
	public double getDeltaL() {
		return deltaL;
	}

	/**
	 * update estimate of average policy p
	 *
	 * @param index
	 * @param action
	 */
	public void updateAvgPolicyValue(int index, int action, int
maxNode) {
		int length = ((MaxNodeWoLFPHC) this.graph.get(maxNode))
				.getAvgPolicyValueLength();
		((MaxNodeWoLFPHC)
this.graph.get(maxNode)).addOneToC(index);
		for (int i = 0; i < length; i++)
			((MaxNodeWoLFPHC) this.graph.get(maxNode))
					.addAvgPolicyValue(
							index,
							i,
							(((double) 1 /
((MaxNodeWoLFPHC) this.graph

	.get(maxNode)).getC(index)) * (((MaxNodeWoLFPHC) this.graph

	.get(maxNode)).getPolicyValue(index, i) - ((MaxNodeWoLFPHC)
this.graph

	.get(maxNode)).getAvgPolicyValue(index, i))));
	}

	/**
	 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
ena state
	 * tou avgPolicyValue
	 *
	 * @param index
```

```
	 * @return
	 */
	public double findMinNegativeAvgPolicyValue(int index, int
maxNode) {
		int length = ((MaxNodeWoLFPHC) this.graph.get(maxNode))
				.getAvgPolicyValueLength();
		double temp = 0;
		for (int i = 0; i < length; i++) {
			if (((MaxNodeWoLFPHC)
this.graph.get(maxNode)).getAvgPolicyValue(
					index, i) < temp) {
				temp = ((MaxNodeWoLFPHC)
this.graph.get(maxNode))
						.getAvgPolicyValue(index, i);
			}
		}
		return temp;
	}

	/**
	 * methodos i opoia kanonikopoiei tis pithanottites tou
avgPolicyValue meta
	 * to update
	 *
	 * @param index
	 */
	public void normalizeAvgPolicyValue(int index, int maxNode) {
		int length = ((MaxNodeWoLFPHC) this.graph.get(maxNode))
				.getAvgPolicyValueLength();
		double minNegative =
this.findMinNegativeAvgPolicyValue(index, maxNode);
		double temp = Math.abs(2 * minNegative);
		double sum = 0;
		for (int i = 0; i < length; i++)
			((MaxNodeWoLFPHC)
this.graph.get(maxNode)).addAvgPolicyValue(index,
					i, temp);
		for (int i = 0; i < length; i++)
			sum += ((MaxNodeWoLFPHC) this.graph.get(maxNode))
					.getAvgPolicyValue(index, i);
		for (int i = 0; i < length; i++)
			((MaxNodeWoLFPHC)
this.graph.get(maxNode)).setAvgPolicyValue(index,
					i, (((MaxNodeWoLFPHC)
this.graph.get(maxNode))
							.getAvgPolicyValue(index, i)
/ sum));

	}
```

```
	/**
	 * methodos i opoia vriski pio delta (winning or losing) tha
	 * xrisimopoihthei gia na ginoun update oi times tou policyValue
	 *
	 * @param index
	 * @return
	 */
	public double chooseDeltaForUpdate(int index, int maxNode) {
		int length = ((MaxNodeWoLFPHC) this.graph.get(maxNode))
				.getPolicyValueLength();
		double sumPolicy = 0;
		double sumAvgPolicy = 0;
		for (int i = 0; i < length; i++) {
			sumPolicy += (((MaxNodeWoLFPHC)
this.graph.get(maxNode))
					.getPolicyValue(index, i) *
this.findValue(maxNode, index));
			sumAvgPolicy += (((MaxNodeWoLFPHC)
this.graph.get(maxNode))
					.getAvgPolicyValue(index, i) *
this.findValue(maxNode,
					index));
			;
		}
		if (sumPolicy > sumAvgPolicy)
			return this.getDeltaW();
		else
			return this.getDeltaL();
	}

	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi (simfona me to megalo to arthro)
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue2(int index, int action, int
maxNode) {
		int length = ((MaxNodeWoLFPHC) this.graph.get(maxNode))
				.getPolicyValueLength();
		double delta = this.chooseDeltaForUpdate(index, maxNode);
		if (action != this.findGreedyAction(maxNode, index)) {
			((MaxNodeWoLFPHC)
this.graph.get(maxNode)).addPolicyValue(index,
					action, Math.min(((MaxNodeWoLFPHC)
this.graph.get(maxNode))
							.getPolicyValue(index,
action),
							(delta / (length - 1))));
		} else {
			double sum = 0;
			for (int i = 0; i < length; i++) {
				if (i == action)
					continue;
				sum += Math.min(((MaxNodeWoLFPHC)
this.graph.get(maxNode))
```

```
                                        .getPolicyValue(index, i), (delta
/ (length - 1)));
                  }
                  ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).addPolicyValue(index,
                              action, sum);
            }
      }

      /**
       * methodos i opoia kani update tin timi tis pithanotitas gia ti
       * sigkekrimeni katastasi
       *
       * @param index
       * @param action
       */
      public void updatePolicyValue(int index, int action, int
maxNode) {
            double delta = this.chooseDeltaForUpdate(index, maxNode);
            if (action == this.findGreedyAction(maxNode, index))
                  ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).addPolicyValue(index,
                              action, delta);
            else
                  ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).addPolicyValue(index,
                              action, ((-delta) / (((MaxNodeWoLFPHC)
this.graph

      .get(maxNode)).getPolicyValueLength() - 1)));
      }

      /**
       * methodos i opoia dimiourgei k arxikopoiei tous pinakes
avgPolicyValues se
       * olous tous komvous tou grafou
       */
      public void makeAvgPolicyValues() {
            for (int i = 0; i < this.numMaxNodes; i++) {
                  ((MaxNodeWoLFPHC)
this.graph.get(i)).makeNewAvgPolicyValue(
                              SA_Gridworld.states,
this.graph.get(i).children.size());
                  ((MaxNodeWoLFPHC)
this.graph.get(i)).initAvgPolicyValue(
                              SA_Gridworld.states,
this.graph.get(i).children.size());
            }
      }

      /**
       * methodos i opoia dimiourgei k arxikopoiei tous pinakes C
       * olous tous komvous tou grafou
       */
      public void makeC(int num) {
            for (int i = 0; i < this.numMaxNodes; i++) {
                  ((MaxNodeWoLFPHC)
this.graph.get(i)).makeNewC(SA_Gridworld.states);
```

```
                        ((MaxNodeWoLFPHC) this.graph.get(i)).initC(num);
                }
        }

        /**
         * methodos i opia ekpedevi ton agent
         */
        public void calculate(int max_episode, double epsilon, double
df,
                        double deltaW, double deltaL, String filename,double
lr) {
                SA_Gridworld.makeGridworld(filename);
                double timer;
                this.setEpsilon(epsilon);
                this.setDiscountFactor(df);
                this.setDeltaL(deltaL);
                this.setDeltaW(deltaW);
                this.graph.clear();
                this.makeGraph(SA_Gridworld.states,2);
                this.makePolicyValues();
                this.makeAvgPolicyValues();
                this.makeC(0);
                State_Variables s;
                this.initLearningRate(lr);
                this.makeNewEpisodeSteps(max_episode);
                this.makeNewTimePerEpisode(max_episode);
                for (int i = 0; i < max_episode; i++) {
                        timer = (double) System.currentTimeMillis() / 1000;
                        // System.out.println("episode=" + i);
                        s = new State_Variables(SA_Gridworld.size_x,
SA_Gridworld.size_y,
                                        SA_Gridworld.numOfStartPosition,
SA_Gridworld.numOfGoals);
                        this.next_state = new
State_Variables(SA_Gridworld.size_x,
                                        SA_Gridworld.size_y,
SA_Gridworld.numOfStartPosition,
                                        SA_Gridworld.numOfGoals);
                        this.maxQ_Wolf_phc(this.MaxRoot, s, i);
                        //this.maxQ_Wolf_phc_iterative(this.MaxRoot, s, i);
                        this.setTimePerEpisode(i,
                                        ((double) System.currentTimeMillis() /
1000 - timer));
                }

                /*
                 * PrintInFile p = new PrintInFile();
                 * p.printResults("MaxQ_WoLF_PHC-results2.txt",
this.getEpisodeSteps(),
                 * max_episode); p.printResults("MaxQ_WoLF_PHC-clock
time2.txt",
                 * this.getTimePerEpisode(), max_episode);
                 */
        }
```

```
        /**
         * methodos i opoia ilopoiei ton MAXQ-WoLF-PHC algorithmo
         *
         * @param maxNode
         * @param s
         * @param episode
         * @return
         */
        public ArrayList<State_Variables> maxQ_Wolf_phc(int maxNode,
                    State_Variables s, int episode) {

                ArrayList<State_Variables> seq = new
ArrayList<State_Variables>();
                int reward = 0;
                int index = 0;
                double lr = 0;
                // int count = 0;
                // int n = 0;

                // an einai Qnode tote pernoume to child tou (ton maxnode
pou exteli to
                // action)
                if (this.graph.get(maxNode) instanceof Qnode) {
                    // kathorizoume tin parametro t gia to
MaxNavigate(t)
                    if (maxNode == this.QNavigateForGet) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                .getPasLoc(s.passenger));
                    } else if (maxNode == this.QNavigateForPut) {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                        ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).setDest(SA_Gridworld
                                .getDestLoc(s.destination));
                    } else {
                        maxNode = ((Qnode)
this.graph.get(maxNode)).children.get(0);
                    }
                }

                // i periptosi opou o maxNode einai primitive
                if (((MaxNodeWoLFPHC)
this.graph.get(maxNode)).isPrimitive()) {
                    index = this.findIndex(s);
                    State_Variables nextState = new State_Variables();
                    reward = this.takeAction(s, nextState,
                            ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).getAction());
                    this.setNext_state(nextState);
                    lr = ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).getLearningRate();
                    ((MaxNodeWoLFPHC) this.graph.get(maxNode))
                            .setValueFunctForPrimitive(index, (1 -
lr)
```

```
                                                * ((MaxNodeWoLFPHC)
this.graph.get(maxNode))

      .getValueFunctForPrimitive(index)
                                                + (lr * reward));

                  // System.out.println("s.destination=" +
s.destination
                  // + "\ns.passenger=" + s.passenger + "\ns.taxi=" +
s.taxi);

                  // System.exit(-1);
                  State_Variables tmp = new State_Variables(s);
                  seq.add(0, tmp);
                  this.addEpisodeSteps(episode);
            } else {// i periptosi opou o maxnode einai composite (not
primitive)
                  count = 0;
                  int action = 0;
                  int childAction = 0;
                  int next_index = 0;
                  while (!(this.isTerminal(maxNode, s))) {
                        ArrayList<State_Variables> childSeq = new
ArrayList<State_Variables>();
                        index = this.findIndex(s);

                        // choose action
                        childAction = this.chooseAction(index,
                                    ((MaxNodeWoLFPHC)
this.graph.get(maxNode))
                                                .getPolicyValue());
                        action = ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).children
                                    .get(childAction);

                        this.updatePolicyValue(index, childAction,
maxNode);
                        this.normalizePolicyValue(index, maxNode);
                        this.updateAvgPolicyValue(index, childAction,
maxNode);
                        this.normalizeAvgPolicyValue(index, maxNode);

                        // einai to childSeq= tou algorithmou
                        this.setChildSeq(this.maxQ_Wolf_phc(action, s,
episode),
                                    childSeq);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNodeWoLFPHC)
this.graph.get(maxNode))
                                    .getLearningRate();

                        this
                                    .setGreedyAction(((MaxNodeWoLFPHC)
this.graph

      .get(maxNode)).children.get(this
```

```
      .findGreedyAction(maxNode, next_index)));
                        this.setN(1);
                        int tmp_index = 0;
                        for (int i = 0; i < childSeq.size(); i++) {
                              tmp_index =
this.findIndex(childSeq.get(i));

                              // set Cinside
                              ((Qnode) this.graph.get(action))
                                          .setCinsideValue(
                                                      tmp_index,
                                                      ((1 - lr)
                                                                  *
((Qnode) this.graph.get(action))

            .getCinsideValue(tmp_index) + (lr
                                                                  *
Math

            .pow(

                        this

                                    .getDiscountFactor(),

                        n) * (this

      .getPseudoreward(maxNode,

            this.next_state)
                                                                  +
((Qnode) this.graph.get(this

            .getGreedyAction()))

            .getCinsideValue(next_index) + this

      .findValue(this.getGreedyAction(),

            tmp_index)))));

                              // set C value
                              ((Qnode) this.graph.get(action))
                                          .setCvalue(
                                                      tmp_index,
                                                      ((1 - lr)
                                                                  *
((Qnode) this.graph.get(action))

            .getCvalue(tmp_index) + (lr
                                                                  *
Math

            .pow(

                        this
```

```
                                        .getDiscountFactor(),

                                n) * (((Qnode) this.graph

        .get(this.getGreedyAction()))

        .getCvalue(next_index) + this

        .findValue(this.getGreedyAction(),

              next_index)))));

                                    this.addOneToN();
                                }
                                // append childSeq onto the front of seq
                                this.appendChildSeqOntoSeq(childSeq, seq);

                                s = new State_Variables(this.next_state.taxi,
                                            this.next_state.passenger,
this.next_state.destination);
                                index = next_index;

                          }
                    }

                    return seq;
              }

              /**
               * methodos i opoia ilopoiei ton MAXQ-PHC algorithmo (iterative)
               *
               * @param maxNode
               * @param s
               * @param episode
               * @return
               */
              public void maxQ_Wolf_phc_iterative(int node, State_Variables s,
int episode) {
                    Stack<ArrayList<State_Variables>> seqStack = new
Stack<ArrayList<State_Variables>>();
                    Stack<Integer> stack = new Stack<Integer>();
                    stack.push(node);
                    int reward = 0;
                    int index = 0;
                    double lr = 0;
                    // int count = 0;
                    // int n = 0;
                    int maxNode;
                    int action = 0;
                    int childAction = 0;
                    int next_index = 0;

                    while (!stack.isEmpty()) {
                          ArrayList<State_Variables> seq = new
ArrayList<State_Variables>();
                          seqStack.push(seq);
                          maxNode = stack.peek();
```

```
                    // elegxos an imaste se Max node
                    maxNode = checkInstanceOf(maxNode, s);
                    // i periptosi opou o maxNode einai primitive
                    if (((MaxNodeWoLFPHC)
this.graph.get(maxNode)).isPrimitive()) {
                        index = this.findIndex(s);
                        State_Variables nextState = new
State_Variables();
                        reward = this.takeAction(s, nextState,
                                ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).getAction());
                        this.setNext_state(nextState);
                        lr = ((MaxNodeWoLFPHC)
this.graph.get(maxNode))
                                .getLearningRate();
                        ((MaxNodeWoLFPHC) this.graph.get(maxNode))
                                .setValueFunctForPrimitive(index,
(1 - lr)
                                        * ((MaxNodeWoLFPHC)
this.graph.get(maxNode))

      .getValueFunctForPrimitive(index)
                                        + (lr * reward));

                        // System.out.println("s.destination=" +
s.destination
                        // + "\ns.passenger=" + s.passenger +
"\ns.taxi=" + s.taxi);

                        // System.exit(-1);
                        State_Variables tmp = new State_Variables(s);
                        seqStack.peek().add(0, tmp);
                        this.addEpisodeSteps(episode);
                        stack.pop();
                        maxNode = stack.peek();

                        // elegxos an imaste se Max node
                        maxNode = checkInstanceOf(maxNode, s);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNodeWoLFPHC)
this.graph.get(maxNode))
                                .getLearningRate();
                        this
                                .setGreedyAction(((MaxNodeWoLFPHC)
this.graph

      .get(maxNode)).children.get(this

      .findGreedyAction(maxNode, next_index)));

                        changeCompletionValue(seqStack, index,
next_index, action,
                                maxNode, lr);
                        s = new State_Variables(this.next_state.taxi,
                                this.next_state.passenger,
this.next_state.destination);
```

```
                        index = next_index;

                  } else {
                        if (!(this.isTerminal(maxNode, s))) {
                              index = this.findIndex(s);

                              // choose action
                              childAction = this.chooseAction(index,
                                          ((MaxNodeWoLFPHC)
this.graph.get(maxNode))

      .getPolicyValue());
                              action = ((MaxNodeWoLFPHC)
this.graph.get(maxNode)).children
                                          .get(childAction);

                              this.updatePolicyValue2(index,
childAction, maxNode);
                              this.normalizePolicyValue(index,
maxNode);
                              this.updateAvgPolicyValue(index,
childAction, maxNode);
                              this.normalizeAvgPolicyValue(index,
maxNode);
                              stack.push(action);
                        } else {
                              stack.pop();
                              if (stack.isEmpty())
                                    break;
                              maxNode = stack.peek();

                              // elegxos an imaste se Max node
                              maxNode = checkInstanceOf(maxNode, s);
                              // observe next state
                              next_index =
this.findIndex(this.next_state);
                              lr = ((MaxNodeWoLFPHC)
this.graph.get(maxNode))
                                          .getLearningRate();

                              this.setGreedyAction((MaxNodeWoLFPHC)
this.graph

      .get(maxNode)).children.get(this.findGreedyAction(
                                          maxNode, next_index)));
                              changeCompletionValue(seqStack, index,
next_index, action,
                                          maxNode, lr);
                              s = new
State_Variables(this.next_state.taxi,
                                          this.next_state.passenger,

      this.next_state.destination);
                              index = next_index;
                        }
                  }

            }
```

```
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MaxQ_WoLF_PHC m = new MaxQ_WoLF_PHC();
		//m.calculate(500);

	}
}
```

MA_Enironment.java

```
import java.util.Random;

public abstract class MA_Environment {

	public static Random r = new Random();
	// private double[][] Q;//lookup table
	private int[] episodeSteps;
	private double[] timePerEpisode;
	private double discount_factor;
	private double learning_rate;
	private double lr0 = 0.8;
	private double epsilon0;
	private double epsilon;
	private double timeConst;
	private double temperature;// xrisimopoieite sto boltzmann
exploration
	private double coolingRate; // xrisimopoieite gia na miwnete to
temperature
	private double offset = 1; // xrisimopieite gia tin allagi
thermokrasias

	private int[] ranges;
	private int[] offsetTable;

	/****** methodoi set,get,add gia tis private metavlites ******/

	/**
	 * methodos i opoia dimiourga ton pinaka episodeSteps
	 *
	 * @param max_episode
	 */
	abstract void makeNewEpisodeSteps(int max_episode);

	/**
	 * methodos i opoia dimiourgei ton pinaka episodeSteps
	 *
	 * @param max_episode
	 */
	abstract void makeNewTimePerEpisode(int max_episode);
```

```
/**
 * methodos i opoia dini timi sta steps tou episodiou i
 *
 * @param i
 * @param steps
 */
abstract void setTimePerEpisode(int i, double time);

/**
 * methodos i opoia epistrefi ton pinaka timePerEpisode
 *
 * @return
 */
abstract double[] getTimePerEpisode();

/**
 * methodos i opoia epistrefi to xrono pou xriastike to
 * episodiou i
 * @param i
 * @return
 */
abstract double getTimePerEpisode(int i);

/**
 * methodos i opoia epistrefi ton pinaka episodeSteps
 *
 * @return
 */
abstract int[] getEpisodeSteps();

/**
 * methodos i opoia dini timi sta steps tou episodiou i
 *
 * @param i
 * @param steps
 */
abstract void setEpisodeSteps(int i, int steps);

/**
 * methodos i opoia epistrefi ta steps tou episodiou i
 *
 * @param i
 * @return
 */
abstract int getEpisodeSteps(int i);
/**
 * methodos i opoia af3ani kata 1 ta steps tou episodiou i
 *
 * @param i
 */
abstract void addEpisodeSteps(int i);
/**
 * methodos i opoia kathorizei to learning rate
 *
 * @param num
 */
abstract void setLearningRate(double num);
```

```
/**
 * methodos i opoia epistrefi to learning rate
 *
 * @return
 */
abstract double getLearningRate();

/**
 * methodos i opoia kathorizei to discount factor
 *
 * @param num
 */
abstract void setDiscountFactor(double num);

/**
 * methodos i opoia epistrefi to discount factor
 *
 * @return
 */
abstract double getDiscountFactor();

/**
 * methodos i opoia kathorizei to epsilon
 *
 * @param num
 */
abstract void setEpsilon(double num);

/**
 * methodos i opoia epistrefi to epsilon
 *
 * @return
 */
abstract double getEpsilon();

/**
 * methodos i opoia epistrefi to epsilon0
 *
 * @return
 */
abstract double getEpsilon0();

/**
 * methodos i opoia kathorizi to temperature
 *
 * @param temperature
 */
abstract void setTemperature(double temperature);

/**
 * methodos i opoia epistrefi to temperature
 *
 * @return
 */
abstract double getTemperature();
```

```
	/**
	 * methodos i opoia kathorizi to coolingRate
	 *
	 * @param coolingRate
	 *            the coolingRate to set
	 */
	abstract void setCoolingRate(double coolingRate);

	/**
	 * methodos i opoia epistrefi to coolingRate
	 *
	 * @return the coolingRate
	 */
	abstract double getCoolingRate();

	/**
	 * methodos i opoia kathorizi ta ranges
	 */
	abstract void setRanges();

	/**
	 * @return the ranges
	 */
	abstract int[] getRanges();

	/**
	 * @param offsetTable
	 *            the offsetTable to set
	 */
	abstract void setOffsetTable(int[] offsetTable);

	/**
	 * @return the offsetTable
	 */
	abstract int[] getOffsetTable();

	/************************************************/

	/**
	 * methodos i opoia arxikopoiei ton pinaka offsetTable
	 */
	abstract void initOffsetTable();
	/**
	 * methodos i opoia ipologizi to index apo ena vector (vriski
tin katastasi
	 * pou imaste sto perivallon,tin grammi tou Q table)
	 *
	 * @param s
	 * @return
	 */
	abstract int findIndex(MA_State_Variables s);
	/**
	 * methodos i opoia vriski tin thesi tou max action se ena state
	 *
	 * @param index
	 * @return
	 */
	abstract int findMaxQ(int index, double[][] Q);
```

```
        /**
         * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
         * sigkekrimeni katastasi (state) simfona me tin policy e-greedy
         *
         * @param index
         * @return
         */
        abstract int chooseAction(int index, double[][] Q);

        /****************** Boltzmann Exploration ********************/

        /**
         * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
sigkekrimeni
         * katastasi (state) simfona me tin Boltzmann Exploration
         *
         * @param index
         * @return
         */
        abstract int chooseActionWithBoltzmannExploration(int index,
double[][] Q);

        /**
         * methodos i opoia kanonikopoiei tis pithanottites enos pinaka
         *
         * @param index
         */
        abstract void normalizeProbability(double[] table);

        /**
         * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
         * ena pinaka pithanotitwn
         *
         * @param index
         * @return
         */
        abstract double findMinNegativeValue(double[] table);

        /**
         * methodos i opoia epilegi mia kinisi (action) vasi tis
         * pithanotitas p(s,a)
         * @param index
         * @return
         */
        abstract int chooseProbAction(double[] table);
        /**
         * methodos i opoia vriski pia thesi tou pinaka exi ti
         * megaliteri timi
         * @param table
         * @return
         */
        abstract int findMax(double[] table);
        /**
         * methodos i opoia allazi to temperature vasi kapiou cooling
rate
         */
        abstract void changeTemperature(int numStep);
```

```
	/**
	 * methodos i opoia allazti to epsilon0 se kathe iteration
	 *
	 * @param i
	 * @param max_iteration
	 */
	abstract void changeEpsilon0(int i, int max_episode);

	/**
	 * methodos i opoia allazi to epsilon se kathe step tou
	 * episodiou
	 * @param episode
	 */
	abstract void changeEpsilon(int step);

	/**
	 * methodos i opoia allazi to learning rate se kathe step tou
	 * episodiou
	 * @param episode
	 */
	abstract void changeLearningRate(int episode);

	/**
	 * methodos i opoia allazi to timeConst gia kathe epsilon0
	 */
	abstract void changeTimeConst();

}
```

MA_ Taxi_Problem.java

```
import java.util.Random;

/**
 * klasi i opoia epiliei to taxi problem gia multi agent
 */
public class MA_Taxi_Problem extends MA_Environment{

	public static Random r = new Random();
	// private double[][] Q;//lookup table
	private int[] episodeSteps;
	private double[] timePerEpisode;
	private double discount_factor;
	private double learning_rate;
	private double lr0 = 0.8;
	private double epsilon0 ;
	private double epsilon;
	private double timeConst;
	private double temperature;// xrisimopoieite sto boltzmann
exploration
	private double coolingRate; // xrisimopoieite gia na miwnete to
temperature
	private double offset = 1; // xrisimopieite gia tin allagi
thermokrasias
```

```
private int[] ranges;
private int[] offsetTable;

/*****methodoi set,get,add gia tis private metavlites ********/

/**
 * methodos i opoia dimiourga ton pinaka episodeSteps
 *
 * @param max_episode
 */
public void makeNewEpisodeSteps(int max_episode) {
      this.episodeSteps = new int[max_episode];
      // initialize episodeSteps
      for (int i = 0; i < max_episode; i++) {
            // this.episodeSteps[i] = 0;
            this.setEpisodeSteps(i, 0);
      }
}

/**
 * methodos i opoia dimiourgei ton pinaka episodeSteps
 *
 * @param max_episode
 */
public void makeNewTimePerEpisode(int max_episode) {
      this.timePerEpisode = new double[max_episode];
      // initialize episodeSteps
      for (int i = 0; i < max_episode; i++) {
            // this.episodeSteps[i] = 0;
            this.setTimePerEpisode(i, 0.0);
      }
}

/**
 * methodos i opoia dini timi sta steps tou episodiou i
 *
 * @param i
 * @param steps
 */
public void setTimePerEpisode(int i, double time) {
      this.timePerEpisode[i] = time;
}

/**
 * methodos i opoia epistrefi ton pinaka timePerEpisode
 *
 * @return
 */
public double[] getTimePerEpisode() {
      return this.timePerEpisode;

}
```

```java
/**
 * methodos i opoia epistrefi to xrono pou xriastike to
 * episodiou i
 * @param i
 * @return
 */
public double getTimePerEpisode(int i) {
	return this.timePerEpisode[i];
}

/**
 * methodos i opoia epistrefi ton pinaka episodeSteps
 *
 * @return
 */
public int[] getEpisodeSteps() {
	return this.episodeSteps;

}

/**
 * methodos i opoia dini timi sta steps tou episodiou i
 *
 * @param i
 * @param steps
 */
public void setEpisodeSteps(int i, int steps) {
	this.episodeSteps[i] = steps;
}

/**
 * methodos i opoia epistrefi ta steps tou episodiou i
 *
 * @param i
 * @return
 */
public int getEpisodeSteps(int i) {
	return this.episodeSteps[i];
}

/**
 * methodos i opoia af3ani kata 1 ta steps tou episodiou i
 *
 * @param i
 */
public void addEpisodeSteps(int i) {
	this.episodeSteps[i]++;
}

/**
 * methodos i opoia kathorizei to learning rate
 *
 * @param num
 */
public void setLearningRate(double num) {
	this.learning_rate = num;
}
```

```
/**
 * methodos i opoia epistrefi to learning rate
 *
 * @return
 */
public double getLearningRate() {
	return this.learning_rate;
}

/**
 * methodos i opoia kathorizei to discount factor
 *
 * @param num
 */
public void setDiscountFactor(double num) {
	this.discount_factor = num;
}

/**
 * methodos i opoia epistrefi to discount factor
 *
 * @return
 */
public double getDiscountFactor() {
	return this.discount_factor;
}
/**
 * methodos i opoia kathorizei to epsilon
 *
 * @param num
 */
public void setEpsilon(double num) {
	this.epsilon = num;
}
/**
 * methodos i opoia epistrefi to epsilon
 *
 * @return
 */
public double getEpsilon() {
	return this.epsilon;
}
/**
 * methodos i opoia epistrefi to epsilon0
 *
 * @return
 */
public double getEpsilon0() {
	return this.epsilon0;
}
	/**
 * methodos i opoia kathorizi to temperature
 *
 * @param temperature
 */
public void setTemperature(double temperature) {
	this.temperature = temperature;
}
```

```
	/**
	 * methodos i opoia epistrefi to temperature
	 *
	 * @return
	 */
	public double getTemperature() {
		return temperature;
	}

	/**
	 * methodos i opoia kathorizi to coolingRate
	 *
	 * @param coolingRate
	 *            the coolingRate to set
	 */
	public void setCoolingRate(double coolingRate) {
		this.coolingRate = coolingRate;
	}

	/**
	 * methodos i opoia epistrefi to coolingRate
	 *
	 * @return the coolingRate
	 */
	public double getCoolingRate() {
		return coolingRate;
	}

	/**
	 * methodos i opoia kathorizi ta ranges
	 */
	public void setRanges() {
		ranges = new int[MA_Gridworld.numOfAgents +
MA_Gridworld.numOfPassengers
				+ MA_Gridworld.numOfPassengers];

		for (int i = 0; i < MA_Gridworld.numOfAgents; i++)
			this.ranges[i] = MA_Gridworld.size_x *
MA_Gridworld.size_y;
		for (int i = 0; i < MA_Gridworld.numOfPassengers; i++)
			this.ranges[i + MA_Gridworld.numOfAgents] =
MA_Gridworld.numOfStartPosition + 1;
		for (int i = 0; i < MA_Gridworld.numOfPassengers; i++)
			this.ranges[i + MA_Gridworld.numOfAgents +
MA_Gridworld.numOfPassengers] = MA_Gridworld.numOfGoals;
	}

	/**
	 * @return the ranges
	 */
	public int[] getRanges() {
		return ranges;
	}
```

```
	/**
	 * @param offsetTable
	 *            the offsetTable to set
	 */
	public void setOffsetTable(int[] offsetTable) {
		this.offsetTable = offsetTable;
	}

	/**
	 * @return the offsetTable
	 */
	public int[] getOffsetTable() {
		return offsetTable;
	}

	/*************************************************/

	/**
	 * methodos i opoia arxikopoiei ton pinaka offsetTable
	 */
	public void initOffsetTable() {
		offsetTable = new int[MA_Gridworld.numOfAgents +
MA_Gridworld.numOfPassengers
				+ MA_Gridworld.numOfPassengers];

		int product = 1;
		for (int i = offsetTable.length - 1; i >= 0; --i) {
			offsetTable[i] = product;
			product *= ranges[i];
		}
	}

	/**
	 * methodos i opoia ipologizi to index apo ena vector (vriski
tin katastasi
	 * pou imaste sto perivallon,tin grammi tou Q table)
	 *
	 * @param s
	 * @return
	 */
	public int findIndex(MA_State_Variables s) {
		int offset = 0;
		for (int i = 0; i < s.taxi.size(); ++i) {
			offset += s.taxi.get(i) * offsetTable[i];
		}
		for (int i = 0; i < s.passenger.size(); ++i) {
			offset += s.passenger.get(i) * offsetTable[i +
s.taxi.size()];
		}
		for (int i = 0; i < s.destination.size(); ++i) {
			offset += s.destination.get(i)
					* offsetTable[i + s.taxi.size() +
s.passenger.size()];
		}
		return offset;

	}
```

```
	/**
	 * methodos i opoia vriski tin thesi tou max action se ena state
	 *
	 * @param index
	 * @return
	 */
	public int findMaxQ(int index, double[][] Q) {
		double temp = Q[index][0];
		int pos = 0;
		for (int i = 1; i < MA_Gridworld.actions; i++) {
			// for (int i = 1; i < 6; i++) {
			if (Q[index][i] > temp) {
				temp = Q[index][i];
				pos = i;
			}
		}
		return pos;
	}

	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
sigkekrimeni
	 * katastasi (state) simfona me tin policy e-greedy
	 *
	 * @param index
	 * @return
	 */
	public int chooseAction(int index, double[][] Q) {
		if (r.nextDouble() < this.epsilon)
			return (r.nextInt(MA_Gridworld.actions));
		// return (r.nextInt(6));
		return this.findMaxQ(index, Q);
	}

	/***************** Boltzmann Exploration ******************/

	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
sigkekrimeni
	 * katastasi (state) simfona me tin Boltzmann Exploration
	 *
	 * @param index
	 * @return
	 */
	public int chooseActionWithBoltzmannExploration(int index,
double[][] Q) {
		double[] probTable = new double[MA_Gridworld.actions];
		double sum = 0.0;
		for (int i = 0; i < MA_Gridworld.actions; i++){

	//System.out.println("Q[index]["+i+"]="+Q[index][i]);
			//System.out.println(" this.getTemperature()="+
this.getTemperature());
			//System.out.println("Math.exp((Q[index][i] /
this.getTemperature()))"+Math.exp((Q[index][i] /
this.getTemperature())));
			sum += Math.exp((Q[index][i] /
this.getTemperature()));
```

```
			}
		//	 for (int i = 0; i < probTable.length; i++)
		//
		System.out.println("probTable["+i+"]="+probTable[i]);

				for (int i = 0; i < MA_Gridworld.actions; i++)
					probTable[i] = Math.exp((Q[index][i] /
this.getTemperature()))
								/ sum;

				//	for (int i = 0; i < probTable.length; i++)
				//
		System.out.println("probTable["+i+"]="+probTable[i]);
				//	if(this.getTemperature()==0)
				//	System.exit(0);
				this.normalizeProbability(probTable);
				return this.chooseProbAction(probTable);//
this.findMax(probTable);
		}

		/**
		 * methodos i opoia kanonikopoiei tis pithanottites enos pinaka
		 *
		 * @param index
		 */
		public void normalizeProbability(double[] table) {
				double minNegative = this.findMinNegativeValue(table);
				double increment = Math.abs(2 * minNegative);
				double sum = 0;

				for (int i = 0; i < table.length; i++)
					table[i] += increment;
				for (int i = 0; i < table.length; i++)
					sum += table[i];
				for (int i = 0; i < table.length; i++)
					table[i] /= sum;

		//	for (int i = 0; i < table.length; i++)
		//		System.out.println("table["+i+"]="+table[i]);

		//	System.exit(0);
		}
		/**
		 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
		 * ena pinaka pithanotitwn
		 *
		 * @param index
		 * @return
		 */
		public double findMinNegativeValue(double[] table) {
				double temp = 0;
				for (int i = 0; i < table.length; i++) {
					if (table[i] < temp) {
						temp = table[i];
					}
				}
				return temp;
		}
```

```
	/**
	 * methodos i opoia epilegi mia kinisi (action) vasi tis)
	 * pithanotitas p(s,a
	 * @param index
	 * @return
	 */
	public int chooseProbAction(double[] table) {
		double offset = 0;
		double rand = r.nextDouble();
		for (int i = 0; i < table.length; i++) {
			offset += table[i];
			if (offset > rand)
				return i;
		}

		return r.nextInt(table.length);// en tha erti potte dame
	}
	/**
	 * methodos i opoia vriski pia thesi tou pinaka exi ti
	 * megaliteri timi
	 * @param table
	 * @return
	 */
	public int findMax(double[] table) {
		int maxPos = 0;
		double max = table[0];
		for (int i = 0; i < table.length; i++) {
			if (table[i] > max) {
				max = table[i];
				maxPos = i;
			}
		}
		return maxPos;

	}

	/**
	 * methodos i opoia allazi to temperature vasi kapiou cooling
rate
	 */
	public void changeTemperature(int numStep) {
		this.setTemperature(this.offset
				+ (this.temperature *
Math.pow(this.coolingRate, numStep)));
	}

	/**
	 * methodos i opoia allazti to epsilon0 se kathe iteration
	 *
	 * @param i
	 * @param max_iteration
	 */
	public void changeEpsilon0(int i, int max_episode) {
				this.epsilon0 = 0.7 * Math.exp(-((double) i /
max_episode));
	}
```

```
	/**
	 * methodos i opoia allazi to epsilon se kathe step tou
episodiou
	 *
	 * @param episode
	 */
	public void changeEpsilon(int step) {
		this.epsilon=0.09+(this.epsilon0*Math.pow(0.995,step ));
	}

	/**
	 * methodos i opoia allazi to learning rate se kathe step tou
episodiou
	 *
	 * @param episode
	 */
	public void changeLearningRate(int episode) {
		this.learning_rate = this.lr0
				* Math.exp(-((double)
this.getEpisodeSteps(episode) / 1000000));
	}

	/**
	 * methodos i opoia allazi to timeConst gia kathe epsilon0
	 */
	public void changeTimeConst() {
		this.timeConst = Math.log(0.01 / this.epsilon0);
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		// Taxi_Problem t = new Taxi_Problem();
		// t.calculate(500);
	}
}
```

MA_Gridword.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * klasi pou antiprosopevi ena geniko gridworld gia multi agent
 *
 * @author Giannis
 *
 */
public class MA_Gridworld {
```

```
	public static int size_x; // grammes tou grid
	public static int size_y; // stiles tou grid
	public static ArrayList<Wall> wall = new ArrayList<Wall>();;
	public static int numOfStartPosition; // oxi mesa sto ta3i
	public static int numOfGoals;
	public static int numOfPassengers;
	public static int numOfAgents; // taxi
	public static int states;
	public static int actions;
	private static int[] pasLoc; // passenger location
	private static int[] destLoc; // destination location

	// rewards gia kathe action
	public static int wall_hit;
	public static int wrong_put;
	public static int wrong_get;
	public static int success;
	public static int simple_move;
	public static int no_move; // otan o agent diale3i tin pra3i
"idle" dld na
	// min kani kinisi
	public static int collision; // otan o agent sigkrousti me ena
allo agent

	// statheres gia actions
	public static final int north = 0;
	public static final int south = 1;
	public static final int east = 2;
	public static final int west = 3;
	public static final int pickup = 4;
	public static final int putdown = 5;
	public static final int idle = 6;

	public static void setStates() {

		states = (int) ((int) Math.pow(size_x * size_y,
numOfAgents)
				* Math.pow((numOfStartPosition+1),
numOfPassengers) * Math.pow(
				numOfGoals, numOfPassengers));
	}

	/**
	 * methodos i opia epistrefi ton arithmo ton grammwn (x) tou
	 * grid
	 * @return the size_x
	 */
	public static int getSize_x() {
		return size_x;
	}

	/**
	 * methodos i opia epistrefi ton arithmo ton stilon (y) tou grid
	 *
	 * @return the size_y
	 */
	public static int getSize_y() {
		return size_y;
```

```
	}

	/**
	 * methodos i opoia epistrefi ton pinaka me tis sintetagmenes
	 * twn wall
	 * @return the wall
	 */
	public ArrayList<Wall> getWall() {
		return wall;
	}

	/**
	 * methodos i opoia epistrefi ton arithmo ton arxikon
	 * thesewn pou iparxoun sto grid
	 *
	 * @return the numOfStartPosition
	 */
	public static int getNumOfStartPosition() {
		return numOfStartPosition;
	}

	/**
	 * methodos i opoia epistrefi ton arithmon ton telikon thesewn
	 * pou iparxoun sto grid
	 *
	 * @return the numOfGoals
	 */
	public static int getNumOfGoals() {
		return numOfGoals;
	}

	/**
	 * methodos i opoia dimiourgi ena wall meta3i ton dio cells
	 *
	 * @param cell1
	 * @param cell2
	 */
	public static void setWall(int cell1, int cell2) {
		if (((Math.abs(cell1 - cell2) == 1) || ((Math.abs(cell1 -
cell2) == getSize_y())))
				&& (cell1 >= 0)
				&& (cell1 < ((getSize_x() * getSize_y())))
				&& (cell2 >= 0) && (cell2 < ((getSize_x() *
getSize_y())))) {
			Wall temp = new Wall(cell1, cell2);
			wall.add(temp);
		} else {
			System.err
					.println("You cannot enter a wall
between these cells\nExiting..");
			System.exit(-1);
		}

	}
```

```
	/**
	 * methodos i opoia filaei ti thesi tou passenger
	 *
	 * @param i
	 * @param location
	 */
	public static void setPasLoc(int i, int location) {
		pasLoc[i] = location;
	}

	/**
	 * methodos i opoia epistrefi tin thesi tou passenger simfwna me
	 * tin metavliti i
	 *
	 * @param i
	 * @return
	 */
	public static int getPasLoc(int i) {
		return pasLoc[i];
	}

	/**
	 * methodos i opoia anatheti filaei to destination
	 *
	 * @param i
	 * @param location
	 */
	public static void setDestLoc(int i, int location) {
		destLoc[i] = location;
	}

	/**
	 * methodos i opoia epistrefi to destination
	 *
	 * @param i
	 * @return
	 */
	public static int getDestLoc(int i) {
		return destLoc[i];
	}

	/**
	 * methodos i opoia elegxi an iparxi wall stin katefthinsi opou
theli na
	 * metakinithi o agent epistrefi true an iparxi kai false an den
iparxi
	 *
	 * @param cur_pos
	 * @param next_pos
	 * @return
	 */
	public static boolean checkForWall(int cur_pos, int next_pos) {
		for (int i = 0; i < wall.size(); i++) {
			if (((cur_pos == wall.get(i).cell1) && (next_pos ==
wall.get(i).cell2))
				|| ((cur_pos == wall.get(i).cell2) &&
(next_pos == wall
					.get(i).cell1)))
```

```
                        return true;
                }
                return false;
        }

        /**
         * methodos i opoia dimiourga to gridworld
         *
         * @param filename
         */
        public static void makeGridworld(String filename) {
                BufferedReader in = null;
                String[] splitline;

                try {
                        in = new BufferedReader(new FileReader(filename));
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }
                String line = null;

                // grammes tou gridworld
                try {
                        line = in.readLine();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                splitline = line.split("=");
                size_x = Integer.parseInt(splitline[1]);

                // stiles tou gridworld
                try {
                        line = in.readLine();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                splitline = line.split("=");
                size_y = Integer.parseInt(splitline[1]);

                // actions
                try {
                        line = in.readLine();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                splitline = line.split("=");
                actions = Integer.parseInt(splitline[1]);

                // number of start position
                try {
                        line = in.readLine();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                splitline = line.split("=");
                numOfStartPosition = Integer.parseInt(splitline[1]);
                pasLoc = new int[numOfStartPosition + 1];
```

```
		// start positions
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		String[] temp = splitline[1].split(",");
		for (int i = 0; i < temp.length; i++)
			setPasLoc(i, Integer.parseInt(temp[i]));
		// i periptosi opou o passenger vriskete mesa sto taxi
		setPasLoc(numOfStartPosition, -1);

		// number of destinations
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		numOfGoals = Integer.parseInt(splitline[1]);
		destLoc = new int[numOfGoals];

		// destinations
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		temp = splitline[1].split(",");
		for (int i = 0; i < temp.length; i++)
			setDestLoc(i, Integer.parseInt(temp[i]));

		// walls
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		temp = splitline[1].split(",");
		String[] tmp;
		for (int i = 0; i < temp.length; i++) {
			tmp = temp[i].split("-");
			setWall(Integer.parseInt(tmp[0]),
Integer.parseInt(tmp[1]));
		}

		// number of passengers
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		numOfPassengers = Integer.parseInt(splitline[1]);
```

```
// number of agents(taxi)
try {
	line = in.readLine();
} catch (IOException e) {
	e.printStackTrace();
}
splitline = line.split("=");
numOfAgents = Integer.parseInt(splitline[1]);

// reward for wall hit
try {
	line = in.readLine();
} catch (IOException e) {
	e.printStackTrace();
}
splitline = line.split("=");
wall_hit = Integer.parseInt(splitline[1]);

// reward for simple move
try {
	line = in.readLine();
} catch (IOException e) {
	e.printStackTrace();
}
splitline = line.split("=");
simple_move = Integer.parseInt(splitline[1]);

// reward for wrong get
try {
	line = in.readLine();
} catch (IOException e) {
	e.printStackTrace();
}
splitline = line.split("=");
wrong_get = Integer.parseInt(splitline[1]);

// reward for wrong put
try {
	line = in.readLine();
} catch (IOException e) {
	e.printStackTrace();
}
splitline = line.split("=");
wrong_put = Integer.parseInt(splitline[1]);

// reward for no move (idle)
try {
	line = in.readLine();
} catch (IOException e) {
	e.printStackTrace();
}
splitline = line.split("=");
no_move = Integer.parseInt(splitline[1]);

// reward for collision
try {
	line = in.readLine();
} catch (IOException e) {
```

```
			e.printStackTrace();
		}
		splitline = line.split("=");
		collision = Integer.parseInt(splitline[1]);

		// reward for succes
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		success = Integer.parseInt(splitline[1]);

		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}

		setStates();

	}

	public static void main(String[] args) {
		MA_Gridworld.makeGridworld("file.txt");
	}
}
```

MA_Graph.java

```
import java.util.ArrayList;

/**
 * klasi i opoia antiprosopevi ton grafo
 *
 * @author giannis
 *
 */
public class MA_Graph extends Graph {

	ArrayList<Node> graph = new ArrayList<Node>();

	public final int MaxRoot = 0;
	public final int MaxGet = 1;
	public final int MaxPut = 2;
	public final int Pickup = 3;
	public final int Putdown = 4;
	public final int MaxNavigate = 5;
	public final int North = 6;
	public final int East = 7;
	public final int South = 8;
	public final int West = 9;
	public final int Idle=10;
```

```
	public final int Qget = 11;
	public final int Qput = 12;
	public final int Qpickup = 13;
	public final int QNavigateForGet = 14;
	public final int QNavigateForPut = 15;
	public final int Qputdown = 16;
	public final int Qnorth = 17;
	public final int Qeast = 18;
	public final int Qsouth = 19;
	public final int Qwest = 20;
	public final int Qidle=21;

	public final int numMaxNodes = 11;
	public final int numQnodes = 11;

	/**
	 * methodos i opoia dimiourgi ton grafo
	 */
	public void makeGraph(int states,int num) {

		// add the MaxNodes exoume 4 composite kai 6 primitive
		for (int i = 0; i < this.numMaxNodes; i++)
			this.addMaxNode(states,num);

		// add the Qnodes
		for (int i = 0; i < this.numQnodes; i++)
			this.addQnode(states);

		// add childrens
		this.addChild(this.MaxRoot, this.Qget);
		this.addChild(this.MaxRoot, this.Qput);
		this.addChild(this.MaxGet, this.Qpickup);
		this.addChild(this.MaxGet, this.QNavigateForGet);
		this.addChild(this.MaxPut, this.QNavigateForPut);
		this.addChild(this.MaxPut, this.Qputdown);
		this.addChild(this.MaxNavigate, this.Qnorth);
		this.addChild(this.MaxNavigate, this.Qeast);
		this.addChild(this.MaxNavigate, this.Qsouth);
		this.addChild(this.MaxNavigate, this.Qwest);
		this.addChild(this.MaxNavigate, this.Qidle);
		this.addChild(this.Qget, this.MaxGet);
		this.addChild(this.Qput, this.MaxPut);
		this.addChild(this.Qpickup, this.Pickup);
		this.addChild(this.QNavigateForGet, this.MaxNavigate);
		this.addChild(this.QNavigateForPut, this.MaxNavigate);
		this.addChild(this.Qputdown, this.Putdown);
		this.addChild(this.Qnorth, this.North);
		this.addChild(this.Qeast, this.East);
		this.addChild(this.Qsouth, this.South);
		this.addChild(this.Qwest, this.West);
		this.addChild(this.Qidle, this.Idle);

		// add actions
		((MaxNode)
this.graph.get(this.Pickup)).setAction(MA_Gridworld.pickup);
		((MaxNode)
this.graph.get(this.Putdown)).setAction(MA_Gridworld.putdown);
```

```
            ((MaxNode)
this.graph.get(this.North)).setAction(MA_Gridworld.north);
            ((MaxNode)
this.graph.get(this.East)).setAction(MA_Gridworld.east);
            ((MaxNode)
this.graph.get(this.South)).setAction(MA_Gridworld.south);
            ((MaxNode)
this.graph.get(this.West)).setAction(MA_Gridworld.west);

      }

      /**
       * methodos i opoia prostheti ena MaxNode sto grafo
       *
       * @param i
       * @param num
       */
      public void addMaxNode(int i, int num) {
            switch (num) {
            case 0:// gia ton algorithmo MAXQ-Q
                  MaxNode tmp = new MaxNode(i);
                  this.graph.add(tmp);
                  break;
            case 1:// gia ton algorithmo MAXQ-PHC
                  MaxNodePHC tmp1 = new MaxNodePHC(i);
                  this.graph.add(tmp1);
                  break;
            case 2:// gia ton algorithmo MAXQ-WoLF-PHC
                  MaxNodeWoLFPHC tmp2 = new MaxNodeWoLFPHC(i);
                  this.graph.add(tmp2);
                  break;
            default:
                  System.err.println("Invalid number for
num..Exiting..");
                  System.exit(-1);
            }
      }

      /**
       * methodos i opoia prostheti ena Qnode sto grafo
       *
       * @param i
       * @param j
       */
      public void addQnode(int i) {
            Qnode tmp = new Qnode(i);
            this.graph.add(tmp);
      }

      /**
       * methodos i opoia prostheti ena paidi se ena komvo
       *
       * @param parent
       * @param child
       */
      public void addChild(int parent, int child) {
            this.graph.get(parent).children.add(child);
      }
```

```
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

MA_RunTrials.java

```
import org.jfree.data.xy.XYSeries;

public class MA_RunTrials {
	public static double time = 0;

	private int[][] episodeSteps;
	private double[] averageSteps;
	public static int num;

	/*************** set,get for private variables *************/

	/**
	 * @param episodeSteps
	 *            the episodeSteps to set
	 */
	public void setEpisodeSteps(int[][] episodeSteps) {
		this.episodeSteps = episodeSteps;
	}

	/**
	 * methodos i opoia kathorizi ta steps se kapio episodio kapiou
	 * trial
	 * @param i
	 * @param j
	 * @param steps
	 */
	public void setEpisodeSteps(int trial, int[] episodeSteps) {
		for (int i = 0; i < episodeSteps.length; i++)
			this.episodeSteps[trial][i] = episodeSteps[i];
	}

	/**
	 * @return the episodeSteps
	 */
	public int[][] getEpisodeSteps() {
		return episodeSteps;
	}
```

```java
/**
 * methodos i opoia epistrefi ta steps enos episode enos trial
 *
 * @param trial
 * @param episode
 * @return
 */
public int getEpisodeStep(int trial, int episode) {
	return this.episodeSteps[trial][episode];
}

/**
 * methodos i opoia kathorizi ta avarege step enos episode
 *
 * @param i
 * @param steps
 */
public void setAverageSteps(int episode, double avgSteps) {
	this.averageSteps[episode] = avgSteps;
}

/**
 * @return the averageSteps
 */
public double[] getAverageSteps() {
	return averageSteps;
}

/**
 * methodos i opoia epistrefi ta average steps enos trial
 *
 * @param trial
 * @return
 */
public double getAverageSteps(int trial) {
	return averageSteps[trial];
}

/*************************************************************/

/**
 * methodos i opoia dimiourgi ton pinaka episodeSteps
 */
public void makeNewEpisodeSteps(int trials, int episodes) {
	this.episodeSteps = new int[trials][episodes];
}

/**
 * methodos i opoia dimiourgi ton pinaka averageSteps
 *
 * @param trials
 */
public void makeNewAverageSteps(int episodes) {
	this.averageSteps = new double[episodes];
}
```

```
	/**
	 * methodos i opoia vriski to average step gia ena episode
	 *
	 * @param trial
	 * @param episodeSteps
	 * @param trials
	 * @return
	 */
	public double findAvarage(int episode, int[][] episodeSteps, int
trials) {
		double sum = 0;
		for (int i = 0; i < trials; i++) {
			sum += episodeSteps[i][episode];
		}
		return sum / trials;
	}

	/**
	 * methodos stin opoia kathorizete to learning rate
	 *
	 * @return
	 */
	public double chooseLearningRate() {
		double lr;
		do {
			System.out
					.println("Please set the learning rate
(between 0 and 1)");
			lr = StdIn.readDouble();
		} while (lr < 0 || lr > 1);
		return lr;
	}

	/**
	 * methodos stin opoia kathorizete to discount factor
	 *
	 * @return
	 */
	public double chooseDiscountFactor() {
		double df;
		do {
			System.out
					.println("Please set the discount factor
(between 0 and 1)");
			df = StdIn.readDouble();
		} while (df < 0 || df > 1);
		return df;
	}

	/**
	 * methodos stin opoia kathorizete to epsilon
	 *
	 * @return
	 */
	public double chooseEpsilon() {
		double epsilon;
		do {
```

```
                  System.out.println("Please set epsilon(between 0 and
1)");
                  epsilon = StdIn.readDouble();
            } while (epsilon < 0 || epsilon > 1);
            return epsilon;
      }

      /**
       * methodos stin opoia kathorizetai to temperature gia boltzmann
exploration
       *
       */
      public double chooseTemperature() {
            double temperature = 0;
            System.out.println("Please set temperature");
            temperature = StdIn.readDouble();
            return temperature;
      }

      /**
       * methodos stin opoia kathorizetai to coolingRate gia boltzmann
exploration
       *
       */
      public double chooseCoolingRate() {
            double coolingRate = 0;
            do {
                  System.out.println("Please set the cooling rate
(between 0-1)");
                  coolingRate = StdIn.readDouble();
            } while (coolingRate < 0 || coolingRate > 1);
            return coolingRate;
      }

      /**
       * methodos stin opoia ginete epilogi gia exploration meta3i e-
greedy kai
       * boltzmann
       */
      public boolean chooseExploration() {
            boolean exploration;
            System.out
                        .println("Please choose what exploration do
you want from the following:");
            System.out.println("\t0: Botzmann explotation");
            System.out.println("\t1: e-greedy");
            exploration = StdIn.readBoolean();
            return exploration;
      }

      /**
       * methodos i opoia kani plot tin grafiki
       *
       * @param title
       */
      public void plotGraph(String title, int num, String xySeries) {
            // dimiourgia grafikis
            XYSeries steps = new XYSeries(xySeries);
```

```
            for (int i = 0; i < this.getAverageSteps().length; i++) {
                  steps.add((i * num), this.getAverageSteps(i));
            }

            MA_PlotGraph.createChart(steps, title, "Episode", "Average
Steps");
      }

      /**
       * methodos i opoia kani polla trials gia tous diaforous
       * algorithmous kai ipologizi ta average steps tous
       *
       * @param trials
       * @param episodes
       */
      public void calculate(int choice) {

            System.out
                        .println("Please enter how many trials you
would like to run:");
            int trials = StdIn.readInt();
            System.out
                        .println("Please enter how many episodes you
would like to run:");
            int episodes = StdIn.readInt();
            int delivery = 0;
            System.out.println("How many delivery do you want?");
            delivery = StdIn.readInt();
            System.out.println("Please insert the name of the input
file:");
            String filename = StdIn.readString();
            boolean exploration = true;
            boolean changeLr = true;
            boolean changeEpsilon = true;
            double epsilon = 0;
            double temperature = 0;
            double coolingRate = 0;
            double lr = 0;
            double df = 0;
            double delta = 0;
            double deltaW = 0;
            double deltaL = 0;
            PrintInFile p = new PrintInFile();
            String tmp;
            int length;
            System.out.println("Ana posa episodia thelis na gini i
grafiki?");
            num = StdIn.readInt();
            if (num == 1)
                  length = episodes;
            else
                  length = (episodes / num) + 1;
            this.makeNewEpisodeSteps(trials, length);
            this.makeNewAverageSteps(length);
            switch (choice) {
            case 0:// MAQ-Learning
                  exploration = chooseExploration();
```

```
                  if (exploration) {
                        epsilon = chooseEpsilon();
                  } else {
                        temperature = chooseTemperature();
                        coolingRate = chooseCoolingRate();
                  }
                  lr = chooseLearningRate();
                  df = chooseDiscountFactor();
                  MAQ_Learning ql = new MAQ_Learning();
                  MA_Taxi_Problem tp = new MA_Taxi_Problem();
                  for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        ql.calculate(episodes, exploration, tp,
epsilon, temperature,
                                    coolingRate, filename, lr, df,
delivery, num, length);
                        this.setEpisodeSteps(i, tp.getEpisodeSteps());
                        ql.agents.clear();
                  }
                  // for (int i = 0; i < episodes; i++) {
                  for (int i = 0; i < length; i++) {
                        this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                    trials));
                  }
                  if (exploration)
                        p.printAvgResults(
                                    "MAQ-Learning Average Results with
e-greedy.txt", this
                                                .getAverageSteps(),
length);
                  else
                        p
                                    .printAvgResults(
                                                "MAQ-Learning Average
Results with boltzmann exploration.txt",

      this.getAverageSteps(), length);
                  plotGraph("Multi agent Q-Learning Average Steps Per
Episode", num,
                              "Multi Agent Q-Learning");
                  break;
            case 1:// PHC
                  System.out.println("Do you want to dicrease
lr?(yes/no)");
                  tmp = StdIn.readString();
                  if (tmp.equals("yes"))
                        changeLr = true;
                  else
                        changeLr = false;
                  System.out.println("Do you want to dicrease
epsilon?(yes/no)");
                  tmp = StdIn.readString();
                  if (tmp.equals("yes"))
                        changeEpsilon = true;
                  else
                        changeEpsilon = false;
                  lr = chooseLearningRate();
```

```
                    df = chooseDiscountFactor();
                    epsilon = chooseEpsilon();
                    do {
                          System.out.println("Please set delta (between
0 and 1)");
                          delta = StdIn.readDouble();
                    } while (delta < 0 || delta > 1);
                    MA_Phc phc = new MA_Phc();
                    MA_Taxi_Problem tp1 = new MA_Taxi_Problem();
                    for (int i = 0; i < trials; i++) {
                          System.out.println("trial=" + i);
                          phc.calculate(episodes, tp1, num, length,
changeEpsilon,
                                      changeLr, epsilon, lr, df, delta,
filename, delivery);
                          phc.agents.clear();
                          this.setEpisodeSteps(i,
tp1.getEpisodeSteps());
                    }
                    // for (int i = 0; i < episodes; i++) {
                    for (int i = 0; i < length; i++) {
                          this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                      trials));
                    }
                    p.printAvgResults("MA_PHC Average Results.txt", this
                                .getAverageSteps(), length);
                    plotGraph("Multi agent PHC Average Steps Per
Episode", num,
                                "Multi Agent PHC");
                    break;
              case 2:// WoLF-PHC
                    System.out.println("Do you want to dicrease
lr?(yes/no)");
                    tmp = StdIn.readString();
                    if (tmp.equals("yes"))
                          changeLr = true;
                    else
                          changeLr = false;
                    System.out.println("Do you want to dicrease
epsilon?(yes/no)");
                    tmp = StdIn.readString();
                    if (tmp.equals("yes"))
                          changeEpsilon = true;
                    else
                          changeEpsilon = false;
                    MA_Wolf w = new MA_Wolf();
                    MA_Taxi_Problem tp2 = new MA_Taxi_Problem();
                    lr = chooseLearningRate();
                    df = chooseDiscountFactor();
                    epsilon = chooseEpsilon();
                    do {
                          System.out
                                      .println("Please set delta winning
(between 0 and 1)");
                          deltaW = StdIn.readDouble();
                    } while (deltaW < 0 || deltaW > 1);
```

```
                do {
                        System.out.println("Please set delta losing
(between 0 and 1)");
                        deltaL = StdIn.readDouble();
                } while (deltaL < 0 || deltaL > 1);

                for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        w.calculate(episodes, tp2, num, length,
epsilon, changeEpsilon,
                                        changeLr, lr, df, deltaW, deltaL,
filename, delivery);
                        w.agents.clear();

                        this.setEpisodeSteps(i,
tp2.getEpisodeSteps());
                }
                // for (int i = 0; i < episodes; i++) {
                for (int i = 0; i < length; i++) {
                        this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                        trials));
                }
                p.printAvgResults("MA_WoLF-PHC Average Results.txt",
this
                                .getAverageSteps(), length);
                plotGraph("Multi agent WoLF-PHC Average Steps Per
Episode", num,
                                "Multi Agent WoLF-PHC");
                break;
            case 3:// MAXQ-Q
                lr = chooseLearningRate();
                df = chooseDiscountFactor();
                epsilon = chooseEpsilon();
                MA_MAXQ_Q mq = new MA_MAXQ_Q();
                for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        mq.calculate(episodes, num, length, epsilon,
df, filename,
                                        delivery,lr);
                        mq.agents.clear();
                        this.setEpisodeSteps(i, mq.getEpisodeSteps());
                }
                // for (int i = 0; i < episodes; i++) {
                for (int i = 0; i < length; i++) {
                        this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                        trials));
                }
                p.printAvgResults("MA_MAXQ-Q Average Results.txt",
this
                                .getAverageSteps(), length);
                plotGraph("Multi agent MAXQ-Q Average Steps Per
Episode", num,
                                "Multi Agent MAXQ-Q");
                break;
            case 4:// MAXQ-PHC
                lr = chooseLearningRate();
```

```
                  df = chooseDiscountFactor();
                  epsilon = chooseEpsilon();
                  do {
                        System.out.println("Please set delta (between
0 and 1)");
                        delta = StdIn.readDouble();
                  } while (delta < 0 || delta > 1);
                  MA_MaxQ_PHC mphc = new MA_MaxQ_PHC();
                  for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        mphc.calculate(episodes, num, length, epsilon,
df, delta,
                                    filename, delivery, lr);
                        mphc.agents.clear();
                        this.setEpisodeSteps(i,
mphc.getEpisodeSteps());
                  }
                  // for (int i = 0; i < episodes; i++) {
                  for (int i = 0; i < length; i++) {
                        this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                    trials));
                  }
                  p.printAvgResults("MA_MAXQ-PHC Average Results.txt",
this
                              .getAverageSteps(), length);
                  plotGraph("Multi agent MAXQ-PHC Average Steps Per
Episode", num,
                              "Multi Agent MAXQ-PHC");
                  break;
            case 5:// MAXQ-WoLF-PHC
                  MA_MaxQ_WoLF_PHC mwphc = new MA_MaxQ_WoLF_PHC();
                  lr = chooseLearningRate();
                  df = chooseDiscountFactor();
                  epsilon = chooseEpsilon();
                  do {
                        System.out
                                    .println("Please set delta winning
(between 0 and 1)");
                        deltaW = StdIn.readDouble();
                  } while (deltaW < 0 || deltaW > 1);

                  do {
                        System.out.println("Please set delta losing
(between 0 and 1)");
                        deltaL = StdIn.readDouble();
                  } while (deltaL < 0 || deltaL > 1);

                  for (int i = 0; i < trials; i++) {
                        System.out.println("trial=" + i);
                        mwphc.calculate(episodes, num, length,
epsilon, df, deltaW,
                                    deltaL, filename, delivery,lr);
                        mwphc.agents.clear();

                        this.setEpisodeSteps(i,
mwphc.getEpisodeSteps());
                  }
```

```
                    // for (int i = 0; i < episodes; i++) {
                    for (int i = 0; i < length; i++) {
                          this.setAverageSteps(i, this.findAvarage(i,
this.episodeSteps,
                                      trials));
                    }
                    p.printAvgResults("MA_MAXQ-WoLF-PHC Average
Results.txt", this
                                .getAverageSteps(), length);
                    plotGraph("Multi agent MAXQ-WoLF-PHC Average Steps
Per Episode",
                                num, "Multi Agent MAXQ-WoLF-PHC");
                    break;
              default:
                    System.err.println("Invalid number for
choice..Exiting..");
                    System.exit(-1);
              }
        }

        /**
         * @param args
         */
        public static void main(String[] args) {
              // TODO Auto-generated method stub
              // MA_RunTrials r = new MA_RunTrials();
              // r.calculate(1, 100, 10);
        }

}
```

MA_PlotGraph.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

/**
 * klasi ipefthini gia tin dimiourgia ton grafikwn parastasevn gia
multi agent
 *
 * @author Giannis
 *
 */
public class MA_PlotGraph {
```

```
	/**
	 * methodos i opoia kani plot tin grafiki
	 *
	 * @param data
	 * @param title
	 * @param xAxis
	 * @param yAxis
	 */
	public static void createChart(ArrayList<XYSeries> data, String
title,
			String xAxis, String yAxis) {

		XYSeriesCollection dataset = new XYSeriesCollection();
		for (int i = 0; i < data.size(); i++) {
			dataset.addSeries(data.get(i));
		}

		JFreeChart chart = ChartFactory.createXYLineChart(title,
xAxis, yAxis,
				dataset, PlotOrientation.VERTICAL, true, true,
false);
		ChartFrame frame = new ChartFrame(title, chart);
		frame.setLocation(0, 0);
		frame.setVisible(true);
		frame.setSize(500, 400);

	}

	/**
	 * methodos i opoia dimiourga tin lista me tis times gia kathe
	 * grafiki
	 * @param algorithms
	 * @param flag
	 */
	public static void makeGraph(boolean[] algorithms, boolean flag,
int num) {
		ArrayList<XYSeries> steps = new ArrayList<XYSeries>();
		int exploration = 0;
		for (int i = 0; i < algorithms.length; i++) {
			if (algorithms[i]) {
				switch (i) {
				case 0:
					do {
						System.out.println("Choose
exploration for Q-Learning");
						System.out.println("\t0: Botzmann
explotation");
						System.out.println("\t1: e-
greedy");
						System.out.println("\t2: both (if
they are exist)");
						exploration = StdIn.readInt();
					} while (exploration < 0 || exploration
> 2);
					if (exploration == 0) {
						readFromFile(
								"MAQ-Learning Average
Results with boltzmann exploration.txt",
```

```
                                                steps, "MA Q-Learning
- boltzmann exploration",
                                                i, num);
                                    } else if (exploration == 1) {
                                          readFromFile(
                                                "MAQ-Learning Average
Results with e-greedy.txt",
                                                steps, "MA Q-Learning
- e-greedy", i, num);
                                    } else {
                                          readFromFile(
                                                "Q-Learning Average
Results with boltzmann exploration.txt",
                                                steps, "MA Q-Learning
- boltzmann exploration",
                                                i, num);
                                          readFromFile(
                                                "Q-Learning Average
Results with e-greedy.txt",
                                                steps, "MA Q-Learning
- e-greedy", i, num);
                                    }
                                    break;
                              case 1:
                                    readFromFile("MA_PHC Average
Results.txt", steps, "MA PHC",
                                                i, num);
                                    break;
                              case 2:
                                    readFromFile("MA_WoLF-PHC Average
Results.txt", steps,
                                                "MA WoLF-PHC", i, num);
                                    break;
                              case 3:
                                    readFromFile("MA_MAXQ-Q Average
Results.txt", steps,
                                                "MA MAXQ-Q", i, num);
                                    break;
                              case 4:
                                    readFromFile("MA_MAXQ-PHC Average
Results.txt", steps,
                                                "MA MAXQ-PHC", i, num);
                                    break;
                              case 5:
                                    readFromFile("MA_MAXQ-WoLF-PHC Average
Results.txt", steps,
                                                "MA MAXQ-WoLF-PHC", i, num);
                                    break;
                              default:
                                    System.err.println("Invalid
number..Exiting..");
                                    System.exit(-1);
                              }
                        }
                  }
                  ArrayList<XYSeries> tmp = new ArrayList<XYSeries>();
                  if (flag) {// different graphs
                        for (int i = 0; i < steps.size(); i++) {
```

```
                        tmp.add(steps.get(i));
                        MA_PlotGraph.createChart(tmp, "Average Steps
Per Episode",
                                    "Episode", "Average Steps");
                        tmp.clear();
                  }
            } else {
                  MA_PlotGraph.createChart(steps, "Average Steps Per
Episode",
                              "Episode", "Average Steps");
            }
      }

      /**
       * methodos i opoia diavazi tis times gia tis grafikes apo to
       * arxio
       * @param filename
       * @param steps
       * @param str
       * @param algorithm
       */
      public static void readFromFile(String filename,
ArrayList<XYSeries> steps,
                  String str, int algorithm, int num) {
            XYSeries tmp = new XYSeries(str);

            BufferedReader in = null;
            String[] splitline;

            try {
                  in = new BufferedReader(new FileReader(filename));
            } catch (FileNotFoundException e) {
                  e.printStackTrace();
            }
            String line = null;

            // grammes tou gridworld
            try {
                  line = in.readLine();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            int count = 0;
            while (line != null) {
                  splitline = line.split("\t");
                  tmp.add(count * num,
Double.parseDouble(splitline[1]));
                  try {
                        line = in.readLine();
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
                  count++;
            }

            steps.add(tmp);
      }
```

```
	public static void createChart(XYSeries data, String title,
String xAxis,
			String yAxis) {

		XYSeriesCollection dataset = new XYSeriesCollection();
		dataset.addSeries(data);

		JFreeChart chart = ChartFactory.createXYLineChart(title,
xAxis, yAxis,
				dataset, PlotOrientation.VERTICAL, true, true,
false);

		ChartFrame frame = new ChartFrame(title, chart);

		frame.setLocation(0, 0);
		frame.setVisible(true);
		frame.setSize(500, 400);

	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}
}
```

MA_Simulation.java

```
/**
 * klasi i opoia kani tin prosomiosi gia tous multi agent algorithmous
 *
 * @author Giannis
 *
 */
public class MA_Simulation extends Simulation {

	/**
	 * methodos i opoia ektiponi tis grafikes gia tous algorithmous
	 * pou exoun ektelesti
	 *
	 * @param table
	 */
	public void plotGraphs(boolean[] table) {
		System.out.println("Make your choice between the executed
algorithms:");
		System.out
				.println("Write your choice in one line
separated with space..");
		for (int i = 0; i < table.length; i++) {
			if (table[i]) {
				switch (i) {
				case 0:
```

```
                              System.out.println("\t0: Q-Learning");
                              break;
                        case 1:
                              System.out.println("\t1: PHC");
                              break;
                        case 2:
                              System.out.println("\t2: WoLF-PHC");
                              break;
                        case 3:
                              System.out.println("\t3: MAXQ-Q");
                              break;
                        case 4:
                              System.out.println("\t4: MAXQ-PHC");
                              break;
                        case 5:
                              System.out.println("\t5: MAXQ-WoLF-
PHC");
                              break;
                        default:
                              System.err.println("Invalid number for
choice..Exiting..");
                              System.exit(-1);
                        }
                  }
            }
            boolean[] algorithms = new boolean[table.length];

            for (int i = 0; i < table.length; i++)
                  algorithms[i] = false;
            StdIn.readLine();
            String line = StdIn.readLine();
            String[] splitline = line.split(" ");

            for (int i = 0; i < splitline.length; i++)
                  algorithms[Integer.parseInt(splitline[i])] = true;

            System.out.println("Plot in the same graph or in different
graphs?");
            System.out.println("\tPress 0 for the same or 1 for
different");
            boolean flag = StdIn.readBoolean();
            //MA_PlotGraph.makeGraph(algorithms, flag,
MA_RunTrials.num);
            MA_PlotGraph.makeGraph(algorithms, flag, 1000);

      }

      /**
       * methodos stin opoia ginete i epilogi tis epomenis energias
       * apo ton xristi
       * @return
       */
      public int chooseNextMove() {
            int nextMove;
            do {
                  System.out.println("Please choose the next move:");
                  System.out.println("\t0: Choose another Multi Agent
algorithm");
```

```
                  System.out.println("\t1: Plot graphs");
                  System.out
                              .println("\t2: Choose between Single
Agent and Multi Agent Algorithms");
                  System.out.println("\t3: Exit");
                  nextMove = StdIn.readInt();
            } while (nextMove < 0 || nextMove > 3);
            return nextMove;
      }

      /**
       * methodos i opoia kani tin prosomiosi
       */
      public void simulation() {
            int choice = 0;
            int count = 0;
            int nextMove = 0;
            boolean[] table = new boolean[6];// 6 einai o arithmos ton
algorithmon
            for (int i = 0; i < table.length; i++)
                  table[i] = false;
            while (true) { // atermon vrogxos skopima
                  System.out.println("Please choose an algorithm:");
                  do {
                        if (count > 0)
                              System.out
                                          .println("Your choice should
be one of the following:");
                        System.out.println("\t0: Q-Learning");
                        System.out.println("\t1: PHC");
                        System.out.println("\t2: WoLF-PHC");
                        System.out.println("\t3: MAXQ-Q");
                        System.out.println("\t4: MAXQ-PHC");
                        System.out.println("\t5: MAXQ-WoLF-PHC");
                        System.out.println();
                        count++;
                        choice = StdIn.readInt();
                  } while (choice < 0 || choice > 5);

                  table[choice] = true;
                  count = 0;
                  MA_RunTrials r = new MA_RunTrials();
                  r.calculate(choice);

                  nextMove = chooseNextMove();

                  if (nextMove == 1) {
                        do {
                              plotGraphs(table);
                              nextMove = chooseNextMove();
                        } while (nextMove == 1);
                  }
                  if (nextMove == 3) {
                        System.out.println("Exiting..");
                        System.exit(-1);
                  }
                  if (nextMove == 2) {
                        break;
```

```
                }

            }
        }

        public static void main(String[] args) {
            // TODO Auto-generated method stub
            MA_Simulation s = new MA_Simulation();
            s.simulation();
        }

}
```

MA_MaxQ.java

```
import java.util.ArrayList;
import java.util.Random;

/**
 * klasis i opoia ilopoiei to MaxQ Value function decomposition
 *
 * @author Giannis
 *
 */
public class MA_MaxQ extends MA_Graph {

        public static Random r = new Random();
        private int[] episodeSteps;
        private double[] timePerEpisode;
        private double discount_factor;
        private double epsilon;

        private int[] ranges;
        private int[] offsetTable;

        /******methodoi set,get,add gia tis private metavlites ********/

        /**
         * methodos i opoia dini timi sta steps tou episodiou i
         *
         * @param i
         * @param steps
         */
        public void setEpisodeSteps(int i, int steps) {
            this.episodeSteps[i] = steps;
        }

        /**
         * methodos i opoia epistrefi ton pinaka episodeSteps
         *
         * @return
         */
        public int[] getEpisodeSteps() {
            return this.episodeSteps;
```

```
}

/**
 * methodos i opoia epistrefi ta steps tou episodiou i
 *
 * @param i
 * @return
 */
public int getEpisodeSteps(int i) {
	return this.episodeSteps[i];
}

/**
 * methodos i opoia af3ani kata 1 ta steps tou episodiou i
 *
 * @param i
 */
public void addEpisodeSteps(int i) {
	this.episodeSteps[i]++;
}

/**
 * methodos i opoia dini timi sta steps tou episodiou i
 *
 * @param i
 * @param steps
 */
public void setTimePerEpisode(int i, double time) {
	this.timePerEpisode[i] = time;
}

/**
 * methodos i opoia epistrefi ton pinaka timePerEpisode
 *
 * @return
 */
public double[] getTimePerEpisode() {
	return this.timePerEpisode;

}
/**
 * methodos i opoia epistrefi to xrono pou xriastike to
 * episodiou i
 * @param i
 * @return
 */
public double getTimePerEpisode(int i) {
	return this.timePerEpisode[i];
}
/**
 * methodos i opoia kathorizei to discount factor
 *
 * @param num
 */
public void setDiscountFactor(double num) {
	this.discount_factor = num;
}
```

```
	/**
	 * methodos i opoia epistrefi to discount factor
	 *
	 * @return
	 */
	public double getDiscountFactor() {
		return this.discount_factor;
	}

	/**
	 * methodos i opoia kathorizei to epsilon
	 *
	 * @param num
	 */
	public void setEpsilon(double num) {
		this.epsilon = num;
	}

	/**
	 * methodos i opoia epistrefi to epsilon
	 *
	 * @return
	 */
	public double getEpsilon() {
		return this.epsilon;
	}

	/**
	 * methodos i opoia kathorizi ta ranges
	 */
	public void setRanges() {
		ranges = new int[MA_Gridworld.numOfAgents +
MA_Gridworld.numOfPassengers
				+ MA_Gridworld.numOfPassengers];

		for (int i = 0; i < MA_Gridworld.numOfAgents; i++)
			this.ranges[i] = MA_Gridworld.size_x *
MA_Gridworld.size_y;
		for (int i = 0; i < MA_Gridworld.numOfPassengers; i++)
			this.ranges[i + MA_Gridworld.numOfAgents] =
MA_Gridworld.numOfStartPosition + 1;
		for (int i = 0; i < MA_Gridworld.numOfPassengers; i++)
			this.ranges[i + MA_Gridworld.numOfAgents +
MA_Gridworld.numOfPassengers] = MA_Gridworld.numOfGoals;
	}

	/**
	 * @return the ranges
	 */
	public int[] getRanges() {
		return ranges;
	}
```

```
	/**
	 * @param offsetTable
	 *            the offsetTable to set
	 */
	public void setOffsetTable(int[] offsetTable) {
		this.offsetTable = offsetTable;
	}

	/**
	 * @return the offsetTable
	 */
	public int[] getOffsetTable() {
		return offsetTable;
	}

	/*************************************************************/

	/**
	 * methodos i opoia arxikopoiei ton pinaka offsetTable
	 */
	public void initOffsetTable() {
		offsetTable = new int[MA_Gridworld.numOfAgents +
MA_Gridworld.numOfPassengers
				+ MA_Gridworld.numOfPassengers];

		int product = 1;
		for (int i = offsetTable.length - 1; i >= 0; --i) {
			offsetTable[i] = product;
			product *= ranges[i];
		}
	}

	/**
	 * methodos i opoia ipologizi to index apo ena vector (vriski
tin katastasi
	 * pou imaste sto perivallon,tin grammi tou Q table)
	 *
	 * @param s
	 * @return
	 */
	public int findIndex(MA_State_Variables s) {
		int offset = 0;
		for (int i = 0; i < s.taxi.size(); ++i) {
			offset += s.taxi.get(i) * offsetTable[i];
		}
		for (int i = 0; i < s.passenger.size(); ++i) {
			offset += s.passenger.get(i) * offsetTable[i +
s.taxi.size()];
		}
		for (int i = 0; i < s.destination.size(); ++i) {
			offset += s.destination.get(i)
					* offsetTable[i + s.taxi.size() +
s.passenger.size()];
		}
		return offset;

	}
```

```
	/**
	 * methodos i opoia dimiourgei ton pinaka episodeSteps
	 *
	 * @param max_episode
	 */
	public void makeNewEpisodeSteps(int max_episode) {
		this.episodeSteps = new int[max_episode];
		// initialize episodeSteps
		for (int i = 0; i < max_episode; i++) {
			this.setEpisodeSteps(i, 0);
		}
	}

	/**
	 * methodos i opoia dimiourgei ton pinaka episodeSteps
	 *
	 * @param max_episode
	 */
	public void makeNewTimePerEpisode(int max_episode) {
		this.timePerEpisode = new double[max_episode];
		// initialize episodeSteps
		for (int i = 0; i < max_episode; i++) {
			this.setTimePerEpisode(i, 0.0);
		}
	}

	/**
	 * methodos i opoia arxikopoiei ta actions se ola ta primitive
MaxNode
	 */
	public void initActionsToPrimitive(ArrayList<MaxQ_Agent> agents)
{
		for (int i = 0; i < agents.size(); i++) {
			((MaxNode) agents.get(i).g.graph.get(this.Pickup))
					.setAction(MA_Gridworld.pickup);
			((MaxNode) agents.get(i).g.graph.get(this.Putdown))
					.setAction(MA_Gridworld.putdown);
			((MaxNode) agents.get(i).g.graph.get(this.North))
					.setAction(MA_Gridworld.north);
			((MaxNode) agents.get(i).g.graph.get(this.South))
					.setAction(MA_Gridworld.south);
			((MaxNode) agents.get(i).g.graph.get(this.East))
					.setAction(MA_Gridworld.east);
			((MaxNode) agents.get(i).g.graph.get(this.West))
					.setAction(MA_Gridworld.west);
		}
	}

	/**
	 * methodos i opoia vriski tin timi (V(i,s)) gia ena MaxNode pou
ine
	 * composite (not primitive) - Greedy Execution of the MAXQ
Graph
	 * (evaluateMaxNode(i,s))
	 *
	 * @param maxnode
	 * @param index
	 *            ( dixni tin katastasi (state) pou vriskomaste)
```

```
	 * @return
	 */
	public NodeValue evaluateMaxNode(MaxQ_Agent agent, int node, int
index) {

		ArrayList<NodeValue> nodeValue = new
ArrayList<NodeValue>();

		int maxpos = 0;

		if (agent.g.graph.get(node) instanceof Qnode) {
			node = ((Qnode)
agent.g.graph.get(node)).children.get(0);
		}

		if (((MaxNode) agent.g.graph.get(node)).isPrimitive()) {
			NodeValue tmp = new NodeValue(((MaxNode)
agent.g.graph.get(node))
					.getValueFunctForPrimitive(index),
node);
			return tmp;
		} else {
			for (int i = 0; i <
agent.g.graph.get(node).children.size(); i++) {
				NodeValue tmp = this.evaluateMaxNode(agent,
agent.g.graph
						.get(node).children.get(i),
index);
				nodeValue.add(tmp);
			}
			maxpos = this.findMax(nodeValue);
			nodeValue.get(maxpos).value += ((Qnode)
agent.g.graph
					.get(((MaxNode)
agent.g.graph.get(node)).children

	.get(maxpos))).getCvalue(index);

			return nodeValue.get(maxpos);
		}

	}

	/**
	 * methodos i opoia vriski pio node apo ena arraylist exi ti pio
megali timi
	 * (Value) V(j,s)) - xrisimopoieitai apo tin evaluateMaxNode
	 *
	 * @param nodeValue
	 * @param node
	 * @param index
	 * @return
	 */
	public int findMax(ArrayList<NodeValue> nodeValue) {
		double max = 0;
		int maxpos = 0;
		double temp = 0;
		if (nodeValue.isEmpty()) {
```

```
                  System.err.println("Error in evaluating Max
Node\nExiting..");
                  System.exit(-1);
            }
            max = nodeValue.get(0).value;
            for (int i = 1; i < nodeValue.size(); i++) {
                  temp = nodeValue.get(i).value;
                  if (temp > max) {
                        max = temp;
                        maxpos = i;
                  }
            }
            return maxpos;
      }

      /**
       * methodos i opoia elegxi an imaste se terminal state enos
subtask
       * (MaxNode)
       *
       * @param node
       * @param s
       * @return
       */
      public boolean isTerminal(MaxQ_Agent agent, int node,
MA_State_Variables s) {
            switch (node) {
            case 0:// MaxRoot goal state (skopima epistrefi mono
false)
                  //if(agent.isTransferPrevious()==true &&
agent.isTransfer()==false)
                  //return true;
                  return false;
            case 1:// MaxGet
                  if(agent.getPassenger()!=-1)
                        return true;
                  return false;
            case 2:// MaxPut
                  // goal state
                  if(agent.getPassenger()==-1)
                        return true;
                  if
(((MA_Gridworld.getPasLoc(s.passenger.get(agent.getPassenger())) ==
MA_Gridworld

      .getDestLoc(s.destination.get(agent.getPassenger()))) && (agent
                              .getAgentPos() ==
MA_Gridworld.getDestLoc(s.destination
                              .get(agent.getPassenger()))))
                              || MA_Gridworld.getPasLoc(s.passenger
                                          .get(agent.getPassenger()))
!= -1)
                        // terminal state (to OR) (simeni oti o
passenger dn ine mesa
                        // sto
                        // taxi ara prepi na kamw get prwta)
                        return true;
                  return false;
```

```
            case 5:// MaxNavigate(t) goal state
                  if (((MaxNode) agent.g.graph.get(node)).getDest() ==
agent
                              .getAgentPos())
                        return true;
                  return false;
            default:
                  return false;
            }
      }


      /**
       * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
       * sigkekrimeni katastasi (state) simfona me tin policy e-greedy
       *
       * @param node
       * @param index
       * @return
       */
      public int chooseAction(MaxQ_Agent agent,int node, int index) {
            if (r.nextDouble() < this.epsilon)
                  return
(r.nextInt(agent.g.graph.get(node).children.size()));
            return this.findMaxQvalue(agent,node, index);
      }

      /**
       * methodos i opoia epistrefi tin timi Value gia mia katastasi
       * enos node analoga me to an ine primitive i composite node
       *
       * @param maxNode
       * @param index
       * @return
       */
      public double findValue(MaxQ_Agent agent, int maxNode, int
index) {
            if (!(agent.g.graph.get(maxNode) instanceof Qnode))
                  if (((MaxNode)
agent.g.graph.get(maxNode)).isPrimitive()) {

                        return (((MaxNode) agent.g.graph.get(maxNode))

      .getValueFunctForPrimitive(index));
                  }
            NodeValue nValue = this.evaluateMaxNode(agent, maxNode,
index);

            return nValue.value;

      }
```

```
	/**
	 * methodos i opoia vriski pia thesi tou pinaka exi ti
	 * megaliteri timi
	 * @param table
	 * @return
	 */
	public int findMax(double[] table) {
		int maxPos = 0;
		double max = table[0];
		for (int i = 1; i < table.length; i++) {
			if (table[i] > max) {
				max = table[i];
				maxPos = i;
			}
		}
		return maxPos;
	}

	/**
	 * methodos i opoia vriski pios Qnode exi ti megaliteri Expected
Discounted
	 * Cumulative Reward ( C(i,s,a) )
	 *
	 * @param node
	 * @param index
	 * @return
	 */
	public int findMax(int node, int index) {
		double maxValue = ((Qnode)
this.graph.get(this.graph.get(node).children
				.get(0))).getCvalue(index);
		int maxPos = 0;
		double temp = 0;
		for (int i = 1; i < this.graph.get(node).children.size();
i++) {
			temp = ((Qnode) this.graph

	.get(this.graph.get(node).children.get(i)))
					.getCvalue(index);
			if (temp > maxValue) {
				maxValue = temp;
				maxPos = i;
			}
		}
		return maxPos;
	}

	/**
	 * methodos i opoia vriski pios Qnode apo ta children enos
Maxnode exi to
	 * max Q value
	 *
	 * @param maxNode
	 * @param index
	 * @return
	 */
	public int findMaxQvalue(MaxQ_Agent agent, int maxNode, int
index) {
```

```
            double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
            table = this.findQvalues(agent, maxNode, index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia vriski ta Q values gia kathe children enos
       * MaxNode
       * @param node
       * @param index
       * @return
       */
      public double[] findQvalues(MaxQ_Agent agent, int maxNode, int
index) {
            double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
            NodeValue nValue;
            for (int i = 0; i <
agent.g.graph.get(maxNode).children.size(); i++) {
                  nValue = this.evaluateMaxNode(agent,

      agent.g.graph.get(maxNode).children.get(i
, index);
                  table[i] = nValue.value;
            }



            for (int i = 0; i < table.length; i++)
                  table[i] += ((Qnode) agent.g.graph

      .get(agent.g.graph.get(maxNode).children.get(i)))
                              .getCvalue(index);

            return table;
      }

      /**
       * methodos i opoia vriski pios Qnode exi ti mikroteri Expected
       * Discounted Cumulative Reward ( C(i,s,a) )
       *
       * @param node
       * @param index
       * @return
       */
      public int findMin(int node, int index) {
            double minValue = ((Qnode)
this.graph.get(this.graph.get(node).children
                        .get(0))).getCvalue(index);
            int minPos = 0;
            double temp = 0;
            for (int i = 1; i < this.graph.get(node).children.size();
i++) {
                  temp = ((Qnode) this.graph

      .get(this.graph.get(node).children.get(i)))
                              .getCvalue(index);
```

```
                  if (temp < minValue) {
                        minValue = temp;
                        minPos = i;
                  }
            }
            return minPos;
      }

      /**
       * methodos i opoia arxikopoiei ta learning rates ton Max nodes
       */
      public void initLearningRate(ArrayList<MaxQ_Agent> agents,double
lr) {
            // set learning rate to Max nodes
            for (int i = 0; i < agents.size(); i++) {
                  ((MaxNode) agents.get(i).g.graph.get(this.MaxRoot))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.MaxGet))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.MaxPut))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.Pickup))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.Putdown))
                              .setLearningRate(lr);
                  ((MaxNode)
agents.get(i).g.graph.get(this.MaxNavigate))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.North))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.East))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.South))
                              .setLearningRate(lr);
                  ((MaxNode) agents.get(i).g.graph.get(this.West))
                              .setLearningRate(lr);
            }
      }

      /**
       * methodos i opoia arxikopoiei tis thermokrasies stous MaxNodes
       */
      public void initTemperature(ArrayList<MaxQ_Agent> agents, int t)
{
            for (int i = 0; i < agents.size(); i++) {
                  ((MaxNode) agents.get(i).g.graph.get(this.MaxRoot))
                              .setTemperature(t);
                  ((MaxNode) agents.get(i).g.graph.get(this.MaxGet))
                              .setTemperature(t);
                  ((MaxNode) agents.get(i).g.graph.get(this.MaxPut))
                              .setTemperature(t);
                  ((MaxNode)
agents.get(i).g.graph.get(this.MaxNavigate))
                              .setTemperature(t);
            }
      }
```

```
	/**
	 * methodos i opoia arxikopoiei ta cooling rates stous MaxNodes
	 */
	public void initCoolingRates() {
		((MaxNode)
this.graph.get(this.MaxRoot)).setCoolingRate(0.9996);
		((MaxNode)
this.graph.get(this.MaxGet)).setCoolingRate(0.9939);
		((MaxNode)
this.graph.get(this.MaxPut)).setCoolingRate(0.9996);
		((MaxNode)
this.graph.get(this.MaxNavigate)).setCoolingRate(0.9879);
	}
	/**
	 * methodos i opoia allazi ta learning rate ton Max Nodes
	 *
	 * @param episode
	 * @param max_episode
	 */
	public void changeLearningRate(int episode, int max_episode) {
		((MaxNode)
this.graph.get(this.MaxRoot)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.MaxGet)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.MaxPut)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.Pickup)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.Putdown)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.MaxNavigate)).changeLearningRate(
				episode, max_episode);
		((MaxNode)
this.graph.get(this.North)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.East)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.South)).changeLearningRate(episode,
				max_episode);
		((MaxNode)
this.graph.get(this.West)).changeLearningRate(episode,
				max_episode);
	}
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
	}
}
```

Agent.java

```
/**
 * afairetiki klasi pou antiprosopevi tous agent
 *
 * @author Giannis
 *
 */
public abstract class Agent {
	private double[][] Q;// lookup table
	private int agentNum;// i thesi tou agent mesa sti lista
	private int agentPos; // pou vriskete o agent sto gridworld
	// pou vriskotan o agent sto gridworld stin
	// prohgoumeni katastasi
	private int agentPreviousPos;
	private int passenger; // pios passenger metaferete(i thesi tou
sti lista)
	private int action;
	private int prevAction;
	private int reward;
	private boolean transfer; // mas leei an metaferete kapios
passenger
	// mas leei an stin proigoumeni katastasi
	// metaferotan kapios passenger
	private boolean transferPrevious;
	// mas leei an o agent mas exi sigkrousti me
	// kapion allo i oxi
	private boolean collision;

	/**
	 * @param q
	 *            the q to set
	 */
	abstract void setQ(double[][] q);

	/**
	 * @return the q
	 */
	abstract double[][] getQ();
	/**
	 * @param agentNum
	 *            the agentNum to set
	 */
	abstract void setAgentNum(int agentNum);
	/**
	 * @return the agentNum
	 */
	abstract int getAgentNum();
	/**
	 * @param agentPos
	 *            the agentPos to set
	 */
	public void setAgentPos(int agentPos) {
		this.agentPos = agentPos;
	}
```

```
/**
 * @return the agentPos
 */
abstract int getAgentPos();

/**
 * @param agentPreviousPos
 *            the agentPreviousPos to set
 */
abstract void setAgentPreviousPos(int agentPreviousPos);

/**
 * @return the agentPreviousPos
 */
abstract int getAgentPreviousPos();

/**
 * @param passenger
 *            the passenger to set
 */
abstract void setPassenger(int passenger);

/**
 * @return the passenger
 */
abstract int getPassenger();

/**
 * @param action
 *            the action to set
 */
abstract void setAction(int action);

/**
 * @return the action
 */
abstract int getAction();

/**
 * @param prevAction
 *            the prevAction to set
 */
abstract void setPrevAction(int prevAction);
/**
 * @return the prevAction
 */
abstract int getPrevAction();
/**
 * @param reward
 *            the reward to set
 */
abstract void setReward(int reward);

/**
 * @return the reward
 */
abstract int getReward();
```

```
        /**
         * @param transfer
         *            the transfer to set
         */
        abstract void setTransfer(boolean transfer);

        /**
         * @return the transfer
         */
        abstract boolean isTransfer();

        /**
         * @param transferPrevious
         *            the transferPrevious to set
         */
        abstract void setTransferPrevious(boolean transferPrevious);

        /**
         * @return the transferPrevious
         */
        abstract boolean isTransferPrevious();

        /**
         * @param collision
         *            the collision to set
         */
        abstract void setCollision(boolean collision);

        /**
         * @return the collision
         */
        abstract boolean isCollision();

        /**
         * methodos i opoia dimiourga ton pinaka Q
         */
        abstract void makeQ();

        /**
         * methodos i opia dini timi stin sigkekrimeni thesi tou Q
         *
         * @param i
         * @param j
         * @param value
         */
        abstract void setQValue(int i, int j, double value);

        /**
         * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
tou Q
         *
         * @param i
         * @param j
         * @return
         */
        abstract double getQValue(int i, int j);
```

```
	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou Q to
	 * neo value
	 * @param i
	 * @param j
	 * @param value
	 */
	abstract void addQValue(int i, int j, double value);

	/**
	 * methodos i opoia arxikopoiei to Q
	 */
	abstract void initQ(double num);
}
```

Q_Agent.java

```
/**
 * klasi i opoia antiprosopevi ton agent gia tin methodo Q-Learning
 *
 * @author Giannis
 *
 */
public class Q_Agent extends Agent {
	// constructor
	Q_Agent(int agentNum, int agentPos, double qValue) {
		this.agentNum = agentNum;
		this.agentPos = agentPos;
		this.action = -1;
		this.prevAction = -1;
		this.reward = -1;
		this.makeQ();
		this.initQ(qValue);
		this.agentPreviousPos = -1; // (-1 = dn iparxi proigoumeni
katastasi)
		this.passenger = -1; // -1 =dn metaferete kapios passenger
		this.transfer = false;
		this.transferPrevious = false;
		this.collision = false;
	}

	private double[][] Q;// lookup table
	private int agentNum;// i thesi tou agent mesa sti lista
	private int agentPos; // pou vriskete o agent sto gridworld
	// pou vriskotan o agent sto gridworld stin
	// prohgoumeni katastasi
	private int agentPreviousPos;
	private int passenger; // pios passenger metaferete(i thesi tou
sti lista)
	private int action;
	private int prevAction;
	private int reward;
	private boolean transfer; // mas leei an metaferete kapios
passenger
	// mas leei an stin proigoumeni katastasi
```

```
// metaferotan kapios passenger
private boolean transferPrevious;
// mas leei an o agent mas exi sigkrousti me
// kapion allo i oxi
private boolean collision;

/**
 * @param q
 *            the q to set
 */
public void setQ(double[][] q) {
      Q = q;
}

/**
 * @return the q
 */
public double[][] getQ() {
      return Q;
}

/**
 * @param agentNum
 *            the agentNum to set
 */
public void setAgentNum(int agentNum) {
      this.agentNum = agentNum;
}

/**
 * @return the agentNum
 */
public int getAgentNum() {
      return agentNum;
}

/**
 * @param agentPos
 *            the agentPos to set
 */
public void setAgentPos(int agentPos) {
      this.agentPos = agentPos;
}

/**
 * @return the agentPos
 */
public int getAgentPos() {
      return agentPos;
}

/**
 * @param agentPreviousPos
 *            the agentPreviousPos to set
 */
public void setAgentPreviousPos(int agentPreviousPos) {
      this.agentPreviousPos = agentPreviousPos;
}
```

```
/**
 * @return the agentPreviousPos
 */
public int getAgentPreviousPos() {
	return agentPreviousPos;
}

/**
 * @param passenger
 *            the passenger to set
 */
public void setPassenger(int passenger) {
	this.passenger = passenger;
}

/**
 * @return the passenger
 */
public int getPassenger() {
	return passenger;
}

/**
 * @param action
 *            the action to set
 */
public void setAction(int action) {
	this.action = action;
}

/**
 * @return the action
 */
public int getAction() {
	return action;
}

/**
 * @param prevAction
 *            the prevAction to set
 */
public void setPrevAction(int prevAction) {
	this.prevAction = prevAction;
}

/**
 * @return the prevAction
 */
public int getPrevAction() {
	return prevAction;
}

/**
 * @param reward
 *            the reward to set
 */
public void setReward(int reward) {
```

```
		this.reward = reward;
	}

	/**
	 * @return the reward
	 */
	public int getReward() {
		return reward;
	}

	/**
	 * @param transfer
	 *            the transfer to set
	 */
	public void setTransfer(boolean transfer) {
		this.transfer = transfer;
	}

	/**
	 * @return the transfer
	 */
	public boolean isTransfer() {
		return transfer;
	}

	/**
	 * @param transferPrevious
	 *            the transferPrevious to set
	 */
	public void setTransferPrevious(boolean transferPrevious) {
		this.transferPrevious = transferPrevious;
	}

	/**
	 * @return the transferPrevious
	 */
	public boolean isTransferPrevious() {
		return transferPrevious;
	}
	/**
	 * @param collision
	 *            the collision to set
	 */
	public void setCollision(boolean collision) {
		this.collision = collision;
	}
	/**
	 * @return the collision
	 */
	public boolean isCollision() {
		return collision;
	}
	/**
	 * methodos i opoia dimiourga ton pinaka Q
	 */
	public void makeQ() {
		Q = new double[MA_Gridworld.states][MA_Gridworld.actions];
	}
```

```
	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou Q
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setQValue(int i, int j, double value) {
		this.Q[i][j] = value;
	}

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou Q
	 * @param i
	 * @param j
	 * @return
	 */
	public double getQValue(int i, int j) {
		return this.Q[i][j];
	}

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou Q to
	 * neo value
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addQValue(int i, int j, double value) {
		this.Q[i][j] += value;
	}

	/**
	 * methodos i opoia arxikopoiei to Q
	 */
	public void initQ(double num) {
		for (int i = 0; i < MA_Gridworld.states; i++) {
			for (int j = 0; j < MA_Gridworld.actions; j++) {
				this.setQValue(i, j, num);
			}
		}
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}
}
```

PHC_Agent.java

```
/**
 * klasi i opoia ntiprosopevi ton agent gia tin methodo PHC
 *
 * @author Giannis
 *
 */
public class PHC_Agent extends Q_Agent {
	// constructor
	PHC_Agent(int agentNum, int agentPos, double qValue, double lr)
{
		super(agentNum, agentPos, qValue);
		this.setPolicyValue(new
double[MA_Gridworld.states][MA_Gridworld.actions]);
		this.initPolicyValue();
		this.actionCounter = new
int[MA_Gridworld.states][MA_Gridworld.actions];
		this.setLr(new
double[MA_Gridworld.states][MA_Gridworld.actions]);
		lr0 = lr;
		this.initLr(lr);

	}

	protected double[][] policyValue;// gia ton PHC kai WoLF-PHC

	protected int[][] actionCounter;
	protected double[][] lr;// learning rates for state-action pairs
	protected double lr0;

	/**
	 * @param policyValue
	 *            the policyValue to set
	 */
	public void setPolicyValue(double[][] policyValue) {
		this.policyValue = policyValue;
	}

	/**
	 * @return the policyValue
	 */
	public double[][] getPolicyValue() {
		return policyValue;
	}
	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou
	 * policyValue
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setPolicyValue(int i, int j, double value) {
		this.policyValue[i][j] = value;
	}
```

```
	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou policyValue
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	public double getPolicyValue(int i, int j) {
		return this.policyValue[i][j];
	}

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou
	 * policyValue to neo value
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addPolicyValue(int i, int j, double value) {
		this.policyValue[i][j] += value;
	}

	/**
	 * methodos i opoia arxikopoiei to Q
	 */
	public void initQ(double num) {
		for (int i = 0; i < MA_Gridworld.states; i++) {
			for (int j = 0; j < MA_Gridworld.actions; j++) {
				this.setQValue(i, j, num);
			}
		}
	}

	/**
	 * methodos i opoia arxikopoiei to policyValue
	 *
	 * @param actions
	 */
	public void initPolicyValue() {
		for (int i = 0; i < MA_Gridworld.states; i++) {
			for (int j = 0; j < MA_Gridworld.actions; j++) {
				this.setPolicyValue(i, j, ((double) 1 /
MA_Gridworld.actions));
			}
		}
	}

	/**
	 * @param actionCounter
	 *            the actionCounter to set
	 */
	public void setActionCounter(int[][] actionCounter) {
		this.actionCounter = actionCounter;
	}
```

```java
/**
 * methodos i opoia anatheti mia timi se mia thesi tou pinaka
 * actionCounter
 * @param i
 * @param j
 * @param count
 */
public void setActionCounter(int i, int j, int count) {
	this.actionCounter[i][j] = count;
}

/**
 * methodos i opoia prostheti 1 se mia thesi tou actionCounter
 *
 * @param i
 * @param j
 */
public void addActionCounter(int i, int j) {
	this.actionCounter[i][j]++;
}

/**
 * @return the actionCounter
 */
public int[][] getActionCounter() {
	return actionCounter;
}

/**
 * methodos i opoia epistrefi tin timi mias thesis tou pinaka
 * actionCounter
 * @param i
 * @param j
 * @return
 */
public int getActionCounter(int i, int j) {
	return actionCounter[i][j];
}

/**
 * methodos i opoia arxikopoiei ta learning rates
 *
 * @param value
 */
public void initLr(double value) {
	for (int i = 0; i < MA_Gridworld.states; i++) {
		for (int j = 0; j < MA_Gridworld.actions; j++) {
			this.lr[i][j] = value;
		}
	}
}

/**
 * @param lr
 *            the lr to set
 */
public void setLr(double[][] lr) {
	this.lr = lr;
```

```
	}

	/**
	 * methodos i opoia dini mia timi se mia sigkekrimeni thesi tou
	 * pinaka lr
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setLr(int state, int action, int value) {
		this.lr[state][action] = value;
	}

	/**
	 * methodos i opoia epistrefi to learning rate gia sigkekrimeno
	 * state-action pair
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	public double getLr(int state, int action) {
		return lr[state][action];
	}

	/**
	 * @return the lr
	 */
	public double[][] getLr() {
		return lr;
	}

	/**
	 * methodos i opoia miwni to learning rate se ena state-action
	 * pair
	 * @param state
	 * @param action
	 * @param episodes
	 */
	public void changeLr(int state, int action, int episodes) {
		this.lr[state][action] = this.lr0
				* Math
						.exp(-((double)
getActionCounter(state, action) / episodes));
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}
}
```

WP_Agent.java

```
/**
 * klasi i opoia ntiprosopevi ton agent gia tin methodo WoLF-PHC
 *
 * @author Giannis
 *
 */
public class WP_Agent extends PHC_Agent {

	WP_Agent(int agentNum, int agentPos, double value, double lr) {
		super(agentNum, agentPos, value, lr);
		// TODO Auto-generated constructor stub
		this.avgPolicyValue = new
double[MA_Gridworld.states][MA_Gridworld.actions];
		this.C = new int[MA_Gridworld.states];
		this.initAvgPolicyValue();
		this.initC(0);
	}
	private double[][] avgPolicyValue; // gia ton WoLF-PHC average
policy value
	private int[] C;// gia ton WoLF-PHC
	/****************** gia ton WoLF-PHC ************************/
	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou
	 * avgPolicyValue
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setAvgPolicyValue(int i, int j, double value) {
		this.avgPolicyValue[i][j] = value;
	}
	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou avgPolicyValue
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	public double getAvgPolicyValue(int i, int j) {
		return this.avgPolicyValue[i][j];
	}
	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou
	 * avgPolicyValue to neo value
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addAvgPolicyValue(int i, int j, double value) {
		this.avgPolicyValue[i][j] += value;
	}
```

```
	/**
	 * methodos i opoia dini timi se mia thesi tou pinaka C
	 *
	 * @param state
	 * @param num
	 */
	public void setC(int state, int num) {
		C[state] = num;
	}

	/**
	 * methodos i opoia epistrefi mia timi tou pinaka C
	 *
	 * @return
	 */
	public int getC(int state) {
		return C[state];
	}
	/**
	 * methodos i opoia prostheti 1 se mia thesi tou pinaka C
	 *
	 * @param state
	 */
	public void addOneToC(int state) {
		C[state]++;
	}
	/**
	 * methodos i opoia arxikopoiei to avgPolicyValue
	 *
	 * @param actions
	 */
	public void initAvgPolicyValue() {
		for (int i = 0; i < MA_Gridworld.states; i++) {
			for (int j = 0; j < MA_Gridworld.actions; j++) {
				this.setAvgPolicyValue(i, j, ((double) 1 /
MA_Gridworld.actions));
			}
		}
	}
	/**
	 * methodos i opoia arxikopoiei to C
	 *
	 * @param actions
	 */
	public void initC(int num) {
		for (int i = 0; i < 500; i++) {
			this.setC(i, num);
		}
	}
}
```

MaxQ_Agent.java

```
import java.util.ArrayList;
import java.util.Stack;

public class MaxQ_Agent {
	// constructor
	MaxQ_Agent(int agentNum, int agentPos, int num) {
		this.agentNum = agentNum;
		this.agentPos = agentPos;
		this.action=-1;
		this.prevAction=-1;
		this.reward=-1;
		this.agentPreviousPos = -1; // (-1 = dn iparxi proigoumeni
katastasi)
		this.passenger = -1; // -1 =dn metaferete kapios passenger
		this.transfer = false;
		this.transferPrevious = false;
		this.collision = false;
		stack.push(g.MaxRoot);
		g.makeGraph(MA_Gridworld.states,num);
	}
	MA_Graph g=new MA_Graph();
	Stack<ArrayList<MA_State_Variables>> seqStack = new
Stack<ArrayList<MA_State_Variables>>();
	Stack<Integer> stack = new Stack<Integer>();
	private int agentNum;// i thesi tou agent mesa sti lista
	private int agentPos; // pou vriskete o agent sto gridworld
	// pou vriskotan o agent sto gridworld stin
	// prohgoumeni katastasi
	private int agentPreviousPos;
	private int passenger; // pios passenger metaferete(i thesi tou
sti lista)
	private int action;
	private int prevAction;
	private int reward;
	private boolean transfer; // mas leei an metaferete kapios
passenger
	// mas leei an stin proigoumeni katastasi
	// metaferotan kapios passenger
	private boolean transferPrevious;
	// mas leei an o agent mas exi sigkrousti me
	// kapion allo i oxi
	private boolean collision;
	private int greedyAction;
	public int maxNode;

	/**
	 * @param agentNum
	 *            the agentNum to set
	 */
	public void setAgentNum(int agentNum) {
		this.agentNum = agentNum;
	}
```

```
/**
 * @return the agentNum
 */
public int getAgentNum() {
	return agentNum;
}

/**
 * @param agentPos
 *            the agentPos to set
 */
public void setAgentPos(int agentPos) {
	this.agentPos = agentPos;
}

/**
 * @return the agentPos
 */
public int getAgentPos() {
	return agentPos;
}

/**
 * @param agentPreviousPos
 *            the agentPreviousPos to set
 */
public void setAgentPreviousPos(int agentPreviousPos) {
	this.agentPreviousPos = agentPreviousPos;
}

/**
 * @return the agentPreviousPos
 */
public int getAgentPreviousPos() {
	return agentPreviousPos;
}

/**
 * @param passenger
 *            the passenger to set
 */
public void setPassenger(int passenger) {
	this.passenger = passenger;
}

/**
 * @return the passenger
 */
public int getPassenger() {
	return passenger;
}

/**
 * @param action
 *            the action to set
 */
public void setAction(int action) {
	this.action = action;
```

```
}

/**
 * @return the action
 */
public int getAction() {
	return action;
}

/**
 * @param prevAction
 *            the prevAction to set
 */
public void setPrevAction(int prevAction) {
	this.prevAction = prevAction;
}

/**
 * @return the prevAction
 */
public int getPrevAction() {
	return prevAction;
}

/**
 * @param reward
 *            the reward to set
 */
public void setReward(int reward) {
	this.reward = reward;
}

/**
 * @return the reward
 */
public int getReward() {
	return reward;
}

/**
 * @param transfer
 *            the transfer to set
 */
public void setTransfer(boolean transfer) {
	this.transfer = transfer;
}

/**
 * @return the transfer
 */
public boolean isTransfer() {
	return transfer;
}

/**
 * @param transferPrevious
 *            the transferPrevious to set
 */
```

```
	public void setTransferPrevious(boolean transferPrevious) {
		this.transferPrevious = transferPrevious;
	}

	/**
	 * @return the transferPrevious
	 */
	public boolean isTransferPrevious() {
		return transferPrevious;
	}

	/**
	 * @param collision
	 *            the collision to set
	 */
	public void setCollision(boolean collision) {
		this.collision = collision;
	}

	/**
	 * @return the collision
	 */
	public boolean isCollision() {
		return collision;
	}

	/**
	 * methodos i opoia kathorizi to greedy action
	 *
	 * @param greedyAction
	 */
	public void setGreedyAction(int greedyAction) {
		this.greedyAction = greedyAction;
	}

	/**
	 * methodos i opoia epistrefi to greedy action
	 *
	 * @return
	 */
	public int getGreedyAction() {
		return greedyAction;
	}


	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}
}
```

Q_Controller.java

```
import java.util.ArrayList;

/**
 * klasi i opoia einai ipefthini gia ton elegxo twn Q_Agent
 *
 * @author Giannis
 *
 */
public class Q_Controller {

	/**
	 * methodos i opoia ine ipefthini gia tis kinisis twn agent kai
	 * ta rewards pou pairnoun
	 *
	 * @param s
	 * @param tempState
	 * @param agents
	 * @param passDelivered
	 * @param index
	 * @param exploration
	 * @param episodeSteps
	 */
	public void agentController(MA_Taxi_Problem tp,
MA_State_Variables s,
			MA_State_Variables tempState, ArrayList<Q_Agent>
agents,
			ArrayList<Integer> passDelivered, int index, boolean
exploration,
			int episodeSteps) {

		ArrayList<Integer> collisions = new ArrayList<Integer>();
		// choose action
		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {

	agents.get(i).setPrevAction(agents.get(i).getAction());
			if (exploration)// e-greedy exploration
				agents.get(i).setAction(
						tp.chooseAction(index,
agents.get(i).getQ()));
			else
				// boltzmann exploration
				agents.get(i).setAction(

	tp.chooseActionWithBoltzmannExploration(index, agents
							.get(i).getQ()));
		}
		// change temperature for boltzmmann exploration
		if (!(exploration))
			tp.changeTemperature(episodeSteps);

		// take action
		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
			// if (!(agents.get(i).isCollision())) {
```

```
                  takeAction(s, tempState, agents.get(i).getAction(),
agents.get(i),
                              passDelivered);
                  // }
            }
            // collision check
            //boolean flag;
            //do {
            //    System.out.println("apoel");
            //    flag = false;
                  for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                        if (collisionCheck(tempState, i,
agents.get(i), collisions)) {
                              //flag = true;
                              for (int j = 0; j < collisions.size();
j++) {

      agents.get(collisions.get(j)).setCollision(true);
                              }
                              removeCollisions(tempState, agents,
collisions);
                              // diagrafi collision apo ton agent
                        //    for (int j = 0; j <
MA_Gridworld.numOfAgents; j++) {
                        //          agents.get(j).setCollision(false);
                        //    }
                        }
                        collisions.clear();
                  }
            //} while (flag);

            // get rewards
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  agents.get(i).setReward(
                              findReward(s, tempState,
agents.get(i).getAction(), i,
                                          agents.get(i)));
            }
            // diagrafi collision apo ton agent
             for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
             agents.get(i).setCollision(false);
             }


      /**
       * methodos i opoia ekteli to action gia ena agent kai vriski
       * tin epomeni katastasi (state)
       *
       * @param s
       * @param tempState
       * @param action
       * @param agent
       * @return
       */
      public void takeAction(MA_State_Variables s, MA_State_Variables
tempState,
                  int action, Q_Agent agent, ArrayList<Integer>
passDelivered) {
```

```
            if (agent.isTransfer() && (!(agent.isTransferPrevious()))
                        && (agent.getPrevAction() ==
MA_Gridworld.pickup))
                  agent.setTransferPrevious(true);
            if ((!agent.isTransfer()) && (agent.isTransferPrevious())
                        && (agent.getPrevAction() ==
MA_Gridworld.putdown))
                  agent.setTransferPrevious(false);

            int taxi = agent.getAgentPos();
            switch (action) {
            // north (move up)
            case 0: // ektos orion tou grid
                  if ((!((taxi - MA_Gridworld.size_x) < 0))
                              && (!(MA_Gridworld.checkForWall(taxi,
                                          (taxi -
MA_Gridworld.size_x))))) {
                        taxi = taxi - MA_Gridworld.size_x;
                  }
                  break;
            // south (move down)
            case 1: // ektos orion tou grid
                  if ((!((taxi + MA_Gridworld.size_y) >=
(MA_Gridworld.size_x * MA_Gridworld.size_y)))
                              && (!(MA_Gridworld.checkForWall(taxi,
                                          (taxi +
MA_Gridworld.size_y))))) {
                        taxi = taxi + MA_Gridworld.size_y;
                  }
                  break;
            // east (move right)
            case 2: // hit the wall or ektos orion tou grid
                  if ((!((taxi % MA_Gridworld.size_y) ==
(MA_Gridworld.size_y - 1)))
                              && (!(MA_Gridworld.checkForWall(taxi,
(taxi + 1))))) {
                        taxi = taxi + 1;
                  }
                  break;
            // west (move left)
            case 3: // hit the wall or ektos orion tou grid
                  if ((!((taxi % MA_Gridworld.size_y) == 0))
                              && (!(MA_Gridworld.checkForWall(taxi,
(taxi - 1))))) {
                        taxi = taxi - 1;
                  }
                  break;
            // pickup the passenger
            case 4:
                  // elegxos an metaferete kapios passenger apo afto
to taxi
                  if (!(agent.isTransfer()))
                        // elegxos an sti thesi pou imaste iparxi
passenger gia pick up
                        for (int i = 0; i <
MA_Gridworld.numOfPassengers; i++)
```

```
                                    if (taxi ==
MA_Gridworld.getPasLoc(s.passenger.get(i))) {
                                          //
MA_Gridworld.numOfStartPosition+1 antistixi sti thesi
                                          // opou o passenger einai mesa sto
taxi (-1)
                                          tempState.passenger
                                                      .set(i,
MA_Gridworld.numOfStartPosition);
                                          agent.setPassenger(i);
                                          agent.setTransfer(true);
                                          break;
                                    }
                        break;
                  // put down the passenger
                  case 5: // elegxos an metaferete kapios passenger apo afto
to taxi
                        if (agent.isTransfer())
                              // elegxos an imasten sto destination tou
passenger pou
                              // metaferete apo afto to taxi
                              if (taxi ==
MA_Gridworld.getDestLoc(s.destination.get(agent
                                          .getPassenger()))) {
                                    // Successful passenger delivery

      tempState.passenger.set(agent.getPassenger(), s.destination
                                                .get(agent.getPassenger()));
                                    passDelivered.add(agent.getPassenger());
                                    agent.setPassenger(-1);
                                    agent.setTransfer(false);
                              }
                        break;
                  case 6: // idle
                        break;
                  default:
                        System.err.println("Invalid number for
action..Exiting..");
                        System.exit(-1);
                  }

                  agent.setAgentPreviousPos(agent.getAgentPos());
                  agent.setAgentPos(taxi);
                  tempState.taxi.set(agent.getAgentNum(), taxi);
      }
```

```
	/**
	 * methodos i opoia epistrefi to reward gia 1 agent
	 *
	 * @param s
	 * @param tempState
	 * @param action
	 * @param agentNum
	 * @param agent
	 * @return
	 */
	public int findReward(MA_State_Variables s, MA_State_Variables
tempState,
			int action, int agentNum, Q_Agent agent) {
		int reward = 0;
		if (agent.isCollision())
			reward = MA_Gridworld.collision;

		switch (action) {
		case 0: // north (move up)
		case 1: // south (move down)
		case 2: // east (move right)
		case 3: // west (move left)
			if (s.taxi.get(agentNum) !=
tempState.taxi.get(agentNum)) {
				reward += MA_Gridworld.simple_move;
			} else {
				reward += MA_Gridworld.wall_hit;
			}
			break;
		// pickup the passenger
		case 4:
			if (agent.isTransfer() &&
(!(agent.isTransferPrevious()))) {
				reward += MA_Gridworld.simple_move;
			} else {
				reward += MA_Gridworld.wrong_get;
			}
			break;
		// put down the passenger
		case 5:
			if ((!agent.isTransfer()) &&
(agent.isTransferPrevious())) {
				// Successful passenger delivery
				reward += MA_Gridworld.success;
			} else {
				reward += MA_Gridworld.wrong_put;
			}
			break;
		case 6: // idle
			reward += MA_Gridworld.no_move;
			break;
		default:
			System.err.println("Invalid number for
action..Exiting..");
			System.exit(-1);
		}
		return reward;
	}
```

```
	/**
	 * methodos i opoia elegxi an enas agent (taxi) exi sigkrousti
	 * me kapio allo agent (taxi)
	 *
	 * @param tempState
	 * @param agentNum
	 * @param agent
	 * @param collisions
	 * @return
	 */
	public boolean collisionCheck(MA_State_Variables tempState, int
agentNum,
			Q_Agent agent, ArrayList<Integer> collisions) {
		int agentPos = tempState.taxi.get(agentNum);
		boolean col = false;
		// collisions.add(agentNum);
		for (int i = 0; i < tempState.taxi.size(); i++) {
			if (i == agentNum)
				continue;
			if (agentPos == tempState.taxi.get(i)) {

	collisions.add(i);
				col = true;
			}
		}
		if (col)
			collisions.add(0, agentNum);
		return col;
	}

	/**
	 * methodos i opoia apalifi tis sigkrousis
	 *
	 * @param tempState
	 * @param agents
	 * @param collisions
	 */
	public void removeCollisions(MA_State_Variables tempState,
			ArrayList<Q_Agent> agents, ArrayList<Integer>
collisions) {

		int temp = -1;
		boolean flag = false;
		for (int i = 0; i < collisions.size(); i++) {
			// an o agent stin prohgoumeni katastasi viskotan
stin idia thesi me
			// afti tin katastasi simeni oti aftos prepi na
paramini edw
			 if
(agents.get(collisions.get(i)).getAgentPreviousPos() == agents
			.get(collisions.get(i)).getAgentPos()) {

				flag = true;
				temp = i;
				break;
			}
```

```
			}
			if (!(flag)) {
				temp = findAgentWhoTakesTheAction(collisions);
			}

			for (int i = 0; i < collisions.size(); i++) {
				if (i != temp) {
					agents.get(collisions.get(i)).setAgentPos(

	agents.get(collisions.get(i)).getAgentPreviousPos());
					tempState.taxi.set(collisions.get(i),
agents.get(

	collisions.get(i)).getAgentPreviousPos());
				}
			}

		}

		/**
		 * methodos i opoia vriski vasi pithanotitwn pios agent tha kani
		 * tin kinisi
		 * @param collisions
		 * @return
		 */
		public int findAgentWhoTakesTheAction(ArrayList<Integer>
collisions) {
			int n = collisions.size();
			double probTable[] = new double[n];
			for (int i = 0; i < n; i++)
				probTable[i] = (double) 1 / n;
			double offset = 0;
			double rand = MA_Taxi_Problem.r.nextDouble();
			for (int i = 0; i < probTable.length; i++) {
				offset += probTable[i];
				if (offset > rand)
					return i;
			}
			return MA_Taxi_Problem.r.nextInt(probTable.length);// en
tha erti potte
			// dame
		}

		/**
		 * @param args
		 */
		public static void main(String[] args) {
			// TODO Auto-generated method stub

		}
}
```

```
import java.util.ArrayList;

/**
 * klasi i opoia einai ipefthini gia ton elegxo twn agent
 *
 * @author Giannis
 *
 */
public class MaxQ_Controller {

	/**
	 * methodos i opoia ine ipefthini gia tis kinisis twn agent kai
	 * ta rewards pou pairnoun
	 *
	 * @param s
	 * @param tempState
	 * @param agents
	 * @param passDelivered
	 * @param index
	 * @param exploration
	 * @param episodeSteps
	 */
	public void agentController(MA_State_Variables s,
			MA_State_Variables tempState, ArrayList<MaxQ_Agent>
agents,
			ArrayList<Integer> passDelivered, int index) {

		ArrayList<Integer> collisions = new ArrayList<Integer>();

		// take action
		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
			// if (!(agents.get(i).isCollision())) {
			takeAction(s, tempState, agents.get(i).getAction(),
agents.get(i),
					passDelivered);
			// }
		}
		// collision check
		// boolean flag;
		// do {
		// System.out.println("apoel");
		// flag = false;
		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
			if (collisionCheck(tempState, i, agents.get(i),
collisions)) {
				// flag = true;
				for (int j = 0; j < collisions.size(); j++) {

	agents.get(collisions.get(j)).setCollision(true);
				}
				removeCollisions(tempState, agents,
collisions);
				// diagrafi collision apo ton agent
```

```
                        // for (int j = 0; j <
MA_Gridworld.numOfAgents; j++) {
                        // agents.get(j).setCollision(false);
                        // }
                  }
                  collisions.clear();
            }
            // } while (flag);

            // get rewards
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  agents.get(i).setReward(
                              findReward(s, tempState,
agents.get(i).getAction(), i,
                                          agents.get(i)));
            }
            // diagrafi collision apo ton agent
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  agents.get(i).setCollision(false);
            }
      }

      /**
       * methodos i opoia ekteli to action gia ena agent kai vriski
       * tin epomeni katastasi (state)
       *
       * @param s
       * @param tempState
       * @param action
       * @param agent
       * @return
       */
      public void takeAction(MA_State_Variables s, MA_State_Variables
tempState,
                  int action, MaxQ_Agent agent, ArrayList<Integer>
passDelivered) {

            if (agent.isTransfer() && (!(agent.isTransferPrevious()))
                        && (agent.getPrevAction() ==
MA_Gridworld.pickup))
                  agent.setTransferPrevious(true);
            if ((!agent.isTransfer()) && (agent.isTransferPrevious())
                        && (agent.getPrevAction() ==
MA_Gridworld.putdown))
                  agent.setTransferPrevious(false);

            int taxi = agent.getAgentPos();
            switch (action) {
            // north (move up)
            case 0: // ektos orion tou grid
                  if ((!((taxi - MA_Gridworld.size_x) < 0))
                              && (!(MA_Gridworld.checkForWall(taxi,
                                          (taxi -
MA_Gridworld.size_x))))) {
                        taxi = taxi - MA_Gridworld.size_x;
                  }
                  break;
            // south (move down)
```

```
            case 1: // ektos orion tou grid
                  if ((!((taxi + MA_Gridworld.size_y) >=
(MA_Gridworld.size_x * MA_Gridworld.size_y)))
                              && (!(MA_Gridworld.checkForWall(taxi,
                                          (taxi +
MA_Gridworld.size_y))))) {
                        taxi = taxi + MA_Gridworld.size_y;
                  }
                  break;
            // east (move right)
            case 2: // hit the wall or ektos orion tou grid
                  if ((!((taxi % MA_Gridworld.size_y) ==
(MA_Gridworld.size_y - 1)))
                              && (!(MA_Gridworld.checkForWall(taxi,
(taxi + 1))))) {
                        taxi = taxi + 1;
                  }
                  break;
            // west (move left)
            case 3: // hit the wall or ektos orion tou grid
                  if ((!((taxi % MA_Gridworld.size_y) == 0))
                              && (!(MA_Gridworld.checkForWall(taxi,
(taxi - 1))))) {
                        taxi = taxi - 1;
                  }
                  break;
            // pickup the passenger
            case 4:
                  // elegxos an metaferete kapios passenger apo afto
to taxi
                  if (!(agent.isTransfer()))
                        // elegxos an sti thesi pou imaste iparxi
passenger gia pick up
                        for (int i = 0; i <
MA_Gridworld.numOfPassengers; i++)
                              if (taxi ==
MA_Gridworld.getPasLoc(s.passenger.get(i))) {
                                    //
MA_Gridworld.numOfStartPosition+1 antistixi sti thesi
                                    // opou o passenger einai mesa sto
taxi (-1)
                                    tempState.passenger.set(i,

      MA_Gridworld.numOfStartPosition);
                                    agent.setPassenger(i);
                                    agent.setTransfer(true);
                                    break;
                              }
                  break;
            // put down the passenger
            case 5: // elegxos an metaferete kapios passenger apo afto
to taxi
                  if (agent.isTransfer())
                        // elegxos an imasten sto destination tou
passenger pou
                        // metaferete apo afto to taxi
                        if (taxi ==
MA_Gridworld.getDestLoc(s.destination.get(agent
```

```
                                        .getPassenger())))) {
                                    // Successful passenger delivery

      tempState.passenger.set(agent.getPassenger(), s.destination
                                              .get(agent.getPassenger()));
                                    passDelivered.add(agent.getPassenger());
                                    agent.setPassenger(-1);
                                    agent.setTransfer(false);
                              }
                        break;
                  case 6: // idle
                        break;
                  default:
                        System.err.println("Invalid number for
action..Exiting..");
                        System.exit(-1);
                  }

                  agent.setAgentPreviousPos(agent.getAgentPos());
                  agent.setAgentPos(taxi);
                  tempState.taxi.set(agent.getAgentNum(), taxi);
      }

      /**
       * methodos i opoia epistrefi to reward gia 1 agent
       *
       * @param s
       * @param tempState
       * @param action
       * @param agentNum
       * @param agent
       * @return
       */
      public int findReward(MA_State_Variables s, MA_State_Variables
tempState,
                  int action, int agentNum, MaxQ_Agent agent) {

            int reward = 0;
            if (agent.isCollision())
                  reward = MA_Gridworld.collision;

            switch (action) {
            case 0: // north (move up)
            case 1: // south (move down)
            case 2: // east (move right)
            case 3: // west (move left)
                  if (s.taxi.get(agentNum) !=
tempState.taxi.get(agentNum)) {
                        reward += MA_Gridworld.simple_move;
                  } else {
                        reward += MA_Gridworld.wall_hit;
                  }
                  break;
            // pickup the passenger
            case 4:
                  if (agent.isTransfer() &&
(!(agent.isTransferPrevious()))) {
                        reward += MA_Gridworld.simple_move;
```

```
                        } else {
                              reward += MA_Gridworld.wrong_get;
                        }
                        break;
                  // put down the passenger
                  case 5:
                        if ((!agent.isTransfer()) &&
(agent.isTransferPrevious())) {
                              // Successful passenger delivery
                              reward += MA_Gridworld.success;
                        } else {
                              reward += MA_Gridworld.wrong_put;
                        }
                        break;
                  case 6: // idle
                        reward += MA_Gridworld.no_move;
                        break;
                  default:
                        System.err.println("Invalid number for
action..Exiting..");
                        System.exit(-1);
                  }

                  return reward;
            }

            /**
             * methodos i opoia elegxi an enas agent (taxi) exi sigkrousti
me kapio allo
             * agent (taxi)
             *
             * @param tempState
             * @param agentNum
             * @param agent
             * @param collisions
             * @return
             */
            public boolean collisionCheck(MA_State_Variables tempState, int
agentNum,
                        MaxQ_Agent agent, ArrayList<Integer> collisions) {
                  int agentPos = tempState.taxi.get(agentNum);
                  boolean col = false;
                  // collisions.add(agentNum);
                  for (int i = 0; i < tempState.taxi.size(); i++) {
                        if (i == agentNum)
                              continue;
                        if (agentPos == tempState.taxi.get(i)) {
                              collisions.add(i);
                              col = true;
                        }
                  }
                  if (col)
                        collisions.add(0, agentNum);
                  return col;
            }
```

```
	/**
	 * methodos i opoia apalifi tis sigkrousis
	 *
	 * @param tempState
	 * @param agents
	 * @param collisions
	 */
	public void removeCollisions(MA_State_Variables tempState,
			ArrayList<MaxQ_Agent> agents, ArrayList<Integer>
collisions) {

		int temp = -1;
		boolean flag = false;
		for (int i = 0; i < collisions.size(); i++) {
			// an o agent stin prohgoumeni katastasi viskotan
stin idia thesi me
			// afti tin katastasi simeni oti aftos prepi na
paramini edw
			if
(agents.get(collisions.get(i)).getAgentPreviousPos() == agents
					.get(collisions.get(i)).getAgentPos()) {
				flag = true;
				temp = i;
				break;
			}
		}
		if (!(flag)) {
			temp = findAgentWhoTakesTheAction(collisions);
		}
		for (int i = 0; i < collisions.size(); i++) {
			if (i != temp) {
				agents.get(collisions.get(i)).setAgentPos(

	agents.get(collisions.get(i)).getAgentPreviousPos());
				tempState.taxi.set(collisions.get(i),
agents.get(

	collisions.get(i)).getAgentPreviousPos());
			}
		}
	}

	/**
	 * methodos i opoia vriski vasi pithanotitwn pios agent tha kani
tin kinisi
	 *
	 * @param collisions
	 * @return
	 */
	public int findAgentWhoTakesTheAction(ArrayList<Integer>
collisions) {
		int n = collisions.size();
		double probTable[] = new double[n];
		for (int i = 0; i < n; i++)
			probTable[i] = (double) 1 / n;
		double offset = 0;
		double rand = MA_MaxQ.r.nextDouble();
		for (int i = 0; i < probTable.length; i++) {
```

```
                  offset += probTable[i];
                  if (offset > rand)
                        return i;
            }
            return MA_MaxQ.r.nextInt(probTable.length);// en tha erti
potte
            // dame
      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub

      }

}
```

PHC_Controller.java

```
import java.util.ArrayList;

/**
 * klasi i opoia einai ipefthini gia ton elegxo twn PHC_Agent
 *
 * @author Giannis
 *
 */
public class PHC_Controller {

      /**
       * methodos i opoia ine ipefthini gia tis kinisis twn agent kai
       * ta rewards pou pairnoun
       *
       * @param s
       * @param tempState
       * @param agents
       * @param passDelivered
       * @param index
       * @param exploration
       * @param episodeSteps
       */
      public void agentController(MA_Taxi_Problem tp,
MA_State_Variables s,
                  MA_State_Variables tempState, ArrayList<PHC_Agent>
agents,
                  ArrayList<Integer> passDelivered, int index, int
step,
                  boolean changeEpsilon) {

            ArrayList<Integer> collisions = new ArrayList<Integer>();
            // choose action
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
```

```
      agents.get(i).setPrevAction(agents.get(i).getAction());
                  agents.get(i).setAction(
                              this.chooseAction(step,
agents.get(i).getPolicyValue(),
                                                index, tp, changeEpsilon));
                  agents.get(i).addActionCounter(index,
agents.get(i).getAction());
            }

            // take action
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  // if (!(agents.get(i).isCollision())) {
                  takeAction(s, tempState, agents.get(i).getAction(),
agents.get(i),
                              passDelivered);
                  // }
            }
            // collision check
            // boolean flag;
            // do {
            // System.out.println("apoel");
            // flag = false;
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  if (collisionCheck(tempState, i, agents.get(i),
collisions)) {
                        // flag = true;
                        for (int j = 0; j < collisions.size(); j++) {

      agents.get(collisions.get(j)).setCollision(true);
                        }
                        removeCollisions(tempState, agents,
collisions);
                        // diagrafi collision apo ton agent
                        // for (int j = 0; j <
MA_Gridworld.numOfAgents; j++) {
                        // agents.get(j).setCollision(false);
                        // }
                  }
                  collisions.clear();
            }
            // } while (flag);

            // get rewards
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  agents.get(i).setReward(
                              findReward(s, tempState,
agents.get(i).getAction(), i,
                                                agents.get(i)));
            }
            // diagrafi collision apo ton agent
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  agents.get(i).setCollision(false);
            }
      }
```

```
	/**
	 * methodos i opoia ekteli to action gia ena agent kai vriski
	 * tin epomeni katastasi (state)
	 *
	 * @param s
	 * @param tempState
	 * @param action
	 * @param agent
	 * @return
	 */
	public void takeAction(MA_State_Variables s, MA_State_Variables
tempState,
			int action, PHC_Agent agent, ArrayList<Integer>
passDelivered) {

		if (agent.isTransfer() && (!(agent.isTransferPrevious()))
				&& (agent.getPrevAction() ==
MA_Gridworld.pickup))
			agent.setTransferPrevious(true);
		if ((!agent.isTransfer()) && (agent.isTransferPrevious())
				&& (agent.getPrevAction() ==
MA_Gridworld.putdown))
			agent.setTransferPrevious(false);

		int taxi = agent.getAgentPos();
		switch (action) {
		// north (move up)
		case 0: // ektos orion tou grid
			if ((!((taxi - MA_Gridworld.size_x) < 0))
					&& (!(MA_Gridworld.checkForWall(taxi,
							(taxi -
MA_Gridworld.size_x))))) {
				taxi = taxi - MA_Gridworld.size_x;
			}
			break;
		// south (move down)
		case 1: // ektos orion tou grid
			if ((!((taxi + MA_Gridworld.size_y) >=
(MA_Gridworld.size_x * MA_Gridworld.size_y)))
					&& (!(MA_Gridworld.checkForWall(taxi,
							(taxi +
MA_Gridworld.size_y))))) {
				taxi = taxi + MA_Gridworld.size_y;
			}
			break;
		// east (move right)
		case 2: // hit the wall or ektos orion tou grid
			if ((!((taxi % MA_Gridworld.size_y) ==
(MA_Gridworld.size_y - 1)))
					&& (!(MA_Gridworld.checkForWall(taxi,
(taxi + 1))))) {
				taxi = taxi + 1;
			}
			break;
		// west (move left)
		case 3: // hit the wall or ektos orion tou grid
			if ((!((taxi % MA_Gridworld.size_y) == 0))
```

```
                                    && (!(MA_Gridworld.checkForWall(taxi,
(taxi - 1))))) {
                              taxi = taxi - 1;
                        }
                        break;
                  // pickup the passenger
                  case 4:
                        // elegxos an metaferete kapios passenger apo afto
to taxi
                        if (!(agent.isTransfer()))
                              // elegxos an sti thesi pou imaste iparxi
passenger gia pick up
                              for (int i = 0; i <
MA_Gridworld.numOfPassengers; i++)
                                    // if(s.passenger.get(i)!=-1)
                                    if (taxi ==
MA_Gridworld.getPasLoc(s.passenger.get(i))) {
                                          //
MA_Gridworld.numOfStartPosition+1 antistixi sti thesi
                                          // opou o passenger einai mesa sto
taxi (-1)
                                          // tempState.passenger
                                          // .set(i,
                                          //
MA_Gridworld.getPasLoc(MA_Gridworld.numOfStartPosition));
                                          tempState.passenger.set(i,

      MA_Gridworld.numOfStartPosition);
                                          agent.setPassenger(i);
                                          agent.setTransfer(true);
                                          break;
                                    }
                        break;
                  // put down the passenger
                  case 5: // elegxos an metaferete kapios passenger apo afto
to taxi
                        if (agent.isTransfer())
                              // elegxos an imasten sto destination tou
passenger pou
                              // metaferete apo afto to taxi
                              if (taxi ==
MA_Gridworld.getDestLoc(s.destination.get(agent
                                          .getPassenger()))) {
                                    // Successful passenger delivery

      tempState.passenger.set(agent.getPassenger(), s.destination
                                                .get(agent.getPassenger()));
                                    passDelivered.add(agent.getPassenger());
                                    agent.setPassenger(-1);
                                    agent.setTransfer(false);
                              }
                        break;
                  case 6: // idle
                        break;
                  default:
                        System.err.println("Invalid number for
action..Exiting..");
                        System.exit(-1);
```

```
            }

            agent.setAgentPreviousPos(agent.getAgentPos());
            agent.setAgentPos(taxi);
            tempState.taxi.set(agent.getAgentNum(), taxi);
      }

      /**
       * methodos i opoia epistrefi to reward gia 1 agent
       *
       * @param s
       * @param tempState
       * @param action
       * @param agentNum
       * @param agent
       * @return
       */
      public int findReward(MA_State_Variables s, MA_State_Variables
tempState,
                  int action, int agentNum, PHC_Agent agent) {

            int reward = 0;
            if (agent.isCollision())
                  reward = MA_Gridworld.collision;

            switch (action) {
            case 0: // north (move up)
            case 1: // south (move down)
            case 2: // east (move right)
            case 3: // west (move left)
                  if (s.taxi.get(agentNum) !=
tempState.taxi.get(agentNum)) {
                        reward += MA_Gridworld.simple_move;
                  } else {
                        reward += MA_Gridworld.wall_hit;
                  }
                  break;
            // pickup the passenger
            case 4:
                  if (agent.isTransfer() &&
(!(agent.isTransferPrevious()))) {
                        reward += MA_Gridworld.simple_move;
                  } else {
                        reward += MA_Gridworld.wrong_get;
                  }
                  break;
            // put down the passenger
            case 5:
                  if ((!agent.isTransfer()) &&
(agent.isTransferPrevious())) {
                        // Successful passenger delivery
                        reward += MA_Gridworld.success;
                  } else {
                        reward += MA_Gridworld.wrong_put;
                  }
                  break;
            case 6: // idle
                  reward += MA_Gridworld.no_move;
```

```
                  break;
            default:
                  System.err.println("Invalid number for
action..Exiting..");
                  System.exit(-1);
            }

            return reward;
      }

      /**
       * methodos i opoia elegxi an enas agent (taxi) exi sigkrousti
me kapio allo
       * agent (taxi)
       *
       * @param tempState
       * @param agentNum
       * @param agent
       * @param collisions
       * @return
       */
      public boolean collisionCheck(MA_State_Variables tempState, int
agentNum,
                  PHC_Agent agent, ArrayList<Integer> collisions) {
            int agentPos = tempState.taxi.get(agentNum);
            boolean col = false;
            // collisions.add(agentNum);
            for (int i = 0; i < tempState.taxi.size(); i++) {
                  if (i == agentNum)
                        continue;
                  if (agentPos == tempState.taxi.get(i)) {
                        collisions.add(i);
                        col = true;
                  }
            }
            if (col)
                  collisions.add(0, agentNum);
            return col;
      }

      /**
       * methodos i opoia apalifi tis sigkrousis
       *
       * @param tempState
       * @param agents
       * @param collisions
       */
      public void removeCollisions(MA_State_Variables tempState,
                  ArrayList<PHC_Agent> agents, ArrayList<Integer>
collisions) {

            int temp = -1;
            boolean flag = false;
            for (int i = 0; i < collisions.size(); i++) {
                  // an o agent stin prohgoumeni katastasi viskotan
stin idia thesi me
                  // afti tin katastasi simeni oti aftos prepi na
paramini edw
```

```
                  if
(agents.get(collisions.get(i)).getAgentPreviousPos() == agents
                              .get(collisions.get(i)).getAgentPos()) {
                        flag = true;
                        temp = i;
                        break;
                  }
            }

            if (!(flag)) {
                  temp = findAgentWhoTakesTheAction(collisions);
            }
            for (int i = 0; i < collisions.size(); i++) {
                  if (i != temp) {
                        // System.out.println("damesa");
                        agents.get(collisions.get(i)).setAgentPos(

      agents.get(collisions.get(i)).getAgentPreviousPos());
                        tempState.taxi.set(collisions.get(i),
agents.get(

      collisions.get(i)).getAgentPreviousPos());
                  }
            }

      }

      /**
       * methodos i opoia vriski vasi pithanotitwn pios agent tha kani
       * tin kinisi
       * @param collisions
       * @return
       */
      public int findAgentWhoTakesTheAction(ArrayList<Integer>
collisions) {
            int n = collisions.size();
            double probTable[] = new double[n];
            for (int i = 0; i < n; i++)
                  probTable[i] = (double) 1 / n;
            double offset = 0;
            double rand = MA_Taxi_Problem.r.nextDouble();
            for (int i = 0; i < probTable.length; i++) {
                  offset += probTable[i];
                  if (offset > rand)
                        return i;
            }
            return MA_Taxi_Problem.r.nextInt(probTable.length);// en
tha erti potte
            // dame
      }
```

```
	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
sigkekrimeni
	 * katastasi (state) me pithanotita p(s,a) kai me kapia
anixnefsi
	 *
	 * @param index
	 * @return
	 */
	public int chooseAction(int step, double[][] policyValue, int
index,
			MA_Taxi_Problem tp, boolean changeEpsilon) {
		if (changeEpsilon)
			tp.changeEpsilon(step);

		if (MA_Taxi_Problem.r.nextDouble() < tp.getEpsilon())
			return
(MA_Taxi_Problem.r.nextInt(MA_Gridworld.actions));
		return this.chooseProbAction(policyValue, index);
	}

	/**
	 * methodos i opoia epilegi mia kinisi (action) vasi tis
pithanotitas p(s,a)
	 *
	 * @param index
	 * @return
	 */
	public int chooseProbAction(double[][] policyValue, int index) {
		double offset = 0;
		double rand = MA_Taxi_Problem.r.nextDouble();
		for (int i = 0; i < MA_Gridworld.actions; i++) {
			offset += policyValue[index][i];
			if (offset >= rand)
				return i;
		}
		return 5; // en tha erti pote dame

	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

```
import java.util.ArrayList;

/**
 * klasi i opoia einai ipefthini gia ton elegxo twn WoLF-PHC_Agent
 *
 * @author Giannis
 *
 */
public class WP_Controller {
	/**
	 * methodos i opoia ine ipefthini gia tis kinisis twn agent kai
	 * ta rewards pou pairnoun
	 *
	 * @param s
	 * @param tempState
	 * @param agents
	 * @param passDelivered
	 * @param index
	 * @param exploration
	 * @param episodeSteps
	 */
	public void agentController(MA_Taxi_Problem tp,
MA_State_Variables s,
			MA_State_Variables tempState, ArrayList<WP_Agent>
agents,
			ArrayList<Integer> passDelivered, int index, int
step,
			boolean changeEpsilon) {

		ArrayList<Integer> collisions = new ArrayList<Integer>();
		// choose action
		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {

	agents.get(i).setPrevAction(agents.get(i).getAction());
			agents.get(i).setAction(
					this.chooseAction(step,
agents.get(i).getPolicyValue(),
							index, tp, changeEpsilon));
			agents.get(i).addActionCounter(index,
agents.get(i).getAction());
		}

		// take action
		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
			// if (!(agents.get(i).isCollision())) {
			takeAction(s, tempState, agents.get(i).getAction(),
agents.get(i),
					passDelivered);
			// }
		}
		// collision check
		// boolean flag;
		// do {
```

```
            // System.out.println("apoel");
            // flag = false;
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  if (collisionCheck(tempState, i, agents.get(i),
collisions)) {
                        // flag = true;
                        for (int j = 0; j < collisions.size(); j++) {

      agents.get(collisions.get(j)).setCollision(true);
                        }
                        removeCollisions(tempState, agents,
collisions);
                        // diagrafi collision apo ton agent
                        // for (int j = 0; j <
MA_Gridworld.numOfAgents; j++) {
                        // agents.get(j).setCollision(false);
                        // }
                  }
                  collisions.clear();
            }
            // } while (flag);

            // get rewards
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  agents.get(i).setReward(
                              findReward(s, tempState,
agents.get(i).getAction(), i,
                                          agents.get(i)));
            }
            // diagrafi collision apo ton agent
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  agents.get(i).setCollision(false);
            }

      }

      /**
       * methodos i opoia ekteli to action gia ena agent kai vriski
       * tin epomeni katastasi (state)
       *
       * @param s
       * @param tempState
       * @param action
       * @param agent
       * @return
       */
      public void takeAction(MA_State_Variables s, MA_State_Variables
tempState,
                  int action, WP_Agent agent, ArrayList<Integer>
passDelivered) {

            if (agent.isTransfer() && (!(agent.isTransferPrevious()))
                        && (agent.getPrevAction() ==
MA_Gridworld.pickup))
                  agent.setTransferPrevious(true);
            if ((!agent.isTransfer()) && (agent.isTransferPrevious())
                        && (agent.getPrevAction() ==
MA_Gridworld.putdown))
```

```
                  agent.setTransferPrevious(false);

            int taxi = agent.getAgentPos();
            switch (action) {
            // north (move up)
            case 0: // ektos orion tou grid
                  if ((!((taxi - MA_Gridworld.size_x) < 0))
                              && (!(MA_Gridworld.checkForWall(taxi,
                                          (taxi -
MA_Gridworld.size_x))))) {
                        taxi = taxi - MA_Gridworld.size_x;
                  }
                  break;
            // south (move down)
            case 1: // ektos orion tou grid
                  if ((!((taxi + MA_Gridworld.size_y) >=
(MA_Gridworld.size_x * MA_Gridworld.size_y)))
                              && (!(MA_Gridworld.checkForWall(taxi,
                                          (taxi +
MA_Gridworld.size_y))))) {
                        taxi = taxi + MA_Gridworld.size_y;
                  }
                  break;
            // east (move right)
            case 2: // hit the wall or ektos orion tou grid
                  if ((!((taxi % MA_Gridworld.size_y) ==
(MA_Gridworld.size_y - 1)))
                              && (!(MA_Gridworld.checkForWall(taxi,
(taxi + 1))))) {
                        taxi = taxi + 1;
                  }
                  break;
            // west (move left)
            case 3: // hit the wall or ektos orion tou grid
                  if ((!((taxi % MA_Gridworld.size_y) == 0))
                              && (!(MA_Gridworld.checkForWall(taxi,
(taxi - 1))))) {
                        taxi = taxi - 1;
                  }
                  break;
            // pickup the passenger
            case 4:
                  // elegxos an metaferete kapios passenger apo afto
to taxi
                  if (!(agent.isTransfer()))
                        // elegxos an sti thesi pou imaste iparxi
passenger gia pick up
                        for (int i = 0; i <
MA_Gridworld.numOfPassengers; i++)
                              // if(s.passenger.get(i)!=-1)
                              if (taxi ==
MA_Gridworld.getPasLoc(s.passenger.get(i))) {
                                    //
MA_Gridworld.numOfStartPosition+1 antistixi sti thesi
                                    // opou o passenger einai mesa sto
taxi (-1)
                                    tempState.passenger.set(i,
```

```
      MA_Gridworld.numOfStartPosition);
                                    agent.setPassenger(i);
                                    agent.setTransfer(true);
                                    break;
                              }
                        break;
                  // put down the passenger
                  case 5: // elegxos an metaferete kapios passenger apo afto
to taxi
                        if (agent.isTransfer())
                              // elegxos an imasten sto destination tou
passenger pou
                              // metaferete apo afto to taxi
                              if (taxi ==
MA_Gridworld.getDestLoc(s.destination.get(agent
                                          .getPassenger()))) {
                                    // Successful passenger delivery

      tempState.passenger.set(agent.getPassenger(), s.destination
                                                .get(agent.getPassenger()));
                                    passDelivered.add(agent.getPassenger());
                                    agent.setPassenger(-1);
                                    agent.setTransfer(false);
                              }
                        break;
                  case 6: // idle
                        break;
                  default:
                        System.err.println("Invalid number for
action..Exiting..");
                        System.exit(-1);
                  }

                  agent.setAgentPreviousPos(agent.getAgentPos());
                  agent.setAgentPos(taxi);
                  tempState.taxi.set(agent.getAgentNum(), taxi);
      }

      /**
       * methodos i opoia epistrefi to reward gia 1 agent
       *
       * @param s
       * @param tempState
       * @param action
       * @param agentNum
       * @param agent
       * @return
       */
      public int findReward(MA_State_Variables s, MA_State_Variables
tempState,
                  int action, int agentNum, WP_Agent agent) {

            int reward = 0;
            if (agent.isCollision())
                  reward = MA_Gridworld.collision;

            switch (action) {
```

```
            case 0: // north (move up)
            case 1: // south (move down)
            case 2: // east (move right)
            case 3: // west (move left)
                  if (s.taxi.get(agentNum) !=
tempState.taxi.get(agentNum)) {
                        reward += MA_Gridworld.simple_move;
                  } else {
                        reward += MA_Gridworld.wall_hit;
                  }
                  break;
            // pickup the passenger
            case 4:
                  if (agent.isTransfer() &&
(!(agent.isTransferPrevious()))) {
                        reward += MA_Gridworld.simple_move;
                  } else {
                        reward += MA_Gridworld.wrong_get;
                  }
                  break;
            // put down the passenger
            case 5:
                  if ((!agent.isTransfer()) &&
(agent.isTransferPrevious())) {
                        // Successful passenger delivery
                        reward += MA_Gridworld.success;
                  } else {
                        reward += MA_Gridworld.wrong_put;
                  }
                  break;
            case 6: // idle
                  reward += MA_Gridworld.no_move;
                  break;
            default:
                  System.err.println("Invalid number for
action..Exiting..");
                  System.exit(-1);
            }

            return reward;
      }

      /**
       * methodos i opoia elegxi an enas agent (taxi) exi sigkrousti
       * me kapio allo agent (taxi)
       *
       * @param tempState
       * @param agentNum
       * @param agent
       * @param collisions
       * @return
       */
      public boolean collisionCheck(MA_State_Variables tempState, int
agentNum,
                  WP_Agent agent, ArrayList<Integer> collisions) {
            int agentPos = tempState.taxi.get(agentNum);
            boolean col = false;
            // collisions.add(agentNum);
```

```
            for (int i = 0; i < tempState.taxi.size(); i++) {
                  if (i == agentNum)
                        continue;
                  if (agentPos == tempState.taxi.get(i)) {
                        collisions.add(i);
                        col = true;
                  }
            }
            if (col)
                  collisions.add(0, agentNum);
            return col;
      }

      /**
       * methodos i opoia apalifi tis sigkrousis
       *
       * @param tempState
       * @param agents
       * @param collisions
       */
      public void removeCollisions(MA_State_Variables tempState,
                  ArrayList<WP_Agent> agents, ArrayList<Integer>
collisions) {

            int temp = -1;
            boolean flag = false;
            for (int i = 0; i < collisions.size(); i++) {
                  // an o agent stin prohgoumeni katastasi viskotan
stin idia thesi me
                  // afti tin katastasi simeni oti aftos prepi na
paramini edw
                  if
(agents.get(collisions.get(i)).getAgentPreviousPos() == agents
                              .get(collisions.get(i)).getAgentPos()) {
                        flag = true;
                        temp = i;
                        break;
                  }
            }
            if (!(flag)) {
                  temp = findAgentWhoTakesTheAction(collisions);
            }
            for (int i = 0; i < collisions.size(); i++) {
                  if (i != temp) {
                        // System.out.println("damesa");
                        agents.get(collisions.get(i)).setAgentPos(

      agents.get(collisions.get(i)).getAgentPreviousPos());
                        tempState.taxi.set(collisions.get(i),
agents.get(

      collisions.get(i)).getAgentPreviousPos());
                  }
            }
      }
```

```
	/**
	 * methodos i opoia vriski vasi pithanotitwn pios agent tha kani
	 * tin kinisi
	 * @param collisions
	 * @return
	 */
	public int findAgentWhoTakesTheAction(ArrayList<Integer>
collisions) {
		int n = collisions.size();
		double probTable[] = new double[n];
		for (int i = 0; i < n; i++)
			probTable[i] = (double) 1 / n;
		double offset = 0;
		double rand = MA_Taxi_Problem.r.nextDouble();
		for (int i = 0; i < probTable.length; i++) {
			offset += probTable[i];
			if (offset > rand)
				return i;
		}
		return MA_Taxi_Problem.r.nextInt(probTable.length);// en
tha erti potte
		// dame
	}

	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
sigkekrimeni
	 * katastasi (state) me pithanotita p(s,a) kai me kapia
anixnefsi
	 *
	 * @param index
	 * @return
	 */
	public int chooseAction(int step, double[][] policyValue, int
index,
			MA_Taxi_Problem tp, boolean changeEpsilon) {
		// if (tp.getEpsilon() != 0)
		if (changeEpsilon)
			tp.changeEpsilon(step);

		if (MA_Taxi_Problem.r.nextDouble() < tp.getEpsilon())
			return
(MA_Taxi_Problem.r.nextInt(MA_Gridworld.actions));
		return this.chooseProbAction(policyValue, index);
	}

	/**
	 * methodos i opoia epilegi mia kinisi (action) vasi tis
	 * pithanotitas p(s,a)
	 * @param index
	 * @return
	 */
	public int chooseProbAction(double[][] policyValue, int index) {
		double offset = 0;
		double rand = MA_Taxi_Problem.r.nextDouble();
		for (int i = 0; i < MA_Gridworld.actions; i++) {
			offset += policyValue[index][i];
			if (offset >= rand)
```

```
                        return i;
                }
                return 5; // en tha erti pote dame
        }
}
```

MAQ_Learning.java

```
import java.util.ArrayList;

/**
 * klasi i opoia ilopoiei ton algorithmo Q-Learning gia to Taxi
 * Problem gia multi agent
 *
 * @author Giannis
 *
 */
public class MAQ_Learning {

      protected ArrayList<Q_Agent> agents = new ArrayList<Q_Agent>();
      int countStep;
      int count;

      // Taxi_Problem tp = new Taxi_Problem();
      /**
       * methodos i opia ekpedevi ton agent
       */
      public void calculate(int max_episode, boolean exploration,
                  MA_Taxi_Problem tp, double epsilon, double
temperature,
                  double coolingRate, String filename, double lr,
double df,
                  int delivery, int num, int length) {
            MA_Gridworld.makeGridworld(filename);
            tp.setRanges();
            tp.initOffsetTable();

            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  Q_Agent tmp = new Q_Agent(i, -1, 1);
                  agents.add(tmp);
            }
            double timer;
            // botzmann exploration
            if (!(exploration))
                  tp.setCoolingRate(coolingRate);
            tp.setEpsilon(epsilon);
            tp.setLearningRate(lr);
            tp.setDiscountFactor(df);
            MA_State_Variables s;
            tp.makeNewEpisodeSteps(length);
            tp.makeNewTimePerEpisode(length);
            count=0;
            for (int i = 0; i < max_episode; i++) {
                  if ((count % num == 0))
                        System.out.println("episode=" + count);
```

```
                    count++;
                    if (!(exploration))
                          tp.setTemperature(temperature);
                    timer = (double) System.currentTimeMillis() / 1000;
                    //System.out.println("episode=" + i);
                    s = new MA_State_Variables(MA_Gridworld.size_x,
                                MA_Gridworld.size_y,
MA_Gridworld.numOfStartPosition,
                                MA_Gridworld.numOfGoals);
                    for (int j = 0; j < s.taxi.size(); j++) {
                          agents.get(j).setAgentPos(s.taxi.get(j));
                    }
                    this.q_learning(s, i, exploration, tp,
delivery,num);
                    if (num != 1) {
                          if (i == 0)
                                tp.setTimePerEpisode(i, ((double) System
                                            .currentTimeMillis() / 1000
- timer));
                          else if ((count % num == 0))
                                tp.setTimePerEpisode((i + 1) / num,
((double) System
                                            .currentTimeMillis() / 1000
- timer));
                    } else {
                          tp.setTimePerEpisode(i,
                                      ((double)
System.currentTimeMillis() / 1000 - timer));
                    }
                    for (int j = 0; j < s.taxi.size(); j++) {
                          agents.get(j).setAgentPreviousPos(-1);
                          agents.get(j).setPassenger(-1);
                          agents.get(j).setTransfer(false);
                          agents.get(j).setTransferPrevious(false);
                          agents.get(j).setCollision(false);
                          agents.get(j).setAction(-1);
                          agents.get(j).setPrevAction(-1);
                          agents.get(j).setReward(-1);
                    }
              }
              Runtime.getRuntime().gc();
              /*
               * PrintInFile p = new PrintInFile(); if (exploration) {
               * p.printResults("Q-Learning_Results e-greedy.txt", tp
               * .getEpisodeSteps(), max_episode);
               * p.printResults("Q-Learning_clock time e-greedy.txt", tp
               * .getTimePerEpisode(), max_episode); } else {
               * p.printResults("Q-Learning_Results-Boltzmann.txt", tp
               * .getEpisodeSteps(), max_episode);
               * p.printResults("Q-Learning_clock time - Boltzmann.txt",
tp
               * .getTimePerEpisode(), max_episode); }
               */
      }
```

```
	/**
	 * methodos i opoia ilopoiei to taxi problem simfona me ton
	 * algorithmo Q-Learning
	 *
	 * @param s
	 * @param episode
	 */
	public void q_learning(MA_State_Variables s, int episode,
			boolean exploration, MA_Taxi_Problem tp, int
delivery,int num) {

		int passengerDelivered = 0;
		Q_Controller control = new Q_Controller();
		ArrayList<Integer> passDelivered = new
ArrayList<Integer>();

		int index = tp.findIndex(s);
		int next_index;
		MA_State_Variables tempState;// to next state
		countStep = 0;
		do {
			tempState = new MA_State_Variables(s);
			control.agentController(tp, s, tempState, agents,
passDelivered,
					index, exploration, countStep);
			// topothetounte neoi passenger stin thesi aftwn pou
metaferthikan
			// ston proorismo tous
			next_index = tp.findIndex(tempState);

			for (int i = 0; i < passDelivered.size(); i++) {
				tempState.passenger.set(passDelivered.get(i),
MA_Taxi_Problem.r

	.nextInt(MA_Gridworld.numOfStartPosition));

	tempState.destination.set(passDelivered.get(i),

	MA_Taxi_Problem.r.nextInt(MA_Gridworld.numOfGoals));
			}

			for (int i = 0; i < s.taxi.size(); i++) {
				agents.get(i).addQValue(
						index,
						agents.get(i).getAction(),
						((double) tp.getLearningRate() *
((double) agents
.get(i).getReward()
+ (double)
tp.getDiscountFactor()
*
agents.get(i).getQValue(

	next_index,

	tp.findMaxQ(next_index, agents.get(i)

	.getQ())) - agents.get(i)
```

```
                                                        .getQValue(index,
agents.get(i).getAction())));
                    }

                    s = new MA_State_Variables(tempState);
                    passengerDelivered += passDelivered.size();
                    if (num != 1) {
                          if (episode == 0)
                                tp.addEpisodeSteps(0);
                          else if ((count % num == 0)) {
                                // System.out.println("count="+count);
                                // if(count==1)
                                // tp.addEpisodeSteps(0);
                                // else
                                tp.addEpisodeSteps((episode + 1) / num);
                          }
                    } else {
                          tp.addEpisodeSteps(episode);
                    }
                    //tp.addEpisodeSteps(episode);
                    index = next_index;
                    passDelivered.clear();
                    countStep++;
              } while (passengerDelivered < delivery);
      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            MAQ_Learning ql = new MAQ_Learning();
            MA_Taxi_Problem tp = new MA_Taxi_Problem();
            // ql.calculate(100000, true, tp);
            // TODO Auto-generated method stub

      }
}
```

MA_MAXQ_Q.java

```
import java.util.ArrayList;

/**
 * klasi i opia antiprosopevi ton algorithmo MAXQ-Q gia multi agent
 *
 * @author Giannis
 *
 */
public class MA_MAXQ_Q extends MA_MaxQ{


      protected int count;
      protected int n;
      protected MA_State_Variables next_state;
```

```
ArrayList<MaxQ_Agent> agents = new ArrayList<MaxQ_Agent>();

/**
 * methodos i opoia anatheti timi sto count
 *
 * @param count
 *            the count to set
 */
public void setCount(int count) {
      this.count = count;
}

/**
 * methodos i opoia epistrefi to count
 *
 * @return the count
 */
public int getCount() {
      return count;
}

/**
 * methodos i opoia anatheti timi sto n
 *
 * @param n
 */
public void setN(int n) {
      this.n = n;
}

/**
 * methodos i opoia epistrefi to n
 *
 * @return
 */
public int getN() {
      return n;
}

/**
 * methodos i opoia prostheti 1 sto n
 *
 * @return
 */
public void addOneToN() {
      this.n++;
}

/**
 * methodos i opoia kathorizi to next state
 *
 * @param next_state
 *            the next_state to set
 */
public void setNext_state(MA_State_Variables next_state) {
      this.next_state.destination = next_state.destination;
```

```
            this.next_state.passenger = next_state.passenger;
            this.next_state.taxi = next_state.taxi;
      }

      /**
       * methodos i opoia epistrefi to next state
       *
       * @return the next_state
       */
      public MA_State_Variables getNext_state() {
            return next_state;
      }

      /**
       * methodos i opoia vriski ta Q values gia kathe children enos
       * MaxNode xrisimopoiontas to Cinside
       *
       * @param node
       * @param index
       * @return
       */
      public double[] findQvaluesWithCinside(MaxQ_Agent agent, int
maxNode,
                  int index) {
            double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
            NodeValue nValue;
            for (int i = 0; i <
agent.g.graph.get(maxNode).children.size(); i++) {
                  nValue = this.evaluateMaxNode(agent, agent.g.graph

      .get(agent.g.graph.get(maxNode).children.get(i)).children
                              .get(0), index);
                  table[i] = nValue.value;
            }

            for (int i = 0; i < table.length; i++)
                  table[i] += ((Qnode) agent.g.graph

      .get(agent.g.graph.get(maxNode).children.get(i)))
                              .getCinsideValue(index);

            return table;
      }

      /**
       * methodos i opoia vriski ta Q values gia kathe children enos
       * MaxNode xrisimopoiontas to C
       *
       * @param node
       * @param index
       * @return
       */
      public double[] findQvaluesWithC(MaxQ_Agent agent, int maxNode,
int index) {
            double[] table = new
double[this.graph.get(maxNode).children.size()];
            NodeValue nValue;
```

```
            for (int i = 0; i <
this.graph.get(maxNode).children.size(); i++) {
                  nValue = this.evaluateMaxNode(agent,
this.graph.get(this.graph

      .get(maxNode).children.get(i)).children.get(0), index);
                  table[i] = nValue.value;
            }

            for (int i = 0; i < table.length; i++)
                  table[i] += ((Qnode) this.graph

      .get(this.graph.get(maxNode).children.get(i)))
                              .getCvalue(index);

            return table;
      }

      /**
       * methodos i opoia vriski to greedy action gia ena Max node
       * xrisimopoiontas to Cinside
       *
       * @param node
       * @param index
       * @return
       */
      public int findGreedyAction(MaxQ_Agent agent, int maxNode, int
index) {
            double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
            table = this.findQvaluesWithCinside(agent, maxNode,
index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia vriski to greedy action gia ena Max node
       * xrisimopoiontas to C
       *
       * @param node
       * @param index
       * @return
       */
      public int findGreedyActionWithC(MaxQ_Agent agent, int maxNode,
int index) {
            double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
            table = this.findQvaluesWithC(agent, maxNode, index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia kani copy ta nodes tou seq sto childSeq
       */
      public void setChildSeq(ArrayList<MA_State_Variables> seq,
                  ArrayList<MA_State_Variables> childSeq) {
            childSeq.clear();
            for (int i = 0; i < seq.size(); i++)
```

```
                childSeq.add(seq.get(i));
        }

        /**
         * methodos i opia ekpedevi ton agent
         */
        public void calculate(int max_episode, int num, int length,
double epsilon,
                double df, String filename, int delivery,double lr)
{

            MA_Gridworld.makeGridworld(filename);
            this.setRanges();
            this.initOffsetTable();

            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                MaxQ_Agent tmp = new MaxQ_Agent(i, -1, 0);//
1=MaxNode
                agents.add(tmp);
            }
            double timer;
            this.setEpsilon(epsilon);
            this.setDiscountFactor(df);
            MA_State_Variables s;
            this.initLearningRate(agents,lr);
            this.makeNewEpisodeSteps(length);
            this.makeNewTimePerEpisode(length);
            count=0;
            for (int i = 0; i < max_episode; i++) {
                if ((count % num == 0))
                    System.out.println("episode=" + count);
                count++;
                timer = (double) System.currentTimeMillis() / 1000;
                // System.out.println("episode=" + i);
                s = new MA_State_Variables(MA_Gridworld.size_x,
                        MA_Gridworld.size_y,
MA_Gridworld.numOfStartPosition,
                        MA_Gridworld.numOfGoals);
                for (int j = 0; j < s.taxi.size(); j++) {
                    agents.get(j).setAgentPos(s.taxi.get(j));
                }
                this.next_state = new MA_State_Variables(s);
                // this.maxQ_PHC(this.MaxRoot, s, i);
                this.maxQ_Q(s, i, delivery, num);
                if (num != 1) {
                    if (i == 0)
                        this.setTimePerEpisode(i, ((double)
System
                                .currentTimeMillis() / 1000
- timer));
                    else if ((count % num == 0))
                        this.setTimePerEpisode((i + 1) / num,
((double) System
                                .currentTimeMillis() / 1000
- timer));
                } else {
                    this.setTimePerEpisode(i,
```

```
                                        ((double)
System.currentTimeMillis() / 1000 - timer));
                  }
                  // this.setTimePerEpisode(i,
                  // ((double) System.currentTimeMillis() / 1000 -
timer));
                  for (int j = 0; j < s.taxi.size(); j++) {
                        agents.get(j).setAgentPreviousPos(-1);
                        agents.get(j).setPassenger(-1);
                        agents.get(j).setTransfer(false);
                        agents.get(j).setTransferPrevious(false);
                        agents.get(j).setCollision(false);
                        agents.get(j).setAction(-1);
                        agents.get(j).setPrevAction(-1);
                        agents.get(j).setReward(-1);
                        agents.get(j).seqStack.clear();
                        agents.get(j).stack.clear();
                        agents.get(j).stack.push(this.MaxRoot);
                  }
            }

            // PrintInFile p = new PrintInFile();
            // p.printResults("MaxQ_PHC-results.txt",
this.getEpisodeSteps(),
            // length);
            // p.printResults("MaxQ_PHC-clock time.txt",
this.getTimePerEpisode(),
            // length);

      }

      /**
       * methodos i opoia elegxei an o node pou imaste einai Max node
i Qnode an
       * einai Qnode tote pernoume to child tou (ton maxnode pou
exteli to action)
       *
       * @param maxNode
       * @param s
       */
      public int checkInstanceOf(MaxQ_Agent agent, int maxNode,
                  MA_State_Variables s) {
            if (agent.g.graph.get(maxNode) instanceof Qnode) {
                  // kathorizoume tin parametro t gia to
MaxNavigate(t)
                  if (maxNode == this.QNavigateForGet) {
                        maxNode = ((Qnode)
agent.g.graph.get(maxNode)).children.get(0);
                        ((MaxNode)
agent.g.graph.get(maxNode)).setDest(MA_Gridworld
                                          .getPasLoc(s.passenger.get(r

      .nextInt(MA_Gridworld.numOfPassengers))));
                  } else if (maxNode == this.QNavigateForPut) {
                        maxNode = ((Qnode)
agent.g.graph.get(maxNode)).children.get(0);
                        ((MaxNode)
agent.g.graph.get(maxNode)).setDest(MA_Gridworld
```

```
      .getDestLoc(s.destination.get(agent.getPassenger())));
                  } else {
                        maxNode = ((Qnode)
agent.g.graph.get(maxNode)).children.get(0);
                  }
            }
            return maxNode;
      }

      /**
       * methodos i opoia allazi tin timi tou Completion function (C
       * kai Cinside)
       * @param seqStack
       * @param index
       * @param next_index
       * @param action
       * @param maxNode
       * @param lr
       */
      public void changeCompletionValue(MaxQ_Agent agent, int index,
                  int next_index, int action, int maxNode, double lr)
{
            this.setN(1);
            int tmp_index = 0;

            for (int i = 0; i < agent.seqStack.peek().size(); i++) {
                  tmp_index =
this.findIndex(agent.seqStack.peek().get(i));

                  // set Cinside
                  ((Qnode)
agent.g.graph.get(action)).setCinsideValue(tmp_index,
                              ((1 - lr)
                                          * ((Qnode)
agent.g.graph.get(action))

      .getCinsideValue(tmp_index) + (lr
                                          *
Math.pow(this.getDiscountFactor(), n) * (this
                                          .getPseudoreward(agent,
maxNode, this.next_state)
                                          + ((Qnode)
agent.g.graph.get(agent

      .getGreedyAction()))

      .getCinsideValue(next_index) + this
                                          .findValue(agent,
agent.getGreedyAction(),
                                                      tmp_index)))));

                  // set C value
                  ((Qnode) agent.g.graph.get(action))
                              .setCvalue(
                                          tmp_index,
                                          ((1 - lr)
```

```
                                                                  * ((Qnode)
agent.g.graph.get(action))

      .getCvalue(tmp_index) + (lr
                                                                  *
Math.pow(this.getDiscountFactor(), n) * (((Qnode) agent.g.graph

      .get(agent.getGreedyAction()))

      .getCvalue(next_index) + this.findValue(
                                                                  agent,
agent.getGreedyAction(), next_index)))));

                  this.addOneToN();
            }
            // append childSeq onto the front of seq
            if (!agent.seqStack.isEmpty())
                  // if (agent.seqStack.size()>1)
                  this.appendChildSeqOntoSeq(agent.seqStack.pop(),
agent.seqStack
                              .peek());

      }

      /**
       * methodos i opoia ilopoiei ton MAXQ-PHC algorithmo gia Multi
       * Agent (iterative)
       *
       * @param maxNode
       * @param s
       * @param episode
       * @return
       */
      public void maxQ_Q(MA_State_Variables s, int episode,
                  int delivery, int num) {

            int passengerDelivered = 0;
            MaxQ_Controller control = new MaxQ_Controller();
            ArrayList<Integer> passDelivered = new
ArrayList<Integer>();

            int index = 0;
            double lr = 0;

            int action = 0;
            int childAction = 0;
            int next_index = 0;
            for (int i = 0; i < agents.size(); i++) {
                  ArrayList<MA_State_Variables> seq = new
ArrayList<MA_State_Variables>();
                  agents.get(i).seqStack.push(seq);
            }

            do {
                  for (int i = 0; i < agents.size(); i++) {
                        agents.get(i).maxNode =
agents.get(i).stack.peek();
```

```
                        agents.get(i).maxNode =
checkInstanceOf(agents.get(i), agents
                                    .get(i).maxNode, s);
                        // i periptosi opou o maxNode den einai
primitive
                        while (!((MaxNode) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).isPrimitive()) {
                              ArrayList<MA_State_Variables> seq = new
ArrayList<MA_State_Variables>();
                              agents.get(i).seqStack.push(seq);
                              if (!(this.isTerminal(agents.get(i),
agents.get(i).maxNode,
                                          s))) {
                                    index = this.findIndex(s);
                                    // choose action
                                    childAction =
this.chooseAction(agents.get(i),agents.get(i).maxNode,index);
                                    action = ((MaxNode)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).children.get(childAction);

                                    agents.get(i).stack.push(action);
                              } else {
                                    agents.get(i).stack.pop();
                                    if (agents.get(i).stack.isEmpty())
{
                                          break;
                                    }
                                    agents.get(i).maxNode =
agents.get(i).stack.peek();

                                    // elegxos an imaste se Max node
                                    agents.get(i).maxNode =
checkInstanceOf(agents.get(i),
                                                agents.get(i).maxNode,
s);
                                    // observe next state
                                    next_index =
this.findIndex(this.next_state);
                                    lr = ((MaxNode)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).getLearningRate();

                                    agents.get(i).setGreedyAction(
                                                ((MaxNode)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).children.get(this

      .findGreedyAction(agents.get(i), agents

      .get(i).maxNode, next_index)));


      changeCompletionValue(agents.get(i), index, next_index,
```

```
                                                      action,
agents.get(i).maxNode, lr);
                                    s = new
MA_State_Variables(this.next_state);
                                    index = next_index;
                              }

                              agents.get(i).maxNode =
agents.get(i).stack.peek();
                              // elegxos an imaste se Max node
                              agents.get(i).maxNode =
checkInstanceOf(agents.get(i),
                                          agents.get(i).maxNode, s);

                        }

      agents.get(i).setPrevAction(agents.get(i).getAction());
                        agents.get(i).setAction(
                              ((MaxNode) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getAction());
                  }
                  index = this.findIndex(s);
                  control
                              .agentController(s, next_state, agents,
passDelivered,
                                          index);

                  // topothetounte neoi passenger stin thesi aftwn pou
metaferthikan
                  // ston proorismo tous
                  next_index = this.findIndex(next_state);
                  for (int i = 0; i < passDelivered.size(); i++) {
                        next_state.passenger.set(passDelivered.get(i),
r

      .nextInt(MA_Gridworld.numOfStartPosition));

      next_state.destination.set(passDelivered.get(i), r

      .nextInt(MA_Gridworld.numOfGoals));
                  }

                  for (int i = 0; i < agents.size(); i++) {
                        lr = ((MaxNode) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getLearningRate();
                        ((MaxNode)
agents.get(i).g.graph.get(agents.get(i).maxNode))
                              .setValueFunctForPrimitive(index,
(1 - lr)
                                          * ((MaxNode)
agents.get(i).g.graph

      .get(agents.get(i).maxNode))

      .getValueFunctForPrimitive(index)
```

```
                                                      + (lr *
agents.get(i).getReward()));

                        MA_State_Variables tmp = new
MA_State_Variables(s);
                        agents.get(i).seqStack.peek().add(0, tmp);
                        agents.get(i).stack.pop();

                        agents.get(i).maxNode =
agents.get(i).stack.peek();
                        // elegxos an imaste se Max node
                        agents.get(i).maxNode =
checkInstanceOf(agents.get(i), agents
                                        .get(i).maxNode, s);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNode) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getLearningRate();
                        agents.get(i).setGreedyAction(
                                        ((MaxNode) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).children.get(this

      .findGreedyAction(agents.get(i),

      agents.get(i).maxNode, next_index)));

                        changeCompletionValue(agents.get(i), index,
next_index, action,
                                        agents.get(i).maxNode, lr);
                  }

                  s = new MA_State_Variables(next_state);
                  passengerDelivered += passDelivered.size();
                  if (num != 1) {
                        if (episode == 0)
                              this.addEpisodeSteps(0);
                        else if ((count % num == 0)) {
                              this.addEpisodeSteps((episode + 1) /
num);
                        }
                  } else {
                        this.addEpisodeSteps(episode);
                  }
                  // this.addEpisodeSteps(episode);
                  index = next_index;
                  passDelivered.clear();

            } while (passengerDelivered < delivery);
      }
```

```
	/**
	 * methodos i opoia epistrefi to pseudo-reward gia ena maxnode
	 * otan vriskete se kapia katastasi
	 *
	 * @param maxnode
	 * @param s
	 * @return
	 */
	public int getPseudoreward(MaxQ_Agent agent, int maxnode,
			MA_State_Variables s) {
		if (this.isTerminal(agent, maxnode, s))
			return 0;
		else
			return -100;
	}

	/**
	 * methodos i opoia vazi to childSeq stin arxi tou seq
	 *
	 * @param childSeq
	 * @param seq
	 */
	public void appendChildSeqOntoSeq(ArrayList<MA_State_Variables>
childSeq,
			ArrayList<MA_State_Variables> seq) {
		for (int i = childSeq.size() - 1; i >= 0; i--) {
			seq.add(0, childSeq.get(i));
		}

		// for (int i = 0; i < childSeq.size(); i++)
		// seq.add(0, childSeq.get(i));
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

MA_Phc.java

```
import java.util.ArrayList;

/**
 * klasi i opoia ilopoiei ton algorithmo PHC gia to Taxi Problem
 *
 * @author Giannis
 *
 */
public class MA_Phc {

	protected ArrayList<PHC_Agent> agents = new
ArrayList<PHC_Agent>();
	private double delta; // learning rate (to delta)
	// Taxi_Problem tp = new Taxi_Problem();
	int count = 0;
	protected int max_episodes;
	int countStep;

	/*************** methodoi get,set,add ******************/

	/**
	 * methodos i opoia arxikopoiei to delta
	 *
	 * @param delta
	 */
	public void setDelta(double delta) {
		this.delta = delta;
	}

	/**
	 * methodos i opoia epistrefi to delta
	 *
	 * @return
	 */
	public double getDelta() {
		return delta;
	}

	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi (simfona me to megalo to arthro)
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue2(PHC_Agent agent, int index,
			MA_Taxi_Problem tp) {
		if (agent.getAction() != tp.findMaxQ(index, agent.getQ()))
{
			agent.addPolicyValue(index, agent.getAction(),
Math.min(agent
					.getPolicyValue(index,
agent.getAction()),
```

```
                                    (this.getDelta() / (MA_Gridworld.actions
- 1))));
            } else {
                  double sum = 0;
                  for (int i = 0; i < MA_Gridworld.actions; i++) {
                        if (i == agent.getAction())
                              continue;
                        sum += Math.min(agent.getPolicyValue(index,
i), (this
                                          .getDelta() /
(MA_Gridworld.actions - 1)));
                  }
                  agent.addPolicyValue(index, agent.getAction(), sum);
            }
      }

      /**
       * methodos i opoia kani update tin timi tis pithanotitas gia ti
       * sigkekrimeni katastasi
       *
       * @param index
       * @param action
       */
      public void updatePolicyValue(PHC_Agent agent, int index,
MA_Taxi_Problem tp) {
            if (agent.getAction() == tp.findMaxQ(index, agent.getQ()))
                  agent.addPolicyValue(index, agent.getAction(),
this.getDelta());
            else
                  agent.addPolicyValue(index, agent.getAction(),
                              ((-this.getDelta()) /
(MA_Gridworld.actions - 1)));
      }

      /**
       * methodos i opoia kanonikopoiei tis pithanottites tou
       * policyValue meta to update
       *
       * @param index
       */
      public void normalizePolicyValue(PHC_Agent agent, int index) {
            double minNegative = this.findMinNegativePolicyValue(agent
                        .getPolicyValue(), index);
            double temp = Math.abs(2 * minNegative);
            double sum = 0;
            for (int i = 0; i < MA_Gridworld.actions; i++)
                  agent.addPolicyValue(index, i, temp);
            for (int i = 0; i < MA_Gridworld.actions; i++)
                  sum += agent.getPolicyValue(index, i);
            for (int i = 0; i < MA_Gridworld.actions; i++)
                  agent.setPolicyValue(index, i,
                              (agent.getPolicyValue(index, i) / sum));

      }
```

```
	/**
	 * methodos i opoia vriski tin thesi tis megaliteris
	 * pithanotitas se ena state tou policy value
	 *
	 * @param index
	 * @return
	 */
	public int findMaxValue(double[][] policyValue, int index) {
		double temp = policyValue[index][0];
		int pos = 0;
		for (int i = 1; i < MA_Gridworld.actions; i++) {
			if (policyValue[index][i] > temp) {
				temp = policyValue[index][i];
				pos = i;
			}
		}
		return pos;
	}

	/**
	 * methodos i opoia vriski tin thesi tis mikroteris pithanotitas
	 * se ena state tou policyValue
	 *
	 * @param index
	 * @return
	 */
	public int findMinValue(double[][] policyValue, int index) {
		double temp = policyValue[index][0];
		int pos = 0;
		for (int i = 1; i < MA_Gridworld.actions; i++) {
			if (policyValue[index][i] < temp) {
				temp = policyValue[index][i];
				pos = i;
			}
		}
		return pos;
	}

	/**
	 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
	 * ena state tou policyValue
	 *
	 * @param index
	 * @return
	 */
	public double findMinNegativePolicyValue(double[][] policyValue,
int index) {
		double temp = 0;
		for (int i = 0; i < MA_Gridworld.actions; i++) {
			if (policyValue[index][i] < temp) {
				temp = policyValue[index][i];
			}
		}
		return temp;
	}
```

```
	/**
	 * methodos i opia ekpedevi ton agent
	 */
	public void calculate(int max_episode, MA_Taxi_Problem tp, int
num,
			int length, boolean changeEpsilon, boolean changeLr,
			double epsilon, double lr, double df, double delta,
			String filename, int delivery) {
		this.max_episodes = max_episode;
		MA_Gridworld.makeGridworld(filename);
		tp.setRanges();
		tp.initOffsetTable();

		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
			PHC_Agent tmp = new PHC_Agent(i, -1, 1, lr);
			agents.add(tmp);
		}
		double timer;
		if (!changeEpsilon)
			tp.setEpsilon(epsilon);
		// tp.setLearningRate(0.3);
		tp.setDiscountFactor(df);
		this.setDelta(delta);
		MA_State_Variables s;
		tp.makeNewEpisodeSteps(length);
		tp.makeNewTimePerEpisode(length);
		count = 0;
		for (int i = 0; i < max_episode; i++) {
			if (changeEpsilon)
				tp.changeEpsilon0(i, max_episode);
			if ((count % num == 0))
				System.out.println("episode=" + count);
			count++;
			timer = (double) System.currentTimeMillis() / 1000;
			// System.out.println("episode=" + i);
			s = new MA_State_Variables(MA_Gridworld.size_x,
					MA_Gridworld.size_y,
MA_Gridworld.numOfStartPosition,
					MA_Gridworld.numOfGoals);
			for (int j = 0; j < s.taxi.size(); j++) {
				agents.get(j).setAgentPos(s.taxi.get(j));
			}
			this.phc(s, i, tp, num, changeEpsilon, changeLr,
delivery);
			if (num != 1) {
				if (i == 0)
					tp.setTimePerEpisode(i, ((double) System
							.currentTimeMillis() / 1000
- timer));
				else if ((count % num == 0))
					tp.setTimePerEpisode((i + 1) / num,
((double) System
							.currentTimeMillis() / 1000
- timer));
			} else {
				tp.setTimePerEpisode(i,
						((double)
System.currentTimeMillis() / 1000 - timer));
```

```
                }
                // tp.setTimePerEpisode(i,
                // ((double) System.currentTimeMillis() / 1000 -
timer));
                for (int j = 0; j < s.taxi.size(); j++) {
                      agents.get(j).setAgentPreviousPos(-1);
                      agents.get(j).setPassenger(-1);
                      agents.get(j).setTransfer(false);
                      agents.get(j).setTransferPrevious(false);
                      agents.get(j).setCollision(false);
                      agents.get(j).setAction(-1);
                      agents.get(j).setPrevAction(-1);
                      agents.get(j).setReward(-1);
                      agents.get(j).initLr(lr);
                }
          }

          /*
           * PrintInFile p = new PrintInFile(); p.printResults("PHC
Results.txt",
           * tp.getEpisodeSteps(), max_episode);
           * p.printResults("PHC_clock time.txt",
tp.getTimePerEpisode(),
           * max_episode);
           */
    }

    /**
     * methodos i opoia ilopoiei to taxi problem simfona me ton
     * algorithmo PHC
     * @param s
     * @param episode
     */
    public void phc(MA_State_Variables s, int episode,
MA_Taxi_Problem tp,
                int num, boolean changeEpsilon, boolean changeLr,
int delivery) {

          int passengerDelivered = 0;
          PHC_Controller control = new PHC_Controller();
          ArrayList<Integer> passDelivered = new
ArrayList<Integer>();

          int index = tp.findIndex(s);
          int next_index;
          MA_State_Variables tempState;// to next state
          countStep = 0;
          do {
                tempState = new MA_State_Variables(s);
                control.agentController(tp, s, tempState, agents,
passDelivered,
                            index, countStep, changeEpsilon);
                // topothetounte neoi passenger stin thesi aftwn pou
metaferthikan
                // ston proorismo tous
                next_index = tp.findIndex(tempState);

                for (int i = 0; i < passDelivered.size(); i++) {
```

```
                        tempState.passenger.set(passDelivered.get(i),
MA_Taxi_Problem.r

      .nextInt(MA_Gridworld.numOfStartPosition));

      tempState.destination.set(passDelivered.get(i),

      MA_Taxi_Problem.r.nextInt(MA_Gridworld.numOfGoals));
                  }

                  for (int i = 0; i < s.taxi.size(); i++) {
                        agents
                                    .get(i)
                                    .setQValue(
                                                index,

      agents.get(i).getAction(),
                                                (((double) 1 -
agents.get(i).getLr(index,

      agents.get(i).getAction()))
                                                                  *
agents.get(i).getQValue(index,

      agents.get(i).getAction()) + agents

      .get(i).getLr(index,

      agents.get(i).getAction())
                                                                  *
((double) agents.get(i).getReward() + (double) tp

      .getDiscountFactor()

      * agents

                  .get(i)

                  .getQValue(

                              next_index,

                              tp

                                          .findMaxQ(

                                                      next_index,

                                                      agents


      .get(

            i)


      .getQ())))));
```

```
                        if (changeLr)
                              agents.get(i).changeLr(index,
agents.get(i).getAction(),
                                          max_episodes);

                        // update policyValue(s,a)
                        this.updatePolicyValue(agents.get(i), index,
tp);

                        this.normalizePolicyValue(agents.get(i),
index);

                  }
                  s = new MA_State_Variables(tempState);
                  passengerDelivered += passDelivered.size();
                  if (num != 1) {
                        if (episode == 0)
                              tp.addEpisodeSteps(0);
                        else if ((count % num == 0)) {
                              // System.out.println("count="+count);
                              // if(count==1)
                              // tp.addEpisodeSteps(0);
                              // else
                              tp.addEpisodeSteps((episode + 1) / num);
                        }
                  } else {
                        tp.addEpisodeSteps(episode);
                  }
                  countStep++;
                  index = next_index;
                  passDelivered.clear();

            } while (passengerDelivered < delivery);

      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            MA_Phc phc = new MA_Phc();
            // phc.calculate(20000);

      }
}
```

```
import java.util.ArrayList;

/**
 * klasi i opoia ilopoiei ton algorithmo WoLF-PHC gia to Taxi Problem
 *
 * @author Giannis
 *
 */
public class MA_Wolf extends MA_Phc {

	double t = 0;
	private double deltaW; // learning rate for winning
	private double deltaL; // learning rate for losing
	// private int max_episodes;
	protected ArrayList<WP_Agent> agents = new
ArrayList<WP_Agent>();

	/*************** methodoi get,set,add ************/

	/**
	 * methodos i opoia arxikopoiei to deltaW
	 *
	 * @param deltaW
	 */
	public void setDeltaW(double deltaW) {
		this.deltaW = deltaW;
	}

	/**
	 * methodos i opoia epistrefi to deltaW
	 *
	 * @return
	 */
	public double getDeltaW() {
		return deltaW;
	}

	/**
	 * methodos i opoia arxikopoiei to deltaL
	 *
	 * @param deltaL
	 */
	public void setDeltaL(double deltaL) {
		this.deltaL = deltaL;
	}

	/**
	 * methodos i opoia epistrefi to deltaL
	 *
	 * @return
	 */
	public double getDeltaL() {
		return deltaL;
	}
```

```
	/**
	 * update estimate of average policy p
	 *
	 * @param index
	 * @param action
	 */
	public void updateAvgPolicyValue(WP_Agent agent, int index) {
		agent.addOneToC(index);
		for (int i = 0; i < MA_Gridworld.actions; i++)
			agent.addAvgPolicyValue(index, i,
					(((double) 1 / agent.getC(index)) *
(agent.getPolicyValue(
							index, i) -
agent.getAvgPolicyValue(index, i))));
	}

	/**
	 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
	 * ena state tou avgPolicyValue
	 *
	 * @param index
	 * @return
	 */
	public double findMinNegativeAvgPolicyValue(WP_Agent agent, int
index) {
		double temp = 0;
		for (int i = 0; i < MA_Gridworld.actions; i++) {
			if (agent.getAvgPolicyValue(index, i) < temp) {
				temp = agent.getAvgPolicyValue(index, i);
			}
		}
		return temp;
	}

	/**
	 * methodos i opoia kanonikopoiei tis pithanottites tou
	 * avgPolicyValue meta to update
	 *
	 * @param index
	 */
	public void normalizeAvgPolicyValue(WP_Agent agent, int index) {
		double minNegative =
this.findMinNegativeAvgPolicyValue(agent, index);
		double temp = Math.abs(2 * minNegative);
		double sum = 0;
		for (int i = 0; i < MA_Gridworld.actions; i++)
			agent.addAvgPolicyValue(index, i, temp);
		for (int i = 0; i < MA_Gridworld.actions; i++)
			sum += agent.getAvgPolicyValue(index, i);
		for (int i = 0; i < MA_Gridworld.actions; i++)
			agent.setAvgPolicyValue(index, i, (agent
					.getAvgPolicyValue(index, i) / sum));

	}
```

```
	/**
	 * methodos i opoia vriski pio delta (winning or losing) tha
	 * xrisimopoihthei gia na ginoun update oi times tou policyValue
	 *
	 * @param index
	 * @return
	 */
	public double chooseDeltaForUpdate(WP_Agent agent, int index) {
		double sumPolicy = 0;
		double sumAvgPolicy = 0;
		for (int i = 0; i < MA_Gridworld.actions; i++) {
			sumPolicy += (agent.getPolicyValue(index, i) *
agent.getQValue(
					index, i));
			sumAvgPolicy += (agent.getAvgPolicyValue(index, i) *
agent
					.getQValue(index, i));
		}
		if (sumPolicy > sumAvgPolicy)
			return this.getDeltaW();
		else
			return this.getDeltaL();
	}

	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi (simfona me to megalo to arthro)
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue2(WP_Agent agent, int index,
MA_Taxi_Problem tp) {
		double delta = this.chooseDeltaForUpdate(agent, index);
		if (agent.getAction() != tp.findMaxQ(index, agent.getQ()))
{
			agent.addPolicyValue(index, agent.getAction(),
Math.min(agent
					.getPolicyValue(index,
agent.getAction()),
					(delta / (MA_Gridworld.actions - 1))));
		} else {
			double sum = 0;
			for (int i = 0; i < MA_Gridworld.actions; i++) {
				if (i == agent.getAction())
					continue;
				sum += Math.min(agent.getPolicyValue(index,
i),
						(delta / (MA_Gridworld.actions -
1)));
			}
			agent.addPolicyValue(index, agent.getAction(), sum);
		}
	}
```

```
	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue(WP_Agent agent, int index,
MA_Taxi_Problem tp) {

double delta = this.chooseDeltaForUpdate(agent, index);
		if (agent.getAction() == tp.findMaxQ(index, agent.getQ()))
			agent.addPolicyValue(index, agent.getAction(),
delta);
		else
			agent.addPolicyValue(index, agent.getAction(),
					((-delta) / (MA_Gridworld.actions -
1)));
	}

	/**
	 * methodos i opia ekpedevi ton agent
	 */
	public void calculate(int max_episode, MA_Taxi_Problem tp, int
num,
			int length,double epsilon, boolean changeEpsilon,
boolean changeLr, double lr,
			double df, double deltaW, double deltaL, String
filename,int delivery) {
		this.max_episodes = max_episode;
		MA_Gridworld.makeGridworld(filename);
		tp.setRanges();
		tp.initOffsetTable();

		for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
			WP_Agent tmp = new WP_Agent(i, -1, 1, lr);
			agents.add(tmp);
		}
		double timer;
		if (!changeEpsilon)
			tp.setEpsilon(epsilon);
		// tp.setLearningRate(0.9);
		tp.setDiscountFactor(df);
		this.setDeltaL(deltaL);
		this.setDeltaW(deltaW);
		//this.setDeltaL(0.2);
		//this.setDeltaW(0.05);
		// this.setDeltaL(0.4);
		// this.setDeltaW(0.1);
		MA_State_Variables s;
		tp.makeNewEpisodeSteps(length);
		tp.makeNewTimePerEpisode(length);
		count = 0;
		for (int i = 0; i < max_episode; i++) {
			if (changeEpsilon)
				tp.changeEpsilon0(i, max_episode);
			if ((count % num == 0))
				System.out.println("episode=" + count);
```

```
                        count++;
                        timer = (double) System.currentTimeMillis() / 1000;
                        // System.out.println("episode=" + i);
                        s = new MA_State_Variables(MA_Gridworld.size_x,
                                    MA_Gridworld.size_y,
MA_Gridworld.numOfStartPosition,
                                    MA_Gridworld.numOfGoals);
                        for (int j = 0; j < s.taxi.size(); j++) {
                              agents.get(j).setAgentPos(s.taxi.get(j));
                        }
                        this.wolf(s, i, tp, num, changeEpsilon,
changeLr,delivery);
                        if (num != 1) {
                              if (i == 0)
                                    tp.setTimePerEpisode(i, ((double) System
                                                .currentTimeMillis() / 1000
- timer));
                              else if ((count % num == 0))
                                    tp.setTimePerEpisode((i + 1) / num,
((double) System
                                                .currentTimeMillis() / 1000
- timer));
                        } else {
                              tp.setTimePerEpisode(i,
                                          ((double)
System.currentTimeMillis() / 1000 - timer));
                        }
                        // tp.setTimePerEpisode(i,
                        // ((double) System.currentTimeMillis() / 1000 -
timer));
                        for (int j = 0; j < s.taxi.size(); j++) {
                              agents.get(j).setAgentPreviousPos(-1);
                              agents.get(j).setPassenger(-1);
                              agents.get(j).setTransfer(false);
                              agents.get(j).setTransferPrevious(false);
                              agents.get(j).setCollision(false);
                              agents.get(j).setAction(-1);
                              agents.get(j).setPrevAction(-1);
                              agents.get(j).setReward(-1);
                              agents.get(j).initLr(lr);
                        }
                  }

                  /*
                   * PrintInFile p = new PrintInFile();
                   * p.printResults("WoLF-PHC Results.txt",
tp.getEpisodeSteps(),
                   * max_episode); p.printResults("WoLF-PHC_clock time.txt",
                   * tp.getTimePerEpisode(), max_episode);
                   */
            }

            /**
             * methodos i opoia ilopoiei to taxi problem simfona me ton
             * algorithmo WoLF-PHC
             *
             * @param s
             * @param episode
```

```
       */
      public void wolf(MA_State_Variables s, int episode,
MA_Taxi_Problem tp,
                  int num, boolean changeEpsilon, boolean changeLr,int
delivery) {

            int passengerDelivered = 0;
            WP_Controller control = new WP_Controller();
            ArrayList<Integer> passDelivered = new
ArrayList<Integer>();

            int index = tp.findIndex(s);
            int next_index;
            MA_State_Variables tempState;// to next state

            countStep = 0;
            do {
                  // t=(double) System.currentTimeMillis() / 3600000;
                  // t=t-RunTrials.time;

                  // System.out.println("t="+t);
                  // if(t>=1)
                  // System.out.println("t="+t);
                  tempState = new MA_State_Variables(s);
                  control.agentController(tp, s, tempState, agents,
passDelivered,
                              index, countStep, changeEpsilon);

                  // topothetounte neoi passenger stin thesi aftwn pou
metaferthikan
                  // ston proorismo tous
                  next_index = tp.findIndex(tempState);

                  for (int i = 0; i < passDelivered.size(); i++) {
                        tempState.passenger.set(passDelivered.get(i),
MA_Taxi_Problem.r

      .nextInt(MA_Gridworld.numOfStartPosition));

      tempState.destination.set(passDelivered.get(i),

      MA_Taxi_Problem.r.nextInt(MA_Gridworld.numOfGoals));
                  }

                  for (int i = 0; i < s.taxi.size(); i++) {



                        agents
                                    .get(i)
                                    .setQValue(
                                                index,

      agents.get(i).getAction(),
                                                (((double) 1 -
agents.get(i).getLr(index,

      agents.get(i).getAction()))
```

```
                                                                  *
agents.get(i).getQValue(index,
      agents.get(i).getAction()) + agents
      .get(i).getLr(index,
      agents.get(i).getAction())
                                                                  *
((double) agents.get(i).getReward() + (double) tp
      .getDiscountFactor()
      * agents
                  .get(i)
                  .getQValue(
                              next_index,
                              tp
                                          .findMaxQ(
                                                      next_index,
                                                      agents

      .get(

            i)

      .getQ())))));
                        // System.out.println("prin
lr="+agents.get(i).getLr(index,
                        // agents.get(i).getAction()));
                        if (changeLr)
                              agents.get(i).changeLr(index,
agents.get(i).getAction(),
                                          max_episodes);
                        // System.out.println("meta
lr="+agents.get(i).getLr(index,
                        // agents.get(i).getAction()));
                        // update avgPolicyValue(s,a)
                        this.updateAvgPolicyValue(agents.get(i),
index);
                        this.normalizeAvgPolicyValue(agents.get(i),
index);

                        // update policyValue(s,a)
                        this.updatePolicyValue(agents.get(i), index,
tp);
```

```
                        this.normalizePolicyValue(agents.get(i),
index);

                  }
                  s = new MA_State_Variables(tempState);
                  passengerDelivered += passDelivered.size();

                  if (num != 1) {
                        if (episode == 0)
                              tp.addEpisodeSteps(0);
                        else if ((count % num == 0)) {
                              // System.out.println("count="+count);
                              // if(count==1)
                              // tp.addEpisodeSteps(0);
                              // else
                              tp.addEpisodeSteps((episode + 1) / num);
                        }
                  } else {
                        tp.addEpisodeSteps(episode);
                  }
                  countStep++;
                  // tp.addEpisodeSteps(episode);
                  index = next_index;
                  passDelivered.clear();

            } while (passengerDelivered < delivery);

      }
      /**
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            MA_Wolf wolf = new MA_Wolf();
            MA_Taxi_Problem tp = new MA_Taxi_Problem();
            int length = 2;
            // wolf.calculate(1, tp, 1, length);
      }
}
```

MA_MaxQ_PHC.java

```
import java.util.ArrayList;
import java.util.Stack;

/**
 * klasi i opoia ilopoiei ton algorithmo MAXQ-PHC
 *
 * @author Giannis
 *
 */
public class MA_MaxQ_PHC extends MA_MaxQ {

      protected int count;
      protected int n;
```

```
protected MA_State_Variables next_state;
// protected int greedyAction;

private double delta; // learning rate (to delta)

ArrayList<MaxQ_Agent> agents = new ArrayList<MaxQ_Agent>();

/****************** methodoi set,get,add ********************/
/**
 * methodos i opoia arxikopoiei to delta
 *
 * @param delta
 */
public void setDelta(double delta) {
	this.delta = delta;
}

/**
 * methodos i opoia epistrefi to delta
 *
 * @return
 */
public double getDelta() {
	return delta;
}

/**
 * methodos i opoia anatheti timi sto count
 *
 * @param count
 *            the count to set
 */
public void setCount(int count) {
	this.count = count;
}

/**
 * methodos i opoia epistrefi to count
 *
 * @return the count
 */
public int getCount() {
	return count;
}

/**
 * methodos i opoia anatheti timi sto n
 *
 * @param n
 */
public void setN(int n) {
	this.n = n;
}

/**
 * methodos i opoia epistrefi to n
 *
 * @return
```

```
	 */
	public int getN() {
		return n;
	}

	/**
	 * methodos i opoia prostheti 1 sto n
	 *
	 * @return
	 */
	public void addOneToN() {
		this.n++;
	}

	/**
	 * methodos i opoia kathorizi to next state
	 *
	 * @param next_state
	 *            the next_state to set
	 */
	public void setNext_state(MA_State_Variables next_state) {
		this.next_state.destination = next_state.destination;
		this.next_state.passenger = next_state.passenger;
		this.next_state.taxi = next_state.taxi;
	}

	/**
	 * methodos i opoia epistrefi to next state
	 *
	 * @return the next_state
	 */
	public MA_State_Variables getNext_state() {
		return next_state;
	}

	/********************** PHC ***************************/

	/**
	 * methodos i opoia epilegi tin epomeni kinisi (action) apo tin
sigkekrimeni
	 * katastasi (state) me pithanotita p(s,a) kai me kapia
anixnefsi
	 *
	 * @param index
	 * @return
	 */
	public int chooseAction(int index, double[][] policyValue) {
		if (r.nextDouble() < this.getEpsilon())
			return (r.nextInt(policyValue[index].length));
		return this.chooseProbAction(index, policyValue);
	}
```

```
	/**
	 * methodos i opoia epilegi mia kinisi (action) vasi tis
	 * pithanotitas p(s,a)
	 * @param index
	 * @return
	 */
	public int chooseProbAction(int index, double[][] policyValue) {
		double offset = 0;
		double rand = r.nextDouble();
		for (int i = 0; i < policyValue[index].length; i++) {
			offset += policyValue[index][i];
			if (offset >= rand)
				return i;
		}
		System.out.println("rand=" + rand);
		System.out.println("offset=" + offset);
		return 5; // en tha erti pote dame

	}

	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue2(MaxQ_Agent agent, int index, int
action,
			int maxNode) {
		int length = ((MaxNodePHC) agent.g.graph.get(maxNode))
				.getPolicyValueLength();

		if (action != this.findGreedyAction(agent, maxNode,
index))
			((MaxNodePHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
					action, Math.min(((MaxNodePHC)
agent.g.graph.get(maxNode))
							.getPolicyValue(index,
action),
							(this.getDelta() / (length -
1))));
		else {
			double sum = 0;
			for (int i = 0; i < length; i++) {
				if (i == action)
					continue;
				sum += Math.min(((MaxNodePHC)
agent.g.graph.get(maxNode))
						.getPolicyValue(index, i),
						(this.getDelta() / (length - 1)));
			}
			((MaxNodePHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
					action, sum);
		}
	}
```

```
	/**
	 * methodos i opoia kani update tin timi tis pithanotitas gia ti
	 * sigkekrimeni katastasi
	 *
	 * @param index
	 * @param action
	 */
	public void updatePolicyValue(MaxQ_Agent agent, int index, int
action,
			int maxNode) {
		// System.out.println("index="+index);
		// System.out.println("action="+action);
		//
System.out.println("this.findMaxQ(index)="+this.findMaxQ(index));
		if (action == this.findGreedyAction(agent, maxNode,
index))
			((MaxNodePHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
					action, this.getDelta());
		else
			((MaxNodePHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
					action, ((-this.getDelta()) /
(((MaxNodePHC) agent.g.graph

	.get(maxNode)).getPolicyValueLength() - 1)));

		// System.out.println("this.getDelta()="+this.getDelta());
		// System.out.println("-this.getDelta() / (this.actions -
1)="+(-this.getDelta()
		// / (this.actions - 1)));

	}

	/**
	 * methodos i opoia kanonikopoiei tis pithanottites tou
policyValue meta to
	 * update
	 *
	 * @param index
	 */
	public void normalizePolicyValue(MaxQ_Agent agent, int index,
int maxNode) {
		int length = ((MaxNodePHC) agent.g.graph.get(maxNode))
				.getPolicyValueLength();
		double minNegative =
this.findMinNegativePolicyValue(agent, index,
				maxNode);
		double temp = Math.abs(2 * minNegative);
		double sum = 0;
		for (int i = 0; i < length; i++)
			((MaxNodePHC)
agent.g.graph.get(maxNode)).addPolicyValue(index, i,
					temp);
		for (int i = 0; i < length; i++)
			sum += ((MaxNodePHC)
agent.g.graph.get(maxNode)).getPolicyValue(
```

```
                                    index, i);
                  for (int i = 0; i < length; i++)
                        ((MaxNodePHC)
agent.g.graph.get(maxNode)).setPolicyValue(index, i,
                                    (((MaxNodePHC)
agent.g.graph.get(maxNode)).getPolicyValue(
                                                index, i) / sum));

            }

            /**
             * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
             * ena state tou policyValue
             *
             * @param index
             * @return
             */
            public double findMinNegativePolicyValue(MaxQ_Agent agent, int
index,
                        int maxNode) {
                  double temp = 0;
                  int length = ((MaxNodePHC) agent.g.graph.get(maxNode))
                              .getPolicyValueLength();
                  for (int i = 0; i < length; i++) {
                        if (((MaxNodePHC)
agent.g.graph.get(maxNode)).getPolicyValue(index,
                                    i) < temp) {
                              temp = ((MaxNodePHC)
agent.g.graph.get(maxNode))
                                          .getPolicyValue(index, i);
                        }
                  }
                  return temp;
            }

            /**
             * methodos i opoia dimiourgei k arxikopoiei tous pinakes
             * policyValues se olous tous komvous tou grafou enos agent
             */
            public void makePolicyValues(MaxQ_Agent agent) {
                  for (int i = 0; i < this.numMaxNodes; i++) {
                        ((MaxNodePHC)
agent.g.graph.get(i)).makeNewPolicyValue(
                                    MA_Gridworld.states,
agent.g.graph.get(i).children.size());
                        ((MaxNodePHC) agent.g.graph.get(i)).initPolicyValue(
                                    MA_Gridworld.states,
agent.g.graph.get(i).children.size());
                  }
            }
```

```
	/**
	 * methodos i opoia vriski ta Q values gia kathe children enos
	 * MaxNode xrisimopoiontas to Cinside
	 *
	 * @param node
	 * @param index
	 * @return
	 */
	public double[] findQvaluesWithCinside(MaxQ_Agent agent, int
maxNode,
			int index) {
		double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
		NodeValue nValue;
		for (int i = 0; i <
agent.g.graph.get(maxNode).children.size(); i++) {
			nValue = this.evaluateMaxNode(agent, agent.g.graph

	.get(agent.g.graph.get(maxNode).children.get(i)).children
					.get(0), index);
			table[i] = nValue.value;
		}

		for (int i = 0; i < table.length; i++)
			table[i] += ((Qnode) agent.g.graph

	.get(agent.g.graph.get(maxNode).children.get(i)))
					.getCinsideValue(index);

		return table;
	}

	/**
	 * methodos i opoia vriski ta Q values gia kathe children enos
MaxNode
	 * xrisimopoiontas to C
	 *
	 * @param node
	 * @param index
	 * @return
	 */
	public double[] findQvaluesWithC(MaxQ_Agent agent, int maxNode,
int index) {
		double[] table = new
double[this.graph.get(maxNode).children.size()];
		NodeValue nValue;
		for (int i = 0; i <
this.graph.get(maxNode).children.size(); i++) {
			nValue = this.evaluateMaxNode(agent,
this.graph.get(this.graph

	.get(maxNode).children.get(i)).children.get(0), index);
			table[i] = nValue.value;
		}

		for (int i = 0; i < table.length; i++)
			table[i] += ((Qnode) this.graph
```

```
      .get(this.graph.get(maxNode).children.get(i)))
                              .getCvalue(index);

            return table;
      }

      /**
       * methodos i opoia vriski to greedy action gia ena Max node
       * xrisimopoiontas to Cinside
       *
       * @param node
       * @param index
       * @return
       */
      public int findGreedyAction(MaxQ_Agent agent, int maxNode, int
index) {
            double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
            table = this.findQvaluesWithCinside(agent, maxNode,
index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia vriski to greedy action gia ena Max node
       * xrisimopoiontas to C
       *
       * @param node
       * @param index
       * @return
       */
      public int findGreedyActionWithC(MaxQ_Agent agent, int maxNode,
int index) {
            double[] table = new
double[agent.g.graph.get(maxNode).children.size()];
            table = this.findQvaluesWithC(agent, maxNode, index);
            return this.findMax(table);
      }

      /**
       * methodos i opoia kani copy ta nodes tou seq sto childSeq
       */
      public void setChildSeq(ArrayList<MA_State_Variables> seq,
                  ArrayList<MA_State_Variables> childSeq) {
            childSeq.clear();
            for (int i = 0; i < seq.size(); i++)
                  childSeq.add(seq.get(i));
      }

      /**
       * methodos i opia ekpedevi ton agent
       */
      public void calculate(int max_episode, int num, int length,
double epsilon,
                  double df, double delta, String filename, int
delivery,double lr) {
```

```
            MA_Gridworld.makeGridworld(filename);
            this.setRanges();
            this.initOffsetTable();

            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                  MaxQ_Agent tmp = new MaxQ_Agent(i, -1, 1);//
1=MaxNodePHC
                  this.makePolicyValues(tmp);
                  agents.add(tmp);
            }
            double timer;
            this.setEpsilon(epsilon);
            this.setDiscountFactor(df);
            this.setDelta(delta);
            MA_State_Variables s;
            this.initLearningRate(agents,lr);
            this.makeNewEpisodeSteps(length);
            this.makeNewTimePerEpisode(length);
            count=0;
            for (int i = 0; i < max_episode; i++) {
                  if ((count % num == 0))
                        System.out.println("episode=" + count);
                  count++;
                  timer = (double) System.currentTimeMillis() / 1000;
                  // System.out.println("episode=" + i);
                  s = new MA_State_Variables(MA_Gridworld.size_x,
                              MA_Gridworld.size_y,
MA_Gridworld.numOfStartPosition,
                              MA_Gridworld.numOfGoals);
                  for (int j = 0; j < s.taxi.size(); j++) {
                        agents.get(j).setAgentPos(s.taxi.get(j));
                  }
                  this.next_state = new MA_State_Variables(s);
                  // this.maxQ_PHC(this.MaxRoot, s, i);
                  this.maxQ_PHC(s, i, delivery, num);
                  if (num != 1) {
                        if (i == 0)
                              this.setTimePerEpisode(i, ((double)
System
                                          .currentTimeMillis() / 1000
- timer));
                        else if ((count % num == 0))
                              this.setTimePerEpisode((i + 1) / num,
((double) System
                                          .currentTimeMillis() / 1000
- timer));
                  } else {
                        this.setTimePerEpisode(i,
                                    ((double)
System.currentTimeMillis() / 1000 - timer));
                  }
                  // this.setTimePerEpisode(i,
                  // ((double) System.currentTimeMillis() / 1000 -
timer));
                  for (int j = 0; j < s.taxi.size(); j++) {
                        agents.get(j).setAgentPreviousPos(-1);
                        agents.get(j).setPassenger(-1);
                        agents.get(j).setTransfer(false);
```

```
                        agents.get(j).setTransferPrevious(false);
                        agents.get(j).setCollision(false);
                        agents.get(j).setAction(-1);
                        agents.get(j).setPrevAction(-1);
                        agents.get(j).setReward(-1);
                        agents.get(j).seqStack.clear();
                        agents.get(j).stack.clear();
                        agents.get(j).stack.push(this.MaxRoot);
                  }
            }

            // PrintInFile p = new PrintInFile();
            // p.printResults("MaxQ_PHC-results.txt",
this.getEpisodeSteps(),
            // length);
            // p.printResults("MaxQ_PHC-clock time.txt",
this.getTimePerEpisode(),
            // length);

      }

      /**
       * methodos i opoia elegxei an o node pou imaste einai Max node
i Qnode an
       * einai Qnode tote pernoume to child tou (ton maxnode pou
exteli to action)
       *
       * @param maxNode
       * @param s
       */
      public int checkInstanceOf(MaxQ_Agent agent, int maxNode,
                  MA_State_Variables s) {
            if (agent.g.graph.get(maxNode) instanceof Qnode) {
                  // kathorizoume tin parametro t gia to
MaxNavigate(t)
                  if (maxNode == this.QNavigateForGet) {
                        maxNode = ((Qnode)
agent.g.graph.get(maxNode)).children.get(0);
                        ((MaxNodePHC)
agent.g.graph.get(maxNode)).setDest(MA_Gridworld
                                    .getPasLoc(s.passenger.get(r

      .nextInt(MA_Gridworld.numOfPassengers))));
                  } else if (maxNode == this.QNavigateForPut) {
                        maxNode = ((Qnode)
agent.g.graph.get(maxNode)).children.get(0);
                        ((MaxNodePHC)
agent.g.graph.get(maxNode)).setDest(MA_Gridworld

      .getDestLoc(s.destination.get(agent.getPassenger())));
                  } else {
                        maxNode = ((Qnode)
agent.g.graph.get(maxNode)).children.get(0);
                  }
            }
            return maxNode;
      }
```

```
	/**
	 * methodos i opoia allazi tin timi tou Completion function (C
	 * kai Cinside)
	 * @param seqStack
	 * @param index
	 * @param next_index
	 * @param action
	 * @param maxNode
	 * @param lr
	 */
	public void changeCompletionValue(MaxQ_Agent agent, int index,
			int next_index, int action, int maxNode, double lr)
{
		this.setN(1);
		int tmp_index = 0;

		for (int i = 0; i < agent.seqStack.peek().size(); i++) {
			tmp_index =
this.findIndex(agent.seqStack.peek().get(i));

			// set Cinside
			((Qnode)
agent.g.graph.get(action)).setCinsideValue(tmp_index,
					((1 - lr)
							* ((Qnode)
agent.g.graph.get(action))

	.getCinsideValue(tmp_index) + (lr
							*
Math.pow(this.getDiscountFactor(), n) * (this
							.getPseudoreward(agent,
maxNode, this.next_state)
							+ ((Qnode)
agent.g.graph.get(agent

	.getGreedyAction()))

	.getCinsideValue(next_index) + this
							.findValue(agent,
agent.getGreedyAction(),
									tmp_index)))));

			// set C value
			((Qnode) agent.g.graph.get(action))
					.setCvalue(
							tmp_index,
							((1 - lr)
									* ((Qnode)
agent.g.graph.get(action))

	.getCvalue(tmp_index) + (lr
									*
Math.pow(this.getDiscountFactor(), n) * (((Qnode) agent.g.graph

	.get(agent.getGreedyAction()))

	.getCvalue(next_index) + this.findValue(
```

```
                                                              agent,
agent.getGreedyAction(), next_index)))));

                  this.addOneToN();
            }
            // append childSeq onto the front of seq
            if (!agent.seqStack.isEmpty())
                  // if (agent.seqStack.size()>1)
                  this.appendChildSeqOntoSeq(agent.seqStack.pop(),
agent.seqStack
                              .peek());

      }

      /**
       * methodos i opoia ilopoiei ton MAXQ-PHC algorithmo gia Multi
Agent
       * (iterative)
       *
       * @param maxNode
       * @param s
       * @param episode
       * @return
       */
      public void maxQ_PHC(MA_State_Variables s, int episode,
                  int delivery, int num) {

            int passengerDelivered = 0;
            MaxQ_Controller control = new MaxQ_Controller();
            ArrayList<Integer> passDelivered = new
ArrayList<Integer>();

            int index = 0;
            double lr = 0;

            int action = 0;
            int childAction = 0;
            int next_index = 0;
            for (int i = 0; i < agents.size(); i++) {
                  ArrayList<MA_State_Variables> seq = new
ArrayList<MA_State_Variables>();
                  agents.get(i).seqStack.push(seq);
            }

            do {
                  for (int i = 0; i < agents.size(); i++) {
                        agents.get(i).maxNode =
agents.get(i).stack.peek();
                        agents.get(i).maxNode =
checkInstanceOf(agents.get(i), agents
                                    .get(i).maxNode, s);
                        // i periptosi opou o maxNode den einai
primitive
                        while (!((MaxNodePHC) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).isPrimitive()) {
                              ArrayList<MA_State_Variables> seq = new
ArrayList<MA_State_Variables>();
```

```
                                    agents.get(i).seqStack.push(seq);
                                    if (!(this.isTerminal(agents.get(i),
agents.get(i).maxNode,
                                                s))) {
                                          index = this.findIndex(s);
                                          // choose action
                                          childAction =
this.chooseAction(index,
                                                      ((MaxNodePHC)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).getPolicyValue());
                                          action = ((MaxNodePHC)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).children.get(childAction);


      this.updatePolicyValue(agents.get(i), index,
                                                      childAction,
agents.get(i).maxNode);

      this.normalizePolicyValue(agents.get(i), index, agents
                                                      .get(i).maxNode);
                                          agents.get(i).stack.push(action);
                                    } else {
                                          agents.get(i).stack.pop();
                                          if (agents.get(i).stack.isEmpty())
{
                                                break;
                                          }
                                          agents.get(i).maxNode =
agents.get(i).stack.peek();

                                          // elegxos an imaste se Max node
                                          agents.get(i).maxNode =
checkInstanceOf(agents.get(i),
                                                      agents.get(i).maxNode,
s);
                                          // observe next state
                                          next_index =
this.findIndex(this.next_state);
                                          lr = ((MaxNodePHC)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).getLearningRate();

                                          agents.get(i).setGreedyAction(
                                                      ((MaxNodePHC)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).children.get(this

      .findGreedyAction(agents.get(i), agents

      .get(i).maxNode, next_index)));
```

```
      changeCompletionValue(agents.get(i), index, next_index,
                                                action,
agents.get(i).maxNode, lr);
                                    s = new
MA_State_Variables(this.next_state);
                                    index = next_index;
                              }

                              agents.get(i).maxNode =
agents.get(i).stack.peek();
                              // elegxos an imaste se Max node
                              agents.get(i).maxNode =
checkInstanceOf(agents.get(i),
                                          agents.get(i).maxNode, s);

                        }

      agents.get(i).setPrevAction(agents.get(i).getAction());
                        agents.get(i).setAction(
                                    ((MaxNodePHC)
agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getAction());
                  }
                  index = this.findIndex(s);
                  control
                              .agentController(s, next_state, agents,
passDelivered,
                                          index);

                  // topothetounte neoi passenger stin thesi aftwn pou
metaferthikan
                  // ston proorismo tous
                  next_index = this.findIndex(next_state);
                  for (int i = 0; i < passDelivered.size(); i++) {
                        next_state.passenger.set(passDelivered.get(i),
r

      .nextInt(MA_Gridworld.numOfStartPosition));

      next_state.destination.set(passDelivered.get(i), r

      .nextInt(MA_Gridworld.numOfGoals));
                  }

                  for (int i = 0; i < agents.size(); i++) {
                        lr = ((MaxNodePHC) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getLearningRate();
                        ((MaxNodePHC)
agents.get(i).g.graph.get(agents.get(i).maxNode))
                                    .setValueFunctForPrimitive(index,
(1 - lr)
                                                * ((MaxNodePHC)
agents.get(i).g.graph

      .get(agents.get(i).maxNode))
```

```
      .getValueFunctForPrimitive(index)
                                                  + (lr *
agents.get(i).getReward()));

                        MA_State_Variables tmp = new
MA_State_Variables(s);
                        agents.get(i).seqStack.peek().add(0, tmp);
                        agents.get(i).stack.pop();

                        agents.get(i).maxNode =
agents.get(i).stack.peek();
                        // elegxos an imaste se Max node
                        agents.get(i).maxNode =
checkInstanceOf(agents.get(i), agents
                                    .get(i).maxNode, s);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNodePHC) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getLearningRate();
                        agents.get(i).setGreedyAction(
                                    ((MaxNodePHC)
agents.get(i).g.graph

      .get(agents.get(i).maxNode)).children.get(this

      .findGreedyAction(agents.get(i),

      agents.get(i).maxNode, next_index)));

                        changeCompletionValue(agents.get(i), index,
next_index, action,
                                    agents.get(i).maxNode, lr);
                  }

                  s = new MA_State_Variables(next_state);
                  passengerDelivered += passDelivered.size();
                  if (num != 1) {
                        if (episode == 0)
                              this.addEpisodeSteps(0);
                        else if ((count % num == 0)) {
                              this.addEpisodeSteps((episode + 1) /
num);
                        }
                  } else {
                        this.addEpisodeSteps(episode);
                  }
                  // this.addEpisodeSteps(episode);
                  index = next_index;
                  passDelivered.clear();

            } while (passengerDelivered < delivery);
      }
```

```
	/**
	 * methodos i opoia epistrefi to pseudo-reward gia ena maxnode
	 * otan vriskete se kapia katastasi
	 *
	 * @param maxnode
	 * @param s
	 * @return
	 */
	public int getPseudoreward(MaxQ_Agent agent, int maxnode,
			MA_State_Variables s) {
		if (this.isTerminal(agent, maxnode, s))
			return 0;
		else
			return -100;
	}

	/**
	 * methodos i opoia vazi to childSeq stin arxi tou seq
	 *
	 * @param childSeq
	 * @param seq
	 */
	public void appendChildSeqOntoSeq(ArrayList<MA_State_Variables>
childSeq,
			ArrayList<MA_State_Variables> seq) {
		for (int i = childSeq.size() - 1; i >= 0; i--) {
			seq.add(0, childSeq.get(i));
			if (i > 0)
				System.out.println("apoel i=" + i);
		}

		// for (int i = 0; i < childSeq.size(); i++)
		// seq.add(0, childSeq.get(i));
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MA_MaxQ_PHC maxq = new MA_MaxQ_PHC();
		// maxq.calculate(500);
	}
}
```

```java
import java.util.ArrayList;
import java.util.Stack;

/**
 * klasi i opoia ilopoiei ton algorithmo MAXQ-WoLF-PHC
 *
 * @author giannis
 *
 */
public class MA_MaxQ_WoLF_PHC extends MA_MaxQ_PHC {

	private double deltaW; // learning rate for winning
	private double deltaL; // learning rate for losing

	/***************** methodoi set get **********************/

	/**
	 * methodos i opoia arxikopoiei to deltaW
	 *
	 * @param deltaW
	 */
	public void setDeltaW(double deltaW) {
		this.deltaW = deltaW;
	}

	/**
	 * methodos i opoia epistrefi to deltaW
	 *
	 * @return
	 */
	public double getDeltaW() {
		return deltaW;
	}

	/**
	 * methodos i opoia arxikopoiei to deltaL
	 *
	 * @param deltaL
	 */
	public void setDeltaL(double deltaL) {
		this.deltaL = deltaL;
	}

	/**
	 * methodos i opoia epistrefi to deltaL
	 *
	 * @return
	 */
	public double getDeltaL() {
		return deltaL;
	}
```

```
	/**
	 * update estimate of average policy p
	 *
	 * @param index
	 * @param action
	 */
	public void updateAvgPolicyValue(MaxQ_Agent agent, int index,
int action,
			int maxNode) {
		int length = ((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
				.getAvgPolicyValueLength();
		((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode)).addOneToC(index);
		for (int i = 0; i < length; i++)
			((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
					.addAvgPolicyValue(
							index,
							i,
							(((double) 1 /
((MaxNodeWoLFPHC) agent.g.graph

	.get(maxNode)).getC(index)) * (((MaxNodeWoLFPHC) agent.g.graph

	.get(maxNode)).getPolicyValue(index, i) - ((MaxNodeWoLFPHC)
agent.g.graph

	.get(maxNode)).getAvgPolicyValue(index, i))));
	}

	/**
	 * methodos i opoia vriski tin mikroteri arnitiki pithanotita se
ena state
	 * tou avgPolicyValue
	 *
	 * @param index
	 * @return
	 */
	public double findMinNegativeAvgPolicyValue(MaxQ_Agent agent,
int index,
			int maxNode) {
		int length = ((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
				.getAvgPolicyValueLength();
		double temp = 0;
		for (int i = 0; i < length; i++) {
			if (((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
					.getAvgPolicyValue(index, i) < temp) {
				temp = ((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode))
						.getAvgPolicyValue(index, i);
			}
		}
		return temp;
	}
```

```
	/**
	 * methodos i opoia kanonikopoiei tis pithanottites tou
	 * avgPolicyValue meta to update
	 *
	 * @param index
	 */
	public void normalizeAvgPolicyValue(MaxQ_Agent agent, int index,
int maxNode) {
		int length = ((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
				.getAvgPolicyValueLength();
		double minNegative =
this.findMinNegativeAvgPolicyValue(agent, index,
				maxNode);
		double temp = Math.abs(2 * minNegative);
		double sum = 0;
		for (int i = 0; i < length; i++)
			((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode)).addAvgPolicyValue(
					index, i, temp);
		for (int i = 0; i < length; i++)
			sum += ((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
					.getAvgPolicyValue(index, i);
		for (int i = 0; i < length; i++)
			((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode)).setAvgPolicyValue(
					index, i, (((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode))
							.getAvgPolicyValue(index, i)
/ sum));

	}

	/**
	 * methodos i opoia vriski pio delta (winning or losing) tha
	 * xrisimopoihthei gia na ginoun update oi times tou policyValue
	 *
	 * @param index
	 * @return
	 */
	public double chooseDeltaForUpdate(MaxQ_Agent agent, int index,
int maxNode) {
		int length = ((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
				.getPolicyValueLength();
		double sumPolicy = 0;
		double sumAvgPolicy = 0;
		for (int i = 0; i < length; i++) {
			sumPolicy += (((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode))
					.getPolicyValue(index, i) *
this.findValue(agent, maxNode,
					index));
			sumAvgPolicy += (((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode))
					.getAvgPolicyValue(index, i) *
this.findValue(agent,
					maxNode, index));
			;
		}
```

```
            if (sumPolicy > sumAvgPolicy)
                  return this.getDeltaW();
            else
                  return this.getDeltaL();
      }

      /**
       * methodos i opoia kani update tin timi tis pithanotitas gia ti
       * sigkekrimeni katastasi (simfona me to megalo to arthro)
       *
       * @param index
       * @param action
       */
      public void updatePolicyValue2(MaxQ_Agent agent, int index, int
action,
                  int maxNode) {
            int length = ((MaxNodeWoLFPHC) agent.g.graph.get(maxNode))
                        .getPolicyValueLength();
            double delta = this.chooseDeltaForUpdate(agent, index,
maxNode);
            if (action != this.findGreedyAction(agent, maxNode,
index)) {
                  ((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
                              action, Math.min(((MaxNodeWoLFPHC)
agent.g.graph

      .get(maxNode)).getPolicyValue(index, action),
                                          (delta / (length - 1))));
            } else {
                  double sum = 0;
                  for (int i = 0; i < length; i++) {
                        if (i == action)
                              continue;
                        sum += Math.min(((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode))
                                    .getPolicyValue(index, i), (delta
/ (length - 1)));
                  }
                  ((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
                              action, sum);
            }
      }

      /**
       * methodos i opoia kani update tin timi tis pithanotitas gia ti
       * sigkekrimeni katastasi
       *
       * @param index
       * @param action
       */
      public void updatePolicyValue(MaxQ_Agent agent, int index, int
action,
                  int maxNode) {
            double delta = this.chooseDeltaForUpdate(agent, index,
maxNode);
```

```
            if (action == this.findGreedyAction(agent, maxNode,
index))
                  ((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
                              action, delta);
            else
                  ((MaxNodeWoLFPHC)
agent.g.graph.get(maxNode)).addPolicyValue(index,
                              action, ((-delta) / (((MaxNodeWoLFPHC)
agent.g.graph

      .get(maxNode)).getPolicyValueLength() - 1)));
      }

      /**
       * methodos i opoia dimiourgei k arxikopoiei tous pinakes
avgPolicyValues se
       * olous tous komvous tou grafou
       */
      public void makeAvgPolicyValues(MaxQ_Agent agent) {
            for (int i = 0; i < this.numMaxNodes; i++) {
                  ((MaxNodeWoLFPHC)
agent.g.graph.get(i)).makeNewAvgPolicyValue(
                              MA_Gridworld.states,
agent.g.graph.get(i).children.size());
                  ((MaxNodeWoLFPHC)
agent.g.graph.get(i)).initAvgPolicyValue(
                              MA_Gridworld.states,
agent.g.graph.get(i).children.size());
            }
      }

      /**
       * methodos i opoia dimiourgei k arxikopoiei tous pinakes C
       * olous tous komvous tou grafou
       */
      public void makeC(MaxQ_Agent agent, int num) {
            for (int i = 0; i < this.numMaxNodes; i++) {
                  ((MaxNodeWoLFPHC) agent.g.graph.get(i))
                              .makeNewC(MA_Gridworld.states);
                  ((MaxNodeWoLFPHC) agent.g.graph.get(i)).initC(num);
            }
      }

      /**
       * methodos i opia ekpedevi ton agent
       */
      public void calculate(int max_episode, int num, int length,
double epsilon,
                  double df, double deltaW, double deltaL, String
filename,
                  int delivery,double lr) {

            MA_Gridworld.makeGridworld(filename);
            this.setRanges();
            this.initOffsetTable();

            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
```

```
                    MaxQ_Agent tmp = new MaxQ_Agent(i, -1, 2);//
2=MaxNodeWoLFPHC
                    this.makePolicyValues(tmp);
                    this.makeAvgPolicyValues(tmp);
                    this.makeC(tmp, 0);
                    agents.add(tmp);
              }
              double timer;
              this.setEpsilon(epsilon);
              this.setDiscountFactor(df);
              this.setDeltaL(deltaL);
              this.setDeltaW(deltaW);
              MA_State_Variables s;
              this.initLearningRate(agents,lr);
              this.makeNewEpisodeSteps(length);
              this.makeNewTimePerEpisode(length);
              count=0;
              for (int i = 0; i < max_episode; i++) {
                    if ((count % num == 0))
                          System.out.println("episode=" + count);
                    count++;
                    timer = (double) System.currentTimeMillis() / 1000;
                    // System.out.println("episode=" + i);
                    s = new MA_State_Variables(MA_Gridworld.size_x,
                                MA_Gridworld.size_y,
MA_Gridworld.numOfStartPosition,
                                MA_Gridworld.numOfGoals);
                    for (int j = 0; j < s.taxi.size(); j++) {
                          agents.get(j).setAgentPos(s.taxi.get(j));
                    }
                    this.next_state = new MA_State_Variables(s);
                    // this.maxQ_PHC(this.MaxRoot, s, i);
                    this.maxQ_WoLF_PHC(s, i, delivery, num);
                    if (num != 1) {
                          if (i == 0)
                                this.setTimePerEpisode(i, ((double)
System
                                            .currentTimeMillis() / 1000
- timer));
                          else if ((count % num == 0))
                                this.setTimePerEpisode((i + 1) / num,
((double) System
                                            .currentTimeMillis() / 1000
- timer));
                    } else {
                          this.setTimePerEpisode(i,
                                      ((double)
System.currentTimeMillis() / 1000 - timer));
                    }
                    // this.setTimePerEpisode(i,
                    // ((double) System.currentTimeMillis() / 1000 -
timer));
                    for (int j = 0; j < s.taxi.size(); j++) {
                          agents.get(j).setAgentPreviousPos(-1);
                          agents.get(j).setPassenger(-1);
                          agents.get(j).setTransfer(false);
                          agents.get(j).setTransferPrevious(false);
                          agents.get(j).setCollision(false);
```

```
                        agents.get(j).setAction(-1);
                        agents.get(j).setPrevAction(-1);
                        agents.get(j).setReward(-1);
                        agents.get(j).seqStack.clear();
                        agents.get(j).stack.clear();
                        agents.get(j).stack.push(this.MaxRoot);
                  }
            }

            // PrintInFile p = new PrintInFile();
            // p.printResults("MaxQ_WoLF_PHC-results.txt",
this.getEpisodeSteps(),
            // length);
            // p.printResults("MaxQ_WoLF_PHC-clock time.txt",
            // this.getTimePerEpisode(), length);

      }

      /**
       * methodos i opoia ilopoiei ton MAXQ-WoLF-PHC algorithmo gia
       * Multi Agent (iterative)
       *
       * @param maxNode
       * @param s
       * @param episode
       * @return
       */
      public void maxQ_WoLF_PHC(MA_State_Variables s, int episode,
                  int delivery, int num) {

            int passengerDelivered = 0;
            MaxQ_Controller control = new MaxQ_Controller();
            ArrayList<Integer> passDelivered = new
ArrayList<Integer>();

            int index = 0;
            double lr = 0;

            int action = 0;
            int childAction = 0;
            int next_index = 0;
            for (int i = 0; i < agents.size(); i++) {
                  ArrayList<MA_State_Variables> seq = new
ArrayList<MA_State_Variables>();
                  agents.get(i).seqStack.push(seq);
            }

            do {
                  for (int i = 0; i < agents.size(); i++) {
                        agents.get(i).maxNode =
agents.get(i).stack.peek();
                        agents.get(i).maxNode =
checkInstanceOf(agents.get(i), agents
                                    .get(i).maxNode, s);
                        // i periptosi opou o maxNode den einai
primitive
                        while (!((MaxNodePHC) agents.get(i).g.graph
```

```
      .get(agents.get(i).maxNode)).isPrimitive()) {
                              ArrayList<MA_State_Variables> seq = new
ArrayList<MA_State_Variables>();
                              agents.get(i).seqStack.push(seq);

                              if (!(this.isTerminal(agents.get(i),
agents.get(i).maxNode,
                                          s))) {
                                    index = this.findIndex(s);
                                    // choose action
                                    childAction =
this.chooseAction(index,
                                                ((MaxNodePHC)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).getPolicyValue());
                                    action = ((MaxNodePHC)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).children.get(childAction);


      this.updatePolicyValue(agents.get(i), index,
                                          childAction,
agents.get(i).maxNode);

      this.normalizePolicyValue(agents.get(i), index, agents
                                          .get(i).maxNode);

      this.updateAvgPolicyValue(agents.get(i), index,
                                          childAction,
agents.get(i).maxNode);

      this.normalizeAvgPolicyValue(agents.get(i), index,

      agents.get(i).maxNode);
                                    agents.get(i).stack.push(action);
                              } else {
                                    agents.get(i).stack.pop();
                                    if (agents.get(i).stack.isEmpty())
{
                                          break;
                                    }
                                    agents.get(i).maxNode =
agents.get(i).stack.peek();

                                    // elegxos an imaste se Max node
                                    agents.get(i).maxNode =
checkInstanceOf(agents.get(i),
                                                agents.get(i).maxNode,
s);
                                    // observe next state
                                    next_index =
this.findIndex(this.next_state);
                                    lr = ((MaxNodePHC)
agents.get(i).g.graph.get(agents
```

```
      .get(i).maxNode)).getLearningRate();

                                    agents.get(i).setGreedyAction(
                                                ((MaxNodePHC)
agents.get(i).g.graph.get(agents

      .get(i).maxNode)).children.get(this

      .findGreedyAction(agents.get(i), agents

      .get(i).maxNode, next_index)));


      changeCompletionValue(agents.get(i), index, next_index,
                                                action,
agents.get(i).maxNode, lr);
                                    s = new
MA_State_Variables(this.next_state);
                                    index = next_index;
                              }

                              agents.get(i).maxNode =
agents.get(i).stack.peek();
                              agents.get(i).maxNode =
checkInstanceOf(agents.get(i),
                                          agents.get(i).maxNode, s);

                        }

      agents.get(i).setPrevAction(agents.get(i).getAction());
                        agents.get(i).setAction(
                                    ((MaxNodePHC)
agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getAction());
                  }
                  index = this.findIndex(s);
                  control
                              .agentController(s, next_state, agents,
passDelivered,
                                          index);

                  // topothetounte neoi passenger stin thesi aftwn pou
metaferthikan
                  // ston proorismo tous
                  next_index = this.findIndex(next_state);
                  for (int i = 0; i < passDelivered.size(); i++) {
                        next_state.passenger.set(passDelivered.get(i),
r

      .nextInt(MA_Gridworld.numOfStartPosition));

      next_state.destination.set(passDelivered.get(i), r

      .nextInt(MA_Gridworld.numOfGoals));
                  }
```

```
                  for (int i = 0; i < agents.size(); i++) {
                        lr = ((MaxNodePHC) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getLearningRate();
                        ((MaxNodePHC)
agents.get(i).g.graph.get(agents.get(i).maxNode))
                                    .setValueFunctForPrimitive(index,
(1 - lr)
                                                * ((MaxNodePHC)
agents.get(i).g.graph

      .get(agents.get(i).maxNode))

      .getValueFunctForPrimitive(index)
                                                + (lr *
agents.get(i).getReward()));

                        MA_State_Variables tmp = new
MA_State_Variables(s);
                        agents.get(i).seqStack.peek().add(0, tmp);
                        agents.get(i).stack.pop();

                        agents.get(i).maxNode =
agents.get(i).stack.peek();
                        // elegxos an imaste se Max node
                        agents.get(i).maxNode =
checkInstanceOf(agents.get(i), agents
                                    .get(i).maxNode, s);

                        // observe next state
                        next_index = this.findIndex(this.next_state);
                        lr = ((MaxNodePHC) agents.get(i).g.graph

      .get(agents.get(i).maxNode)).getLearningRate();
                        agents.get(i).setGreedyAction(
                                    ((MaxNodePHC)
agents.get(i).g.graph

      .get(agents.get(i).maxNode)).children.get(this

      .findGreedyAction(agents.get(i),

      agents.get(i).maxNode, next_index)));

                        changeCompletionValue(agents.get(i), index,
next_index, action,
                                    agents.get(i).maxNode, lr);
                  }

                  s = new MA_State_Variables(next_state);
                  passengerDelivered += passDelivered.size();
                  if (num != 1) {
                        if (episode == 0)
                              this.addEpisodeSteps(0);
                        else if ((count % num == 0)) {
                              this.addEpisodeSteps((episode + 1) /
num);
                        }
```

```
                } else {
                        this.addEpisodeSteps(episode);
                }
                // this.addEpisodeSteps(episode);
                index = next_index;
                passDelivered.clear();

            } while (passengerDelivered < delivery);
        }

        /**
         * @param args
         */
        public static void main(String[] args) {
            // TODO Auto-generated method stub
            MA_MaxQ_WoLF_PHC m = new MA_MaxQ_WoLF_PHC();
            // m.calculate(500);

        }

}
```

MA_State_Variables.java

```
import java.util.ArrayList;
import java.util.Random;

/**
 * klasi i opoia antiprosopevi ta state variables tou episodiou
 */
public class MA_State_Variables {

        // Constructor 1
        MA_State_Variables() {
            super();
        }

        // Constructor 2
        MA_State_Variables(int size_x, int size_y, int
numOfStartPosition,
                int numOfGoals) {
            int tmp = size_x * size_y;
            boolean flag = false;
            for (int i = 0; i < MA_Gridworld.numOfAgents; i++) {
                taxi.add(r.nextInt(tmp));
                while (true) {
                    flag = false;
                    for (int j = 0; j < i; j++) {
                        if (taxi.get(j) == taxi.get(i)) {
                            flag = true;
                            break;
                        }
                    }
                    if (!(flag))
                        break;
```

```
                        taxi.set(i, r.nextInt(tmp));
                  }
            }

            for (int i = 0; i < MA_Gridworld.numOfAgents; i++)
                  for (int j = 0; j < MA_Gridworld.numOfAgents; j++) {
                        if (i == j)
                              continue;
                        if (taxi.get(i) == taxi.get(j))
                              while (true) {
                                    taxi.set(i, (r.nextInt(tmp)));

                              }
                  }

            for (int i = 0; i < MA_Gridworld.numOfPassengers; i++) {
                  passenger.add(r.nextInt(numOfStartPosition));
                  destination.add(r.nextInt(numOfGoals));
            }
      }

      // Constructor 3
      MA_State_Variables(MA_State_Variables s) {
            for (int i = 0; i < s.taxi.size(); i++)
                  this.taxi.add(s.taxi.get(i));
            for (int i = 0; i < s.passenger.size(); i++)
                  this.passenger.add(s.passenger.get(i));
            for (int i = 0; i < s.destination.size(); i++)
                  this.destination.add(s.destination.get(i));
      }

      public Random r = new Random();
      ArrayList<Integer> taxi = new ArrayList<Integer>();
      ArrayList<Integer> passenger = new ArrayList<Integer>();
      ArrayList<Integer> destination = new ArrayList<Integer>();

}
```

Node.java

```
import java.util.ArrayList;

/**
 * klasi i opoia antiprosopevi ena komvo tou grafou
 *
 * @author giannis
 *
 */
public class Node {
      ArrayList<Integer> children=new ArrayList<Integer>();
}
```

Qnode.java

```
/**
 * klasi i opoia antiprosopevi tous Q nodes tou grafou
 *
 * @author giannis
 *
 */
public class Qnode extends Node {
	// Constructor
	Qnode(int i) {
		this.C = new double[i];
		this.Cinside=new double [i]; //xrisimopoieite ston MAXQ_Q
	}

	private double[] C;
	private double[] Cinside;



	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou C
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setCvalue(int i, double value) {
		this.C[i] = value;
	}

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
tou C
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	public double getCvalue(int i) {
		return this.C[i];
	}

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou C to
	 * neo value
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addCvalue(int i, double value) {
		this.C[i] += value;
	}
```

```
	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou C
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setCinsideValue(int i, double value) {
		this.Cinside[i] = value;
	}

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou C
	 * @param i
	 * @param j
	 * @return
	 */
	public double getCinsideValue(int i) {
		return this.Cinside[i];
	}


	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou C to
	 * neo value
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addCinsideValue(int i, double value) {
		this.Cinside[i] += value;
	}

}
```

MaxNode.java

```
/**
 * klasi i opoia antiprosopevi tous Max nodes tou grafou
 *
 * @author giannis
 *
 */
public class MaxNode extends Node {
	// Constructor
	MaxNode(int i) {
		this.ValueFunctForPrimitive = new double[i];
		this.initValueFunctForPrimitive(i, 0);
	}

	private double[] ValueFunctForPrimitive; // gia primitive node
	private int action;
	private double lr0; // to arxiko learning rate
	private double learning_rate;
```

```
	private int dest;// i parametros t sto MaxNavigate(t)

	private double temperature;// xrisimopoieite sto boltzmann
exploration
	private double coolingRate; // xrisimopoieite gia na miwnete to
temperature
	private double offset=1;	//xrisimopieite gia tin allagi
thermokrasias

	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou
	 * ValueFunctForPrimitive
	 *
	 * @param i
	 * @param value
	 */
	public void setValueFunctForPrimitive(int i, double value) {
		this.ValueFunctForPrimitive[i] = value;
	}



	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou ValueFunct
	 * @param i
	 * @return
	 */
	public double getValueFunctForPrimitive(int i) {
		return ValueFunctForPrimitive[i];
	}

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou
	 * ValueFunctForPrimitive to neo value
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addValueFunctForPrimitive(int i, double value) {
		this.ValueFunctForPrimitive[i] += value;
	}

	/**
	 * @param action
	 *            the action to set
	 */
	public void setAction(int action) {
		this.action = action;
	}

	/**
	 * @return the action
	 */
	public int getAction() {
		return action;
```

```
}

/**
 * methodos i opoia kathorizei to learning rate
 *
 * @param num
 */
public void setLearningRate(double num) {
	this.learning_rate = num;
}

/**
 * methodos i opoia epistrefi to learning rate
 *
 * @return
 */
public double getLearningRate() {
	return this.learning_rate;
}

/**
 * methodos i opoia kathorizei to lr0
 *
 * @param lr0
 */
public void setLr0(double lr0) {
	this.lr0 = lr0;
}

/**
 * methodos i opoia epistrefi to lr0
 *
 * @return the lr0
 */
public double getLr0() {
	return this.lr0;
}

/**
 * methodos i opoia kathorizi to temperature
 *
 * @param temperature
 */
public void setTemperature(double temperature) {
	this.temperature = temperature;
}

/**
 * methodos i opoia epistrefi to temperature
 *
 * @return
 */
public double getTemperature() {
	return temperature;
}
```

```
	/**
	 * methodos i opoia kathorizi to coolingRate
	 *
	 * @param coolingRate
	 *            the coolingRate to set
	 */
	public void setCoolingRate(double coolingRate) {
		this.coolingRate = coolingRate;
	}

	/**
	 * methodos i opoia epistrefi to coolingRate
	 *
	 * @return the coolingRate
	 */
	public double getCoolingRate() {
		return coolingRate;
	}

	/**
	 * methodos i opoia allazi to temperature vasi kapiou cooling
rate
	 */
	public void changeTemperature(int numStep) {
		this.setTemperature(this.offset+(this.temperature
*Math.pow( this.coolingRate,numStep)));
	}

	/**
	 * methodos i opoia allazi to learning rate tou Max node
	 *
	 * @param episode
	 * @param max_episode
	 */
	public void changeLearningRate(int episode, int max_episode) {
		//this.learning_rate = this.getLr0()
			//	* Math.exp(-((double) episode / max_episode));
		this.learning_rate = this.getLr0();
	}

	/**
	 * methodos i opoia arxikopoi ton pinaka ValueFunctForPrimitive
	 * me tin timi value
	 *
	 * @param x
	 * @param y
	 * @param value
	 */
	public void initValueFunctForPrimitive(int x, double value) {
		for (int i = 0; i < x; i++)
			this.ValueFunctForPrimitive[i] = value;
	}
```

```
	/**
	 * methodos i opoia elegxei an enas komvos tou grafou einai
	 * primitive action (otan einai filo)
	 *
	 * @param node
	 * @return
	 */
	public boolean isPrimitive() {
		return this.children.isEmpty();
	}
	/**
	 * methodos i opoia dini timi sto dest
	 *
	 * @param dest
	 *            the dest to set
	 */
	public void setDest(int dest) {
		this.dest = dest;
	}
	/**
	 * methodos i opoia epistrefi tin timi dest
	 *
	 * @return the dest
	 */
	public int getDest() {
		return dest;
	}
}
```

MaxNodePHC.java

```
/**
 * klasi i opoia antiprosopevi tous Max nodes tou grafou kai
 * xrisimopoieitati gia ton algorithmo MAXQ-PHC kai klironomei apo tin
 * klasi MaxNode
 * @author Giannis
 *
 */
public class MaxNodePHC extends MaxNode {

	MaxNodePHC(int i) {
		super(i);
		// TODO Auto-generated constructor stub
	}

	private double[][] policyValue;

	/************** methodoi get,set,add phc ******************/
	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou
	 * policyValue
	 * @param i
	 * @param j
	 * @param value
	 */
```

```
	public void setPolicyValue(int i, int j, double value) {
		this.policyValue[i][j] = value;
	}

	/**
	 * methodos i opoia epistrefi ton pinaka policyValue
	 *
	 * @return
	 */
	public double[][] getPolicyValue() {
		return this.policyValue;
	}

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * policyValue
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	public double getPolicyValue(int i, int j) {
		return this.policyValue[i][j];
	}

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou
policyValue to neo
	 * value
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addPolicyValue(int i, int j, double value) {
		this.policyValue[i][j] += value;
	}

	/**
	 * methodos i opoia dimiourga ena pinaka policyValue
	 */
	public void makeNewPolicyValue(int states, int actions) {
		policyValue = new double[states][actions];
	}

	/**
	 * methodos i opoia arxikopoiei to policyValue
	 *
	 * @param actions
	 */
	public void initPolicyValue(int states, int actions) {
		for (int i = 0; i < states; i++) {
			for (int j = 0; j < actions; j++) {
				this.setPolicyValue(i, j, ((double) 1 /
actions));
			}
		}
	}
```

```
	/**
	 * methodos i opoia epistrefi to mikos tou pinaka policyValue
	 *
	 * @return
	 */
	public int getPolicyValueLength() {
		return this.policyValue[0].length;
	}

}
```

MaxNodeWoLFPHC.java

```
/**
 * klasi i opoia antiprosopevi tous Max nodes tou grafou kai
xrisimopoieitati
 * gia ton algorithmo MAXQ-WoLF-PHC kai klironomei apo tin klasi
MaxNodePHC
 *
 * @author Giannis
 *
 */
public class MaxNodeWoLFPHC extends MaxNodePHC {

	MaxNodeWoLFPHC(int i) {
		super(i);
		// TODO Auto-generated constructor stub
	}

	private double[][] avgPolicyValue; // average policy value
	private int[] C;

	/***************** WoLF-PHC *********************/
	/**
	 * methodos i opia dini timi stin sigkekrimeni thesi tou
	 * avgPolicyValue
	 * @param i
	 * @param j
	 * @param value
	 */
	public void setAvgPolicyValue(int i, int j, double value) {
		this.avgPolicyValue[i][j] = value;
	}

	/**
	 * methodos i opoia epistrefi tin timi sti sigkekrimeni thesi
	 * tou avgPolicyValue
	 *
	 * @param i
	 * @param j
	 * @return
	 */
	public double getAvgPolicyValue(int i, int j) {
```

```
		return this.avgPolicyValue[i][j];
	}

	/**
	 * methodos i opoia epistrefi ton pinaka avgPolicyValue
	 *
	 * @return
	 */
	public double[][] getAvgPolicyValue() {
		return this.avgPolicyValue;
	}

	/**
	 * methodos i opoia prostheti stin sigkekrimeni timi tou
	 * avgPolicyValue to neo value
	 *
	 * @param i
	 * @param j
	 * @param value
	 */
	public void addAvgPolicyValue(int i, int j, double value) {
		this.avgPolicyValue[i][j] += value;
	}

	/**
	 * methodos i opoia dini timi se mia thesi tou pinaka C
	 *
	 * @param state
	 * @param num
	 */
	public void setC(int state, int num) {
		C[state] = num;
	}

	/**
	 * methodos i opoia epistrefi mia timi tou pinaka C
	 *
	 * @return
	 */
	public int getC(int state) {
		return C[state];
	}

	/**
	 * methodos i opoia prostheti 1 se mia thesi tou pinaka C
	 *
	 * @param state
	 */
	public void addOneToC(int state) {
		C[state]++;
	}

	/**
	 * methodos i opoia dimiourga ena pinaka avgPolicyValue
	 */
	public void makeNewAvgPolicyValue(int states, int actions) {
		avgPolicyValue = new double[states][actions];
	}
```

```
	/**
	 * methodos i opoia dimiourga ena pinaka C
	 */
	public void makeNewC(int states) {
		C = new int[states];
	}

	/**
	 * methodos i opoia arxikopoiei to avgPolicyValue
	 *
	 * @param actions
	 */
	public void initAvgPolicyValue(int states, int actions) {
		for (int i = 0; i < states; i++) {
			for (int j = 0; j < actions; j++) {
				this.setAvgPolicyValue(i, j, ((double) 1 /
actions));
			}
		}
	}
	/**
	 * methodos i opoia epistrefi to mikos tou pinaka avgPolicyValue
	 *
	 * @return
	 */
	public int getAvgPolicyValueLength() {
		return this.avgPolicyValue[0].length;
	}

	/**
	 * methodos i opoia arxikopoiei to C
	 *
	 * @param actions
	 */
	public void initC(int num) {
		for (int i = 0; i < 500; i++) {
			this.setC(i, num);
		}
	}
}
```

NodeValue.java

```
/**
 * klasi i opoia xrisimopoieite stin greedy execution policy gia na
 * apothikevontai to node k to value tou
 *
 * @author Giannis
 *
 */
public class NodeValue {
	NodeValue(double value, int node) {
		this.value = value;
```

```
			this.node = node;
		}

		double value;
		int node;
}
```

Wall.java

```
/**
 * klasi i opoia antiprosopevi tis sintetagmenes enos wall sto grid
 *
 * @author Giannis
 *
 */
public class Wall {
		//Constructor
		Wall(int cell1, int cell2) {
			this.cell1 = cell1;
			this.cell2 = cell2;
		}

		int cell1;
		int cell2;
}
```

PrintInFile.java

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;

/**
 * klasi i opoia einai ipefthini gie ektiposi apotelesmaton se arxio
 */
public class PrintInFile {

		/**
		 * methodos i opoia tiponi se arxio ta steps per episode
		 *
		 * @param filename
		 * @param episodeSteps
		 * @param max_episode
		 */
		public void printResults(String filename, int[] episodeSteps,
int max_episode) {
			FileOutputStream fout;
			try {
				// Anoigma arxeiou eksodou
				fout = new FileOutputStream(filename);
				PrintStream p = new PrintStream(fout);
				for (int i = 0; i < max_episode; i++) {
```

```
                        p.println(i + "\t" + episodeSteps[i]);
                  }
                  fout.close();
            }
            // Emfanisi minimatos se periptwsi exception. (Catch
exception)
            catch (IOException e) {
                  System.err.println("Unable to write to file");
                  System.exit(-1);
            }
      }

      /**
       * methodos i opoia tiponi se arxio to clock time per episode
       *
       * @param filename
       * @param timePerEpisode
       * @param max_episode
       */
      public void printResults(String filename, double[]
timePerEpisode,
                  int max_episode) {
            FileOutputStream fout;
            try {
                  // Anoigma arxeiou eksodou
                  fout = new FileOutputStream(filename);
                  PrintStream p = new PrintStream(fout);
                  for (int i = 0; i < max_episode; i++) {
                        p.println(i + "\t" + timePerEpisode[i]);
                  }
                  fout.close();
            }
            // Emfanisi minimatos se periptwsi exception. (Catch
exception)
            catch (IOException e) {
                  System.err.println("Unable to write to file");
                  System.exit(-1);
            }
      }

      /**
       * methodos i opoia tiponi se arxio ta steps per episode gia
kathe trial
       *
       * @param filename
       * @param episodeSteps
       * @param trials
       * @param episodes
       */
      public void printResults(String filename, int[][] episodeSteps,
int trials,
                  int episodes) {
            FileOutputStream fout;
            try {
                  // Anoigma arxeiou eksodou
                  fout = new FileOutputStream(filename);
                  PrintStream p = new PrintStream(fout);
                  for (int i = 0; i < trials; i++) {
```

```
                        for (int j = 0; j < episodes; j++) {
                              p.print("\t" + episodeSteps[i][j]);
                        }
                        p.println();
                  }
                  fout.close();
            }
            // Emfanisi minimatos se periptwsi exception. (Catch
exception)
            catch (IOException e) {
                  System.err.println("Unable to write to file");
                  System.exit(-1);
            }
      }


      /**
       * methodos i opoia tiponi se arxio ta steps per episode
       *
       * @param filename
       * @param episodeSteps
       * @param max_episode
       */
      public void printAvgResults(String filename, double[]
averageSteps, int episodes) {
            FileOutputStream fout;
            try {
                  // Anoigma arxeiou eksodou
                  fout = new FileOutputStream(filename);
                  PrintStream p = new PrintStream(fout);
                  for (int i = 0; i < episodes; i++) {
                        p.println(i + "\t" + averageSteps[i]);
                  }
                  fout.close();
            }
            // Emfanisi minimatos se periptwsi exception. (Catch
exception)
            catch (IOException e) {
                  System.err.println("Unable to write to file");
                  System.exit(-1);
            }
      }

      /**
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub

      }

}
```