Ατομική Διπλωματική Εργασία

ΜΕΛΕΤΗ ΤΗΣ ΕΞΕΛΙΞΗΣ ΤΟΥ ΑΥΤΟΕΛΕΓΧΟΥ ΜΕΣΩ ΠΡΟΔΕΣΜΕΥΣΗΣ

Ζιανέτ Σιαχουάν

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη της Εξέλιξης του Αυτοέλεγχου Μέσω Προδέσμευσης

Ζιανέτ Σιαχουάν

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 2015

Ευχαριστίες

Με την ολοκλήρωση της πτυχιακής μου εργασίας, θα ήθελα να ευχαριστήσω τους ανθρώπους οι οποίοι βοήθησαν ο καθένας με το δικό του τρόπο στην περάτωση αυτής της εργασίας.

Κατά κύριο λόγο, οφείλω να ευχαριστήσω θερμά τον επιβλέπων καθηγητή μου Δρ. Χρίστο Χριστοδούλου καθώς και τον διδακτορικό φοιτητή Βασίλη Βασιλειάδη για την πολύτιμη βοήθεια και καθοδήγηση τους καθ' όλη την διάρκεια εκπόνησης αυτής της Διπλωματικής Εργασίας.

Ιδιαίτερες ευχαριστίες θα ήθελα να δώσω επίσης στην οικογένεια μου και όλους τους φίλους μου, οι οποίοι ήταν πάντα δίπλα μου καθ' όλη τη διάρκεια της φοίτησης μου παρέχοντας μου κατανόηση και υποστήριξη.

Περίληψη

Ένα άτομο παρουσιάζει αυτοέλεγχο όταν επιλέγει να κάνει μια πράξη η οποία θα του επιφέρει μια αργοπορημένη αλλά μεγαλύτερη αμοιβή (γεγονότα που σχετίζονται με την λογική και την ορθολογιστική σκέψη), παρά μια πράξη η οποία θα του επιφέρει μια συντομότερη αλλά μικρότερη αμοιβή (γεγονότα που σχετίζονται με το συναίσθημα και τον αυθορμητισμό). Ο αυτοέλεγχος μπορεί να επιτευχθεί πιο εύκολα με τη χρήση της προδέσμευσης. Προδέσμευση είναι μια ενέργεια που κάνουμε τώρα η οποία απαλείφει την επιλογή του πειρασμού στο μέλλον.

Σε αυτή τη Διπλωματική Εργασία διερευνάται ο τρόπος με τον οποίο η προδέσμευση συμβάλει στον αυτοέλεγχο ενός οργανισμού κατά την εξέλιξη του. Για τη διεκπεραίωση της μελέτης αυτής γίνεται προσομοίωση της εξέλιξης ατόμων, τα οποία μαθαίνουν να έχουν αυτοέλεγχο με τη χρήση προδέσμευσης.

Η προσομοίωση γίνεται εφικτή δημιουργώντας αρχικά ένα πληθυσμό από άτομα, τα οποία θα εκπαιδευτούν να έχουν βελτιωμένη συμπεριφορά αυτοέλεγχου. Ο εγκέφαλος κάθε ατόμου προσομοιώνεται αναπτύσσοντας ένα υπολογιστικό μοντέλο, το οποίο θα περιέχει όλα τα βασικά χαρακτηριστικά έτσι ώστε να είναι βιολογικά και ψυχολογικά σχετικό, να αντιπροσωπεύει δηλαδή σε μεγάλο βαθμό τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου. Ο εγκέφαλος θα εκπαιδεύεται να έχει αυτοέλεγχο χρησιμοποιώντας ένα παιχνίδι γενικού αθροίσματος, το «Επαναλαμβανόμενο Δίλημμα του φυλακισμένου». Το μοντέλο θα έχει εξελικτικό χαρακτήρα, θα υπόκειται δηλαδή σε προσομοίωση της γενετικής εξέλιξης με μάθηση. Η εφαρμογή της γενετικής εξέλιξης γίνεται χρησιμοποιώντας ένα εξελικτικό αλγόριθμο, τις Εξελικτικές Στρατηγικές (Evolution Strategies). Μέσω αυτών των στρατηγικών τα άτομα υπόκεινται σε συγκεκριμένες διαδικασίες οι οποίες έχουν ως αποτέλεσμα κάθε γενεά να περιέχει άτομα με πιο βελτιστοποιημένα χαρακτηριστικά από αυτά της προηγούμενης γενεάς, τα οποία είναι σχετικά με την επίτευξη αυτοέλεγχου.

Στα αποτελέσματα της προσομοίωσης παρατηρήθηκε ότι αφού τα άτομα εκπαιδευτούν επαρκώς, με την πάροδο των γενεών παρουσιάζεται βελτιωμένη συμπεριφορά αυτοέλεγχου σε αυτά. Η προδέσμευση φάνηκε να αποτελεί σημαντικό παράγοντα στην

επίτευξη αυτοελέγχου καθώς επίσης και η εφαρμογή της εξέλιξης στο μοντέλο. Μέσα από τα πειράματα που εκτελέστηκαν μελετήθηκαν οι παράγοντες που μπορεί να επηρεάσουν τον αυτοέλεγχο, όπως για παράδειγμα ο αλγόριθμος μάθησης που χρησιμοποιείται και οι τιμές των παραμέτρων της εξέλιξης και των ατόμων που απαρτίζουν τον πληθυσμό της προσομοίωσης.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Ο ρόλος και ο στόχος της έρευνας

Σε αυτή τη Διπλωματική Εργασία θα μελετηθεί ο τρόπος με τον οποίο η προδέσμευση συμβάλει στον αυτοέλεγχο των οργανισμών κατά την εξέλιξη τους. Πολλές φορές αναρωτιόμαστε γιατί κάποιοι άνθρωποι κατορθώνουν να ελέγχουν τις ενέργειες τους έχοντας μια οργανωμένη ζωή, ενώ κάποιοι άλλοι δεν μπορούν να ελέγξουν τον παρορμητισμό τους υποκύπτοντας συχνά στην επιλογή του πειρασμού. Οι άνθρωποι που ανήκουν στην πρώτη περίπτωση δρουν με βάση την ορθολογιστική σκέψη παρουσιάζοντας βελτιωμένη συμπεριφορά αυτοελέγχου, ενώ οι άνθρωποι στην δεύτερη περίπτωση δρουν με βάση την ενστικτώδη συμπεριφορά και δεν παρουσιάζουν αυτοέλεγχο. Προβλήματα στην άσκηση αυτοέλεγχου προκύπτουν όταν υπάρχει έλλειψη προθυμίας ή κίνητρου για να πραγματοποιηθεί η αναστολή των επιθυμιών μας. Δηλαδή, ξέρουμε τι είναι καλό για μας (νόηση), αλλά δεν το κάνουμε (κίνητρο). Η διάκριση μεταξύ νόησης και κινήτρου έχει παρομοιαστεί με τη διάκριση μεταξύ των υψηλότερων και χαμηλότερων λειτουργιών του εγκεφάλου μας [1].

Ο αυτοέλεγχος προκύπτει από την επιθυμία να ελέγξουμε την συμπεριφορά μας και πιο συγκεκριμένα να καταλήγουμε να κάνουμε την «ορθή» ενέργεια αντί να δρούμε παρορμητικά. Πολλές φορές βρισκόμαστε σε δίλημμα εάν θα επιλέξουμε να κάνουμε αυτό που θέλουμε αυτή τη στιγμή που μπορεί να μη συνάδει με τη λογική (ενστικτώδης

συμπεριφορά) ή αυτό που είναι ορθό και θεωρείται λογικό (ορθολογιστική σκέψη). Αυτό το δίλημμα, όπως έχει αναφερθεί και προηγουμένως, είναι μια εσωτερική σύγκρουση ανάμεσα στην ορθολογιστική σκέψη που σχετίζεται με τις υψηλότερες λειτουργίες του εγκεφάλου μας και στην ενστικτώδη συμπεριφορά που σχετίζεται με τις χαμηλότερες λειτουργίες του εγκεφάλου μας [1].

Βάση της επιστήμης της ψυχολογίας, αφού τα ανθρώπινα όντα είναι εκ φύσεως ορθολογιστές οποιαδήποτε ενέργεια που δε συνάδει με την λογική ή την ορθότητα θεωρείται παράλογη συμπεριφορά. Είναι σημαντικό λοιπόν ένας οργανισμός να μπορεί να έχει αυτοέλεγχο, να μπορούν δηλαδή να συμβιβάζονται τα δύο μέρη του εγκεφάλου (ορθολογιστική σκέψη και ενστικτώδης συμπεριφορά) και το αποτέλεσμα να ωφελεί τον οργανισμό ως σύνολο. Η προδέσμευση είναι ένας τρόπος για επίτευξη αυτού του στόχου, δηλαδή της βελτιωμένης συμπεριφοράς αυτοελέγχου, και θα επεξηγηθεί σε αυτή τη Διπλωματική Εργασία πως αυτό μπορεί να εφαρμοστεί σε ένα υπολογιστικό μοντέλο.

Υπάρχουν διάφορες πιθανές εξελικτικές εξηγήσεις για την ύπαρξη του αυτοέλεγχου μέσω της προδέσμευσης. Για παράδειγμα ο αυτοέλεγχος μέσω της προδέσμευσης να είναι αποτέλεσμα μιας εσωτερικής σύγκρουσης ανάμεσα στα δύο μέρη του εγκεφάλου, ή μια παρενέργεια της εξέλιξης των μηχανισμών δέσμευσης για καταστάσεις θεωρίας παιγνίου (game-theoretical situations) όπου η προδέσμευση είναι χρήσιμη, ή μια συμπεριφορά που προκύπτει από τον εξελικτικό συμβιβασμό των ειδών στην πολυπλοκότητα του περιβάλλοντος και τη μεταβλητότητα [2] [3].

Μέσω της προσομοίωσης του εγκεφάλου σαν ένα υπολογιστικό σύστημα θα γίνει πιο κατανοητός ο τρόπος με τον οποίο επιτυγχάνεται ο αυτοέλεγχος σε ένα οργανισμό και πως αυτός μπορεί να εξασκηθεί και να ενισχυθεί εφαρμόζοντας προδέσμευση. Ένα λειτουργικά αποσυντιθέμενο υπολογιστικό σύστημα μπορεί να έχει εσωτερικές συγκρούσεις, αφού διαφορετικές μέθοδοι επιδεικνύουν διαφορετικές συμπεριφορές. Ο αυτοέλεγχος μέσω προδέσμευσης είναι ένα παράδειγμα αυτής της εσωτερικής σύγκρουσης[2]. Το υπολογιστικό μοντέλο που θα αναπτυχθεί θα περιέχει όλα τα βασικά χαρακτηριστικά έτσι ώστε να αντιπροσωπεύει σε μεγάλο βαθμό τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου και να είναι βιολογικά και ψυχολογικά σχετικό. Τα δύο μέρη του εγκεφάλου εκ φύσεως συναγωνίζονται για το ποιος θα έχει τον έλεγχο του

οργανισμού. Σκοπός εδώ είναι να κατορθώσουν να συμβιβαστούν αυτά τα δύο μέρη, καταλήγοντας έτσι στον αυτοέλεγχο του οργανισμού. Το σύστημα θα έχει εξελικτικό χαρακτήρα, θα υπόκειται δηλαδή σε προσομοίωση της γενετικής εξέλιξης με μάθηση. Θα μελετηθεί η εφαρμογή της προδέσμευσης σε σχέση με τον αυτοέλεγχο των ατόμων εν όσον αυτά εξελίσσονται σχηματίζοντας καινούριες γενεές.

1.2 Σχετική έρευνα

Ο αυτοέλεγχος μέσω της προδέσμευσης έχει ήδη μοντελοποιηθεί και εφαρμοστεί σε κάποια υπολογιστικά μοντέλα, με διάφορες υλοποιήσεις [2] [1] [4]. Σε αυτή την Διπλωματική Εργασία θα ακολουθηθεί μια νέα προσέγγιση. Πιο συγκεκριμένα, θα αναπτυχθεί ένα μοντέλο το οποίο θα προσομοιώνει τις λειτουργίες του εγκεφάλου που σχετίζονται με την επίτευξη αυτοελέγχου χρησιμοποιώντας προδέσμευση. Αυτό το μοντέλο θα έχει εξελικτικό χαρακτήρα, θα υπόκειται δηλαδή σε προσομοίωση της γενετικής εξέλιξης με μάθηση. Η εφαρμογή της εξέλιξης στο μοντέλο θα γίνεται με τη χρήση ενός εξελικτικού αλγορίθμου, τις Εξελικτικές Στρατηγικές [5] [6]. Για την συγκεκριμένη μελέτη που γίνεται σε αυτή τη Διπλωματική Εργασία έχει ήδη αναπτυχθεί ένα υπολογιστικό μοντέλο, στα πλαίσια της Διδακτορικής Διατριβής της Gaye Delphine Banfield [2], στο οποίο η εφαρμογή της γενετικής εξέλιξης γίνεται χρησιμοποιώντας ένα άλλο εξελικτικό αλγόριθμο, τους Γενετικούς Αλγόριθμους (Genetic Algorithms) [7].

Οι Εξελικτικοί Αλγόριθμοι μπορούν να υλοποιηθούν με μια από τις τρείς ακόλουθες κύριες τεχνικές: Γενετικοί Αλγόριθμοι, Εξελικτικός Προγραμματισμός και Εξελικτικές Στρατηγικές [**2**]. Παρόλο που και οι τρείς τεχνικές αρχίζουν με ένα πληθυσμό ατόμων που θα εξελιχτούν, κάθε μια από τις οποίες δίνει έμφαση σε διαφορετική πτυχή της φυσικής εξέλιξης. Στους Γενετικούς Αλγόριθμους το άτομο συνήθως αναπαρίσταται με μια σειρά από δυαδικά ψηφία, ο Εξελικτικός Προγραμματισμός επικεντρώνεται σε διαδικασίες που αποφέρουν αλλαγές στη συμπεριφορά μέσα σε μια ομάδα και οι Εξελικτικές Στρατηγικές επικεντρώνονται σε αλλαγές στη συμπεριφορά ενός ατόμου.

1.3 Περίγραμμα Διπλωματικής Εργασίας

Στο Κεφάλαιο 2 περιέχεται το γνωσιολογικό υπόβαθρο που είναι απαραίτητο για την κατανόηση και διεκπεραίωση αυτής της Διπλωματικής Εργασίας. Περιγράφονται έννοιες όπως ο Αυτοέλεγχος, Προδέσμευση, Ενισχυτική Μάθηση, το παιχνίδι γενικού αθροίσματος «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» και Εξελικτικές Στρατηγικές.

Στο Κεφάλαιο 3 περιγράφεται ο σχεδιασμός και η υλοποίηση του υπολογιστικού μοντέλου, το οποίο προσομοιώνει την γενετική εξέλιξη ατόμων που μαθαίνουν να έχουν βελτιωμένη συμπεριφορά αυτοελέγχου. Η περιγραφή του σχεδιασμού και της υλοποίησης διαχωρίζονται σε δύο σκέλη, την μοντελοποίηση αυτοελέγχου ενός ατόμου και την εφαρμογή της εξέλιξης στο μοντέλο.

Στο Κεφάλαιο 4 περιγράφονται τα πειράματα που εκτελέστηκαν στο υπολογιστικό μοντέλο, καθώς και μερικές διαφοροποιήσεις που δοκιμάστηκαν κατά την υλοποίηση του για εύρεση βέλτιστων αποτελεσμάτων. Επίσης περιέχεται σχολιασμός των αποτελεσμάτων της προσομοίωσης και συζήτηση διάφορων εναλλακτικών προσεγγίσεων ή βελτιώσεων που μπορεί να προκύψουν.

Στο Κεφάλαιο 5 περιέχονται μια σύνοψη αυτής της Διπλωματικής Εργασίας, τα συμπεράσματα από την υλοποίηση της, καθώς και η μελλοντική εργασία που μπορεί να προκύψει σε αυτό το θέμα.

Κεφάλαιο 2

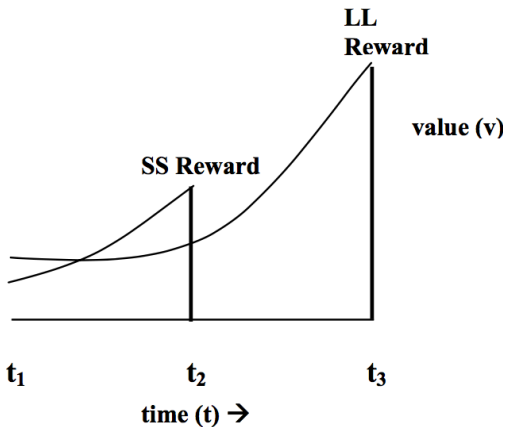
Γνωσιολογικό Υπόβαθρο

2.1 Αυτοέλεγχος

Ο αυτοέλεγχος μπορεί να ορισθεί ως ο έλεγχος των συναισθημάτων, επιθυμιών, ή ενεργειών ενός ατόμου, με θέληση του ίδιου του ατόμου. Πιο συγκεκριμένα, προκύπτει από την επιθυμία να ελέγξουμε την συμπεριφορά μας [1]. Πολλές φορές βρισκόμαστε σε δίλημμα εάν θα επιλέξουμε να κάνουμε αυτό που θέλουμε αυτή τη στιγμή που έχει άμεση αλλά μικρή ανταμοιβή (ενστικτώδης συμπεριφορά) ή αυτό που είναι ορθό για το μέλλον που έχει μεγαλύτερη αλλά καθυστερημένη ανταμοιβή (ορθολογιστική σκέψη). Για παράδειγμα, ένας φοιτητής βρίσκεται σε δίλημμα εάν θα πάει βόλτα το βράδυ ή αν θα μείνει σπίτι να διαβάσει για την εξέταση που έχει σε τρεις ημέρες. Εάν επιλέξει να κάνει αυτό που θέλει αυτή την στιγμή, δηλαδή να πάει βόλτα, η αμοιβή θα έρθει σύντομα αλλά θα είναι μικρής αξίας. Εάν επιλέξει να κάνει αυτό που είναι ορθό για το μέλλον του, δηλαδή να μείνει σπίτι να διαβάσει, η αμοιβή θα αργήσει να έρθει αλλά θα είναι μεγαλύτερης αξίας. Αυτό το δίλημμα είναι μια εσωτερική σύγκρουση ανάμεσα στην ορθολογιστική σκέψη που σχετίζεται με τις υψηλότερες λειτουργίες του εγκεφάλου μας

και στην ενστικτώδη συμπεριφορά που σχετίζεται με τις χαμηλότερες λειτουργίες του εγκεφάλου μας [1].

Ο αυτοέλεγχος μας εξασκείται όταν επιλέγουμε την αργοπορημένη αλλά μεγαλύτερη αμοιβή έναντι της συντομότερης αλλά μικρότερης [2], δηλαδή στο παράδειγμα που αναφέρθηκε πιο πριν ο φοιτητής να μείνει σπίτι ολόκληρο το βράδυ να διαβάσει για την εξέταση που έχει σε τρεις ημέρες. Αυτές οι επιλογές καθώς και οι αξίες τους σε σχέση με τη χρονική στιγμή φαίνονται στο Σχήμα 2.1.



Σχήμα 2.1 Απεικόνιση της επιλογής της καθυστερημένης-μεγαλύτερης έναντι της συντομότερης-μικρότερης ανταμοιβής [8]

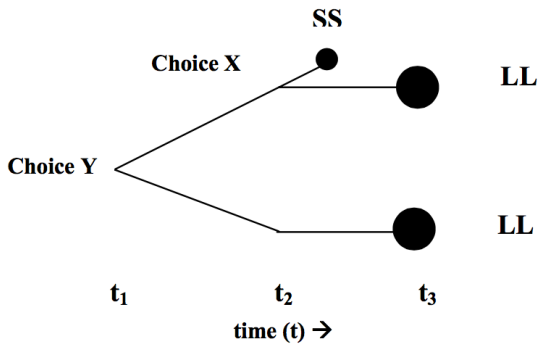
Όπως φαίνεται στο Σχήμα 2.1, αρχικά η αξία της αργοπορημένης-μεγαλύτερης (LL) αμοιβής είναι μεγαλύτερη από την αξία της συντομότερης-μικρότερης (SS) αμοιβής. Όμως τη χρονική στιγμή t_2 που είναι πολύ κοντά η απόκτηση της συντομότερης-μικρότερης και πολύ μακριά η απόκτηση της αργοπορημένης-μεγαλύτερης αμοιβής, οι αξίες αντιστρέφονται με αποτέλεσμα η αξία της συντομότερης-μικρότερης να είναι μεγαλύτερη από την αξία της αργοπορημένης-μεγαλύτερης αμοιβής. Αυτό συμβαίνει επειδή δίνεται μια μικρή έκπτωση στην αξία της αργοπορημένης-μεγαλύτερης αμοιβής διότι βρίσκεται πιο μακριά χρονικά και συνεπώς τα αποτελέσματα θα αργήσουν να έρθουν άρα είναι μικρότερη η σημασία που τους δίνεται. Ως αποτέλεσμα προκύπτει δίλημμα τη χρονική στιγμή t_2 για την πράξη που θα επιλεχθεί, και αφού υπάρχει δίλημμα τότε υπάρχει και εξάσκηση του αυτοέλεγχου του ατόμου εκείνη τη στιγμή. Αξίζει να

σημειωθεί ότι δίλημμα υπάρχει μόνο όταν οι αξίες των δύο αμοιβών είναι κοντινές. Σε περίπτωση που οι δύο αξίες έχουν μεγάλη διαφορά τότε δεν θα υπάρχει δίλημμα αφού όπως είναι φυσικό θα επιλεχθεί η μεγαλύτερη αξία. Το άτομο θα παρουσιάσει αυτοέλεγχο εάν τη χρονική στιγμή t_2 επιλέξει την αργοπορημένη-μεγαλύτερη αμοιβή παρά τη συντομότερη-μικρότερη.

2.2 Προδέσμευση

Η προδέσμευση μπορεί να ορισθεί ως μια ενέργεια που κάνουμε τώρα με σκοπό να απαλείψουμε την επιλογή του πειρασμού στο μέλλον. Δηλαδή, αποτρέπουμε μια ενέργεια μας στο μέλλον η οποία δεν πρέπει να γίνει ή την κάνουμε πιο δύσκολο να εκπληρωθεί [2]. Η προδέσμευση επιλύει την εσωτερική σύγκρουση ανάμεσα στα δύο μέρη του εγκεφάλου μας περιορίζοντας ή απορρίπτοντας μελλοντικές μας επιλογές του πειρασμού [1]. Συνεπώς η προδέσμευση αποτελεί ένα μηχανισμό για υποβοήθηση της επιλογής της αργοπορημένης-μεγαλύτερης αμοιβής παρά της συντομότερης-μικρότερης.

Ένα παράδειγμα εφαρμογής της προδέσμευσης σύμφωνα με το παράδειγμα του φοιτητή που αναφέρθηκε προηγουμένως, θα ήταν να πει ο φοιτητής στους φίλους του να μην του προτείνουν να πάει βόλτα μαζί τους εάν διοργάνωναν κάτι τέτοιο. Έτσι ο φοιτητής δεν θα είχε την επιλογή της βόλτας και θα ήταν πιο πιθανόν να παραμείνει στο σπίτι να διαβάσει για την εξέταση που θα είχε. Ως αποτέλεσμα βοήθησε στην επίτευξη του αυτοέλεγχου του επιλέγοντας την αργοπορημένη αλλά μεγαλύτερη αμοιβή.



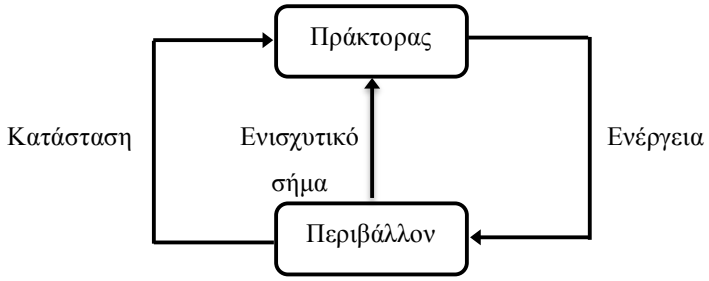
Σχήμα 2.2 Απεικόνιση της εφαρμογής της προδέσμευσης [8]

Όπως φαίνεται στο Σχήμα 2.2, όταν εφαρμόζεται προδέσμευση, τη χρονική στιγμή t_2 απαλείφεται η επιλογή της συντομότερης-μικρότερης αμοιβής με αποτέλεσμα η μόνη δυνατή επιλογή να είναι η αργοπορημένη-μεγαλύτερη αμοιβή. Συνεπώς επιτυγχάνεται ο αυτοέλεγχος.

2.3 Ενισχυτική Μάθηση

2.3.1 Περιγραφή Μοντέλου Ενισχυτικής Μάθησης

Η Ενισχυτική Μάθηση (Reinforcement Learning) είναι μια κατηγορία μηχανικής μάθησης (machine learning) στην οποία το σύστημα προσπαθεί να μάθει μέσω της αλληλεπίδρασης του με το περιβάλλον. Είναι ο πιο φυσικός τρόπος μάθησης και έχει τις ρίζες της στην ψυχολογία της μάθησης των ζώων.



Σχήμα 2.3 Μοντέλο ενισχυτικής μάθησης

Στο Σχήμα 2.3 φαίνονται τα συστατικά στοιχεία της ενισχυτικής μάθησης τα οποία περιγράφουν τον τρόπο λειτουργίας της. Υπάρχουν δύο οντότητες, ο πράκτορας και το περιβάλλον. Ο πράκτορας είναι αυτός που μαθαίνει μέσω της αλληλεπίδρασης του με το περιβάλλον και μέσω της μάθησης του γίνεται καλύτερη η απόδοση του με την πάροδο του χρόνου. Όταν ο πράκτορας εκτελέσει μια ενέργεια, αυτή η ενέργεια στέλνεται στο περιβάλλον. Όταν το περιβάλλον λάβει την ενέργεια του πράκτορα, τότε του στέλνει πίσω ένα ενισχυτικό σήμα το οποίο περιέχει την αμοιβή του για την ενέργεια που μόλις

εκτέλεσε. Αυτό το ενισχυτικό σήμα παράγεται από ένα «κριτή» που εμπεριέχεται στο περιβάλλον, ο οποίος αξιολογεί την ενέργεια του πράκτορα. Ο κριτής δεν λέει ποια ενέργεια είναι η καλύτερη στην παρούσα κατάσταση, αλλά μόνο αξιολογεί την ενέργεια που εκτελέστηκε. Η αμοιβή που λαμβάνει ο πράκτορας μπορεί να είναι είτε επιβράβευση είτε τιμωρία. Δίνεται επιβράβευση όταν η ενέργεια που εκτελέστηκε βοηθάει στην επίτευξη του στόχου του πράκτορα, ενώ τιμωρία όταν συμβαίνει το αντίθετο. Επίσης το περιβάλλον στέλνει στον πράκτορα την κατάσταση στην οποία βρίσκεται. Έτσι ο πράκτορας μαθαίνει να υπολογίζει ποιες είναι οι σωστές ενέργειες σε κάθε κατάσταση βάση των αμοιβών που παίρνει.

Ο πράκτορας κάθε φορά έχει δύο επιλογές, να κάνει εκμετάλλευση ή εξερεύνηση. Στην εκμετάλλευση επιλέγει να εκτελέσει μια ενέργεια που ξέρει από την μέχρι τώρα εμπειρία του ότι θα του επιφέρει μια καλή αμοιβή στην παρούσα κατάσταση, ενώ στην εξερεύνηση επιλέγει να εκτελέσει μια ενέργεια που δεν γνωρίζει τι αμοιβή θα του επιφέρει. Ο πράκτορας είναι καλό να κάνει και εκμετάλλευση και εξερεύνηση για να υπάρξουν ικανοποιητικά αποτελέσματα. Είναι καλό να κάνει μερικές φορές εξερεύνηση γιατί μπορεί να ανακαλύψει μια ενέργεια που θα του επιφέρει καλύτερη αμοιβή από αυτές που ήδη γνωρίζει από το ιστορικό του, αλλά δεν πρέπει να κάνει εξερεύνηση για πάντα. Η συχνότητα εξερεύνησης πρέπει να μικραίνει όσο περνάει ο καιρός καθώς αυτό εγγυάται καλύτερα αποτελέσματα και γρήγορη σύγκλιση.

2.3.2 Πολιτική ε-greedy

Ο πράκτορας ακολουθεί μια συγκεκριμένη πολιτική κατά τη διάρκεια της μάθησης του. Αυτή η πολιτική καθορίζει την ενέργεια που θα εκτελέσει σε κάθε κατάσταση που μπορεί να προκύψει.

Στην υλοποίηση αυτής της διπλωματικής εργασίας έχει χρησιμοποιηθεί η πολιτική ε-greedy. Στην πολιτική ε-greedy χρησιμοποιείται μια ισοζυγισμένη τακτική μεταξύ εκμετάλλευσης και εξερεύνησης. Πιο συγκεκριμένα, υπάρχει μια μικρή πιθανότητα ‘ε’ για εξερεύνηση, αλλιώς εκτελείται εκμετάλλευση. Αυτή η πιθανότητα ‘ε’ μικραίνει με την πάροδο του χρόνου έτσι ώστε να είναι εγγυημένη η επίτευξη ασύμπτωτης σύγκλισης.

2.3.3 Μέθοδος Χρονικών Διαφορών

Η Μέθοδος Χρονικών Διαφορών (Temporal Difference methods) χρησιμοποιείται στην ενισχυτική μάθηση για την πρόβλεψη του ποσού των αμοιβών σε κάθε κατάσταση [9]. Η ιδέα αυτής της μεθόδου είναι ότι η μάθηση εκφράζεται από τη διαφορά μεταξύ δύο διαδοχικών προβλέψεων του κριτή, και αυτή η διαφορά πρέπει να μειώνεται με την πάροδο του χρόνου. Συνεπώς έτσι μειώνεται η πιθανότητα μετά από μια επιτυχημένη ενέργεια του πράκτορα να ακολουθείται μια αποτυχημένη ενέργεια.

Στο Σχήμα 2.4 παρουσιάζεται ο κανόνας μάθησης Χρονικών Διαφορών για δύο χρονικές στιγμές. Εκφράζει την αξία μιας συγκεκριμένης κατάστασης σε μια χρονική στιγμή.

$$\mathbf{V(S_t) = V(S_t) + \eta\ [\gamma\ V(S_{t+1}) - V(S_t)]}$$

όπου: $V(S_t)$ είναι η εκτίμηση της αξίας για την κατάστασης S τη χρονική στιγμή t

$V(S_{t+1})$ είναι η εκτίμηση της αξίας για την κατάσταση S τη χρονική στιγμή t+1

η είναι ο ρυθμός μάθησης (learning rate)

γ είναι ο παράγοντας έκπτωσης (discount factor) για μελλοντικές αμοιβές

Σχήμα 2.4 Κανόνας μάθησης Χρονικών Διαφορών (TD-learning rule)

Ο παράγοντας έκπτωσης 'γ' μπορεί να πάρει τιμές από 0 μέχρι 1 και χρησιμοποιείται για να δώσει την επιθυμητή βαρύτητα σε μια αμοιβή. Δηλαδή, εάν θέλουμε να δώσουμε μεγαλύτερη βαρύτητα σε μια αμοιβή τότε η παράμετρος 'γ' θα έχει μια μεγάλη τιμή κοντά στο 1, ενώ εάν θέλουμε να δώσουμε μικρότερη βαρύτητα σε μια αμοιβή τότε η παράμετρος 'γ' θα έχει μια μικρή τιμή κοντά στο 0.

Η Μέθοδος Χρονικών Διαφορών έχει δύο τύπους συναρτήσεων αξίας:

- $V_\pi(s)$: Καθορίζει την αξία της κατάστασης 's' βάση μιας πολιτικής 'π'
- $Q_\pi(s,a)$: Καθορίζει την αξία εκτέλεσης μιας ενέργειας 'a' σε μια κατάσταση 's' βάση μιας πολιτικής 'π'

Η συνάρτηση αξίας μπορεί να διαφέρει σε κάθε αλγόριθμο. Σε αυτή τη Διπλωματική εργασία έχουν δοκιμαστεί δύο διαφορετικοί αλγόριθμοι Χρονικών Διαφορών, το Q-Learning [10] και το SARSA [11]. Η συνάρτηση αξίας για τον αλγόριθμο Q-Learning περιγράφεται στο Σχήμα 2.5 και για τον αλγόριθμο SARSA στο
Σχήμα 2.6. Ο αλγόριθμος SARSA ακολουθεί μια πολιτική (on-policy learning algorithm). Αρχίζει με μια τυχαία πολιτική και προσπαθεί να την κάνει καλύτερη με την πάροδο του χρόνου, καθώς επίσης μαθαίνει την αξία της πολιτικής του πράκτορα βάση των ενεργειών του. Ο αλγόριθμος Q-Learning δεν ακολουθεί μια πολιτική (off-policy learning algorithm). Χρησιμοποιεί οποιαδήποτε πολιτική για να υπολογίσει το 'Q' και μαθαίνει την αξία της βέλτιστης πολιτικής ανεξαρτήτως των ενεργειών του πράκτορα.

$$\mathbf{Q(s,a) = Q(s,a) + \alpha*[r' + \gamma*max\beta Q(s',\beta) - Q(s,a)]}$$

όπου: *Q(s,a)* είναι η αξία για την εκτέλεση μιας ενέργειας 'a' στην κατάσταση 's'
a είναι ο ρυθμός μάθησης
r είναι η αμοιβή
maxβQ(s',β) είναι η μεγαλύτερη αμοιβή στην κατάσταση s'

Σχήμα 2.5 Συνάρτηση αξίας αλγόριθμου Q-Learning

$$\mathbf{Q(s,a) = Q(s,a) + \alpha*[r' + \gamma*Q(s',a') - Q(s,a)]}$$

όπου: *Q(s,a)* είναι η αξία για την εκτέλεση μιας ενέργειας 'a' στην κατάσταση 's'
a είναι ο ρυθμός μάθησης
r είναι η αμοιβή
Q(s',a') είναι η αξία για την εκτέλεση μιας ενέργειας 'a'' στην κατάσταση 's''

Σχήμα 2.6 Συνάρτηση αξίας αλγόριθμου SARSA

2.4 Παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου»

Το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» ανήκει στην κατηγορία των παιγνίων γενικού αθροίσματος και μοντελοποιεί καταστάσεις πραγματικού κόσμου. Το παιχνίδι παίζεται σε πολλές επαναλήψεις και η λογική του βασίζεται σε δύο παίκτες που είναι αντίπαλοι και δρουν με βάση τα δικά τους συμφέροντα, με σκοπό να πάρουν την μεγαλύτερη δυνατή συνολική αμοιβή. Οι παίκτες έχουν μόνο δύο επιλογές σε κάθε επανάληψη, να συνεργαστούν ή να συνεχίσουν να επιμένουν. Ανάλογα με την επιλογή που κάνει ο κάθε παίκτης κάθε φορά, παίρνει την ανάλογη αμοιβή. Η υψηλότερη ολική αμοιβή δίνεται όταν συνεργαστούν και οι δύο παίκτες και η χαμηλότερη όταν επιμένουν και οι δύο. Όταν ο ένας επιλέξει συνεργασία και ο άλλος επιμονή τότε παίρνουν μια ενδιάμεση ολική αμοιβή.

Πιο συγκεκριμένα, οι αναλογίες των αμοιβών για κάθε επιλογή των παικτών καθώς και ο κανόνας που πρέπει να ισχύει για τις τιμές τους φαίνονται στο Σχήμα 2.7.

	Cooperate	**Defect**
Cooperate	R	S
Defect	T	P

Rules:
1. T>R>P>S
2. 2R>T+S

Σχήμα 2.7 Πίνακας αμοιβών για το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου»[6]

2.5 Εξελικτικές Στρατηγικές

Οι Εξελικτικές Στρατηγικές (Evolution Strategies) είναι μια υποκατηγορία των μεθόδων άμεσης αναζήτησης και βελτιστοποίησης, και ανήκουν στην κατηγορία των Εξελικτικών Αλγορίθμων (Evolutionary Algorithms) [5]. Ο συνήθης στόχος των Εξελικτικών Στρατηγικών είναι να βελτιστοποιηθούν κάποιες αντικειμενικές συναρτήσεις (objective functions) σε σχέση με ένα σύνολο αντικειμενικών παραμέτρων (object parameters) [6].

Οι Εξελικτικές Στρατηγικές προσομοιώνουν την εξέλιξη των ατόμων. Αυτό γίνεται εφικτό εφαρμόζοντας στη μοντελοποίηση "άτομα" τα οποία με το πέρασμα των γενεών θα έχουν όλο και πιο βελτιστοποιημένα χαρακτηριστικά. Πιο συγκεκριμένα, υπάρχει ένας αρχικός πληθυσμός από άτομα. Αυτά τα άτομα έχουν κάποιες παραμέτρους, τις οποίες εμείς θέτουμε ανάλογα με το στόχο μας, οι οποίες θα βοηθούν στην βελτιστοποίηση της αντικειμενικής συνάρτησης των ατόμων με το πέρασμα των γενεών. Χρησιμοποιούνται διαδικασίες όπως η μετάλλαξη, ο ανασυνδυασμός και η επιλογή [6]. Αυτές οι διαδικασίες εφαρμόζονται στα άτομα, με σκοπό κάθε καινούρια γενεά που θα παράγεται να περιέχει ένα πληθυσμό με άτομα πιο εξελιγμένα σε στοιχεία που εμείς επιλέγουμε. Στον αλγόριθμο αυτό εκτελούνται πολλές επαναλήψεις, και σε κάθε επανάληψη δημιουργείται μια καινούρια γενεά η οποία περιέχει ένα πληθυσμό με τα ισχυρότερα άτομα της προηγούμενης γενεάς. Μετά το τέλος των επαναλήψεων, θα έχουμε μια γενεά αποτελούμενη από άτομα με βελτιστοποιημένα τα χαρακτηριστικά που επιθυμούμε.

Ένα άτομο στις Εξελικτικές Στρατηγικές έχει την εξής δομή:

$$a := (y, s, F(y))$$

όπου 'y' είναι οι αντικειμενικές παράμετροι (object parameters), 's' είναι οι παράμετροι στρατηγικής (strategy parameters) και 'F(y)' είναι η αντικειμενική συνάρτηση (objective function) του ατόμου.

Υπάρχουν δύο τρόποι για επιλογή των ατόμων που θα αποτελούν τη νέα γενεά:

- comma-selection $(\mu/\rho, \lambda)$: Τα άτομα που θα αποτελούν την επόμενη γενεά επιλέγονται από τον πληθυσμό των απογόνων μόνο
- plus-selection $(\mu/\rho + \lambda)$: Τα άτομα που θα αποτελούν την επόμενη γενεά επιλέγονται από τον πληθυσμό των γονέων και των απογόνων

Το 'μ' υποδηλώνει τον αριθμό των γονέων, το 'λ' τον αριθμό των απογόνων που θα παραχθούν και το 'ρ' τον αριθμό ανάμιξης (πόσοι γονείς θα συμβάλουν στην παραγωγή ενός απογόνου) [5].

Κεφάλαιο 3

Σχεδιασμός και Υλοποίηση

3.1 Εισαγωγή

Σκοπός αυτής της Διπλωματικής Εργασίας είναι να μελετηθεί πως η προδέσμευση βοηθάει στον αυτοέλεγχο ενός οργανισμού κατά την εξέλιξη του. Για την εφαρμογή της μελέτης αυτής θα γίνει προσομοίωση της εξέλιξης ατόμων. Σε κάθε γενεά τα άτομα που την αποτελούν θα εκπαιδεύονται ώστε να έχουν αυτοέλεγχο χρησιμοποιώντας προδέσμευση. Βάση της θεωρίας της εξέλιξης που υποστηρίζει ότι επιβιώνει ο ισχυρότερος [12], τα άτομα με τις βέλτιστες συμπεριφορές αυτοελέγχου θα σχηματίζουν την επόμενη γενεά. Έτσι με την πάροδο των γενεών και αφού τα άτομα εκπαιδευτούν, θα παρουσιάζεται βελτιωμένη συμπεριφορά αυτοέλεγχου στο σύνολο.

3.2 Περιγραφή Σχεδιασμού και Υλοποίησης

3.2.1 Μοντελοποίηση Αυτοέλεγχου

3.2.1.1 Σχεδιασμός

Τα άτομα που αποτελούν μια γενεά θα εκπαιδεύονται να έχουν αυτοέλεγχο χρησιμοποιώντας προδέσμευση. Η εκπαίδευση για επίτευξη του αυτοέλεγχου θα γίνεται στον ‘εγκέφαλο’ των ατόμων, και πιο συγκεκριμένα στο άνω και κάτω μέρος του εγκεφάλου που αντιπροσωπεύουν την ορθολογιστική σκέψη και την ενστικτώδη συμπεριφορά αντίστοιχα. Ο εγκέφαλος κάθε ατόμου θα προσομοιώνεται αναπτύσσοντας ένα υπολογιστικό μοντέλο το οποίο θα περιέχει όλα τα βασικά χαρακτηριστικά έτσι ώστε να είναι βιολογικά και ψυχολογικά σχετικό, να αντιπροσωπεύει δηλαδή σε μεγάλο βαθμό τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου. Τα δύο μέρη του εγκεφάλου εκ φύσεως συναγωνίζονται για το ποιος θα έχει τον έλεγχο του οργανισμού. Σκοπός εδώ είναι να κατορθώσουν να συμβιβαστούν αυτά τα δύο μέρη, καταλήγοντας έτσι στον αυτοέλεγχο του οργανισμού. Αυτός ο ανταγωνισμός μπορεί να εφαρμοστεί στο μοντέλο με το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» (Iterated Prisoner’s Dilemma (IPD)) [1].

Ο εγκέφαλος θα εκπαιδεύεται να έχει αυτοέλεγχο χρησιμοποιώντας το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου». Αυτό το παιχνίδι παίζεται σε πολλές επαναλήψεις και η λογική του βασίζεται σε δύο παίκτες που είναι αντίπαλοι και δρουν με βάση τα δικά τους συμφέροντα, με σκοπό να πάρουν την μεγαλύτερη δυνατή συνολική αμοιβή. Οι παίκτες έχουν μόνο δύο επιλογές σε κάθε επανάληψη, να συνεργαστούν ή να συνεχίσουν να επιμένουν στα θέλω τους. Ανάλογα με την επιλογή που κάνει ο κάθε παίκτης κάθε φορά, παίρνει την ανάλογη αμοιβή. Συνεπώς η λογική αυτού του παιχνιδιού είναι πολύ κοντά στον τρόπο λειτουργίας του άνω και κάτω μέρους του εγκεφάλου, αφού αυτά τα δύο μέρη συναγωνίζονται μεταξύ τους για τον έλεγχο του οργανισμού. Επομένως κρίθηκε κατάλληλο για τη μοντελοποίηση αυτών των λειτουργιών του εγκεφάλου.

Το υπολογιστικό μοντέλο που προσομοιώνει τον εγκέφαλο θα αποτελείται από δύο πράκτορες οι οποίοι θα διαγωνίζονται στο παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου». Ο ένας πράκτορας θα αντιπροσωπεύει το άνω μέρος του εγκεφάλου και

ο άλλος το κάτω μέρος του εγκεφάλου. Αυτοί οι πράκτορες θα αλληλοεπιδρούν με το περιβάλλον τους λαμβάνοντας πληροφορίες για το ιστορικό των καταστάσεων του και θα στέλνουν στο περιβάλλον την ενέργεια που αποφασίζουν να εκτελέσουν βάση αυτών των καταστάσεων. Ακολούθως το περιβάλλον στέλνει ένα ενισχυτικό σήμα στους πράκτορες το οποίο περιέχει την αμοιβή του κάθε πράκτορα βάσει της ενέργειας που εκτέλεσε προηγουμένως. Οι αμοιβές ανάλογα με την πράξη κάθε παίκτη θα είναι αυτές που φαίνονται στον πίνακα στο Σχήμα 3.1.

		Κάτω μέρος εγκεφάλου	
		Συμβιβασμός	Επιμονή
Άνω μέρος εγκεφάλου	Συμβιβασμός	4,4	-3,5 -ψ
	Επιμονή	5,-3 +ψ	-2,-2

Σχήμα 3.1 Πίνακας αμοιβών για το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» [4]

Στον πίνακα αμοιβών στο Σχήμα 3.1 εφαρμόζεται μια τιμή ψ, το differential bias. Αφού στόχος είναι η απόκτηση της αργοπορημένης αλλά μεγάλης ανταμοιβής, που αντιστοιχεί στην ορθολογιστική σκέψη, δηλαδή στις λειτουργίες του άνω μέρους του εγκεφάλου, όταν το κάτω μέρος του εγκεφάλου επιλέξει να συμβιβαστεί την στιγμή που το άνω μέρος έχει επιλέξει να επιμένει, δίνεται ένα επιπλέον ποσό στην αμοιβή (ψ) αφού είναι ένα βήμα προς την επίτευξη του στόχου μας. Στην αντίθετη περίπτωση που το κάτω μέρος επιμένει και το άνω μέρος συμβιβαστεί, γεγονός που είναι εναντίον στον στόχο μας, αφαιρείται ένα ποσό από την αμοιβή (ψ). Η τιμή που θα πάρει το ψ πρέπει να είναι μεταξύ 0 και 1 έτσι ώστε να ισχύουν οι κανόνες στο Σχήμα 2.7. Αφού σκοπός των παικτών είναι η απόκτηση όσο δυνατόν μεγαλύτερης αμοιβής, όσο αυξάνεται η αμοιβή όταν συμβιβάζεται το κάτω μέρος του εγκεφάλου τόσο αυξάνεται και η πιθανότητα συμβιβασμού του στο μέλλον. Συνεπώς, με την χρησιμοποίηση του differential bias παραπέμπουμε τις μελλοντικές επιλογές στην αργοπορημένη αλλά μεγάλη αμοιβή και επιτυγχάνεται η ενίσχυση της συνεργασίας. Αυτός είναι και ο σκοπός της προδέσμευσης. Με αυτό τον τρόπο το αποτέλεσμα της προδέσμευσης θα προσομοιωθεί στο υπολογιστικό μοντέλο [2].

Οι πράκτορες θα υλοποιηθούν ως πίνακες στους οποίους θα εφαρμόζεται ενισχυτική μάθηση έτσι ώστε να εκπαιδεύονται, και πιο συγκεκριμένα ο αλγόριθμος Χρονικών Διαφορών. Ο αλγόριθμος Χρονικών Διαφορών διατηρεί ένα ιστορικό των προηγούμενων αμοιβών και είναι σε θέση να προβλέπει τις μελλοντικές αμοιβές του πράκτορα (Barto, 2007). Μπορεί να εφαρμοστεί με δύο μεθόδους, Q-Learning [10] ή SARSA [11]. Με αυτές τις μεθόδους μπορεί να εκτιμηθεί, βάσει του ιστορικού των κινήσεων του πράκτορα, η αξία εάν εκτελέσει μια συγκεκριμένη πράξη, βάσει της κατάστασης στην οποία βρίσκεται. Ο πράκτορας όμως μπορεί να μην επιλέξει να εκτελέσει την καλύτερη εκτιμώμενη κίνηση βάσει του ιστορικού του, δηλαδή τακτική εκμετάλλευσης, αλλά να εξερευνήσει εκτελώντας μια κίνηση που δεν γνωρίζει τι αμοιβή θα του επιφέρει. Αυτή η αμοιβή μπορεί να είναι καλύτερη από την περίπτωση που θα ακολουθούσε την 'σίγουρη' τακτική ή χειρότερη. Η εξερεύνηση θα γίνεται από τον πράκτορα με μια πιθανότητα ε, η οποία θα μειώνεται με την πάροδο του χρόνου. Αυτή η μέθοδος για επιλογή κίνησης μέσω εξερεύνησης ή εκμετάλλευσης ονομάζεται e-greedy και θα εφαρμοστεί στο μοντέλο.

Στον υπολογισμό της αξίας μιας κίνησης υπάρχει μια παράμετρος, ο παράγοντας έκπτωσης της αξίας των αμοιβών 'γ'. Για τον πίνακα που αντιπροσωπεύει το άνω μέρος του εγκεφάλου η παράμετρος 'γ' θα έχει τιμή κοντά στο 1 καθιστώντας το έτσι πιο διορατικό και συνεπώς κοντά στην λογική της ορθολογιστικής σκέψης. Για τον πίνακα που αντιπροσωπεύει το κάτω μέρος του εγκεφάλου η παράμετρος 'γ' θα έχει τιμή κοντά στο 0 καθιστώντας το έτσι πιο παρορμητικό και συνεπώς κοντά στην λογική της ενστικτώδους συμπεριφοράς.

3.2.1.2 Υλοποίηση

Αρχικά δημιουργούνται οι δύο πράκτορες, οι οποίοι προγραμματιστικά είναι αντικείμενα που έχουν τα εξής χαρακτηριστικά:

- ρυθμός μάθησης (learning rate) : καθορίζει πόσο γρήγορα μαθαίνει ο πράκτορας
- παράγοντας έκπτωσης (discount factor) : καθορίζει την αξία που δίνει ο πράκτορας στις μελλοντικές αμοιβές

- ε (epsilon) : τιμή που καθορίζει εάν ο πράκτορας θα κάνει εκμετάλλευση ή εξερεύνηση
- τρέχουσα ενέργεια (current action) : η ενέργεια που επέλεξε να κάνει ο πράκτορας τη τρέχουσα στιγμή
- πίνακας Q : Είναι το ιστορικό του πράκτορα. Περιέχει τις τιμές Q(s,a) που υπολογίζονται μετά από κάθε ενέργεια του
- πίνακας αμοιβών: περιέχει την αμοιβή που θα πάρει ο πράκτορας βάση της ενέργειας που εκτέλεσε

Το αντικείμενο που αντιπροσωπεύει τον πράκτορα είναι η κλάση Player.java στο Παράρτημα Α.

Αφού δημιουργηθούν οι πράκτορες, θέτονται οι κατάλληλες τιμές στις παραμέτρους τους (Παράρτημα Α, IPD_Game.java, 130-144). Ο παράγοντας έκπτωσης παίρνει μια τιμή κοντά στο 1 για τον πράκτορα που αντιπροσωπεύει το άνω μέρος του εγκεφάλου, καθιστώντας το έτσι πιο διορατικό και συνεπώς κοντά στην λογική της ορθολογιστικής σκέψης. Για τον πράκτορα που αντιπροσωπεύει το κάτω μέρος του εγκεφάλου ο παράγοντας έκπτωσης έχει τιμή κοντά στο 0, καθιστώντας το έτσι πιο παρορμητικό και συνεπώς κοντά στην λογική της ενστικτώδους συμπεριφοράς. Ο πίνακας αμοιβών κάθε πράκτορα παίρνει τις τιμές που φαίνονται στο Σχήμα 3.1 και η παράμετρος ε μια τιμή πολύ μικρή ώστε να μην εκτελείται εξερεύνηση σε μεγάλο βαθμό.

Όταν αρχικοποιηθούν οι δύο πράκτορες, αρχίζουν να εκτελούνται τα επεισόδια του παιχνιδιού (Παράρτημα Α, IPD_Game.java, 154). Σε κάθε επεισόδιο ο κάθε παίκτης αρχικά επιλέγει μια ενέργεια που θα εκτελέσει και βάσει αυτών των ενεργειών θέτεται η τρέχουσα κατάσταση του παιχνιδιού (Παράρτημα Α, IPD_Game.java, 155-163). Ακολούθως μέσω της συνάρτησης update_Q υπολογίζεται η αξία Q(s,a) για κάθε παίκτη και ενημερώνονται οι πίνακες Q (Παράρτημα Α, IPD_Game.java, 165-168; Player.java, 101-116). Αυτή η συνάρτηση είναι που καθορίζει εάν ο αλγόριθμος Χρονικών Διαφορών που εφαρμόζεται είναι ο Q-Learning ή ο SARSA. Στο Παράρτημα Β όπου παρατίθεται ο πηγαίος κώδικας για τη προσομοίωση που χρησιμοποιεί τον αλγόριθμο SARSA, μπορούμε να δούμε την διαφορά ανάμεσα στους δύο αλγόριθμους στην συνάρτηση update_Q (Παράρτημα Β, Player.java, 114-124). Τέλος, αφού εκτελεστούν όλα τα επεισόδια υπολογίζεται η συνολική αμοιβή κάθε παίκτη καθώς και οι επισκέψεις που

έγιναν σε κάθε κατάσταση. Με αυτά τα στατιστικά στοιχεία μπορούμε να παρατηρήσουμε πόσες φορές συνεργάστηκαν τα δύο μέρη του εγκεφάλου έτσι ώστε να επιτυγχάνεται αυτοέλεγχος (Παράρτημα Α, IPD_Game.java, 179, 77-112).

3.2.2 Εφαρμογή Εξέλιξης στο Μοντέλο

3.2.2.1 Σχεδιασμός

Το σύστημα όπως αναφέρθηκε και προηγουμένως θα έχει εξελικτικό χαρακτήρα, θα υπόκειται δηλαδή σε προσομοίωση της γενετικής εξέλιξης με μάθηση. Η εφαρμογή της γενετικής εξέλιξης θα γίνει χρησιμοποιώντας ένα εξελικτικό αλγόριθμο, τις Εξελικτικές Στρατηγικές (Evolution Strategies) [6]. Μέσω αυτών των στρατηγικών συμπεριλαμβάνονται στη μοντελοποίηση "άτομα" τα οποία θα υπόκεινται σε συγκεκριμένες διαδικασίες, οι οποίες θα έχουν ως αποτέλεσμα κάθε γενεά να περιέχει άτομα με πιο βελτιστοποιημένα χαρακτηριστικά από αυτά της προηγούμενης γενεάς, τα οποία είναι σχετικά με την επίτευξη αυτοέλεγχου.

Στις εξελικτικές στρατηγικές δημιουργείται ένας αρχικός πληθυσμός από άτομα. Αυτά τα άτομα έχουν κάποιες αντικειμενικές παραμέτρους οι οποίες θα προσαρμόζονται ανάλογα με το πέρασμα των γενεών έτσι ώστε να βοηθούν στην επίτευξη του αυτοέλεγχου. Οι αντικειμενικές παράμετροι των ατόμων είναι το differential bias που αναφέρθηκε πιο πάνω, ο ρυθμός έκπτωσης 'γ' και ο ρυθμός μάθησης 'η', και έχουν επίσης μια στρατηγική παράμετρο το 'σ'. Η προσαρμογή των αντικειμενικών παραμέτρων των ατόμων γίνεται μέσω της διαδικασίας της μετάλλαξης. Κατά τη διαδικασία της μετάλλαξης προστίθεται στην τρέχουσα τιμή της αντικειμενικής παραμέτρου μια γκαουσιανή τιμή που παράγεται χρησιμοποιώντας την στρατηγική παράμετρο 'σ'. Στα "άτομα" θα εκτελούνται δύο βασικές διαδικασίες με σκοπό να δημιουργούνται καινούριες γενεές αποτελούμενες από άτομα που έχουν στοιχεία πιο κοντινά στον στόχο μας. Αυτές οι διαδικασίες είναι η μεταλλαγή και η επιλογή [5]. Κάθε άτομο θα καλείται να παίξει το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου». Τα άτομα που θα έχουν λάβει τις μεγαλύτερες αμοιβές, δηλαδή αυτά που παρουσίασαν μεγαλύτερο αυτοέλεγχο (επιτυχής συνεργασία των δύο μερών του

εγκεφάλου), θεωρούνται ως τα ισχυρότερα και για αυτό τον λόγο θα συμπεριληφθούν στην επόμενη γενεά. Μετά από ένα αριθμό επαναλήψεων αυτών των διαδικασιών καταλήγουμε σε μια γενεά η οποία αποτελείται από άτομα που παρουσιάζουν βέλτιστη συμπεριφορά αυτοέλεγχου χρησιμοποιώντας προδέσμευση.

Κάθε άτομο στα πλαίσια των Εξελικτικών Στρατηγικών έχει μια Αντικειμενική Συνάρτηση η οποία καθορίζει την ισχύ (fitness) του ατόμου. Η Αντικειμενική Συνάρτηση στις Εξελικτικές Στρατηγικές βελτιστοποιείται με το πέρασμα των γενεών έτσι ώστε να παράγονται καλύτερες λύσεις κάθε φορά. Αυτή η αντικειμενική συνάρτηση αποτελεί κριτήριο στην επιλογή των ισχυρότερων ατόμων από τον πληθυσμό για να σχηματίσουν την επόμενη γενεά, σύμφωνα με τον νόμο της εξέλιξης που υποστηρίζει ότι πάντα επιβιώνει ο ισχυρότερος. Αφού το θέμα αυτής της μελέτης είναι η βελτίωση της συμπεριφοράς του αυτοέλεγχου, η αντικειμενική συνάρτηση των ατόμων πρέπει να σχετίζεται άμεσα με τον αυτοέλεγχο ο οποίος θα βελτιώνεται σε κάθε γενεά. Συνεπώς η αντικειμενική συνάρτηση των ατόμων θα αντιπροσωπεύει το επίπεδο επίτευξης του αυτοέλεγχου τους.

3.2.2.2 Υλοποίηση

Αρχικά δημιουργείται το αντικείμενο που αντιπροσωπεύει τις Εξελικτικές Στρατηγικές (Παράρτημα Α, EvolutionStrategies.java). και ανατίθενται τιμές στις παραμέτρους των Εξελικτικών Στρατηγικών (‘λ’, ‘μ’, ‘ρ’, ‘σ’) (Παράρτημα Α, MainProgram.java, 147-153). Επίσης δημιουργείται το αντικείμενο που αντιπροσωπεύει το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» (Παράρτημα Α, IPD_Game.java). και ανατίθενται τιμές στις παραμέτρους του παιχνιδιού) (Παράρτημα Α, MainProgram.java, 155-159).

Στη συνέχεια, αφού ανατεθούν οι επιθυμητές τιμές στις πιο πάνω παραμέτρους, αρχίζει η διαδικασία της εξέλιξης όπου θα εκπαιδεύονται τα άτομα για να παρουσιάζουν βελτιωμένη συμπεριφορά αυτοελέγχου σε κάθε νέα γενεά που δημιουργείται χρησιμοποιώντας προδέσμευση. Το αντικείμενο που αντιπροσωπεύει το κάθε άτομο είναι η κλάση Individual.java στο Παράρτημα Α. Αρχικά δημιουργείται ένας πληθυσμός

με άτομα όπως αυτά που αναφέρθηκαν πιο πάνω και γίνεται αρχικοποίηση των αντικειμενικών παραμέτρων τους (Παράρτημα Α, MainProgram.java, 162; EvolutionStrategies.java, 109-130). Ακολούθως, για κάθε καινούρια γενεά, δημιουργείται ένας αριθμός απογόνων (Παράρτημα Α, MainProgram.java, 207; EvolutionStrategies.java, 136-152) και εκτελείται μετάλλαξη στις αντικειμενικές παραμέτρους τους προσθέτοντας στην τρέχουσα τιμή τους μια γκαουσιανή τιμή η οποία παράγεται με την χρήση της στρατηγικής παραμέτρου ‘σ’ (Παράρτημα Α, MainProgram.java, 210; EvolutionStrategies.java, 155-249). Οι απόγονοι που δημιουργούνται είναι κλώνοι των γονέων, συνεπώς δημιουργείται ο απαραίτητος αριθμός αντιγράφων του κάθε ατόμου από τον πληθυσμό των γονέων (Παράρτημα Α, EvolutionStrategies.java, 133-152). Κάθε απόγονος παίζει το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» έτσι ώστε να εκπαιδευτεί ο εγκέφαλος του να έχει βελτιωμένη συμπεριφορά αυτοελέγχου (Παράρτημα Α, MainProgram.java, 212-225; IPD_Game.java, 115-187). Τα αποτελέσματα του κάθε ατόμου από αυτό το παιχνίδι χρησιμοποιούνται για τον υπολογισμό της αντικειμενικής τους συνάρτησης (Παράρτημα Α, MainProgram.java, 218-219; EvolutionStrategies.java, 176-249).. Τέλος, γίνεται επιλογή των ισχυρότερων ατόμων, δηλαδή αυτών με τον περισσότερο αυτοέλεγχο, βάση της μεθόδου plus-selection ($\mu/\rho + \lambda$) που θα αποτελούν τον πληθυσμό της επόμενης γενεάς (Παράρτημα Α, MainProgram.java, 227-228; EvolutionStrategies.java, 266-294). Η σύγκριση των αντικειμενικών συναρτήσεων των ατόμων έτσι ώστε να καθοριστούν οι ισχυρότεροι γίνεται με την κλάση FitnessComparator.java όπως φαίνεται στο Παράρτημα Α.

Κεφάλαιο 4

Αποτελέσματα και Συζήτηση

4.1 Πειράματα και Αποτελέσματα

4.1.1 Μοντελοποίηση Αυτοέλεγχου

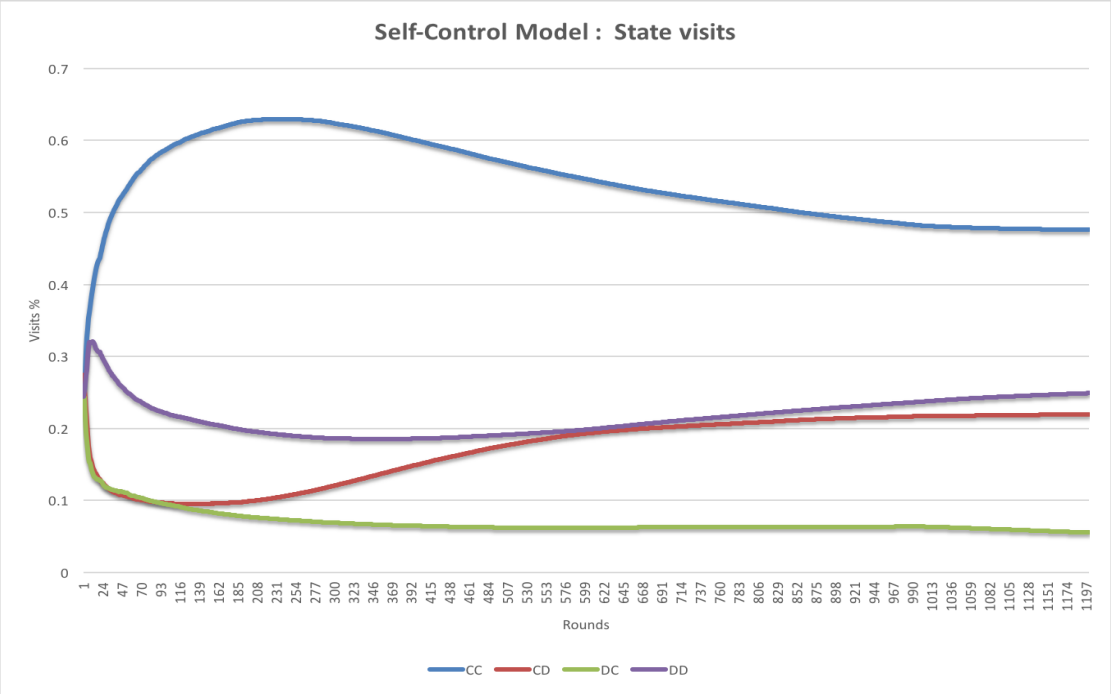
Για την υλοποίηση του συστήματος αυτής της Διπλωματικής Εργασίας, το οποίο προσομοιώνει την εξέλιξη ατόμων τα οποία εκπαιδεύονται και παρουσιάζουν βελτιωμένη συμπεριφορά αυτοελέγχου σε κάθε καινούρια γενεά, αρχικά είχε σχεδιαστεί και υλοποιηθεί το βασικό μοντέλο για μοντελοποίηση του αυτοέλεγχου, που προσομοιώνει τις άνω και κάτω λειτουργίες του εγκεφάλου οι οποίες σχετίζονται με την εξάσκηση και επίτευξη του αυτοέλεγχου του οργανισμού χρησιμοποιώντας προδέσμευση. Για την προσομοίωση του ανταγωνισμού αυτών των λειτουργιών του εγκεφάλου έχει χρησιμοποιηθεί το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του

Φυλακισμένου» στο οποίο υπάρχουν δύο παίκτες, ο ένας παίκτης αντιπροσωπεύει τις άνω λειτουργίες του εγκεφάλου και ο άλλος τις κάτω λειτουργίες του εγκεφάλου, οι οποίοι συναγωνίζονται για τα δικά τους συμφέροντα με σκοπό να λάβουν την μεγαλύτερη δυνατή συνολική αμοιβή. Οι αμοιβές που λαμβάνει ο κάθε παίκτης ανάλογα με την ενέργεια που εκτελεί είναι αυτές που φαίνονται στο Σχήμα 3.1. Οι δύο παίκτες στο παιχνίδι εκπαιδεύονται να έχουν βελτιωμένη συμπεριφορά αυτοέλεγχου χρησιμοποιώντας Ενισχυτική Μάθηση, και πιο συγκεκριμένα τον αλγόριθμο Χρονικών Διαφορών [9]. Ο κανόνας μάθησης του αλγόριθμου Χρονικών Διαφορών φαίνεται στο Σχήμα 2.4. Ο αλγόριθμος Χρονικών Διαφορών μπορεί να εφαρμοσθεί με δύο μεθόδους, SARSA και Q-Learning. Η επίδοση των δύο παικτών, και συνεπώς η ικανότητα μάθησης τους, επηρεάζεται τόσο από τον κανόνα μάθησης που θα χρησιμοποιηθεί όσο και από τις τιμές που θα ανατεθούν στις παραμέτρους ‘γ’, ‘η’ και ‘ε’. Στο μοντέλο έχουν δοκιμαστεί και οι δύο αλγόριθμοι μάθησης έτσι ώστε να συγκρίνουμε τα αποτελέσματα τους καθώς και διάφορες τιμές στις παραμέτρους. Τα αποτελέσματα που παρουσιάζονται σε αυτό το υποκεφάλαιο (4.1.1) είναι αποκλειστικά από την μοντελοποίηση του αυτοέλεγχου μόνο χρησιμοποιώντας το βασικό μοντέλο που αναφέρθηκε πιο πάνω. Αυτά τα αποτελέσματα δηλαδή είναι πριν εφαρμοστεί η έννοια της εξέλιξης στο σύστημα, δηλαδή δεν εφαρμόζεται η εξέλιξη σε ένα πληθυσμό από άτομα και δεν δημιουργούνται καινούριες γενεές, απλά γίνεται προσομοίωση της εκπαίδευσης ενός μόνο εγκεφάλου έτσι ώστε να έχει βελτιωμένη συμπεριφορά αυτοελέγχου. Παρουσιάζονται αυτά τα αποτελέσματα αρχικά έτσι ώστε να συγκριθούν με τα αποτελέσματα όπου θα γίνεται η εφαρμογή της προδέσμευσης και της έννοιας της εξέλιξης.

4.1.1.1 Αλγόριθμος Εκπαίδευσης Πρακτόρων

Ο αλγόριθμος Χρονικών Διαφορών που χρησιμοποιείται για την εκπαίδευση των δύο παικτών μπορεί να εφαρμοσθεί με δύο μεθόδους, SARSA και Q-Learning. Με τη χρήση αυτών των μεθόδων μπορεί να εκτιμηθεί η αξία εκτέλεσης μιας ενέργειας από τον πράκτορα βάση της κατάστασης στην οποία βρίσκεται το περιβάλλον. Χρησιμοποιώντας αυτές τις συναρτήσεις αξίας ο κάθε παίκτης αναγνωρίζει ποια ενέργεια μπορεί να του αποφέρει την επιθυμητή αμοιβή, και ως αποτέλεσμα εκπαιδεύεται να λειτουργεί με βάση την απόκτηση μεγαλύτερης αμοιβής. Η συνάρτηση αξίας για τον αλγόριθμο Q-Learning

περιγράφεται στο Σχήμα 2.5 και για τον αλγόριθμο SARSA στο Σχήμα 2.6. Αυτό που διαφοροποιεί τον ένα αλγόριθμο από τον άλλο είναι ότι στο SARSA η συνάρτηση αξίας λαμβάνει υπόψη τη συνάρτηση αξίας της επόμενης χρονικής στιγμής (t+1), δηλαδή της επόμενης ενέργειας που θα εκτελέσει ο πράκτορας στην επόμενη κατάσταση του παιχνιδιού, ενώ στο Q-Learning η συνάρτηση αξίας λαμβάνει υπόψη την βέλτιστη ενέργεια που μπορεί να εκτελέσει ο πράκτορας στην επόμενη κατάσταση του παιχνιδιού. Στο βασικό μοντέλο όπως αναφέρθηκε και προηγουμένως, έχουν δοκιμαστεί και οι δύο αλγόριθμοι μάθησης έτσι ώστε να συγκρίνουμε τα αποτελέσματα τους.

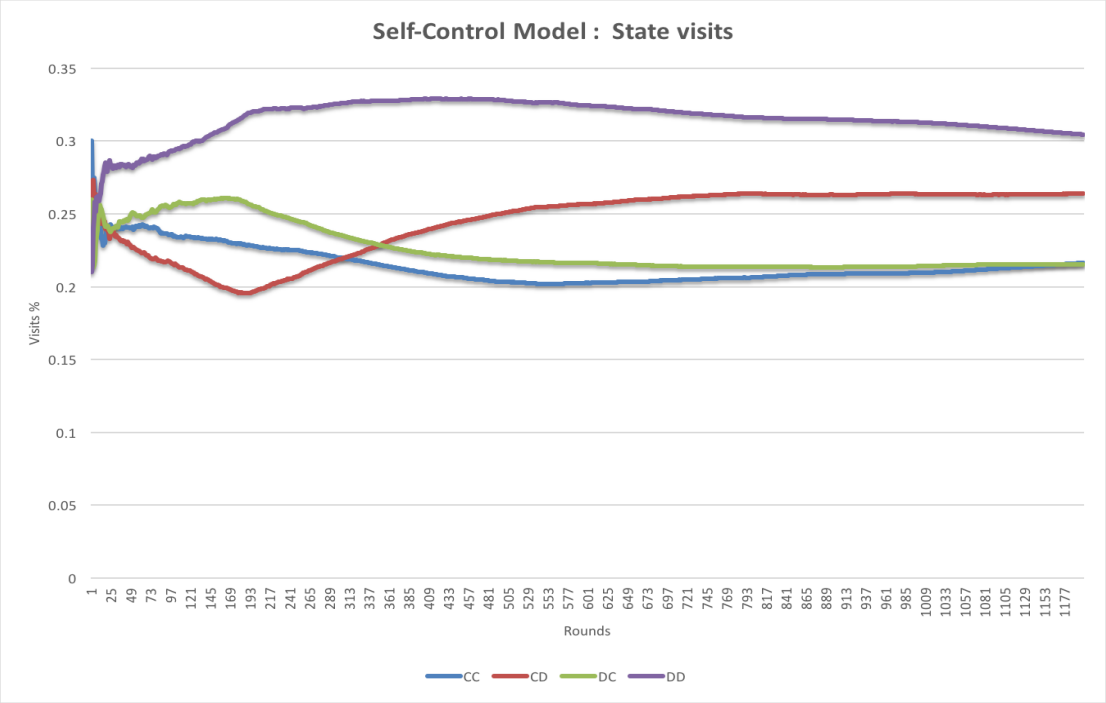


Σχήμα 4.1 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning

	Παίκτης 1 (άνω μέρος εγκεφάλου)	Παίκτης 2 (κάτω μέρος εγκεφάλου)
Παράγοντας Έκπτωσης (γ)	0.9	0.1
Ρυθμός Μάθησης (η)	0.1	0.1
ε	0.1	0.1

Πίνακας 4.1 Τιμές Παραμέτρων για τη γραφική παράσταση στο Σχήμα 4.1

Τα αποτελέσματα των εκτελέσεων για τους δύο αλγόριθμους, Q-Learning και SARSA, φαίνονται στις γραφικές παραστάσεις των σχημάτων 4.1 και 4.2 αντίστοιχα. Στις δύο αυτές εκτελέσεις χρησιμοποιήθηκαν ίδιες τιμές στις παραμέτρους των παικτών (‘γ’, ‘η’, ‘ε’), συνεπώς η μόνη διαφορά είναι ο αλγόριθμος μάθησης.



Σχήμα 4.2 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA.

	Παίκτης 1 (άνω μέρος εγκεφάλου)	Παίκτης 2 (κάτω μέρος εγκεφάλου)
Παράγοντας Έκπτωσης (γ)	0.9	0.1
Ρυθμός Μάθησης (η)	0.1	0.1
ε	0.1	0.1

Πίνακας 4.2 Τιμές Παραμέτρων για τη γραφική παράσταση στο Σχήμα 4.2

Από τα αποτελέσματα μπορούμε να συμπεράνουμε ότι ο αλγόριθμος Q-Learning επιτυγχάνει ευκολότερα την εκπαίδευση των παικτών παρά ο αλγόριθμος SARSA αφού

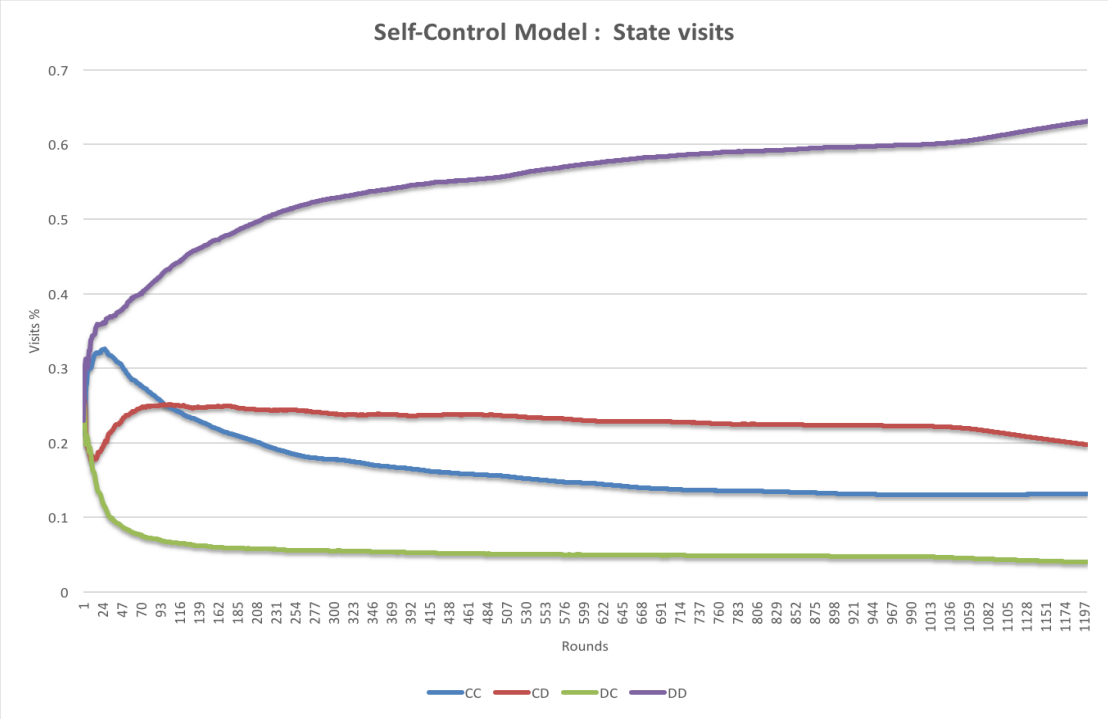
η διαφορά στα αποτελέσματα προκύπτει μόνο από τον αλγόριθμο μάθησης. Όπως φαίνεται στη γραφική παράσταση στο Σχήμα 4.1, η κατάσταση ‘CC’ έχει το μεγαλύτερο ποσοστό επισκέψεων από όλες τις καταστάσεις του παιχνιδιού. Συνεπώς αφού η κατάσταση ‘CC’ αντιπροσωπεύει την επιλογή συνεργασίας και από τους δύο παίκτες, οι δύο παίκτες επέλεξαν να συνεργαστούν τις πλείστες φορές κατά τη διάρκεια του παιχνιδιού. Αυτός είναι και ο σκοπός της εκπαίδευσης που εκτελείται για την επίτευξη του αυτοέλεγχου.

4.1.1.2 Τιμές Παραμέτρων

Από τις διάφορες εκτελέσεις που έγιναν παρατηρήθηκε ότι και στους δύο αλγόριθμους (SARSA, Q-Learning) οι δύο πράκτορες δεν κατορθώνουν πάντα να εκπαιδεύονται επαρκώς έτσι ώστε να παρουσιάζουν βελτιωμένη συμπεριφορά αυτοελέγχου. Αυτό που καθορίζει εάν θα μάθουν να έχουν αυτοέλεγχο μέχρι το τέλος της εκπαίδευσης τους είναι οι τιμές που δίνουμε στις παραμέτρους των παικτών. Αυτές οι παράμετροι είναι ο παράγοντας έκπτωσης (γ), ο ρυθμός μάθησης (η) και το ε. Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, ο παράγοντας έκπτωσης (γ) καθορίζει την αξία που δίνει ο παίκτης σε μελλοντικές αμοιβές, ο ρυθμός μάθησης (η) καθορίζει τον ρυθμό με τον οποίο μαθαίνει ο παίκτης, και το (ε) είναι μια τιμή η οποία καθορίζει εάν ο παίκτης θα εκτελέσει εξερεύνηση για εκτέλεση μιας ενέργειας που δεν γνωρίζει τι αμοιβή θα του αποφέρει ή εκμετάλλευση των όσων γνωρίζει ως τώρα για εκτέλεση μιας ενέργειας που θα του φέρει μια καλή αμοιβή.

Έχουν γίνει αρκετές εκτελέσεις του προγράμματος μοντελοποίησης αυτοέλεγχου του ανθρώπινου εγκεφάλου, δοκιμάζοντας διάφορες τιμές στις τρείς παραμέτρους που αναφέρθηκαν πιο πάνω. Όπως παρατηρείται στις γραφικές παραστάσεις των σχημάτων 4.1, 4.2, 4.3 και 4.4, οι παίκτες μαθαίνουν να συνεργάζονται στα πλαίσια του παιχνιδιού όταν ο ρυθμός μάθησης παίρνει μια πολύ μικρή τιμή, κοντά στο 0. Το εύρος τιμών όλων των παραμέτρων είναι από 0 μέχρι 1. Όταν δοθεί στο ρυθμό μάθησης μια τιμή κοντά στο 1 τότε οι πράκτορες δεν εκπαιδεύονται όπως αναμένεται, με αποτέλεσμα να επιλέγουν να μην συνεργάζονται κατά την εκπαίδευση τους, δηλαδή το παιχνίδι να είναι τις περισσότερες φορές στην κατάσταση ‘DD’. Επίσης η παράμετρος ‘ε’ πρέπει να έχει τιμή

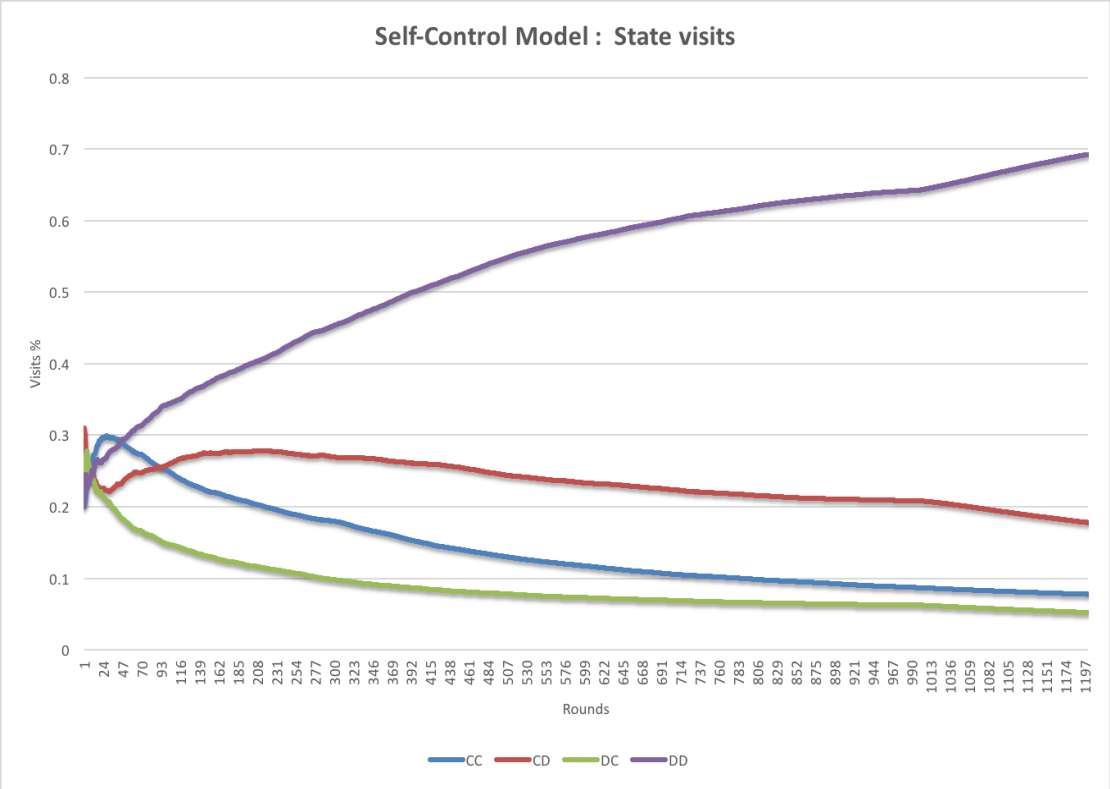
μικρότερη από 0.2 για να επιτυγχάνεται συχνή συνεργασία ανάμεσα στους δύο παίκτες. Αυτό είναι λογικό καθώς όσο πιο μικρή είναι η τιμή της παραμέτρου ‘ε’ τόσο λιγότερη εξερεύνηση θα γίνεται από τον πράκτορα, και συνεπώς θα υπάρχουν λιγότερες πιθανότητες να εκτελεί μια λανθασμένη πράξη που θα του επιφέρει τιμωρία αντί αμοιβή. Τα αποτελέσματα από τις διάφορες εκτελέσεις που έγιναν δοκιμάζοντας διάφορες τιμές στις παραμέτρους φαίνονται στα σχήματα 4.1, 4.2, 4.3, 4.4 και 4.5.



Σχήμα 4.3 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning

	Παίκτης 1 (άνω μέρος εγκεφάλου)	Παίκτης 2 (κάτω μέρος εγκεφάλου)
Παράγοντας Έκπτωσης (γ)	0.9	0.1
Ρυθμός Μάθησης (η)	0.9	0.9
ε	0.1	0.1

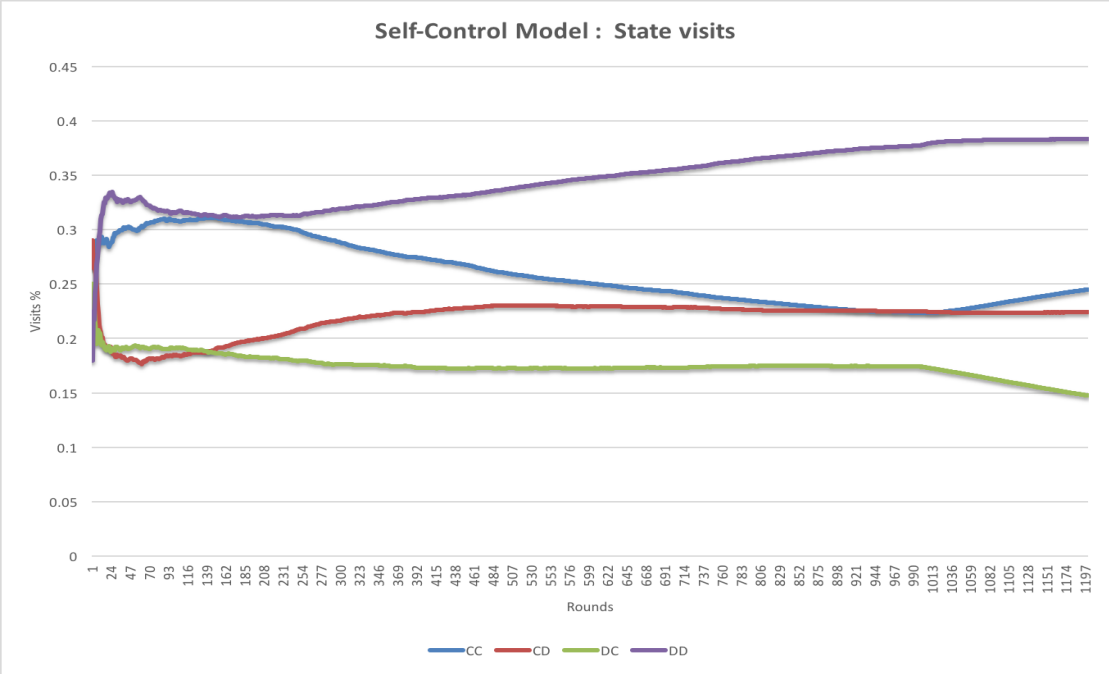
Πίνακας 4.3 Τιμές Παραμέτρων για τη γραφική παράσταση στο Σχήμα 4.3



Σχήμα 4.4 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA.

	Παίκτης 1 (άνω μέρος εγκεφάλου)	Παίκτης 2 (κάτω μέρος εγκεφάλου)
Παράγοντας Έκπτωσης (γ)	0.9	0.1
Ρυθμός Μάθησης (η)	0.9	0.9
ε	0.1	0.1

Πίνακας 4.4 Τιμές Παραμέτρων για τη γραφική παράσταση στο Σχήμα 4.4



Σχήμα 4.5 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning.

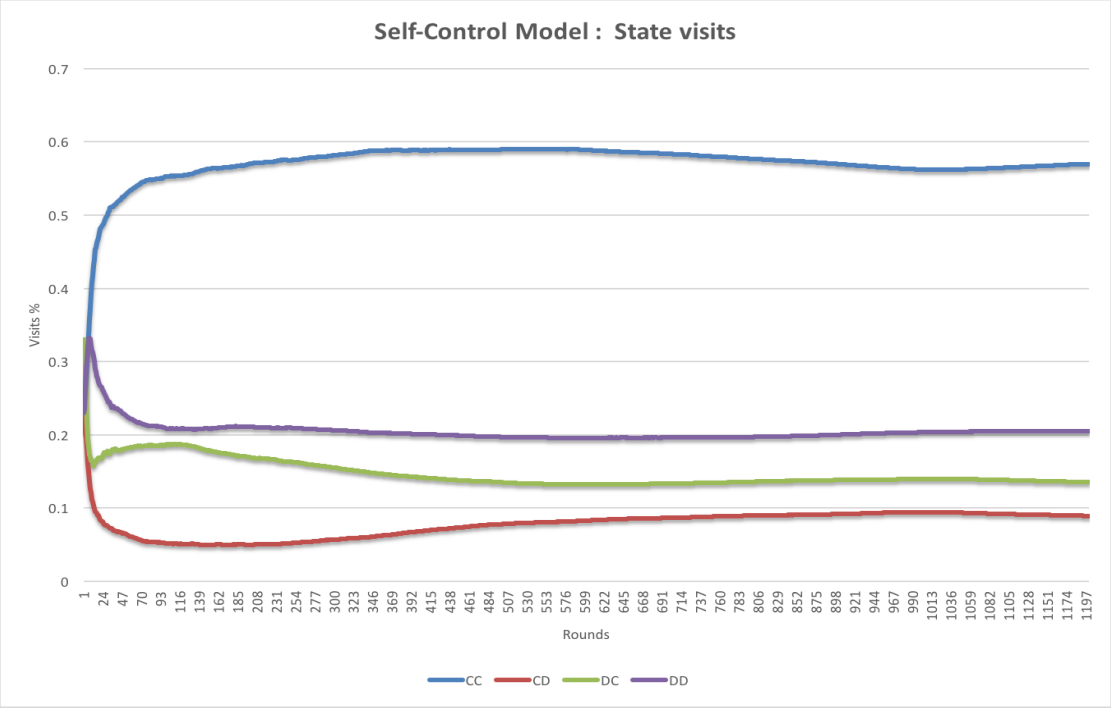
	Παίκτης 1 (άνω μέρος εγκεφάλου)	Παίκτης 2 (κάτω μέρος εγκεφάλου)
Παράγοντας Έκπτωσης (γ)	0.9	0.1
Ρυθμός Μάθησης (η)	0.1	0.1
ε	0.4	0.4

Πίνακας 4.5 Τιμές Παραμέτρων για τη γραφική παράσταση στο Σχήμα 4.5

4.1.1.3 Εφαρμογή Προδέσμευσης στο Μοντέλο

Μέρος των πειραμάτων που εκτελέστηκαν στο βασικό μοντέλο ήταν και η εφαρμογή του αποτελέσματος της προδέσμευσης στις αμοιβές των δύο πρακτόρων, έτσι ώστε να παρατηρηθεί η διαφορά στην εκπαίδευση των πρακτόρων πριν την εφαρμογή της προδέσμευσης και μετά. Τα αποτελέσματα μετά την εφαρμογή της προδέσμευσης στο μοντέλο για τον αλγόριθμο Q-Learning και SARSA φαίνονται στις γραφικές παραστάσεις των σχημάτων 4.6 και 4.7. Συγκρίνοντας τα σχήματα 4.1 και 4.2 όπου παρουσιάζονται τα αποτελέσματα πριν εφαρμοσθεί η προδέσμευση στο μοντέλο, με τα

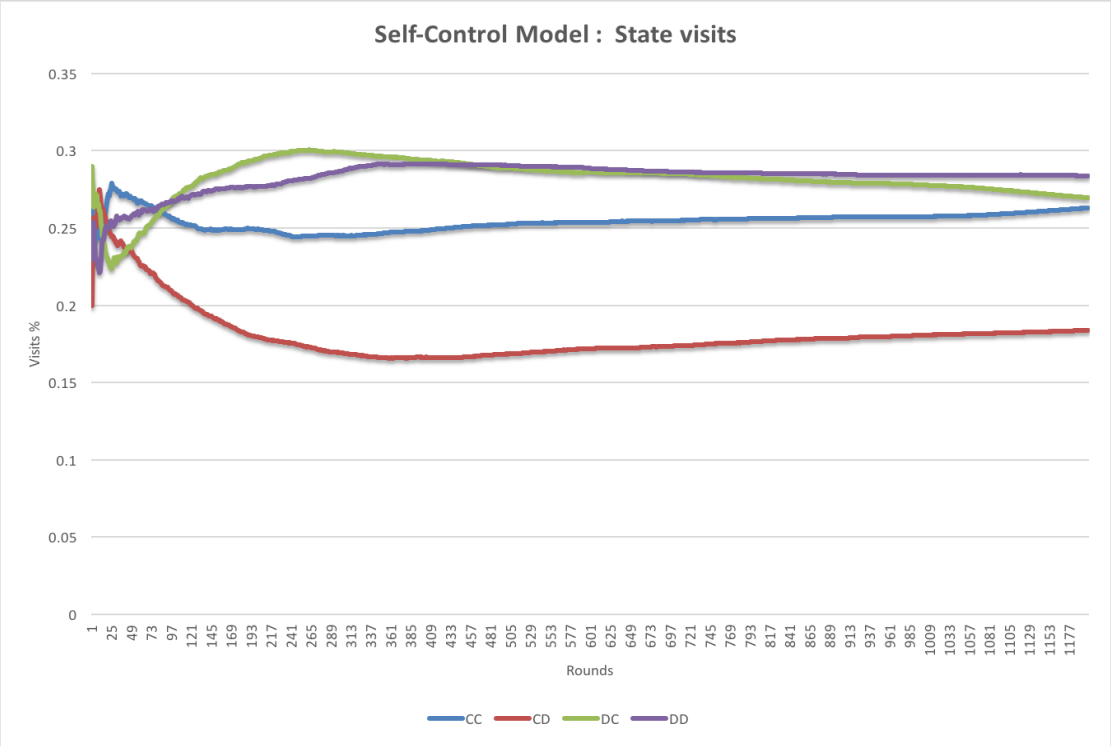
σχήματα 4.6 και 4.7 όπου παρουσιάζονται τα αποτελέσματα μετά την εφαρμογή της προδέσμευσης στο μοντέλο, μπορούμε να δούμε την εμφανή βελτίωση στην επίδοση των πρακτόρων όταν χρησιμοποιείται η προδέσμευση. Με την εφαρμογή της προδέσμευσης η συχνότητα συνεργασίας των δύο πρακτόρων αυξάνεται ως επί το πλείστον και κυμαίνεται σε παρόμοια επίπεδα με την πάροδο των επεισοδίων, σε αντίθεση με την περίπτωση όπου δεν εφαρμόζεται προδέσμευση όπου η συχνότητα μειωνόταν με την πάροδο των επεισοδίων του παιχνιδιού.



Σχήμα 4.6 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά μετά την εφαρμογή της προδέσμευσης με ψ=0.9. Αλγόριθμος: Q-Learning.

	Παίκτης 1 (άνω μέρος εγκεφάλου)	Παίκτης 2 (κάτω μέρος εγκεφάλου)
Παράγοντας Έκπτωσης (γ)	0.9	0.1
Ρυθμός Μάθησης (η)	0.1	0.1
ε	0.1	0.1

Πίνακας 4.6 Τιμές Παραμέτρων για τη γραφική παράσταση στο Σχήμα 4.6



Σχήμα 4.7 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά μετά την εφαρμογή της προδέσμευσης με ψ=0.9. Αλγόριθμος: SARSA.

	Παίκτης 1 (άνω μέρος εγκεφάλου)	Παίκτης 2 (κάτω μέρος εγκεφάλου)
Παράγοντας Έκπτωσης (γ)	0.9	0.1
Ρυθμός Μάθησης (η)	0.1	0.1
ε	0.1	0.1

Πίνακας 4.7 Τιμές Παραμέτρων για τη γραφική παράσταση στο Σχήμα 4.7

4.1.2 Εφαρμογή Εξέλιξης στο Μοντέλο

Σκοπός αυτής της Διπλωματικής Εργασίας είναι να μελετηθεί ο τρόπος με τον οποίο η προδέσμευση συμβάλει στον αυτοέλεγχο ενός οργανισμού κατά την εξέλιξη του. Όπως αναφέρθηκε και προηγουμένως, αρχικά είχε μοντελοποιηθεί η εξάσκηση του

αυτοέλεγχου ενός ατόμου. Δημιουργήθηκε ένα βασικό μοντέλο το οποίο αναπαριστούσε τον ανταγωνισμό ανάμεσα στις άνω και κάτω λειτουργίες του εγκεφάλου οι οποίες σχετίζονται με την εξάσκηση του αυτοέλεγχου ενός ατόμου. Αυτό το μοντέλο αποτελείται από δύο πράκτορες οι οποίοι αντιπροσωπεύουν αυτές τις λειτουργίες, και ανταγωνίζονται στο παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου». Οι αμοιβές που λαμβάνει ο κάθε παίκτης ανάλογα με την ενέργεια που εκτελεί είναι αυτές που φαίνονται στο Σχήμα 3.1. Αυτοί οι δύο πράκτορες εκπαιδεύονται χρησιμοποιώντας Ενισχυτική Μάθηση. Έχουν δοκιμαστεί δύο διαφορετικοί αλγόριθμοι για την εκπαίδευση τους έτσι ώστε να συγκρίνουμε τα αποτελέσματα τους, ο Q-Learning και ο SARSA. Όπως παρατηρήθηκε στα αποτελέσματα των πειραμάτων στο βασικό μοντέλο, δεν ήταν πάντα επιτυχής η εκπαίδευση των δύο παικτών για επίτευξη συχνής συνεργασίας μεταξύ τους. Η επίτευξη της συνεργασίας μεταξύ τους, και συνεπώς η επίτευξη του αυτοέλεγχου του ατόμου, εξαρτώταν από τον αλγόριθμο μάθησης που χρησιμοποιούσαμε στους πράκτορες καθώς και από τις τιμές των παραμέτρων τους (‘η’, ‘γ’, ‘ε’).

Αυτά τα αποτελέσματα φάνηκαν να βελτιώνονται και να επιτυγχάνεται βελτιωμένη συμπεριφορά αυτοελέγχου στα άτομα με την ενσωμάτωση της προδέσμευσης καθώς και της έννοιας της εξέλιξης στο βασικό μοντέλο. Η προδέσμευση ενσωματώνεται στο μοντέλο εφαρμόζοντας το αποτέλεσμα της στις αμοιβές των δύο πρακτόρων στο παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου». Προστίθεται δηλαδή μια τιμή, το differential bias, όπως φαίνεται στο Σχήμα 3.1. Η έννοια της εξέλιξης ενσωματώθηκε στο μοντέλο χρησιμοποιώντας ένα εξελικτικό αλγόριθμο, τις Εξελικτικές Στρατηγικές[4],[5]. Αυτό έχει ως αποτέλεσμα να συμπεριληφθούν στη μοντελοποίηση άτομα τα οποία θα εκπαιδεύονται να έχουν αυτοέλεγχο και κάθε γενεά που θα σχηματίζεται θα περιέχει άτομα με όλο και πιο βελτιωμένη συμπεριφορά αυτοέλεγχου. Οι παράμετροι που χαρακτήριζαν κάθε πράκτορα στο μοντέλο αυτοέλεγχου (η, γ, ψ), στο εξελικτικό μοντέλο θα αποτελούν τις αντικειμενικές παραμέτρους κάθε ατόμου, οι οποίες αρχικοποιούνται τυχαία στην έναρξη της προσομοίωσης και στη συνέχεια μεταβάλλονται οι τιμές τους μέσω της διαδικασίας της μετάλλαξης κατά τις Εξελικτικές Στρατηγικές. Σε αυτό το υποκεφάλαιο φαίνονται τα πειράματα που πραγματοποιήθηκαν για μελέτη της συμπεριφοράς του αυτοέλεγχου μέσω προδέσμευσης εν όσον εκτελείται εξέλιξη στο μοντέλο. Θα μελετηθεί ο τρόπος με τον οποίο κάθε αλγόριθμος μάθησης επηρεάζει την εκπαίδευση των ατόμων και μέχρι πιο βαθμό επιτυγχάνει τον αυτοέλεγχο τους.

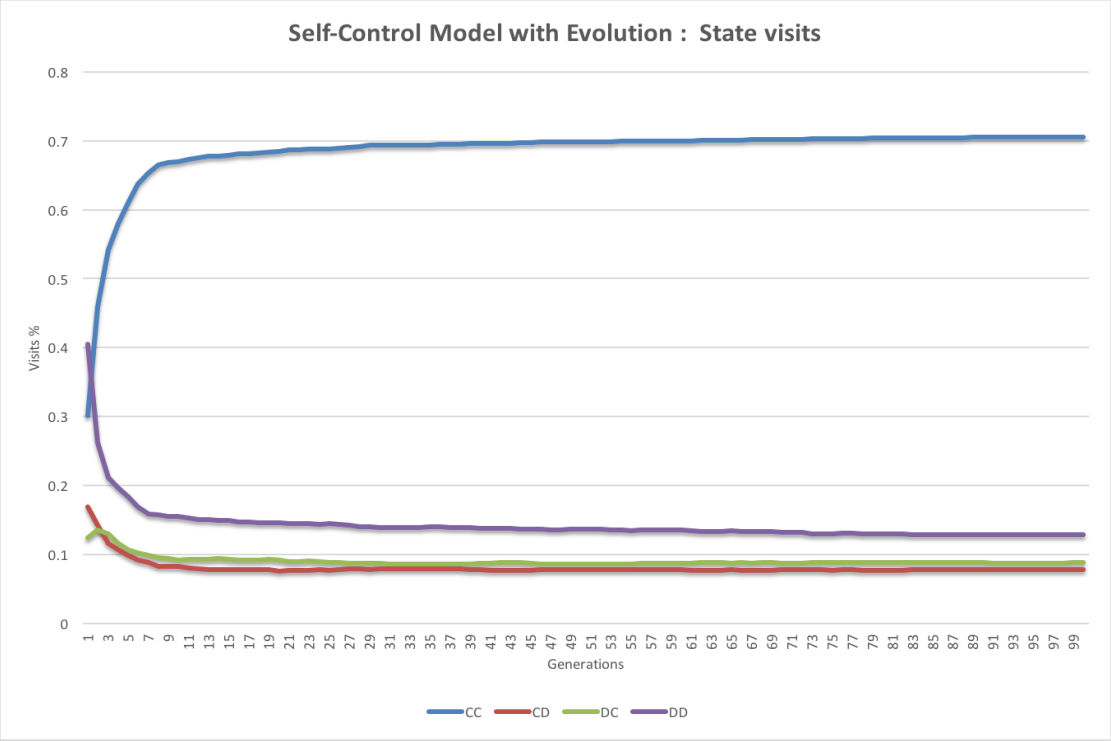
4.1.2.1 Αλγόριθμος Εκπαίδευσης Πρακτόρων

Ο αλγόριθμος Χρονικών Διαφορών που χρησιμοποιείται για την εκπαίδευση των δύο παικτών μπορεί να εφαρμοσθεί με δύο μεθόδους, SARSA και Q-Learning. Με τη χρήση αυτών των μεθόδων μπορεί να εκτιμηθεί η αξία εκτέλεσης μιας ενέργειας από τον πράκτορα βάση της κατάστασης στην οποία βρίσκεται το περιβάλλον. Χρησιμοποιώντας αυτές τις συναρτήσεις αξίας ο κάθε παίκτης μπορεί να αναγνωρίσει ποια ενέργεια μπορεί να του αποφέρει την επιθυμητή αμοιβή, και ως αποτέλεσμα εκπαιδεύεται να λειτουργεί με βάση την απόκτηση μεγαλύτερης αμοιβής. Η συνάρτηση αξίας για τον αλγόριθμο Q-Learning περιγράφεται στο Σχήμα 2.5 και για τον αλγόριθμο SARSA στο Σχήμα 2.6. Αυτό που διαφοροποιεί τον ένα αλγόριθμο από τον άλλο είναι ότι στο SARSA η συνάρτηση αξίας λαμβάνει υπόψη τη συνάρτηση αξίας της επόμενης χρονικής στιγμής (t+1), δηλαδή της επόμενης ενέργειας που θα εκτελέσει ο πράκτορας στην επόμενη κατάσταση του παιχνιδιού, ενώ στο Q-Learning η συνάρτηση αξίας λαμβάνει υπόψη την βέλτιστη ενέργεια που μπορεί να εκτελέσει ο πράκτορας στην επόμενη κατάσταση του παιχνιδιού. Στο βασικό μοντέλο όπως αναφέρθηκε και προηγουμένως, έχουν δοκιμαστεί και οι δύο αλγόριθμοι μάθησης έτσι ώστε να συγκρίνουμε τα αποτελέσματα τους.

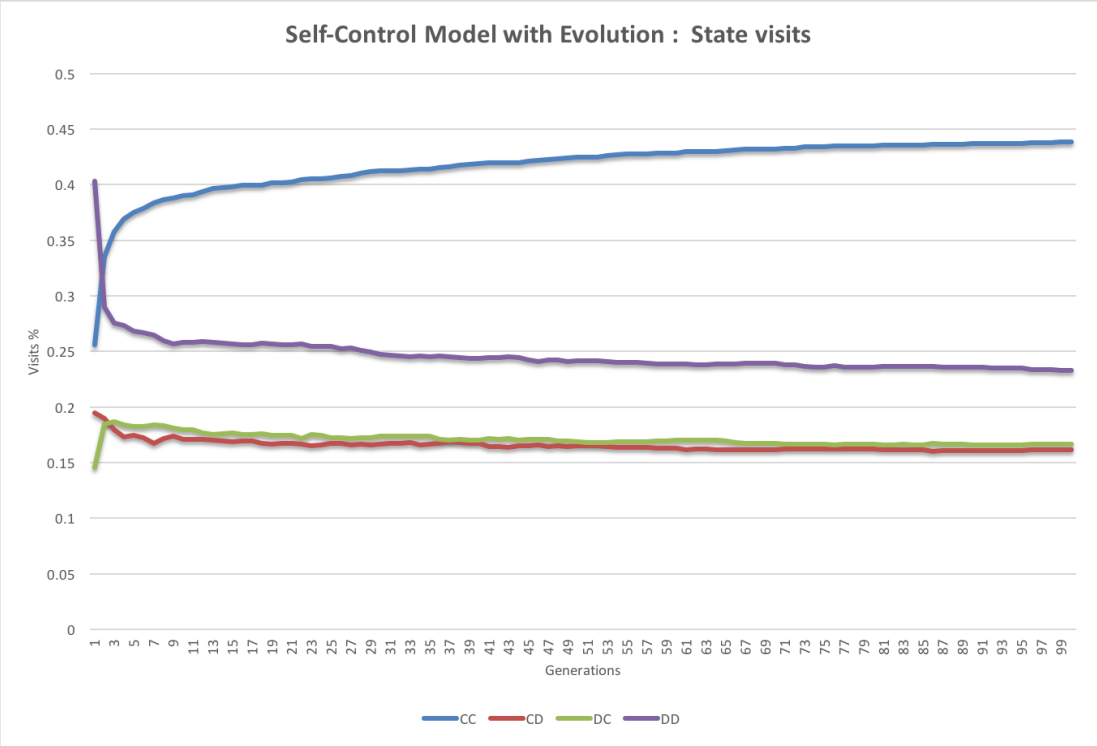
Τα αποτελέσματα αυτοελέγχου μετά την εφαρμογή της προδέσμευσης και της εξέλιξης στο σύστημα, φαίνονται στο Σχήμα 4.8 για τον αλγόριθμο Q-Learning και στο Σχήμα 4.9 για τον αλγόριθμο SARSA. Στις δύο αυτές εκτελέσεις χρησιμοποιήθηκαν ίδιες τιμές στις παραμέτρους των Εξελικτικών Στρατηγικών [5] [6] (‘μ’, ‘λ’, ‘ρ’, αριθμός γενεών, αριθμός δοκιμών), συνεπώς η μόνη διαφορά είναι ο αλγόριθμος μάθησης.

Συγκρίνοντας τη γραφική παράσταση στο Σχήμα 4.8 με αυτή στο Σχήμα 4.9 παρατηρούμε πως όπως και στο βασικό μοντέλο, ο αλγόριθμος Q-Learning κατορθώνει να εκπαιδεύσει καλύτερα τα άτομα να έχουν αυτοέλεγχο παρά ο αλγόριθμος SARSA. Αυτό συμπεραίνεται από την μεγάλη συχνότητα συνεργασίας των δύο λειτουργιών του εγκεφάλου στα άτομα όταν χρησιμοποιείται ο αλγόριθμος Q-Learning, αφού ο αυτοέλεγχος επιτυγχάνεται από την συνεργασία των άνω και κάτω λειτουργιών του εγκεφάλου. Πιο συγκεκριμένα, στον αλγόριθμο Q-Learning η συχνότητα συνεργασίας των άνω και κάτω λειτουργιών φτάνει μέχρι και 70% ενώ στον αλγόριθμο SARSA λίγο περισσότερο από 40% όπως φαίνεται στις γραφικές παραστάσεις των σχημάτων 4.8 και

4.9. Επίσης από αυτές τις γραφικές παραστάσεις παρατηρείται πως η συχνότητα επίσκεψης της κατάστασης CC (συνεργασία των άνω και κάτω λειτουργιών του εγκεφάλου που υπονοεί αυτοέλεγχο) αυξάνεται με την πάροδο των γενεών, σε αντίθεση με τις συχνότητες επίσκεψης των υπόλοιπων καταστάσεων του παιχνιδιού (CD, DC, DD) οι οποίες μειώνονται με την πάροδο των γενεών, υποδεικνύοντας πως τα άτομα εκπαιδεύονται σωστά και επιτυγχάνουν βελτιωμένη συμπεριφορά αυτοελέγχου σε κάθε γενεά.



Σχήμα 4.8 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 5

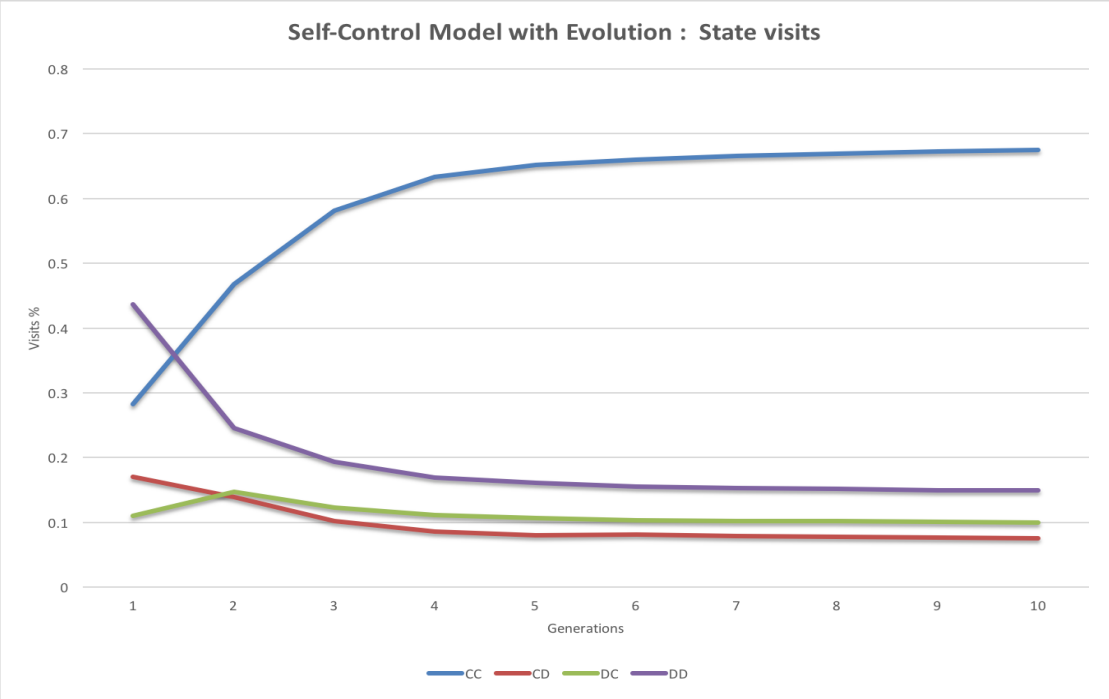


Σχήμα 4.9 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 5

4.1.2.2 Τιμές Παραμέτρων

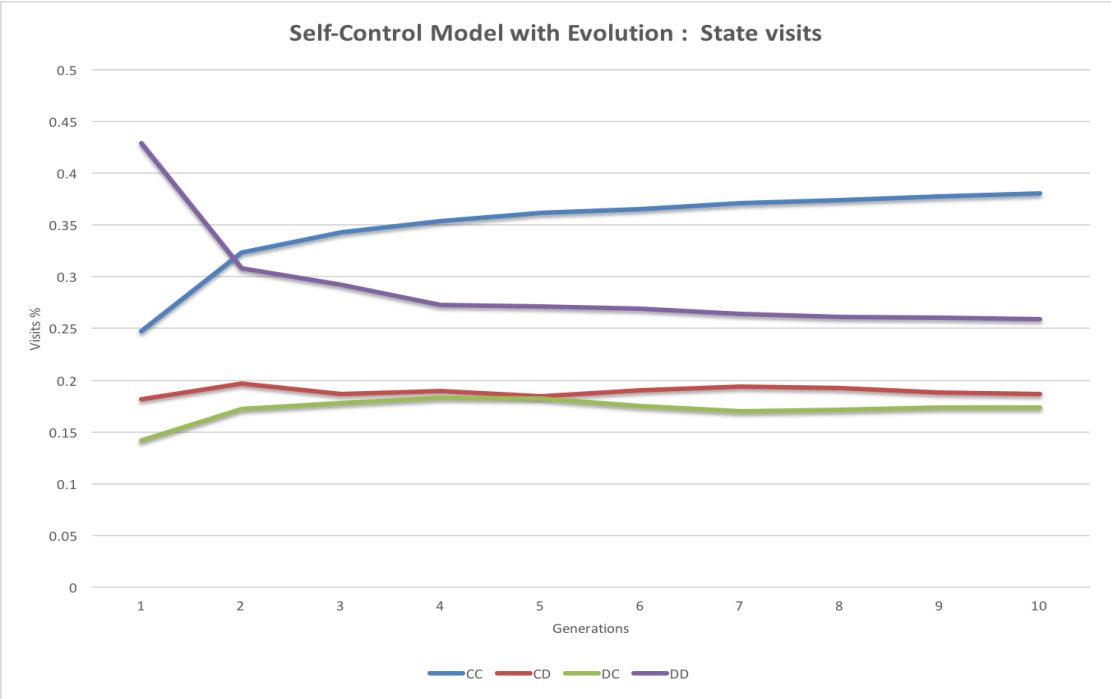
Από τις δοκιμές που έγιναν μετά την εφαρμογή της προδέσμευσης και της εξέλιξης στο σύστημα, παρατηρήθηκε ότι και με τους δύο αλγόριθμους (SARSA, Q-Learning) τα άτομα κατορθώνουν πάντα να εκπαιδεύονται επαρκώς έτσι ώστε να παρουσιάζουν βελτιωμένη συμπεριφορά αυτοελέγχου. Δηλαδή, το μεγαλύτερο ποσοστό των ενεργειών των άνω και κάτω λειτουργιών του εγκεφάλου του κάθε ατόμου να είναι η συνεργασία μεταξύ τους. Αυτό που καθορίζει τον βαθμό που θα φθάσει ο αυτοέλεγχος είναι οι τιμές που δίνουμε σε κάποιες από τις παραμέτρους των ατόμων και στην προσομοίωση της εξέλιξης. Πιο συγκεκριμένα, οι παράμετροι που προκαλούν διαφορές στα αποτελέσματα είναι ο αριθμός των γενεών που θα δημιουργηθούν, το ‘σ’ που επηρεάζει την μετάλλαξη που γίνεται στον γονότυπο των ατόμων και το ‘ε’ που καθορίζει μέχρι πιο βαθμό θα κάνει εξερεύνηση και εκμετάλλευση το κάθε άτομο.

Η διαφορά που προκύπτει μεταβάλλοντας τον αριθμό των γενεών φαίνεται στα σχήματα 4.10, 4.11, 4.12 και 4.13. Χρησιμοποιώντας τον αλγόριθμο Q-Learning ως αλγόριθμο μάθησης και εκτελώντας την προσομοίωση για 10 γενεές, το ποσοστό επιτυχίας αυτοέλεγχου στα άτομα έφθανε μέχρι 67% όπως φαίνεται στο Σχήμα 4.10, ενώ στο Σχήμα 4.11 όπου ο αριθμός των γενεών ήταν 600 το ποσοστό επιτυχίας αυτοέλεγχου στα άτομα έφθανε μέχρι 73%. Το αντίστοιχο πείραμα εκτελέστηκε και για τον αλγόριθμο SARSA και τα αποτελέσματα παρουσιάζονται στα σχήματα 4.12 και 4.13. Επίσης όπως αναφέρθηκε και πιο πάνω, η παράμετρος 'σ' φαίνεται να επηρεάζει την επίδοση του αυτοέλεγχου στα άτομα. Το έυρος τιμών αυτής της παραμέτρου είναι από 0 μέχρι 1. Όταν ανατεθεί σε αυτή την παράμετρο μια τιμή κοντά στο 1 τότε η επίδοση των ατόμων δεν είναι όσο καλή, σε αντίθεση με την περίπτωση που ανατίθεται μια τιμή κοντά στο 0 όπου η επίδοση είναι καλύτερη. Αυτές οι διαφορές φαίνονται στα σχήματα 4.8 και 4.14 για τον αλγόριθμο Q-Learning, όπου η μόνη διαφορά ανάμεσα σε αυτά τα σχήματα είναι η τιμή της παραμέτρου 'σ', και για τον αλγόριθμο SARSA στα σχήματα 4.9 και 4.15. Η τελευταία παράμετρος που αποδείχτηκε να επηρεάζει την εκπαίδευση για επίτευξη αυτοελέγχου είναι το 'ε', το οποίο καθορίζει τον βαθμό στον οποίο οι άνω και κάτω λειτουργίες του εγκεφάλου θα εκτελούν εκμετάλλευση ή εξερεύνηση όπως περιγράφηκε στην υλοποίηση του βασικού μοντέλου προσομοίωσης του εγκεφάλου. Μέσω των πειραμάτων που εκτελέστηκαν παρατηρήθηκε ότι η παράμετρος 'ε' πρέπει να έχει μια αρκετά μικρή τιμή για βέλτιστα αποτελέσματα. Στο Σχήμα 4.16 φαίνεται η μέγιστη συχνότητα συνεργασίας των δύο μερών του εγκεφάλου για επίτευξη αυτοέλεγχου, η οποία δεν υπερβαίνει το 42% όταν το 'ε' έχει τιμή 0.3, σε αντίθεση με τη μέγιστη συχνότητα στο Σχήμα 4.8 που φθάνει μέχρι 71% όταν το 'ε' έχει τιμή 0.1. Τα αντίστοιχα αποτελέσματα για τον αλγόριθμο SARSA παρουσιάζονται στις γραφικές παραστάσεις των σχημάτων 4.9 και 4.17.

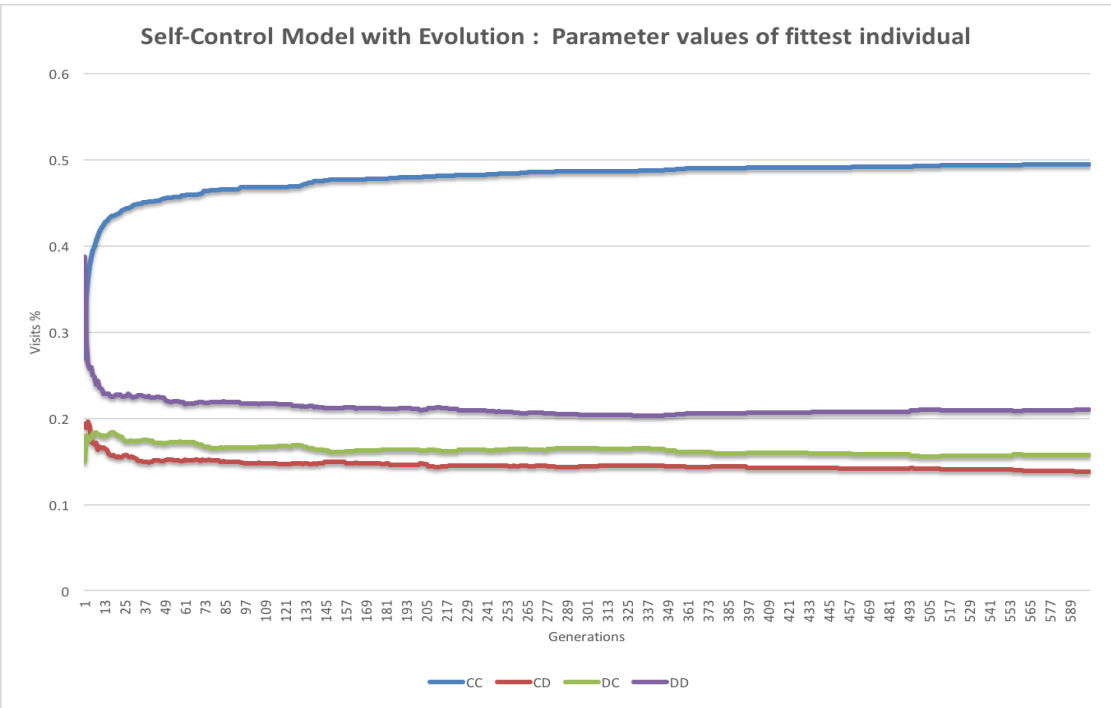


Σχήμα 4.10 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 10, αρ. εκτελέσεων = 5

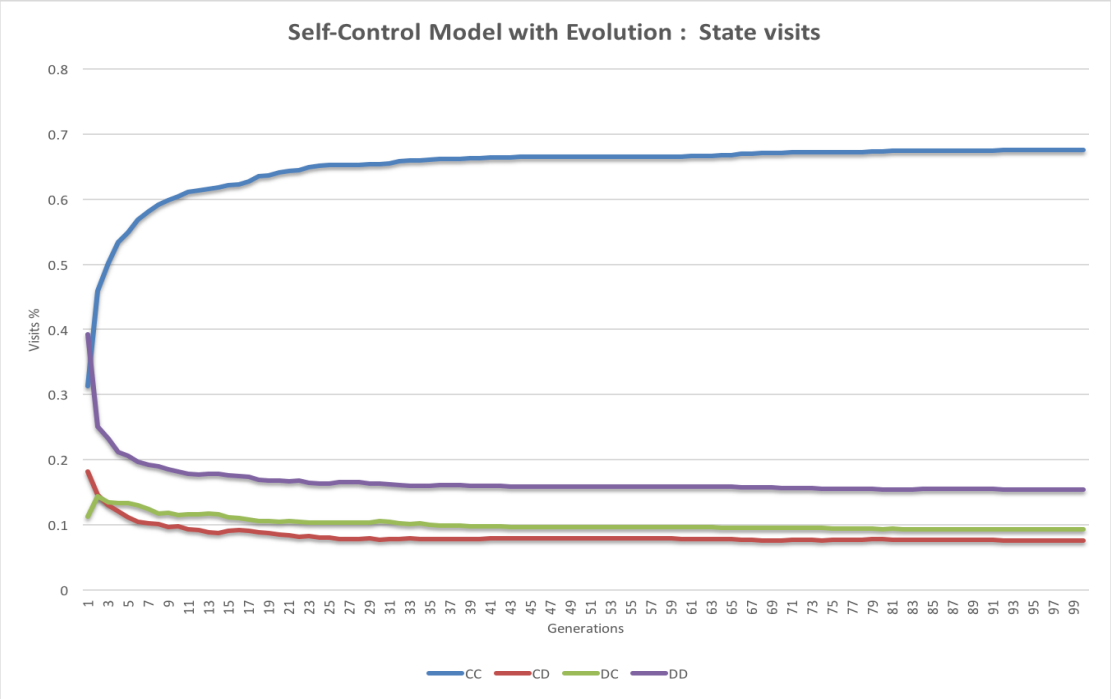
Σχήμα 4.11 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 600, αρ. εκτελέσεων = 5



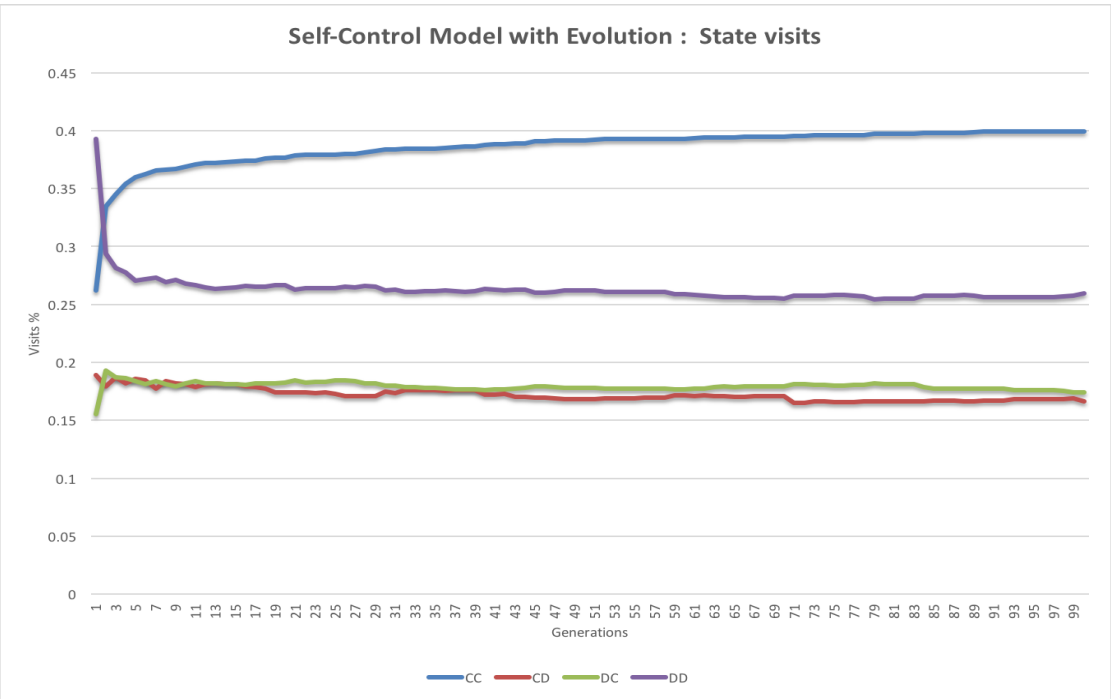
Σχήμα 4.12 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 10, αρ. εκτελέσεων = 5



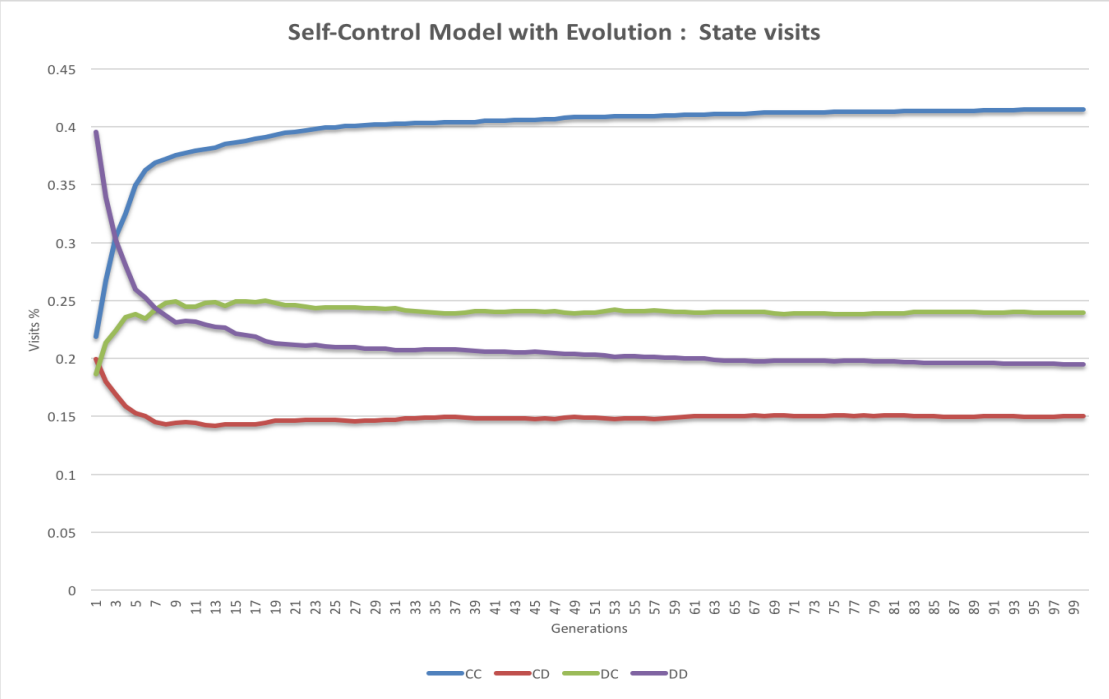
Σχήμα 4.13 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 600, αρ. εκτελέσεων = 5



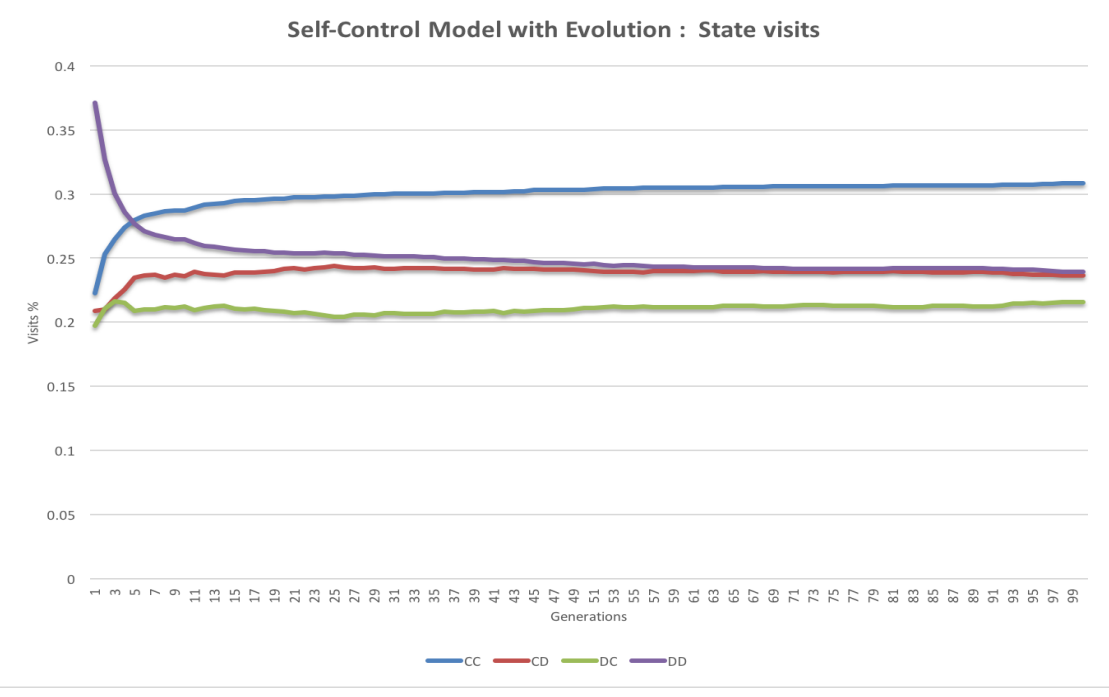
Σχήμα 4.14 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning. Παράμετροι: μ=10, λ=70, σ=0.8, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 5



Σχήμα 4.15 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA. Παράμετροι: μ=10, λ=70, σ=0.8, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 5



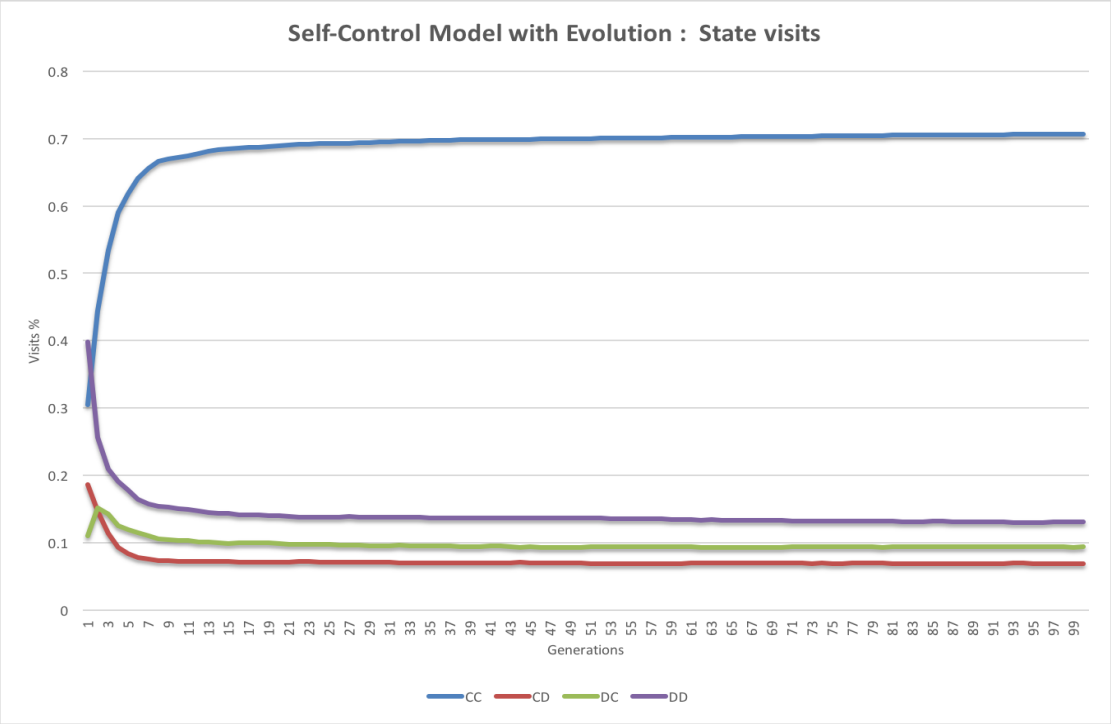
Σχήμα 4.16 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.3, αρ. γενεών = 100, αρ. εκτελέσεων = 5



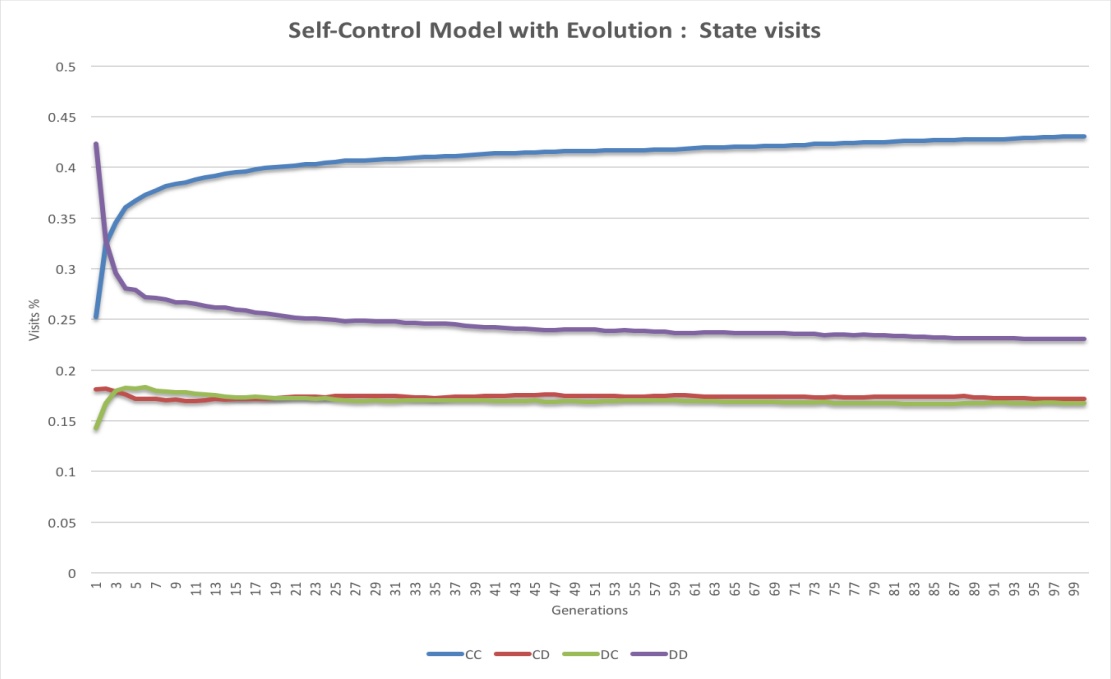
Σχήμα 4.17 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.3, αρ. γενεών = 100, αρ. εκτελέσεων = 5

Έχουν γίνει αρκετές δοκιμές αναθέτοντας διάφορες τιμές και στις υπόλοιπες παραμέτρους (αριθμός εκτελέσεων, ‘μ’, ‘λ’) έτσι ώστε να μελετηθεί η επίδραση τους στην εκπαίδευση των ατόμων. Οι τιμές αυτών των παραμέτρων δεν φάνηκε να επηρεάζουν την εκπαίδευση των ατόμων και συνεπώς τον βαθμό επίτευξης αυτοέλεγχου σε μια γενεά η οποία εξελίσσεται.

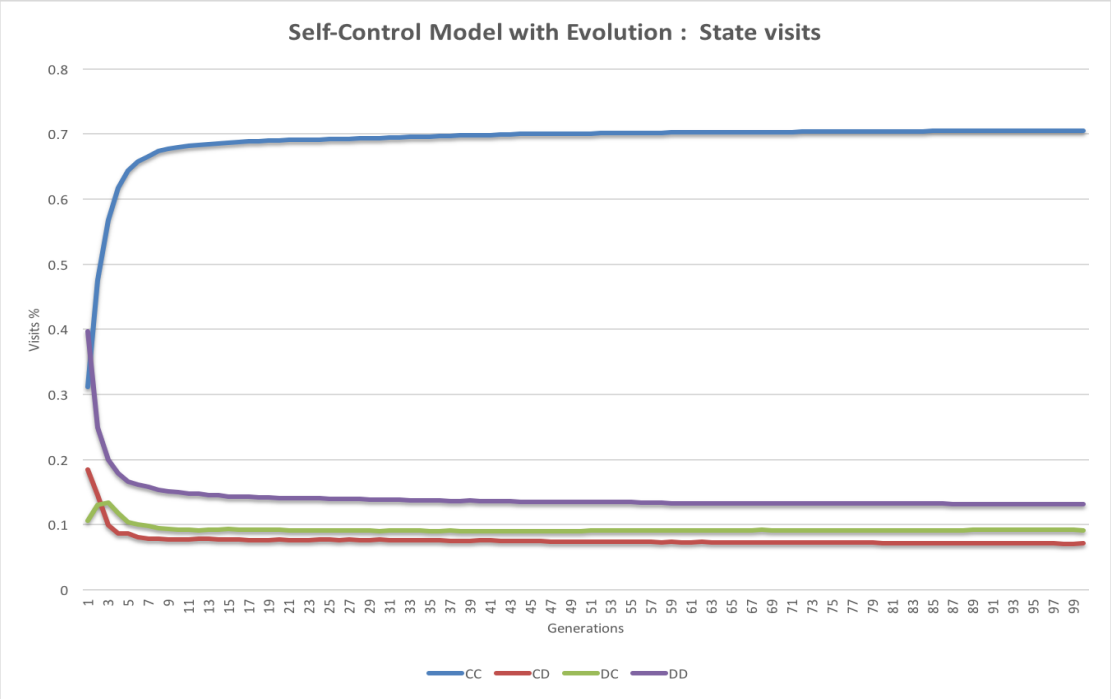
Ο αριθμός εκτελέσεων αντιπροσωπεύει τον αριθμό των εκτελέσεων της προσομοίωσης, έτσι ώστε τα αποτελέσματα στις γραφικές παραστάσεις να είναι ο μέσος όρων των αποτελεσμάτων αυτών των προσομοιώσεων για να υπάρχει μεγαλύτερη ακρίβεια στα συμπεράσματα. Όπως φαίνεται και στα σχήματα 4.8 και 4.18 για τον αλγόριθμο μάθησης Q-Learning με αριθμό εκτελέσεων ίσο με 5 και 10 αντίστοιχα, δεν υπάρχει εμφανής διαφορά στα αποτελέσματα των επιδόσεων τον ατόμων για επίτευξη αυτοελέγχου. Το ίδιο ισχύει και για τον αλγόριθμο μάθησης SARSA στα σχήματα 4.9 και 4.19. Ως αποτέλεσμα για κάθε πείραμα θεωρήθηκαν ικανοποιητικές 5 προσομοιώσεις και υπολογιζόταν ο μέσος όρος για ανάλυση των αποτελεσμάτων. Επίσης έγιναν πειράματα με διάφορες άλλες τιμές από αυτές που φαίνονται στις πιο πάνω γραφικές για τις παραμέτρους ‘μ’ που αντιπροσωπεύει το μέγεθος του πληθυσμού των απογόνων στις και ‘λ’ που αντιπροσωπεύει το μέγεθος του πληθυσμού των γονέων στις Εξελικτικές Στρατηγικές. Όπως παρατηρείται στα σχήματα 4.20 και 4.21 για τους αλγόριθμους Q-Learning και SARSA αντίστοιχα, ενώ διπλασιάστηκε η τιμή αυτών των δύο παραμέτρων σε σχέση με τα πειράματα στα σχήματα 4.8 και 4.9 δεν παρατηρείται οποιαδήποτε διαφορά στον αυτοέλεγχο των ατόμων.



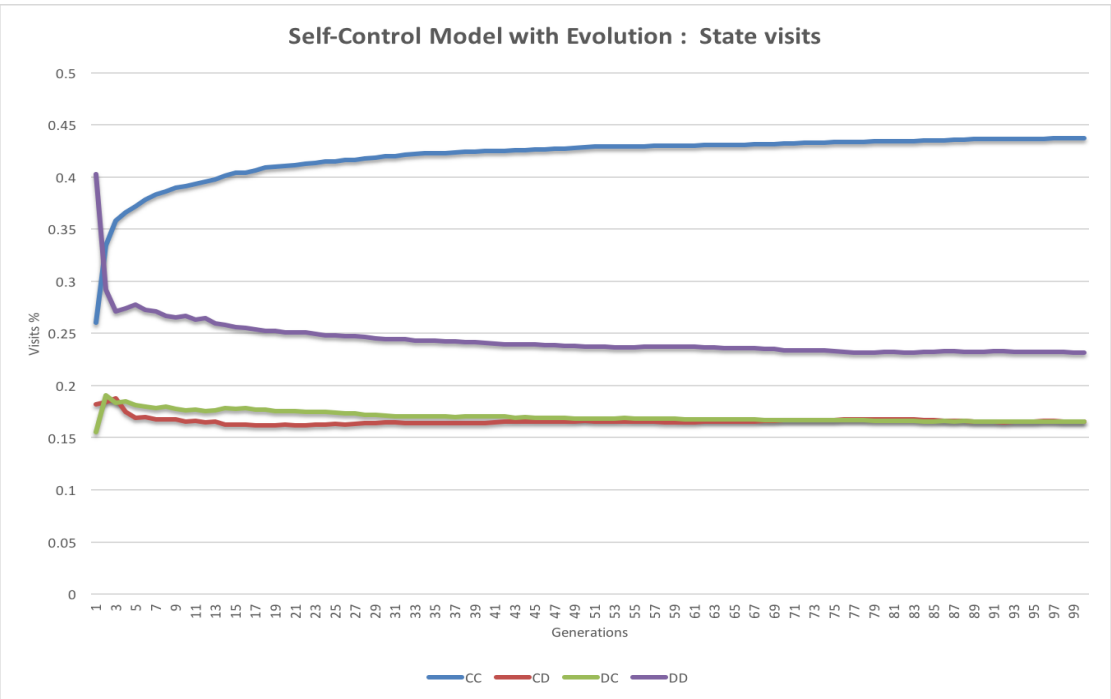
Σχήμα 4.18 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 10



Σχήμα 4.19 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA. Παράμετροι: μ=10, λ=70, σ=0.1, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 10



Σχήμα 4.20 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: Q-Learning. Παράμετροι: μ=20, λ=140, σ=0.1, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 5



Σχήμα 4.21 Γραφική Παράσταση της συχνότητας εφαρμογής κάθε ενέργειας (CC, CD, DC, DD) από τα άτομα σε κάθε γενεά. Αλγόριθμος: SARSA. Παράμετροι: μ=20, λ=140, σ=0.1, ε=0.1, αρ. γενεών = 100, αρ. εκτελέσεων = 5

4.2 Ανάλυση και Συζήτηση Αποτελεσμάτων

Αρχικά στο πρώτο μέρος αυτής της Διπλωματικής Εργασίας είχε υλοποιηθεί η προσομοίωση της εξάσκησης αυτοέλεγχου στον εγκέφαλο ενός ατόμου., η οποία αποτελεί το βασικό μοντέλο αυτής της εργασίας. Η προσομοίωση έγινε αναπτύσσοντας ένα υπολογιστικό μοντέλο το οποίο περιείχε όλα τα βασικά χαρακτηριστικά έτσι ώστε να είναι βιολογικά και ψυχολογικά σχετικό, αντιπροσώπευε δηλαδή σε μεγάλο βαθμό τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου. Κατά την εξάσκηση του αυτοέλεγχου ενός οργανισμού, συγκρούονται οι επιθυμίες της ορθολογιστικής σκέψης που σχετίζεται με τις άνω λειτουργίες του εγκεφάλου και οι επιθυμίες της ενστικτώδης συμπεριφοράς που σχετίζεται με τις κάτω λειτουργίες του εγκεφάλου. Αυτή η σύγκρουση, η οποία μπορεί να προσομοιωθεί από ένα υπολογιστικό σύστημα, υλοποιήθηκε στο βασικό μοντέλο προσομοίωσης του εγκεφάλου με το παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» χρησιμοποιώντας Ενισχυτική Μάθηση και συγκεκριμένα τη μέθοδο Χρονικών Διαφορών για την εκπαίδευση αυτών των λειτουργιών. Η μέθοδος Χρονικών Διαφορών όπως έχει αναφερθεί μπορεί να εφαρμοσθεί με διάφορους αλγόριθμους. Δύο από αυτούς τους αλγόριθμους που δοκιμάστηκαν στην υλοποίηση αυτής της Διπλωματικής Εργασίας είναι ο Q-Learning και ο SARSA. Όπως παρουσιάστηκε στα πειράματα που εκτελέστηκαν σε αυτό το βασικό μοντέλο στο Κεφάλαιο 4, ο εγκέφαλος δεν κατόρθωνε πάντα να εκπαιδευτεί επαρκώς έτσι ώστε να παρουσιάζει αυτοέλεγχο ή αν παρουσίαζε τότε δεν έφθανε σε ικανοποιητικά επίπεδα και δεν βελτιωνόταν με την πάροδο του χρόνου. Η επιτυχία της εκπαίδευσης επηρεαζόταν από διάφορους παράγοντες όπως ο αλγόριθμος μάθησης που εφαρμοζόταν (Q-Learning ή SARSA) και οι τιμές των παραμέτρων των παικτών (‘η’, ‘γ’, ‘ε’).

Τα προβλήματα που αναφέρθηκαν πιο πάνω ότι προέκυπταν στο βασικό μοντέλο, λύθηκαν με την εφαρμογή της προδέσμευσης και της εξέλιξης στο σύστημα. Η έννοια της εξέλιξης εφαρμόστηκε στο σύστημα χρησιμοποιώντας ένα εξελικτικό αλγόριθμο, τις Εξελικτικές Στρατηγικές [5] [6]. Αυτό είχε ως αποτέλεσμα να συμπεριληφθούν στη μοντελοποίηση άτομα, τα οποία εκπαιδεύονταν να έχουν αυτοέλεγχο εφαρμόζοντας το αποτέλεσμα της προδέσμευσης στην εκπαίδευση τους έτσι ώστε να υποβοηθάτε η εξάσκηση του αυτοέλεγχου τους, και κάθε γενεά που σχηματιζόταν περιείχε άτομα με όλο και πιο βελτιωμένη συμπεριφορά αυτοέλεγχου. Μετά την εφαρμογή της

προδέσμευσης και της εξέλιξης στην προσομοίωση, όπως παρουσιάστηκε και στα πειράματα στο κεφάλαιο 4, τα άτομα κατόρθωναν σε οποιαδήποτε περίπτωση να έχουν όλο και πιο βελτιωμένη συμπεριφορά αυτοελέγχου με την πάροδο των γενεών. Όπως φαίνεται και στο Σχήμα 4.8 όπου χρησιμοποιήθηκε ο αλγόριθμος Q-Learning, η συχνότητα επίτευξης αυτοέλεγχου (δηλαδή η κατάσταση CC που υποδηλώνει συνεργασία των άνω και κάτω λειτουργιών του εγκεφάλου) εν όσον περνούσαν οι γενεές αυξανόταν όλο και περισσότερο και ήταν πολύ μεγαλύτερη σε σχέση με τις υπόλοιπες ενέργειες που μπορούσε να επιλέξει ο εγκέφαλος των ατόμων (δηλαδή καταστάσεις CD, DC και DD οι οποίες δεν οδηγούσαν σε αυτοέλεγχο). Η διαφορά με τα αποτελέσματα του αρχικού βασικού μοντέλου όπου δεν είχε εφαρμοσθεί η προδέσμευση και η έννοια της εξέλιξης μπορεί να φανεί συγκρίνοντας με το Σχήμα 4.1, όπου ο αυτοέλεγχος ενώ αυξανόταν αρχικά (δηλαδή η κατάσταση CC που υποδηλώνει συνεργασία των άνω και κάτω λειτουργιών του εγκεφάλου) ακολούθως μειωνόταν την ίδια στιγμή που η συχνότητα εκτέλεσης των υπόλοιπων ενεργειών (δηλαδή καταστάσεις CD, DC και DD οι οποίες δεν οδηγούσαν σε αυτοέλεγχο) αυξανόταν. Επίσης εμφανής είναι και η βελτίωση όταν χρησιμοποιήθηκε ο αλγόριθμος SARSA εάν συγκρίνουμε τα σχήματα 4.2 και 4.9. Στο Σχήμα 4.2 είναι τα αποτελέσματα του βασικού μοντέλου πριν εφαρμοσθεί η προδέσμευση και η εξέλιξη. Παρατηρείται πως δεν είχε επιτευχθεί αυτοέλεγχος αφού η συχνότητα τη συνεργασίας των άνω και κάτω λειτουργιών του εγκεφάλου (CC) ήταν πολύ μικρότερη από οποιαδήποτε άλλη επιλογή αυτών των λειτουργιών (CD, DC, DD). Στο Σχήμα 4.9 αντιθέτως επιτεύχθηκε αυτοέλεγχος στα άτομα και παρατηρείται αύξηση της συχνότητας επίτευξης του με την πάροδο των γενεών.

Επίσης, όπως παρατηρήθηκε μέσω των πειραμάτων που εκτελέστηκαν στην προσομοίωση, ο βαθμός που έφθανε ο αυτοέλεγχος εξαρτώταν από τις τιμές που θέταμε σε κάποιες από τις παραμέτρους των ατόμων και στην προσομοίωση της εξέλιξης. Πιο συγκεκριμένα, οι παράμετροι που προκαλούσαν διαφορές στα αποτελέσματα ήταν ο αριθμός των γενεών που σχηματίζονταν, το ‘σ’ που επηρέαζε την μετάλλαξη που γινόταν στον γονότυπο των ατόμων και το ‘ε’ που καθόριζε μέχρι πιο βαθμό θα κάνει εξερεύνηση και εκμετάλλευση το κάθε άτομο. Οι πιο κατάλληλες τιμές για την παράμετρο των γενεών που παρατηρήθηκε να έχουν τα βέλτιστα αποτελέσματα είναι από 100 μέχρι 600, αφού με λιγότερες από 100 γενεές δεν έφθανε σε τόσο ψηλό βαθμό ο αυτοέλεγχος και με περισσότερες από 600 δεν παρουσιαζόταν επιπλέον βελτίωση στον αυτοέλεγχο των

ατόμων καθώς έμενε στα ίδια περίπου επίπεδα η συχνότητα συνεργασίας των άνω και κάτω λειτουργιών του εγκεφάλου των ατόμων. Η πιο κατάλληλη τιμή για την παράμετρο ‘σ’ ήταν μια πολύ μικρή κοντά στο 0, για παράδειγμα 0.1, η οποία προκαλούσε μικρές μεταβολές στις μεταλλάξεις που γίνονταν στον γονότυπο των ατόμων φέρνοντας έτσι βέλτιστα αποτελέσματα για την επίτευξη υψηλών επιπέδων αυτοελέγχου. Τέλος, για την παράμετρο ‘ε’ πάλι η τιμή 0.1 φάνηκε να έχει τα βέλτιστα αποτελέσματα. Αυτή η τιμή για το ‘ε’ υπονοεί πως δεν γινόταν πολύ συχνά εξερεύνηση από τους πράκτορες στο παιχνίδι «Επαναλαμβανόμενο Δίλημμα του Φυλακισμένου» για ενέργειες που θα επιφέρουν άγνωστες και πιθανόν μη επιθυμητές αμοιβές, αλλά γινόταν εκμετάλλευση εκτελώντας ενέργειες που ήταν γνωστό πως θα επιφέρουν υψηλές αμοιβές (δηλαδή η ενέργεια που θα επιλέξει ο πράκτορας να είναι η συνεργασία, που είναι αυτό που επιθυμούμε για επίτευξη αυτοέλεγχου) με ποσοστό 90%.

Όπως αναφέρθηκε και προηγουμένως, για την εκπαίδευση των ατόμων για επίτευξη βελτιωμένης συμπεριφοράς αυτοελέγχου έχουν δοκιμαστεί δύο διαφορετικοί αλγόριθμοι Χρονικών Διαφορών, το Q-Learning και το SARSA. Η συνάρτηση αξίας για τον αλγόριθμο Q-Learning περιγράφεται στο Σχήμα 2.5 και για τον αλγόριθμο SARSA στο Σχήμα 2.62.6. Ο αλγόριθμος SARSA ακολουθεί μια πολιτική (on-policy learning algorithm). Αρχίζει με μια τυχαία πολιτική και προσπαθεί να την κάνει καλύτερη με την πάροδο του χρόνου, καθώς επίσης μαθαίνει την αξία της πολιτικής του πράκτορα βάση των ενεργειών του. Ο αλγόριθμος Q-Learning δεν ακολουθεί μια πολιτική (off-policy learning algorithm). Χρησιμοποιεί οποιαδήποτε πολιτική για να υπολογίσει το ‘Q’ και μαθαίνει την αξία της βέλτιστης πολιτικής ανεξαρτήτως των ενεργειών του πράκτορα.

Και οι δύο αλγόριθμοι συγκλίνουν στα επιθυμητά αποτελέσματα αλλά με διαφορετικούς ρυθμούς. Ένα πλεονέκτημα του αλγόριθμου SARSA είναι ότι λαμβάνει υπόψη την πραγματική πολιτική που ακολουθείται από τον πράκτορα, συμπεριλαμβανομένης και της εξερεύνησης, αφού δίνει σημασία στη μελλοντική κίνηση του πράκτορα στην συνάρτηση αξίας. Σε αντίθεση ο αλγόριθμος Q-Learning υποθέτει πως κάνει το σωστό, λαμβάνοντας υπόψη στη συνάρτηση αξίας μόνο την καλύτερη αμοιβή της παρούσας κατάστασης. Σχετικά με αυτό που μόλις αναφέρθηκε, αξίζει να σημειωθεί ότι έχουν παρατηρηθεί φαινόμενα στον εγκέφαλο που ευθυγραμμίζονται περισσότερο με τον αλγόριθμο SARSA. Σε ένα πείραμα όπου συμμετείχαν πίθηκοι οι οποίοι καλούνταν να

κάνουν κάποιες επιλογές λαμβάνοντας τις ανάλογες αμοιβές, και ακολούθως παραγόταν ένα σήμα σφάλματος βάση της επιλογής τους, παρατηρήθηκε πώς κάποιες νευρωνικές αποκρίσεις στον εγκέφαλο τους συμβάδιζαν περισσότερο με την λογική του αλγόριθμου μάθησης SARSA [13]. Πιο συγκεκριμένα, στα αποτελέσματα των πειραμάτων σε νευρώνες ντοπαμίνης [14] παρατηρήθηκε βάση των σφαλμάτων που αναφέρθηκαν προηγουμένως ότι οι αποκρίσεις των νευρώνων αντανακλούσαν τις μελλοντικές επιλογές του πιθήκου και συνεπώς την λογική του αλγόριθμου SARSA όπου χρησιμοποιείται η μελλοντική επιλογή στον υπολογισμό της αξίας μιας ενέργειας στην παρούσα κατάσταση. Αντ΄αυτού, τα αποτελέσματα σε ένα άλλο πείραμα όπου συμμετείχαν αρουραίοι, οι νευρωνικές αποκρίσεις του εγκεφάλου τους στην διαδικασία επιλογής πάλι ανάμεσα σε ερεθίσματα διάφορων αμοιβών, φαίνεται να αντανακλούσαν την αξία της καλύτερης διαθέσιμης επιλογής, επομένως είναι πιο συμβατές με τον αλγόριθμο Q-Learning [15].

Ακόμη όσο αφορά το θέμα της σύγκλισης, ο ένας αλγόριθμος δεν συγκλίνει απαραίτητα πιο εύκολα από τον άλλο, συνήθως εξαρτάται από τα περιβάλλοντα στα οποία εκτελούνται. Στην υλοποίηση αυτής της Διπλωματικής εργασίας, παρά την υψηλότερη συσχέτιση του αλγόριθμου SARSA με τις λειτουργίες του εγκεφάλου, στην προσομοίωση που πραγματοποιήθηκε ο αλγόριθμος Q-Learning φάνηκε να συγκλίνει γρηγορότερα και να πετυχαίνει σε μεγαλύτερο βαθμό τον αυτοέλεγχο των ατόμων.

Κεφάλαιο 5

Συμπεράσματα και Μελλοντική Εργασία

5.1 Σύνοψη

Ένα άτομο παρουσιάζει αυτοέλεγχο όταν αφήνει την λογική να νικήσει τον παρορμητισμό του. Δηλαδή, όταν επιλέγει να κάνει μια πράξη η οποία θα του επιφέρει μια αργοπορημένη αλλά μεγαλύτερη αμοιβή (γεγονότα που σχετίζονται με την λογική και την ορθολογιστική σκέψη), παρά μια πράξη η οποία θα του επιφέρει μια συντομότερη αλλά μικρότερη αμοιβή (γεγονότα που σχετίζονται με το συναίσθημα και τον αυθορμητισμό). Αυτό το δίλημμα είναι μια εσωτερική σύγκρουση ανάμεσα στην ορθολογιστική σκέψη που σχετίζεται με τις άνω λειτουργίες του εγκεφάλου και την ενστικτώδη συμπεριφορά που σχετίζεται με τις κάτω λειτουργίες του εγκεφάλου. Ο αυτοέλεγχος μπορεί να επιτευχθεί πιο εύκολα με τη χρήση της προδέσμευσης. Προδέσμευση είναι μια ενέργεια που κάνουμε τώρα η οποία απαλείφει την επιλογή του πειρασμού στο μέλλον ή την κάνει πιο δύσκολη να πραγματοποιηθεί, μας προτρέπει δηλαδή στην επιλογή της αργοπορημένης αλλά μεγαλύτερης αμοιβής που σχετίζεται με την λογική και την ορθότητα.

Στα πλαίσια αυτής της Διπλωματικής Εργασίας μελετήθηκε ο τρόπος με τον οποίο η προδέσμευση συμβάλει στον αυτοέλεγχο των ατόμων κατά την εξέλιξη τους. Για την μελέτη αυτή έγινε προσομοίωση της εξέλιξης των ατόμων, τα οποία εκπαιδεύονταν να

έχουν αυτοέλεγχο χρησιμοποιώντας προδέσμευση. Η προσομοίωση έγινε εφικτή δημιουργώντας αρχικά ένα πληθυσμό από άτομα. Ο εγκέφαλος κάθε ατόμου προσομοιώθηκε αναπτύσσοντας ένα υπολογιστικό μοντέλο, το οποίο περιείχε όλα τα βασικά χαρακτηριστικά έτσι ώστε να είναι βιολογικά και ψυχολογικά σχετικό, να αντιπροσωπεύει δηλαδή σε μεγάλο βαθμό τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου. Οι άνω και κάτω λειτουργίες του εγκεφάλου, οι οποίες σχετίζονται με την ορθολογιστική σκέψη και την ενστικτώδη συμπεριφορά αντίστοιχα, εκ φύσεως συναγωνίζονται για το ποιος θα έχει τον έλεγχο του οργανισμού. Σκοπός εδώ είναι να κατορθώσουν να συμβιβαστούν αυτές οι δύο λειτουργίες, καταλήγοντας έτσι στον αυτοέλεγχο του οργανισμού. Αυτός ο ανταγωνισμός εφαρμόστηκε στο μοντέλο με ένα παιχνίδι γενικού αθροίσματος, το «Επαναλαμβανόμενο Δίλημμα του φυλακισμένου» και χρησιμοποιήθηκε Ενισχυτική Μάθηση για την εκπαίδευση των παικτών αυτού του παιχνιδιού. Μετά το τέλος αυτού του παιχνιδιού οι δύο παίκτες ,που αντιπροσωπεύουν τις άνω και κάτω λειτουργίες του εγκεφάλου, μαθαίνουν να συμβιβάζονται από την εκπαίδευση που γίνεται σε αυτούς καταλήγοντας έτσι στην επίτευξη αυτοελέγχου. Επίσης, το μοντέλο όπως αναφέρθηκε έχει εξελικτικό χαρακτήρα, υπόκειται δηλαδή σε προσομοίωση της γενετικής εξέλιξης με μάθηση. Η εφαρμογή της γενετικής εξέλιξης έγινε χρησιμοποιώντας ένα εξελικτικό αλγόριθμο, τις Εξελικτικές Στρατηγικές. Μέσω αυτών των στρατηγικών τα άτομα υπόκεινται σε συγκεκριμένες διαδικασίες οι οποίες έχουν ως αποτέλεσμα κάθε γενεά να περιέχει άτομα με πιο βελτιστοποιημένα χαρακτηριστικά από αυτά της προηγούμενης γενεάς, τα οποία είναι σχετικά με την επίτευξη αυτοέλεγχου. Έτσι, με την πάροδο των γενεών και αφού τα άτομα εκπαιδεύονταν επαρκώς, παρουσιαζόταν βελτιωμένη συμπεριφορά αυτοέλεγχου μέσω προδέσμευσης.

Στα πλαίσια της μελέτης αυτής, έγιναν κάποια πειράματα για διερεύνηση των παραγόντων που μπορεί να επηρεάζουν την επίτευξη αυτοελέγχου και τον βαθμό στον οποίο μπορεί να επιτευχθεί ο αυτοέλεγχος. Δοκιμάστηκαν δύο διαφορετικοί αλγόριθμοι μάθησης για την εκπαίδευση του εγκεφάλου, ο Q-Learning και ο SARSA. Επίσης δοκιμάστηκαν εναλλακτικές τιμές στις παραμέτρους των Εξελικτικών Στρατηγικών καθώς και στις παραμέτρους που χαρακτηρίζουν τα άτομα, όπως περιγράφηκαν στο κεφάλαιο 4. Παρατηρήθηκε ότι το επίπεδο στο οποίο μπορεί να φθάσει ο αυτοέλεγχος

επηρεάζεται άμεσα από τον αλγόριθμο μάθησης που χρησιμοποιείται, και σε μικρότερο βαθμό επηρεάζεται από τις τιμές των παραμέτρων. Με τον αλγόριθμο Q-Learning σημειώθηκε μεγαλύτερη συχνότητα αυτοελέγχου από ότι με τον αλγόριθμο SARSA.

Επίσης στο αρχικό βασικό μοντέλο, το οποίο προσομοίωνε την εξάσκηση του αυτοέλεγχου στον εγκέφαλο ενός ατόμου χωρίς την εφαρμογή της προδέσμευσης και της εξέλιξης, δεν γινόταν πάντα επίτευξη αυτοελέγχου. Μετά την εφαρμογή της προδέσμευσης καθώς και της προσομοίωσης της εξέλιξης, γινόταν πάντα επίτευξη αυτοελέγχου με σταδιακή βελτίωση σε κάθε γενεά.

5.2 Συμπεράσματα

Συγκρίνοντας τα αποτελέσματα της προσομοίωσης πριν την εφαρμογή της προδέσμευσης και της εξέλιξης, με τα αποτελέσματα της προσομοίωσης αφού εφαρμόστηκαν αυτές οι δύο έννοιες, μπορούμε να παρατηρήσουμε την υψηλή σημασία τους στην επίτευξη του αυτοέλεγχου ενός ατόμου. Ενώ στην πρώτη προσομοίωση, δηλαδή πριν εφαρμοσθεί η προδέσμευση και η εξέλιξη, δεν επιτυγχανόταν πάντα αυτοέλεγχος ή όταν επιτυγχανόταν δεν βελτιωνόταν με την πάροδο του χρόνου και αντιθέτως μειωνόταν, στην δεύτερη προσομοίωση όπου εφαρμόσθηκε η προδέσμευση και η εξέλιξη πάντα τα άτομα εκπαιδεύονταν ικανοποιητικά έτσι ώστε να παρουσιάζουν βελτιωμένη συμπεριφορά αυτοελέγχου με την πάροδο των γενεών. Η συχνότητα επιλογής συνεργασίας των άνω και κάτω λειτουργιών του εγκεφάλου και συνεπώς της επίτευξης αυτοέλεγχου του ατόμου, ήταν πολύ μεγαλύτερη κατά μέσο όρο από την συχνότητα επιλογής οποιασδήποτε άλλης ενέργειας που δεν επιφέρει αυτοέλεγχο. Συνεπώς συμπεραίνεται πώς η προδέσμευση ενισχύει τον αυτοέλεγχο ενός ατόμου και μπορεί αυτό να εφαρμοσθεί σε ένα υπολογιστικό μοντέλο επιτυχώς. Επίσης όπως έχει παρατηρηθεί μέσω των πειραμάτων που εκτελέστηκαν, ο αλγόριθμος εκπαίδευσης Q-Learning για το συγκεκριμένο περιβάλλον έχει καλύτερα αποτελέσματα από ότι ο αλγόριθμος SARSA, αφού συγκλίνει γρηγορότερα και επιτυγχάνει υψηλότερα επίπεδα αυτοελέγχου.

5.3 Μελλοντική Εργασία

Βλέποντας τα αποτελέσματα αυτής της διπλωματικής εργασίας προκύπτουν κάποιες ιδέες οι οποίες μπορούν να αποτελέσουν πηγή για μελλοντική εργασία σε αυτή την έρευνα. Όπως είδαμε στα αποτελέσματα των πειραμάτων στο Κεφάλαιο 4, ο αυτοέλεγχος μπορούσε να φθάσει ως κάποιο βαθμό στον κάθε αλγόριθμο μάθησης. Αν και στον αλγόριθμο Q-Learning η συχνότητα εφαρμογής του αυτοέλεγχου ήταν πολύ μεγαλύτερη από αυτήν του αλγόριθμου SARSA, και με τους δύο αλγόριθμους μπορεί να υπάρξουν περιθώρια βελτίωσης. Για παράδειγμα μπορεί να διερευνηθεί πώς θα εξελιχθεί το επίπεδο του αυτοέλεγχου εάν στην προσομοίωση της εξέλιξης εφαρμοσθούν και οι υπόλοιπες διαδικασίες των Εξελικτικών Στρατηγικών, όπως ο ανασυνδυασμός γονιδίων από τον πληθυσμό των γονέων για παραγωγή απογόνων, παρά μόνο η μετάλλαξη και η επιλογή που κρίθηκαν ικανοποιητικές για την επίτευξη του στόχου μας σε αυτή την διπλωματική εργασία. Επίσης θα μπορούσε να εφαρμοσθεί μια άλλη προσέγγιση κατά την διαδικασία της μετάλλαξης των γονιδίων των ατόμων, για παράδειγμα να γίνεται μετάλλαξη και στις στρατηγικές παραμέτρους των ατόμων παρά μόνο στις αντικειμενικές παραμέτρους όπως εκτελείται σε αυτή την υλοποίηση.

Αναφορές

[1] Christodoulou, C., Banfield, G., & Cleanthous, A. (2010). Self-Control with spiking and non-spiking neural networks playing games. *Journal of Physiology - Paris , 104*, 108-117.

[2] Banfield, G. D. (2006). *Simulation of Self-Control through Precommitment Behaviour in an Evolutionary System.* PhD Thesis, University of London, Birkbeck College, UK.

[3] Sozou, P. (2003). The Evolutionary Context of Self-Control Problems. *Workshop on the Evolutionary Biology of Learning.* Fribourg, Switzerland: http://www.unifr.ch/biol/ecology/learningworkshop/abstracts.pdf, 28/12/2014.

[4] Cleanthous, A. (2010). *In search of self-control through computational modeling of internal conflict.* PhD Thesis, University of Cyprus, Department of Computer Science, Cyprus.

[5] Beyer, H.-G. (2007). Evolution Strategies. *Scholarpedia , 2* (8), 1965.

[6] Beyer, H.-G., & Schwefel, H.-P. (2002). Evolution Strategies, A Comprehensive Introduction. *Natural Computing , 1* (1), 3-52.

[7] Holland, J. H. (1992). Genetic Algorithms. *Scientific American , 267* (1), 66-72.

[8] Rachlin H. (1995). Self-Control: Beyond commitment, *Behavioral and Brain Sciences*, 18, 109-159

[9] Barto, A. G. (2007). Temporal difference learning. *Scholarpedia , 2* (11), 1604.

[10] Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Ph.D. thesis, University of Cambridge, Department of Psychology, Cambridge, U.K.

[11] Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems.* University of Cambridge, Department of Engineering. Cambridge, U.K., Technical Report CUED/F-INFENG/TR 166.

[12] Axelrod, R., & Hamilton, W. D. (1981). The Evolution of Cooperation. *Science , 211* (4489), 1390-1396.

[13] Niv Y., Daw N.D., & Dayan P. (2006). Choice values. *Nature Neuroscience*, 9(8): 987-988.

[14] Morris G., Nevet A., Arkadir D., Vaadia E., & Bergman H. (2006). Midbrain dopamine neurons encode decisions for future action. *Nature Neuroscience*, 9(8):1057–1063.

[15] Roesch M., Calu D., & Schoenbaum G. (2007). Dopamine neurons encode the better option in rats deciding between differently delayed or sized rewards. *Nature Neuroscience*, 10(12):1615–1624.

Παράρτημα Α

Πηγαίος κώδικας υλοποίησης με αλγόριθμο μάθησης Q-learning

Player.java

```
1  public class Player {
2
3      private double learning_rate;
4      private double epsilon;
5      private double discount;
6      private double[][] Q_table;
7      private double[] payoff_matrix;
8      private int current_action;
9
10
11     public Player() {
12         this.Q_table = new double[4][2];
13         this.payoff_matrix = new double[4];
14     }
15
16
17     public void setDiscount(double value) {
18         this.discount = value;
19     }
20
21
22     public double getDiscount() {
23         return this.discount;
24     }
25
26
27     public void setEpsilon(double value) {
28         this.epsilon = value;
29     }
```

```
30
31
32      public double getEpsilon() {
33          return this.epsilon;
34      }
35
36
37      public void setLearning_rate(double value) {
38          this.learning_rate = value;
39      }
40
41
42      public double getLearning_rate() {
43          return this.learning_rate;
44      }
45
46
47      public void setQ_table(int row, int column, double value) {
48          this.Q_table[row][column] = value;
49      }
50
51
52      public double getQ_table(int row, int column) {
53          return this.Q_table[row][column];
54      }
55
56
57      public void setPayoff_matrix(double value1, double value2, double value3,
58      double value4) {
59          payoff_matrix[0] = value1;
60          payoff_matrix[1] = value2;
61          payoff_matrix[2] = value3;
62          payoff_matrix[3] = value4;
63      }
64
65
66      public double getPayoff_matrix(int row) {
67          return this.payoff_matrix[row];
68      }
```

```
69
70
71      public void setCurrent_action(int action) {
72          this.current_action = action;
73      }
74
75
76      public int getCurrent_action() {
77          return this.current_action;
78      }
79
80
81      public int choose_action(int state) {
82          int action;
83          double random = Math.random();
84
85          if (random < epsilon) {    //explore
86              action = (int) (Math.random() * 2);  // 0(cooperate) or 1(defect)
87          } else {    //exploit
88              // find best known action
89              if (this.getQ_table(state, 0) > this.getQ_table(state, 1))
90                  action = 0;
91              else if (this.getQ_table(state, 0) < this.getQ_table(state, 1))
92                  action = 1;
93              else
94                  action = (int) (Math.random() * 2); // 0(cooperate) 1(defect)
95          }
96
97          return action;
98      }
99
100
101     public void update_Q(int current_state, int next_state) {
102         double max, Q;
103
104         // find maximum value from Q table
105         if (this.getQ_table(next_state, 0) >= this.getQ_table(next_state, 1))
106             max = this.getQ_table(next_state, 0);
107         else
```

```
108              max = this.getQ_table(next_state, 1);
109
110          // Calculate Q and set value in player's Q table
111          Q = ((1 - this.learning_rate) * this.getQ_table(current_state,
112               this.getCurrent_action()))      +      (this.learning_rate      *
113         (this.getPayoff_matrix(next_state) + (this.getDiscount() * max)));
114
115          this.setQ_table(current_state, this.getCurrent_action(), Q);
116      }
117
118  }
```

IPD_Game.java

```
1   public class IPD_Game {
2
3       private Player lower;
4       private Player higher;
5       private double[][][] states_results;
6       private double[][][] payoff_results;
7       private int episodes;
8       private int trials;
9
10
11      public IPD_Game() {
12
13      }
14
15
16      public void setEpisodes(int episodes) {
17          this.episodes = episodes;
18      }
19
20
21      public int getEpisodes() {
22          return this.episodes;
23      }
24
25
26      public void setTrials(int trials) {
27          this.trials = trials;
28      }
29
30
31      public int getTrials() {
32          return this.trials;
33      }
34
35
36      private int get_state(int action_lower, int action_higher) {
37          int next_state;
```

```
38
39          if (action_lower == 0 && action_higher == 0)
40              next_state = 0;
41          else if (action_lower == 0 && action_higher == 1)
42              next_state = 2;
43          else if (action_lower == 1 && action_higher == 0)
44              next_state = 1;
45          else
46              next_state = 3;
47
48          return next_state;
49      }
50
51
52      private void update_states_results(int episode, int state, int trial) {
53          states_results[episode][trial][0]    =    states_results[episode    -
54  1][trial][0];
55          states_results[episode][trial][1]    =    states_results[episode    -
56  1][trial][1];
57          states_results[episode][trial][2]    =    states_results[episode    -
58  1][trial][2];
59          states_results[episode][trial][3]    =    states_results[episode    -
60  1][trial][3];
61          states_results[episode][trial][state]++;
62      }
63
64
65      private  void  update_payoff_results(int  episode,  double  lower_payoff,
66  double higher_payoff, int trial) {
67          payoff_results[episode][trial][0]    =    payoff_results[episode    -
68  1][trial][0];
69          payoff_results[episode][trial][1]    =    payoff_results[episode    -
70  1][trial][1];
71
72          payoff_results[episode][trial][0] += lower_payoff;
73          payoff_results[episode][trial][1] += higher_payoff;
74      }
75
76
```

```
77      private void statistics() {
78          int e, t, s, p;
79          double sumAll_states;
80          double sumAll_payoffs;
81
82          for (e = 0; e < episodes; e++) {
83              for (t = 0; t < trials; t++) {    // find sum of states visits from
84  each trial of current episode and
85              // save in last column
86                  for (s = 0; s < 4; s++) {     // states
87                      states_results[e][trials][s] += states_results[e][t][s];
88                  }
89
90                  for (p = 0; p < 2; p++) {     // payoffs
91                      payoff_results[e][trials][p] += payoff_results[e][t][p];
92                  }
93              }
94
95              sumAll_states = 0;
96              sumAll_payoffs = 0;
97
98              for (s = 0; s < 4; s++) {     // sum of state fields in last column
99  (#trials+1)
100                 sumAll_states += states_results[e][trials][s];
101             }
102
103             for (s = 0; s < 4; s++) {     // normalise
104                 states_results[e][trials][s] /= sumAll_states;
105             }
106
107             for (p = 0; p < 2; p++) {      // sum of payoff fields in last
108 column (#trials+1)
109                 sumAll_payoffs += payoff_results[e][trials][p];
110             }
111         }
112     }
113
114
115     /**
```

```
116         * Play Iterated Prisoner's Dilemma game
117         * Returns the percentage of cc done by the individual
118         */
119        public double[] play(Individual individual) {
120            int initial_state, current_state, next_state; // Takes values 0-3
121            int i;
122            double[] percentages = new double[4];
123            states_results = new double[episodes][trials + 1][4];
124            payoff_results = new double[episodes][trials + 1][2];
125
126            double differential_bias = individual.getObject_parameter(4);
127
128            for (int t = 0; t < trials; t++) {
129
130                // Create "Lower"  player
131                lower = new Player();
132                lower.setDiscount(individual.getObject_parameter(0));
133                lower.setLearning_rate(individual.getObject_parameter(2));
134                lower.setEpsilon(0.1);
135                lower.setPayoff_matrix(4,   5   -   differential_bias,   -3   +
136    differential_bias, -2);
137
138                // Create "Higher" player
139                higher = new Player();
140                higher.setDiscount(individual.getObject_parameter(1));
141                higher.setLearning_rate(individual.getObject_parameter(3));
142                higher.setEpsilon(0.1);
143                higher.setPayoff_matrix(4,   -3   -   differential_bias,   5   +
144    differential_bias, -2);
145
146                initial_state = (int) (Math.random() * 4);  // Set initial state
147    of game
148
149                current_state = initial_state;
150
151                states_results[0][t][current_state]++;
152
153                // Training
154                for (i = 1; i < episodes; i++) {
```

```
                lower.setCurrent_action(lower.choose_action(current_state));
// Lower player chooses action

higher.setCurrent_action(higher.choose_action(current_state));     //   Higher
player chooses action

                next_state       =       get_state(lower.getCurrent_action(),
higher.getCurrent_action());  // Calculate next
                // state of game

                lower.update_Q(current_state, next_state);  // Update Lower's
Q table
                higher.update_Q(current_state, next_state); // Update Higher's
Q table

                current_state = next_state; // Set current state of game

                update_states_results(i, current_state, t);
                update_payoff_results(i,
lower.getPayoff_matrix(current_state),
                        higher.getPayoff_matrix(current_state), t);
            }
        }

        statistics();

        // Calculate visited percentage of each state
        for (i = 0; i < 4; i++) {
            percentages[i] = states_results[episodes - 1][trials][i];
        }

        return percentages;
    }
}
```

Individual.java

```
1  public class Individual {
2
3      private double[] object_parameters;     // 0: discount factor, 1: learning
4  rate, 2: differential bias
5      private double[] state_visits;
6      private double objective_function;  // percentage of cc
7
8
9      public Individual(int parameters_num) {
10         this.object_parameters = new double[parameters_num];
11         this.state_visits = new double[4];
12     }
13
14
15     public void setObject_parameter(int index, double value) {
16         this.object_parameters[index] = value;
17     }
18
19
20     public double getObject_parameter(int index) {
21         return this.object_parameters[index];
22     }
23
24
25     public double[] getObject_parameters() {
26         return this.object_parameters;
27     }
28
29
30     public void setState_visits(int index, double value) {
31         this.state_visits[index] = value;
32     }
33
34
35     public double getState_visits(int index) {
36         return this.state_visits[index];
37     }
```

```
38
39
40      public void setObjective_function(double value) {
41          this.objective_function = value;
42      }
43
44
45      public double getObjective_function() {
46          return this.objective_function;
47      }
48
49
50      public String toString() {
51          return String.format("Objective function: %f", objective_function);
52      }
53
54  }
```

EvolutionStrategies.java

```
1  import java.util.Arrays;
2
3
4  public class EvolutionStrategies {
5
6      private int mu;     // parents population length
7      private int lambda; // number of offspring to be generated
8      private int ro;     // mixing number
9      private double sigma;   // gaussian for mutation
10     private int parameters_num;
11     private Individual[] parent_population;
12     private Individual[] offspring_population;
13
14
15     public EvolutionStrategies() {
16
17     }
18
19
20     public int getMu() {
21         return this.mu;
22     }
23
24
25     public void setMu(int mu) {
26         this.mu = mu;
27     }
28
29
30     public int getLambda() {
31         return this.lambda;
32     }
33
34
35     public void setLambda(int lambda) {
36         this.lambda = lambda;
37     }
```

```
38
39
40      public int getRo() {
41          return this.ro;
42      }
43
44
45      public void setRo(int ro) {
46          this.ro = ro;
47      }
48
49
50      public double getSigma() {
51          return this.sigma;
52      }
53
54
55      public void setSigma(double sigma) {
56          this.sigma = sigma;
57      }
58
59
60      public void setParameters_num(int number) {
61          this.parameters_num = number;
62      }
63
64
65      public int getParameters_num() {
66          return this.parameters_num;
67      }
68
69
70      public Individual[] getParent_Population() {
71          return this.parent_population;
72      }
73
74
75      public Individual getParent_Individual(int index) {
76          return this.parent_population[index];
```

```
77      }
78
79
80      public Individual[] getOffspring_population() {
81          return this.offspring_population;
82      }
83
84
85      public Individual getOffspring_Individual(int index) {
86          return this.offspring_population[index];
87      }
88
89
90      public Individual create_clone(int original_index) {
91          int i;
92
93          Individual original = this.parent_population[original_index];
94          Individual                clone                =                new
95  Individual(original.getObject_parameters().length);
96
97          // Copy object parameters
98          for (i = 0; i < original.getObject_parameters().length; i++) {
99              clone.setObject_parameter(i, original.getObject_parameter(i));
100         }
101
102         // Copy objective function
103         clone.setObjective_function(original.getObjective_function());
104
105         return clone;
106     }
107
108
109     public void initialize_population() {
110         int i;
111         this.parent_population = new Individual[this.mu];   // create parents
112 array
113         this.offspring_population = new Individual[this.lambda];    // create
114 offspring array
115
```

```
116         for (i = 0; i < this.parent_population.length; i++) {   // for each
117 individual in parent population
118             parent_population[i] = new Individual(parameters_num);  // create
119 individual object
120             this.parent_population[i].setObject_parameter(0,       0.3       *
121 Math.random());    // discount factor: value 0 - 0
122             // .3 for lower part of the brain
123             this.parent_population[i].setObject_parameter(1, 0.7 + (1 - 0.7) *
124 Math.random());  // discount factor:
125             // value 0.7 - 1 for higher part of the brain
126             this.parent_population[i].setObject_parameter(2, Math.random());
127             this.parent_population[i].setObject_parameter(3, Math.random());
128             this.parent_population[i].setObject_parameter(4, Math.random());
129         }
130     }
131
132
133     /**
134      * Generates clones of each individual in parent population
135      */
136     public void generate_offspring() {
137         int i, j, index = 0;
138         Individual offspring;
139
140         for (i = 0; i < this.parent_population.length; i++) {   // for each
141 individual in parent population
142             for (j = 0; j < (lambda / this.parent_population.length); j++) {
143 // generate desired number of clones of
144             // individual 'i'
145                 offspring  =  this.create_clone(i);       //  create  clone  of
146 individual i
147                 this.offspring_population[index] = offspring;   // save it in
148 offspring population
149                 index++;
150             }
151         }
152     }
153
154
```

```
155     public static double uniform(double minValue, double maxValue) {
156         return (Math.random() * (maxValue - minValue)) + minValue;
157     }
158 
159 
160     public static double gaussian(double mu, double sigma) {
161         double p = 1.0, p1 = 1.0, p2;
162 
163         while (p >= 1.0) {
164             p1 = uniform(-1, 1);
165             p2 = uniform(-1, 1);
166             p = p1 * p1 + p2 * p2;
167         }
168 
169         return mu + sigma * p1 * Math.sqrt(-2.0 * Math.log(p) / p);
170     }
171 
172 
173     /**
174      * Random changes in individual's genotype
175      */
176     public void mutation() {
177         int i, j;
178         double current_value, mutated_value;
179 
180         for (i = 0; i < offspring_population.length; i++) {   // for each
181 individual in offspring population
182 
183             // Mutate discount factor for lower part of the brain
184             current_value                                                 =
185 this.getOffspring_Individual(i).getObject_parameter(0);
186 
187             if (current_value <= 0.295) {  // if current parameter value is <=
188 0.295 then mutate it *******
189                 mutated_value = current_value + gaussian(0, sigma);
190 
191                 while (mutated_value > 0.3 || mutated_value < 0) {   // find
192 mutated value between 0 and 0.3 (which
193                 // is the limit)
```

```
                    mutated_value = current_value + gaussian(0, sigma);
                }
            } else {  // if current parameter value is = 0.3 then do not mutate
it
                mutated_value = current_value;
            }
            this.getOffspring_Individual(i).setObject_parameter(0,
mutated_value);


            // Mutate discount factor for lower part of the brain
            current_value                                                =
this.getOffspring_Individual(i).getObject_parameter(1);

            if (current_value <= 0.995) {  // if current parameter value is <
0.995 then mutate it *******
                mutated_value = current_value + gaussian(0, sigma);

                while (mutated_value > 1 || mutated_value < 0.7) {   // find
mutated value between 0.7 and 1 (which
                // is the limit)
                    mutated_value = current_value + gaussian(0, sigma);
                }
            } else {  // if current parameter value is = 1 then do not mutate
it
                mutated_value = current_value;
            }
            this.getOffspring_Individual(i).setObject_parameter(1,
mutated_value);


            // Mutate discount factor for higher part of the brain, learning
rate, differential bias
            for (j = 2; j < this.parameters_num; j++) {
                // Mutate discount factor for higher part of the brain
                current_value                                            =
this.getOffspring_Individual(i).getObject_parameter(j);

```

```
                if (current_value <= 0.995) {  // if current parameter value
is < 1 then mutate it *******
                    mutated_value = current_value + gaussian(0, sigma);

                    while (mutated_value > 1 || mutated_value < 0) {   // find
mutated value between 0 and 1 (which
                    // is the limit)
                        mutated_value = current_value + gaussian(0, sigma);
                    }
                } else {  // if current parameter value is = 1 then do not
mutate it
                    mutated_value = current_value;
                }
                this.getOffspring_Individual(i).setObject_parameter(j,
mutated_value);
            }
        }
    }


    /**
     * Calculates the fitness of each individual.
     * Objective function: f = CC - CD - DC - DD
     */
    public double calc_ObjectiveFunction(double[] percentages) {
        double f;

        f = percentages[0] - percentages[1] - percentages[2] - percentages[3];

        return f;
    }


    /**
     * Select mu individuals for next generation
     * plus-selection (select fittest individuals from parents and offspring
population)
     */
    public void selection() {
```

```
271         int i;
272         Individual[] population = new Individual[this.lambda + this.mu];
273 
274         // Put individuals from parent and offspring population in the same
275 array for sorting
276         for (i = 0; i < this.mu; i++) {
277             population[i] = this.parent_population[i];
278         }
279 
280         for (i = 0; i < this.lambda; i++) {
281             population[i + this.mu] = this.offspring_population[i];
282         }
283 
284         Arrays.sort(population, new FitnessComparator()); // Sort offspring
285 array by individuals' fitness (objective
286         // function)
287 
288         for (i = 0; i < this.mu; i++) {     // select mu fittest individuals
289 from offspring population and set as
290         // parents for next generation
291             this.parent_population[i] = population[i];
292         }
293     }
294 
295 
296     public static void printToConsole(Object[] objects) {
297         for (Object e : objects) {
298             System.out.println(e.toString());
299         }
300         System.out.println();
301     }
302 
303 }
```

FitnessComparator.java

```
1  import java.util.Comparator;
2
3
4  public class FitnessComparator implements Comparator<Individual> {
5
6      @Override
7      public int compare(Individual individual1, Individual individual2) {
8          double fitness1 = individual1.getObjective_function();
9          double fitness2 = individual2.getObjective_function();
10
11         if (fitness1 > fitness2)
12             return -1;
13         else if (fitness1 < fitness2)
14             return 1;
15         else
16             return 0;
17     }
18
19 }
```

MainProgram.java

```
1  import java.io.*;
2  import java.util.Arrays;
3
4
5  public class MainProgram {
6
7      public     static     void     printToFile_states_statistics(double[][][][]
8  state_visits, int generations, int evolutions) {
9          int e, g, s;
10         double cc, cd, dc, dd;
11
12         // Find avg results
13         for (g = 0; g < generations; g++) {
14             cc = 0;
15             cd = 0;
16             dc = 0;
17             dd = 0;
18             for (e = 0; e < evolutions; e++) {
19                 cc += state_visits[g][e][0];
20                 cd += state_visits[g][e][1];
21                 dc += state_visits[g][e][2];
22                 dd += state_visits[g][e][3];
23             }
24             state_visits[g][evolutions][0] = cc / evolutions;
25             state_visits[g][evolutions][1] = cd / evolutions;
26             state_visits[g][evolutions][2] = dc / evolutions;
27             state_visits[g][evolutions][3] = dd / evolutions;
28         }
29
30         // write results to file
31         try {
32             File file_cc = new File("cooperation_results.txt");
33             File file_states = new File("states_results.txt");
34
35             // if file does not exist, then create it
36             if (!file_cc.exists()) {
37                 file_cc.createNewFile();
```

```
            }

            if (!file_states.exists()) {
                file_states.createNewFile();
            }

            FileWriter fw_cc = new FileWriter(file_cc.getAbsoluteFile());
            BufferedWriter bw_cc = new BufferedWriter(fw_cc);

            FileWriter              fw_states              =              new
FileWriter(file_states.getAbsoluteFile());
            BufferedWriter bw_states = new BufferedWriter(fw_states);

            for (g = 0; g < generations; g++) {
                bw_cc.write(Double.toString(state_visits[g][evolutions][0]));
                bw_cc.write("\n");

                for (s = 0; s < 4; s++) {

bw_states.write(Double.toString(state_visits[g][evolutions][s]));
                    bw_states.write(" ");
                }
                bw_states.write("\n");
            }

            bw_cc.close();
            bw_states.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }


    public  static  void  printToFile_bestIndividual_statistics(double[][][]
bestIndividual_parameters, int generations,
                                                                int  evolutions)
{
        int e, g, s;
```

```
76          double      discount_low,      discount_high,      learningRate_low,
77  learningRate_high, differential_bias;
78
79          // Find avg results
80          for (g = 0; g < generations; g++) {
81              discount_low = 0;
82              discount_high = 0;
83              learningRate_low = 0;
84              learningRate_high = 0;
85              differential_bias = 0;
86              for (e = 0; e < evolutions; e++) {
87                  discount_low += bestIndividual_parameters[g][e][0];
88                  discount_high += bestIndividual_parameters[g][e][1];
89                  learningRate_low += bestIndividual_parameters[g][e][2];
90                  learningRate_high += bestIndividual_parameters[g][e][3];
91                  differential_bias += bestIndividual_parameters[g][e][4];
92              }
93              bestIndividual_parameters[g][evolutions][0]   =   discount_low   /
94  evolutions;
95              bestIndividual_parameters[g][evolutions][1]   =   discount_high   /
96  evolutions;
97              bestIndividual_parameters[g][evolutions][2] = learningRate_low /
98  evolutions;
99              bestIndividual_parameters[g][evolutions][3] = learningRate_high /
100 evolutions;
101             bestIndividual_parameters[g][evolutions][4] = differential_bias /
102 evolutions;
103         }
104
105         // write results to file
106         try {
107             File file = new File("bestIndividual_parameters.txt");
108
109             // if file does not exist, then create it
110             if (!file.exists()) {
111                 file.createNewFile();
112             }
113
114             FileWriter fw = new FileWriter(file.getAbsoluteFile());
```

```
115             BufferedWriter bw = new BufferedWriter(fw);
116 
117             for (g = 0; g < generations; g++) {
118                 for (s = 0; s < 5; s++) {
119 
120 bw.write(Double.toString(bestIndividual_parameters[g][evolutions][s]));
121                     bw.write(" ");
122                 }
123                 bw.write("\n");
124             }
125 
126             bw.close();
127         } catch (IOException ex) {
128             ex.printStackTrace();
129         }
130     }
131 
132 
133     public static void main(String[] args) {
134         int i, g, e, s;
135         double[] percentages;
136 
137         // Set parameters
138         int evolutions = 5;
139         int generations = 100;
140 
141         double[][][] state_visits = new double[generations][evolutions +
142 1][4];
143         double[][][] bestIndividual_parameters = new
144 double[generations][evolutions + 1][5];
145 
146         for (e = 0; e < evolutions; e++) {
147             // Create Evolution Strategies object and set parameters
148             EvolutionStrategies ES = new EvolutionStrategies();
149             ES.setLambda(70);
150             ES.setMu(10);
151             ES.setRo(1);
152             ES.setSigma(0.1);
153             ES.setParameters_num(5);
```

```
154
155             //  Create  Iterated  Prisoner's  Dilemma  Game  object  and  set
156 parameters
157             IPD_Game IPD = new IPD_Game();
158             IPD.setEpisodes(1000);
159             IPD.setTrials(100);
160
161             // Initialize population
162             ES.initialize_population();
163
164             // Each individual plays the IPD game
165             for (i = 0; i < ES.getParent_Population().length; i++) {
166                 percentages  =  IPD.play(ES.getParent_Individual(i));      //
167 individual plays IPD game and percentage of
168                 // each state is returned
169
170 ES.getParent_Individual(i).setObjective_function(ES.calc_ObjectiveFunction(pe
171 rcentages));    //
172                 // calculate objective function
173                 for (s = 0; s < 4; s++) {    // for each state
174                     ES.getParent_Individual(i).setState_visits(s,
175 percentages[s]);
176                 }
177             }
178
179             // Evolution
180             for (g = 0; g < generations; g++) {
181
182                 // States statistics
183                 for (i = 0; i < ES.getParent_Population().length; i++) {  //
184 for each individual in parent population
185                     for (s = 0; s < 4; s++) {
186                         state_visits[g][e][s]                              +=
187 ES.getParent_Individual(i).getState_visits(s);
188                     }
189                 }
190                 for (s = 0; s < 4; s++) {
191                     state_visits[g][e][s]                              /=
192 ES.getParent_Population().length;
```

```
193                 }
194
195                 // Best individual of current generation parameters statistics
196                 Arrays.sort(ES.getParent_Population(),                    new
197 FitnessComparator());    // sort parent_population array
198                 // by fitness
199                 for (i = 0; i < ES.getParameters_num(); i++) {     // get
200 parameters of best individual (index 0)
201                     bestIndividual_parameters[g][e][i]                     =
202 ES.getParent_Individual(0).getObject_parameter(i);
203                 }
204
205                 // Generate offspring
206                 ES.generate_offspring();
207
208                 // Mutate offspring
209                 ES.mutation();
210
211                 // Each offspring individual plays the IPD game
212                 for (i = 0; i < ES.getOffspring_population().length; i++) {
213                     percentages   =   IPD.play(ES.getOffspring_Individual(i));
214 // individual plays IPD game and
215                     // percentage of each state is returned
216
217 ES.getOffspring_Individual(i).setObjective_function(ES.calc_ObjectiveFunction
218 (percentages));
219                     // calculate objective function
220                     for (s = 0; s < 4; s++) {
221                         ES.getOffspring_Individual(i).setState_visits(s,
222 percentages[s]);
223                     }
224                 }
225
226                 // (Plus Selection) Select mu individuals for next generation
227                 ES.selection();
228             }
229         }
230
231         // Produce and print to file results
```

```
232             printToFile_states_statistics(state_visits, generations, evolutions);
233             printToFile_bestIndividual_statistics(bestIndividual_parameters,
234     generations, evolutions);
235         }
236     
237     }
```

Παράρτημα Β

Πηγαίος κώδικας υλοποίησης με αλγόριθμο μάθησης SARSA

Player.java

```
1  public class Player {
2
3      private double learning_rate;
4      private double epsilon;
5      private double discount;
6      private double[][] Q_table;
7      private double[] payoff_matrix;
8      private int current_action;
9      private int next_action;
10
11
12     public Player() {
13         this.Q_table = new double[4][2];
14         this.payoff_matrix = new double[4];
15     }
16
17
18     public void setDiscount(double value) {
19         this.discount = value;
20     }
21
22
23     public double getDiscount() {
24         return this.discount;
25     }
26
27
28     public void setEpsilon(double value) {
29         this.epsilon = value;
```

```
30      }
31
32
33      public double getEpsilon() {
34          return this.epsilon;
35      }
36
37
38      public void setLearning_rate(double value) {
39          this.learning_rate = value;
40      }
41
42
43      public double getLearning_rate() {
44          return this.learning_rate;
45      }
46
47
48      public void setQ_table(int row, int column, double value) {
49          this.Q_table[row][column] = value;
50      }
51
52
53      public double getQ_table(int row, int column) {
54          return this.Q_table[row][column];
55      }
56
57
58      public void setPayoff_matrix(double value1, double value2, double value3,
59  double value4) {
60          payoff_matrix[0] = value1;
61          payoff_matrix[1] = value2;
62          payoff_matrix[2] = value3;
63          payoff_matrix[3] = value4;
64      }
65
66
67      public double getPayoff_matrix(int row) {
68          return this.payoff_matrix[row];
```

```
69      }
70
71
72      public void setCurrent_action(int action) {
73          this.current_action = action;
74      }
75
76
77      public int getCurrent_action() {
78          return this.current_action;
79      }
80
81
82      public void setNext_action(int action) {
83          this.next_action = action;
84      }
85
86
87      public int getNext_action() {
88          return this.next_action;
89      }
90
91
92      public int choose_action(int state) {
93          int action;
94          double random = Math.random();
95
96          if (random < epsilon) {    //explore
97              action = (int) (Math.random() * 2);    // values: 0(cooperate) or
98  1(defect)
99          } else {    //exploit
100             // find best known action
101             if (this.getQ_table(state, 0) > this.getQ_table(state, 1))
102                 action = 0;
103             else if (this.getQ_table(state, 0) < this.getQ_table(state, 1))
104                 action = 1;
105             else
106                 action = (int) (Math.random() * 2);    // values: 0(cooperate)
107 or 1(defect)
```

```
108          }
109
110          return action;
111      }
112
113
114      public void update_Q(int current_state, int next_state) {
115          double Q;
116
117          // Calculate Q and set value in player's Q table
118          Q  =  ((1  -  this.learning_rate)  *  this.getQ_table(current_state,
119  this.getCurrent_action())) + (this
120                  .learning_rate   *   (this.getPayoff_matrix(current_state)   +
121  (this.getDiscount() * this.getQ_table
122                  (next_state, this.getNext_action()))));
123          this.setQ_table(current_state, this.getCurrent_action(), Q);
124      }
125
126  }
```

IPD_Game.java

```
1  import java.io.*;
2
3
4  public class IPD_Game {
5
6      private Player lower;
7      private Player higher;
8      private double[][][] states_results;
9      private double[][][] payoff_results;
10     private int episodes;
11     private int trials;
12
13
14     public IPD_Game() {
15
16     }
17
18
19     public void setEpisodes(int episodes) {
20         this.episodes = episodes;
21     }
22
23
24     public int getEpisodes() {
25         return this.episodes;
26     }
27
28
29     public void setTrials(int trials) {
30         this.trials = trials;
31     }
32
33
34     public int getTrials() {
35         return this.trials;
36     }
37
```

```
38
39      private int get_state(int action_lower, int action_higher) {
40          int next_state;
41
42          if (action_lower == 0 && action_higher == 0)
43              next_state = 0;
44          else if (action_lower == 0 && action_higher == 1)
45              next_state = 2;
46          else if (action_lower == 1 && action_higher == 0)
47              next_state = 1;
48          else
49              next_state = 3;
50
51          return next_state;
52      }
53
54
55      private void update_states_results(int episode, int state, int trial) {
56          states_results[episode][trial][0]    =    states_results[episode    -
57  1][trial][0];
58          states_results[episode][trial][1]    =    states_results[episode    -
59  1][trial][1];
60          states_results[episode][trial][2]    =    states_results[episode    -
61  1][trial][2];
62          states_results[episode][trial][3]    =    states_results[episode    -
63  1][trial][3];
64          states_results[episode][trial][state]++;
65      }
66
67
68      private  void  update_payoff_results(int  episode,  double  lower_payoff,
69  double higher_payoff, int trial) {
70          payoff_results[episode][trial][0]    =    payoff_results[episode    -
71  1][trial][0];
72          payoff_results[episode][trial][1]    =    payoff_results[episode    -
73  1][trial][1];
74
75          payoff_results[episode][trial][0] += lower_payoff;
76          payoff_results[episode][trial][1] += higher_payoff;
```

```
77      }
78
79
80      private void statistics() {
81          int e, t, s, p;
82          double sumAll_states;
83          double sumAll_payoffs;
84
85          for (e = 0; e < episodes; e++) {
86              for (t = 0; t < trials; t++) {    // find sum of states visits from
87  each trial of current episode and
88              // save in last column
89                  for (s = 0; s < 4; s++) {     // states
90                      states_results[e][trials][s] += states_results[e][t][s];
91                  }
92
93                  for (p = 0; p < 2; p++) {     // payoffs
94                      payoff_results[e][trials][p] += payoff_results[e][t][p];
95                  }
96              }
97
98              sumAll_states = 0;
99              sumAll_payoffs = 0;
100
101             for (s = 0; s < 4; s++) {     // sum of state fields in last column
102 (#trials+1)
103                 sumAll_states += states_results[e][trials][s];
104             }
105
106             for (s = 0; s < 4; s++) {     // normalise
107                 states_results[e][trials][s] /= sumAll_states;
108             }
109
110             for (p = 0; p < 2; p++) {      // sum of payoff fields in last
111 column (#trials+1)
112                 sumAll_payoffs += payoff_results[e][trials][p];
113             }
114         }
115     }
```

```
116
117
118       /**
119        * Play Iterated Prisoner's Dilemma game
120        * Returns the percentage of cc done by the individual
121        */
122       public double[] play(Individual individual) {
123           int initial_state, current_state, next_state; // Takes values 0-3
124           int i;
125           boolean is_first;
126           double[] percentages = new double[4];
127           states_results = new double[episodes][trials + 1][4];
128           payoff_results = new double[episodes][trials + 1][2];
129
130           double differential_bias = individual.getObject_parameter(4);
131
132           for (int t = 0; t < trials; t++) {
133
134               // Create "Lower"  player
135               lower = new Player();
136               lower.setDiscount(individual.getObject_parameter(0));
137               lower.setLearning_rate(individual.getObject_parameter(2));
138               lower.setEpsilon(0.1);
139               lower.setPayoff_matrix(4,   5   -   differential_bias,   -3   +
140   differential_bias, -2);
141
142               // Create "Higher" player
143               higher = new Player();
144               higher.setDiscount(individual.getObject_parameter(1));
145               higher.setLearning_rate(individual.getObject_parameter(3));
146               higher.setEpsilon(0.1);
147               higher.setPayoff_matrix(4,   -3   -   differential_bias,   5   +
148   differential_bias, -2);
149
150               initial_state = (int) (Math.random() * 4);  // Set initial state
151   of game
152
153               current_state = initial_state;
154
```

```
155             states_results[0][t][current_state]++;
156
157             is_first = true;
158
159             lower.setNext_action(lower.choose_action(current_state));
160             higher.setNext_action(higher.choose_action(current_state));
161
162             // Training
163             for (i = 1; i < episodes; i++) {
164                 lower.setCurrent_action(lower.choose_action(current_state));
165 // Lower player chooses action
166
167 higher.setCurrent_action(higher.choose_action(current_state));     //  Higher
168 player chooses action
169
170                 next_state       =       get_state(lower.getCurrent_action(),
171 higher.getCurrent_action());  // Calculate next
172                 // state of game
173
174                 lower.setNext_action(lower.choose_action(next_state));
175                 higher.setNext_action(higher.choose_action(next_state));
176
177                 if (!is_first) {
178                     lower.update_Q(current_state,  next_state);    //  Update
179 Lower's Q table
180                     higher.update_Q(current_state,  next_state);  //  Update
181 Higher's Q table
182                 }
183
184                 current_state = next_state; // Set current state of game
185
186                 update_states_results(i, current_state, t);
187                 update_payoff_results(i,
188 lower.getPayoff_matrix(current_state),
189                         higher.getPayoff_matrix(current_state), t);
190
191                 is_first = false;
192             }
193         }
```

```
194
195           statistics();
196
197           // Calculate visited percentage of each state
198           for (i = 0; i < 4; i++) {
199               percentages[i] = states_results[episodes - 1][trials][i];
200           }
201
202           return percentages;
203       }
204   }
```

Individual.java

```
1   public class Individual {
2
3       private double[] object_parameters;     // 0: discount factor, 1: learning
4   rate, 2: differential bias
5       private double[] state_visits;
6       private double objective_function;  // percentage of cc
7
8
9       public Individual(int parameters_num) {
10          this.object_parameters = new double[parameters_num];
11          this.state_visits = new double[4];
12      }
13
14
15      public void setObject_parameter(int index, double value) {
16          this.object_parameters[index] = value;
17      }
18
19
20      public double getObject_parameter(int index) {
21          return this.object_parameters[index];
22      }
23
24
25      public double[] getObject_parameters() {
26          return this.object_parameters;
27      }
28
29
30      public void setState_visits(int index, double value) {
31          this.state_visits[index] = value;
32      }
33
34
35      public double getState_visits(int index) {
36          return this.state_visits[index];
37      }
```

```
38
39
40      public void setObjective_function(double value) {
41          this.objective_function = value;
42      }
43
44
45      public double getObjective_function() {
46          return this.objective_function;
47      }
48
49
50      public String toString() {
51          return String.format("Objective function: %f", objective_function);
52      }
53  }
```

EvolutionStrategies.java

```
1   public class EvolutionStrategies {
2
3       private int mu;     // parents population length
4       private int lambda; // number of offspring to be generated
5       private int ro;     // mixing number
6       private double sigma;   // gaussian for mutation
7       private int parameters_num;
8       private Individual[] parent_population;
9       private Individual[] offspring_population;
10
11
12      public EvolutionStrategies() {
13
14      }
15
16
17      public int getMu() {
18          return this.mu;
19      }
20
21
22      public void setMu(int mu) {
23          this.mu = mu;
24      }
25
26
27      public int getLambda() {
28          return this.lambda;
29      }
30
31
32      public void setLambda(int lambda) {
33          this.lambda = lambda;
34      }
35
36
37      public int getRo() {
```

```
38          return this.ro;
39      }
40
41
42      public void setRo(int ro) {
43          this.ro = ro;
44      }
45
46
47      public double getSigma() {
48          return this.sigma;
49      }
50
51
52      public void setSigma(double sigma) {
53          this.sigma = sigma;
54      }
55
56
57      public void setParameters_num(int number) {
58          this.parameters_num = number;
59      }
60
61
62      public int getParameters_num() {
63          return this.parameters_num;
64      }
65
66
67      public Individual[] getParent_Population() {
68          return this.parent_population;
69      }
70
71
72      public Individual getParent_Individual(int index) {
73          return this.parent_population[index];
74      }
75
76
```

```
77      public Individual[] getOffspring_population() {
78          return this.offspring_population;
79      }
80
81
82      public Individual getOffspring_Individual(int index) {
83          return this.offspring_population[index];
84      }
85
86
87      public Individual create_clone(int original_index) {
88          int i;
89
90          Individual original = this.parent_population[original_index];
91          Individual                clone                =                new
92  Individual(original.getObject_parameters().length);
93
94          // Copy object parameters
95          for (i = 0; i < original.getObject_parameters().length; i++) {
96              clone.setObject_parameter(i, original.getObject_parameter(i));
97          }
98
99          // Copy objective function
100         clone.setObjective_function(original.getObjective_function());
101
102         return clone;
103     }
104
105
106     public void initialize_population() {
107         int i;
108         this.parent_population = new Individual[this.mu];   // create parents
109 array
110         this.offspring_population = new Individual[this.lambda];    // create
111 offspring array
112
113         for (i = 0; i < this.parent_population.length; i++) {   // for each
114 individual in parent population
```

```
115             parent_population[i] = new Individual(parameters_num);  // create
116 individual object
117             this.parent_population[i].setObject_parameter(0,       0.3       *
118 Math.random());    // discount factor: value 0 - 0
119             // .3 for lower part of the brain
120             this.parent_population[i].setObject_parameter(1, 0.7 + (1 - 0.7) *
121 Math.random());  // discount factor:
122             // value 0.7 - 1 for higher part of the brain
123             this.parent_population[i].setObject_parameter(2, Math.random());
124             this.parent_population[i].setObject_parameter(3, Math.random());
125             this.parent_population[i].setObject_parameter(4, Math.random());
126         }
127     }
128
129
130     /**
131      * Generates clones of each individual in parent population
132      */
133     public void generate_offspring() {
134         int i, j, index = 0;
135         Individual offspring;
136
137         for (i = 0; i < this.parent_population.length; i++) {   // for each
138 individual in parent population
139             for (j = 0; j < (lambda / this.parent_population.length); j++) {
140 // generate desired number of clones of
141             // individual 'i'
142                 offspring  =  this.create_clone(i);       //  create  clone  of
143 individual i
144                 this.offspring_population[index] = offspring;   // save it in
145 offspring population
146                 index++;
147             }
148         }
149     }
150
151
152     public static double uniform(double minValue, double maxValue) {
153         return (Math.random() * (maxValue - minValue)) + minValue;
```

```
154     }
155 
156 
157     public static double gaussian(double mu, double sigma) {
158         double p = 1.0, p1 = 1.0, p2;
159 
160         while (p >= 1.0) {
161             p1 = uniform(-1, 1);
162             p2 = uniform(-1, 1);
163             p = p1 * p1 + p2 * p2;
164         }
165 
166         return mu + sigma * p1 * Math.sqrt(-2.0 * Math.log(p) / p);
167     }
168 
169 
170     /**
171      * Random changes in individual's genotype
172      */
173     public void mutation() {
174         int i, j;
175         double current_value, mutated_value;
176 
177         // Mutate parameters
178         for (i = 0; i < offspring_population.length; i++) {   // for each
179 individual in offspring population
180 
181             // Mutate discount factor for lower part of the brain
182             current_value                                                 =
183 this.getOffspring_Individual(i).getObject_parameter(0);
184 
185             if (current_value <= 0.295) {  // if current parameter value is <=
186 0.295 then mutate it *******
187                 mutated_value = current_value + gaussian(0, sigma);
188 
189                 while (mutated_value > 0.3 || mutated_value < 0) {   // find
190 mutated value between 0 and 0.3 (which
191                 // is the limit)
192                     mutated_value = current_value + gaussian(0, sigma);
```

```
                }
            } else {  // if current parameter value is = 0.3 then do not mutate
it
                mutated_value = current_value;
            }
            this.getOffspring_Individual(i).setObject_parameter(0,
mutated_value);


            // Mutate discount factor for lower part of the brain
            current_value                                                    =
this.getOffspring_Individual(i).getObject_parameter(1);

            if (current_value <= 0.995) {  // if current parameter value is <
0.995 then mutate it *******
                mutated_value = current_value + gaussian(0, sigma);

                while (mutated_value > 1 || mutated_value < 0.7) {   // find
mutated value between 0.7 and 1 (which
                // is the limit)
                    mutated_value = current_value + gaussian(0, sigma);
                }
            } else {  // if current parameter value is = 1 then do not mutate
it
                mutated_value = current_value;
            }
            this.getOffspring_Individual(i).setObject_parameter(1,
mutated_value);


            // Mutate discount factor for higher part of the brain, learning
rate, differential bias
            for (j = 2; j < this.parameters_num; j++) {
                // Mutate discount factor for higher part of the brain
                current_value                                                =
this.getOffspring_Individual(i).getObject_parameter(j);

                if (current_value <= 0.995) {  // if current parameter value
is < 1 then mutate it *******
```

```
                    mutated_value = current_value + gaussian(0, sigma);

                    while (mutated_value > 1 || mutated_value < 0) {   // find
mutated value between 0 and 1 (which
                    // is the limit)
                        mutated_value = current_value + gaussian(0, sigma);
                    }
                } else {  // if current parameter value is = 1 then do not
mutate it
                    mutated_value = current_value;
                }
                this.getOffspring_Individual(i).setObject_parameter(j,
mutated_value);
            }
        }
    }


    /**
     * Calculates the fitness of each individual.
     * Objective function: f = CC - CD - DC - DD
     */
    public double calc_ObjectiveFunction(double[] percentages) {
        double f;

        f = percentages[0] - percentages[1] - percentages[2] - percentages[3];

        return f;
    }


    /**
     * Select mu individuals for next generation
     * plus-selection (select fittest individuals from parents and offspring
population)
     */
    public void selection() {
        int i;
        Individual[] population = new Individual[this.lambda + this.mu];
```

```
271
272         // Put individuals from parent and offspring population in the same
273 array for sorting
274         for (i = 0; i < this.mu; i++) {
275             population[i] = this.parent_population[i];
276         }
277
278         for (i = 0; i < this.lambda; i++) {
279             population[i + this.mu] = this.offspring_population[i];
280         }
281
282         Arrays.sort(population, new FitnessComparator()); // Sort offspring
283 array by individuals' fitness (objective
284         // function)
285
286         for (i = 0; i < this.mu; i++) {     // select mu fittest individuals
287 from offspring population and set as
288         // parents for next generation
289             this.parent_population[i] = population[i];
290         }
291     }
292
293
294     public static void printToConsole(Object[] objects) {
295         for (Object e : objects) {
296             System.out.println(e.toString());
297         }
298         System.out.println();
299     }
300
301 }
```

FitnessComparator.java

```
1  import java.util.Comparator;
2
3
4  public class FitnessComparator implements Comparator<Individual> {
5
6      @Override
7      public int compare(Individual individual1, Individual individual2) {
8          double fitness1 = individual1.getObjective_function();
9          double fitness2 = individual2.getObjective_function();
10
11         if (fitness1 > fitness2)
12             return -1;
13         else if (fitness1 < fitness2)
14             return 1;
15         else
16             return 0;
17     }
18
19 }
```

MainProgram.java

```
1  import java.io.*;
2  import java.util.Arrays;
3
4
5  public class MainProgram {
6
7      public     static     void     printToFile_states_statistics(double[][][][]
8  state_visits, int generations, int evolutions) {
9          int e, g, s;
10         double cc, cd, dc, dd;
11
12         // Find avg results
13         for (g = 0; g < generations; g++) {
14             cc = 0;
15             cd = 0;
16             dc = 0;
17             dd = 0;
18             for (e = 0; e < evolutions; e++) {
19                 cc += state_visits[g][e][0];
20                 cd += state_visits[g][e][1];
21                 dc += state_visits[g][e][2];
22                 dd += state_visits[g][e][3];
23             }
24             state_visits[g][evolutions][0] = cc / evolutions;
25             state_visits[g][evolutions][1] = cd / evolutions;
26             state_visits[g][evolutions][2] = dc / evolutions;
27             state_visits[g][evolutions][3] = dd / evolutions;
28         }
29
30         // write results to file
31         try {
32             File file_cc = new File("cooperation_results.txt");
33             File file_states = new File("states_results.txt");
34
35             // if file does not exist, then create it
36             if (!file_cc.exists()) {
37                 file_cc.createNewFile();
```

```
            }

            if (!file_states.exists()) {
                file_states.createNewFile();
            }

            FileWriter fw_cc = new FileWriter(file_cc.getAbsoluteFile());
            BufferedWriter bw_cc = new BufferedWriter(fw_cc);

            FileWriter              fw_states              =              new
FileWriter(file_states.getAbsoluteFile());
            BufferedWriter bw_states = new BufferedWriter(fw_states);

            for (g = 0; g < generations; g++) {
                bw_cc.write(Double.toString(state_visits[g][evolutions][0]));
                bw_cc.write("\n");

                for (s = 0; s < 4; s++) {

bw_states.write(Double.toString(state_visits[g][evolutions][s]));
                    bw_states.write(" ");
                }
                bw_states.write("\n");
            }

            bw_cc.close();
            bw_states.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    }


    public  static  void  printToFile_bestIndividual_statistics(double[][][]
bestIndividual_parameters, int generations,
                                                              int  evolutions)
{
        int e, g, s;
```

```
77          double      discount_low,      discount_high,      learningRate_low,
78  learningRate_high, differential_bias;
79
80          // Find avg results
81          for (g = 0; g < generations; g++) {
82              discount_low = 0;
83              discount_high = 0;
84              learningRate_low = 0;
85              learningRate_high = 0;
86              differential_bias = 0;
87              for (e = 0; e < evolutions; e++) {
88                  discount_low += bestIndividual_parameters[g][e][0];
89                  discount_high += bestIndividual_parameters[g][e][1];
90                  learningRate_low += bestIndividual_parameters[g][e][2];
91                  learningRate_high += bestIndividual_parameters[g][e][3];
92                  differential_bias += bestIndividual_parameters[g][e][4];
93              }
94              bestIndividual_parameters[g][evolutions][0]   =   discount_low   /
95  evolutions;
96              bestIndividual_parameters[g][evolutions][1]   =   discount_high   /
97  evolutions;
98              bestIndividual_parameters[g][evolutions][2] = learningRate_low /
99  evolutions;
100             bestIndividual_parameters[g][evolutions][3] = learningRate_high /
101 evolutions;
102             bestIndividual_parameters[g][evolutions][4] = differential_bias /
103 evolutions;
104         }
105
106         // write results to file
107         try {
108             File file = new File("bestIndividual_parameters.txt");
109
110             // if file does not exist, then create it
111             if (!file.exists()) {
112                 file.createNewFile();
113             }
114
115             FileWriter fw = new FileWriter(file.getAbsoluteFile());
```

```
116             BufferedWriter bw = new BufferedWriter(fw);
117 
118             for (g = 0; g < generations; g++) {
119                 for (s = 0; s < 5; s++) {
120 
121 bw.write(Double.toString(bestIndividual_parameters[g][evolutions][s]));
122                     bw.write(" ");
123                 }
124                 bw.write("\n");
125             }
126 
127             bw.close();
128         } catch (IOException ex) {
129             ex.printStackTrace();
130         }
131 
132     }
133 
134 
135     public static void main(String[] args) {
136         int i, g, e, s;
137         double[] percentages;
138 
139         // Set parameters
140         int evolutions = 5;
141         int generations = 100;
142 
143         double[][][] state_visits = new double[generations][evolutions +
144 1][4];
145         double[][][][] bestIndividual_parameters = new
146 double[generations][evolutions + 1][5];
147 
148         for (e = 0; e < evolutions; e++) {
149             // Create Evolution Strategies object and set parameters
150             EvolutionStrategies ES = new EvolutionStrategies();
151             ES.setLambda(70);
152             ES.setMu(10);
153             ES.setRo(1);
154             ES.setSigma(0.1);
```

```
155             ES.setParameters_num(5);
156 
157             //  Create  Iterated  Prisoner's  Dilemma  Game  object  and  set
158 parameters
159             IPD_Game IPD = new IPD_Game();
160             IPD.setEpisodes(1000);
161             IPD.setTrials(100);
162 
163             // Initialize population
164             ES.initialize_population();
165 
166             // Each individual plays the IPD game
167             for (i = 0; i < ES.getParent_Population().length; i++) {
168                 percentages  =  IPD.play(ES.getParent_Individual(i));      //
169 individual plays IPD game and percentage of
170                 // each state is returned
171 
172 ES.getParent_Individual(i).setObjective_function(ES.calc_ObjectiveFunction(pe
173 rcentages));    //
174                 // calculate objective function
175                 for (s = 0; s < 4; s++) {    // for each state
176                     ES.getParent_Individual(i).setState_visits(s,
177 percentages[s]);
178                 }
179             }
180 
181             // Evolution
182             for (g = 0; g < generations; g++) {
183 
184                 // States statistics
185                 for (i = 0; i < ES.getParent_Population().length; i++) {  //
186 for each individual in parent population
187                     for (s = 0; s < 4; s++) {
188                         state_visits[g][e][s]                                 +=
189 ES.getParent_Individual(i).getState_visits(s);
190                     }
191                 }
192                 for (s = 0; s < 4; s++) {
```

```
                    state_visits[g][e][s]                                   /=
ES.getParent_Population().length;
                }

                // Best individual of current generation parameters statistics
                Arrays.sort(ES.getParent_Population(),                      new
FitnessComparator());    // sort parent_population array
                // by fitness
                for (i = 0; i < ES.getParameters_num(); i++) {      // get
parameters of best individual (index 0)
                    bestIndividual_parameters[g][e][i]                       =
ES.getParent_Individual(0).getObject_parameter(i);
                }

                // Generate offspring
                ES.generate_offspring();

                // Mutate offspring
                ES.mutation();

                // Each offspring individual plays the IPD game
                for (i = 0; i < ES.getOffspring_population().length; i++) {
                    percentages   =   IPD.play(ES.getOffspring_Individual(i));
// individual plays IPD game and
                    // percentage of each state is returned

ES.getOffspring_Individual(i).setObjective_function(ES.calc_ObjectiveFunction
(percentages));
                    // calculate objective function
                    for (s = 0; s < 4; s++) {
                        ES.getOffspring_Individual(i).setState_visits(s,
percentages[s]);
                    }
                }

                // (Plus Selection) Select mu individuals for next generation
                ES.selection();
            }
        }
```

```
232
233           // Produce and print to file results
234           printToFile_states_statistics(state_visits, generations, evolutions);
235           printToFile_bestIndividual_statistics(bestIndividual_parameters,
236   generations, evolutions);
237       }
```