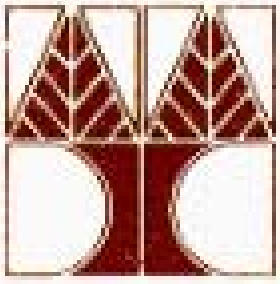
Ατομική Διπλωματική Εργασία

ΑΝΑΠΤΥΞΗ ΕΝΟΣ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΠΑΙΓΝΙΔΙΟΥ

Θεοφάνης Παυλίδης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2004

ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΠΤΥΞΗ ΕΝΟΣ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΠΑΙΓΝΙΔΙΟΥ

Θεοφάνης Παυλίδης

Επιβλέπουσα Καθηγήτρια
Άννα Φιλίππου

Η ατομική αυτή διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτηση του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστήμιου Κύπρου

Ιούνιος 2004

Ευχαριστίες

Με την ευκαιρία που μου δόθηκε με την μελέτη και υλοποίηση της διπλωματικής αυτής θα ήθελα να ευχαριστήσω θερμά την επιβλέπουσα καθηγήτρια μου Κυρία Άννα Φιλίππου για την ανεκτίμητη βοήθεια της στην μελέτη και ανάπτυξη του συστήματος μου, καθώς και την υπομονή που μου έδειξε όλο αυτόν τον καιρό. Επίσης θα ήθελα να ευχαριστήσω όλο τον κόσμο που με βοήθησε με ηθική αλλά και υλική υποστήριξη τους κατά την διάρκεια της διπλωματικής μου.

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου για την υποστήριξη τους που έδειξαν τα τελευταία τέσσερα χρόνια. Χωρίς αυτούς δεν θα ήταν δυνατή η φοίτηση μου στο Πανεπιστήμιο.

Σας ευχαριστώ όλους θερμά. Χωρίς εσάς δεν θα ήταν δυνατή η ολοκλήρωση αυτής της διπλωματικής αλλά ούτε και των σπουδών μου.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της διπλωματικής

Στη σημερινή εποχή οι υπολογιστές έχουν μπει σχεδόν σε κάθε σπίτι και αποτελούν ένα μέρος της ζωής μας. Τα τελευταία χρόνια έκανε και την εμφάνιση του το διαδίκτυο το οποίο ήταν και ένας σημαντικός παράγοντας εξάπλωσης τον ηλεκτρονικών υπολογιστών σε κάθε σπίτι.

Το μεγαλύτερο μέρος του πληθυσμού της Κύπρου έχει καθημερινά πρόσβαση στο διαδίκτυο και κάνει συχνή χρήση email και άλλων υπηρεσιών που προσφέρονται. Συνεχώς βλέπουμε τη δημιουργία νέων και διαφόρων ιστοσελίδων που χρησιμοποιούν το διαδίκτυο σαν το μέσο που τους βοηθά να προωθήσουν τα προϊόντα τους αλλά και ότι άλλες υπηρεσίες επιθυμούν.

Τα κατανεμημένα παιγνίδια[3,8,9] είναι μια σχετικά νέα τεχνολογία για την Κύπρο η οποία κάνει χρήση το διαδίκτυο και τον υπολογιστή του κάθε χρήστη που λαμβάνει μέρος σε ένα τέτοιο παιγνίδι, για να προσφέρει στους εμπλεκόμενους τη δυνατότητα να ανταγωνισθούν από μακριά ο ένας με τον άλλον, χωρίς απαραίτητα να γνωρίζονται μεταξύ τους ή να βρίσκονται στον ίδιο χώρο.

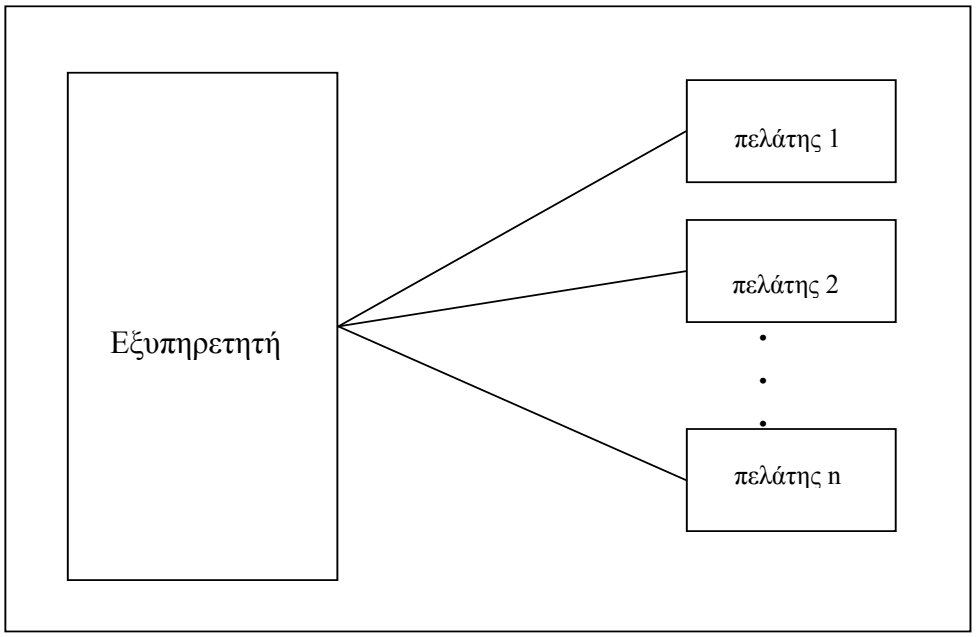
Σκοπός μας λοιπόν είναι η ανάπτυξη και υλοποίηση ενός κατανεμημένου παιγνιδιού το οποίο θα βρίσκεται σε κάποιο δικτυακό τόπο δικής μας επιλογής, και το οποίο θα επιτρέπει σε χρήστες που βρίσκονται σε υπολογιστές σε κάποια απομακρυσμένη τοποθεσία, οι οποίοι έχουν πρόσβαση στο διαδίκτυο, να ανταγωνισθούν μεταξύ τους.

1.2 Απλή περιγραφή

Στο παιγνίδι αυτό ο κάθε χρήστης από το σπίτι του μπορεί να παίξει το παιγνίδι εναντίον άλλων χρηστών που βρίσκονται στο διαδίκτυο σε κάποια άλλη τοποθεσία Υπάρχει δηλαδή μεταξύ του κάθε χρήστη επικοινωνία με τον εξυπηρετητή του παιγνιδιού ο οποίος συντονίζει τις κινήσεις του κάθε παίχτη και είναι υπεύθυνος για την κατανομή και αποστολή της πληροφορίας προς τον κάθε παίχτη. Ο εξυπηρετητής δηλαδή είναι αυτός που λαμβάνει από κάθε παίχτη τις κινήσεις του και οποιαδήποτε άλλη πληροφορία χρειάζεται, και στην συνέχεια αφού επεξεργαστεί τα δεδομένα, αποστέλλει αυτές τις πληροφορίες πίσω στους παίκτες με τέτοιο τρόπο ώστε το παιγνίδι να είναι δίκαιο και αξιόπιστο.

Εναπόκειται στο υπολογιστή του κάθε χρήστη να λάβει αυτές τις πληροφορίες και να σχεδιάσει την κατάσταση του παιγνιδιού στον υπολογιστή του χρήστη ώστε ο χρήστης να έχει άποψη του παιγνιδιού όπως έχει διαμορφωθεί μέχρι εκείνη την στιγμή. Η πληροφορία δηλαδή θα κατανέμεται από το εξυπηρετητή προς τους χρήστες και οι υπολογιστές των χρηστών παράλληλα (ταυτόχρονα) θα χρησιμοποιούν την πληροφορία αυτή.

Η τοπολογία του συστήματος φαίνεται στο ποιο κάτω σχήμα



Σχήμα 1.1 : Η γενική μορφή του συστήματος.

Εμείς έχουμε επικεντρωθούμε στην βασική δομή ενός συστήματος το οποίο μπορεί να στηρίξει ένα κατανεμημένο παιγνίδι. Αρχικά δηλαδή έχουμε δει το κομμάτι που κάνει την επικοινωνία μεταξύ κάθε πελάτη και εξυπηρετητή. Στην συνέχεια επιλέξαμε ένα παιγνίδι το οποίο πληροί τις συνθήκες που είχαμε ορίσει, το οποίο χρησιμοποιήθηκε για τον έλεγχο του συστήματος επικοινωνίας. Το παιγνίδι αυτό ενσωματώθηκε μαζί με το σύστημα επικοινωνίας για να μας δώσει το τελικό προϊόν. Αξίζει να σημειωθεί ότι κατά την υλοποίηση του συστήματος επικοινωνίας δεν γνωρίζαμε το ακριβές παιγνίδι που θα υλοποιούσαμε, αλλά γνωρίζαμε μόνο το είδος του παιγνιδιού. Το παιγνίδι ενσωματώθηκε μετά στο σύστημα επικοινωνίας αφού το δεύτερο ολοκληρώθηκε με επιτυχία.

1.3 Περιγραφή Συστήματος

Το σύστημα μας αποτελείται από τον εξυπηρετητή (Server) και τους πελάτες (Clients). Η κάθε οντότητα (Εξυπηρετητής και πελάτες), τρέχει σε διαφορετικές μηχανές. Μπορεί όμως ακόμα η κάθε οντότητα να τρέχει και στις ίδιες μηχανές. Ο εξυπηρετητής είναι αυτός που συγχρονίζει το παιγνίδι και «ενώνει» όλους του πελάτες μαζί. Ο κάθε πελάτης επικοινωνεί με τον εξυπηρετητή μόνο, και κάθε φορά στέλλει τις πληροφορίες του και παίρνει πίσω από τον εξυπηρετητή την νέα κατάσταση.

Για την ανάλυση του συστήματος μας, χωρίζουμε το σύστημα μας στην περιγραφή και υλοποίηση δύο κύριων κομματιών.

α) Του κομματιού το οποίο είναι υπεύθυνο για την επικοινωνία ανάμεσα στον εξυπηρετητή και των πελάτη, και

β) το παιγνίδι, που από την πλευρά του εξυπηρετητή έχουμε το κομμάτι που υπολογίζει την νέα κατάσταση, και από την πλευρά του πελάτη τον σχεδιασμό στην οθόνη του χρήστη αλλά και την αλληλεπίδραση του χρήστη με το παιγνίδι.

Αυτά τα δύο μέρη θα σχεδιαστούν και θα υλοποιηθούν σαν δύο ξεχωριστές οντότητες και στην συνέχεια θα ενωθούν μαζί για να μας δώσουν το τελικό και ολοκληρωμένο σύστημα.

1.4 Εργαλεία που χρησιμοποιήθηκαν

Τα εργαλεία που χρησιμοποιήθηκαν στην διπλωματική είναι η άλγεβρα διεργασιών[4] ACSR[2], η οποία επιτρέπει την ανάλυση και επαλήθευση συστημάτων πραγματικού

χρόνου, η γλώσσα προγραμματισμού Java[1] η οποία είναι μια γλώσσα αντικειμενοστρεφούς προγραμματισμού και επιτρέπει να τρέχουν παράλληλα (concurrently) διάφορα νήματα (threads), το οποίο είναι κρίσιμο στην υλοποίηση του συστήματος μας, η Java 3D API[5] η οποία επιτρέπει την δημιουργία τρισδιάστατων γραφικών τα οποία μπορούν να χρησιμοποιούνται είτε σε κανονικά προγράμματα είτε σε απλά applets τα οποία μπορούν να φορτωθούν σε ένα φυλλομετρητή (browser), καθώς και το πρόγραμμα επαλήθευσης συστημάτων Versa[6,7], το οποίο μας βοηθά να επαληθεύσουμε μεγάλα συστήματα γραμμένα στην ACSR ώστε να βρούμε λάθη και αδιέξοδα σε αυτά σύντομα

Η ACSR μας βοήθησε στην μελέτη και ανάπτυξη του συστήματος, καθώς και στη εύρεση πολλών λαθών στην αρχή των προδιαγραφών. Με την χρήση του συστήματος επαλήθευσης versa τρέξαμε το σύστημα προδιαγραφών μας και βρήκαμε λάθη στην υλοποίηση πολύ έγκαιρα. Επίσης κατά τον σχεδιασμό και την υλοποίηση βοήθησε πολύ η σχέση μεταξύ του τρόπου με τον οποίο η ACSR βλέπει και χειρίζεται τις διάφορες διεργασίες (process) και ο τρόπος με το οποίο συγχρονίζει και υποστηρίζει διάφορα νήματα στο ίδιο πρόγραμμα

Η Java παρόλο του ότι είναι μια γλώσσα προγραμματισμού εντούτοις παρέχει μηχανισμούς οι οποίοι μπορούν να παρομοιαστούν με ένα σύστημα πραγματικού χρόνου. Μπορούμε να δούμε την Java σαν ένα λειτουργικό σύστημα στο οποίο τρέχουν πολλές διεργασίες. Αυτή η λειτουργία μας παρέχει την δυνατότητα να έχουμε «παράλληλη» επεξεργασία των διαφόρων αιτημάτων των πελατών στην μηχανή που τρέχει ο εξυπηρετητής, εξυπηρετώντας έτσι πολλούς πελάτες με μεγαλύτερη ευκολία και χωρίς μεγάλη πιθανότητα λάθους αλλά και αδιεξόδου (deadlock) στο σύστημα. Η Java επίσης προσφέρει μηχανισμούς για εύκολη επέκταση του συστήματος (κληρονομικότητα), καθώς και μηχανισμούς για εύρεση και ανάκτηση λαθών προσφέροντας έτσι μεγαλύτερη ευρωστία (exceptions).

Για την υλοποίηση του παιγνιδιού κάναμε την χρήση της τεχνολογίας Java 3D API. Η Java 3D API αποτελεί επέκταση της απλής έκδοσης Javas και αποτελεί τεχνολογία για ανάπτυξη 3D Graphics στην Java. Για να μπορούν οι χρήστες να την χρησιμοποιήσουν θα πρέπει να έχουν εγκατεστημένα στην μηχανή τους επιπλέον από τον διερμηνέα της Javas και τον διερμηνέα της Java 3D API.

1.5 Δομή της διπλωματικής

Η διπλωματική αυτή είναι χωρισμένη σε εφτά κεφάλαια και τρία παραρτήματα.

Το δεύτερο κεφάλαιο περιγράφει τα εργαλεία που έχουν χρησιμοποιηθεί. Αναλύεται η άλγεβρα διεργασιών ACSR και εξηγούνται τα άλλα εργαλεία που χρησιμοποιήσαμε καθώς και οι λόγοι για τους οποίους τα επιλέξαμε για την δουλειά αυτή.

Το τρίτο κεφάλαιο περιέχει τις απαιτήσεις του συστήματος. Αυτές χωρίζονται στις απαιτήσεις του συστήματος επικοινωνίας, στις απαιτήσεις του συστήματος παιγνιδιού και τέλος στις απαιτήσεις του ολοκληρωμένου συστήματος.

Το τέταρτο κεφάλαιο περιέχει τις προδιαγραφές του συστήματος. Οι προδιαγραφές χωρίζονται και εδώ σε προδιαγραφές του συστήματος επικοινωνίας, προδιαγραφές του συστήματος παιγνιδιού, και τέλος στις προδιαγραφές του ολοκληρωμένου συστήματος. Επιπλέον σε αυτό το κεφάλαιο δίνονται και οι προδιαγραφές του συστήματος επικοινωνίας σε ACSR.

Τα πέμπτο κεφάλαιο είναι αυτό της μετάφρασης και του σχεδιασμού. Εκεί πρώτα εξηγούμε πως μπορούμε να πάμε από την άλγεβρα διεργασιών ACSR στην γλώσσα προγραμματισμού Java, και στην συνέχεια σχεδιάζουμε το σύστημα επικοινωνίας βάση αυτής της μετάφρασης. Στην συνέχεια δίνουμε την σχεδίαση του συστήματος παιγνιδιού, και στην συνέχεια την σχεδίαση του ολοκληρωμένου συστήματος και πως αυτά τα δύο, σύστημα επικοινωνίας και σύστημα παιγνιδιού, ενώνονται μαζί.

Το έκτο κεφάλαιο είναι αυτό της υλοποίησης. Εκεί παρουσιάζεται το τελικό σύστημα το οποίο έχει υλοποιηθεί. Δίνονται διάφορα screenshot εξηγώντας το τρόπο με τον οποίο δουλεύει. Επιπλέον εξηγούνται οι απαιτήσεις που υπάρχουν γι να παιχτεί το παιγνίδι από πλευράς του εξυπηρετητή, και από πλευράς του πελάτη

Το έβδομο κεφάλαιο περιέχει τα συμπεράσματά μας. Εκεί δίνονται οι γενικές εντυπώσεις από την διπλωματική, η αξιόλογη των αποτελεσμάτων αλλά και τα συμπεράσματά μας από την διπλωματική αυτή

Το παράρτημα Α περιέχει τον κώδικα σε Java του συστήματος επικοινωνίας., το παράρτημα Β περιέχει τον κώδικα σε Java του συστήματος παιγνιδιού, το παράρτημα Γ περιέχει τον κώδικα σε ACSR του συστήματος επικοινωνίας γραμμένο για το σύστημα επαλήθευσης Versa

Κεφάλαιο 2

Γενικά Εργαλεία που χρησιμοποιήθηκαν

2.1 Εργαλεία που χρησιμοποιηθήκαν

Τα εργαλεία που χρησιμοποιήθηκαν στην διπλωματική όπως έχουμε αναφέρει και στην εισαγωγή, είναι η άλγεβρα διεργασιών ACSR, η οποία επιτρέπει την ανάλυση και επαλήθευση συστημάτων πραγματικού χρόνου, η γλώσσα προγραμματισμού Java η οποία είναι μια γλώσσα αντικειμενοστρεφούς προγραμματισμού και η οποία επιτρέπει να τρέχουν παράλληλα (concurrently) διάφορα νήματα (threads), η Java 3D API η οποία επιτρέπει την δημιουργία τρισδιάστατων γραφικών τα οποία μπορούν να χρησιμοποιούνται είτε σε κανονικά προγράμματα είτε σε απλά applets τα οποία μπορούν να φορτωθούν σε ένα φυλλομετρητή (browser), καθώς και το πρόγραμμα επαλήθευσης συστημάτων Versa.

2.2 ACSR

2.2.1 Εισαγωγή

Η ACSR είναι μια άλγεβρα διεργασιών η οποία με την χρήση τυπικών μεθόδων μπορεί να βοηθήσει την μελέτη και ανάλυση συστημάτων πραγματικού χρόνου. Οι τυπικές μέθοδοι μεταχειρίζονται ένα σύστημα ως κάποιο μαθηματικό αντικείμενο, παρέχοντας μας έτσι μαθηματικά μοντέλα για περιγραφή και μελέτη των διάφορών ιδιοτήτων και συμπεριφορών αυτών των αντικειμένων.

Άλγεβρες διεργασιών σαν την ACSR έχουν δημιουργηθεί για την περιγραφή και ανάλυση συστημάτων επικοινωνίας και συγχρονισμού. Οι άλγεβρες διεργασιών βασίζονται στην ιδέα ότι οι δύο ποιο σημαντικές έννοιες στην κατανόηση μεγάλων δυναμικών και πολύπλοκων συστημάτων είναι ο συγχρονισμός και επικοινωνία μεταξύ των διαφόρων μερών ενός συστήματος.

Υπάρχουν πολλά πλεονεκτήματα στην χρήση των τυπικών μεθόδων για μελέτη και ανάλυση συστημάτων πραγματικού χρόνου. Αρχικά μας βοηθούν να βρούμε γρήγορα ασάφειες, ασυνέπειες, και ατέλειες σε απαιτήσεις διάφορων συστημάτων. Επίσης με την χρήση προγραμμάτων ανάλυσης και επαλήθευσης συστημάτων όπως την Versa έχουμε αυτοματοποιημένη ανάλυση και εύρεση λαθών μεγάλων συστημάτων. Ακόμα έχουμε την δυνατότητα για εύκολη αλλαγή στο σύστημα χωρίς μεγάλα έξοδα.

Η ACSR βασίζεται στην ιδέα ότι ένα σύστημα πραγματικού χρόνου, περιέχει ένα σύνολο από διεργασίες που επικοινωνούν η μια με την άλλη χρησιμοποιώντας κοινούς πόρους για εκτέλεση και συγχρονισμό μεταξύ τους. Οι διεργασίες σε ένα σύστημα χρησιμοποιούν τους πόρους του συστήματος μέσω χρονικών ενεργειών (time actions) και συγχρονίζονται μεταξύ τους μέσω στιγμιαίων γεγονότων (instantaneous events).

Η εκτέλεση μιας χρονικής ενέργειας ενεργεί για μια χρονική στιγμή, (δηλαδή το πέρασμα μια μονάδας χρόνου υποθέτοντας ένας κοινό ρολόι για όλο το σύστημα), να εκτελεστεί και να καταναλώσει μια μονάδα πόρου. Επίσης όλοι οι πόροι που καταναλώνονται κάθε χρονική στιγμή καταναλώνονται ταυτόχρονα. Δηλαδή υπάρχει συγχρονισμός μεταξύ των πράξεων που θα εκτελεστούν σε κάθε χρονική στιγμή.

Μέσα στο πέρασμα μιας χρονικής στιγμής μπορούμε να έχουμε μη πεπερασμένα γεγονότα (events). Η εκτέλεση ενός γεγονός γίνεται χωρίς να περάσει χρόνος. Όλα τα γεγονότα εκτελούνται ασύγχρονα εκτός όταν δύο διεργασίες συγχρονίζονται πάνω σε κοινά γεγονότα

2.2.2 Επισκόπηση της ACSR

Η ACSR όπως κάθε άλγεβρα διεργασιών περιλαμβάνει

α) ένα σύνολο από πράξεις και συντακτικούς κανόνες για κατασκευή διεργασιών

β) μια σημασιολογική αντιστοιχία η οποία αντιστοιχεί νόημα ή ερμηνεία στις διεργασίες

γ) μια έννοια ισοδυναμίας ή μερικής διάταξης μεταξύ διεργασιών

δ) ένα σύνολο από αλγεβρικούς νόμους οι οποίοι επιτρέπουν συντακτικό χειρισμό διεργασιών

Στην συνέχεια θα δούμε διάφορα ζητήματα που αφορούν την σχεδίαση για ανάπτυξη ενός συστήματος πραγματικού χρόνου με την χρήση της ACSR. Αυτά είναι :

Α) Χρόνος

Η σημαντικότερη πτυχή μιας άλγεβρας διεργασιών πραγματικού χρόνου όπως η ACSR, είναι η ικανότητα της να μπορεί να συλλαμβάνει την χρονική στιγμή που συμβαίνει ένα γεγονός. Το πεδίο τιμών του χρόνου είναι ολικά ορισμένο π.χ. για κάθε δύο στοιχεία τα οποία δεν είναι ίσα το ένα θεωρείται ότι γίνεται πριν από το άλλο. Επίσης το πεδίο ορισμό του χρόνου μπορεί να είναι είτε διακριτό είτε συνεχές.

Το πεδίο ορισμού του χρόνου είναι συνεχές αν και μόνο αν υπάρχει κάποιο στοιχείο μεταξύ οποιοδήποτε δύο στοιχείων μέσα στο πεδίο ορισμού, ενώ είναι διακριτό αν και μόνο αν κάθε στοιχείο στο πεδίο ορισμού έχει ένα μοναδικό διάδοχο, δηλαδή τα διάφορα events συμβαίνουν σε σταθερά χρονικά διαστήματα. Η ACSR όπως και πολλές άλλες άλγεβρες διεργασιών χρησιμοποιούν διακριτό πεδίο ορισμού για τον χρόνο.

Β) Πράξεις

Μια διεργασία μιας άλγεβρας διεργασιών αντιπροσωπεύει μια οντότητα του συστήματος η οποία εξελίσσεται σε μια άλλη αφού εκτελέσει κάποια πράξη. Άλγεβρες διεργασιών περιλαμβάνουν ένα σύνολο από τελεστές οι οποίοι χρησιμοποιούνται για να κατασκευάσουν σύνθετες οντότητες χρησιμοποιώντας πιο απλά κομμάτια. Πολλές

άλγεβρες διεργασιών όπως και η ACSR παρέχουν τελεστές οι οποίοι προσφέρουν τις ακόλουθες λειτουργίες :

α) τελεστής επιλογής ο οποίος χρησιμοποιείται για την επιλογή ανάμεσα σε εναλλακτικές διεργασίες. Στην ACSR χρησιμοποιείται το σύμβολο +, π.χ. έστω η διεργασία P και η διεργασία Q. Μπορούμε να διαλέξουμε μεταξύ τους γράφοντας P + Q.

β) τελεστές παράλληλης εκτέλεσης διεργασιών όπου βάζουμε δύο διεργασίες να τρέχουν παράλληλα. Στην ACSR χρησιμοποιείται το σύμβολο ||, π.χ. έστω οι διεργασίες P και Q μπορούμε να τις κάνουμε να τρέχουν παράλληλα με το να γράψουμε P || Q.

γ) τελεστής περιορισμού ή απόκρυψης για την απόκρυψη καναλιών/ονομάτων στο εσωτερικό ενός συστήματος. Στην ACSR χρησιμοποιείται το σύμβολο P\{α, β, ...ω} για περιορισμό στην διεργασία P των καναλιών α, β,... ω, και $[P]_I$ για την διεργασία P η οποία μονοπωλεί τους πόρους που βρίσκονται στο σύνολο I.

δ) τελεστής μετονομασίας των ονομάτων των καναλιών

ε) τελεστής αναδρομής για να περιγράφονται διεργασίες με άπειρες συμπεριφορές. Στην ACSR χρησιμοποιείται ο συμβολισμός rec X.P όπου η διεργασία P δυνατόν να περιέχει την μεταβλητή X. Ο τελεστής αυτός ουσιαστικά ορίζει αναδρομικά το X ως τη διεργασία P

Επιπλέον, άλγεβρες διεργασιών πραγματικού χρόνου υποστηρίζουν μια πληθώρα πράξεων οι οποίες χειρίζονται τον χρόνο. Βασικά πρόκειται για πράξεις που επιτρέπουν την εκτέλεση διεργασιών με κάποιες χρονικές ιδιότητες.

Γ) Επικοινωνία

Η πράξη της επικοινωνίας μεταξύ των διεργασιών είναι κρίσιμη ώστε να επιτυγχάνεται η συνεργασία μεταξύ παράλληλων διεργασιών. Η επικοινωνία μεταξύ των διεργασιών γίνεται είτε μεταξύ δύο διεργασιών, είτε μεταξύ n διεργασιών. Ο συγχρονισμός μεταξύ δύο διεργασιών επιτρέπει τον συγχρονισμό μεταξύ δύο εμπλεκομένων διεργασιών, π.χ. αν οι διεργασίες είναι πρόθυμες να επικοινωνούν πάνω στο κανάλι α εκτελώντας η μια την ενέργεια α και η δεύτερη την αλληλενέργεια $\bar{a}$, τότε οι δύο μπορούν να συγχρονιστούν. Αφού μιλούμε για συγχρονισμό μεταξύ δύο μόνο διεργασιών τότε για αποφυγή περαιτέρω συγχρονισμού μια τρίτης διεργασίας πάνω στις συγχρονισμένες διεργασίες, έχουμε ως αποτέλεσμα το γεγονός τ, η λεγόμενη εσωτερική ενέργεια, το

οποίο αντί του α και $\bar{a}$ εκτελείται αυτό από τις δύο διεργασίες, π.χ. έστω οι διεργασίες P και Q. Αν οι δύο διεργασίες συγχρονίζονται πάνω στο γεγονός α τότε

α. P = R

$\bar{a}$.Q = S

(α.P || $\bar{a}$.Q)\{α) = τ.(P || Q)\{α} = (R || S)\{α)

Η διεργασία R αφού εκτελέσει το γεγονός α τότε μετατρέπεται στην διεργασία P

Η διεργασία S αφού εκτελέσει το γεγονός $\bar{a}$ τότε μετατρέπεται στην διεργασία Q.

Αφού οι δύο διεργασίες R και S τρέχουν παράλληλα, λόγω του ότι έχουμε περιορισμό στο κανάλι α, τότε έχουμε την εκτέλεση του γεγονότος τ από τις διεργασίες P και Q και έχουμε συγχρονισμό. Στην συνέχεια μετατρέπονται στις διεργασίες Q και S οι οποίες συνεχίζουν να τρέχουν παράλληλα.

Αν ο περιορισμός στα κανάλια δεν υπήρχε τότε θα μπορούσαμε να έχουμε όλους τους πιθανούς συνδυασμούς

(α.P || $\bar{a}$.Q) = τ.(P || Q) + (R || $\bar{a}$.Q) + (α.P || S)

Αφού σε μια χρονική στιγμή μπορούν εκτελεστούν πολλά γεγονότα τότε θα μπορούσε να εκτελεστή το α.P και να μετατραπεί στο P, ή μπορεί να εκτελεστεί το $\bar{a}$.Q και να μετατραπεί στο Q ή μπορεί να γίνει όπως πριν, και τα δύο γεγονότα να εκτελεστούν μαζί και να έχουμε συγχρονισμό. Τότε έχουμε την εκτέλεση και των δύο διεργασιών R και S μαζί, αφού θα εκτελέσουν το κοινό γεγονός τ και στην συνέχεια θα μετατραπούν στις διεργασίες P και Q.

Δ) Αφαιρετικότητα

Σε μεγάλα πολύπλοκα συστήματα είναι σημαντικό να μπορούμε να τα περιγράφουμε με διαφορετικά επίπεδα αφαιρετικότητας. Η αφαιρετικότητα στην ACSR υποστηρίζεται με τον περιορισμό ή την απόκρυψη μια πράξης από το να χρησιμοποιηθεί στην επικοινωνία μεταξύ δύο εμπλεκομένων διεργασιών. Παρόλο ότι μια πράξη περιορίζεται ή ακόμα αποκρύπτεται από την διεργασία που χρησιμοποιείται, η ύπαρξη τους επηρεάζει την συμπεριφορά τους. Επίσης η αφαιρετικότητα υποστηρίζεται μέσω των λεγόμενων σχέσεων ισοδυναμίας, σημείο όμως που δεν έχει χρησιμοποιηθεί στα πλαίσια της διπλωματικής αυτής.

Ε) Πόροι

Παρόλο που τα διάφορα γεγονότα εκτελούνται χωρίς να περνά κάποιος χρόνος. οι χρονικές ενέργειες οι οποίες εκτελούνται σε κάθε κτύπημα του ρολογιού πρέπει να καταναλίσκουν πόρους για να προχωρήσουν από μια κατάσταση P σε μια άλλη κατάσταση Q. Για παράδειγμα αν είχαμε δύο διεργασίες P και Q, οι οποίες θα εκτελέσουν μια ενέργειες α και β αντίστοιχα, αν χρειάζεται την χρονική στιγμή t να χρησιμοποιήσουν το ίδιο πόρο, τότε οι δύο ενέργειες θα πρέπει να γίνουν σε σειρά, η μια την χρονική στιγμή t και η άλλη την χρονική στιγμή t+1, δεδομένου ότι εκείνες τις χρονικές στιγμές δεν υπάρχουν άλλες διεργασίες που θέλουν να χρησιμοποιήσουν πόρους που χρησιμοποιούν η P και Q. Άρα αφού οι δύο διεργασίες θα γίνουν σε σειρά άρα και οι πράξεις τους α και β θα γίνουν με την ίδια σειρά. Σε αντίθετη περίπτωση οι δύο διεργασίες P και Q θα μπορούσαν να εκτελεστούν παράλληλα.

Η ACSR όπως και οι περισσότερες άλγεβρες διεργασιών πραγματικού χρόνου υποστηρίζουν την έννοια των πόρων με δύο τρόπους. Ο πρώτος τρόπος είναι να δίνουν σε κάθε διεργασία που χρειάζονται την χρονική στιγμή t τους κοινούς πόρους του συνόλου I, δηλαδή η κάθε διεργασία έχει στην διάθεση τις τους πόρους που χρειάζεται ανεξαρτήτως αν του χρειάζεται εκείνη την χρονική στιγμή κάποια άλλη διεργασία. Αυτό μπορεί να συμβαίνει όταν υπάρχουν αρκετοί πόροι για όλες τις εμπλεκόμενες διεργασίες, όπως στην περίπτωση ενός συστήματος με πολλούς επεξεργαστές όπου πολλές διεργασίες θα έτρεχαν παράλληλα με την χρήστη πολλών επεξεργαστών. Ο δεύτερος τρόπος είναι να δίνεται κάθε φορά (χρονική στιγμή) ο πόρος που χρειάζεται σε μια διεργασία. Αυτό θα συνέβαινε αν είχαμε μόνο ένα επεξεργαστή σε ένα σύστημα και κάθε διεργασία θα περίμενε την σειρά της (χρονική στιγμή) ώστε να τον χρησιμοποιήσει. Η δεύτερη προσέγγιση είναι ποιο ρεαλιστική αφού δεν υποθέτει ότι έχουμε άπειρους πόρους.

ΣΤ) Προτεραιότητα

Η προτεραιότητα χρησιμοποιείται στις περισσότερες άλγεβρες διεργασιών πραγματικού χρόνου για να προσφέρει σε συστήματα με περιορισμένους πόρους την δυνατότητα για ανάθεση πόρων περισσότερο σε διεργασίες με μεγαλύτερη προτεραιότητα, π.χ. μια διεργασία μπορεί να χρειάζεται τον επεξεργαστή περισσότερο από κάποια άλλη διεργασία, άρα θα πρέπει να έχει μεγαλύτερη προτεραιότητα από αυτή.

2.2.3 Υπολογιστικό μοντέλο

Μια διεργασία στην ACSR μπορεί να έχει δύο τύπους πράξεων: α) πράξεις που καταναλώνουν χρόνο και β) πράξεις που γίνονται στιγμιαία. Οι πράξεις που καταναλώνουν χρόνο αντιπροσωπεύουν την εξέλιξη μιας διεργασίας μέσα σε μια χρονική στιγμή. Επίσης μπορούν να αντιπροσωπεύουν την κατανάλωση πόρων, π.χ. επεξεργαστή κ.τ.λ. Οι πράξεις που γίνονται στιγμιαία προσφέρουν ένα βασικό μηχανισμό για συγχρονισμό και επικοινωνία μεταξύ σύγχρονων διεργασιών.

Όταν μιλούμε για χρονικές ενέργειες εννοούμε ένα σύνολο A από δυάδες (r, p) από το πεδίο ορισμού του καρτεσιανού γινομένου R x N, όπου R είναι το σύνολο των πόρων με την ιδιότητα ότι ο κάθε πόρος εμφανίζεται ακριβώς μια φορά στο R, και N το σύνολο των φυσικών αριθμών. Δηλαδή μια χρονική ενέργεια $\{(r, p)\}$ καταναλίσκει τον πόρο $r \in R$ με προτεραιότητα $p \in N$ σε μια χρονική στιγμή του ρολογιού. Αν έχουμε την εκτέλεση $\{\} = \varnothing$, δηλαδή το πέρασμα μιας χρονικής στιγμής χωρίς τη χρήση κάποιου πόρου, αυτό αντιπροσωπεύει ένα αδράνεια (idle) για μια χρονική στιγμή, δηλαδή εκτελείται η διεργασία για μια χρονική στιγμή του ρολογιού χωρίς να κάνει τίποτε.

Γράφουμε D_R για το σύνολο όλων των χρονικών ενεργειών, και A ένα σύνολο το οποίο παίρνει τιμές από το σύνολο D_R. Με το ρ(A) εννοούμε το σύνολο των πόρων που χρησιμοποιούνται από το A, αν π.χ. $A = \{ (r_1, p_1), (r_2, p_2) \}$ τότε $\rho(A) = \{r_1, r_2\}$. Με το $\pi_r(A)$ εννοούμε την προτεραιότητα p την οποία έχει ο πόρος r στο A. π.χ. $\pi_{r1}(A) = p_1$. Εάν ο πόρος r δεν βρίσκεται στο σύνολο A τότε $\pi_r(A) = 0$.

Όταν μιλούμε για στιγμιαία γεγονότα εννοούμε δυάδες (α, p) όπου το α είναι το όνομα (label) του γεγονότος και p η προτεραιότητα του. Τα γεγονότα παίρνουν τιμές από το σύνολο $L \cup \overline{L} \cup \{\tau\}$. Το σύνολο L περιέχει όλα τα γεγονότα (ενέργειες) και το σύνολο $\overline{L}$ περιέχει τα αντίθετα γεγονότα (αλληλενέργειες) από αυτά που βρίσκονται στο L. Το ειδικό γεγονός τ δημιουργείται όταν έχουμε την ταυτόχρονη εκτέλεση ενός

γεγονός α από το L και του αντίθετού του $\bar{a}$ από το $\bar{L}$. Ισχύει φυσικά ότι $\bar{\bar{a}} = a$. Το p παίρνει τιμές από το σύνολο των φυσικών αριθμών.

Με το D_E συμβολίζουμε το πεδίο τιμών των γεγονότων. Έστω τώρα e ένα γεγονός από το D_E. Τότε με το l(e) εννοούμε το όνομα του γεγονότος και με το π(e) εννοούμε την προτεραιότητα του.

Τέλος η ένωση των ποιο πάνω συνόλων D_E και D_R μας δίνει το $D = D_E \cup D_R$ όπου είναι το πεδίο ορισμού όλων των ενεργειών της άλγεβρας διεργασιών.

Στην συνέχεια θα δούμε την σύνταξη της ACSR

2.2.4 Σύνταξη της ACSR

Μια διεργασία A στην ACSR μπορεί να μετατραπεί σε

P := NIL | A : P | (α,n). P | P + Q | P || Q | P Δ_t^a (Q, R, S) | P \ E | rec X.P | X

όπου το

α) NIL είναι η διεργασία που δεν εκτελεί καμιά ενέργεια (π.χ. βρίσκεται σε αδιέξοδο)

β) A : P είναι η διεργασία που εκτελεί την χρονική ενέργεια A σε μια χρονική στιγμή, και στην συνέχεια γίνεται η διεργασία P

γ) (α, n). P η διεργασία εκτελεί το event (α,n) χωρίς να περνάει χρόνος, και στην συνέχεια πάει στην διεργασία P.

δ) P + Q η διεργασία επιλέγει μεταξύ των δύο διεργασιών P και Q

ε) P || Q η διεργασία τρέχει παράλληλα τις διεργασίες P και Q

στ) P Δ_t^a (Q, R, S) η διεργασία P δεσμεύεται από το πεδίο Δ_t^a για χρόνο t ως εξής. Αν σε αυτό τον χρόνο εμφανιστεί κάποιο event $\bar{a}$ και εξαναγκάσει συγχρονισμό με το event α τότε η εκτέλεση περνά στην διεργασία Q, αν περάσει ο χρόνος t και γίνει timeout τότε η εκτέλεση περνά στην διεργασία R, και τέλος αν η διεργασία P διακοπεί για κάποιο λόγο τότε η εκτέλεση περνάει στην S. Το event α μπορεί να είναι οποιοδήποτε εκτός από το ιδικό event τ. Ο χρόνος t πρέπει να είναι διακριτός και μπορεί να πάρει οποιαδήποτε τιμή. Δηλαδή το όνομα του γεγονότος παίρνει τιμές από το σύνολο L και ο χρόνος $t \in N \cup \{0\} \cup \{\infty\}$

ζ) P \ E η διεργασία P δεν επιτρέπει την χρήση των γεγονότων που βρίσκονται στο σύνολο E

η) rec X.P η διεργασία X αφού εκτελέσει X.P τότε επιστρέφει στο X δίνοντας μας έτσι άπειρες αναδρομές.

Η αυστηρή ερμηνεία των τελεστών της ACSR, η οποία δόθηκε διασθητικά πιο πάνω προσδιορίζεται με ακρίβεια μέσω της σημασιολογίας της άλγεβρας διεργασιών. Η σημασιολογία αυτή δίνεται λειτουργικά (operationally) μέσω ενός συνόλου από κανόνες οι οποίοι προσδιορίζουν τη συμπεριφορά των τελεστών. Οι κανόνες ορίζονται μέσω της σχέσης $\rightarrow$: γράφοντας P $\xrightarrow{A}$ Q ή P $\xrightarrow{(a,n)}$ Q εννοούμε ότι η διεργασία P εξελίσσεται στη διεργασία Q εκτελώντας τη χρονική ενέργεια A ή το στιγμιαίο γεγονός (α,n) αντίστοιχα. Στην συνέχεια θα περιγράψουμε τους κανόνες σημασιολογίας της ACSR.

2.2.5 Κανόνες Σημασιολογίας

Στην ACSR υπάρχουν δύο κανόνες για προθεματικούς τελεστές. Ένα κανόνα για ενέργειες που χρειάζονται να καταναλώσουν χρόνο, και ένα κανόνα για στιγμιαία γεγονότα.

ActT $\dfrac{-}{A:P \xrightarrow{A} P}$ ActI $\dfrac{-}{(a,n).P \xrightarrow{(a,n)} P}$

Ο πρώτος κανόνας είναι για χρονικές ενέργειες. Μας λέει ότι η διεργασία P καταναλώνει τους πόρους που έχει το σύνολο A σε μία χρονική στιγμή, και στην συνέχεια προχωρεί στη P. Ο δεύτερος κανόνας είναι για στιγμιαία γεγονότα. Μας λέει ότι αφού εκτελεστεί στιγμιαία το γεγονός α με προτεραιότητα n τότε προχωρεί στην διεργασία P.

Για παράδειγμα αν είχαμε $\{(r_1, p_1), (r_2, p_2)\}$: P, με τον πρώτο κανόνα θα καταναλισκόνταν οι πόροι r_1 και r_2 και στην συνέχεια θα προχωρούσε στο P. Ομοίως αν είχαμε (α, n). P με τον δεύτερο κανόνα θα εκτελείτο το event (α, n) και στην συνέχεια θα πήγαινε στο P.

Για την επιλογή μεταξύ δύο διεργασιών έχουμε και εδώ δύο κανόνες. Ο πρώτος είναι για επιλογή της αριστερής διεργασίας και ο δεύτερος για την επιλογή της διεργασίας στην δεξιά επιλογή. Αυτοί οι κανόνες μοιάζουν με τους πιο πάνω κανόνες.

$$\text{ChoiceL}\ \frac{P \xrightarrow{a} P}{P + Q \xrightarrow{a} P} \qquad \text{ChoiceR}\ \frac{Q \xrightarrow{a} Q}{P + Q \xrightarrow{a} Q}$$

π.χ. Αν είχαμε (α, 7). P + {(r_1, 3), (r_2, 7)} : Q τότε θα μπορούσαμε να επιλέξουμε από το να εκτελεστή είτε το (α, 7). P είτε το {(r_1, 3), (r_2, 7)} : Q. Χρησιμοποιώντας τον πρώτο κανόνα το ChoiceL θα παίρναμε να εκτελεστεί το αριστερό μέρος δηλαδή το (α, 7). P. Χρησιμοποιώντας τον δεύτερο κανόνα το ChoiceR θα παίρναμε να εκτελεστεί το δεξιό μέρος δηλαδή το {(r_1, 3), (r_2, 7)} : Q. Παρατηρούμε ότι για το πρώτο που έχουμε εκτέλεση event θα χρησιμοποιούσαμε το δεύτερο κανόνα τον προθεματικών πράξεων δηλαδή το ActI, ενώ για το δεύτερο θα χρησιμοποιούσαμε τον πρώτο κανόνα το ActT.

Για τον τελεστή παράλληλης επεξεργασίας έχουμε τέσσερις κανόνες. Έχουμε ένα κανόνα ο οποίος τρέχει δύο παράλληλες διεργασίες οι οποίες καταναλώνουν πόρους. Οι πόροι για κάθε διεργασία δεν μπορούν να είναι κοινοί οπότε οι διεργασίες μπορούν να τρέχουν παράλληλα οι διεργασίες.

$$\text{ParT}\ \frac{P \xrightarrow{A1} P', Q \xrightarrow{A2} Q'}{P \,||\, Q \xrightarrow{A1 \cup A2} P' \,||\, Q'} \qquad (\rho(A_1) \cap \rho(A_2) = \varnothing)$$

Αφού οι διεργασίες P και Q δεν χρησιμοποιούν κοινούς πόρους αφού $\rho(R_1) \cap \rho(R_2) = \varnothing$, τότε οι διεργασίες P και Q μπορούν να τρέχουν πραγματικά παράλληλα αφού σε κάθε χρονική στιγμή καταναλίσκουν κάποιους πόρους τους και συνεχίζουν.

Οι επόμενοι δύο κανόνες για παράλληλη επεξεργασία είναι για στιγμιαία γεγονότα.

$$\text{ParIL}\ \frac{P \xrightarrow{(a,n)} P'}{P \,||\, Q \xrightarrow{(a,n)} P' \,||\, Q} \qquad \text{ParIR}\ \frac{Q \xrightarrow{(a,n)} Q'}{P \,||\, Q \xrightarrow{(a,n)} P \,||\, Q'}$$

Αφού τα γεγονότα γίνονται στιγμιαία, και εκτελούνται δύο παράλληλες διεργασίες P και Q τότε αν υποθέσουμε ότι δεν έχουμε συγχρονισμό σε αυτές τότε μπορεί είτε να εκτελέσει η πρώτη διεργασία P το γεγονός της και να μετατραπεί σε μια άλλη διεργασία την P', είτε μπορεί να εκτελέσει το γεγονός της η δεύτερη διεργασία Q και να μετατραπεί σε μια άλλη διεργασία την Q'.

Ο τέταρτος κανόνας για παράλληλη εκτέλεση είναι επίσης για εκτέλεση στιγμιαίων γεγονότων με την διαφορά ότι εδώ έχουμε συγχρονισμό πάνω στις δύο παράλληλες διεργασίες που εκτελούνται.

$$\text{ParCom}\ \frac{P \xrightarrow{(a,n)} P', Q \xrightarrow{(\bar{a},m)} Q'}{P \,||\, Q \xrightarrow{(\tau,n+m)} P \,||\, Q}$$

Η διεργασία P εκτελεί το γεγονός (α,n) και πάει στο P', και η διεργασία Q αφού εκτελέσει το event $\bar{a}$ πάει στο Q'. Αφού εκτελούνται και τα δύο γεγονότα ταυτόχρονα τότε έχουμε συγχρονισμό του P και Q και μαζί προχωρούν στο P' και Q'. Εδώ έχουμε τις δύο διεργασίες P και Q να τρέχουν πραγματικά παράλληλα αφού θα εκτελέσουν μαζί τα γεγονότα τους και θα συγχρονιστούν. Η προτεραιότητα του γεγονός τ είναι το άθροισμα των προτεραιοτήτων των γεγονότων των εμπλεκομένων διεργασιών P και Q.
π.χ. έστω ότι έχουμε τις επόμενες διεργασίες

$P = ((\alpha, 3).P_1) + (\{(A_3, 8)\} : P_2)$

$Q = ((\bar{a}, 5).Q_1) + (\{(A_1, 7)\} : Q_2)$

Βάσει των πιο πάνω κανόνων μπορούμε να έχουμε οποιαδήποτε από τις επόμενες μεταβλητές

α) $P \,||\, Q \xrightarrow{(a,3)} P_1 \,||\, Q$

β) $P \,||\, Q \xrightarrow{(\bar{a},5)} P \,||\, Q_1$

γ) $P \,||\, Q \xrightarrow{(\tau,8)} P_1 \,||\, Q_1$

δ) $P \,||\, Q \xrightarrow{\{(A1,7),(A3,8)\}} P_2 \,||\, Q_2$

Για το τελεστή Δ έχουμε πέντε κανόνες. Οι δύο πρώτοι κανόνες μας δίνουν την περίπτωση όπου μια διεργασία P την οποία έχουμε δεσμεύσει με κάποιο πεδίο εκτελέσει μια ενέργεια. Αυτό μπορεί να γίνει με δύο τρόπους. Αν εκτελεστεί κάποιο γεγονός το οποίο δεν καταναλίσκει κάποιο πόρο και άρα ούτε και χρόνο και επιπλέον το γεγονός που εκτελείται δεν συγχρονίζεται με το γεγονός του πεδίου που έχει βάλει όριο πάνω στο P, ή αν εκτελεστεί κάποια πράξη που καταναλίσκει κάποιο πόρο τότε ο χρόνος του πεδίο δεν θα πρέπει να έχει φτάσει στο μηδέν

$$\text{ScopeCT}\ \frac{P \xrightarrow{A} P'}{P\Delta_t^b(Q,R,A) \xrightarrow{R1} P'\Delta_{t-1}^b(Q,R,S)} \qquad t > 0$$

$$\text{ScopeCI}\ \frac{P \xrightarrow{(a,n)} P'}{P\Delta_t^b(Q,R,S) \xrightarrow{(a,n)} P'\Delta_{t-1}^b(Q,R,S)} \qquad \overline{a} \neq b, t > 0$$

Και στους δύο πιο πάνω κανόνες η εκτέλεση δεν περνάει σε καμιά από τις διεργασίες που βρίσκονται στο πεδίο. Ο πρώτος κανόνας καταναλώνει ένα πόρο αφού περάσει μια χρονική στιγμή. Αφού ο χρόνος ο οποίος βρίσκεται στο πεδίο δεν έχει φτάσει στο μηδέν τότε δεν θα εκτελεστεί κάποια διεργασία από το πεδίο. Ο δεύτερος από τους πιο πάνω κανόνες εκτελεί ένα γεγονός το οποίο δεν καταναλώνει πόρους και άρα ούτε και χρόνο, επίσης το αντίθετο γεγονός δεν μπορεί να συγχρονιστεί με το γεγονός του πεδίου και άρα δεν μπορεί να συγχρονιστεί και να περάσει η εκτέλεση σε κάποια διεργασία του πεδίου

Ο τρίτος κανόνας δείχνει την εκτέλεση της διεργασίας όταν ταιριάζει με κάποιο γεγονός με όνομα αυτό που έχει διηγηθεί στο περιορισμό Δ. Με αυτό τον τρόπο η διεργασία P δηλώνει τερματισμό και η ροή προχωρεί στην διεργασία Q

$$\text{ScopeE}\ \frac{P \xrightarrow{(b,n)} P'}{P\Delta_t^b(Q,R,S) \xrightarrow{(\tau,n)} Q} \qquad t > 0$$

Εδώ αφού η διεργασία P εκτελέσει κάποιο γεγονός (β, n) και πάει στο B, το γεγονός (β, n) συγχρονίζεται με το γεγονός του πεδίου και η εκτέλεση πάει στο Γ.

Ο επόμενος κανόνας δείχνει την περίπτωση όπου το πεδίο έχει κάνει timeout και η εκτέλεση πάει στην δεύτερη διεργασία του πεδίου.

$$\text{ScopeT}\ \frac{R \xrightarrow{(a,n)} R'}{P\Delta_t^b(Q,R,S) \xrightarrow{(a,n)} R'} \qquad t = 0$$

Εδώ καθώς το P εκτελείται και καταναλίσκει πόρους και άρα και χρόνο, ο χρόνος που έχει στην διάθεση του το πεδίο έχει φτάσει στο μηδέν και αφού η διεργασία R πάει στη R', το πεδίο πάει στην διεργασία R'.

Τέλος ο πέμπτος κανόνας είναι ο κανόνας κατά τον οποίο η εκτέλεση περνάει στη τρίτη διεργασία του πεδίου όταν η διεργασία A διακοπεί για κάποιο λόγο.

$$\text{ScopeI}\ \frac{S \xrightarrow{(a,n)} S'}{P\Delta_t^b(Q,R,S) \xrightarrow{(a,n)} S'} \qquad t > 0$$

Σε αυτή την περίπτωση η εκτέλεση του πεδίου περνάει στο S', αφού το S εκτελώντας (α, n) πάνε στο S' και το πεδίο πάει στο S άρα το P πάει και αυτό στο S'.
π.χ. έχουμε μια διεργασία P η οποία έχει προθεσμία εκτέλεσης για 100 χρονικές στιγμές. Εάν κατά τον χρόνο αυτό εκτελεστεί από το P ένα γεγονός $\bar{a}$ τότε η εκτέλεση θα περάσει στη πρώτη διεργασία του πεδίου. Εάν περάσουν οι 100 χρονικές στιγμές τότε η εκτέλεση θα περάσει στην δεύτερη διεργασία του πεδίου. Αν κατά την διάρκεια της εκτέλεσης το P διακοπεί με εκτέλεση του γεγονότος c, τότε η εκτέλεση περνάει στην τρίτη διεργασία του πεδίου.

P Δ^a_{100} (Q, R, (c, 3).S)

Αν εκτελεστή το γεγονός $\bar{a}$ τότε βάση του κανόνα ScopeE η εκτέλεση θα πάει στο Q.

Αν ο χρόνος των 100 χρονικών στιγμών περάσει τότε βάσει του κανόνα ScopeT τότε η ροή ελέγχου θα μεταφερθεί στο R'.

Αν το P διακοπεί εκτελώντας το γεγονός c τότε βάσει του κανόνα ScopeI η ροή ελέγχου θα περάσει στο S.

Στην συνέχεια θα δούμε τους κανόνες για τον περιορισμό. Ο περιορισμός έχει δύο κανόνες, το περιορισμό για τα γεγονότα και τον περιορισμό και τις χρονικές ενέργειες.

$$\text{ResI}\ \frac{P \xrightarrow{(\alpha,n)} P'}{P \setminus \mathrm{E} \xrightarrow{(\alpha,n)} P' \setminus \mathrm{E}} \qquad a, \bar{a} \notin E$$

$$\text{ResT}\ \frac{P \xrightarrow{A} P'}{P \setminus E \xrightarrow{A} P' \setminus E}$$

Ο πρώτος κανόνας δείχνει την εκτέλεση ενός γεγονότος στο οποίο δεν επιτρέπεται να εκτελέσει τα γεγονότα τα οποία βρίσκονται στο σύνολο Ε. Αφού το α και $\bar{a}$ δεν βρίσκονται στο σύνολο Ε μπορούν να εκτελεστούν και έτσι η διεργασία P πάει στην διεργασία P'.

Ο δεύτερος κανόνας δείχνει τον περιορισμό όταν έχουμε εκτέλεση κάποιας πράξης η οποία χρειάζεται χρόνο να εκτελεστεί. Σε αυτή την περίπτωση ο περιορισμός δεν επηρεάζει και η πράξη εκτελείται κανονικά.

π.χ. Ο περιορισμός χρησιμοποιείται πολύ όταν θέλουμε να πετύχουμε συγχρονισμό μεταξύ δύο διεργασιών. Εάν έχουμε

P = (α,1). P'

Q = ($\bar{a}$,2). Q'

τότε έχουμε

$$(P \parallel Q)\setminus\{\alpha\} \xrightarrow{(\tau,1+2)} P' \parallel Q'\setminus\{\alpha\}$$

Αν ο περιορισμός δεν υπήρχε θα μπορούσαμε να διαλέξουμε να εκτελεστεί ή το γεγονός α ή το γεγονός $\bar{a}$. Με τον περιορισμό όμως εξαναγκάζουμε τον συγχρονισμό.

Στην περίπτωση που έχουμε

$P = \{(A_1,1)\}: P'$

$Q = \{(A_2,2)\}: Q'$

τότε έχουμε

$$(P \parallel Q)\setminus\{\alpha\} \xrightarrow{\{(A1,1),(A2,2)\}} (P' \parallel Q')\setminus\{\alpha\}$$

Εδώ που έχουμε πράξεις που θέλουν χρόνο να εκτελεστούν τότε ο περιορισμός δεν επηρεάζει γιατί για να καταναλωθεί ένας πόρος χρειάζεται μια μονάδα χρόνου. Αφού οι διεργασίες P και Q χρησιμοποιούν από ένα πόρο ο οποίος δεν είναι κοινός τότε και οι δύο διεργασίες μετά από μια χρονική στιγμή προχωρούν μαζί στην επόμενη κατάσταση.

2.3 Γλώσσα προγραμματισμού Java και Java 3D API

2.3.1 Εισαγωγή στην Java και Java 3D API

Η γλώσσα προγραμματισμού Java είναι μια γλώσσα αντικειμενοστρεφούς προγραμματισμού η οποία μπορεί να τρέχει σε όλες τις πλατφόρμες αξιόπιστα και χωρίς προβλήματα. Ο τρόπος που το επιτυγχάνει αυτό είναι με την χρήση διερμηνέα. Ο διερμηνέας χρησιμοποιείται σαν ενδιάμεσο σύστημα το οποίο βρίσκεται μεταξύ του προγράμματος της Javas και του λειτουργικού συστήματος στο οποίο τρέχει. Με αυτό τον τρόπο ένα πρόγραμμα το οποίο έχει γραφτεί στην Java μπορεί να τρέξει σε οποιοιδήποτε λειτουργικό σύστημα το οποίο έχει τον διερμηνέα της Javas εγκατεστημένο σε αυτό. Επιπλέον ο διερμηνέας επιτρέπει την ασφαλή εκτέλεση ενός προγράμματος στο λειτουργικό σύστημα στο οποίο τρέχει, αφού μπορεί να σταματήσει κακόβουλα προγράμματα γραμμένα στην Java.

Η Java 3D API αποτελεί επέκταση της απλής Javas. Η Java 3D API χρησιμοποιείται για να πετύχουμε τρισδιάστατα γραφικά. Με την χρήση των βιβλιοθηκών openGL και DirectX, οι οποίες χρησιμοποιούνται για ανάπτυξη τρισδιάστατων γραφικών, η Java 3D API μας δίνει την δυνατότητα να αναπτύξουμε εύκολα και γρήγορα εφαρμογές που κάνουν την χρήση τρισδιάστατων γραφικών.

Με την βοήθεια της Java και Java 3D API μπορούν να αναπτυχθούν προγράμματα τα οποία κάνουν χρήση τρισδιάστατων γραφικών και τρέχουν σε διάφορες πλατφόρμες, ακόμα και μέσα από ένα φυλλομετρητή (browser), όταν είμαστε συνδεμένοι στο διαδίκτυο. Μπορούμε δηλαδή να αναπτύξουμε ακόμα και applets τα οποία τρέχουν τρισδιάστατα γραφικά.

Η Java προσφέρει ότι προσφέρει μια κανονική γλώσσα προγραμματισμού, και επιπλέον μας προσφέρει λειτουργίες οι οποίες δεν υπάρχουν σε πολλές άλλες γλώσσες προγραμματισμού.

Η Java μεταξύ άλλων επιτρέπει τα εξής :

α) Την χρήση πολλών νημάτων σε μια διεργασία όπου μπορώ να έχω σύγχρονη (concurrent) επεξεργασία δεδομένων σε ένα πρόγραμμα

β) Προσφέρει μηχανισμούς για εύκολη εύρεση και ανάκτηση σε περίπτωση λαθών μέσα σε ένα πρόγραμμα προσφέροντας έτσι μεγάλη ευρωστία.

γ) Ένα πρόγραμμα μπορεί να τρέχει σε διάφορες πλατφόρμες οι οποίες έχουν εγκατεστημένο τον διερμηνέα της Javas χωρίς αλλαγή στον κώδικα του προγράμματος για να υποστηριχτεί η νέα πλατφόρμα.

δ) Μπορεί πολύ εύκολα ένα πρόγραμμα (applet) να φορτωθεί σε ένα φυλλομετρητή και να χρησιμοποιηθεί σε μηχανές που επισκέπτονται τον δικτυακό τόπο.

ε) Προσφέρει μια υψηλή αφαιρετικότητα σε προγραμματιστικό επίπεδο η οποία μας δίνει την δυνατότητα να κάνουμε πολλά πολύπλοκα πράγματα πολύ εύκολα και γρήγορα και κυρίως με λιγότερη πιθανότητα λάθους.

2.4 Versa

Το πρόγραμμα Versa χρησιμοποιείται για να μελετηθεί ένα σύστημα το οποίο έχει γραφτεί στην ACSR. Με την βοήθεια υπολογιστών μπορούμε να βρούμε πιθανά αδιέξοδα σε μεγάλα συστήματα πριν να προχωρήσουμε στην υλοποίηση τους γλιτώνοντας έτσι κόπο και χρόνο.

Με την βοήθεια της Versa μπορούμε να επαληθεύσουμε ένα σύστημα το οποίο είναι γραμμένο στην ACSR και να βρούμε πιθανά αδιέξοδα σε αυτό. Τα πλεονεκτήματα εδώ είναι η γρήγορη ανακάλυψη λαθών στις αρχικές φάσεις σχεδιασμού του συστήματος και να γλυτώσουμε χρόνο και κόστος.

Κεφάλαιο 3

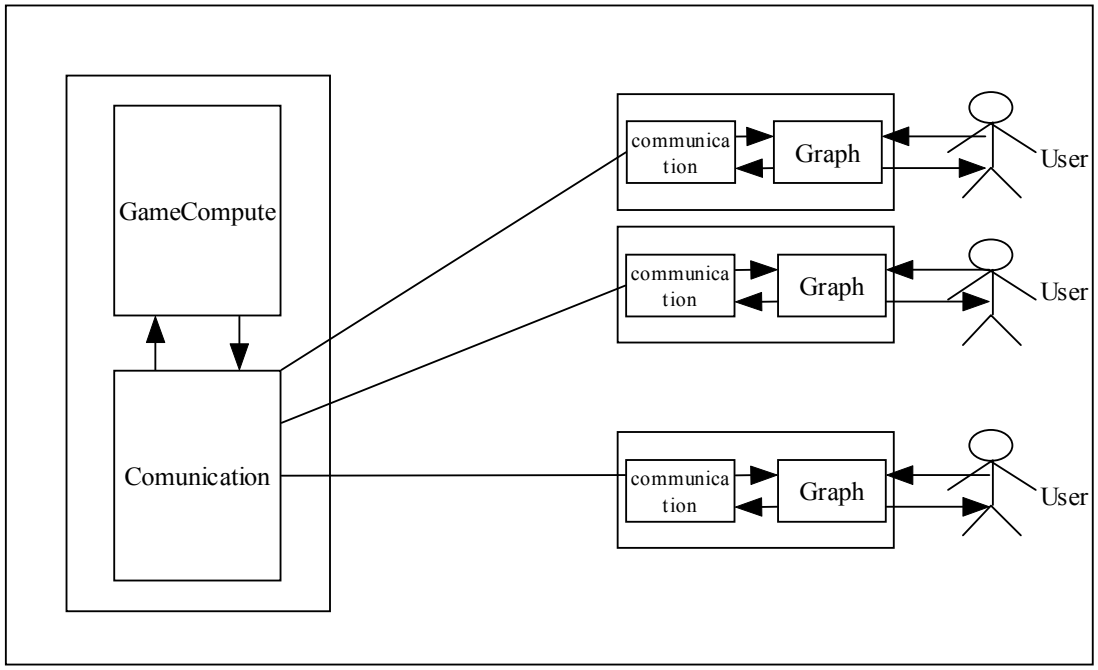
Απαιτήσεις συστήματος

3.1 Περιγραφή κεφαλαίου

Στο κεφάλαιο αυτό θα αναλύσουμε τις απαιτήσεις του συστήματος. Το σύστημα όπως έχουμε πει χωρίζεται σε δύο κύρια κομμάτια Το κομμάτι του συστήματος που κάνει την επικοινωνία μεταξύ των πελατών και του εξυπηρετητή, και το κομμάτι του συστήματος το οποίο είναι υπεύθυνο για το υπολογισμό της νέας κατάστασης και την σχεδίαση στην οθόνη του χρήστη της νέας κατάστασης, καθώς και τη ανταπόκριση του συστήματος στις εντολές του χρήστη.

Πρώτα θα αναλύσουμε τις απαιτήσεις του συστήματος επικοινωνίας και στην συνέχεια του συστήματος του παιγνιδιού. Μετά θα δούμε τις απαιτήσεις της ένωσης των δύο συστημάτων μαζί.

Το πιο κάτω σχήμα δείχνει το σύστημα με τα δύο κομμάτια



Σχήμα 3.1 : Το σύστημα με τον εξυπηρετητή και τους πελάτες

Στο κεφάλαιο αυτό ακολουθούν τρεις ενότητες. Στην ενότητα 3.2 δίνονται οι απαιτήσεις για την επικοινωνία, στην ενότητα 3.3 δίνονται οι απαιτήσεις για το παιγνίδι, και τέλος η 3.4 δίνονται οι απαιτήσεις τις επικοινωνίας και του παιγνιδιού μαζί.

3.2 Απαιτήσεις για την Επικοινωνία

Με τον όρο «επικοινωνία» εννοούμε το κομμάτι της επικοινωνίας ανάμεσα στον εξυπηρετητή καθώς και τους πελάτες. Το κομμάτι επικοινωνίας στον εξυπηρετητή ανταλλάσσει πληροφορίες μεταξύ του κομματιού εξυπηρετητή του κάθε πελάτη. Κάθε φορά ο εξυπηρετητής στέλλει μήνυμα στους πελάτες και οι πελάτες αφού πάρουν το μήνυμα στέλνουν πίσω την κατάσταση τους στον εξυπηρετητή.

Οι απαιτήσεις του συστήματος επικοινωνίας που πρέπει να υλοποιηθεί χωρίζονται σε δύο κατηγορίες. Οι απαιτήσεις που υπάρχουν για τον εξυπηρετητή και οι απαιτήσεις που υπάρχουν για τον πελάτη (χρήστη).

3.2.1 Απαιτήσεις για τον εξυπηρετητή

Ο εξυπηρετητής θα βρίσκεται στο δικτυακό τόπο του παιγνιδιού. Γενικά η δουλειά του θα είναι να συντονίζει τους χρήστες (πελάτες-παίκτες) ώστε το παιγνίδι να είναι δίκαιο και αξιόπιστο.

Ο εξυπηρετητής θα περιμένει να ακούσει από τους n παίκτες. Μόλις έρθει μια αίτηση από κάποιο χρήστη ο εξυπηρετητής, ο οποίος δεν θα εξυπηρετεί άλλους χρήστες εκείνη την στιγμή, θα αποδέχεται τον χρήστη στο παιγνίδι και θα ξεκινάει και το παιγνίδι. Επίσης αμέσως μετά που θα λάβει μήνυμα από το χρήστη για να εισέλθει στο παιγνίδι, η αίτηση του χρήστη γίνετε δεκτή και του στέλνεται μήνυμα πίσω ότι γίνεται αποδεχτεί η αίτηση του.

Στην συνέχεια θα πρέπει να περιμένει να ακούσει από τους υπόλοιπους (n-1 παίκτες) για αίτηση για είσοδο τους στο παιγνίδι. Αφού ο εξυπηρετητής εξυπηρετεί ήδη τον πρώτο χρήστη εκείνη την δεδομένη στιγμή, και άρα το παιγνίδι έχει ήδη αρχίσει, τότε η επόμενη αίτηση που θα έρθει από κάποιον χρήστη για να εισέλθει στο παιγνίδι, θα γίνει αποδεχτή από τον εξυπηρετητή αφού ο εξυπηρετητής δεν έχει φτάσει στον μέγιστο αριθμό χρηστών που μπορεί να δεχτεί. Αφού αποδεχθεί την αίτηση του χρήστη θα του στέλνεται επιβεβαίωση ότι έγινε δεκτή η πρόσκληση του.

Το παιγνίδι θα συνεχίζεται από το σημείο που έχει εισέλθει ο νέος παίκτης, όπως δηλαδή έχει διαμορφωθεί μέχρις στιγμής με τους υφιστάμενους χρήστες (δηλαδή το πρώτο χρήστη). Τώρα ο εξυπηρετητής θα ακούει από του υπόλοιπους (n-2 παίκτες) κάποια αίτηση για να εισέλθουν στο παιγνίδι. Αν κάποιος νέος παίκτης ζητήσει να εισέλθει στο παιγνίδι τότε θα πρέπει να επαναληφθεί η διαδικασία που περιγράφηκε πιο πάνω. Αν έχουν εισέλθει στο παιγνίδι όλοι οι παίκτες τότε ο εξυπηρετητής δεν θα δέχεται άλλες αιτήσεις (request) από άλλους παίκτες για να εισέλθουν στο παιγνίδι. Σε αυτή την φάση ο εξυπηρετητής θα χειρίζεται το παιγνίδι και θα μπορεί να πάρει αιτήσεις από κάποιους παίκτες μόνο για αποχώρηση από το παιγνίδι.

Το μήνυμα που θα αποστέλλει ο εξυπηρετητής στο κάθε παίκτη για να του πει ότι αποδέχεται την αίτηση του για εισαγωγή στο παιγνίδι πρέπει να περιλαμβάνει επιπλέον τις πληροφορίες του παίκτη.

Οι λειτουργίες που θα κάνει ο εξυπηρετητής κατά τη διάρκεια του παιγνιδιού θα είναι να λαμβάνει από κάθε παίκτη που βρίσκεται στο παιγνίδι τις κινήσεις του για μια νέα κατάσταση, και στην συνέχεια να αποστέλλει πίσω σε αυτούς τη νέα κατάσταση του παιγνιδιού όταν όλοι παίκτες παίξουν μια φορά.

Αν κάποιος παίκτης κάνει αίτηση για αποχώρηση, τότε ο εξυπηρετητής θα αποδέχεται την αίτηση του χρήστη για αποχώρηση, και θα τον αφαιρεί από το παιγνίδι θέτοντας το κομμάτι που ήταν υπεύθυνο για την επικοινωνία με τον χρήστη σε μη-ενεργή κατάσταση ώστε να μπορεί να εξυπηρετήσει άλλους χρήστες στο μέλλον. Ο χρήστης αφού στείλει το μήνυμα προς αποχώρηση τότε θα τερματίζει. Το παιγνίδι θα συνεχίζεται με τους υπόλοιπους παίκτες από το σημείο στο οποίο ο παίκτης που αποχώρισε, έκανε την αίτηση του για αποχώρηση. Στην περίπτωση που κάποιοι παίκτες έχουν αποχωρήσει από το παιγνίδι, ο εξυπηρετητής μπορεί να δέχεται αιτήσεις για αποχώρηση και εισαγωγή στο παιγνίδι (αφού υπάρχουν λιγότεροι από n παίκτες), από νέους αλλά και από παλαιότερους παίκτες που έχουν εγκαταλείψει το παιγνίδι. Στην περίπτωση που απομείνει μόνο ένας παίκτης και με την σειρά του και αυτός ζητήσει να αποχωρήσει από το παιγνίδι, ο εξυπηρετητής θα κάνει δεκτή την αίτηση του για αποχώρηση και θα τερματίζει το παιγνίδι. Σε αυτό το σημείο ο εξυπηρετητής δεν θα δέχεται αιτήσεις για αποχώρηση αλλά θα περιμένει αίτηση από κάποιο παίκτη για να δημιουργήσει ένα νέο παιγνίδι.

Σε περίπτωση που κάποιος παίκτης πάψει να επικοινωνεί με το εξυπηρετητή (για κάποιο λόγο), για ορισμένο χρονικό διάστημα τότε ο εξυπηρετητής τον αφαιρεί από το παιγνίδι βάσει του πιο πάνω κανόνα. Δηλαδή ακολουθείται η ίδια διαδικασία που περιγράφεται ποιο πάνω για την αποχώρηση κάποιου παίκτη με την μόνη διαφορά ότι ο παίκτης που αποχωρεί δεν κάνει αίτηση για αποχώρηση αλλά απλά αποβάλλεται γιατί σταμάτησε να επικοινωνεί για κάποιο χρονικό διάστημα με τον εξυπηρετητή.

Αφού ξεκινήσει το παιγνίδι, ο εξυπηρετητής θα παίρνει από κάθε χρήστη (παίκτη) που βρίσκεται στο σύστημα (είτε ξεκίνησε όταν εισήλθε ο πρώτος χρήστης ο οποίος ξεκινάει και το παιγνίδι, ή συνεχίζεται το παιγνίδι από εκεί που εισέρχεται, ή αποχωρεί κάποιος άλλος παίκτης στο σύστημα), τις πιθανές κινήσεις που κάνει. Στην συνέχεια αφού κάνει τους αναγκαίους υπολογισμούς για την επόμενη κατάσταση του παιγνιδιού τότε θα αποστέλλει την πληροφορία αυτή στους παίκτες για να αναβαθμίσουν την κατάσταση του παιγνιδιού τους. Η αναβάθμιση θα πρέπει να γίνεται τουλάχιστον 25 φορές το δευτερόλεπτο για κάθε παίκτη ώστε να μπορεί ο κάθε παίκτης να έχει στην διάθεσή του τουλάχιστον 25 frames το δευτερόλεπτο. Ο αριθμός των frames που θα παράγονται στη μηχανή κάθε παίκτη εξαρτάται από τις δυνατότητες της μηχανής του κάθε χρήστη.

Ο εξυπηρετητής πρέπει μεταξύ άλλων πρέπει να παρέχει α) δικαιοσύνη μεταξύ των παιχτών, β) να είναι ανεκτικός σε λάθη από πλευράς των παιχτών, και γ) να παρέχει ζωτικότητα στο παιγνίδι, να προσφέρει δηλαδή στο παιγνίδι τη δυνατότητα να τρέχει σε ικανοποιητικά επίπεδα.

α) Δικαιοσύνη

Ο κάθε παίχτης θα πρέπει να παίρνει το ίδιο χρόνο που έχει ο κάθε άλλος παίχτης ασχέτως με τη δυνατότητα του συστήματος που διαθέτει ή της επικοινωνίας με το διαδίκτυο. Σε περίπτωση που κάποιος παίκτης δεν χρησιμοποιήσει το χρόνο που του προσφέρεται από τον εξυπηρετητή τότε χάνει την σειρά του και συνεχίζει ο επόμενος. Ο κάθε παίχτης θα παίρνει από το εξυπηρετητή τις ίδιες πληροφορίες που θα παίρνουν και όλοι οι άλλοι. Οι πληροφορίες αυτές είναι εκείνες που καθορίζουν τη ροή του παιγνιδιού.

β) Ανεκτικότητα σε λάθη

Ο εξυπηρετητής πρέπει να είναι ανεκτικός σε λάθη που μπορεί να κάνουν οι παίκτες όπως τη μη αποστολή κάποιων πληροφοριών προς το εξυπηρετητή ή η παραλαβή πληροφοριών που μπορούν να περιέχουν λάθη, ή ακόμα παραλαβή πληροφοριών που μπορεί να φτάσουν καθυστερημένες. Ακόμα θα πρέπει να μπορεί να χειρίζεται τις απρόβλεπτες αποχωρήσεις από παίκτες. Σε όλες αυτές τις περιπτώσεις ο εξυπηρετητής θα πρέπει να τις χειρίζεται ανάλογα και να συνεχίζει το παιγνίδι ώστε να μην παρατηρούνται προβλήματα σε άλλους χρήστες. Ο κάθε χρήστης θα πρέπει να μπορεί να βλέπει την ίδια εικόνα από το εξυπηρετητή όπως και όλοι οι άλλοι.

γ) Ζωτικότητα στο παιγνίδι

Το παιγνίδι στο υπολογιστή του κάθε χρήστη θα πρέπει να παράγει 25 πλαίσια (frames) το δευτερόλεπτο ασχέτως από τη μηχανή που τρέχει. Άρα ο εξυπηρετητής θα πρέπει να στέλνει 25 φορές το δευτερόλεπτο σε κάθε χρήστη την κατάσταση του παιγνιδιού.

3.2.2 Απαιτήσεις για τον πελάτη (χρήστη)

Ο κάθε χρήστης θα βρίσκεται σε κάποιο απομακρυσμένο σύστημα από το οποίο πρέπει να έχει πρόσβαση στο διαδίκτυο. Για να λάβει μέρος στο παιγνίδι θα πρέπει να επισκεφτεί το δικτυακό τόπο του παιγνιδιού και να κάνει αίτηση για να παίξει. Εκεί θα πρέπει να του δοθεί άδεια από τον εξυπηρετητή, ο οποίος δεδομένου ότι δεν έχει συμπληρωθεί ο μέγιστος αριθμός παικτών που επιτρέπεται στο παιγνίδι. Ο παίκτης περιμένει να ακούσει από τον εξυπηρετητή ότι γίνεται δεκτή η αίτηση του. Αφού ακούσει από το εξυπηρετητή ότι έχει γίνει δεκτός στο παιγνίδι τότε ξεκινά το παιγνίδι στη μηχανή του χρήστη (παίκτη) περιμένοντας να ακούσει από τον εξυπηρετητή και να αποστείλει σε αυτόν πίσω πληροφορίες για την κατάσταση του χρήστη.

Ο χρήστης, αφού ξεκινήσει το παιγνίδι, θα αποστέλλει κάθε φορά στο εξυπηρετητή τις κινήσεις που κάνει και θα παίρνει από τον εξυπηρετητή, τουλάχιστον κάθε 1/25 του δευτερολέπτου, τη νέα κατάσταση, ώστε να μπορεί να γίνεται αναβάθμίση στην εικόνα που θα εμφανίζεται στην μηχανή του χρήστη. Στη μηχανή του χρήστη θα σχεδιάζονται στην οθόνη οι νέες πληροφορίες που θα παίρνει από τον εξυπηρετητή. Η λειτουργία της αποστολής και παραλαβής πληροφορίας από τον εξυπηρετητή, και ο σχεδιασμός στην οθόνη του χρήστη της νέας πληροφορίας θα γίνονται παράλληλα (ταυτόχρονα) στη μηχανή του κάθε παίκτη ξεχωριστά. Ο σκοπός αυτός είναι να μπορούν να παραχθούν όσο περισσότερα frames μπορούν στη μηχανή αυτή (τουλάχιστον 25 για να μπορεί το παιγνίδι να είναι υποφερτό).

Μόλις ο παίκτης ξεκινήσει το παιγνίδι θα περιμένει να ακούσει από τον εξυπηρετητή την αρχική για αυτόν κατάσταση του παιγνιδιού. Στη συνέχεια βάσει των πληροφοριών που θα πάρει, θα πάει σε μια αρχική κατάσταση και θα ξεκινήσει η επικοινωνία μεταξύ του και του εξυπηρετητή. Θα αποστέλλει δηλαδή στον εξυπηρετητή τις κινήσεις που κάνει και θα περιμένει να ακούσει την νέα κατάσταση του παιγνιδιού από τον εξυπηρετητή, αφού παίξουν όλοι οι παίκτες που βρίσκονται στο παιγνίδι αυτή την στιγμή.

Ο χρήστης έχει τη δυνατότητα να τερματίσει το παιγνίδι ανά πάσα στιγμή αποστέλλοντας στον εξυπηρετητή μήνυμα (αίτηση) για αποχώρηση από το παιγνίδι. Αφού στείλει το μήνυμα τότε ο παίκτης τερματίζει.

Σε περίπτωση που ο χρήστης δεν παίρνει απόκριση από τον εξυπηρετητή για κάποιο χρονικό διάστημα τότε ο χρήστης θεωρεί ότι ο εξυπηρετητής έπαψε να αποκρίνεται και τερματίζει το παιγνίδι.

Ο πελάτης μεταξύ άλλων πρέπει να έχει τις παρακάτω ιδιότητες α) Δικαιοσύνη, β) Ευρωστία, γ) Συνοχή, δ) Ανταπόκριση, ε) Αποδοτικότητα.

α) Δικαιοσύνη

Να μπορεί να νικήσει το παιγνίδι με την ίδια πιθανότητα με κάποιο άλλο χρήστη ανεξάρτητα από τη μηχανή ή την σύνδεση με το διαδίκτυο που έχει.

β) Ευρωστία

Να μπορεί να συνεχίζει να παίζει το παιγνίδι ανεξάρτητα από την κατάσταση που μπορεί να βρίσκεται κάποιος άλλος παίκτης. Αν έχει τερματίσει κάποιος άλλος τότε το παιγνίδι να συνεχίζεται χωρίς πρόβλημα κ.τ.λ.

γ) Συνοχή

Όλα τα γεγονότα του παιγνιδιού, αποστολή κίνησης, παραλαβής κατάστασης παιγνιδιού, κ.τ.λ. πρέπει να γίνονται στο 1/25 του δευτερολέπτου για κάθε χρήστη ώστε να μπορεί να διατηρηθεί το frame rate του παιγνιδιού. Αφού ο κάθε χρήστης θα πρέπει να είναι συγχρονισμένος με τον εξυπηρετητή αν περάσει ένα δευτερόλεπτο παιγνιδιού στην μηχανή ενός χρήστη θα πρέπει να έχει περάσει ένα δευτερόλεπτο στην μηχανή κάθε χρήστη και ένα δευτερόλεπτο παιγνιδιού στον εξυπηρετητή.

δ) Ανταπόκριση

Όλες οι κινήσεις που κάνει ο χρήστης θα πρέπει να φτάνουν στο εξυπηρετητή εντός ενός frame ώστε να επεξεργάζονται και να αποστέλλονται πίσω στους χρήστες.

ε) Αποδοτικότητα

Η κάθε μηχανή του χρήστη θα πρέπει να μπορεί να κάνει παράλληλα επικοινωνία με το εξυπηρετητή για παραλαβή των καταστάσεων του παιγνιδιού αλλά και αποστολή της νέας κίνησης του παίκτη μαζί με το σχεδιασμό των frames στην οθόνη του υπολογιστή του χρήστη. Αφού αυτές οι πράξεις θα γίνονται ταυτόχρονα θα πρέπει όταν οι πόροι του υπολογιστή δεν χρησιμοποιούνται για επικοινωνία με τον εξυπηρετητή τότε να χρησιμοποιούνται για σχεδιασμό τον frames στην οθόνη.

3.3 Απαιτήσεις για το Παιγνίδι

Με τον όρο «παιγνίδι» εννοούμε το κομμάτι από την πλευρά του εξυπηρετητή το οποίο υπολογίζει την επόμενη κατάσταση του παιγνιδιού, και το κομμάτι από τους πελάτες το οποίο αντιδρά στους χειρισμούς του χρήστη καθώς και σχεδιάζει στην οθόνη του υπολογιστή την νέα κατάσταση όπως λαμβάνεται από τον εξυπηρετητή.

Αυτό το κομμάτι χειρίζεται το παιγνίδι και είναι υπεύθυνο για την σωστή και αξιόπιστη λειτουργία του. Το παιγνίδι μπορεί να είναι οτιδήποτε το οποίο πληροί τις προδιαγραφές του πιο πάνω κομματιού της επικοινωνίας.

Το παιγνίδι που θα υλοποιηθεί ονομάζεται Ghostbusters. Η υλοποίηση του θα γίνει ξεχωριστά από την υλοποίηση του ποιο πάνω συστήματος επικοινωνίας και θα ενσωματωθούν μ' αυτό μετά την υλοποίηση και το δύο μερών.

Οι απαιτήσεις του συστήματος παιγνιδιού που πρέπει να υλοποιηθεί χωρίζονται και εδώ σε δύο κατηγορίες. Οι απαιτήσεις που ζητούνται από τον εξυπηρετητή και οι απαιτήσεις που ζητούνται από τον πελάτη (χρήστη).

3.3.1 Γενικές απαιτήσεις

Το παιγνίδι θα πρέπει να μπορεί να παίζεται από όλες τις μηχανές που έχουν σύνδεση στο διαδίκτυο, ανεξαρτήτως του λειτουργικού συστήματος που τρέχουν και του υλικού που έχουν εγκατεστημένο. Αναπόφευκτα όμως χρειάζεται ο υπολογιστής αυτός να είναι αρκετά γρήγορος ώστε να μπορεί να τρέχει το εν λόγω παιγνίδι.

3.3.2 Απαιτήσεις παιγνιδιού από τον εξυπηρετητή

Το κομμάτι του παιγνιδιού που θα τρέχει στον εξυπηρετητή θα πρέπει να μπορεί να λαμβάνει τις πληροφορίες που εστάλησαν από τους χρήστες, και στη συνέχεια να υπολογίζει την νέα κατάσταση του παιγνιδιού και να τη δίνει πίσω για αποστολή στους χρήστες.

Κάθε φορά θα παίρνει ένα σύνολο από πληροφορίες από τον κάθε χρήστη. Αφού γνωρίζει το τρόπο με τον οποίο ο κάθε πελάτης κωδικοποιεί τις πληροφορίες που στέλνει μπορεί να τις αποκωδικοποιεί και να τις επεξεργάζεται. Στην συνέχεια θα κωδικοποιεί τα αποτελέσματα σε μια μορφή που καταλαβαίνει ο κάθε πελάτης και θα τα δίνει για αποστολή προς αυτούς.

Κατά την επεξεργασία των δεδομένων ο εξυπηρετητής θα κοιτάζει το id κάθε πελάτη, τους βαθμούς που έχει μέχρι στιγμής και πόσα φαντάσματα έχει κτυπήσει. Αν έχει κτυπήσει π.χ. δύο τότε μετά τους βαθμούς του θα λάβει τον αριθμό δύο και το id των δύο φαντασμάτων που έχει κτυπήσει. Αν δεν έχει κτυπήσει κανένα τότε θα στείλει μαζί με τους βαθμούς του τον αριθμό μηδέν.

Στη συνέχεια αφού αποκωδικοποιήσει την πληροφορία θα προσθέτει σε κάθε πελάτη που κτύπησε κάποιο/α τα φάντασμα/τα την ανάλογη βαθμολογία και θα αρχικοποιεί τα φαντάσματα που κτυπήθηκαν.

Αφού η πιο πάνω διαδικασία επαναληφθεί για όλους τους ενεργούς πελάτες που έστειλαν πληροφορία σε ένα frame τότε ο εξυπηρετητής θα υπολογίζει την νέα τους θέση ανάλογα με την κατεύθυνση που έχουν (δεξιά ή αριστερά), τη ταχύτητά τους και το ύψος τους.

Στην συνέχεια αφού η πιο πάνω πληροφορία κωδικοποιηθεί θα δίνεται πίσω για αποστολή στους πελάτες.

3.3.3 Απαιτήσεις παιγνιδιού από τον πελάτη

Στη μηχανή του κάθε πελάτη το παιγνίδι θα είναι υπεύθυνο για να λαμβάνει τις κινήσεις που κάνει ο κάθε χρήστης και να τις αποστέλλει στον εξυπηρετητή. Επιπλέον θα πρέπει να αποκωδικοποιεί κάθε φορά την πληροφορία που στέλλει ο εξυπηρετητής και να χρησιμοποιεί τις πληροφορίες αυτές ώστε να σχεδιάζει στην οθόνη του χρήστη την επόμενη κατάσταση.

Κάθε φορά που παραλαμβάνει μια κωδικοποιημένη πληροφορία, γνωρίζοντας το πρωτόκολλο το οποίο χρησιμοποιεί ο εξυπηρετητής, θα πρέπει να είναι σε θέση να μπορεί να αποκωδικοποιεί την πληροφορία, αυτή και να την χρησιμοποιεί ώστε να σχεδιάζει στην οθόνη του χρήστη την νέα κατάσταση.

Κατά την επεξεργασία των δεδομένων ο πελάτης επεξεργάζεται αρχικά την πληροφορία των οχτώ φαντασμάτων η οποία περιλαμβάνει την συντεταγμένες x,y,z, την κατεύθυνση του φαντάσματος (αριστερά ή δεξιά) καθώς και την ταχύτητα που κινείται το κάθε φάντασμα. Στην συνέχεια υπάρχουν όλοι οι χρήστες οι οποίοι έστειλαν κάποιο μήνυμα στην προηγούμενο frame στο οποίο περιλαμβάνεται το id του χρήστη αλλά και η βαθμολογία που έχει ο χρήστης μέχρι στιγμής.

Ο πελάτης αφού αποκωδικοποιήσει τη πληροφορία τότε θα σχεδιάζει στην οθόνη του υπολογιστή τις νέες θέσεις των φαντασμάτων όπως αυτές έχουν σταλεί από τον εξυπηρετητή, καθώς και τα ονόματα των χρηστών μαζί με την βαθμολογία που έχουν συγκεντρώσει μέχρι στιγμής.

Επιπλέον θα ακούει από τον χρήστη για τις κινήσεις που θα κάνει. Ο χρήστης θα κινεί το ποντίκι (mouse) στην οθόνη και θα προσπαθεί να κτυπήσει με τον πλήκτρο του ποντικιού ένα ή περισσότερα φαντάσματα. Ο πελάτης θα πρέπει να μπορεί να λαμβάνει αυτή την πληροφορία από τον χρήστη και να την κωδικοποιεί κατάλληλα ώστε όταν σταλεί στον εξυπηρετητή να μπορεί να αποκωδικοποιηθεί.

3.4 Απαιτήσεις Επικοινωνίας και Παιγνιδιού

Μετά την υλοποίηση των δύο μερών, Επικοινωνίας και Παιγνιδιού πρέπει να ενωθούν τα δύο μέρη μαζί. Τα δύο μέρη πρέπει να είναι με τέτοιο τρόπο δημιουργημένα ώστε να μπορούν να ενωθούν εύκολα. Το κομμάτι επικοινωνίας πρέπει να μπορεί να καλεί το κομμάτι του παιγνιδιού που είναι υπεύθυνο, α) για το εξυπηρετητή το κομμάτι που υπολογίζει τη νέα κατάσταση, β) για το πελάτη το κομμάτι που σχεδιάζει στην οθόνη την νέα κατάσταση και ακούει από τον χρήστη τις κινήσεις του.

Το κομμάτι επικοινωνίας με αυτό τον τρόπο θα πρέπει να είναι σε θέση να δεκτεί οποιοδήποτε παιγνίδι το οποίο πληροί τις ίδιες προδιαγραφές με το πιο πάνω παιγνίδι. Επιπλέον το παιγνίδι πρέπει να μπορεί να παίζει σε οποιαδήποτε μηχανή χωρίς τη χρήση της κατανεμημένης χρήσης, τοποθετώντας δηλαδή το κομμάτι του παιγνιδιού που υπολογίζει στην νέα κατάσταση από τον εξυπηρετητή, και το κομμάτι που σχεδιάζει την ένα κατάσταση του παιγνιδιού από τον κάθε πελάτη, σε ένα πρόγραμμα μαζί. Δηλαδή το κομμάτι του υπολογισμού της νέας κατάστασης θα «επικοινωνεί» κατευθείαν με το κομμάτι που κάνει τον σχεδιασμό.

Αν ισχύουν τα ποιο πάνω μπορούμε εύκολα να χωρίσουμε το παιγνίδι τοποθετώντας έτσι το κομμάτι που υπολογίζει την επόμενη κατάσταση σε ένα πρόγραμμα που τρέχει σε μια μηχανή κάπου στο διαδίκτυο (τον εξυπηρετητή δηλαδή), και φυσικά το κομμάτι που σχεδιάζει το παιγνίδι στην οθόνη και ακούει από τον

χρήστη τις κινήσεις του σε κάποια άλλη μηχανή κάπου αλλού στο διαδίκτυο (δηλαδή ο πελάτης). Με αυτό τον τρόπο τα δύο μέρη θα συνενωθούν μαζί.

Κεφάλαιο 4

Προδιαγραφές Συστήματος

4.1 Περιγραφή κεφαλαίου

Στο κεφάλαιο αυτό έχουμε τις προδιαγραφές τους συστήματος που αναλύουμε. Οι προδιαγραφές του συστήματος θα γίνουν αρχικά σε φυσική γλώσσα και θα περιέχουν τις απαιτήσεις του συστήματος επικοινωνίας, του συστήματος παιγνιδιού και στη συνέχεια του συστήματος που τα ενώνει.

Για τη μελέτη του συστήματος επικοινωνίας γίνεται επιπλέον ανάλυση στις προδιαγραφές με τη χρήση της άλγεβρας διεργασιών ACSR (**A**lgebra of **C**ommunicating **S**hared **R**esources). Οι προδιαγραφές του συστήματος επικοινωνίας θα αναλυθούν στις δύο κατηγορίες που αναφέρονται πιο πάνω, εκείνες του εξυπηρετητή και εκείνες του πελάτη. Για κάθε μια ξεχωριστά θα μελετήσουμε τα επιμέρους κομμάτια τους αφού τα περιγράψουμε με την χρήση της ACSR. Μετά θα μελετήσουμε

τις προδιαγραφές σε ACSR ολόκληρου του συστήματος μελετώντας τον τρόπο που επικοινωνούν οι δύο κατηγορίες του συστήματος επικοινωνίας μεταξύ τους.

4.2 Προδιαγραφές Συστήματος Επικοινωνίας

Στη μηχανή που τρέχει ο κάθε πελάτης, τρέχει ένα κομμάτι (thread) που είναι υπεύθυνο για την επικοινωνία με τον εξυπηρετητή καθώς και ένα κομμάτι (thread) που είναι υπεύθυνο αφού πάρει τις πληροφορίες από τον χρήστη να καλέσει το κομμάτι που είναι υπεύθυνο για τον σχεδιασμό στην οθόνη του χρήστη το παιγνίδι αλλά και την αλληλεπίδραση του χρήστη με το παιγνίδι .

Ο εξυπηρετητής τρέχει ένα κομμάτι το οποίο είναι υπεύθυνο να επικοινωνεί με τον κάθε πελάτη, και ένα κομμάτι το οποίο είναι υπεύθυνο αφού πάρει τις πληροφορίες από όλους τους ενεργούς πελάτες καλεί το κομμάτι που είναι υπεύθυνο να υπολογίζει την νέα κατάσταση.

Το κάθε κομμάτι που επικοινωνεί με τους πελάτες αποτελείται από n κομμάτια (ένα για κάθε πελάτη), τα οποία ανταλλάσσουν πληροφορίες με τον κάθε πελάτη. Τα κομμάτια αυτά είναι αρχικά ανενεργά και ενεργοποιούνται όταν ένας νέος πελάτης ζητήσει να εισέλθει στο σύστημα. Απενεργοποιούνται όταν ο πελάτης με τον οποίο επικοινωνούν τερματίσει για οποιοδήποτε λόγο. Το κάθε κομμάτι στέλλει την νέα κατάσταση στον πελάτη με τον οποίο επικοινωνεί και στην συνέχεια ακούει πάνω στο κανάλι επικοινωνίας για την νέα κατάσταση του πελάτη. Το κάθε κομμάτι περιμένει από τον πελάτη για την νέα κατάσταση του για κάποιο προκαθορισμένο χρόνο ώστε αν ένας από τους πελάτες είναι αργός, το ίδιο το παιγνίδι να μην πέσει κάτω από το κατώτατο επίπεδο απόδοσης. Το κατώτατο επίπεδο απόδοσης που χρειάζεται είναι 25 frames(καταστάσεις) το δευτερόλεπτο. Αν η πληροφορία αυτή έρθει μέσα στο προκαθορισμένο χρόνο τότε η πληροφορία περνάει στο κομμάτι που είναι υπεύθυνο να υπολογίσει την νέα κατάσταση, αλλιώς η επόμενη κατάσταση θα υπολογιστεί χωρίς την συμβολή του πελάτη που άργησε να στείλει τις πληροφορίες.

Όταν όλοι οι ενεργοί πελάτες είτε τελειώσουν μια επικοινωνία είτε δεν προλάβουν στον προκαθορισμένο χρόνο, τότε καλείται το κομμάτι για υπολογισμό της νέας κατάστασης.

4.2.1 Προδιαγραφές συστήματος επικοινωνίας εξυπηρετητή

Ο εξυπηρετητής τρέχει στο δικτυακό τόπο της εφαρμογής μας και αρχικά βρίσκεται σε μια κατάσταση όπου μπορεί να δεχτεί μέχρι n πελάτες. Σε αυτή την κατάσταση ο εξυπηρετητής δεν κάνει τίποτε απλά περιμένει να ακούσει από κάποιο πελάτη αίτημα για να εισέλθει στο παιγνίδι. Το παιγνίδι δηλαδή βρίσκεται σε κατάσταση αναμονής και κανένας υπολογισμός δεν γίνεται στο εξυπηρετητή.

Αν πάρει μήνυμα από κάποιο πελάτη για να εισέλθει στο παιγνίδι ενώ βρίσκεται σε κατάσταση αναμονής τότε παραχωρεί στο πελάτη το αίτημα του για να εισέλθει στο παιγνίδι αποστέλλοντας του μήνυμα αποδοχής. Αφού προσθέσει το πελάτη στο παιγνίδι, τότε αφού αυτός είναι ο πρώτος πελάτης στο παιγνίδι ο εξυπηρετητής ξεκινάει το παιγνίδι. Σε αυτή την κατάσταση ο εξυπηρετητής ανταλλάσσει μηνύματα με τον πελάτη σε κάθε frame. Ο πελάτης αφού πάρει τις κινήσεις του χρήστη τότε τις κωδικοποιεί βάσει κάποιου γνωστού πρωτοκόλλου που γνωρίζει με τον εξυπηρετητή, και τις στέλλει σε αυτόν. Ο εξυπηρετητής αφού λάβει τις πληροφορίες υπολογίζει την νέα κατάσταση του παιγνιδιού και αφού τις κωδικοποιήσει βάση του γνωστού πρωτοκόλλου τις στέλλει πίσω στο πελάτη. Αυτό συνεχίζεται για να παραχθεί ένα frame στην μηχανή του πελάτη. Για να έχουμε ομαλό παιγνίδι η ποιο πάνω διαδικασία πρέπει να γίνεται τουλάχιστον 25 φορές το δευτερόλεπτο

Αν σε κάποια χρονική στιγμή ένας άλλος πελάτης ζητήσει να εισέλθει στο σύστημα τότε ο εξυπηρετητής αφού μπορεί να το αποδεκτεί (λόγω του ότι μπορεί να δεκτεί ακόμα n-1 πελάτες), ο πελάτης παίρνει μήνυμα αποδοχής και εισέρχεται στο παιγνίδι. Ο πελάτης προστίθεται στο παιγνίδι και αρχίζει η ανταλλαγή μηνυμάτων με τον εξυπηρετητή. Ο εξυπηρετητής τότε θα ακούει και από τους δύο πελάτες κάθε φορά για τις κινήσεις που παίρνουν από τους χρήστες τους. Αφού πάρει τις κινήσεις τους τότε υπολογίζει την νέα κατάσταση βασισμένος στις κινήσεις και των δύο παικτών. Στην συνέχεια στέλλει την νέα κατάσταση πίσω στους πελάτες. Το κατώτατο όριο ανταλλαγής πληροφοριών είναι όπως είπαμε πιο πάνω 25 frames το δευτερόλεπτο.

Για να πετύχουμε αυτό το κατώτατο όριο ο εξυπηρετητής περιμένει να ακούσει από τους πελάτες για κάποιο προκαθορισμένο χρονικό διάστημα. Αν κάποιος πελάτης δεν προλάβει να στείλει τις κινήσεις του χρήστη σε αυτό το χρόνο τότε ο εξυπηρετητής προχωρεί να υπολογίσει την επόμενη κατάσταση χωρίς να λαμβάνει υπόψη του τους πελάτες που άργησαν να στείλουν τις πληροφορίες τους.

Για του υπόλοιπους n-2 πελάτες μπορούν να εισέλθουν στο σύστημα με τον ίδιο τρόπο με τον προηγούμενο πελάτη.

Όταν το σύστημα έχει δεκτεί όλους του πελάτες τότε δεν μπορεί να δεκτή άλλους χρήστες και το μόνο που μπορεί να κάνει είναι επιτρέψει την αποχώρηση κάποιων.

Αν ένας πελάτης επιθυμεί να φύγει από το σύστημα τότε στέλνει μήνυμα για να αποχωρήσει από το σύστημα. Αν το σύστημα εξυπηρετεί από ένα πελάτη εως τον μέγιστο αριθμό από πελάτες που μπορεί να εξυπηρετήσει τότε ο πελάτης που κάνει αίτηση για αποχώρηση αποχωρεί από το σύστημα και ο εξυπηρετητής συνεχίζει με τους υπόλοιπους πελάτες. Αν ο εξυπηρετητής εξυπηρετεί μόνο ένα πελάτη και αυτός ζητήσει να αποχωρίσει τότε ο πελάτης αποχωρεί και ο εξυπηρετητής πάει σε κατάσταση αναμονής και περιμένει να ακούσει αίτηση από κάποιους πελάτες να εισέλθουν στο σύστημα.

Αν κάποιος πελάτης έχει σταματήσει για κάποιο χρονικό διάστημα να επικοινωνεί με τον εξυπηρετητή τότε ο εξυπηρετητής τερματίζει τον πελάτη χωρίς να χρειάζεται η συγκατάθεση του πελάτη. Για την αποχώρηση ισχύουν οι ίδιοι κανόνες που ισχύουν και για την κανονική αποχώρηση ενός πελάτη.

4.2.2 Προδιαγραφές συστήματος επικοινωνίας πελάτη

Ο πελάτης βρίσκεται επίσης στον δικτυακό τόπο της επιλογής μας. Κάθε φορά που ένας χρήστης επιθυμεί να παίξει το παιγνίδι τότε δημιουργεί ένα στιγμιότυπο του πελάτη στην μηχανή του. Ο πελάτης ζητά από τον χρήστη να δώσει ένα όνομα και στην συνέχεια ο πελάτης αφού πάρει την πληροφορία αυτή την στέλλει στον εξυπηρετητή μαζί με το αίτημα για να εισέλθει στο παιγνίδι. Αν του παραχωρηθεί το αίτημα τότε ο πελάτης προχωρεί στην επικοινωνία μεταξύ του πελάτη και του εξυπηρετητή. Αν όχι τότε ο πελάτης απορρίπτεται και δεν εισέρχεται στο παιγνίδι.

Κατά την επικοινωνία ο πελάτης ακούει από το χρήστη τις κινήσεις του. Στη συνέχεια τις κωδικοποιεί βάση κάποιου γνωστού πρωτοκόλλου με τον εξυπηρετητή, και τις αποστέλλει σε αυτόν. Αφού πάρει πίσω τη νέα κατάσταση από τον εξυπηρετητή τότε την αποκωδικοποιεί και τη χρησιμοποιεί για να σχεδιάσει στην οθόνη του χρήστη τη νέα κατάσταση. Αν ο εξυπηρετητής σταματήσει για κάποιο χρονικό διάστημα να επικοινωνεί με τον πελάτη τότε ο πελάτης θεωρεί ότι ο εξυπηρετητής σταμάτησε να δουλεύει και τότε τερματίζει το παιγνίδι.

4.3 Προδιαγραφές Συστήματος Παιγνιδιού

Εδώ ο κάθε πελάτης τρέχει ένα κομμάτι παιγνιδιού το οποίο ουσιαστικά αποτελεί το παιγνίδι μας. Αυτό το κομμάτι είναι υπεύθυνο για την αλληλεπίδραση μεταξύ χρήστη και πελάτη αλλά και για τον σχεδιασμό στην οθόνη του χρήστη της νέας κατάστασης την οποία έστειλε ο εξυπηρετητής.

Το παιγνίδι που αποφασίσαμε να υλοποιήσουμε είναι το Ghostbusters. Σε αυτό το παιγνίδι ζητείται από το κάθε χρήστη που έρχεται στο σύστημα να δώσει ένα όνομα και στη συνέχεια αφού γίνει δεκτός από τον εξυπηρετητή, ο πελάτης ενεργοποιείται και στην οθόνη του χρήστη εμφανίζεται η κατάσταση του παιγνιδιού η οποία είναι διαμορφωμένη ως εκείνη την στιγμή.

Στο παιγνίδι υπάρχουν οχτώ φαντάσματα (εχθροί), οι οποίοι κινούνται κατά μήκος της οθόνης με διαφορετική ταχύτητα και σε διαφορετικό ύψος. Ο χρήστης καλείται να κτυπήσει κάποιο φάντασμα με την κίνηση και την χρήση ενός κουμπιού του ποντικιού. Αν ο χρήστης επιτυχώς κτυπήσει ένα φάντασμα (εχθρό), τότε πιστώνεται με πέντε μονάδες. Κάθε φορά που ένας χρήστης κτυπήσει ένα φάντασμα τότε το φάντασμα επαναδημιουργήται. Το όνομα του κάθε χρήστη εμφανίζεται στην κάτω αριστερά θέση τις οθόνης μαζί με τους βαθμούς του.

Από την πλευρά του εξυπηρετητή, τρέχεται ένα κομμάτι παιγνιδιού το οποίο είναι υπεύθυνο να υπολογίζει την νέα κατάσταση βάσει των πληροφοριών που λαμβάνει και στέλνει πίσω στους πελάτες. Ο εξυπηρετητής κοιτάζει κατά πόσο ένας πελάτης κτύπησε ένα φάντασμα το οποίο δεν έχει κτυπηθεί από κάποιο άλλο. Αν ναι τότε πιστώνει το χρήστη με πέντε μονάδες. Στην συνέχεια ξαναρχικοποιεί το φάντασμα. Για όλα τα υπόλοιπα φαντάσματα ανάλογα με το που κινούνται (δεξιά ή αριστερά), ανάλογα με την ταχύτητα τους και ανάλογα με το ύψος που βρίσκονται υπολογίζει την νέα θέση τους. Στην συνέχεια δίνει πίσω τις πληροφορίες στο κομμάτι που είναι υπεύθυνο για την επικοινωνία ώστε να σταλούν οι πληροφορίες πίσω στους πελάτες.

4.3.1 Προδιαγραφές συστήματος παιγνιδιού εξυπηρετητή

Στον εξυπηρετητή το κομμάτι για το παιγνίδι χρειάζεται για να υπολογίζει την νέα κατάσταση του παιγνιδιού. Το κομμάτι αυτό καλείται κάθε φορά που χρειάζεται να υπολογιστεί η νέα κατάσταση του παιγνιδιού.

Αφού το κομμάτι επικοινωνίας του εξυπηρετητή τελειώσει μια επικοινωνία με όλους τους ενεργούς χρήστες, τότε καλείται το κομμάτι αυτό για να υπολογίσει την νέα κατάσταση. Παίρνοντας τις πληροφορίες που έστειλαν οι πελάτες από το κομμάτι επικοινωνίας, αποκωδικοποιεί τις πληροφορίες αυτές βάση κάποιου κοινού πρωτοκόλλου με τον πελάτη, και χρησιμοποιώντας αυτές τις πληροφορίες υπολογίζει την νέα κατάσταση. Στην συνέχεια καλείται το κομμάτι επικοινωνίας το οποίο περνώντας τις πληροφορίες για την νέα κατάσταση τις αποστέλλει στους πελάτες.

4.3.2 Προδιαγραφές συστήματος παιγνιδιού πελάτη

Αυτό το κομμάτι του παιγνίδι καλείται κάθε φορά που ζητείται να σχεδιαστεί στην οθόνη του υπολογιστή του χρήστη η νέα κατάσταση του παιγνιδιού, αλλά και να ακούσει από τον χρήστη τις κινήσεις του.

Αφού στο πελάτη τελειώσει μια επικοινωνία με τον εξυπηρετητή, τότε καλείται το κομμάτι παιγνίδι στο οποίο περνιούνται οι πληροφορίες που ήρθαν από τον εξυπηρετητή. Με τη σειρά του το κομμάτι παιγνίδι αποκωδικοποιεί τις πληροφορίες αυτές βάσει του γνωστού πρωτοκόλλου που έχει με τον εξυπηρετητή, και χρησιμοποιεί τις πληροφορίες αυτές για να σχεδιάσει στην οθόνη του υπολογιστή την νέα κατάσταση. Ταυτόχρονα ακούει από το χρήστη τις κινήσεις του και ακολούθως τις κωδικοποιεί βάσει του πρωτοκόλλου. Στη συνέχεια περνά τις πληροφορίες προς το κομμάτι επικοινωνίας για να μεταφερθούν προς τον εξυπηρετητή.

4.4 Προδιαγραφές ACSR

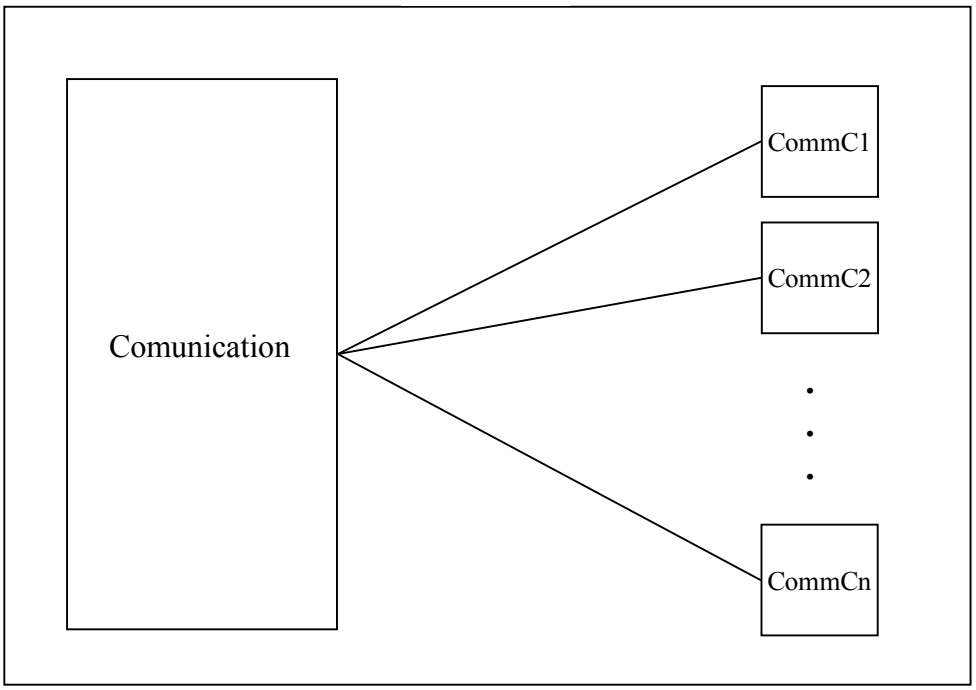
Παρακάτω δίδεται η περιγραφή του συστήματος επικοινωνίας στην άλγεβρα διεργασιών ACSR. Και εδώ θα ακολουθήσουμε μια από πάνω προς τα κάτω προσέγγιση ξεκινώντας με το σύστημα και μπαίνοντας σιγά σιγά στην συνέχεια στα επιμέρους κομμάτια του.

4.4.1 Περιγραφή του συστήματος στην ACSR

Για να γίνει πιο κατανοητή η όλη διαδικασία θα περιγράψουμε αρχικά τον τρόπο που θα λειτουργεί το σύστημα. Η προσέγγιση που θα χρησιμοποιήσουμε θα είναι από πάνω προς τα κάτω. Θα ξεκινήσουμε από το σύστημα και αφού δούμε το τρόπο που επικοινωνούν τα κύρια κομμάτια τότε θα δούμε το κάθε ένα κομμάτι ξεχωριστά. Για κάθε ένα κομμάτι από αυτά θα περιγράψουμε τα δικά του μέρη και θα συνεχίσουμε έτσι για οποιαδήποτε άλλα κομμάτια έχουν αυτά κ.ο.κ.

Το σύστημα αρχικά τρέχει ένα αντίγραφο του εξυπηρετητή το οποίο θα περιμένει να ακούσει από n παίκτες να εισέλθουν στο παιγνίδι. Για κάθε ένα παίκτη που εισέρχεται στο σύστημα ο εξυπηρετητής του δίνει άδεια να παίξει και ο κάθε πελάτης τρέχει παράλληλα με τον εξυπηρετητή στο σύστημα.

Το ποιο κάτω σχήμα δείχνει το σύστημα με τα βασικά μέρη του.



Σχήμα 4.1 : Το σύστημα της επικοινωνίας

Ο κάθε ένας πελάτης (χρήστης) επικοινωνεί με ένα κανάλι με τον εξυπηρετητή το οποίο χρησιμοποιείται για όλους τους χρήστες. Ο εξυπηρετητής δηλαδή δίνει περιοδικά την χρήση αυτού του νοητού καναλιού σε κάθε ένα χρήστη που είναι ενεργός

στο σύστημα. Στην υλοποίηση αυτό το κανάλι μπορεί να χαρακτηριστεί σαν το port μέσω του οποίου γίνετε η επικοινωνία του εξυπηρετητή με τους χρήστες (πελάτες).

Ορίζουμε ως Np = Number of players το μέγιστο αριθμό των παικτών που μπορεί να είναι ενεργοί στο σύστημα ανά πάσα στιγμή.

Ανά πάσα στιγμή στο σύστημα τρέχουν ο εξυπηρετητής και οι Np πελάτες οι οποίοι μπορεί να είναι ενεργοί ή ανενεργοί. Αρχικά όλοι είναι ανενεργοί (δηλαδή όλοι περιμένουν να σε μια κατάσταση όπου δεν κάνουν τίποτε)

System = (Server || AllClients) \{*joinGame, quitGame, gameState, moveInfo*}

AllClients = $\underset{1 \le i \le Np}{||}$ [Client[i]] i = 1, 2, … Np

Στο σύστημα έχουμε Restriction (περιορισμό) πάνω στα κανάλια επικοινωνίας *joinGame, quitGame, gameState, moveInfo*. Τα κανάλια αυτά χρησιμοποιούνται για συγχρονισμό μεταξύ του εξυπηρετητή και των πελατών.

4.4.2 Περιγραφή του εξυπηρετητή στην ACSR

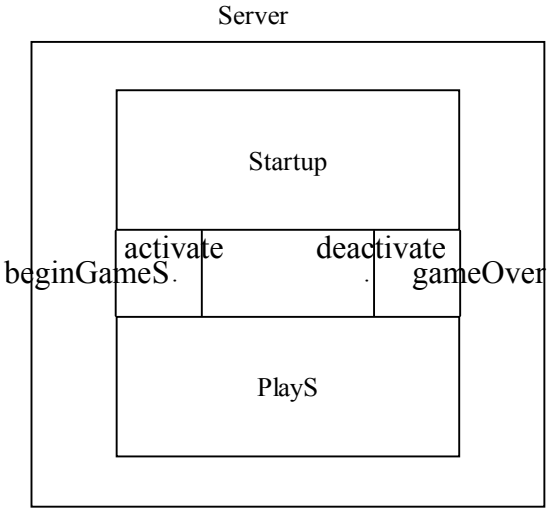
Αρχικά θα δούμε μια περιγραφή του εξυπηρετητή. Ο εξυπηρετητής τρέχει στο δικτυακό μας τόπο και είναι αυτός που θα χειρίζεται το παιγνίδι και θα είναι υπεύθυνος για τη σωστή λειτουργία του παιγνιδιού. Θα είναι αυτός που θα ξεκινά και θα σταματά το παιγνίδι, θα επιτρέπει την είσοδο σε νέους χρήστες αλλά και την αποχώρηση υφιστάμενων χρηστών. Σε κάθε επανάληψη(frame) θα παραλαμβάνει από κάθε χρήστη τις κινήσεις του και αφού υπολογίσει την νέα κατάσταση θα την στέλλει πίσω στους χρήστες.

Ο εξυπηρετητής αποτελείται από δύο μέρη :

Α) Το μέρος που θα χρησιμοποιείται για την εισαγωγή και αποχώρηση των παικτών από το σύστημα το οποίο ονομάζεται Startup

Β) Το μέρος το οποίο είναι υπεύθυνο για τον υπολογισμό της νέας κατάστασης του παιγνιδιού αλλά και τον συγχρονισμό της επικοινωνίας του εξυπηρετητή με τους χρήστες το οποίο ονομάζεται PlayS.

Αυτά τα δύο κομμάτια είναι υπεύθυνα για την δίκαιη, αξιόπιστη, αλλά και σωστή λειτουργία του παιγνιδιού. Επικοινωνούν μεταξύ τους με τέσσερα κανάλια, το beginGameS, το gameOver, το activate και το deactivate, τα οποία χρησιμοποιούνται για να συγχρονιστεί ο εξυπηρετητής να ξεκινήσει ή να τερματίσει το παιγνίδι.
Το παρακάτω σχήμα δείχνει αυτά τα κομμάτια



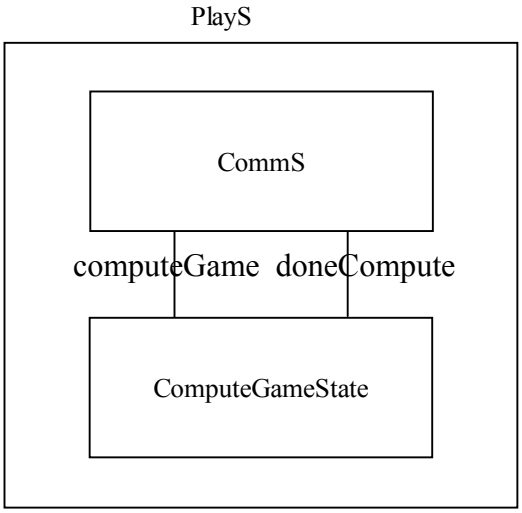
Σχήμα 4.2 : Το σύστημα επικοινωνίας στον εξυπηρετητή

Το PlayS χωρίζετε σε δύο μέρη :

i) Το κομμάτι που είναι υπεύθυνο για την επικοινωνία με τον κάθε χρήστη το CommS, και

ii) Το κομμάτι που είναι υπεύθυνο για τον υπολογισμό της νέας κατάστασης του παιγνιδιού το ComputeGameState

Τα δύο αυτά κομμάτια επικοινωνούν μεταξύ τους μέσω δύο καναλιών του computeGame και του doneCompute το οποία χρησιμοποιούνται για να συγχρονίζονται τα κομμάτια CommS και το ComputeGameState για να σταλούν πληροφορίες και στην συνέχεια να υπολογιστεί η νέα κατάσταση του παιγνιδιού.
Το παρακάτω σχήμα δείχνει το PlayS και τα επιμέρους κομμάτια του.



Σχήμα 4.3 : Το μέρος StartUp του εξυπηρετητή με τα κύρια μέρη του

Στην συνέχεια θα δούμε τις προδιαγραφές σε ACSR του εξυπηρετητή. Θα δούμε τον εξυπηρετητή στο πιο ψηλό επίπεδο και στη συνέχεια θα μελετήσουμε τα επιμέρους κομμάτια του.

Πρώτα έχουμε τον εξυπηρετητή σαν ένα κομμάτι όπου τρέχουν το Startup και το PlayS παράλληλα.

Server = (Startup[Np] || PlayS) \ {*beginGameS, gameOverS, activate, deactivate* }

Εδώ ο εξυπηρετητής τρέχει το κομμάτι Startup[Np] που είναι υπεύθυνο για τους παίκτες που μπαίνουν και βγαίνουν από το σύστημα. Το PlayS είναι αυτό που τρέχει το παιγνίδι. Εδώ έχουμε περιορισμό πάνω στα *beginGameS, gameOverS, activate, deactivate,* όπου είναι τα εσωτερικά κανάλια με τα οποία επικοινωνούν τα δύο κύρια κομμάτια του εξυπηρετητή.

Μετά ακολουθεί η υλοποίηση σε ACSR του κομματιού Startup[Np].

Startup [Np] = rec x ($\emptyset$: x + (*joinGame*, 1). ($\overline{activate}$, 2). StartGame[Np])

StartGame [Np] = ($\overline{beginGameS}$, 2) . Startup [Np-1])

Startup [Np-1] = rec x ($\emptyset$: x + (*joinGame*, 1) .($\overline{activate}$, 2). Startup [Np-2])

+ (*quitGame*, 1).FinishGame[Np-1])

+ (*deactivate*, 1). FinishGame[Np-1]))

Startup [Np-(i+1)] = rec x (∅ : x + (*joinGame*, 1) . ($\overline{activate}$, 2).
Startup [Np – (i+2)])
+ (*quitGame*, 1). Startup [Np-i])
+ (*deactivate*, 1). Startup [Np-i]))

όπου i = 2,…, Np-2

Startup[0] = rec x (∅ : x + (*quitGame*, 1) . Startup [1])
+ (*deactivate*, 1). Startup [1]))

FinishGame[Np-1] = ($\overline{gameOver}$, 2) . Startup[Np])

Εδώ ο εξυπηρετητής περιμένει να ακούσει από κάποιο χρήστη αίτημα για να εισέλθει στο σύστημα. Αν ακούσει από κάποιο και κανένας άλλος δεν βρίσκεται στο σύστημα τότε αφού του επιτρέπει να μπει στο σύστημα ξεκινά το παιγνίδι. Αν είναι ο δεύτερος έως ο Np-2 παίκτης τότε απλά του λέει να μπει στο παιγνίδι, ενώ αν έχουν μπει και οι Np παίκτες στο σύστημα τότε δεν επιτρέπεται να μπουν άλλοι και επιτρέπεται μόνο η αποχώρηση ενός παίκτη. Η αποχώρηση ενός παίκτη γίνεται μόνο αφού ο παίχτης πει στον εξυπηρετητή ότι θέλει να τερματίσει (*quitGame*) ή το thread του αντίστοιχου πελάτη κάνει timeout (*deactivate*)

Στην συνέχεια θα δούμε την υλοποίηση του PlayS. Το PlayS χωρίζεται σε δύο κομμάτια, το CommS και το ComputeGameState.

PlayS = rec x (∅ : x + (*beginGameS*, 2) . Game $\Delta_{\infty}^{gameOver}$ (x, Nil, Nil))
\{*computeGame, doneCompute, addPlayers, removePlayers*}

Εδώ η διεργασία PlayS περιμένει να συγχρονιστεί με το Startup πάνω στο κανάλι *beginGameS* ώστε να πάει στην κατάσταση Game $\Delta_{\infty}^{gameOver}$ (x, Nil, Nil), όπου η κατάσταση αυτή είναι δεσμευμένη να τρέχει έως ακούσει από το startup ότι έχουν φύγει όλοι οι παίκτες και τότε επιστρέφει πίσω στην αρχική κατάσταση x όπου θα περιμένει επ’ άπειρω ή εως ότου έρθει ένας νέος χρήστης στο σύστημα. Το PlayS έχει

τα κανάλια *computeGame, doneCompute, addPlayers, removePlayers,* περιορισμένα γιατί χρησιμοποιούμε αυτά τα κανάλια εσωτερικά στο PlayS.

Game = (CommThreads || ComputeGameState || ActivePlayers[0])

i = 1,2,… Np

CommThreads = $\|_{1 \le i \le Np}$ [InactiveThread[i]] i = 1,2,… Np

Όταν ξεκινήσει το παιγνίδι με τον συγχρονισμό του PlayS και του Startup τότε τρέχουν στον εξυπηρετητή Np διαφορετικά κομμάτια ένα για κάθε πιθανό ενεργό παίκτη που θα ζητήσει να έρθει στο σύστημα. Παράλληλα τρέχει και το ComputeGameState το οποίο είναι υπεύθυνο να υπολογίζει την νέα κατάσταση του παιγνιδιού. Τα Np διαφορετικά κομμάτια μπορεί να παρομοιαστούν με Np διαφορετικά νήματα μέσα σε μια διεργασία.

Σε κάθε επανάληψη θα τρέχουν παράλληλα τα Np διαφορετικά threads και αφού τελειώσουν μια ανταλλαγή μηνύματος με τον πελάτη που εξυπηρετούν, τότε δίνουν τον έλεγχο στην διαδικασία ComputeGameState για να υπολογίσει την νέα κατάσταση του παιγνιδιού. Ο χρόνος που πρέπει να γίνεται η πιο πάνω διαδικασία θα πρέπει να είναι T_{frame} = 1000/25. Οι χρόνοι για την διαδικασία CommS θα πρέπει να είναι T_{comm}, ενώ αντίστοιχα για την διαδικασία ComputeGameState θα είναι $T_{compute}$. Θα έχω δηλαδή $T_{frame} = T_{comm} + T_{compute}$. Το κάθε thread θα παίρνει χρόνο T_{comm} αφού θα τρέχει παράλληλα και αφού τελειώσουν θα τρέχει το ComputeGameState για χρόνο $T_{compute}$.

Το InactiveThread κομμάτι όπως αναφέρθηκε πιο πάνω, τρέχει Np διαφορετικά InactiveThread[i] όπου το 1<=i<=Np, τα οποία αρχικά είναι ανενεργά και περιμένουν να έρθει ο αντίστοιχος Client[i] στο σύστημα για να προχωρήσουν στη φάση όπου γίνεται η επικοινωνία μεταξύ εξυπηρετητή και πελάτη.

Στη συνέχεια θα δούμε την υλοποίηση του μέρους InactiveThread.

InactiveThread[i] = rec y (∅ : y + (*activate*, 2). CommS[i])

CommS[i] = CommS1[i] Δ^{Null}_{10000} (Nil, ($\overline{deactivate}$, 1). y, Nil)

CommS1[i] = ExchangeInfo[i] $\Delta^{contStage}_{Tcomm}$ (NextStage[i], ($\overline{computeGame}$,1).
NextStage[i], Nil)

ExchangeInfo[i] = ($\overline{gameState}$, 2). GetData[i]

GetData[i] = rec w (∅ : w + (*moveInfo*, 1). ($\overline{computeGame}$, 1).

$(\overline{contStage}, 1)$. Nil

NextStage [i] = rec z ($\varnothing$: z + (*doneCompute*, 3) . CommS[i])

i = 1, 2, …, Np

Εδώ έχουμε το κάθε κομμάτι(thread) που θα επικοινωνεί με κάποιο χρήστη. Σε κάθε επικοινωνία θα του στέλλει την κατάσταση του παιγνιδιού όπως έχει διαμορφωθεί, και στην συνέχεια θα περιμένει να ακούσει από αυτόν την νέα του κίνηση. Αν λάβει την νέα κίνηση του εντός του χρόνου T_{comm} τότε προχωρεί και ενημερώνει το ComputeGameState μέσω του καναλιού *computeGame* ότι έχει τελειώσει. Στην περίπτωση που η κίνηση του χρήστη δεν έρθει στο πιο πάνω καθορισμένο χρόνο τότε ο χρήστης χάνει την σειρά του, δηλαδή το ComputeGameState ενημερώνεται να υπολογίσει την νέα κατάσταση του παιγνιδιού χωρίς την κίνηση του χρήστη. Μετά το τέλος κάθε υπολογισμού της επόμενης κατάστασης τότε επαναλαμβάνεται η ίδια διαδικασία.

Μετά βλέπουμε την υλοποίηση του ComputeGameState που είναι υπεύθυνο για συγχρονισμό αλλά και τον υπολογισμό της επόμενης κατάστασης του παιγνιδιού. Κάθε φορά που το ComputeGameState[i] ακούει από κάποιο πελάτη ότι έχει τελειώσει με μια ανταλλαγή πληροφορίας τότε αυξάνει το αριθμό των πελατών που έχουν τελειώσει μέχρι στιγμής ώστε να μπορεί μετά να στείλει τον σωστό αριθμό μηνυμάτων πίσω όταν υπολογίσει την επόμενη κατάσταση. Για να το πετύχει αυτό αυξάνεται ο αριθμός των ActivePlayers[i] κάθε φορά που τελειώνει ένας πελάτης, και ελαττώνεται ο αριθμός των ActivePlayers[i] κάθε φορά που στέλνεται ένα μήνυμα στον κάθε πελάτη μετά τον υπολογισμό της νέας κατάστασης.

ComputeGameState[Np] = rec x($\varnothing$: x + (*computeGame*, 1). ($\overline{addPlayer}$, 2).
ComputeGameState[Np-1])

ComputeGameState[i] = rec x($\varnothing$: x + (*computeGame*, 1). ($\overline{addPlayer}$, 2).
ComputeGameState[i-1]
+ ($\overline{doneCompute}$, 3).(*removePlayers*, 2) .
ComputeGameState[i+1])

όπου i = 1,2 ..Np-1

ComputeGameState[0] = Compute Δ_{Tstate}^{Null} (Nil, ($\overline{doneCompute}$, 3).(*removePlayers*, 2).
ComputeGameState[1], Nil)

ActivePlayers[0] = rec w(∅ : w + (*addPlayers*, 2). ActivePlayers[1])

ActivePlayers[i] = rec w(∅ : w + (*addPlayers*, 2).ActivePlayers[i+1]
+ ($\overline{removePlayer}$, 2).ActivePlayers[i-1])
όπου i = 1,2 ..Np-1

ActivePlayers[Np] = rec w(∅ :w + .($\overline{removePlayer}$, 2).ActivePlayers[Np-1])

Αφού όλοι οι ενεργοί πελάτες τελειώσουν με την επικοινωνία τους τότε στέλνουν μήνυμα στο ComputeGameState[i]. Το ComputeGameState[i] τότε στέλλει μήνυμα στο ActivePlayers[i] να αυξηθεί κατά ένα. Όταν το ComputeGameState[i] φτάσει στο ComputeGameState[0] τότε υπολογίζεται η νέα κατάσταση. Μετά κάθε φορά που το ComputeGameState[i] στέλλει μήνυμα στο κάθε πελάτη να συνεχίσει επιπλέον στέλλει μήνυμα και στο ActivePlayers[i] να ελαττώσει το αριθμό του.

4.4.3 Περιγραφή του πελάτη στην ACSR

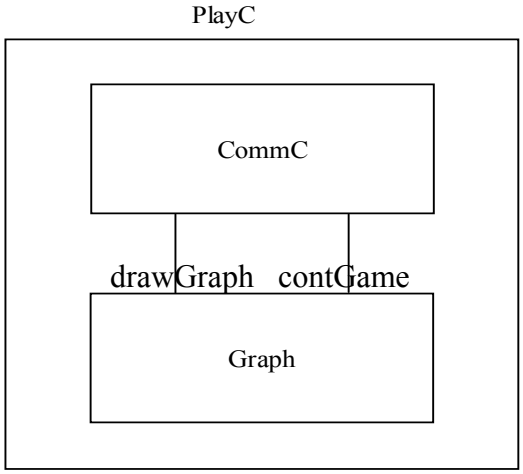
Ο κάθε παίκτης που θα εισέρχεται στο σύστημα θα δημιουργεί ένα αντικείμενο τύπου «πελάτης» το οποίο θα τρέχει τοπικά στην μηχανή του και θα επικοινωνεί μέσω του διαδικτύου με τον εξυπηρετητή. Η μόνη δουλειά που θα κάνει θα είναι να αποστέλλει σε κάθε επανάληψη(frame) στον εξυπηρετητή τις κινήσεις που έκανε ο παίκτης και να παίρνει την νέα κατάσταση του παιγνιδιού πίσω από τον εξυπηρετητή. Επιπλέον πρέπει σε κάθε παραλαβή ενός frame να σχεδιάζει στην οθόνη του χρήστη την νέα κατάσταση του παιγνιδιού.

Ο χρήστης (πελάτης) αποτελείται από το PlayC το οποίο αποτελείται από δύο μέρη

Α) Το κομμάτι που είναι υπεύθυνο για την επικοινωνία με τον εξυπηρετητή, το CommC, και

Β) Το κομμάτι που είναι υπεύθυνο για τον σχεδιασμό στην οθόνη του χρήστη την κατάσταση του παιγνιδιού, το Graph

Αυτά τα κομμάτια επικοινωνούν μέσω δύο καναλιών του drawGraph και του contGame. Μέσω συγχρονισμού σε αυτά τα δύο κανάλια η μηχανή του κάθε χρήστη αφού τελειώσει με την αποστολή παραλαβή πληροφοριών από τον εξυπηρετητή τότε δίνει τον έλεγχο στον κομμάτι Graph για να σχεδιάσει στην οθόνη του υπολογιστή την νέα κατάσταση του παιγνιδιού. Στη συνέχεια ο έλεγχος επιστρέφεται πίσω στο CommC αφού γίνει συγχρονισμός πάνω στο κανάλι contGame και η επικοινωνία , μεταξύ εξυπηρετητή – πελάτη συνεχίζεται.
Τα μέρη αυτά φαίνονται στο πιο κάτω σχήμα.



Σχήμα 4.4 : Το μέρος PlayC του πελάτη μαζί με τα δύο μέρη του.

Στην συνέχεια θα δούμε την περιγραφεί σε ACSR του πελάτη. Θα δούμε τον πελάτη σαν μια διεργασία και στη συνέχεια θα μελετήσουμε τα επιμέρους κομμάτια του.

Πρώτα έχουμε τον πελάτη όπου αρχικά στέλλει ένα μήνυμα *joinGame* στον εξυπηρετητή για να ζητήσει να εισέλθει στο παιγνίδι.

$$Client[i] = \text{rec } x\ (\varnothing : x + (\overline{joinGame}, 1)\ .\ PlayC[i])$$

$i = 1, 2, \ldots, Np$

Στη συνέχεια αφού συγχρονιστεί με τον εξυπηρετητή στο κανάλι αυτό, τότε πάει στο PlayC που είναι υπεύθυνο για την επικοινωνία και το σχεδιασμό στην οθόνη του χρήστη της νέας κατάστασης του παιγνιδιού όπως αυτή διαμορφώνεται από τις κινήσεις των διαφόρων παιχτών.

$$PlayC[i] = (CommC[i] \parallel Graph[i]) \setminus \{drawGraph, contGame\}$$

Το κομμάτι PlayC του χρήστη ξεκινάει να τρέχει παράλληλα τα δύο κομμάτια του το CommC και το Graph. Τα κανάλια επικοινωνίας αυτών των δύο καναλιών είναι τα *drawGraph*, *contGame*. Και εδώ σε κάθε επανάληψη θα τρέχει το CommC και στην συνέχεια θα δίνει τον έλεγχο στην διαδικασία Graph για να υπολογίσει την νέα κατάσταση του παιγνιδιού. Ο χρόνος που πρέπει να γίνεται η πιο πάνω διαδικασία είναι και εδώ T_{frame} = 1000/25. Οι χρόνοι για τη διαδικασία CommC θα πρέπει να είναι T_{comm}, ενώ αντίστοιχα για την διαδικασία Graph θα είναι T_{graph}. Θα έχω δηλαδή $T_{frame} = T_{comm} + T_{graph}$.

Στην συνέχεια θα δούμε την υλοποίηση του μέρους CommC.

CommC[i] = CommC1[i] Δ^{Null}_{10000} (Nil, x, Nil)

CommC1[i] = ExchangeData [i] $\Delta^{contComm}_{Tcomm}$ (NextFrame[i],
($\overline{drawGraph}$,1) . NextFrame[i], Nil)

GetGameState[i] = rec x (∅ : x + (*gameState*, 2). ExchangeData[i])

ExchangeData[i] = ($\overline{moveInfo}$, 1). ($\overline{drawGraph}$, 1). ($\overline{contComm}$, 1).NIL +
($\overline{quitGame}$, 1). x

NextFrame[i] = rec x (∅ : x + (*contGame*,1). CommC[i])
i = 1, 2, …, Np

Αφού ξεκινήσει το παιγνίδι τότε ο κάθε χρήστης συγχρονίζεται με τον εξυπηρετητή και ανταλλάσσει πληροφορίες. Στη συνέχεια συγχρονίζεται με το κομμάτι Graph ώστε να σχεδιαστεί στην οθόνη του χρήστη η νέα κατάσταση. Μετά περιμένει να τελειώσει το κομμάτι Graph για να πάρει πίσω τον έλεγχο και να συνεχίσει τις λειτουργίες του.

Graph[i] = rec y (∅ : y + (*drawGraph*, 1).Draw Δ^{Null}_{Tgraph} (Nil,
DoneGraph[i], Nil)))

όπου Draw η διεργασία που σχεδιάζει στην οθόνη τη νέα κατάσταση του παιγνιδιού

DoneGraph[i] = rec z ($\varnothing$: z + ($\overline{contGame}$,1). Graph[i])

Εδώ μόλις το κομμάτι Graph συγχρονιστεί με το PlayC τότε σχεδιάζει στην οθόνη την νέα κατάσταση και στην συνέχεια δίνει τον έλεγχο πίσω στην PlayC αφού συγχρονιστεί με το κανάλι contGame

4.4.4 Περιγραφή του συστήματος μεταξύ εξυπηρετητή - πελάτη

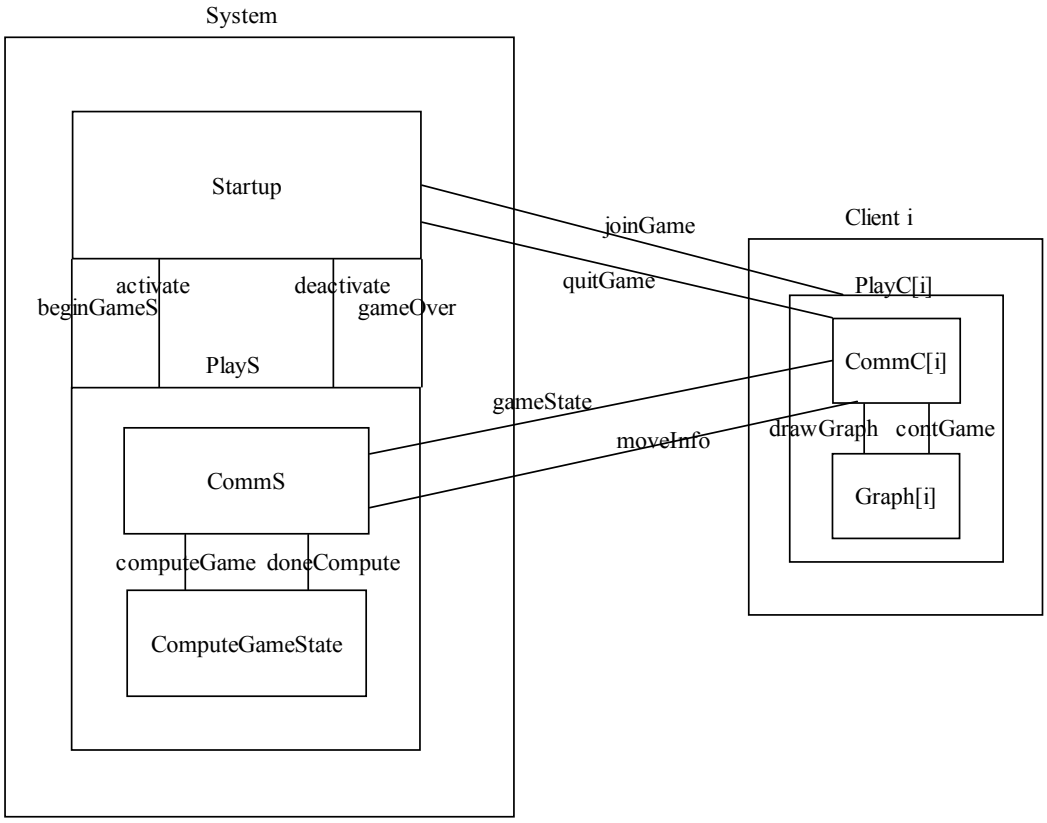
Εδώ θα δούμε την περιγραφή του συστήματος επικοινωνίας. Εκτός από τα κανάλια μεταξύ των επιμέρους κομματιών του εξυπηρετητή και πελάτη έχουμε και τα κανάλια μεταξύ του εξυπηρετητή και του πελάτη.

Αρχικά καθώς τρέχει ο εξυπηρετητής πρέπει να συγχρονιστεί με το πρώτο χρήστη (πελάτη) που έρχεται στο σύστημα και στη συνέχεια να ξεκινήσει το παιγνίδι. Επίσης θα πρέπει να υπάρχει συγχρονισμός κάθε φορά που φεύγει ένας παίκτης από το σύστημα.

Η επικοινωνία αυτή επιτυγχάνεται με τα δύο κανάλια joinGame, quitGame, τα οποία ενώνουν κάθε χρήστη (πελάτη) με τον εξυπηρετητή. Συγκεκριμένα το κανάλι joinGame ενώνει το κομμάτι PlayC του κάθε χρήστη με το κομμάτι Startup του εξυπηρετητή, ενώ το quitGame ενώνει το υπομέρος CommC του κομματιού PlayC του χρήστη με το κομμάτι Startup του εξυπηρετητή.

Επιπλέον έχουμε την επικοινωνία που αφορά την μεταφορά των δεδομένων με τα οποία συγχρονίζεται ο εξυπηρετητής με τον κάθε χρήστη. Αυτή επιτυγχάνεται με τα κανάλια gameState, moveInfo. Το gameState, χρησιμοποιείται για την αποστολή της νέας κατάστασης του παιγνιδιού από τον εξυπηρετητή προς ένα χρήστη (πελάτη). Το moveInfo χρησιμοποιείται για την αποστολή της κίνησης του κάθε χρήστη προς τον εξυπηρετητή.

Στο σχήμα 4.5 φαίνεται το ολοκληρωμένο σύστημα με ένα πελάτη. Βλέπουμε την επικοινωνία μεταξύ των δύο στα επιμέρους κομμάτια του κάθε ενός



Σχήμα 4.5 : Το ολοκληρωμένο σχήμα του συστήματος επικοινωνίας.

Κεφάλαιο 5

Μετάφραση και Σχεδίαση

5.1 Εισαγωγή στην μετάφραση και σχεδιασμό

Στις προηγούμενες ενότητες μελετήσαμε και αναλύσαμε το σύστημα, το παιγνίδι, καθώς και την σχέση που τα ενώνει. Επιπλέον είδαμε και τις προδιαγραφές του κομματιού της επικοινωνίας του με την χρήση της ACSR.

Σε αυτό το κεφάλαιο θα δούμε τη «μετάφραση» από την ACSR στη γλώσσα προγραμματισμού Java, και θα τη σχεδίαση του συστήματος της επικοινωνίας και του παιγνιδιού. Στην συνέχεια θα δούμε τον τρόπο που προγραμματιστικά θα ενωθούν τα δύο μέρη.

5.2 Μετάφραση

Στο υποκεφάλαιο αυτό θα μιλήσουμε για τους τρόπους με τους οποίους μπορούμε από τις προδιαγραφές στην άλγεβρα διεργασιών ACSR ενός συστήματος, να πάμε στην υλοποίηση του συστήματος με τη χρήση της γλώσσας προγραμματισμού Java.

Η μετάφραση αυτή δεν αποτελεί ένα συγκεκριμένο τρόπο με τον οποίο κάποιο σύστημα γραμμένο στην ACSR μπορεί να μεταφραστεί απόλυτα στην γλώσσα

προγραμματισμού Java, αλλά αποτελεί κατευθυντήριες γραμμές με τις οποίες κάποιος χρήστης θα βοηθηθεί ώστε να καταλάβει κάποιες έννοιες αλλά και την σχέση μεταξύ της ACSR και της Java. Ακόμα, πιο γενικά, μπορούμε να πούμε ότι αυτή η μετάφραση θα μπορούσε να αποτελεί κάποιες κατευθυντήριες γραμμές για μετάφραση από ACSR και σε κάποια άλλη γλώσσα προγραμματισμού η οποία υποστηρίζει παρόμοιες λειτουργίες με την Java, ή ακόμα και από άλλες άλγεβρες διεργασιών σε γλώσσες με παρόμοιες λειτουργίες με την Java.

5.2.1 Κοινά Χαρακτηριστικά Java και ACSR

Για να μιλήσουμε για τους τρόπους με τους οποίους θα γίνει η μετάφραση από την ACSR στην Java θα πρέπει πρώτα να καταλάβουμε κάποια κοινά χαρακτηριστικά τα οποία έχουν μεταξύ τους.

Η ACSR, όπως οι περισσότερες άλγεβρες διεργασιών, χρησιμοποιούνται για να μελετούν και να αναλύουν συστήματα πραγματικού χρόνου. Άρα και οι λειτουργίες που προσφέρουν θα πρέπει να περιέχουν μηχανισμούς οι οποίοι περιγράφουν αυτά τα συστήματα. Μερικές λειτουργίες που υποστηρίζουν είναι αυτή των πολλών διεργασιών που μπορούν να τρέχουν σε ένα σύστημα παράλληλα η σύγχρονα, οι λειτουργίες για επικοινωνία μεταξύ διάφορων διεργασιών καθώς και λειτουργίες για συγχρονισμό διεργασιών μεταξύ τους. Αυτές οι λειτουργίες είναι λειτουργίες που εμφανίζονται σε συστήματα πραγματικού χρόνου όπου μπορούμε να έχουμε πολλές διεργασίες να τρέχουν οι οποίες μπορεί να συναγωνίζονται μεταξύ τους για πόρους του συστήματος, να συνεργάζονται για την επίτευξη κάποιου κοινού στόχου ή και να λειτουργούν εντελώς ανεξάρτητα η μια από την άλλη.

Η Java από την άλλη, είναι μια γλώσσα προγραμματισμού η οποία υποστηρίζει πολλές από τις λειτουργίες αυτές. Επιτρέπει την εκτέλεση πολλών νημάτων (threads) τα οποία μπορούν να παρομοιαστούν με τις διεργασίες που τρέχει ένα σύστημα πραγματικού χρόνου. Επιπλέον υποστηρίζει εύκολα την επικοινωνία μεταξύ προγραμμάτων που βρίσκονται στην ίδια αλλά και διαφορετικών μηχανών. Υποστηρίζει την χρήση monitors τα οποία επιτρέπουν συγχρονισμό μεταξύ δύο ή περισσότερων εμπλεκομένων διεργασιών, και επίσης προσφέρει και μηχανισμούς για προστασία της μηχανής στην οποία τρέχει το πρόγραμμα από κακόβουλες ενέργειες. Τα

πιο πάνω κάνουν την Java να παρομοιάζεται με ένα λειτουργικό σύστημα το οποίο προσφέρει πολλές από αυτές τις λειτουργίες.

5.2.2 Κανόνες μετάφρασης

Όπως είπαμε και πριν, η μετάφραση αυτή δεν αποτελεί ένα συγκεκριμένο τρόπο με τον οποίο κάποιο σύστημα γραμμένο στην ACSR μπορεί να μεταφραστεί απόλυτα στην γλώσσα προγραμματισμού Java, αλλά αποτελεί κατευθυντήριες γραμμές με τις οποίες κάποιος χρήστης θα βοηθηθεί ώστε να καταλάβει κάποιες έννοιες αλλά και τη σχέση μεταξύ της ACSR και της Java, ώστε να μπορέσει να μετατρέψει κάποιο ACSR μοντέλο σε κάποιο πρόγραμμα γραμμένο στη Java. Αν κοιτάξουμε προσεχτικά τα κοινά χαρακτηριστικά πιο πάνω βλέπουμε μια σχέση μεταξύ ACSR και Java. Συγκεκριμένα η ACSR έχει διεργασίες που τρέχουν παράλληλα ή σύγχρονα, και η Java υποστηρίζει νήματα τα οποία τρέχουν σε μια διεργασία. Άρα θα μπορούσαμε να δούμε την κάθε παράλληλη διεργασία της ACSR σαν ένα νήμα το οποίο τρέχει σε ένα πρόγραμμα Java. Η διαφορά είναι ότι στην ACSR έχουμε διεργασίες οι οποίες τρέχουν πραγματικά ταυτόχρονα, και διεργασίες οι οποίες τρέχουν σύγχρονα. Με τον όρο παράλληλα εννοούμε ότι δύο διεργασίες καταναλώνοντας ένα μη κοινό πόρο προχωρούν μαζί σε μια χρονική στιγμή στην επόμενη κατάσταση (κανόνας ParT κεφάλαιο 2), ή οι δύο εμπλεκόμενες διεργασίες εκτελώντας ένα κοινό γεγονός (π.χ. η μια α και η άλλη $\bar{a}$) συγχρονίζονται μαζί και προχωρούν στην επόμενη κατάσταση τους (κανόνας ParCom κεφάλαιο 2). Με τον όρο σύγχρονα εννοούμε τις διεργασίες που μοιράζονται κάποιους κοινούς πόρους, έτσι αναγκαστικά κατά την εκτέλεση τους γεγονότα των διεργασιών αυτών παρεμβάλλονται μεταξύ τους (κανόνας ParT όταν $(\rho(R_1) \cap \rho(R_2) \neq \varnothing$ κεφάλαιο 2 και κανόνες ParIL και ParIR κεφάλαιο 2). Η Java αφού τρέχει σε ένα υπολογιστή τρέχει όλα τα νήματα της σύγχρονα. Υπάρχει η δυνατότητα όμως να τρέχουμε την Java σε διαφορετικούς υπολογιστές οι οποίοι επικοινωνούν μαζί τους, και τα νήματα να τρέχουν πραγματικά ταυτόχρονα.

Ο τρόπος που θα δούμε εμείς την μετάφραση της Java και της ACSR είναι ο εξής. Κάθε παράλληλη διεργασία της ACSR να μπορεί να μετατραπεί σε ένα ξεχωριστό πρόγραμμα Java το οποίο τρέχει στην ίδια ή διαφορετική μηχανή. Η κάθε σύγχρονη διεργασία θα μπορούσε να γίνει σαν ένα νήμα το οποίο τρέχει σε ένα πρόγραμμα Java.

Για παράδειγμα έστω ότι έχω ένα σύστημα στο οποίο δύο χρήστες οι οποίοι βρίσκονται σε διαφορετικές μηχανές επικοινωνούν μεταξύ τους μέσω κάποιας άλλης τρίτης μηχανής η οποία συγχρονίζει τους δύο και τους φέρνει μαζί.

Το σύστημα σε ACSR θα ήταν το εξής

System = User1 || User2 || UserCommunication

όπου στο σύστημα μας τρέχουν φυσικά ο χρήστης 1, ο χρήστης 2 και το σύστημα πους τους ενώνει. Αφού οι τρεις διεργασίες βρίσκονται σε διαφορετικές μηχανές άρα ισχύει $\rho(R_1) \cap \rho(R_2) \cap \rho(R_3) = \varnothing$ όπου R_1 οι πόροι του User1, R_2 οι πόροι του User2, R_3 οι πόροι του User3. Άρα με βάση τον κανόνα του ParT κεφάλαιο 2, οι διεργασίες τρέχουν πραγματικά παράλληλα. Άρα θα μπορούσαμε να δούμε το κάθε κομμάτι του πιο πάνω συστήματος σαν ένα ξεχωριστό πρόγραμμα, μια ξεχωριστή κλάση η οποία θα τρέχει σε διαφορετικές μηχανές.

Τώρα για τον κάθε χρήστη θα μπορούσαμε να πούμε ότι έχουμε δύο κύρια μέρη σε αυτό. Το πρώτο μέρος είναι εκείνο το οποίο επικοινωνεί με τον χρήστη και α) δίνει σε αυτό την κατάσταση του άλλου χρήστη την οποία πήρε από στην μηχανή που συγχρονίζει τους χρήστες, και β) λαμβάνει από αυτόν τα δεδομένα του. Το δεύτερο μέρος θα είναι εκείνο που επικοινωνεί με την μηχανή που συγχρονίζει την επικοινωνία με τους χρήστες.

Οι προδιαγραφές του είναι οι εξής.

User1 = (UserInteraction1 || UserComm1)\{*userDone*, *showUser*}

UserInteraction1 = rec x.($\varnothing$: x + ($\overline{userDone}$,1). x + (*showUser*,1). x)

UserComm1 = rec y.($\varnothing$: y + (*receiveInfo1*,1).($\overline{showUser}$,1).y + (*userDone*, 1). ($\overline{sendInfo1}$,1).y)

Ο χρήστης δίνει τις πληροφορίες του και αφού το σύστημα τις παραλάβει τις μεταφέρει στο μέρος που είναι υπεύθυνο για την επικοινωνία. Το μέρος αυτό τότε στέλνει τις πληροφορίες στην μηχανή που συντονίζει την επικοινωνία. Επίσης το μέρος αυτό περιμένει να ακούσει για πληροφορίες τους άλλου χρήστη και αφού τις λάβει τις δίνει στο πρώτο μέρος για να μεταδοθούν στον χρήστη.

Ομοίως ο User2 θα έχει παρόμοια περιγραφή με την διαφορά ότι τα κανάλια *receiveInfo1* και *sendInfo1* γίνονται *receiveInfo2* και *sendInfo2*.

Άρα στην μηχανή κάθε χρήστη που τρέχει ένα πρόγραμμα Java πρέπει να τρέξουν και τα δύο μέρη του User. Αυτό μπορούμε να το δούμε σαν δύο νήματα τα οποία τρέχουν σύγχρονα μέσα σε μια διεργασία, αφού το ένα ολοκληρώνοντας την λειτουργία του δίνει τον έλεγχο στον άλλο για να συνεχιστεί η εκτέλεση του. Για να συνεχιστεί δηλαδή η εκτέλεση του συστήματος στην επόμενη κατάσταση, αν σταλεί κάποιο μήνυμα προς κάποιο χρήστη τότε το μήνυμα (το οποίο μπορούμε να πούμε ότι είναι ο κοινός πόρος μεταξύ των δύο μερών), λαμβάνεται από τον ένα μέρος του User και στην συνέχεια δίνεται στο άλλο μέρος για να το δώσει στον χρήστη.

Η μηχανή τώρα που συγχρονίζει τους δύο χρήστες θα πρέπει να στέλλει τις πληροφορίες που λαμβάνει από ένα χρήστη στο άλλο και αντίστροφα.
Οι προδιαγραφές του είναι οι εξής.

UserCommunication = rec x.($\varnothing$: x + (*sendInfo1*, 1). ($\overline{\textit{receiveInfo2}}$, 1). x + (*sendInfo2*, 1).($\overline{\textit{receiveInfo1}}$,1). x)

Δηλαδή τις πληροφορίες που παίρνει από τον ένα χρήστη τις δίνει στον άλλο και αντίστροφα.

Εδώ στην μηχανή που συγχρονίζει τους δύο χρήστες δεν υπάρχουν άλλες παράλληλες διεργασίες και μπορούμε να δούμε αυτή την κλάση σαν ένα πρόγραμμα Java το οποίο τρέχει στην μηχανή που συντονίζει, και ανταλλάσσει μηνύματα με τους χρήστες (Users).

Αφού θέλουμε να πετύχουμε συγχρονισμό στο σύστημα μεταξύ των χρηστών και της μηχανής που τους συγχρονίζει, θα πρέπει να βάλουμε περιορισμό στο σύστημα πάνω στα κανάλια *receiveInfo1*, *sendInfo*, *receiveInfo2*, και *sendInfo2*.
System = (User1 || User2 || UserCommunication) \ {*receiveInfo1*, *sendInfo1*, *receiveInfo2*, *sendInfo2*}

Το επόμενο που θα δούμε στην μετάφραση θα είναι η χρήση των events. Τα events αντιπροσωπεύουν μια επικοινωνία μεταξύ των διεργασιών. Μπορούμε να δούμε τον συγχρονισμό μεταξύ δύο events σαν την αμφίδρομη επικοινωνία μεταξύ δύο διεργασιών. Αφού έχουμε μιλήσει για την μετατροπή δύο διεργασιών σε είτε παράλληλες διεργασίες που τρέχουν σε διαφορετικές μηχανές, είτε σε διεργασίες που

τρέχουν σύγχρονα, θα πρέπει να μιλήσουμε για το πως μεταφράζονται τα events σε αυτές τις δύο περιπτώσεις.

Για την πρώτη περίπτωση ένα event τύπου $\bar{a}$ εννοεί την αποστολή μηνύματος πάνω στο κανάλι α, ενώ το event τύπου a εννοεί την παραλαβή ενός μηνύματος πάνω στο κανάλι α. Αφού όταν οι διεργασίες τρέχουν πραγματικά παράλληλα τότε ο μόνος τρόπος επικοινωνίας αλλά και συγχρονισμού γίνεται με μηνύματα, αρά σε αυτή την περίπτωση όπου έχω δύο ξεχωριστά προγράμματα θα ανταλλάσσουν μηνύματα για συγχρονισμό μεταξύ τους.

Για την δεύτερη περίπτωση όπου δηλαδή έχω διεργασίες να τρέχουν σύγχρονα σε ένα πρόγραμμα, ο συγχρονισμός μπορεί να γίνει και εδώ με την ανταλλαγή μηνυμάτων μεταξύ αντικειμένων, δηλαδή με την αποστολή μηνυμάτων από ένα αντικείμενο σε ένα άλλο. Αυτό επιτυγχάνεται με το κάλεσμα τις κατάλληλης συνάρτησης ενός αντικείμενου από ένα άλλο. Στην περίπτωση του χρήστη (User1) πιο πάνω, το μέρος που επικοινωνεί με τη μηχανή που συγχρονίζει, αφού πάρει τις πληροφορίες θα πρέπει να καλέσει (να στείλει μήνυμα στο αντικείμενο δηλαδή), την κατάλληλη συνάρτηση από το μέρος το οποίο επικοινωνεί με τον χρήστη ώστε να μπορεί να συγχρονιστεί και να περάσει σε αυτόν τις αναγκαίες πληροφορίες. Όταν ο χρήστης δώσει δικές του πληροφορίες για αποστολή, το κομμάτι που επικοινωνεί με τον χρήστη θα καλέσει κατάλληλη συνάρτηση από το μέρος που κάνει την επικοινωνία με την μηχανή που συντονίζει για να συγχρονιστεί μαζί του και να του περάσει τις πληροφορίες για αποστολή.

Γενικά μπορούμε να πούμε ότι τα γεγονότα μεταξύ δύο διεργασιών τα οποία μεταφράζονται σε διαφορετικά προγράμματα, μετατρέπονται σε μηνύματα μεταξύ των προγραμμάτων, ενώ τα γεγονότα σε διεργασίες που μεταφράζονται σε νήματα σε ένα πρόγραμμα θα μεταφράζονται σε ανταλλαγή μηνύματος του ενός νήματος με το άλλο, δηλαδή κάλεσμα κατάλληλης συνάρτησης από το ένα νήμα σε κάποιο άλλο.

Η πιο πάνω μετάφραση φυσικά δεν αποτελεί το μοναδικό τρόπο με τον οποίο ένα σύστημα στην ACSR μπορεί να μετατραπεί σε πρόγραμμα γραμμένο στην Java, αλλά αποτελεί όπως έχουμε πει κατευθυντήριες γραμμές για κάποιον να καταλάβει μια σχέση μεταξύ ACSR και Java. Υπάρχουν πολλοί προγραμματιστικοί τρόποι να δούμε την μετάφραση από μια άλγεβρα διεργασιών σε μια γλώσσα προγραμματισμού. Εδώ απλά αναφέρουμε μια πιθανή μετάφραση από ένα σύστημα με προδιαγραφές στην ACSR στην υλοποίηση του συστήματος αυτού στη γλώσσα προγραμματισμού Java.

Αυτές οι ιδέες μπορεί να χρησιμοποιηθούν όμως και για άλλες πιθανές άλγεβρες διεργασιών και γλώσσες προγραμματισμού οι οποίες έχουν κάποιες κοινές ιδιότητες με αυτές εδώ.

Για τους υπόλοιπους τελεστές τώρα δεν μπορούμε να πούμε με ακρίβεια σε τί ακριβώς θα μεταφραστούν στην Java, π.χ. για το A = rec x.(∅ : x + B), όπου το A βρίσκεται είτε σε αδράνεια είτε εξελίσσεται σε B, μπορεί να μεταφραστεί σε ένα πρόγραμμα το οποίο βρίσκεται ανενεργό σε ένα monitor χωρίς να κάνει τίποτε, ή μπορεί να περιμένει μια εντολή I/O, ή ακόμα μπορεί να κάνει μια άπειρη επανάληψη χωρίς να κάνει τίποτε. Εξαρτάται από τί θέλει ο καθένας στην σχεδιάση ή στην υλοποίηση του. Το ίδιο ισχύει και για τους υπόλοιπους τελεστές όπως της επιλογής (+) το οποίο μπορεί να μεταφραστεί σαν τις επιλογές που κάνει ένα πρόγραμμα κατά την εκτέλεση, ο τελεστής του πεδίου (Δ_t^a) ο οποίος μπορεί να μεταφραστεί σαν μια άλλη κλάση η οποία τρέχει παράλληλα με την διεργασία που δεσμεύεται από αυτή κ.τ.λ.

5.2.3 Μετάφραση συστήματος επικοινωνίας

Γνωρίζοντας τα ποιο πάνω θα προχωρήσουμε να μεταφράσουμε αρχικά το σύστημα επικοινωνίας από τις προδιαγραφές που έχουμε δώσει σε ACSR στην γλώσσα προγραμματισμού Java. Ας θυμηθούμε λίγο το σύστημα επικοινωνίας. Στο σύστημα μας έχουμε δύο κύρια μέρη. Το μέρος του εξυπηρετητή και το μέρος των πελατών. Το πρώτο τρέχει σε μια μηχανή στον δικτυακό μας τόπο, και το δεύτερο αποτελείται από n πελάτες που τρέχουν σε διαφορετικές μηχανές το καθένα στον διαδίκτυο. Ο εξυπηρετητής τρέχει μια διεργασία, την StartUp, η οποία είναι υπεύθυνη για να ακούει για αιτήσεις από πελάτες για να εισέλθουν στο παιγνίδι, και μια άλλη διεργασία η PlayS η οποία ξεκινά ή σταματά το παιγνίδι, η οποία όμως χωρίζεται στη διεργασία η οποία υπολογίζει την επόμενη κατάσταση του παιγνιδιού, την ComputeGameState, αλλά και μια άλλη διεργασία, την AllClients, η οποία τρέχει n άλλες διεργασίες, μία για κάθε πελάτη, οι οποίες έχουν την ευθύνη για ανταλλαγή πληροφορίας με αυτούς. Ο πελάτης τρέχει ένα μέρος το οποίο είναι υπεύθυνο για την επικοινωνία με τον εξυπηρετητή, το CommC, και ένα μέρος, το Graph, το οποίο επικοινωνεί με τον χρήστη και παίρνει από αυτόν τις κινήσεις του αλλά και του δίνει τη νέα κατάσταση του παιγνιδιού σχεδιάζοντας την στην οθόνη του υπολογιστή του χρήστη.

Με βάση τους κανόνες μετάφρασης που δώσαμε πιο πάνω αφού ο εξυπηρετητής και οι n πελάτες τρέχουν σε διαφορετικές μηχανές, τότε δεν έχουν κοινούς πόρους. Άρα θα μπορούσαμε να τους δούμε σαν πραγματικά παράλληλα προγράμματα τα οποία επικοινωνούν μεταξύ του με αποστολές μηνυμάτων.

Άρα αρχικά θα πρέπει να έχουμε δύο κύρια προγράμματα, το ένα θα είναι ο εξυπηρετητής, και το άλλο θα είναι οι πελάτες. Αφού έχω n πελάτες που τρέχουν σε διαφορετικές μηχανές θα πρέπει να έχω και n διαφορετικά προγράμματα. Αφού όμως οι πελάτες είναι συμμετρικοί ο ένας με το άλλο τότε θα μπορούσα να δημιουργήσω μόνο ένα πρόγραμμα τύπου πελάτης το οποίο θα τρέχει ένα στιγμιότυπου σε κάθε μηχανή που ο χρήστης θα θελήσει να χρησιμοποιήσει. Άρα αρχικά στην υλοποίηση θα έχω δύο κύρια προγράμματα, ένα για τον εξυπηρετητή και ένα για τον πελάτη. Τα δύο αυτά κομμάτια θα επικοινωνούν και θα συγχρονίζονται μεταξύ τους μέσω ανταλλαγή μηνύματος, αφού δεν θα τρέχουν σύγχρονα στο ίδιο πρόγραμμα και ο μόνος τρόπος επικοινωνίας είναι η ανταλλαγή μηνύματος όπως αναφέρουμε πιο πάνω. Άρα θα υπάρχει ένα κανάλι επικοινωνίας μεταξύ του κάθε πελάτη και του εξυπηρετητή.

Ο εξυπηρετητής τώρα τρέχει τα διάφορα μέρη του σύγχρονα αφού, πρώτον, το κάθε μέρος του από αυτά δεν χρειάζεται να βρίσκεται σε διαφορετικές μηχανές, και, δεύτερον, θα θέλαμε το καθένα να συγχρονίζεται με κάποιο άλλο για να γίνουν σωστά οι λειτουργίες γρήγορα ώστε να πετύχουμε καλύτερη απόδοση. Το πρώτο μέρος το StartUp, χρειάζεται να συγχρονίζεται με το PlayS, το μέρος που ξεκινά το παιγνίδι, το ComputeGameState χρειάζεται να συγχρονίζεται με τις διεργασίες που επικοινωνούν με τους πελάτες, οι διεργασίες που επικοινωνούν με τους πελάτες να συγχρονίζονται με το StartUp όταν ένα παίχτης φύγει κ.τ.λ.

Άρα στο εξυπηρετητή θα έχουμε ένα πρόγραμμα στο οποίο θα τρέχουν συνολικά n + 2 νήματα. Ένα νήμα για να περιμένει να ακούσει από κάποιο πελάτη να εισέλθει στο παιγνίδι, ένα νήμα για να υπολογίζει τη νέα κατάσταση του παιγνιδιού, και n νήματα ένα για κάθε πελάτη που έρχεται στο παιγνίδι και επικοινωνεί κάθε φορά με τον εξυπηρετητή. Τα νήματα αυτά μέσα στο εξυπηρετητή θα αποστέλλουν μηνύματα το ένα αντικείμενο στο άλλο καλώντας τις κατάλληλες μεθόδους, ώστε να επιτυγχάνεται επικοινωνία κα συγχρονισμός σε αυτά.

Ο πελάτης επίσης θα τρέχει όλα του τα μέρη σύγχρονα αφού και εδώ δεν χρειάζεται να βρίσκονται οι διάφορες διεργασίες σε διαφορετικές μηχανές αλλά και οι διεργασίες αυτές πρέπει να συγχρονίζονται όσο το δυνατό πιο γρήγορα για να έχουμε

όσο το δυνατό καλύτερα αποτελέσματα. Το πρώτο μέρος του πελάτη αφού κάνει μια ανταλλαγή πληροφορίας με τον εξυπηρετητή, τότε συγχρονίζεται με το κομμάτι που επικοινωνεί με τον χρήστη. Στη συνέχεια αυτό αφού σχεδιάσει στην οθόνη του χρήστη τη νέα κατάσταση και ακούσει από αυτόν τις κινήσεις του (αν υπάρχουν), τότε συγχρονίζεται με το μέρος που επικοινωνεί με τον εξυπηρετητή και του περνάει τον έλεγχο.

Άρα στον κάθε πελάτη θέλω δύο νήματα, ένα το οποίο αφού στείλει αίτημα στο εξυπηρετητή για να εισέλθει στο παιγνίδι, τότε αφού πάρει θετική ανταπόκριση από τον εξυπηρετητή ξεκινάει την επικοινωνία, και ένα άλλο νήμα το οποίο σχεδιάζει στην οθόνη του υπολογιστή αλλά ακούει και από τον χρήστη την επόμενη κατάσταση του. Τα νήματα αυτά και εδώ επικοινωνούν με ανταλλαγή μηνυμάτων του ενός με του άλλου (κάλεσμα κατάλληλης μεθόδου), ώστε να επιτυγχάνεται η επικοινωνία και ο συγχρονισμός σε αυτά.

5.3 Σχεδίαση

Στη σχεδίαση θα δούμε το σύστημα που έχουμε περιγράψει πιο πάνω, το σύστημα επικοινωνίας δηλαδή, αλλά θα δούμε και την σχεδίαση του συστήματος παιγνιδιού. Τέλος θα δούμε πως τα δύο αυτά ενώνονται μεταξύ τους.

5.3.1 Σχεδίαση συστήματος επικοινωνίας

Στην σχεδίαση του συστήματος επικοινωνίας όπως έχουμε πει στην μετάφραση του συστήματος ποιο πάνω, έχουμε δύο κύρια προγράμματα, άρα αρχικά έχουμε και δύο διαφορετικές κλάσεις. Επιπλέον μιλήσαμε για τα μέρη του εξυπηρετητή και του πελάτη.

Από την πλευρά του εξυπηρετητή όπως έχουμε αναφέρει, υπάρχουν n+2 νήματα. Το κάθε νήμα αποτελεί διαφορετικό αντικείμενο το οποίο τρέχει από μόνο του. Από αυτά τα n+2 νήματα ξεχωρίζουμε τα δύο που είναι α) εκείνο που υπολογίζει την νέα κατάσταση και β) εκείνο που ακούει για πελάτες που κάνουν αίτηση για να εισέλθουν στο σύστημα. Το κάθε νήμα αποτελεί ξεχωριστό αντικείμενο στην Java και άρα χρειάζεται να σχεδιάσουμε ξεχωριστή κλάση για το καθένα.

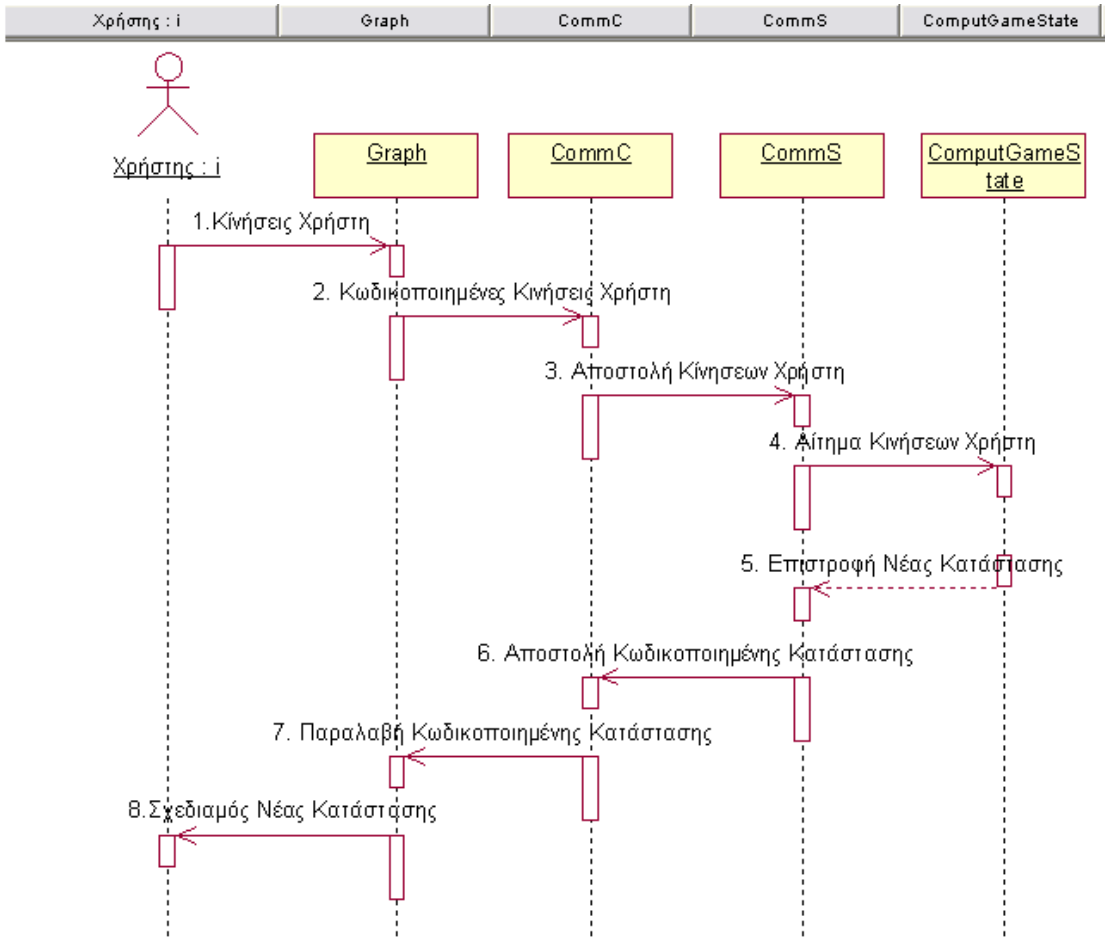
Τα υπόλοιπα n νήματα είναι εκείνα που επικοινωνούν με τους n πελάτες. Λόγο συμμετρικότητας ανάμεσα σε αυτά χρειάζεται ο σχεδιασμός μια μόνο κλάσης

(αντικειμένου), και κάθε ένα από αυτά τα νήματα θα είναι ένα στιγμιότυπο αυτού του αντικειμένου.

Ο πελάτης τώρα έχει μόνο δύο νήματα να τρέχουν σε αυτό. Αυτό μας λέει ότι χρειάζονται δύο κλάσεις για να περιγράψουμε την λειτουργία του. Ένα αντικείμενο το οποίο θα ανταλλάσσει πληροφορίες με τον εξυπηρετητή, και ένα άλλο το οποίο θα επικοινωνεί με τον χρήστη. Ας θυμηθούμε από το υποκεφάλαιο 5.2.3 Μετάφραση συστήματος επικοινωνίας, οι πελάτες είναι συμμετρικοί μεταξύ τους και για αυτό όλοι οι πελάτες του συστήματος θα είναι στιγμιότυπα αυτής της κλάσης.

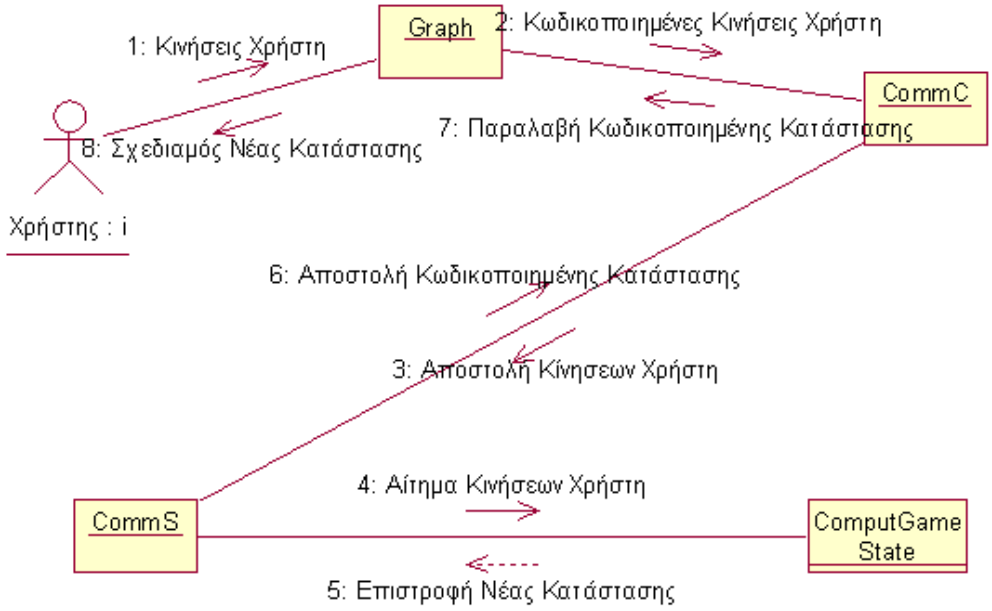
Στην συνέχεια θα αναλύσουμε το σενάριο κατά το οποίο επιτυγχάνεται μια επικοινωνία μεταξύ ενός πελάτη και του εξυπηρετητή. Για να το δούμε αυτό κάνουμε την χρήση του sequence και collaboration diagram.

Το ποιο κάτω sequence diagram δείχνει την ανταλλαγή μηνυμάτων κατά την διάρκεια μιας χρονικής περιόδου κατά την οποία γίνεται μια ανταλλαγή μηνυμάτων μεταξύ χρήστη - πελάτη, πελάτη – εξυπηρετητή, εξυπηρετητή – πελάτη, πελάτη - χρήστη.



Σχήμα 5.1 : Ανταλλαγή μηνύματος ανάμεσα στο εξυπηρετητή και πελάτη

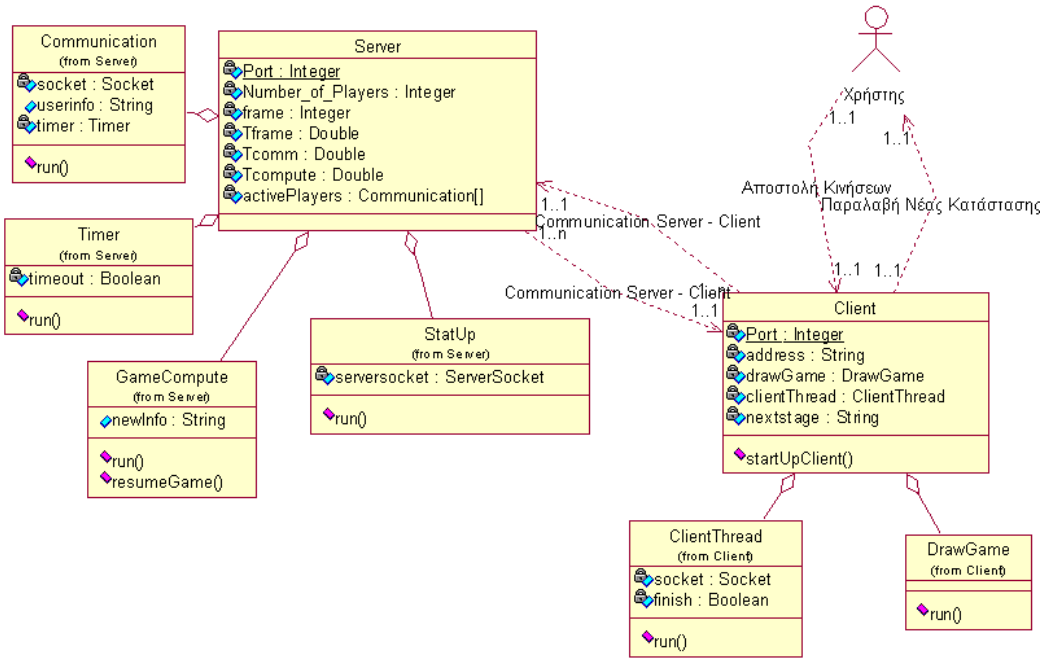
Στην συνέχεια βλέπουμε το collaboration diagram που δείχνει την σχέση που υπάρχει από τον χρήστη εώς τον εξυπηρετητή και πίσω



Σχήμα 5.2 : Ανταλλαγή μηνυμάτων μεταξύ χρήστη και παιγνιδιού

Στα δύο πιο πάνω σχήματα βλέπουμε τη ροή πληροφορίας από τον χρήστη προς το σύστημα μέχρι να υπολογιστεί η νέα κατάσταση, και στην συνέχεια τη ροή πληροφορίας από τον σύστημα προς το χρήστη.

Στην συνέχεια θα δούμε το class diagram το οποίο μας δείχνει την σχέση μεταξύ των κλάσεων που θα σχεδιάσουμε.



Σχήμα 5.3 : Οι κλάσεις του συστήματος επικοινωνίας σε σχέση με τον χρήστη

Στο ποιο πάνω σχεδιάγραμμα φαίνεται η σχέση μεταξύ των κλάσεων του συστήματος που τρέχουν σε διαφορετικές μηχανές (εξυπηρετητής – Server , πελάτης – Client), αλλά και τις υποκλάσεις σε κάθε μια από αυτές. Επίσης φαίνεται ο χρήστης σαν μια κλάση η οποία στέλλει μήνυμα στο σύστημα, και συγκεκριμένα στον πελάτη, ο οποίος με την σειρά του στέλλει τις πληροφορίες στο εξυπηρετητή. Όταν υπολογιστεί η νέα κατάσταση στέλνεται πίσω στον πελάτη για να την μεταδώσει στον χρήστη.

5.3.2 Σχεδίαση συστήματος παιγνιδιού

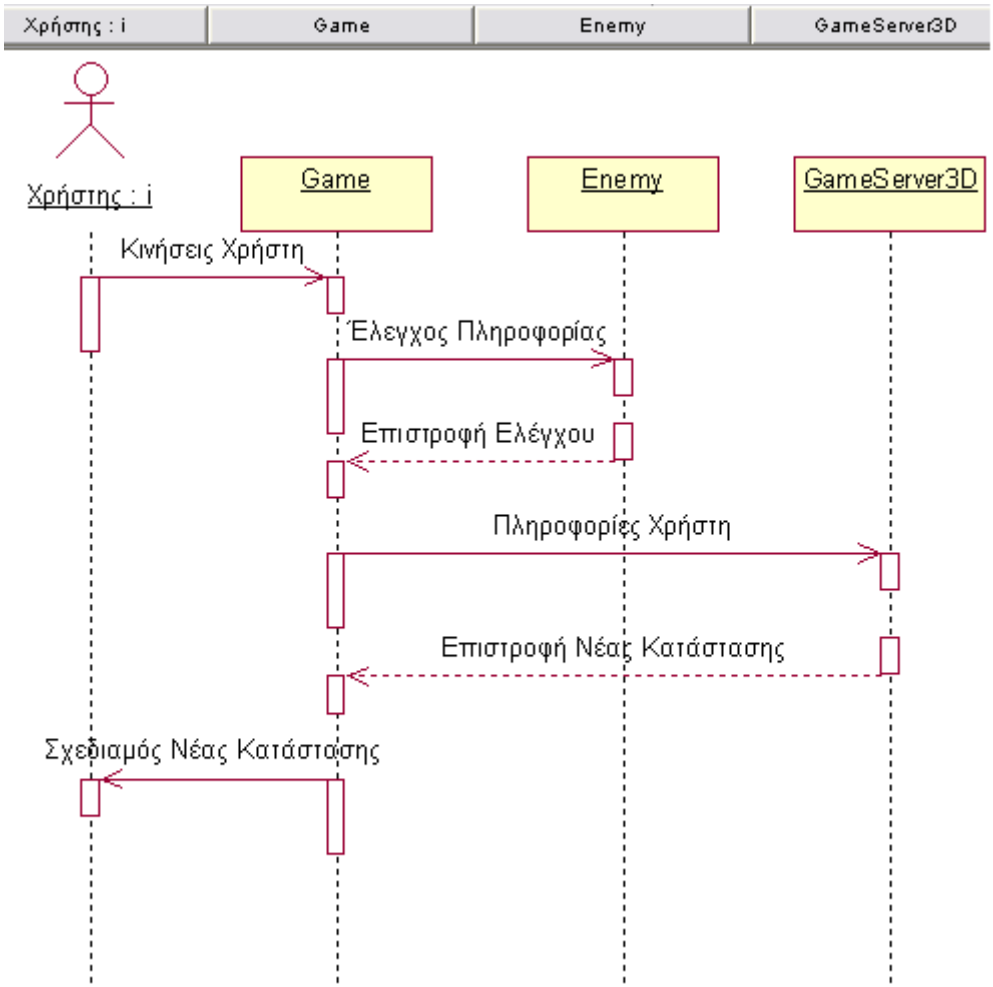
Το παιγνίδι που έχουμε αποφασίσει να υλοποιήσουμε όπως έχουμε πει είναι το Ghostbusters. Ας θυμηθούμε πως δουλεύει ακριβώς.

Στο παιγνίδι υπάρχουν οχτώ φαντάσματα (εχθροί), τα οποία κινούνται κατά μήκος της οθόνης με διαφορετική ταχύτητα και σε διαφορετικό ύψος. Ο χρήστης καλείται να κτυπήσει κάποιο φάντασμα με την κίνηση και την χρήση ενός κουμπιού του ποντικιού. Αν ο χρήστης επιτυχώς κτυπήσει ένα φάντασμα (εχθρό), τότε

πιστώνεται με πέντε μονάδες. Κάθε φορά που ένας χρήστης κτυπήσει ένα φάντασμα τότε το φάντασμα επαναδημιουργήται. Το όνομα του κάθε χρήστη εμφανίζεται στην κάτω αριστερά θέση τις οθόνης μαζί με τους βαθμούς του.

Το παιγνίδι θα πρέπει να σχεδιαστεί με τέτοιο τρόπο ώστε να μπορεί να χωριστεί σε δύο κύρια μέρη. Το μέρος που υπολογίζει την νέα κατάσταση το οποίο θα πρέπει να πάει στον εξυπηρετητή, και το μέρος που λαμβάνει από τον χρήστη κινήσεις και σχεδιάζει στην οθόνη την νέα κατάσταση το οποίο θα πάει στον πελάτη.
Άρα και εδώ χωρίζω το παιγνίδι σε δύο κύριες κλάσεις. Την κλάση που υπολογίζει την νέα κατάσταση του παιγνιδιού την GameServer3D, και την κλάση που σχεδιάζει τις πληροφορίες στην οθόνη του χρήστη την Game. Επιπλέον θα έχουμε και μία κλάση την Enemy όπου θα είναι υπεύθυνη για τις πληροφορίες του κάθε εχθρού (φάντασμα) στο παιγνίδι. Αφού το παιγνίδι θα χωριστεί σε δύο μέρη τα οποία θα μεταφερθούν σε διαφορετικές μηχανές, θα πρέπει να μπορούν να επικοινωνούν με τρόπους που να καταλαβαίνει το ένα το άλλο.

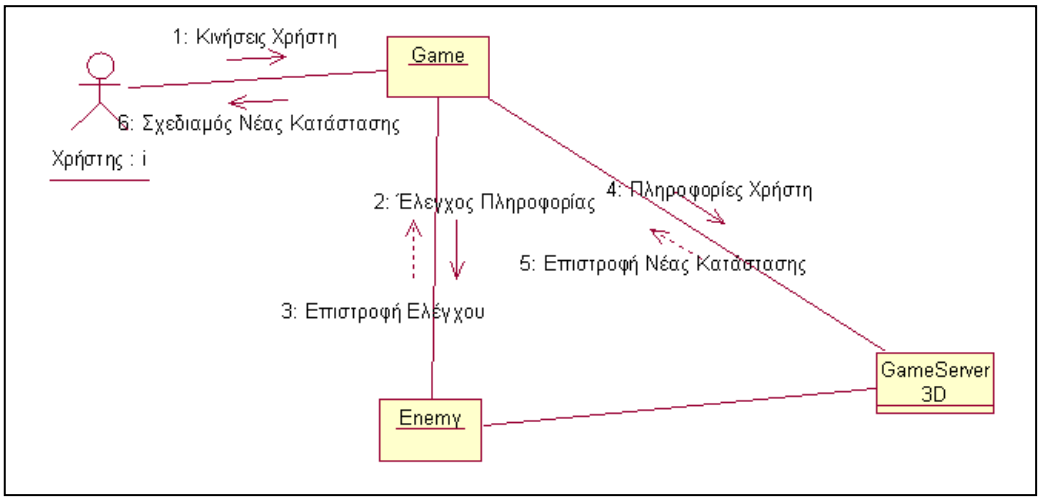
Στην συνέχεια δείχνουμε το sequence diagram του συστήματος παιγνιδιού. Στο σχεδιάγραμμα αυτό φαίνεται η ανταλλαγή μηνυμάτων μεταξύ των κλάσεων του παιγνιδιού. Ο χρήστης στέλλει την πληροφορία του στο παιγνίδι και αυτό αφού «ρωτήσει» όλους τους εχθρούς αν έχουνε χτυπηθεί (στείλει μήνυμα σε κάθε στιγμιότυπο της κλάσης Enemy για να ρωτήσει κατά πόσο ο χρήστης κτύπησε το φάντασμα ή όχι), τότε στέλλει την πληροφορία με την μορφή μηνύματος στο μέρος που υπολογίζει την νέα κατάσταση. Στην συνέχεια η νέα κατάσταση επιστρέφεται πίσω και χρησιμοποιείται για να σχεδιαστεί στην οθόνη η νέα κατάσταση. Εδώ παρόλο ότι το σύστημα επικοινωνεί με μηνύματα το παιγνίδι δεν βρίσκεται σε άλλη μηχανή αλλά όλο το σύστημα βρίσκεται στην ΄δια μηχανή. Δεν γίνεται η χρήση της κατανεμημένης πληροφορίας δηλαδή.



Σχήμα 5.4 : Η πορεία της πληροφορίας μεταξύ του χρήστη και του παιγνιδιού

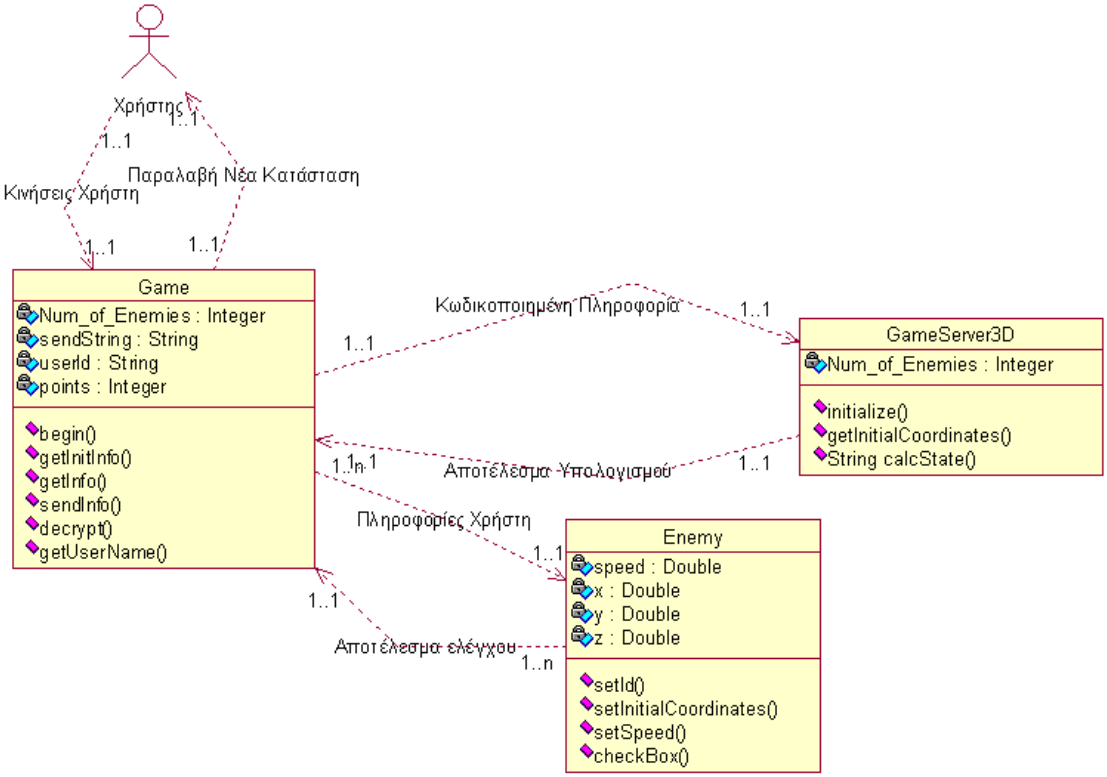
Το παιγνίδι (Game) παίρνει τις κινήσεις από τον χρήστη και αφού ελέγξει κατά πόσο κάποιο φάντασμα έχει χτυπηθεί από τον χρήστη, τότε αποστέλλει τις πληροφορίες αυτές στο μέρος που υπολογίζει την νέα κατάσταση (GameServer3D). Στην συνέχεια αφού υπολογιστεί η νέα κατάσταση επιστέφεται στο παιγνίδι όπου η πληροφορία αυτή χρησιμοποιείται για να σχεδιαστεί στην οθόνη η νέα κατάσταση.

Στην συνέχεια βλέπουμε και το collaboration diagram του συστήματος παιγνιδιού. Σε αυτό βλέπουμε την σχέση μεταξύ των κλάσεων του παιγνιδιού κατά την επικοινωνία από την στιγμή που κάποιος χρήστης δώσει μια κίνηση στο σύστημα και το σύστημα αφού την επεξεργαστεί επιστρέφει πίσω στον χρήστη σχεδιασμένη την νέα κατάσταση



Σχήμα 5.4 : Η ανταλλαγή μηνύματος του χρήστη σε σχέση παιγνίδι

Στην συνέχεια βλέπουμε το class diagram το οποίο δείχνει την σχέση μεταξύ των κλάσεων του παιγνιδιού. Εδώ βλέπουμε την επικοινωνία μεταξύ του χρήστη και του συστήματος παιγνιδιού χωρίς την χρήσης της κατανεμημένης πληροφορίας.

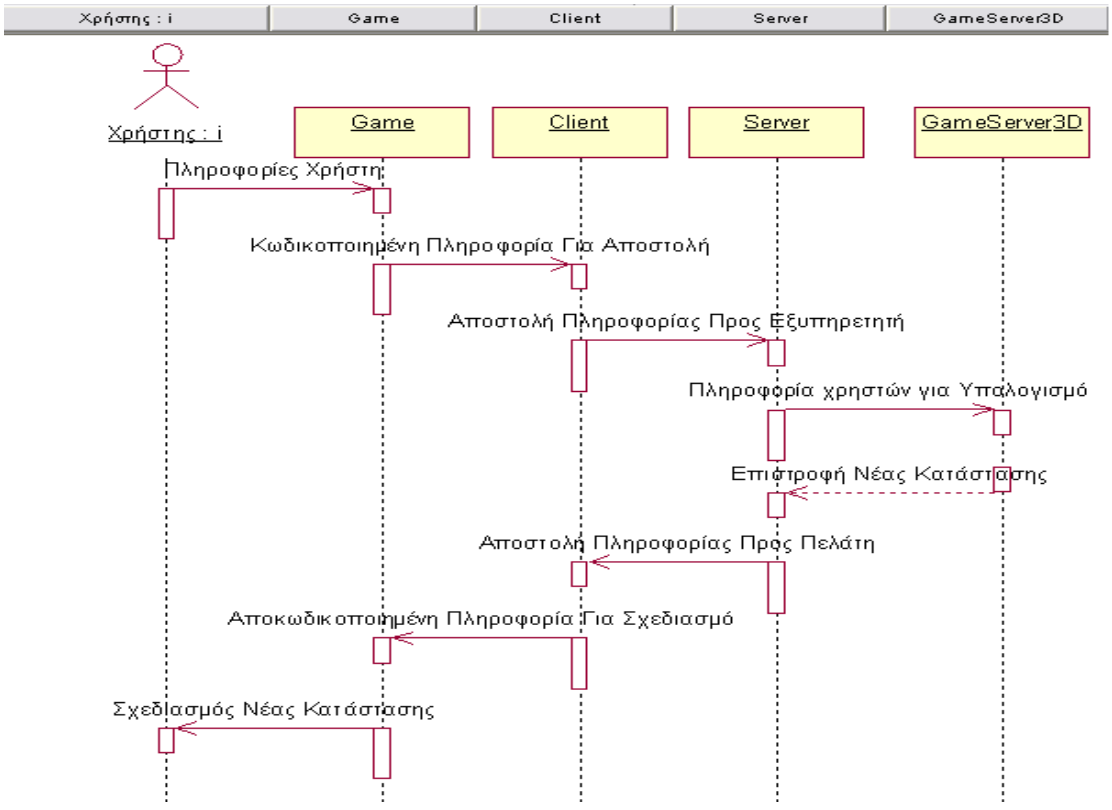


Σχήμα 5.6 : Το class diagram το οποίο δείχνει την σχέση μεταξύ των κλάσεων του συστήματος παιγνιδιού και του χρήστη.

5.3.3 Σχεδίαση ολόκληρου του συστήματος

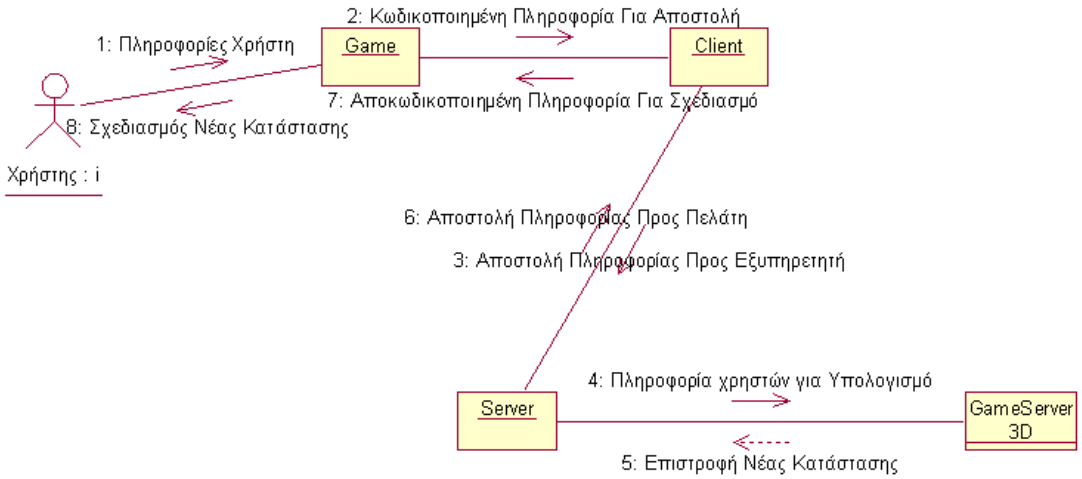
Στην συνέχεια θα δούμε πως τα δύο μέρη, σύστημα επικοινωνίας και σύστημα παιγνιδιού ενώνονται μαζί. Το παιγνίδι όπως έχουμε δει είναι χωρισμένο σε δύο κύρια μέρη, το μέρος που υπολογίζει την νέα κατάσταση, και το μέρος που επικοινωνεί με τον χρήστη. Αφού τα δύο μέρη επικοινωνούν μεταξύ τους με μηνύματα, και αφού το σύστημα επικοινωνίας μας επιτρέπει την επικοινωνία μεταξύ του εξυπηρετητή και του πελάτη, τότε το μόνο που χρειάζεται να κάνουμε είναι να τοποθετήσουμε το μέρος του παιγνιδιού που υπολογίζει την νέα κατάσταση στον εξυπηρετητή, και το μέρος που επικοινωνεί με τον χρήστη στην κάθε μηχανή που τρέχει ένας πελάτης. Με αυτό τον τρόπο τα δύο συστήματα ενώνονται σε ένα για να πάρουμε το τελικό.

Πάρακατω δείχνουμε το sequence diagram του ολοκληρωμένου συστήματος για να δούμε την ανταλλαγή μηνύματος μεταξύ του χρήστη και του ολοκληρωμένου συστήματος.



Σχήμα 5.7 : Το sequence diagram του ολικού συστήματος δείχνει την πορεία των μηνυμάτων από τον χρήστη εώς το εξυπηρετητή και πίσω

Στην συνέχεια βλέπουμε το collaboration diagram του ολικού συστήματος το οποίο δείχνει την σχέση μεταξύ των διαφόρων μερών του συστήματος κατά την ανταλλαγή μηνυμάτων με τον χρήστη.



Σχήμα 5.8 : Το collaboration diagram για ολόκληρο το σύστημα

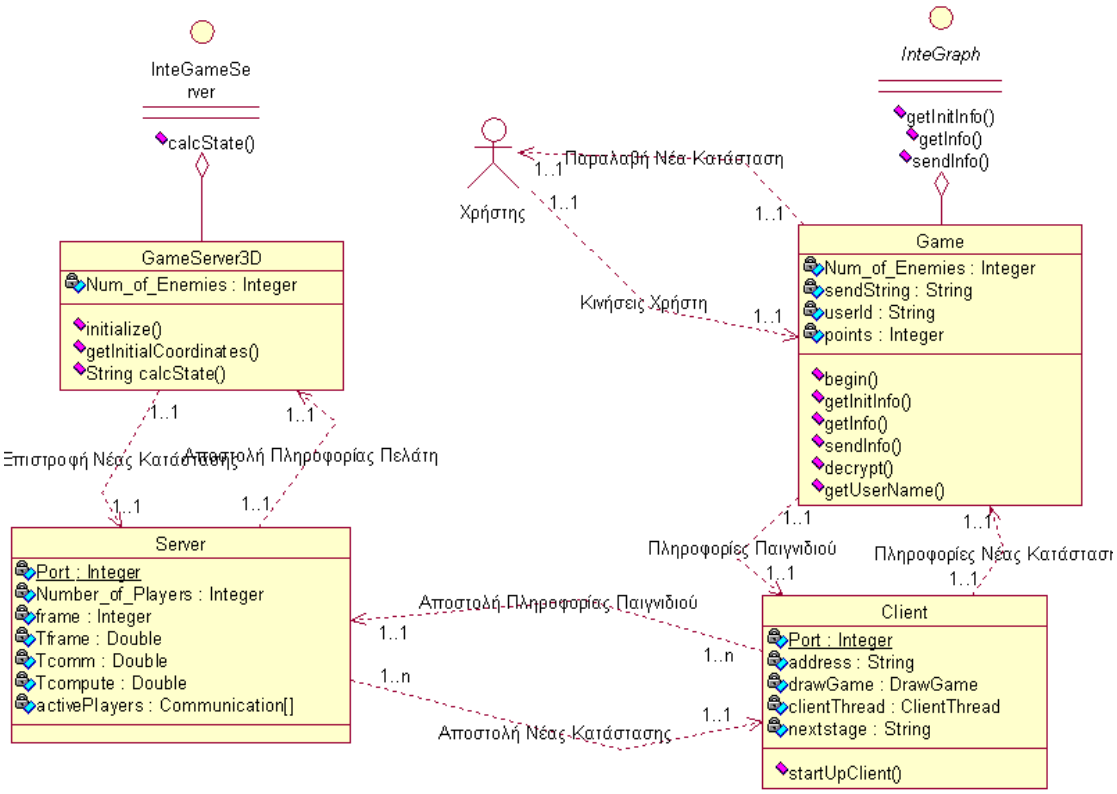
Αφού έχουμε δει το class diagram των δύο συστημάτων τώρα θα δούμε το class diagram του ενιαίου συστήματος. Ο τρόπος με τον οποίο θα ενωθούν οι κλάσεις μας είναι ο εξής.

Στον εξυπηρετητή στο σύστημα παιγνιδιού θα καλείται συνάρτηση από το σύστημα επικοινωνίας το οποίο θα περνά τις πληροφορίες που ελήφθησαν από τον πελάτη για υπολογισμό. Η συνάρτηση αυτή θα επιστρέφει πίσω την νέα κατάσταση του παιγνιδιού την οποία το σύστημα επικοινωνίας θα χρησιμοποιεί για να την στείλει πίσω στους πελάτες.

Στο πελάτη στο σύστημα παιγνιδιού θα καλούνται δύο συναρτήσεις από τον σύστημα επικοινωνίας οι οποίες η μια θα στέλλει τις πληροφορίες του πελάτη πίσω στο σύστημα επικοινωνίας για αποστολή στον εξυπηρετητή, και η δεύτερη θα περνά σαν παράμετρο στο παιγνίδι την νέα κατάσταση στο παιγνίδι για σχεδιασμό στην οθόνη.

Για να πετύχουμε το πιο πάνω στην υλοποίηση του παιγνιδιού χρησιμοποιούμε δύο Interfaces (ένα για το παιγνίδι του πελάτη και ένα για το παιγνίδι του εξυπηρετητή), τα οποία θα αποτελούν κατευθυντήριες γραμμές για να χρησιμοποιήσει κάποιος αυτές τις κλάσεις.

Πάρακάτω βλέπουμε το class diagram ολόκληρου του συστήματος χωρίς κάποιες επιπλέον λεπτομέρειες. Φαίνεται η σχέση μεταξύ των κλάσεων του ολοκληρωμένου συστήματος και επιπλέον βλέπουμε τα δύο interfaces τα οποία αποτελούν κατευθυντήριες γραμμές για κάποιο για να χρησιμοποιήσει το σύστημα επικοινωνίας.



Σχήμα 5.9 : Το class diagram του ολοκληρωμένου συστήματος

Κεφάλαιο 6

Υλοποίηση

6.1 Περιγραφή Κεφαλαίου

Στο κεφάλαιο υλοποίησης θα παρουσιάσουμε το τελικό σύστημα το οποίο υλοποιήσαμε. Θα δείξουμε διάφορα screenshot του παιγνιδιού μας για να εξηγήσουμε τον τρόπο που δουλεύει. Επιπλέον θα εξηγήσουμε τί χρειάζεται για να παιχτεί το παιγνίδι από διάφορους χρήστες στο διαδίκτυο. Αυτό περιλαμβάνει τι χρειαζόμαστε από πλευράς του εξυπηρετητή, και τί χρειαζόμαστε από πλευράς πελάτη.

6.2 Αναγκαία Εργαλεία

Εδώ θα περιγράψουμε τα εργαλεία που χρειάζεται ο εξυπηρετητής αλλά και ο χρήστης για να μπορεί να παίξει το παιγνίδι, καθώς και τί σύστημα πρέπει να έχει ο καθένας από αυτούς.

Ο εξυπηρετητής θα τρέχει το μέρος του κώδικα το οποίο αφού ακούσει από όλους τους χρήστες την πληροφορία για την κίνησή τους, τότε υπολογίζει την νέα κατάσταση. Ο εξυπηρετητής δεν χρειάζεται δηλαδή να τρέχει κάποιο συγκεκριμένο γραφικό περιβάλλον απλά να είναι συνδεδεμένος με το διαδίκτυο και να έχει μια διεύθυνση σε αυτό. Πρέπει απαραίτητα σε αυτόν να τρέχει εξυπηρετητής ιστοσελιδών (web server), για να μπορεί να εξυπηρετεί τους πελάτες που έρχονται για να φορτώσουν το παιγνίδι. Απαραίτητη προϋπόθεση είναι η ιστοσελίδα με το παιγνίδι (το οποίο είναι Applet), να βρίσκεται στην ίδια μηχανή με το πρόγραμμα που υπολογίζει την νέα κατάσταση, λόγο κάποιων περιορισμών των οποίων έχει η Java και αφορούν την

ασφάλεια του χρήστη στο διαδίκτυο. Επιπλέον στην μηχανή αυτή θα πρέπει να υπάρχει ένα ανοιχτό port το οποίο θα επιτρέπει την επικοινωνία με τον διερμηνέα της Java.

Το λειτουργικό σύστημα του υπολογιστή δεν έχει σημασία, αφού η Java τρέχει σε οποιοιδήποτε σύστημα χωρίς πρόβλημα, δεδομένου φυσικά ότι υπάρχει έκδοση Java για αυτό το σύστημα. Η έκδοση Java που χρειάζεται να είναι εγκατεστημένη είναι η J2SE 1.4.2 SDK ή νεότερη, η οποία αποτελεί το βασικό πακέτο για ανάπτυξη προγραμμάτων Java. Επιπλέον χρειάζεται να έχει εγκατεστημένη την έκδοση Java 3D 1.3.1 API ή νεότερη, ώστε να μπορεί να μεταγλωττιστεί το παιγνίδι το οποίο θα τοποθετηθεί στο κατάλληλο φάκελο ώστε να μπορεί κάποιος χρήστης να το ανοίξει με τον φυλλομετρητή (browser) του.

Η υπολογιστική ισχύς του υπολογιστή είναι ανάλογη με τον αριθμό χρηστών που θα εξυπηρετεί το σύστημά μας, και από το είδος του παιγνιδιού. Ενδεικτικά ένας σημερινός επιτραπέζιος υπολογιστής έχει αρκετή ισχύ για να τρέχει οποιαδήποτε λογική εφαρμογή παιγνιδιού θέλουμε.

Αφού το πρόγραμμα μεταγλωττιστεί και ξεκινήσει στον εξυπηρετητή θα χρησιμοποιεί το πιο πάνω port και θα περιμένει να ακούσει αιτήσεις από πελάτες για εισέλθουν στο σύστημα.

Ο πελάτης θα τρέχει το κομμάτι το οποίο είναι υπεύθυνο για να επικοινωνεί με τον χρήστη αλλά και την μηχανή που τρέχει ο εξυπηρετητής. Ο υπολογιστής του χρήστη χρειάζεται να είναι συνδεδεμένος με το διαδίκτυο καθ' όλη την διάρκεια του παιγνιδιού. Πρέπει να έχει φυλλομετρητή (browser) ο οποίος να μπορεί να τρέχει applets. Για να το πετύχει αυτό πρέπει να έχει εγκατεστημένη έκδοση του διερμηνέα της Java. Και εδώ δε μας ενοχλεί τί λειτουργικό σύστημα έχει ο πελάτης φτάνει να υπάρχει έκδοση Java για αυτό.

Αφού το παιγνίδι χρειάζεται τρισδιάστατα γραφικά τότε χρειάζεται επιπλέον από τον απλό διερμηνέα της Java να υπάρχει εγκατεστημένος στην μηχανή του χρήστη και ο διερμηνέα της Java 3D API. Και εδώ χρειάζεται η χρήση ανοικτού κάποιου port για τον διερμηνέα της Java ώστε να μπορεί να επικοινωνεί με το διαδίκτυο.

Η υπολογιστική ισχύς του υπολογιστή είναι ανάλογη με τον τύπο του παιγνιδιού που θα υλοποιηθεί. Ενδεικτικά και εδώ ένας σημερινός επιτραπέζιος υπολογιστής έχει αρκετή ισχύ για να τρέχει οποιαδήποτε λογική εφαρμογή παιγνιδιού θέλουμε. Επιπλέον όμως εδώ χρειάζεται ο υπολογιστής να έχει την ικανότητα να έχει εγκατεστημένο κατάλληλο υλικό για τρισδιάστατα γραφικά, δηλαδή κάρτα γραφικών.

Αφού ο χρήστη μπει στο διαδίκτυο το μόνο που έχει να κάνει είναι να ανοίξει το φυλλομετρητή του και να πάει στην ηλεκτρονική διεύθυνση του παιγνιδιού και να κάνει αίτηση για να εισέλθει στο παιγνίδι. Αν του παραχωρηθεί η άδεια από τον εξυπηρετητή τότε το παιγνίδι ξεκινάει.

6.3 Υλοποίηση

Εδώ αφού έχει υλοποιηθεί το σύστημα θα δούμε μια περιγραφή του και θα εξηγήσουμε τα βασικά του μέρη. Επιπλέον θα δείξουμε και μερικά screenshots του προγράμματος.

Το μέρος του εξυπηρετητή τρέχει στη μηχανή του εξυπηρετητή με όνομα astarti. Εκεί αφού ξεκινήσει από εμάς στο παρασκήνιο (background) τρέχει περιμένοντας να ακούσει από το port 8642 από πελάτες. Καθώς τρέχει χωρίς να εξυπηρετεί πελάτες δεν χρησιμοποιεί σχεδόν καθόλου πόρους αφού απλά περιμένει να ακούσει από πελάτες και βρίσκεται σε I/O κατάσταση. Ο εξυπηρετείς astarti τρέχει λειτουργικό σύστημα Linux με εγκατεστημένη την έκδοση Java SDK 1.4.2_02.

Το μέρος εξυπηρετητή του προγράμματος μας αποτελείται από τρία αρχεία Java. Το GameServer.java το οποίο περιέχει τον κώδικα του μέρους επικοινωνία του εξυπηρετητή (βλέπε Παράρτημα Α), το InteGameServer.java (βλέπε Παράρτημα Β), το οποίο είναι το interface περιέχει τον κώδικα για την υλοποίησης των κλάσεων παιγνιδιού και άρα αποτελεί κατευθυντήριες γραμμές για να υλοποιηθεί το μέρος παιγνίδι του εξυπηρετητή, και το GameServer3D.java (βλέπε Παράρτημα Β), το οποίο περιέχει τον κώδικα του μέρους παιγνιδιού του εξυπηρετητή.

Αφού το πρόγραμμα μεταγλωττιστεί με την εντολή “javac GameServer3D.java”, τότε ξεκινούμε το πρόγραμμα με την εντολή “nohup java GameServer3D &”. Στην συνέχεια το πρόγραμμα αφού δώσει το μήνυμα στην οθόνη “Server Started”, το οποίο σημαίνει ότι ο εξυπηρετητής έχει ξεκινήσει χωρίς προβλήματα, ακούει πάνω στο πιο πάνω port για αιτήσεις από πελάτες να εισέλθουν στο παιγνίδι.

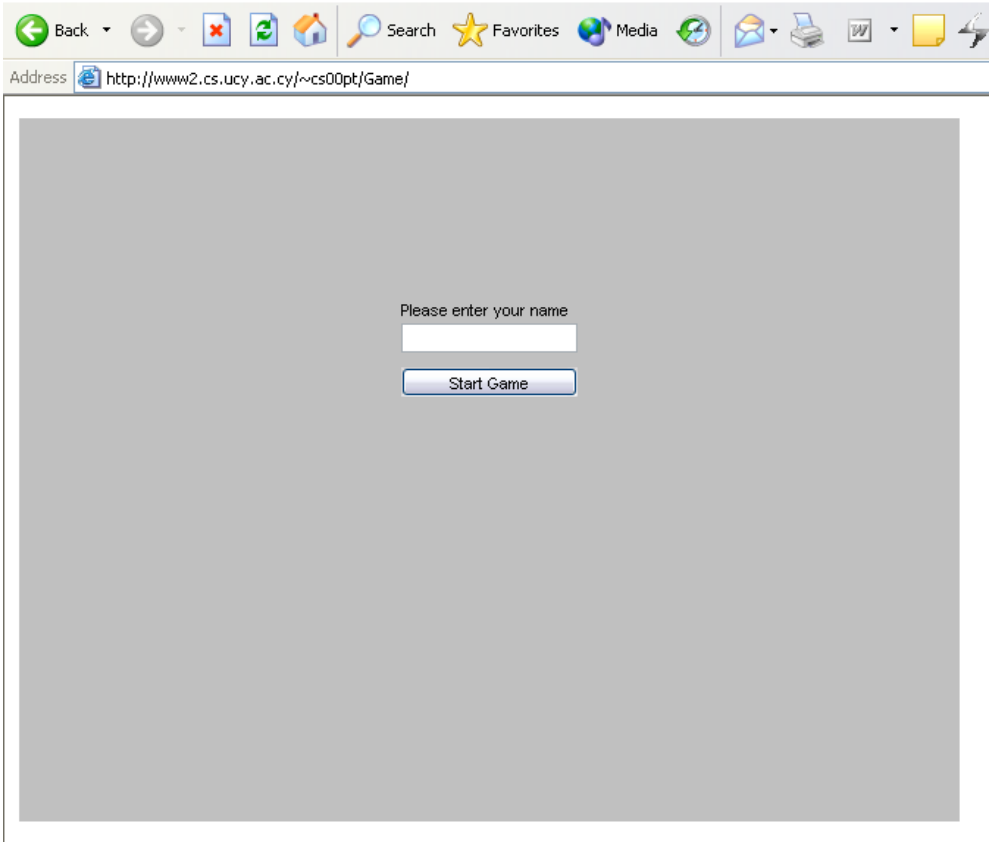
Το μέρος του πελάτη τώρα βρίσκεται στον δικτυακό τόπο www.cs.ucy.ac.cy/~cs00pt/Game. Ο εξυπηρετητής σελίδων (Web Server) βρίσκεται επίσης στην μηχανή astarti, δηλαδή στην ίδια μηχανή που τρέχει το μέρος του εξυπηρετητή. Το πρόγραμμα αποτελείται από το Client.java (βλέπε Παράρτημα Α), το οποίο αποτελεί το μέρος επικοινωνία του πελάτη, το InteGraph.java (βλέπε Παράρτημα Β), το οποίο είναι το interface το οποίο αποτελεί κατευθυντήριες γραμμές για να

υλοποιηθεί το μέρος παιγνίδι του πελάτη, καθώς και το Game.java και το Enemy.java (βλέπε Παράρτημα Β), το οποίο αποτελεί μέρος του παιγνιδιού του πελάτη.

Το πρόγραμμα μεταγλωττίστηκε σε άλλη μηχανή αφού ο astarti δεν έχει την Java 3D API 1.3.1 ή νεότερη εγκατεστημένη σε αυτό. Η εντολή για να μεταγλωττιστεί είναι "javac Game.java". Αφού το πρόγραμμα μεταγλωττίστηκε μεταφέρθηκε στον ποιο πάνω δικτυακό τόπο όπου είναι διαθέσιμο για τους χρήστες. Αν κάποιος με οποιοδήποτε λειτουργικό σύστημα στο οποίο υπάρχει εγκατεστημένη η Java και η Java 3D API επισκεφθεί την ποιο πάνω ιστοσελίδα θα είναι σε θέση να παίξει το παιγνίδι.

6.4 Παιγνίδι

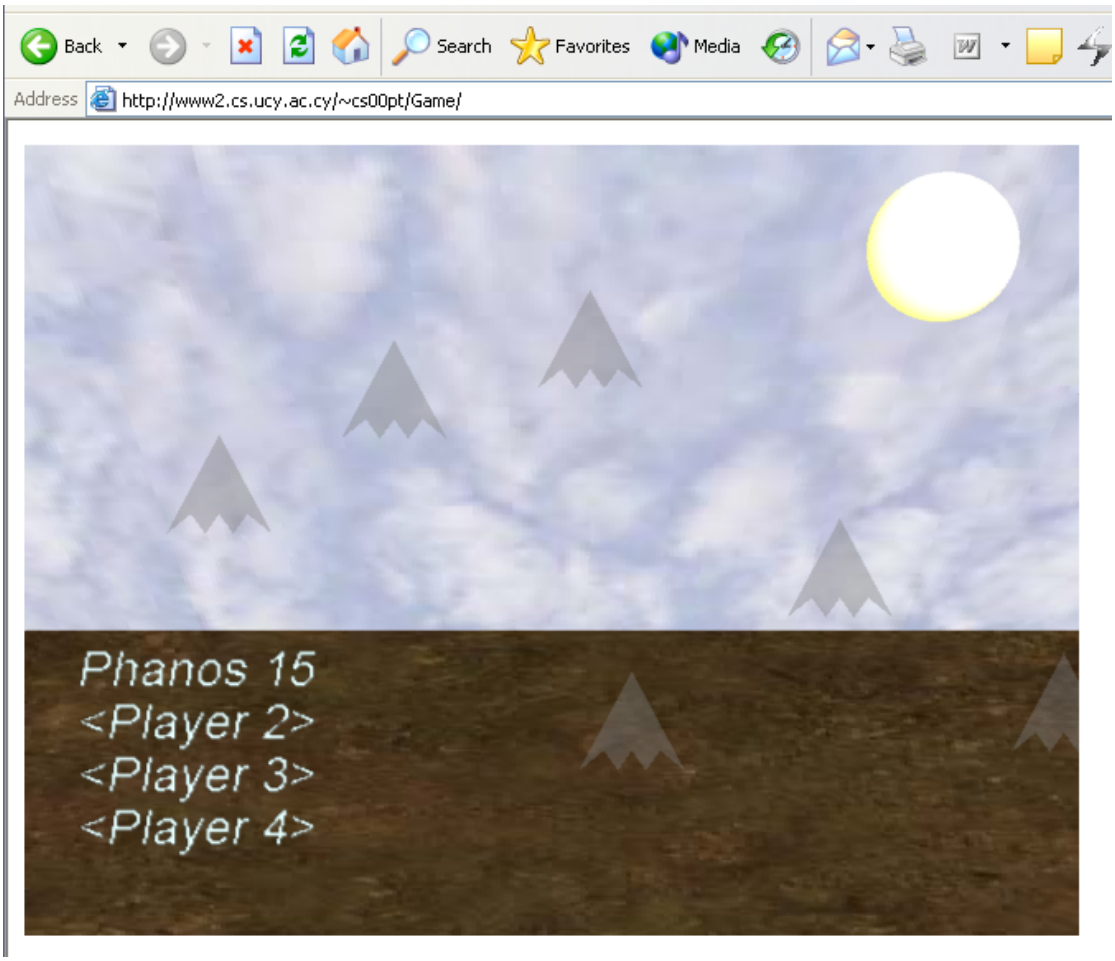
Το παιγνίδι έχει ξεκινήσει στο εξυπηρετητή και στην συνέχεια εμείς μέσω ενός φυλλομετρητή (browser) επισκεφτόμαστε την πιο πάνω ιστοσελίδα. Η ιστοσελίδα μας δείχνει την εικόνα στο σχήμα 6.1. Στην εικόνα αυτή μας ζητείται να δώσουμε ένα όνομα το οποίο θα χρησιμοποιείται για να ξεχωρίζει ο κάθε πελάτης στο παιγνίδι. Ο κάθε πελάτης ξεχωρίζει στο παιγνίδι από το όνομα που δίνει και από ένα μοναδικό αριθμό τον οποίο παίρνει στην αρχή του παιγνιδιού. Σαν ο μοναδικός αριθμός του κάθε χρήστη επιλέγεται ο μοναδικός αριθμός ip της μηχανής του χρήστη από τον οποίο ξεκινάει το παιγνίδι.



Σχήμα 6.1 : Στο σχήμα φαίνεται η αρχική σελίδα του παιγνιδιού.

Στην συνέχεια αφού ο χρήστης δώσει ένα όνομα και πατήσει το κουμπί “Start Game” τότε ενεργοποιείται το παιγνίδι. Σε αυτή την φάση στέλνεται μήνυμα προς τον εξυπηρετητή για να εισέλθει ο χρήστης στο παιγνίδι. Μόλις γίνει δεκτή η αίτηση του παίχτη, τότε το παιγνίδι ξεκινάει. Ο πελάτης σχεδιάζει στην οθόνη κάθε φορά τα φαντάσματα (εχθροί) τα οποία στέλλει ο εξυπηρετητής και στέλλει πίσω τις πιθανές κινήσεις του πελάτη στο εξυπηρετητή.

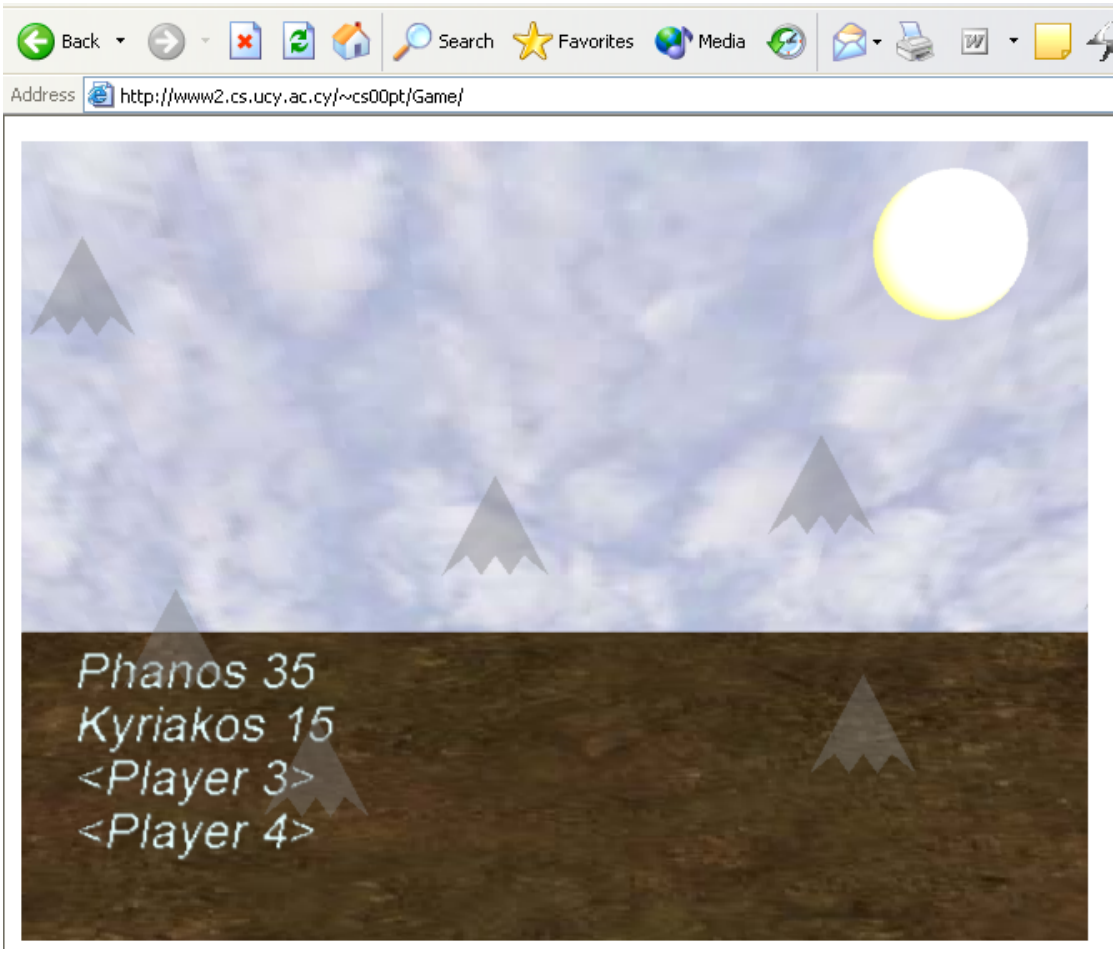
Όταν το παιγνίδι ξεκινήσει βλέπουμε την εικόνα στο σχήμα 6.2. Εδώ βλέπουμε μια εικόνα του παιγνιδιού με ένα ενεργό πελάτη καθώς και τους βαθμούς που έχει κερδίσει μέχρι στιγμής.



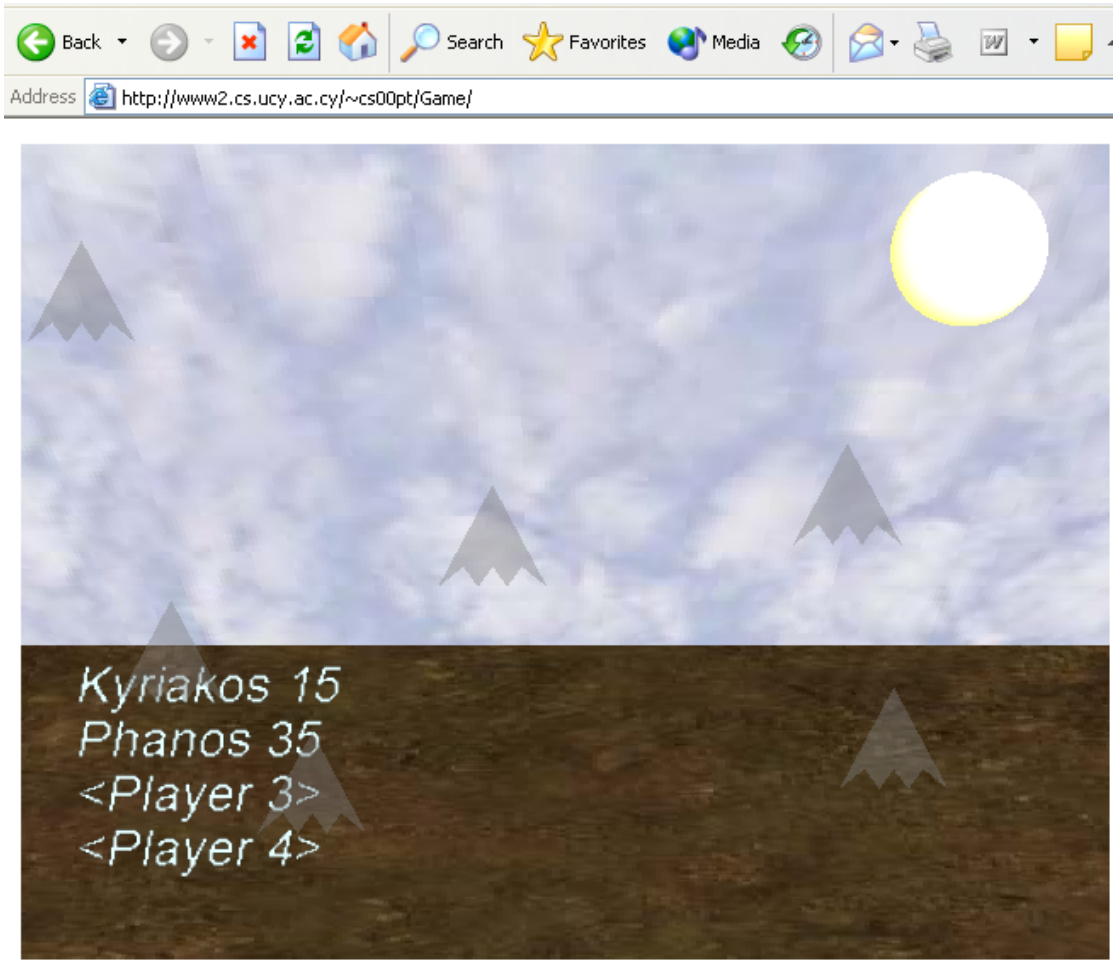
Σχήμα 6.2 : Το παιγνίδι με ένα ενεργό παίχτη

Αν στο παιγνίδι έρθει ακόμα ένα παίχτης τότε βλέπουμε την εικόνα στο σχήμα 6.3 αλλά και στο 6.4. Εδώ οι δύο παίκτες συναγωνίζονται ποιος θα χτυπήσει τα φαντάσματα και πόσο γρήγορα θα το κάνει ώστε να πάρει όσο το δυνατό περισσότερους βαθμούς. Παρατηρούμε ότι και τα δύο σχήματα τα οποία έχουν παρθεί από διαφορετικές μηχανές την ίδια χρονική στιγμή δείχνουν το ίδιο γραφικό αποτέλεσμα στους χρήστες. Αυτό συμβαίνει γιατί ο εξυπηρετητής στέλνει την ίδια κατάσταση σε όλους τους χρήστες κάθε φορά. Στο μόνο που διαφέρουν τα δύο σχήματα είναι στο σημείο των ονομάτων. Την δεδομένη στιγμή που ήρθε ο δεύτερος χρήστης στο σύστημα ο πρώτος δεν είχε προφανώς στείλει μήνυμα στο εξυπηρετητή με αποτέλεσμα ο εξυπηρετητής να μην τον είχε περιλάβει σε αυτή την ανταλλαγή μηνυμάτων. Αυτό είχε σαν αποτέλεσμα στο σχήμα 6.4 να φαίνεται πρώτος ο δεύτερος χρήστης που ήρθε στο σύστημα αφού ο εξυπηρετητής έστειλε μόνο το όνομα του δεύτερου χρήστη σε κάθε πελάτη. Άρα ο κάθε πελάτης πρόσθεσε το δεύτερο όνομα, ο

πρώτος στην δεύτερη θέση αφού έχει είδη προσθέσει το δικό του στην πρώτη θέση, και ο δεύτερος στην πρώτη θέση ο οποίος δεν έχει ακόμα κάποιο όνομα στην οθόνη του. Για αυτό το σχήμα 6.4 το οποίο πάρθηκε από την μηχανή του δεύτερου χρήστη δείχνει τον δεύτερο χρήστη πρώτο. Στην συνέχεια που στάλθηκε το όνομα του πρώτου τότε προστέθηκε το όνομα του πρώτου χρήστη σαν δεύτερο στην μηχανή του δεύτερου χρήστη.



Σχήμα 6.3 : Το παιγνίδι με δύο ενεργούς παίκτες από την μηχανή του χρήστη Phanos



Σχήμα 6.4 : Το παιγνίδι με δύο ενεργούς παίκτες από την μηχανή του δεύτερου χρήστη

Κεφάλαιο 7

Συμπεράσματα

7.1 Περιγραφή κεφαλαίου

Στο κεφάλαιο αυτό δίνουμε τα συμπεράσματά μας από την διπλωματική αυτή καθώς πιθανές βελτιώσεις που μπορούν να γίνουν στο σύστημα μας. Επιπλέον αξιολογούμε τα αποτελέσματα μας ανάλογα με τους στόχους που είχαμε βάλει αρχικά.

7.2 Γενικές εντυπώσεις

Στην διπλωματική αυτή χρησιμοποιήθηκε η άλγεβρα διεργασιών ACSR για περιγραφή και σχεδίαση του συστήματος. Αυτό αποδείχθηκε ιδιαίτερα βοηθητικό στην υλοποίηση του συστήματος μας, αφού η ACSR μας επέτρεψε να δημιουργήσουμε τις κλάσεις με τέτοιο τρόπο ώστε να γίνεται σωστά ο συγχρονισμός μεταξύ των εμπλεκομένων μερών, πριν την υλοποίηση του συστήματος.

Επιπλέον στην πορεία αποφασίστηκε το παιγνίδι που θα χρησιμοποιούσαμε για να ελέγξουμε το σύστημα μας, αφού στην πορεία αποφασίστηκε ότι θα ήταν καλύτερα να υλοποιήσουμε ένα σύστημα το οποίο να μπορεί να δέχεται χωρίς ιδιαίτερες προσθήκες οτιδήποτε παιγνίδι πληροί τις προδιαγραφές που είχαμε θέσει.

Υλοποιήσαμε δηλαδή αρχικά το σύστημα στην ACSR και στην συνέχεια μεταφράσθηκε στην Java. Μετά αφού ελέγξαμε ότι δούλευε σωστά προσθέσαμε σε αυτό το παιγνίδι το οποίο επιλέχθηκε μετά.

Υλοποιήσαμε δηλαδή όλους τους στόχους που είχαμε θέσει κατά καιρούς με τον καλύτερο δυνατό τρόπο που μπορούσαμε ώστε να έχουμε στο τέλος ένα λειτουργικό παιγνίδι το οποίο πληροί τις προδιαγραφές μας.

7.3 Αξιολόγηση

Το σύστημα το οποίο έχει τελειώσει βρίσκεται στο δικτυακό τόπο που έχουμε αναφέρει στο Κεφάλαιο 6 και μέχρι στιγμής δουλεύει ικανοποιητικά. Μπορεί να εξυπηρετεί πελάτες που επιθυμούν να παίξουν το παιγνίδι, σωστά και αξιόπιστα και εντός των προδιαγραφών μας.

Το παιγνίδι είναι ικανό ακόμα και σε περίπτωση που έρθει κάποιος χρήστης ο οποίος δεν πληροί τις προδιαγραφές, ή ακόμα προσπαθήσει κακόβουλα να επηρεάσει το παιγνίδι, να χειριστεί την κατάσταση ανάλογα. με τέτοιο τρόπο ώστε να μην επηρεάζεται το παιγνίδι με τους άλλους χρήστες.

Το κατώτατο επίπεδο πλαισίων (frames) που παράγεται σε μια μηχανή, δεδομένου ότι η μηχανή του χρήστη έχει τις αναγκαίες προδιαγραφές, είναι 25. Το παιγνίδι έχει ελεγχθεί σε συστήματα τα οποία βρίσκονται σε αρκετή απόσταση από τον τόπο του παιγνιδιού, π.χ. από άλλες πόλεις, και ο αριθμός των frames ανά δευτερόλεπτο ήταν μεγαλύτερος από 25.

Ο εξυπηρετητής έχει βρεθεί να είναι ιδιαίτερα ανεκτικός σε λάθη που μπορούν να γίνουν είτε στην επικοινωνία με κάποιο πελάτη, είτε στην απότομη αποχώρηση ενός πελάτης από το σύστημα ώστε να μην δημιουργούνται αδιέξοδα και το παιγνίδι να μην μπορεί να συνεχιστεί.

Παρόλα αυτά το σύστημα το οποίο φαίνεται να είναι ιδιαίτερα αποδοτικό σε συστήματα τα οποία βρίσκονται σε μακρινές αποστάσεις από τον εξυπηρετητή, κάνοντας έτσι δυνατό περισσότεροι χρήστες να μπορούν να ανταγωνισθούν μεταξύ τους, στην περίπτωση όπου το παιγνίδι παίζεται από πολλούς χρήστες οι οποίοι βρίσκονται σε τοπικό δίκτυο (π.χ. LAN), το σύστημα χάνει στην απόδοση των frames από τον εξυπηρετητή, ιδιαίτερα σε ώρες που στο τοπικό δίκτυο υπάρχουν πολλοί χρήστες. Αυτό ίσως να συμβαίνει λόγο του ότι το σύστημα στην μηχανή κάθε πελάτη αφού πρέπει να στέλλει και να λαμβάνει πακέτα συνεχώς με τον εξυπηρετητή, προκαλεί έτσι ίσως μια πιθανή συμφόρηση στο δίκτυο με αποτέλεσμα μια σημαντική μείωση στην απόδοση του συστήματος μας. Αυτό έχει βρεθεί να συμβαίνει μόνο σε ένα εργαστήριο της πληροφορικής το οποίο παρόλο ότι βρισκόταν πολύ κοντά στο

εξυπηρετητή η απόδοση ήταν πολύ μειωμένη σε σχέση με άλλους υπολογιστές που βρίσκονται ποιο μακριά. Συγκεκριμένα από το συγκεκριμένο εργαστήριο όπου υπήρχε gigabit σύνδεση με τον εξυπηρετητή, η απόδοση μόλις που έφτανε τα 25 frames το δευτερόλεπτο, ενώ από τις εστίες του Πανεπιστήμιου, η απόδοση έφτανε μέχρι 200 frames το δευτερόλεπτο. Ακόμα και από την Λάρνακα η οποία έχει απόσταση από τον εξυπηρετητή αρκετά χιλιόμετρα η απόδοση έφτανε τα 50 με 60 frames το δευτερόλεπτο. Συγκεκριμένα το πρόβλημα αυτό έχει βρεθεί να συμβαίνει μόνο στο εργαστήριο αυτό.

7.4 Συμπεράσματα

Η ACSR αποτελεί ένα πολύ χρήσιμο εργαλείο το οποίο μας βοηθά να μελετήσουμε και να αναλύσουμε μεγάλα συστήματα εύκολα και χωρίς μεγάλες δαπάνες σε χρόνο ή κόστος. Αναλύοντας ένα σύστημα στην ACSR βλέπουμε τον τρόπο με τον οποίο γίνεται η επικοινωνία και ο συγχρονισμός μεταξύ των υπομέρων, ώστε να κατανοήσουμε το σύστημα μας καλύτερα στα αρχικά στάδια της μελέτης, αλλά και με την βοήθεια του προγράμματος Versa μπορούμε να βρούμε πολλά από τα προβλήματα που θα είχαμε κατά την υλοποίηση έγκαιρα και πριν να γίνουν.

Η Java αποτελεί ένα καταπληκτικό εργαλείο το οποίο μας βοηθά να αναπτύξουμε πολύ μεγάλα προγράμματα πολύ εύκολα και γρήγορα, προσφέροντας ταυτόχρονα μεγάλη ευρωστία και μικρότερη πιθανότητα λάθους στην υλοποίηση μας με την επαναχρησιμοποίηση κώδικα. Το αντίκτυπο σε αυτό είναι η μειωμένη απόδοση σε ένα πρόγραμμα το οποίο γραμμένο σε μια άλλη γλώσσα θα ήταν, πολλές φορές, σημαντικότερα ποιο αποδοτικό.

Το σύστημα το οποίο έχει αναπτυχθεί με την χρήση της ACSR και την Java πληροί όλες τις προδιαγραφές οι οποίες έχουν αρχικά τεθεί δείχνοντας έτσι ότι η συνεργασία των δύο ποιο πάνω εργαλείων βοήθησε, το κάθε εργαλείο με τον δικό του τρόπο, με τον μέγιστο δυνατό τρόπο στην σωστή ανάπτυξη του συστήματος. Άρα και τα δύο μαζί όταν χρησιμοποιηθούν σωστά, προσφέρουν ένα σωστό τρόπο με το οποίο μπορούμε να αναπτύξουμε μεγάλα συστήματα τα οποία αρχικά μπορούν να φαίνονται δύσκολα στην κατανόηση του τρόπου που λειτουργούν, αλλά και του τρόπου που μπορούν να υλοποιηθούν.

Βιβλιογραφία

[1] Bruce Eckel. *Thinking in Java*. Prentice Hall Upper Saddle River, New jersey 07458, Second Edition, 2000

[2] Richard Gerber. *A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems*. University of Pennsylvania Philadelphia, PA 19104-6389, 1993

[3] Nick Foster. *ACSR GameNet*. CIS642 Real-Time System Project, 1996

[4] Robin Milner. *Communication and Concurrency*. Prentice-Hall International, 1989

[5] Dennis J Bouvier. *Getting Started with the Java 3D API.* Sun Microsystems Inc, tutorial v1.6.2, 1999-2001

[6] Duncan Clarke. *VERSA: Verification, Execution and Rewrite System for ACSR.* Department of Computer and Information Science University of Pennsylvania Philadelphia, PA 19104-6389, 1998

[7] Duncan Clarke. *VERSA: A Tool for the Specification and Analysis of Resource-Bound Real-Time Systems.* Department of Computer and Information Science University of Pennsylvania Philadelphia, PA 19104-6389, 1994

[8] Xiaotong Zhuang. *Building Interactive Distributes Applications – MultiPlayer Doom.* Project of CS 7001, 2002

[9] Ghita Kouadri Mostefaoui and Soraya Kouadri Mostefaoui. *Implementing a Distributed Multi-Player Blackjack Game Using the Java Shared Data Toolkit.* International Conference on Application and Development of Computer Games in the 21st Century, 2001

Παράρτημα Α

Κώδικας Συστήματος Επικοινωνίας

Στο παράρτημα Α δίνεται ο κώδικας του συστήματος επικοινωνίας. Το σύστημα επικοινωνίας αποτελείται από τον κώδικα του εξυπηρετητή και των κώδικα του πελάτη. Για τον κώδικα του εξυπηρετητή παραθέτουμε τον κώδικα που αποτελεί την υλοποίηση του συστήματος επικοινωνίας, καθώς και τον κώδικα για τον interface το οποίο αποτελεί οδηγό για την σχεδίαση και υλοποίηση του παιγνιδιού που θα χρησιμοποιήσουμε. Το ίδιο ισχύει και τον πελάτη. Έχουμε τον κώδικα που αποτελεί την υλοποίηση του συστήματος επικοινωνίας, καθώς και τον κώδικα ο οποίος αποτελεί οδηγό για την σχεδίαση και υλοποίηση του παιγνιδιού.

Α.1 Κώδικας Εξυπηρετητή

Α.1.1 Κώδικας συστήματος επικοινωνίας

```
// GameServer.java Finished at 01:16 15/4/2004
import java.io.*;
import java.net.*;
import java.util.*;
import java.sql.Time;

/** This class is created after the GameServer3D class and has
 *  the responsibility of communicating with every client
```

```
 *  and exhange information
 *  @author Phanos Pavlides
 *  @version 2.0
*/

public class GameServer{
   // Variable for the Port that the Server listens to
   private static final int PORT = 8642;
  // Variable for the number of Players the Server serves
   private int Num_of_Players = 4;
   // Variable for the minimun number of frames require
   private int frames = 25;
   // Variable for the maximun time a frame should last
   private double Tframe = 1000/frames;
   // Variable for the maximum time an exchange should last
   private double Tcomm = (3.0/4.0)*Tframe;
   // Variable for the diffence between Tframe and Tcomm
   private double Tcompute = Tframe - Tcomm;
   // Variable for the number of the active users
   private int ActiveUsers = 0;
   // Variable for the users finished in an exchange
   private int UserFinished = 0;
   // Variable for the output log file
   private PrintWriter outf;
   // Variable for the GameServer3D calculation class
   private GameServer3D calcnextstate;
   // Container to hold the clients that are active in the game
   private ArrayList ActivePlayers = new ArrayList();
   // Variable for the class that is responsiple to call
   // the calculation method
   private GameCompute nextstage = new GameCompute();

   /** Creates a GameServer object with the default values of
    *  Port = 8642
    */
   public GameServer(){
   }
   /** Creates a GameServer object with the default values of
    *  Port = 8642 and with reference to the the class GameServer3D
    *  that is responsible in calculating the next state of the game
    */
   public GameServer(GameServer3D calcnextstate){
         this.calcnextstate = calcnextstate;
   }
   /** Creates a GameServer object with the default values of
    *  Port = 8642 and with the given maximum number of players
    */
   public GameServer(int Num_of_Players){
         this.Num_of_Players = Num_of_Players;
```

```
}
/** Creates a GameServer object with the default values of
 *  Port = 8642, reference to the the class GameServer3D
 *  that is responsible in calculating the next state of the game,
 *  and with the given maximum number of players
 */
public GameServer(int Num_of_Players, int frames){
      this.Num_of_Players = Num_of_Players;
      this.frames = frames;
}

/** Returns the number if the active Players in the Game
 *  at the time this function is called
 *  @return the number of active users
 */
private int getActiveUsers(){
      return ActiveUsers;
}

/** This class responsible for the timeouts that occur in the Game.
 *  During a communication with a client a client info might take
 *  a lot of time to reach the server and there is the need continue
 *  the execuution without to wait the info to come
 */
public class ScopeObject extends TimerTask{
      private Communicate c;
      /** Creates a ScopeObject object with reference to the
       *  given class Communicate. The given class will be used
       *  for the timeout
       */
      public ScopeObject(Communicate c){
         this.c=c;
      }
      /** This method is run when the object is created and
       *  it runs while the Communicate class is running
       */
      public void run(){
         synchronized(nextstage){
               {
                  UserFinished++;
                  if(UserFinished == ActiveUsers)
                        nextstage.resumeGame();
                  c.timeout = true;
               }
         }
      }
}

/** This class is for communicating to each client that enters
```

```
 *  the game. It exhange information with the clients that
 *  it connects
 *  the responsibility of communicating with every client
 *  and exhange information
 */
public class Communicate extends Thread {
	// Variable to hold the socket
	private Socket socket;
	// Variable for the input stream
	private BufferedReader in;
	// Variable for the output stream
	private PrintWriter out;
	// Variable to hole the clients ip
	private String User_Ip;
	// Variable for the timeout
	private Timer timer;
	// Variable for the timeout
	private boolean timeout = false;
	// Variable to hold the input information from the client
	private String str = null;

	/** Creates a Communicate Thread that it exchange information
	 *  whith the client that connects to the Socket s
	 */
	public Communicate(Socket s)
	   throws Exception {
	   socket = s;
	   // input stream from the client
	   in =
	         new BufferedReader(
	           new InputStreamReader(
	                 socket.getInputStream()));
	   // output stream for the server to the client
	   out =
	         new PrintWriter(
	            new BufferedWriter(
	              new OutputStreamWriter(
	                    socket.getOutputStream())), true);
	   synchronized(nextstage){
	         ActiveUsers++;
	         start();
	   }
	}

	/** This method is run when the communication with a client
	 *  is achive. Its purpose is to exchange information
	 *  to the client that is connected to
	 */
	public void run() {
```

```
            int time_count = 0;
            int times = 0;
            boolean time = false;
            int max_Allowed = 10;
            try {
                // receive the initial information
                String msg = in.readLine();
                User_Ip = msg;
                System.out.println("User " + msg + " has entered the Game");
                msg = msg + "OK";
                out.println(msg);
                outf.println("User " + User_Ip + " " + socket + " has entered the game at
" + new Date());
                outf.close();
                nextstage.createFile();
                while (true) {
                   str = null;
                   timeout = false;
                   // set a maximum timeout in case
                   // the client has left the game
                   socket.setSoTimeout(10000);
                   // set a timer in case the client dosen't finish
                   // in a Tframe time
                   timer = new Timer();
                   timer.schedule(new ScopeObject(this),(int)Tcomm,(int)Tcomm);
                   // receive the information from the client
                   str = in.readLine();
                   // if the timer didn't go off cancel it
                   timer.cancel();
                   timer = null;
                   if(str==null)
                        times++;
                   else
                        times=0;
                   if(times == max_Allowed)
                        throw new SocketException();
                   if(time == false)
                        time_count = 0;
                   time = false;
                   // send the next stage to the client
                   out.println(nextstage.all);
                   synchronized(nextstage){
                        if(str!=null)
                           // of a client has quit then the Server thread
                           // should quit too
                           if(str.equals("QUIT")){
                                ActivePlayers.remove(ActivePlayers.indexOf(this));
                                timer.cancel();
                                timer = null;
```

```
                                ActiveUsers--;
                                if(UserFinished == ActiveUsers)
                                   nextstage.resumeGame();
                                break;
                             }
                          // if no timeout has occured
                          if(timeout == false)
                             {
                                UserFinished++;
                                // if all users has finished the resume
                                // the Server calculation class
                                if(UserFinished == ActiveUsers)
                                   nextstage.resumeGame();
                                // suspend the Thread until the next
                                // stage is calculated
                                try{
                                   nextstage.wait();
                                }catch(InterruptedException
e){System.out.println("Unable to pause Thread");
                                outf.println("Unable to pause Thread " + e.getMessage());
                                e.printStackTrace(outf);
                                outf.close();
                                nextstage.createFile();
                                }
                             }
                          else
                             {
                                timeout = false;
                             }
                       }
                    }
              } catch(Exception e) {
                    synchronized(nextstage){
                       System.out.println(this);
                       try{
                             ActivePlayers.remove(ActivePlayers.indexOf(this));
                             timer.cancel();
                             timer = null;
                       }catch(Exception ie){ie.printStackTrace();}
                       ActiveUsers--;
                       if(UserFinished == ActiveUsers)
                             nextstage.resumeGame();
                       System.err.println("IO Exception" + e.getMessage());
                       outf.println("IO Exception" + e.getMessage());
                       e.printStackTrace(outf);
                       e.printStackTrace();
                       outf.close();
                       nextstage.createFile();
                    }
```

```
            } finally {
                  try {
                     socket.close();
                  } catch(Exception e) {
                     System.err.println("Socket not closed " + e.getMessage());
                     outf.println("Socket not closed " + e.getMessage());
                     e.printStackTrace(outf);
                     outf.close();
                     nextstage.createFile();
                  }
            }
            // finish user
            System.out.println("Closing User " + User_Ip);
            System.out.println(User_Ip + " Closed");
         }
   }

   /** This class is used for claclulating the next stage of the game
    *  after all the active players has either send there information
    *  or their time to accomplish that is over
    */
   public class GameCompute extends Thread{
         // Variable to hold the next stage
         String all;
         public GameCompute(){
            try
                  {
                     // output file for the log
                     outf = new PrintWriter(
                      new BufferedWriter(
                        new FileWriter(
                          new File("Output.log"),true)));
                  }catch(IOException e){System.out.println("Unable to create files "
+e.getMessage());
                  outf.println("Unable to create files " + e.getMessage());
                  e.printStackTrace(outf);
                  outf.close();
                  nextstage.createFile();
                  }
            start();
         }

         /** This method is run every time the communication has over
          *  and it calculating the next stage of the game
          */
         public void run(){
            Communicate tmp;
            String[] strall;
            try{
```

```
                while(true){
                   synchronized(this){
                        strall = new String[Num_of_Players];
                        // for every player
                        for(int i=0;i<ActivePlayers.size();i++)
                           {
                                tmp = (Communicate)(ActivePlayers.get(i));
                                // built a string array with the
                                // input information
                                if(i==0)
                                   strall[i] = tmp.str;
                                else
                                   strall[i]= tmp.str;
                           }
                        try{
                           // call the method for claculating the next stage
                           all = calcnextstate.calcState(strall,ActivePlayers.size());
                           }catch(Exception e){System.out.println("Error in calculation "
+ e.getMessage());
                        e.printStackTrace();
                        }
                        UserFinished = 0;
                        // notify the clients to start a new exchange
                        notifyAll();
                        // suspend the computation until
                        // a new exhange is done
                        try{
                           wait();
                        }catch(InterruptedException e){
                           System.out.println("Unable to pause Thread " +
e.getMessage());
                           outf.println("Unable to pause Thread " + e.getMessage());
                           e.printStackTrace(outf);
                           outf.close();
                           nextstage.createFile();
                        }
                   }
                }
             }catch(Exception e){
                System.out.println("Error in GameCompute " + e.getMessage());
                outf.println("Error in GameCompute " + e.getMessage());
                e.printStackTrace(outf);
                outf.close();
                nextstage.createFile();
             }
          }

          /** This method is responsible for resuming the Game
           */
```

```
        public synchronized void resumeGame(){
           notify();
        }

        /** This method is called every time that the program needs
         *  to create a file for a log entry
         */
        private void createFile(){
           try
                {
                   // output file for the log
                   outf = new PrintWriter(
                    new BufferedWriter(
                      new FileWriter(
                       new File("Output.log"),true)));
                }catch(IOException e){System.out.println("Unable to create files "
+e.getMessage());
                outf.println("Unable to create files " + e.getMessage());
                e.printStackTrace(outf);
                outf.close();
                nextstage.createFile();
                }
        }
   }

   /** This method is called when the game begins and it waits
    *  to hear for request for a new client. Every time a new client
    *  request to enter the Game if the maximun number of players is
    *  not reach the a new communicate object is create else the
    *  client is rejected
    */
   public synchronized void gamePlay() throws IOException{
        ServerSocket s = new ServerSocket(PORT);
        System.out.println("Server Started");
        try {
           while(true) {
                // if the maximum number of clients is not reach
                if(ActivePlayers.size()<Num_of_Players)
                   {
                        // if a new client is requesting a connection
                        Socket socket = s.accept();
                        synchronized(nextstage){
                           try {
                                // create a new Communication Thread
                                // communicate with the client
                                ActivePlayers.add(new Communicate(socket));
                           } catch(Exception e) {
                                System.out.println("Unable to close Socket " +
e.getMessage());
```

```
                                socket.close();
                                outf.println("Unable to close Socket " + e.getMessage());
                                e.printStackTrace(outf);
                                e.printStackTrace();
                                outf.close();
                                nextstage.createFile();
                            }
                        }
                    }
                    // sleep every 100 milisecond
                    try{
                        Thread.currentThread().sleep(100);
                    }catch(InterruptedException e){System.out.println("Unable to sleep
gamePlay");
                    outf.println("Unable to sleep Thread " + e.getMessage());
                    e.printStackTrace(outf);
                    outf.close();
                    nextstage.createFile();
                    }
                }
            } finally {
                s.close();
                outf.println("Error in Startup");
                outf.close();
                nextstage.createFile();
            }
    }

    /** This method returns the port the Server listens to
     *  @return The port the the Server listens to
     */
    public static int getServerPort() throws Exception{
            return PORT;
    }

    /** This method returns the Address the Server is located to
     *  @exception
     *  @return The port the the Server listens to
     */
    public static InetAddress getServerip() throws Exception{
            return InetAddress.getLocalHost();
    }
    /*
    public static void main(String[] args) throws Exception{
            GameServer gp = new GameServer();
            gp.gamePlay();
    }
    */
}
```

A.1.2 Κώδικας interface

```
// InteGameServer.java Finished at 22:40 18/4/2004

/** This Interface is created in order to guide a user to create the
 *  correct methods for the GameServer interface
 *  @author Phanos Pavlides
 *  @version 2.0
 *  @see GameServer3D
 */
public interface InteGameServer{
   /** This method is called every time the game needs to calculate
    *  the next state. It gets as parameters an array of string that
    *  represents the received information of the users. The s is the
    *  number of users that send information before this method is called
    */
   public String calcState(String[] all,int s);
}
```

A.2 Κώδικας Πελάτη

A.2.1 Κώδικας συστήματος επικοινωνίας

```
// Client.java Finished at 00:29 15/4/2004
import java.net.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.math.*;


/** This class is created after the Game class and has the responsibility
 *  of communicating with the server and exhange information
 *  @author Phanos Pavlides
 *  @version 2.0
*/
public class Client {
   // Variable to the class tha calls for the next frame
   private DrawGame draw = new DrawGame();
   // Variable for the connection thread
   private ClientThread clientthread;
   // Variable for the port to which you  connect to
   private static int PORT;
   // Variable to the address to which the client resides on
   private String address = null;
   // Variable to the next frame class
```

```
private Game nextframe;
// Variable to hold the recieved information
private String nextstage;
// Variable to hold the initilized information
private String initInfo = null;

/** Creates a Client object with the default values of
 *  Port = 8080, address = localhost
 *  @exception
*/
public Client()
        throws IOException, InterruptedException, Exception{
        PORT = 8080;
        this.address = getClientip();
        gpClient();
}

/** Creates a Client object with the default values of
 *  address = localhost and Port the given port
 *  @exception
*/
public Client(int port)
        throws IOException, InterruptedException, Exception{
        this.PORT = port;
        this.address = getClientip();
        gpClient();
}

/** Creates a Client object with the value of
 *  address equals to the given address and
 *  Port the given port
 *  @exception
 */
public Client(String address, int port)
        throws IOException, InterruptedException, Exception{
        this.address = address;
        this.PORT = port;
        gpClient();
}

/** Creates a Client object with the value of
  *  address equals to the given address and
  *  Port the given port
  *  @exception
  */
public Client(InetAddress address, int port, Game nextframe)
        throws IOException, InterruptedException, Exception{
        String ip = address.toString();
        int locofip = ip.indexOf("/");
```

```
        ip = ip.substring(locofip+1);
        this.address = ip;
        this.PORT = port;
        this.nextframe = nextframe;
        if(this.nextframe !=null)
           initInfo = this.nextframe.getInitInfo();
        gpClient();
}

/** Returns as a string the ip of
 * that machine the class is run
 * @exception
 * @return The ip address of the machine that the class is run
 */
public String getClientip() throws Exception{
        InetAddress a = InetAddress.getLocalHost();
        String ip = a.toString();
        int locofip = ip.indexOf("/");
        ip = ip.substring(locofip+1);
        return ip;
}

/** Returns as a string the address of
 * that machine the class is run
 * @exception
 * @return The address of the machine that the class is run
 */
public String getip() throws Exception{
        InetAddress a = InetAddress.getLocalHost();
        String ip = a.toString();
        return ip;
}

/** This is an inner class that is responsible
 *  for the communication with the server
 */
public class ClientThread extends Thread {
        private Socket socket;
        private BufferedReader in;
        private PrintWriter out;
        private String User_Ip;
        private boolean finish = false;
        private boolean finish_comm = false;

        /** Creates a ClientThread object that connect
         *  whith the computer whith the address addr
         */
        public ClientThread(String addr) {
           try{
```

```
            // A conection is made to the Server that is located
            // to the addrres addr and listens to the port Port
            socket =
               new Socket(addr, PORT);
      }catch(IOException e) {
            System.err.println("Socket failed");
      }
      try {
            // creates the input stream from the Server
            in =
           new BufferedReader(
            new InputStreamReader(
             socket.getInputStream()));
            // creates the output stream to the SErver
            out =
               new PrintWriter(
           new BufferedWriter(
            new OutputStreamWriter(
              socket.getOutputStream())), true);
            start();
      } catch(IOException e) {

            try {
               // close the socket
               socket.close();
            } catch(IOException e2) {
               System.err.println("Socket not closed");
            }
      }
}


/** Set the values finish = true so the game can continue
 */
private void finishGame(){
   finish = true;
}

/** This method is run when the object is created and
 *  it runs while the Game is active is order to communicate
 *  with the server
 */
public void run() {
   try {
            // initialized information is exchange with the Server after
            // the connection is made
            String msg = null;
            out.println(initInfo);
            msg = in.readLine();
```

```
                //if(msg != null)
                   //(msg.compareTo(initInfo + "OK")==0)
                         //tem.out.println("User " + initInfo + " has Entered the Game");
                int i = 0;
                while(true) {
                   String str = null;
                   // if the game has finished then break the loop
                   if(finish == true)
                         break;
                   synchronized(draw){
                         if(nextframe !=null)
                            // Get the information from the Player
                            str = nextframe.getInfo();
                         // send it to the Server
                         out.println(str);
                         // receive the new information fronm the Server
                         nextstage = in.readLine();
                         i++;
                         finish_comm = true;
                         try{
                            // suspend until the the new information is given
                            // to the Player
                            draw.wait();
                         }catch(InterruptedException e){}
                   }
                }
                // if the loop breaks the send to Server the information
                // that the client is done
                out.println("QUIT");
          } catch(IOException e) {
                System.err.println("IO Exception" + e.getMessage());
                e.printStackTrace();
          } finally {
                try {
                   out.println("QUIT");
                   socket.close();
                } catch(IOException e) {
                   System.err.println("Socket not closed");
                }
          }
       }
}

/** This is an inner class that is responsible
 *  for the sending a singal to the game that
 *  a new information has game from the server
 */
public class DrawGame extends Thread{
       /** Creates a DrawGame object and starts the Thread
```

```
         */
        public DrawGame(){
           start();
        }
        /** This method is run when the object is created and
         *  it runs while the Game is active is to send messages
         *  to the game
         */
        public void run(){
           while(true){
                synchronized(this){
                   if(clientthread!=null)
                        // if the client has done an exchange
                        if(clientthread.finish_comm == true){
                           {
                                if(nextframe!=null)
                                   // then send the information to the Player
                                   nextframe.sendInfo(nextstage);
                                clientthread.finish_comm = false;
                           }
                           // notify the Client to begin a new exchange
                           notify();
                        }
                }
           }
        }
    }

    /** Starts the game by creating an instance of the ClientThread
     *  object that is responsible of communicating with the server
     */
    private void gpClient(){
        clientthread = new ClientThread(address);
    }

    /*public static void main(String[] args)
       throws Exception{
           Client client;
           if(args.length == 2)
              client = new Client(args[0], Integer.parseInt(args[1]));
           else
              if(args.length == 0)
                   client = new Client();
              else
                   System.out.println("Usage : java Client REMOTEIP PORT");
                   }*/
}
```

A.2.2 Κώδικας interface

```
// InteGraph.java Finished at 22:40 18/4/2004

/** This Interface is created in order to guide a user to create the
 *  correct methods for the Game interface
 *  @author Phanos Pavlides
 *  @version 2.0
 *  @see Game
 */

public interface InteGraph{
   /** This method is called after the Game object
    *  is created and is purpose is to pass initial information
    *  to the server
    */
   public String getInitInfo();
   /** This method is called every time the player wants to pass some new
    *  information to the Server
    */
   public String getInfo();
   /** This method is called every time a new update has come
    *  from the Server
    */
   public void sendInfo(String info);
}
```

Παράρτημα Β

Κώδικας Συστήματος Παιγνιδιού

Στο παράρτημα Β δίνεται ο κώδικας του συστήματος παιγνιδιού. Το σύστημα παιγνιδιού αποτελείται από τον κώδικα του εξυπηρετητή και των κώδικα του πελάτη. Για τον κώδικα του εξυπηρετητή παραθέτουμε τον κώδικα που αποτελεί την υλοποίηση του κομματιού που υπολογίζει τη νέα κατάσταση. Στον πελάτη έχουμε τον κώδικα που αποτελεί το παιγνίδι το οποίο επικοινωνεί με τον χρήστη. Επιπλέον υπάρχουν δυο κλάσεις η κλάση παιγνίδι η οποία χειρίζεται το παιγνίδι και η κλάση εχθρός η οποία αποτελεί τους εχθρούς (φαντάσματα) στο παιγνίδι μας.

Β.1 Κώδικας Εξυπηρετητή

Β.1.1 Κώδικας παιγνιδιού

```
// GameServer3D.java Finished at 21:55 19/4/2004
import java.lang.Math;

/** This class is created to calculate the next stage of the Game.
 *  It receives from the GameServer the given information
 *  from the players and calculates the next stage.
 *  Then pass the information to the GameServer to send the
 *  information to the Clients
 *  @author Phanos Pavlides
 *  @version 2.0
 */
public class GameServer3D implements InteGameServer{
   /** Variable to hold te number of Enemies in the Game */
   public static int Num_of_Enemies = 8;
```

```
// Variable to hold the information for every enemy
private String[] va = new String[Num_of_Enemies];
// Variable for the GameServer object
private GameServer gp;
// Variable for the x, y, z coordinate of the initial enemy
private float x,y,z;
// Variable for the GameData structure
private GameData[] game = new GameData[Num_of_Enemies];
// Variable for the point gredit
private int point = 5;
//Variable for the enemy_id
private int enemy_id = 0;
//Variable for the point previus
private int p_previus = 0;

/** This method initialize the Enemies data*/
public void initialize(){
      for(int i=0;i<Num_of_Enemies;i++)
         va[i] = getInitialCoordinates();
      for(int i=0;i<Num_of_Enemies;i++)
         game[i] = getData(va[i]);
}

/** This method calculates the initialCoordinates
 *  of every Enemy
 *  @return A String representing the initial data
 */
public String getInitialCoordinates(){
      String s = "#";
      z = 0.0f;
      y = (float)Math.random();
      if(y<0.5)
         {
            y = - (float)Math.random();
            while(y<=-0.5)
               y = -(float)Math.random();
         }
      if(y>=0.5)
         {
            y = (float)Math.random();
            while(y>=0.5)
               y = (float)Math.random();
         }

      x = (float)Math.random();
      if(x<0.5)
         x = -1.1f;
      if(x>=0.5)
         x = 1.1f;
```

```
        s = s + x;
        s = s + "#";
        s = s + y;
        s = s + "#";
        s = s + z;
        s = s + "#";

        if(x<0.5)
           s = s + "left";
        if(x>=0.5)
            s = s + "right";
        s = s + "#";

        double ran = Math.random();
        while(ran>0.010 || ran<0.001)
           ran = Math.random();
        s = s + ran;
        s = s + "#";
        s = s + enemy_id;
        s = s + "#";
        enemy_id++;
        return s;
}

/** Creates a GameServer3D object. It initialize the Enemies and
 *  creates a GameServer reference. Then it begins the game
 *  @exception
 */
public GameServer3D() throws Exception{
        initialize();
        gp = new GameServer(this);
        gp.gamePlay();
}

/** This methods decrypts the data taht it has received from the
 *  GameServer. This data are the information pass to the GameServer
 *  after a communication with the users. It returns a userData
 *  structure that represents the user selection
 *  @param data A String that represents a users selection
 *  @return A UserData object that represents the users selection
 */
public UserData decrypt(String data){
        // create a UserData
        UserData userData;
        // Create a array of string to represent every input
        String[] allstr = new String[Num_of_Enemies + 3];
        int i=0;
        int index = -1;
```

```
        // while not found the last entry
        while((index=data.indexOf("!"))!=-1)
           {
                // get the next entry and put in the allstr
                data = data.substring(index+1,data.length());
                 if(data.indexOf("!")!=-1)
                    {

                        allstr[i] = data.substring(index,data.indexOf("!"));
                        data = data.substring(data.indexOf("!"),data.length());
                        i++;
                    }
           }
        // Create a new UserData object and add the data to it
        userData = new UserData(allstr[0]);
        int num_enemies = Integer.parseInt(allstr[2]);
        userData.setNumEnemies(num_enemies);
        userData.setPoints(Integer.parseInt(allstr[1]));
        int m = 3;
        for(int j=3;j<num_enemies+3;j++)
           {
                userData.set(j-m,Integer.parseInt(allstr[j]));
                m++;
           }

        return userData;
}

/** This methods is called by the GameServer every time a nest stage
 *  calculation is needed. It takes as parameters an array of
 *  String that represents every users selection, and s a number
 *  of the users that send information. It returns a string that represents
 *  the next stage of the game. This string is return to the GameServer
 *  class, in order to be passed on to the users
 *  @param all An array of String that represents every users selection
 *  @param s An integer of the number of users send information
 *  @return A String repesenting the next stage of the game
 */
public String calcState(String[] all,int s){
        // Varibale to hold the return String of the next stage
        String returnString = null;
        // Array of UserData to represent every users selection
        UserData[] userdata = new UserData[s];
        // Array of string to represents every users seletion
        String[] userpoints = new String[s];

        // For every user that has succefully send information
        for(int i=0;i<s;i++)
           {
```

```
                int p = 0;
                if(all[i]!=null && all[i].compareTo("null")!=0){
                   // Get the user data and save it to a UserData variable
                   userdata[i] = decrypt(all[i]);
                   p = userdata[i].points;
                   p_previus = p;
                   int n_e = userdata[i].numEnemies;
                   userpoints[i] = "";
                   userpoints[i] = userpoints[i] + userdata[i].getUserId();
                   // for every enemy check to see if the user has hitted it
                   for(int j=0;j<n_e;j++)
                         {
                            GameData g = null;
                            g =
game[userdata[i].enemyId[j]].getData(userdata[i].enemyId[j]);
                            // if true then gredit the user with the
                            //appropriate points and reset the Enemy
                            //to the begining of the screen
                            if(g!=null)
                                  {
                                     if(g.left==true)
                                           g.dmoveall[0] = -1.1f;
                                     if(g.right==true)
                                           g.dmoveall[0] = 1.1f;
                                     g.setSpeed();
                                     g.setHeight();
                                     p = p + point;
                                  }
                         }
                   if(p==p_previus && p!=0)
                         userpoints[i] = "null!null";
                   else
                         {
                            userpoints[i] = userpoints[i] + "!";
                            userpoints[i] = userpoints[i] + p;
                         }
                }
             }
          // for every enemy add the new location to the return String
          for(int i=0;i<Num_of_Enemies;i++)
             {
                if(game[i].left == true)
                   {
                      game[i].dmoveall[0] = game[i].dmoveall[0] + game[i].speed;
                      if(game[i].dmoveall[0]>1.5f)
                         game[i].setSide("right");
                   }
                if(game[i].right == true)
                   {
```

```
                        game[i].dmoveall[0] = game[i].dmoveall[0] - game[i].speed;
                        if(game[i].dmoveall[0]<-1.5f)
                           game[i].setSide("left");
                     }
                  if(returnString == null)
                     returnString = game[i].toString();
                  else
                     returnString = returnString + game[i].toString();
               }
            for(int i=0;i<Num_of_Enemies;i++)
               {
                     returnString = returnString + "!";
                     returnString = returnString + game[i].id;
               }
            for(int i=0;i<s;i++)
               {
                     returnString = returnString + "!";
                     if(userpoints[i] == null)
                        userpoints[i] = "null!null";
                     returnString = returnString + userpoints[i];
               }
            returnString = returnString + "!";
            return returnString;
      }

      /** This method is used to return an Enemy data object
       *  given the information in a string representation
       *  @param data A String representing tha data
       *  @return A GameData object with the information provided
       */
      private GameData getData(String data){
            GameData gamedata;
            String[] allstr = new String[6];
            int i=0;
            int index = -1;
            while((index=data.indexOf("#"))!=-1)
               {
                     data = data.substring(index+1,data.length());
                     if(data.indexOf("#")!=-1)
                        {

                           allstr[i] = data.substring(index,data.indexOf("#"));
                           data = data.substring(data.indexOf("#"),data.length());
                           i++;
                        }
               }
            gamedata = new GameData(allstr[0], allstr[1], allstr[2], allstr[3],
allstr[4],allstr[5]);
            return gamedata;
```

```
    }

    public static void main(String[] args) throws Exception{
        GameServer3D game = new GameServer3D();
    }
}

/** This class represents The Game data of the an Enemy
 */
class GameData{
    /** An array  that represents the location x,y,z of the Enemy */
    public float[] dmoveall = new float[3];
    /** Variable for the left location of the Enemy */
    public boolean left = false;
    /** Variable for the right location of the Enemy */
    public boolean right = false;
    /** Variable for the enemy speed */
    public float speed = 0.0f;
    /** Variable for the enemy id */
    public int id = -1;

    /** Creates a GameData object with the selected information
     *  @param a1 A String that represents the x coordinate of the Enemy
     *  @param a2 A String that represents the y coordinate of the Enemy
     *  @param a3 A String that represents the z coordinate of the Enemy
     *  @param a4 A String that represents the left or right location
     *  @param a5 A String that represents the enemy id
     */
    public GameData(String a1,String a2, String a3, String a4, String a5, String a6){
        dmoveall[0] = Float.parseFloat(a1);
        dmoveall[1] = Float.parseFloat(a2);
        dmoveall[2] = Float.parseFloat(a3);
         if(a4.compareTo("left")==0)
           {
                left = true;
                right = false;
           }
        if(a4.compareTo("right")==0)
           {
                left = false;
                right = true;
           }
        speed = Float.parseFloat(a5);
        id = Integer.parseInt(a6);
    }

    /** This method returns a reference to this GameData object
     *  if the given id equal this GameData id
     *  @param id The id to which to check to see if it equals to the
```

```
 *  GameData  id
 *  @return A reference to this GameData object
*/
public GameData getData(int id){
      if(this.id == id)
         return this;
      else
         return null;
}

/** This method sets the side to with the Enemy will begin
 *  @param str A String representing either left or right
 */
public void setSide(String str){
      if(str.compareTo("left")==0)
         {
               left = true;
               right = false;
         }
      if(str.compareTo("right")==0)
         {
               left = false;
               right = true;
         }
}

/** This method is called every time a GameData reference
 *  is printed in order to give a string representaion of
 *  the GameData object
 *  @return A String that represents the GameData Object
 */
public String toString(){
      String s = "!";
      s = s + dmoveall[0];
      s = s + " ";
      s = s + dmoveall[1];
      s = s + " ";
      s = s + dmoveall[2];
      s = s + " ";
      s = s + "!";
      if(left == true)
         s = s + "left";
      if(right == true)
         s = s + "right";
      s = s + "!";
      s = s + speed;
      return s;
}
```

```
    /** This method sets the speed of the Enemy */
    public void setSpeed(){
        double ran = Math.random();
        while(ran>0.010 || ran<0.001)
           ran = Math.random();
        speed = (float)ran;
    }

    /** This method sets the height of the Enemy */
    public void setHeight(){
        float y = (float)Math.random();
        if(y<0.5)
           {
              y = - (float)Math.random();
              while(y<-0.5)
                 y = -(float)Math.random();
           }
        if(y>=0.5)
           {
              y = (float)Math.random();
              while(y>=0.5)
                 y = (float)Math.random();
           }
        dmoveall[1] = y;
    }
}

/** This Class represents a users selection */
class UserData{
    // Variable to hold the users id
    private String usersId;
    /** Variable for the user number of points */
    public int points = 0;
    /** Variable for the user hitted enemies */
    public int numEnemies = 0;
    /** Varaible for the ids of the enemies the user hitted */
    public int[] enemyId = new int[GameServer3D.Num_of_Enemies];

    /** Creates a UserData object whith the give id
     *  @param usersId An integer representing the users id
     */
    public UserData(String usersId){
        this.usersId = usersId;
    }

    /** This method sets number of enemies the user has hitted
     *  @param n The number of enemies
     */
    public void setNumEnemies(int n){
```

```
        numEnemies = n;
    }

    /** This method sets the enemy id of an enemy the user has hitted
     *  @param i The location on the array to save the enemy id
     *  @param id The Enemy id to be saved
     */
    public void set(int i, int id){
        enemyId[i] = id;
    }

    /** This method sets the points of the user
     *  @param p The number of points the user allready has
     */
    public void setPoints(int p){
        points = p;
    }

    /** This method adds to the points the given points
     *  @param p The number of points to be added
     */
    public int addPoints(int p){
        points = points + p;
        return points;
    }

    /** This method returns the users id
     *  @return The users id
     */
    public String getUserId(){
        return usersId;
    }

     /** This method is called every time a UserData reference
     *  is printed in order to give a string representaion of
     *  the UserData object
     *  @return A String that represents the UserData Object
     */
    public String toString(){
        String s = "";
        s = s + usersId;
        for(int i=0;i<GameServer3D.Num_of_Enemies;i++)
           s = s + " " + enemyId[i];
        return s;
    }
}
```

B.2 Κώδικας Πελάτη

B.2.1 Κώδικας παιγνιδιού

```
// Game.java Finished at 23:05 18/4/2004
import java.applet.Applet;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.image.*;
import java.net.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;

/** This Class contains the Code for the 3D Game. It creates a client
 *  object for communicating with the Server, and it draws the game
 *  at the clients screen
 *  @author Phanos Pavlides
 *  @version 2.0
 *  @see GameServer3D
 */
public class Game extends Applet implements InteGraph{
   // Variable for a bounding sphere
   private BoundingSphere sphere;
   // Variable for transforming the camera
   private TransformGroup cameraTransform;
   // Variable for holding the camera objects
   private TransformGroup cameraGroup;
   // Variable for the ground texture
   private TextureLoader loader2;
   // Variable for the sky texture
   private TextureLoader loader3;
   // Variable for the mouse movement
   private TransformGroup TGT;
   // Variable to hole the Client Thread for communication
   private Client client;
   // Variable for the information beetwen Server and Client
   private String info;
   // Variable for the initial information
   private String initInfo;
   // Variable for the information send by the Player
   private String up_info;
   // Variable for the address of the Server
```

```
private InetAddress address = null;
// Variable Array to hold the Enemie
private Enemy[] enemy;
// Variable for the number of Enemies in the Game
private int Num_of_Enemies = 8;
// Variable for the information that a Players send to the Client
private String sendString;
// Variable to hold the Players unique id
private String userId = "";
// Variable for the user points
private int points = 0;
// Variable for the usersname
private JTextField username;
// Variable for the start button
private JButton start;
// Variable for the canvas3D where the Game renders
private Canvas3D canvas3D;
// Variable to hold the number of Players
private int Num_of_Players = 0;
private int Num_of_Player = 4;
// Variable s for then Text that appears in the Screen
private Text2D text2d1;
private Text2D text2d2;
private Text2D text2d3;
private Text2D text2d4;
// Variables for the game synchrinization
private int first = 0;
private Vector3f[] vectors = new Vector3f[10];
private Vector3f tmpvec = new Vector3f();
private float[] floatvector = new float[3];
private double x1;
private double y1;
private boolean enter = false;
private String s = "";
private String ts = "";
private String[] usersIds = new String[Num_of_Player];
private String[] usersPoints = new String[Num_of_Player];
private int[] usertimeout = new int[Num_of_Player];
private int Max_Allowed = 250;

/** This method is called once in the begining to initialized
 *  the Game by calling the Client Thread constructor
 *  @param initInfo The initial information passed to the Server
 *  @exception
 */
private void begin(String initInfo)
        throws IOException, InterruptedException, Exception{
        this.initInfo = initInfo;
        address = InetAddress.getByName("www2.cs.ucy.ac.cy");
```

```
        client = new Client(address,8642,this);
}

/** This method is called by the client in order
 *  to received the initialized information
 *  @return A String representing the initial information
 */
public String getInitInfo(){
        return initInfo;
}

/** This method is called by the client in order
 *  to received the information from the Player
 *  every time
 *  @return A String representing the Players information
 */
public String getInfo(){
        synchronized(this){
           if(sendString!=null)
                {
                   up_info = sendString;
                   sendString = null;
                }
           return up_info;
        }
}

/** This method is called by the Player in order
 *  to received the information from the Server
 *  every time
 *  @param info A String representing the Servers information
 */
public void sendInfo(String info){
        this.info = info;
        String[] al;
        // Decrypt the information
        al = decrypt();
        int m = 0;
        int enemies_num = 3*Num_of_Enemies;
        int start = 0;
        int enemies_points = enemies_num + Num_of_Enemies + 2 * Num_of_Player;
        // For every Enemy get the new location
        for(int i=0;i<enemies_num;i=i+3)
           {
                String altmp = al[i];
                floatvector[0] = Float.parseFloat(altmp.substring(0,altmp.indexOf(" ")));
                altmp = altmp.substring(altmp.indexOf(" ")+1,altmp.length());
                floatvector[1] = Float.parseFloat(altmp.substring(0,altmp.indexOf(" ")));
```

```
            altmp = altmp.substring(altmp.indexOf(" ")+1,altmp.length());
            floatvector[2] = Float.parseFloat(altmp.substring(0,altmp.indexOf(" ")));

            tmpvec.set(floatvector);
            enemy[m].setLocation(tmpvec);
            vectors[m] = tmpvec;
            m++;
        }

    // The first time initialized the enemies
    if(first==0)
        {
            int j=1;
            for(int i=2;i<=enemies_num;i=i+3)
                {
                    enemy[i-2*j].setSpeed(Float.parseFloat(al[i]));
                    j++;
                }
            for(int i = enemies_num;i<enemies_num + Num_of_Enemies;i++)
                {
                    enemy[i-enemies_num].setId(Integer.parseInt(al[i]));
                }
            first = 1;
        }
    // set the variable for the users that have update there points
    int ui;
    // set the variable so the user can be update the first time that
    // comes in the game
    boolean ft = false;
    // Set the Players interface and text
    for(int i=enemies_num + Num_of_Enemies;i<enemies_points;i=i+2)
        {
            if(al[i]!=null)
                if(al[i].compareTo("null")!=0)
                    {
                        // set the ui variable to 0
                        ui = 0;
                        // if the input name exist
                        if(al[i].indexOf("_")!=-1)
                            ts = al[i].substring(0,al[i].indexOf("_"));
                        // Look for it in the array userIds
                        while(ui<Num_of_Player && usersIds[ui].compareTo(ts)!=0)
                            {
                                usertimeout[ui]++;
                                ui++;
                            }
                        // if not found
                        if(ui==4)
                            {
```

```
                                    ui = 0;
                                    // find an empty slot and add the new name
                                    while(ui<Num_of_Player &&
usersIds[ui].compareTo("")!=0)
                                        ui++;
                                    // if an empty slot found then
                                    if(ui<4)
                                        {
                                          // add the name
                                          usersIds[ui] = ts;
                                          // and the points
                                          usersPoints[ui] = al[i+1];
                                          ft = true;
                                          usertimeout[ui] = 0;
                                        }
                                  }
                                // if the name is found originally
                                if(ui<4){
                                   usertimeout[ui] = 0;
                                     // if the users points have been updated
                                     if(usersPoints[ui].compareTo(al[i+1])!=0
                                       || ft ==true)
                                       {
                                          // update the user points and name
                                          // to the appropriate slot
                                          usersPoints[ui] = al[i+1];
                                          s = ts + " " + al[i+1];
                                          ft = false;
                                          if(al[i].compareTo(userId)==0)
                                             points = Integer.parseInt(al[i+1]);
                                          switch(ui){
                                          case 0:
                                             text2d1.setString(s);
                                             break;
                                          case 1:
                                             text2d2.setString(s);
                                             break;
                                          case 2:
                                             text2d3.setString(s);
                                             break;
                                          case 3:
                                             text2d4.setString(s);
                                             break;
                                          }
                                       }
                                  }
                              }

                    }
```

```
        for(int a = 0;a<Num_of_Player;a++)
           if(usertimeout[a]>Max_Allowed)
                 {
                    usersIds[a] = "";
                    usersPoints[a] = "";
                 }

}

/** This method is called every time a Player
 *  requires to decrypt the information send by the Server
 *  @return An array representing the decrypted information
 */
private String[] decrypt(){
        String[] al = new String[4 * Num_of_Enemies + 2 * Num_of_Player];
        int index = 0;
        String str = info;
        int i = 0;
        if(info!=null)
           {
                 while((index=str.indexOf("!"))!=-1)
                    {
                          str = str.substring(index+1,str.length());
                          if(str.indexOf("!")!=-1)
                             {
                           al[i] = str.substring(index,str.indexOf("!"));
                              str = str.substring(str.indexOf("!"),str.length());
                              i++;
                             }
                    }
           }
        return al;
}

/** This method is called whent the object first created
 *  in order to get the users name
 */
private void getUserName(){
        // Label for user message
        JLabel label = new JLabel("Please enter your name");
        // set the label possition on the applet
        label.setLocation(320-60,240-120);
        label.setBounds(320-60,240-120,120,20);

        // Set the TextField for the users selection
        username = new JTextField(20);
        // set the textfield possition on the applet
        username.setLocation(320-60,240-100);
        username.setBounds(320-60,240-100,120,20);
```

```
        // Create the button to start the game
        start = new JButton("Start Game");
        // make it listen for user interaction
        start.addActionListener(new BeginGame());
        // set the buttons possition on the applet
        start.setLocation(320-60,240-70);
        start.setBounds(320-60,240-70,120,20);

        // add all the above on to the applet
        add(label);
        add(username);
        add(start);
    }

    /** This method is called whent the user has entered his/hers choice,
     *  in order for the applet to continue with the game
     */
    private void startGame(){
        // remove the text field
        remove(username);
        // remove the start button
        remove(start);
        // repaint so the clear the screen
        repaint();
        // set the canvas visible so the user can begin
        canvas3D.setVisible(true);
        // create a client Thread object to star communicating with
        // the server
        try{
           userId = username.getText() + "_" +  InetAddress.getLocalHost();
           up_info = "!" + userId + "!" + 0 + "!0!0!";
           begin(userId);
        }catch(Exception e){System.out.println("Unable to start client " +
e.getMessage());
        e.printStackTrace();
        }
    }

    /** The constructor creates a Game object and initilize the Game
      */
    public Game() {
        // Set the Layout manager
        setLayout(null);
        // Set a bounding sphere for the Graphics
        sphere = new BoundingSphere(new Point3d(),1000);
        try{
           UIManager.setLookAndFeel(UIManager.
          getSystemLookAndFeelClassName());
```

```
        }catch(Exception e){e.printStackTrace(System.err);}
        // Get the users name
        getUserName();
        // Create the Graphics configuration
        GraphicsConfiguration config =
       SimpleUniverse.getPreferredConfiguration();
    // Creates a canvas object
        canvas3D = new Canvas3D(config);
        // Add the canvas to the applet
    canvas3D.setLocation(0,0);
        canvas3D.setBounds(0,0,640,480);
        add("Center", canvas3D);

        // Create Listener for the mouse movement and interaction
        ML ml = new ML();
        MML mml = new MML();
        // Add the listeners to the canvas
        canvas3D.addMouseListener(ml);
        canvas3D.addMouseMotionListener(mml);
        // Change the mouse to the target icon
        canvas3D.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        // Create a virtual universe
        VirtualUniverse virtualUniverse = new VirtualUniverse();
        try{
            // create a url for the ground texture
            loader2 = new TextureLoader(new
java.net.URL("http://www2.cs.ucy.ac.cy/~cs00pt/Game/Ground.jpg"),canvas3D);
            // create a url for the sky texture
            loader3 = new TextureLoader(new
java.net.URL("http://www2.cs.ucy.ac.cy/~cs00pt/Game/Sky.jpg"),canvas3D);
        }catch(java.net.MalformedURLException e){
        System.out.println("Unable to find the url "+ e.getMessage());
        e.printStackTrace();
        }
        // Create the locale for the virtual schene
        Locale locale = new Locale(virtualUniverse);

        // add the camera to the scene
        locale.addBranchGraph(createCamera(canvas3D));
        // add the scene to the virtual scene
        locale.addBranchGraph(createSceneGraph());
        // initialize the userPoints
        for(int init = 0;init<Num_of_Player;init++)
            {
                usersIds[init] = "";
                usersPoints[init] = "";
            }
        String host = "localhost";
        try{
```

```
            address = InetAddress.getByName(host);
        }catch(UnknownHostException e){System.out.println("Unable to find host" +
e.getMessage());}
        // Sset the canvas3D not visible after the Game constructor finishes
        new DummyThread(canvas3D);
    }


    /** This class is created once in order to se the canvas3D
     *  to not visible
     */
    class DummyThread extends Thread{
        // The canvas3D object
        Canvas3D canvas3D;
        /** Creates a DummyThread objcet with the given
         *  canvas3D object
         */
        public DummyThread(Canvas3D canvas3D){
            this.canvas3D = canvas3D;
            start();
        }

        /** This method is called when the object is created */
        public void run(){
            // set the canvas3D to not visible
            canvas3D.setVisible(false);
        }
    }

    /** This method is called once to create and initialize
     *  the camera for the 3D world
     *  @param canvas3D The canvas3D object to which to ad the view
     *  @return A BranchGroup object that contains the camera objects
     */
    private BranchGroup createCamera(Canvas3D canvas3D){
        // Create a branch object to hold the Camera objects
        BranchGroup cameraBranch = new BranchGroup();
        // add a camera transform group
        cameraTransform = new TransformGroup();
        cameraTransform.setCapability(TransformGroup.ALLOW_TRANSFORM_WR
ITE);
        cameraTransform.setCapability(TransformGroup.ALLOW_TRANSFORM_RE
AD);

        // Gives to the object the capability to be change
        // after the scene is created
        cameraGroup = new TransformGroup();
        Transform3D T3D2 = new Transform3D();
        T3D2.setTranslation(new Vector3f(0.0f,0.0f,-2.41f));
```

```
        cameraGroup.setTransform(T3D2);
        cameraGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE
);
        cameraGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        cameraGroup.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND)
;

        //Create a camera view object and add it to the scene
        Transform3D T3D = new Transform3D();
        T3D.setTranslation(new Vector3f(0.0f,0.0f,2.41f));
        cameraTransform.setTransform(T3D);
        ViewPlatform cameraView = new ViewPlatform();

        // create the camera view and referencing it to
        // the physical body and physical enviroment
        View view = new View();
        view.attachViewPlatform(cameraView);
        PhysicalBody physicalBody = new PhysicalBody();
        view.setPhysicalBody(physicalBody);
        PhysicalEnvironment physicalEnvironment = new PhysicalEnvironment();
        view.setPhysicalEnvironment(physicalEnvironment);
        view.addCanvas3D(canvas3D);

        // add all the objects to the branch group
        cameraTransform.addChild(cameraView);
        cameraTransform.addChild(cameraGroup);
        cameraBranch.addChild(cameraTransform);

        // compile the scene and return it
        cameraBranch.compile();
        return cameraBranch;
    }

    /** This method creates an object that represents a sun
     *  in the scene
     *  @return A TransformGroup that contains the new object
     */
    private TransformGroup createSun(){
        TransformGroup sunTrans = new TransformGroup();
        Transform3D T3D = new Transform3D();
        Vector3f tran = new Vector3f(0.8f, 0.6f, -0.2f);
        T3D.setTranslation(tran);
        sunTrans.setTransform(T3D);
        sunTrans.addChild(new Sphere(0.15f ,Sphere.GENERATE_NORMALS,60,
createAppearanceSun()));
        return sunTrans;
    }

    /** This method creates the appearance for the sun object
```

```
     *  @return A Appearance for the selected object
     */
    private Appearance createAppearanceSun() {
            Appearance appear = new Appearance();
            Material material = new Material();
            material.setEmissiveColor(new Color3f(0.8f,0.8f,0.4f));
            appear.setMaterial(material);
            return appear;
    }

    /** This method creates the scene graph objects
     *  and adds the to the scene
     *  @return A BranchGroup object that contains the scene
     */
    private BranchGroup createSceneGraph() {
            // Create the root of the branch graph
            BranchGroup objRoot = new BranchGroup();

            // Transorm group for navigate the camera
             Vector3f translate = new Vector3f();
             Transform3D T3D = new Transform3D();

            // Change the background color to white
            Background bg = new Background();
            bg.setColor(1.0f, 1.0f, 1.0f);
            bg.setApplicationBounds(sphere);
            objRoot.addChild(bg);

            // setting the ambient light for the scene in its default values
            AmbientLight light = new AmbientLight();
            light.setInfluencingBounds(sphere);
            objRoot.addChild(light);

            // setting the directional light1 for the scene
            // whith color white
            DirectionalLight lightD1 = new DirectionalLight();
            lightD1.setInfluencingBounds(sphere);
            Vector3f direction1 = new Vector3f(-1.0f, -1.0f, 0.0f);
            direction1.normalize();
            lightD1.setDirection(direction1);
            lightD1.setColor(new Color3f(0.0f, 1.0f, 1.0f));
            objRoot.addChild(lightD1);

            // setting the directional light2 for the scene
            // whith color white
            DirectionalLight lightD2 = new DirectionalLight();
            lightD2.setInfluencingBounds(sphere);
            Vector3f direction2 = new Vector3f(0.0f, 0.0f, -1.0f);
            direction2.normalize();
```

```
lightD2.setDirection(direction2);
lightD2.setColor(new Color3f(1.0f, 1.0f, 1.0f));

BranchGroup BG = new BranchGroup();

TGT = new TransformGroup();
TGT.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
TGT.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
objRoot.addChild(TGT);

BG.addChild(lightD2);
BG.compile();
cameraGroup.addChild(BG);

// Create and adds the Sky and Ground objects to the scene
objRoot.addChild(new Sky());
objRoot.addChild(new Ground());

// Creates and initialize the enemies to the scene
enemy = new Enemy[Num_of_Enemies];
for(int i=0;i<Num_of_Enemies;i++)
   enemy[i] = new Enemy(0.3f,0.2f);
for(int i=0;i<Num_of_Enemies;i++)
   objRoot.addChild(enemy[i].getEnemy());

// Create and add the sun to the scene
objRoot.addChild(createSun());

//create user inteface
Transform3D trText = new Transform3D();

TransformGroup textTg1 = new TransformGroup();
trText.setTranslation(new Vector3f(-0.9f,-0.3f,0.0f));
textTg1.setTransform(trText);

TransformGroup textTg2 = new TransformGroup();
trText.setTranslation(new Vector3f(-0.9f,-0.4f,0.0f));
textTg2.setTransform(trText);

TransformGroup textTg3 = new TransformGroup();
trText.setTranslation(new Vector3f(-0.9f,-0.5f,0.0f));
textTg3.setTransform(trText);

TransformGroup textTg4 = new TransformGroup();
trText.setTranslation(new Vector3f(-0.9f,-0.6f,0.0f));
textTg4.setTransform(trText);

text2d1 = new Text2D("<Player 1>",
                     new Color3f(0.9f, 1.0f, 1.0f),
```

```
                        "Helvetica", 24, Font.ITALIC);
        text2d2 = new Text2D("<Player 2>",
                        new Color3f(0.9f, 1.0f, 1.0f),
                        "Helvetica", 24, Font.ITALIC);
        text2d3 = new Text2D("<Player 3>",
                        new Color3f(0.9f, 1.0f, 1.0f),
                        "Helvetica", 24, Font.ITALIC);
        text2d4 = new Text2D("<Player 4>",
                        new Color3f(0.9f, 1.0f, 1.0f),
                        "Helvetica", 24, Font.ITALIC);

        text2d1.setCapability(Text2D.ALLOW_APPEARANCE_READ);
        Appearance textAppear1 = text2d1.getAppearance();
        textAppear1.setCapability(Appearance.ALLOW_TEXTURE_READ);
        textAppear1.setCapability(Appearance.ALLOW_TEXTURE_WRITE);

        text2d2.setCapability(Text2D.ALLOW_APPEARANCE_READ);
        Appearance textAppear2 = text2d2.getAppearance();
        textAppear2.setCapability(Appearance.ALLOW_TEXTURE_READ);
        textAppear2.setCapability(Appearance.ALLOW_TEXTURE_WRITE);

        text2d3.setCapability(Text2D.ALLOW_APPEARANCE_READ);
        Appearance textAppear3 = text2d3.getAppearance();
        textAppear3.setCapability(Appearance.ALLOW_TEXTURE_READ);
        textAppear3.setCapability(Appearance.ALLOW_TEXTURE_WRITE);

        text2d4.setCapability(Text2D.ALLOW_APPEARANCE_READ);
        Appearance textAppear4 = text2d4.getAppearance();
        textAppear4.setCapability(Appearance.ALLOW_TEXTURE_READ);
        textAppear4.setCapability(Appearance.ALLOW_TEXTURE_WRITE);

        // adding the text to the scene
        textTg1.addChild(text2d1);
        textTg2.addChild(text2d2);
        textTg3.addChild(text2d3);
        textTg4.addChild(text2d4);

        objRoot.addChild(textTg1);
        objRoot.addChild(textTg2);
        objRoot.addChild(textTg3);
        objRoot.addChild(textTg4);
        objRoot.compile();
        return objRoot;
    }

    /** A class the implements the MouseListener interface
     *  in order to listen for users interactions
     */
    class ML implements MouseListener{
```

```
/** This method is called whene the mouse key is Released
 *  @param e A MouseEvent object
 */
public void mouseReleased(MouseEvent e){
}

/** This method is called whene the mouse key is Pressed
 *  @param e A MouseEvent object
 */
public void mousePressed(MouseEvent e){
}

/** This method is called whene the mouse Enters an area
 *  @param e A MouseEvent object
 */
public void mouseExited(MouseEvent e){
   enter = false;
}

/** This method is called whene the mouse key leaves am area
 *  @param e A MouseEvent object
 */
public void mouseEntered(MouseEvent e){
   enter = true;
}

/** This method is called whene the mouse key is Clicked.
 *  It ckecks to see if a the users has fire on an enemy.
 *  It then saves the information in string to be send to the server
 *  @param e A MouseEvent object
 */
public void mouseClicked(MouseEvent e){
   synchronized(this){
   sendString = null;
   String s = "!";
   s = s + userId;
   s = s + "!";
   s = s + points;
   s = s + "!";
   String tmps = "";
   int num_shoot = 0;
   // For every enemy check to see if you fire on an enemy
   for(int i=0;i<Num_of_Enemies;i++)
         {
            // if true the add the information to a string
            // to send it the server
            if(enemy[i].checkBox((float)x1,(float)y1,0.0f)==true)
            {
```

```
                        tmps = tmps + enemy[i].id + "!";
                        num_shoot++;
                    }
                }
            s = s + num_shoot;
            s = s + "!";
            s = s + tmps;
            sendString = s;
            }
        }
}

/** A class the implements the MouseMotionListener interface
 *  in order to listen for users interactions
 */
class MML implements MouseMotionListener{
        private int x_c = 320;
        private int y_c = 240;
        private int x;
        private int y;
        private double a = 0.65f/x_c;
        private double b = 0.45f/y_c;
        private Transform3D T3DC = new Transform3D();
        private Vector3f vec = new Vector3f();
        private float[] vecfloat = new float[3];

        /** This method is called when the mouse is Moved.
         *  Every time the users moves the mouse then the new
         *  location on the screen is calculated
         *  @param e A MouseEvent object
         */
        public void mouseMoved(MouseEvent e){
            if(enter == true){
                x = e.getX();
                y = e.getY();
                x1 = (x-x_c) * a;
                y1 = (y-y_c) * b;
                y1 = - y1;
                vecfloat[0] = (float)x1;
                vecfloat[1] = (float)y1;
                vecfloat[2] = 0.0f;
                vec.set(vecfloat);
                T3DC.setTranslation(vec);
                TGT.setTransform(T3DC);
            }
        }

        /** This method is called when the mouse drag an object
         *  @param e A MouseEvent object
```

```
         */
        public void mouseDragged(MouseEvent e){
        }
    }

    public static void main(String[] args) {
        Frame frame = new MainFrame(new Game(), 640, 480);
    }

    /** A class that create the Ground object in the scene
     *  it loads the texture for the ground and creating the
     *  geometry and appearance of the object
     */
    public class Ground extends Shape3D{
        // Variable for the Geometry
        private Geometry uiGeometry;
        // Variable for the Appearance
        private Appearance uiAppearance;
        private float size = 0.05f;

        /** Creates the object and initialize the object parameters
         */
        public Ground(){
           // Create the object geometry
           uiGeometry = createGeometry();
            // Create the object appearance
           uiAppearance = createAppearance();
           // add the geometry and appearance to the object
           this.setGeometry(uiGeometry);
           this.setAppearance(uiAppearance);
        }

        /** This method is used for creating the object geometry
         *  @return A geometry representing the objects gemetry
         */
        private Geometry createGeometry(){
           int vc[] = {4};
           TriangleStripArray plane = new
TriangleStripArray(4,TriangleStripArray.COORDINATES |
TriangleStripArray.COLOR_3 |  TriangleStripArray.TEXTURE_COORDINATE_2,
vc);
           Color3f color = new Color3f(0.0f,1.0f,0.0f);
           plane.setCoordinate(0,new Point3f(-1.2f,-0.2f,-0.4f));
           plane.setCoordinate(1,new Point3f(-1.2f,-1.0f,0.0f));
           plane.setCoordinate(2,new Point3f(1.2f,-0.2f,-0.4f));
           plane.setCoordinate(3,new Point3f(1.2f,-1.0f,0.0f));

           TexCoord2f tc = new TexCoord2f();
           tc.set(0.0f,0.0f);
```

```
            plane.setTextureCoordinate(0,0,tc);
            tc.set(1.0f,0.0f);
            plane.setTextureCoordinate(0,1,tc);
            tc.set(0.0f,1.0f);
            plane.setTextureCoordinate(0,2,tc);
            tc.set(1.0f,1.0f);
            plane.setTextureCoordinate(0,3,tc);

            for(int i=0;i<4;i++)
              plane.setColor(i,color);
            return plane;
         }

         /** This method is used for creating the object appearance
          *  @return An appearance representing the objects gemetry
          */
         private Appearance createAppearance(){
            Appearance appear = new Appearance();
            Material material = new Material();
            appear.setMaterial(material);
            TransparencyAttributes ta = new
TransparencyAttributes(TransparencyAttributes.NICEST,0.5f);
            ImageComponent2D image = loader2.getImage();
            Texture2D texture = new Texture2D(Texture.BASE_LEVEL,
Texture.RGBA,
                              image.getWidth(), image.getHeight());
            texture.setImage(0,image);
            appear.setTexture(texture);
            return appear;
         }
   }

   /** A class that create the Sky object in the scene
    *  it loads the texture for the ground and creating the
    *  geometry and appearance of the object
    */
   public class Sky extends Shape3D{
         // Variable for the Geometry
         private Geometry uiGeometry;
         // Variable for the Appearance
         private Appearance uiAppearance;
         private float size = 0.05f;

         /** Creates the object and initialize the object parameters
          */
         public Sky(){
            // Create the object geometry
            uiGeometry = createGeometry();
            // Create the object appearance
```

```
            uiAppearance = createAppearance();
            // add the geometry and appearance to the object
            this.setGeometry(uiGeometry);
            this.setAppearance(uiAppearance);
        }

        /** This method is used for creating the object geometry
         *  @return A geometry representing the objects gemetry
         */
        private Geometry createGeometry(){
            int vc[] = {4};
            TriangleStripArray plane = new
TriangleStripArray(4,TriangleStripArray.COORDINATES |
TriangleStripArray.COLOR_3 |  TriangleStripArray.TEXTURE_COORDINATE_2,
vc);
            Color3f color = new Color3f(0.0f,1.0f,0.0f);
            plane.setCoordinate(0,new Point3f(-1.8f,1.2f,0.0f));
            plane.setCoordinate(1,new Point3f(-1.8f,-0.4f,-1.8f));
            plane.setCoordinate(2,new Point3f(1.8f,1.2f,0.0f));
            plane.setCoordinate(3,new Point3f(1.8f,-0.4f,-1.8f));

            TexCoord2f tc = new TexCoord2f();
            tc.set(0.0f,0.0f);
            plane.setTextureCoordinate(0,0,tc);
            tc.set(1.0f,0.0f);
            plane.setTextureCoordinate(0,1,tc);
            tc.set(0.0f,1.0f);
            plane.setTextureCoordinate(0,2,tc);
            tc.set(1.0f,1.0f);
            plane.setTextureCoordinate(0,3,tc);

            for(int i=0;i<4;i++)
              plane.setColor(i,color);
            return plane;
        }

        /** This method is used for creating the object appearance
         *  @return An appearance representing the objects gemetry
         */
        private Appearance createAppearance(){
            Appearance appear = new Appearance();
            Material material = new Material();
            appear.setMaterial(material);
            TransparencyAttributes ta = new
TransparencyAttributes(TransparencyAttributes.NICEST,0.5f);
            ImageComponent2D image = loader3.getImage();
            Texture2D texture = new Texture2D(Texture.BASE_LEVEL,
Texture.RGBA,
                              image.getWidth(), image.getHeight());
```

```
            texture.setImage(0,image);
            appear.setTexture(texture);
            return appear;
        }
    }
    /** This class listens for users ineraction */
     class BeginGame implements ActionListener{
        /** This method is called every time
         *  the object gets an interaction from the
         *  user
         *  @param e An ActionEvent object
         */
        public void actionPerformed(ActionEvent e){
            if(username.getText().compareTo("")!=0)
                {
                    // When the user has enter text and pressed
                    // the button the start the game
                    startGame();
                }
        }
     }
}
```

B.2.2 Κώδικας εχθρού (φαντάσματος)

```
// Enemy.java Finished at 21:34 19/4/2004
import javax.media.j3d.*;
import javax.vecmath.*;

/** This Class contains the Code for the Enemy object. It creates a Enemy
 *  using the geometry and appearance provided and setting the speed
 *  and other parameters of the Enemy
 *  @author Phanos Pavlides
 *  @version 2.0
 */
public class Enemy extends Shape3D{
    // Variable for the geometry
    private Geometry uiGeometry;
     // Variable for the appearance
    private Appearance uiAppearance;
    // Variables for the enemy geometry
```

```
private float h = 0.3f;
private float w = 0.2f;
private float l = 0.02f;
// Variable for the speed
private float speed = 0.005f;
// Variable for the transformation of the enemy
private  TransformGroup  tg = new TransformGroup();
private Transform3D tmpT3D = new Transform3D();
/** Variable to hold the enemy id */
public int id;
// Variable for getting the location information for the Enemy
private Vector3f tmpvec = new Vector3f();
private Transform3D tmpT = new Transform3D();
private float[] dim = new float[3];
private float[] dim2 = new float[3];

/** Creates an Enemy object by setting the geometry and appearance
 *  @param h The height of the Enemy
 *  @param w The width of the Enemy
 */
public Enemy(float h, float w){
        // save the given info to the variable provided
        this.h = h;
        this.w = w;
        // set the object geometry
        uiGeometry = createGeometry();
        // set the object appearance
        uiAppearance = createAppearance();
        this.setGeometry(uiGeometry);
        this.setAppearance(uiAppearance);
}

/** This methods sets the Enemys id
```

```
     *  @param id The enemys id
     */
    public void setId(int id){
        this.id = id;
    }

    /** This method creates the objects geometry
     *  @return A geometry representing the objects geometry
     */
    private Geometry createGeometry(){
        TriangleArray line = new TriangleArray(42, TriangleArray.COORDINATES |
TriangleArray.COLOR_4);
        Color4f color = new Color4f(0.5f,0.5f,0.5f,0.8f);
        line.setCoordinate(0,new Point3f(-(w/2-l),0.0f,0.0f));
        line.setCoordinate(1,new Point3f(-(w/2),-h/8,0.0f));
        line.setCoordinate(2,new Point3f(-2*(w/2-l)/3,0.0f,0.0f));
        line.setCoordinate(3,new Point3f(-2*(w/2-l)/3,0.0f,0.0f));
        line.setCoordinate(4,new Point3f(-((w/2-l)/3),-h/8,0.0f));
        line.setCoordinate(5,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(6,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(7,new Point3f((w/2-l)/3,-h/8,0.0f));
        line.setCoordinate(8,new Point3f(2*(w/2-l)/3,0.0f,0.0f));
        line.setCoordinate(9,new Point3f(2*(w/2-l)/3,0.0f,0.0f));
        line.setCoordinate(10,new Point3f(w/2,-h/8,0.0f));
        line.setCoordinate(11,new Point3f(w/2-l,0.0f,0.0f));

        line.setCoordinate(12,new Point3f(w/2-l,0.0f,0.0f));
        line.setCoordinate(13,new Point3f(4*(w/2-l)/5,(h/2)/5,0.0f));
        line.setCoordinate(14,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(15,new Point3f(4*(w/2-l)/5,(h/2)/5,0.0f));
        line.setCoordinate(16,new Point3f(3*(w/2-l)/5,2*(h/2)/5,0.0f));
        line.setCoordinate(17,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(18,new Point3f(3*(w/2-l)/5,2*(h/2)/5,0.0f));
```

```
        line.setCoordinate(19,new Point3f(2*(w/2-l)/5,3*(h/2)/5,0.0f));
        line.setCoordinate(20,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(21,new Point3f(2*(w/2-l)/5,3*(h/2)/5,0.0f));
        line.setCoordinate(22,new Point3f(1*(w/2-l)/5,4*(h/2)/5,0.0f));
        line.setCoordinate(23,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(24,new Point3f(1*(w/2-l)/5,4*(h/2)/5,0.0f));
        line.setCoordinate(25,new Point3f(0*(w/2-l)/5,5*(h/2)/5,0.0f));
        line.setCoordinate(26,new Point3f(0.0f,0.0f,0.0f));

        line.setCoordinate(27,new Point3f(0.0f,h/2,0.0f));
        line.setCoordinate(28,new Point3f(-1*(w/2-l)/5,4*(h/2)/5,0.0f));
        line.setCoordinate(29,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(30,new Point3f(-1*(w/2-l)/5,4*(h/2)/5,0.0f));
        line.setCoordinate(31,new Point3f(-2*(w/2-l)/5,3*(h/2)/5,0.0f));
        line.setCoordinate(32,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(33,new Point3f(-2*(w/2-l)/5,3*(h/2)/5,0.0f));
        line.setCoordinate(34,new Point3f(-3*(w/2-l)/5,2*(h/2)/5,0.0f));
        line.setCoordinate(35,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(36,new Point3f(-3*(w/2-l)/5,2*(h/2)/5,0.0f));
        line.setCoordinate(37,new Point3f(-4*(w/2-l)/5,1*(h/2)/5,0.0f));
        line.setCoordinate(38,new Point3f(0.0f,0.0f,0.0f));
        line.setCoordinate(39,new Point3f(-4*(w/2-l)/5,1*(h/2)/5,0.0f));
        line.setCoordinate(40,new Point3f(-(w/2-l),0.0f,0.0f));
        line.setCoordinate(41,new Point3f(0.0f,0.0f,0.0f));

        for(int i=0;i<42;i++)
           line.setColor(i,color);
        return line;
}

/** This method sets the new location the that the enemy will appear on
 *  @param v3f A Vector3f that represents the new location
 */
```

```
    public void setLocation(Vector3f v3f){
        tmpT3D.setTranslation(v3f);
        tg.setTransform(tmpT3D);
        }

    /** This method creates the objects Appearance
     *  @return An appearance representing the objects appearance
     */
    private Appearance createAppearance(){
        Appearance appear = new Appearance();
        Material material = new Material();
        appear.setMaterial(material);
        TransparencyAttributes ta = new
TransparencyAttributes(TransparencyAttributes.NICEST,0.5f);
        appear.setTransparencyAttributes(ta);
        return appear;
    }

    /** This method sets the InitialCoordinates of the Enemy
     *  @param v A Vector3f that represents the location
     */
    public void setInitialCoordinates(Vector3f v){
        setLocation(v);
    }

    /** This method return the transform group that the Enemy is located
     *  @return A TransformGroup with the Enemy node
     */
    public TransformGroup getEnemy(){
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        tg.addChild(this);
        return tg;
```

```
}

/** This method sets the speed which the Enemy will use to be update
 *  @param speed A float number representing the speed
 */
public void setSpeed(float speed){
        this.speed = speed;
}

/** This method check to see when the user has hit this Enemy or not
 *  @param v A vector3f representing the targets location
 *  @return a boolean on when the enemy has been hitted or not
 */
public boolean checkBox(Vector3f v){
        tg.getTransform(tmpT);
        tmpT.get(tmpvec);
        tmpvec.get(dim);
        v.get(dim2);
        dim[0] = 0.6f * dim[0];
        dim[1] = 0.6f * dim[1];
        dim[2] = 0.6f * dim[2];
        if(dim2[0]<=(dim[0]+0.05) && dim2[0]>=(dim[0]-0.05)
          &&
          dim2[1]<=(dim[1]+0.08) && dim2[1]>=(dim[1]))
           return true;
        else
           return false;
}

 /** This method check to see when the user has hit this Enemy or not
 *  @param x1 A float number representing the x variable of
 *  the targets location
 *  @param y1 A float number representing the y variable of
```

```
     *  the targets location
     *  @param z1 A float number representing the z variable of
     *  the targets location
     *  @return a boolean on when the enemy has been hitted or not
     */
    public boolean checkBox(float x1, float y1, float z1){
            tg.getTransform(tmpT);
            tmpT.get(tmpvec);
            tmpvec.get(dim);
            dim[0] = 0.6f * dim[0];
            dim[1] = 0.6f * dim[1];
            dim[2] = 0.6f * dim[2];
            if(x1<=(dim[0]+0.05) && x1>=(dim[0]-0.05)
              &&
              y1<=(dim[1]+0.08) && y1>=(dim[1]))
               return true;
            else
               return false;
    }

}
```

Παράρτημα Γ

Κώδικας επαλήθευσης συστήματος στην Versa

Στο παράρτημα Γ δίνεται ο κώδικας του συστήματος σε versa. Το σύστημα έχει μελετηθεί με την χρήση του συστήματος επαλήθευσής versa ώστε να βρεθούν λάθη και αδιέξοδα στο σύστημα από την φάση των προδιαγραφών. Ο κώδικας του συστήματος βρίσκεται σε ένα αρχείο το spec.acsr. Στην συνέχεια βλέπουμε τον κώδικα αυτό.

Γ.1 Κώδικας επαλήθευσης συστήματος στην versa

```
#define Np 2     // O arithmos ton paixton
#define Tcomm 30 // o xronos pou dinete se kathee paixth gia na kanei
mia epeikeinonia
#define Tcompute 10 // O xronos pou xreiazete gia na ypologisei thn
nea katastash

// to kommati strtup tou play
StartUp[Np] = rec x.({}:x + (joinGame,
1).('activate,2).StartGame[Np]);
StartGame[Np] = ('beginGameS,2).StartUp[Np-1];
StartUp[Np-1] = rec x.({}:x + (joinGame,1).('activate,2).StartUp[Np-2]
                        + (quitGame,1).FinishGame[Np-1]
                        + (deactivate,1).FinishGame[Np-1]);
StartUp[i] = rec x.({}:x +(joinGame,1).StartUp[i+1]
                     +(quitGame,1).StartUp[i-1]
                     +(deactivate,1).StartUp[i-1]){i,1,Np-2};
StartUp[0] = rec x.({}:x + (quitGame,1).StartUp[1] +
(deactivate,1).StartUp[1]);
FinishGame[Np-1] = ('gameOver,2).StartUp[Np];
```

```
PlayS = rec x.({}:x +
(beginGameS,2).scope(Game,gameOver,infinite,x,NIL,NIL))\{computeGame,
doneCompute, addPlayers, removePlayers};

// to kommati pou trexei gia to commounication me ton kahte paixtei
// kai to kommati pou ypologizei thn nea katastash
Game = (InactiveCommS1 || InactiveCommS2 || ComputeGameState[Np] ||
ActivePlayers[0]);

//Begin of parallel users
// o kathe paixths perimenei gia kapoio xrono sto opoio prepei na
steilei
// kai na labei plhrofories apo kahte paixth
InactiveCommS1 = rec y1.({}:y1+(activate,2).CommS1);
CommS1 = scope(CommS11,Null,10000,NIL,('deactivate,1).y1, NIL);

CommS11 = scope(ExchangeInfo1, contStage, Tcomm, NextStage1,
('computeGame,1).NextStage1, NIL);
ExchangeInfo1 = ('gameState, 2).GetData1;
GetData1 = rec w1.({}:w1 + (moveInfo,
1).('computeGame,1).('contStage,1).NIL);
NextStage1 = rec z1.({}:z1 + (doneCompute, 3).CommS1);

InactiveCommS2 = rec y2.({}:y2 + (activate,2).CommS2);
CommS2 = scope(CommS22,Null,10000,NIL,('deactivate,1).y2, NIL);

CommS22 = scope(ExchangeInfo2, contStage, Tcomm, NextStage2,
('computeGame,1).NextStage2,NIL);

ExchangeInfo2 = ('gameState, 2).GetData2;
GetData2 = rec w2.({}:w2 + (moveInfo,
1).('computeGame,1).('contStage,1).NIL);
NextStage2 = rec z2.({}:z2+ (doneCompute, 3).CommS2);

//End of parallel users

// to kommati pou upologizei thn nea katastash tvn paixton
ComputeGameState[i] = rec xc.({}:xc + (computeGame,1).('addPlayer,
2).ComputeGameState[i-1]
+('doneCompute,3).(removePlayers, 2).ComputeGameState[i+1]){i,1,Np-1};
```

```
ComputeGameState[Np] = rec xc.({}:xc + (computeGame,1).('addPlayer,
2).ComputeGameState[Np-1]);


ComputeGameState[0] =
scope(Compute,Null,Tcompute,NIL,('doneCompute,3).(removePlayers,
2).ComputeGameState[1],NIL);

Compute = {(computState,1)}:Compute;

ActivePlayers[0] = rec xa.({}:xa + (addPlayers, 2). ActivePlayers[1]);
ActivePlayers[i] = rec xa.({}:xa + (addPlayers, 2). ActivePlayers[i+1]
                                + ('removePlayers, 2).
ActivePlayers[i-1]){i,1,Np-1};
ActivePlayers[Np] = rec xa.({}:xa + ('removePlayers, 2).
ActivePlayers[Np-1]);


// o server trexei to startup kai kai to PlayS
Server = (StartUp[Np] || PlayS)\{beginGameS,gameOver, activate};


//Acsr specification client

//  O kahte xrhsths prota kanei joing me ton server
//  kai afou o server apodektei thn prosklhsh tou xekina to paignidi
Client1 = rec xcl.({}:xcl + ('joinGame,1).PlayC1);
PlayC1 = (CommC1 || Graph1)\{drawGraph,contGame};

// o kathe xrhsths perimenei kampoio xrono ston opoio tha prepei
// na epeikeinvnish me ton server kai na steilei kai na parei
plhrofories

CommC1 = scope(CommC11,Null,10000,NIL,xcl,NIL);

CommC11 =
scope(ExchangeData1,contComm,Tcomm,NextFrame1,('drawGraph,1).NextFrame
1,NIL);
```

```
GetGameState1 = rec xc1.({}:xc1 + (gameState, 2).ExchangeData1);

ExchangeData1 = ('moveInfo, 1).('drawGraph,1).('contGomm, 1).NIL +
('quitGame, 1).xcl1;

NextFrame1 = rec x.({}:x + (contGame,1).CommC1);

// o client sxediazei sthn othonh tou upologisth thn nea katastash
Graph1 = rec x.({}:x +
(drawGraph,1).scope(Draw1,Null,Tcompute,NIL,DoneGraph1,NIL));
DoneGraph1 = rec x.({}:x + ('contGame,1).Graph1);

Draw1 = {(dummy2,1)}:Draw1;

Client2 = rec xcl2.({}:xcl2 + ('joinGame,1).PlayC2);
PlayC2 = (CommC2 || Graph2)\{drawGraph,contGame};


CommC2= scope(CommC22,Null,10000,NIL,xcl2,NIL);

CommC22 =
scope(ExchangeData2,contComm,Tcomm,NextFrame2,('drawGraph,1).NextFrame
2,NIL);

GetGameState2 = rec xc2.({}:xc2 + (gameState, 2).ExchangeData2);

ExchangeData2 = ('moveInfo, 1).('drawGraph,1).('contGomm, 1).NIL +
('quitGame, 1).xcl2;

NextFrame2 = rec x.({}:x + (contGame,1).CommC2);


Graph2 = rec x.({}:x +
(drawGraph,1).scope(Draw2,Null,Tcompute,NIL,DoneGraph2,NIL));
DoneGraph2 = rec x.({}:x + ('contGame,1).Graph2);

Draw2 = {(dummy2,1)}:Draw2;

//end client specification
```

```
// to system trexei parallhla ton server kai tou clients
System = (Server || Client1 || Client2)\{joinGame, quitGame,
gameState, moveInfo};
```