
Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»


**Μεταπτυχιακή Διατριβή**

| | |
|---|---|
| Τίτλος Διατριβής | **Δημιουργία Τρισδιάστατου Ηλεκτρονικού Παιχνιδιού Στρατηγικής** |
| Ονοματεπώνυμο Φοιτητή | **Ευστάθιος Μωραϊτίδης** |
| Πατρώνυμο | **Ιωάννης** |
| Αριθμός Μητρώου | **ΜΠΠΛ/ 06027** |
| Επιβλέπων | **Θέμις Παναγιωτόπουλος, καθηγητής** |

Ημερομηνία Παράδοσης **Φεβρουάριος 2010**

**Τριμελής Εξεταστική Επιτροπή**

| | | |
|---|---|---|
| (υπογραφή) | (υπογραφή) | (υπογραφή) |
| Όνομα Επώνυμο<br>Βαθμίδα | Όνομα Επώνυμο<br>Βαθμίδα | Όνομα Επώνυμο<br>Βαθμίδα |

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω για την πραγμάτωση της παρούσας μελέτης, όλους εκείνους που με βοήθησαν και με στήριξαν καθ' όλη την διάρκεια. Συγκεκριμένα, θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Παναγιωτόπουλο Θεμιστοκλή που μου έδωσε την ευκαιρία να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Επίσης, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον κ. Αναστασάκη Γεώργιο για την πολύτιμη βοήθειά του και καθοδήγηση σε τεχνικά και όχι μόνο θέματα. Ακόμα πολύτιμη υπήρξε η βοήθεια από την κοινότητα της LWJGL μέσω του forum της, και τον Jouvieje για την παροχή του API εισαγωγής τρισδιάστατων μοντέλων.

Θα ήθελα, ακόμα, να πω ένα μεγάλο ευχαριστώ στους φίλους μου για την ψυχολογική υποστήριξη που μου προσέφεραν. Ιδιαιτέρως ευχαριστώ τον φίλο μου Κωνσταντίνο Καρρά που συνέφερε στο δημιουργικό μέρος του παιχνιδιού με τις ιδέες του και την φίλη μου Κλέμου Νεράιδα που με βοήθησε στην αναζήτηση τρισδιάστατων μοντέλων στο διαδίκτυο.

**Περίληψη**

Τα ευφυή εικονικά περιβάλλοντα γίνονται όλο και πιο έξυπνα και οπτικά εντυπωσιακά. Αυτό ισχύει και για μία από τις εφαρμογές τους, τα ηλεκτρονικά παιχνίδια. Πρόκειται για ένα σημαντικό κλάδο που δίνει συνεχώς μεγάλη ώθηση στην ανάπτυξη της τεχνολογίας, της ψυχαγωγίας, και της παγκόσμιας οικονομίας.

Έχοντας ως αφετηρία το πρώτο τρισδιάστατο παιχνίδι 1980, τα ηλεκτρονικά παιχνίδια έχουνε διανύσει ένα μακρύ δρόμο ως σήμερα και η εξέλιξη και στον τομέα της Τεχνητή Νοημοσύνης καθώς και της γραφικής απεικόνισης, είναι τεράστια. Τα ηλεκτρονικά παιχνίδια πλησιάζουνε οπτικά όλο και πιο πολύ την πραγματικότητα, ενώ οι τεχνολογίες Τεχνητής Νοημοσύνης που χρησιμοποιούνε εξομοιώνουν συνεχώς πιο πετυχημένα ρεαλιστικότερες συμπεριφορές.

Ειδικότερα όσον αφορά τα παιχνίδια στρατηγικής, η αυξημένη υπολογιστική ισχύς επιτρέπει εκτός από την ελκυστικότερη απεικόνιση που απαιτείται από τον ανταγωνισμό, τη χρήση πολύπλοκων αλγορίθμων με σκοπό την εξομοίωση ρεαλιστικών συνθηκών.

Απώτερος σκοπός της διπλωματικής διατριβής είναι η δημιουργία ενός λειτουργικού τρισδιάστατου ηλεκτρονικού παιχνιδιού στρατηγικής. Το συγκεκριμένο είδος του παιχνιδιού είναι παιχνίδι στρατηγικής βασισμένο σε σειρά. Το API που χρησιμοποιήθηκε για την οπτική απεικόνιση, είναι η OpenGL, έναντι του Direct3D της Microsoft. Το παιχνίδι μπορεί να το παίξει ένας χρήστης, αντιμέτωπος με τον υπολογιστή ο οποίος χρησιμοποιεί μια υποτυπώδης Τεχνητή Νοημοσύνη.

**Abstract**

Intelligent virtual environments become more and more clever and visually impressive. This also applies to one of their applications, computer games. They are a significant sector that constantly boosts the development of the technology, entertainment and global economy.

Starting from the first 3D game at 1980, computer games have traversed a long road up to date and the evolution on the field of Artificial Intelligence as well as the graphical representation is tremendous. Computer games approach all the more the reality, whereas the technologies of Artificial Intelligence that they use simulate more and more realistic behaviors.

Especially as far as it concerns strategy games, the increased computing power allows, beyond the attractive graphical display is required from the competition, the use of complex algorithms with the simulating realistic Artificial Intelligence.

The ultimate goal of the dissertation thesis is the creation of a functional 3D computer strategy game. The specific genre of the game is turn based strategy game. The API used for the graphical representation is the OpenGL, instead Microsoft's Direct3D. The game can be played by one player, confronted by the CPU which uses a rudimentary Artificial Intelligence.

# Περιεχόμενα

# 1. Εισαγωγή

## 1.1. Κίνητρο της διπλωματικής εργασίας

Τα ηλεκτρονικά παιχνίδια αποτελούνε μια κύρια ασχολία της σύγχρονης ζωής με ολοένα και περισσότερο κόσμο να τα επιλέγει σαν μέσο ψυχαγωγίας. Πρόκειται για μια τεράστια βιομηχανία, η οποία παρασύρει με την ανάπτυξή της και άλλους τομείς όπως τη ψηφιακή διασκέδαση στο σύνολό της και την τεχνολογία των ηλεκτρονικών υπολογιστών.

Πέραν της σπουδαιότητας του κλάδου, ο λόγος που επέλεξα ένα τέτοιο θέμα είναι το ιδιαίτερο ενδιαφέρον που έχω για το χώρο των ηλεκτρονικών παιχνιδιών. Εκτός της ίδιας της ψυχαγωγίας που προσφέρει ένα ηλεκτρονικό παιχνίδι, το ενδιαφέρον αυτό έχει να κάνει και με τον τρόπο δημιουργίας ενός παιχνιδιού. Ποιές είναι οι τεχνολογίες που χρησιμοποιούνται, ποιά λογική υπάρχει πίσω από το παιχνίδι, ποιοί περιορισμοί παρουσιάζονται και πώς ξεπερνιούνται, και γενικότερα ποιά είναι η διαδικασία στο σύνολό της.

## 1.2. Σκοπός της διπλωματικής διατριβής

Σκοπός της διπλωματικής διατριβής, όπως υποδηλώνει και ο τίτλος της, είναι η δημιουργία ενός τρισδιάστατου παιχνιδιού στρατηγικής.

### 1.2.1. Ερευνητική περιοχή

Η ερευνητική περιοχή στην οποία ανήκει η συγκεκριμένη διατριβή είναι τα Ευφυή Εικονικά Περιβάλλοντα. Πρόκειται για ένα πολύ ενδιαφέρον τομέα της πληροφορικής που συνδυάζει το χώρο της Τεχνητή Νοημοσύνης με το χώρο της Εικονικής Πραγματικότητας.

## 1.3. Δομή της διπλωματικής εργασίας

Στο δεύτερο κεφάλαιο γίνεται μια γενική επισκόπηση των ηλεκτρονικών παιχνιδιών. Αναλύεται ο χώρος των Ευφυϊών Εικονικών Περιβαλλόντων και παρουσιάζεται η σημερινή κατάσταση των ηλεκτρονικών παιχνιδιών. Στο κεφάλαιο 3 παρουσιάζονται οι τεχνολογίες και το λογισμικό που υπάρχει διαθέσιμο για τη δημιουργία παιχνιδιών. Έμφαση δίνεται στη βιβλιοθήκη OpenGL. Στο τέταρτο κεφάλαιο ξεκινάει η σχεδίαση και ανάλυση του παιχνιδιού που θα δημιουργηθεί. Πιο αναλυτικά παρουσιάζεται η σχεδίαση του χάρτη του παιχνιδιού. Το πέμπτο κεφάλαιο περιέχει την υλοποίηση με την τεκμηρίωση των σημαντικότερων σημείων του κώδικα. Στο κεφάλαιο 6 γίνεται μια παρουσίαση της τελικής μορφής του παιχνιδιού, με όλες τις πιθανές περιπτώσεις χρήσης. Στο κεφάλαιο 7 αναφέρονται τα συμπεράσματα της διατριβής καθώς και οι πιθανές μελλοντικές της προοπτικές.

# 2. Γενική Επισκόπηση

Θα πρέπει πρώτα να αναλυθούν οι έννοιες των Εικονικών Περιβαλλόντων, Τεχνητής Νοημοσύνης και Εικονικών Πρακτόρων, για να φτάσουμε στο γνωστικό αντικείμενο στο οποίο ανήκει και αυτή η διατριβή, τα Ευφυή Εικονικά Περιβάλλοντα.

## 2.1. Εικονικά Περιβάλλοντα

Για περίπου 30 χρόνια, εικόνες που παράγονται από υπολογιστές χρησιμοποιούνται στον κινηματογράφο, τις διαφημίσεις και την τηλεόραση. Παράλληλα, επιστημονικοί ερευνητές, άνθρωποι που απασχολούνται στην ιατρική, αρχιτέκτονες ανακάλυψαν μια τεράστια δυναμική σε αυτές τις εικόνες και τις χρησιμοποιούσαν για να απεικονίσουν το αθέατο ή να αναπαραστήσουν αυτό που δεν υπάρχει καν. Αυτή ήταν η γέννηση των εικονικών κόσμων [Thalmann1994]. Ωστόσο, αυτοί οι εικονικοί κόσμοι είχαν δύο σοβαρούς περιορισμούς.

- Υπήρχε πολύ χαμηλή οπτική αναπαράσταση των ζωντανών οργανισμών ή πολύ απλά «πλάσματα» ζούσανε μέσα σε αυτούς τους κόσμους.
- Κανένας δεν μπορούσε ουσιαστικά να εισέλθει σε αυτούς τους κόσμους, η είσοδος γινόταν με την παρατήρηση δισδιάστατων εικόνων και διεπαφές δισδιάστατης αλληλεπίδρασης.

Σήμερα, νέες διεπαφές και τρισδιάστατες συσκευές μας επιτρέπουν μια ολοκληρωμένη συμμετοχή σε αυτούς τους εικονικούς κόσμους ή έστω μια άμεση και πραγματικού χρόνου επικοινωνία με αυτούς. Αυτός ο νέος τρόπος εμβύθισης σε αυτούς τους εικονικούς κόσμους καλείται Εικονική Πραγματικότητα. Οι ερευνητές έχουν καταφέρει να δημιουργήσουν φυτά, δέντρα και ζώα. Η έρευνα ανθρώπινη κίνηση έχει οδηγήσει στη δημιουργία συνθετικών ηθοποιών με πολύπλοκη κίνηση. Επιπλέον, ένας νέος τομέας βασισμένος στη βιολογία έχει προσπαθήσει να ξαναδημιουργήσει τη ζωή με μια προσέγγιση από τη βάση, η αποκαλούμενη προσέγγιση της Τεχνητής Ζωής. Όλες αυτές οι προσεγγίσεις θα πρέπει να ενσωματωθούν ώστε να δημιουργηθούν πραγματικοί εικονικοί κόσμοι με αυτόνομα έμβια όντα, φυτά, ζώα και ανθρώπους με δικές τους συμπεριφορές και πραγματικοί άνθρωποι θα μπορούν να εισέρχονται σε αυτούς τους κόσμους και να συναντούν τους κατοίκους τους.

Τα εικονικά περιβάλλοντα, που αναφέρονται στη βιβλιογραφία και ως συστήματα εικονικών περιβαλλόντων (virtual environment systems) ή και ως εικονικοί κόσμοι (virtual worlds), είναι σε γενικές γραμμές τα συστήματα τα οποία στο παρελθόν αναφέρονταν με το γενικότερο όρο εικονική πραγματικότητα. Δυστυχώς, μέχρι σήμερα δεν έχουν υπάρξει γενικώς αποδεκτοί ορισμοί για τους παραπάνω όρους. Οι Loffler και Anderson [Loeffler1994] ορίζουν την εικονική πραγματικότητα ως εξής: «Εικονική πραγματικότητα είναι ένα τρισδιάστατο περιβάλλον προσομοίωσης σε υπολογιστή του οποίου η απεικόνιση γίνεται σε πραγματικό χρόνο και εξαρτάται από τη συμπεριφορά του χρήστη».

Ο Ellis [Ellis1993] παρέχει έναν πιο λεπτομερή ορισμό: «Ένα εικονικό περιβάλλον αποτελείται από περιεχόμενο (αντικείμενα και δράστες (actors)), γεωμετρία και δυναμική, με ένα εγωκεντρικό πλαίσιο αναφοράς, που περιλαμβάνει την αντίληψη των αντικειμένων σε βάθος και που εγείρει διάφορες αισθήσεις ταυτόχρονα».


**Εικόνα 2.1 - Playstation Home, ένα σύγχρονο εικονικό περιβάλλον της Sony**

Τέλος, ο Gigante (1993) ορίζει την εμπειρία της αλληλεπίδρασης με ένα εικονικό περιβάλλον:

«Η εικονική πραγματικότητα είναι μια εμβυθισμένη, πολυ-αισθητική εμπειρία. Χαρακτηρίζεται από την ψευδαίσθηση της συμμετοχής σε ένα συνθετικό περιβάλλον και όχι απλώς από την εξωτερική παρατήρηση ενός τέτοιου περιβάλλοντος».

Με βάση τους παραπάνω ορισμούς τα βασικά χαρακτηριστικά των εικονικών περιβαλλόντων φαίνεται να είναι τα τρισδιάστατα γραφικά και ένα μοντέλο περιβάλλοντος, που αναπαριστά μια τοποθεσία από την πραγματική ζωή ή κάποια τεχνητή δομή. Ο λόγος για τον οποίο έχει επικρατήσει τα τελευταία χρόνια ο όρος «εικονικό περιβάλλον» έναντι του όρου «εικονική πραγματικότητα» είναι, γιατί στα συστήματα αυτά αφενός δε γίνεται προσπάθεια μοντελοποίησης ολόκληρου του σύμπαντος αλλά ενός περιορισμένου περιβάλλοντος ανάλογα με την εφαρμογή και αφετέρου γιατί δεν είναι υποχρεωτικό η αναπαράσταση να αφορά κάποια ρεαλιστική δομή, αλλά μπορεί κάλλιστα να είναι και κάποιος φανταστικός χώρος με ιδιόμορφους νόμους.

Ένας χρήστης εννοιολογικά «κατοικεί» στο περιβάλλον έχοντας μια τρέχουσα θέση σε αυτό και, κατά συνέπεια, μια περιορισμένη άποψη του χώρου. Έχει την ικανότητα να ταξιδεύει σε αυτό και να αλληλεπιδρά με τα αντικείμενά που τον περιβάλλουν. Τόσο η αντίληψη του περιβάλλοντος όσο και η αλληλεπίδραση του χρήστη με αυτό μπορούν να μοντελοποιηθούν με βάση την πραγματικότητα. Στην περίπτωση της αντίληψης, ένα τέτοιο παράδειγμα είναι η χρήση πολυαισθητήριων ερεθισμάτων που μιμούνται τα ερεθίσματα του πραγματικού κόσμου ενισχύοντας έτσι την αληθοφάνεια του εικονικού περιβάλλοντος. Αντίστοιχο παράδειγμα αλληλεπίδρασης είναι το βάδισμα του χρήστη πάνω σε κυλιόμενο επίπεδο που προκαλεί την πλοήγησή του στο εικονικό περιβάλλον.

## Κατηγορίες

Τα εικονικά περιβάλλοντα συνήθως συνδέονται με εξειδικευμένο υλικό που παρέχει πολύπλοκα συστήματα διεπαφής και προσφέρει νέες δυνατότητες αλληλεπίδρασης του χρήστη. Ανάλογα με τη σχέση τους με τον πραγματικό κόσμο και τις συσκευές που χρησιμοποιούνται τα εικονικά περιβάλλοντα χωρίζονται στις εξής κατηγορίες:

- Περιβάλλοντα εμβύθισης (immersive environments): οι χρήστες είναι εφοδιασμένοι με οθόνη προσαρμοσμένη στο κεφάλι (head-mounted display) και δε δέχονται δεδομένα από το φυσικό κόσμο.
- Περιβάλλοντα οθόνης (desktop environments): Η αναπαράσταση γίνεται σε οθόνη υπολογιστή κα έτσι ο χρήστης εξακολουθεί να έχει αντίληψη του φυσικού κόσμου.
- Περιβάλλοντα προβολής (projected environments): Το εικονικό περιβάλλον προβάλλεται σε ένα φυσικό χώρο, όπως ένα δωμάτιο ή μια επιφάνεια εργασίας.
- Ενισχυμένα περιβάλλοντα (augmented environments): Τα εικονικά αντικείμενα προβάλλονται πάνω στον πραγματικό κόσμο, πιθανώς με τη χρήση οθονών προσαρμοσμένων στο κεφάλι που επιτρέπουν στο χρήστη να βλέπει και μέσα από αυτές (see through).

Οι παραπάνω τρόποι διεπαφής με εικονικά περιβάλλοντα έχουν μια σειρά πλεονεκτημάτων αλλά και μειονεκτημάτων. Τα περιβάλλοντα εμβύθισης έχουν το μεγάλο προσόν ότι προσφέρουν στο χρήστη την απόλυτη εμπειρία του περιβάλλοντος. Παρόλα αυτά, απομονώνουν το χρήστη, χρησιμοποιούν πολύ εξειδικευμένη και ακριβή τεχνολογία και έχουν συνδεθεί με διάφορα προβλήματα υγείας. Τα περιβάλλοντα οθόνης αποφεύγουν αυτά τα προβλήματα, αλλά δεν παρέχουν εμπειρία πλήρους εμβύθισης. Αντίθετα, βασίζονται στην «ψυχολογική εμβύθιση», δηλαδή σε αρκετές περιπτώσεις καταφέρνουν να επιστήσουν όλη την προσοχή του χρήστη στο περιβάλλον. Τα περιβάλλοντα προβολής περιλαμβάνουν ένα πιο φυσικό σκηνικό και τη δυνατότητα να μοιραστεί η εμπειρία και με άλλους χρήστες, απαιτούν όμως κι αυτά τη χρήση εξειδικευμένης τεχνολογίας. Τέλος, τα μεικτά περιβάλλοντα προσθέτουν στον πραγματικό κόσμο επιπλέον εικονικό περιεχόμενο και πληροφορίες, αλλά απαιτούν μεγάλη ακρίβεια στο συσχετισμό μεταξύ της πραγματικής εικόνας και της προβαλλόμενης εικονικής πληροφορίας.

Πολλές φορές στα περιβάλλοντα αυτά χρησιμοποιείται εξειδικευμένο υλικό για την επικοινωνία με το χρήστη, που καθορίζει και τον τρόπο αλληλεπίδρασής του με αυτά. Κατά συνέπεια, Περιβάλλοντα εμβύθισης (immersive environments) ονομάζονται τα εικονικά περιβάλλοντα στα οποία οι χρήστες είναι εφοδιασμένοι με κατάλληλο υλικό, ώστε να δέχονται ερεθίσματα μόνο από το συνθετικό περιβάλλον και όχι από τον πραγματικό κόσμο. Στην περίπτωση που τα εικονικά αντικείμενα προστίθενται πάνω στην άποψη του πραγματικού κόσμου, τότε το

περιβάλλον ονομάζεται ενισχυμένο (augmented environment). Αντίθετα,όταν ο συνθετικός κόσμος προβάλλεται ολόκληρος πάνω σε μια ή περισσότερες επιφάνειες του πραγματικού κόσμου (π.χ. τοίχος, τραπέζι ή δωμάτιο), τότε πρόκειται για περιβάλλον προβολής (projected environment), ενώ περιβάλλον οθόνης (desktop environment) είναι η συνήθης περίπτωση που ο τρισδιάστατος κόσμος απεικονίζεται στην οθόνη του υπολογιστή.

Τα εικονικά περιβάλλοντα φαίνεται να έχουν πολλές και ενδιαφέρουσες εφαρμογές σε διάφορους τομείς, γεγονός που αναδεικνύει τη σημασία τους ως σύγχρονου μέσου αλληλεπίδρασης ανθρώπου-υπολογιστή. Στο χώρο της διασκέδασης και των τεχνών χρησιμοποιούνται, για να παρέχουν στο χρήστη ένα πιο ρεαλιστικό περιβάλλον απεικόνισης και κίνησης, και έχουν προκύψει ενδιαφέρουσες νέες εφαρμογές, όπως αλληλεπιδραστικό θέατρο, τρισδιάστατοι χώροι συνομιλίας, παιχνίδια εξομοίωσης, κλπ. Μεγάλη φαίνεται να είναι όμως και η χρησιμότητά τους στο χώρο της εκπαίδευσης, καθώς η δυνατότητα παρατήρησης των δεδομένων από διαφορετικές απόψεις και πειραματισμού με αυτά ενισχύει τη διαδικασία της μάθησης και της κατανόησης σε σχέση με την χρήση στατικού κειμένου και εικόνων. Ακόμα, η δυνατότητα προσομοίωσης ρεαλιστικών κόσμων βρίσκει μεγάλη χρησιμότητα στην εξάσκηση, ιδιαίτερα σε περιπτώσεις που οι αντίστοιχες ενέργειες σε πραγματικές συνθήκες μπορεί να έχουν μεγάλο κόστος ή και να είναι επικίνδυνες, όπως οι προσομοιωτές πτήσης και οι εικονικές χειρουργικές επεμβάσεις. Τέλος, σε ό,τι αφορά τον σχεδιασμό και την αξιολόγηση νέων προϊόντων τα εικονικά περιβάλλοντα επιτρέπουν την αναπαράσταση και το χειρισμό ενός μοντέλου, με σκοπό τον έλεγχο και τη βελτίωση της λειτουργικότητάς του, πριν ακόμα βγει στο στάδιο παραγωγής.

Παρά τη χρησιμότητα των εικονικών περιβαλλόντων σε διάφορους τομείς και την πληθώρα έρευνας που έχει πραγματοποιηθεί γύρω από αυτά τις τελευταίες δεκαετίες, ο χώρος παρουσιάζει ακόμα σημαντικές ελλείψεις. Πρώτα απ' όλα, το εξειδικευμένο υλικό που χρησιμοποιείται είναι αφενός υπερβολικά ακριβό για το μέσο χρήστη, αφετέρου όχι ιδιαίτερα αξιόπιστο, με αποτέλεσμα η συνεχής χρήση του για μεγάλο χρονικό διάστημα να προκαλεί δυσφορία. Κατά συνέπεια, ένα μεγάλο μέρος της έρευνας επικεντρώνεται στα εικονικά περιβάλλοντα οθόνης, που φαίνεται να είναι η μόνη προσιτή λύση για το μέσο χρήστη.

Δεύτερον, φαίνεται να υπάρχει έλλειψη γενικού τύπου αρχιτεκτονικών και εργαλείων ανάπτυξης για εικονικά περιβάλλοντα, με αποτέλεσμα να είναι ιδιαίτερα επίπονη και χρονοβόρα η διαδικασία κατασκευής ενός τέτοιου περιβάλλοντος, ενώ απαιτείται και σημαντικό χρονικό διάστημα για τον έλεγχο και την αποσφαλμάτωσή του, ώστε να παρέχει στο χρήστη ένα αξιόπιστο και φιλικό περιβάλλον αλληλεπίδρασης. Τέλος, στα υπάρχοντα εικονικά περιβάλλοντα παρουσιάζεται έλλειψη αυτονομίας. Σύμφωνα με την ταξινόμηση του Zeltzer [Zeltzer1992] οι τρεις διαστάσεις αξιολόγησης ενός εικονικού περιβάλλοντος είναι η αυτονομία (autonomy), δηλαδή η ικανότητα των αντικειμένων να δρουν από μόνα τους, η αλληλεπίδραση (interaction), δηλαδή το ποσοστό στο οποίο το σύστημα παρέχει έλεγχο των αντικειμένων, και η παρουσία (presence), που καθορίζεται από το εύρος και την πιστότητα των αισθητηρίων καναλιών του χρήστη που ελέγχει το περιβάλλον. Ενώ λοιπόν σήμερα οι χρήστες μπορούν να έχουν πολύπλοκες αλληλεπιδράσεις με τα αντικείμενα χρησιμοποιώντας εξειδικευμένο υλικό, όπως τα γάντια δεδομένων (data gloves), και η παρουσία τους είναι ιδιαίτερα ενισχυμένη στα περιβάλλοντα προβολής και εμβύθισης, δε φαίνεται να υπάρχει αντίστοιχη πρόοδος στον τομέα της αυτονομίας. Αυτή η έλλειψη οφείλεται κυρίως στο γεγονός ότι μέχρι πρόσφατα η υπολογιστική ισχύς των μηχανημάτων αφιερωνόταν σχεδόν αποκλειστικά στην απεικόνιση και συνθετική κίνηση του κόσμου, με αποτέλεσμα να μην περισσεύει επιπλέον υπολογιστικός χρόνος για την αυτόνομη δράση των αντικειμένων του περιβάλλοντος.

## Ανάπτυξη εικονικού περιβάλλοντος

Η δημιουργία ενός εικονικού περιβάλλοντος περιλαμβάνει την κατασκευή του μοντέλου του περιβάλλοντος και το σχεδιασμό των αλληλεπιδράσεων με το χρήστη. Όσον αφορά στο μοντέλο του περιβάλλοντος μια κατηγοριοποίηση των επιμέρους τμημάτων του είναι η εξής [Ellis1993; Thalmann1994]:

- ο χώρος σκηνικού ή η γεωμετρία, που παραμένει αμετάβλητη,
- οι χρήστες, που μπορούν να εκτελούν ενέργειες και ελέγχουν το δικό τους οπτικό πεδίο,
- οι πράκτορες ή εικονικοί ηθοποιοί, οι οποίοι έχουν ευφυία και μπορούν να ενεργούν ανεξάρτητα από τους χρήστες, και

- τα αντικείμενα που υπάρχουν στο χώρο. Αυτά διαφοροποιούνται στο επίπεδο αλληλεπίδρασης με το χρήστη και στις δυνατότητες αλλαγής της κατάστασής τους.

### 2.1.1. Τι είναι Τεχνητή Νοημοσύνη

Δεν είναι πολύ εύκολο να οριστεί το τι είναι Τεχνητή Νοημοσύνη. Η δυσκολία της έγκειται στον ορισμό του όρου «νοημοσύνη» (ή «ευφυΐα»). Άλλοι ορισμοί τον ορίζουν μέσω του εαυτού του. Άλλοι τον παρακάμπτουν. Πολλές μέθοδοι της Τεχνητής Νοημοσύνης προέρχονται από ερευνητικές προσπάθειες στο πεδίο της γνωστικής επιστήμης(cognitive science) και όχι της επιστήμης των υπολογιστών.

Οι Russell και Norvig [Russell2009] διακρίνουν τους ορισμούς της Τεχνητής Νοημοσύνης σε τέσσερις κατηγορίες, με βάση το αν χαρακτηρίζουν ένα σύστημα ως ευφυές με κριτήριο το:

- αν σκέφτεται σαν άνθρωπος (Μηχανισμός, Γνωστική Επιστήμη
- αν ενεργεί σαν άνθρωπος (Συμπεριφορά, Turing test)
- αν σκέφτεται ορθολογικά (Μηχανισμός, Νόμοι Ορθής Σκέψης)
- αν ενεργεί ορθολογικά (Συμπεριφορά, Ορθολογικοί Πράκτορες)

Σύμφωνα με τους Bourg και Seeman [Bourgage2004], τεχνητή νοημοσύνη είναι η ικανότητα του υπολογιστή ή κάποιας άλλη μηχανής να εκτελεί τέτοιες λειτουργίες που γενικά πιστεύεται ότι απαιτούνε κάποια νοημοσύνη, ευφυΐα. Αυτή η πρόταση προκαλεί και την εύλογη ερώτηση: Τι είναι νοημοσύνη; Για μερικούς, το πόσο επιτυχημένη είναι η τεχνητή νοημοσύνης εξαρτάται από το πόσο κοντά είναι στην ανθρώπινη νοημοσύνη. Άλλοι λένε πως επιπλέον χαρακτηριστικά θα πρέπει να τηρούνται ώστε να θεωρείται μια μηχανή νοήμον. Ίσως θα πρέπει να έχει συνείδηση, ακόμα και συναισθήματα καθώς αυτά συνδέονται με την νοημοσύνη. Επιπλέον, η τεχνητή νοημοσύνη θα πρέπει να μπορεί να μαθαίνει και να υιοθετεί ώστε να θεωρείται έξυπνη.

Η τεχνητή νοημοσύνη που ικανοποιεί όλα τα παραπάνω, θεωρείται δυνατή τεχνητή νοημοσύνη. Αντίθετα με τη δυνατή τεχνητή νοημοσύνη, η αδύναμη τεχνητή νοημοσύνη χρησιμοποιεί ένα πλήθος από τεχνολογίες και λειτουργίες, που δίνουνε σε μια μηχανή ειδικευμένη ευφυΐα. Η τεχνητή νοημοσύνη που χρησιμοποιείται στα παιχνίδια, ανήκει σε αυτή τη περίπτωση, της αδύναμης τεχνητής νοημοσύνης.

#### Η προσέγγιση με τη δοκιμασία Turing

Η δοκιμασία Turing (Turing Test), η οποία προτάθηκε από τον Alan Turing (1950), σχεδιάστηκε για να παρέχει έναν ικανοποιητικό λειτουργικό ορισμό της νοημοσύνης. Αντί να προτείνει μια εντεταμένη και ενδεχομένων αντιφατική λίστα γνωρισμάτων που απαιτούνται για τη νοημοσύνη, ο Turing πρότεινε μια δοκιμασία που βασιζόταν στην αδυναμία να γίνει διάκριση από τις αναμφίβολα νοήμονες οντότητες - τους ανθρώπους. Ο υπολογιστής περνά τη δοκιμασία αν ένας άνθρωπος εξεταστής, αφού θέσει μερικές γραπτές ερωτήσεις, δεν μπορεί να συμπεράνει αν οι γραπτές απαντήσεις προέρχονται από άνθρωπο ή όχι. Για να προγραμματιστεί ένας υπολογιστής για να περάσει τη δοκιμασία, χρειάζεται να γίνουν πολλά. Ο υπολογιστής θα πρέπει να έχει τις εξής ικανότητες:

- επεξεργασία φυσικής γλώσσας, ώστε να μπορεί να επικοινωνεί ικανοποιητικά σε μια γλώσσα όπως η Αγγλική
- αναπαράσταση γνώσης, ώστε να αποθηκεύει αυτά που γνωρίζει ή ακούει
- αυτοματοποιημένη συλλογιστική, ώστε να χρησιμοποιεί τις αποθηκευμένες πληροφορίες για να απαντά ερωτήσει και να παράγει νέα συμπεράσματα.
- μηχανική μάθηση, ώστε να προσαρμόζεται σε νέες περιστάσεις και εντοπίζει ή να συμπεραίνει πρότυπα.

Η δοκιμασία Turing απέφευγε εσκεμμένα την άμεση φυσική αλληλεπίδραση μεταξύ του εξεταστή και του υπολογιστή, επειδή η φυσική προσομοίωση ενός ανθρώπου δεν είναι απαραίτητη για τη νοημοσύνη. Όμως, η λεγόμενη πλήρης δοκιμασία Turing, (total Turing Test) περιλαμβάνει οπτικό σήμα, ώστε να μπορεί ο εξεταστής να εξετάζει τις αντιληπτικές ικανότητες του υποκειμένου και να έχει τη δυνατότητα να του δίνει φυσικά αντικείμενα «από το παραθυράκι». Για να περάσει την πλήρη δοκιμασία, ο υπολογιστής θα χρειαστεί να έχει:

- μηχανική όραση, ώστε να αντιλαμβάνεται αντικείμενα, και
- ρομποτική, ώστε να χειρίζεται αντικείμενα και να μετακινείται.

Αυτά τα έξι πεδία αποτελούν το μεγαλύτερο μέρος της Τεχνητής Νοημοσύνης, και ο Turing αξίζει την αναγνώριση επειδή σχεδίασε μια δοκιμασία που διατηρεί την ισχύ της μετά από 50 χρόνια. Πάντως, οι ερευνητές της Τεχνητής Νοημοσύνης έχουν αφιερώσει ελάχιστη προσπάθεια στη δημιουργία μηχανών που μπορούν να περάσουν τη δοκιμασία Turing, καθώς πίστευαν ότι είναι πιο σημαντικό να μελετήσουν τις αρχές στις οποίες βασίζεται η νοημοσύνη παρά να αντιγράψουν ένα πρωτότυπο. Η έρευνα για την «τεχνητή πτήση» έφτασε στην επιτυχία όταν οι αδελφοί Wright και άλλοι σταμάτησαν να προσπαθούν να μιμηθούν τα πουλιά και έμαθαν την αεροδυναμική.

### 2.1.2. Εικονικοί πράκτορες

Τα τελευταία χρόνια έχουν λάβει χώρα σημαντικές προσπάθειες στον τομέα της ενίσχυσης της αυτονομίας των εικονικών περιβαλλόντων με την ανάπτυξη ενός νέου πεδίου έρευνας στο χώρο αυτό, των εικονικών πρακτόρων. Εικονικοί πράκτορες (virtual agents) ονομάζονται οι αυτόνομες οντότητες σε ένα εικονικό περιβάλλον, οι οποίες έχουν συνήθως τη μορφή συνθετικών χαρακτήρων, και αλληλεπιδρούν με το περιβάλλον μέσω αισθητήρων (sensors) και επιδραστών (effectors) λαμβάνοντας αποφάσεις με βάση κάποιο μοντέλο συμπεριφοράς [Badler 1993]. Οι χαρακτήρες αυτοί μπορούν να είναι άνθρωποι, ζώα, ή ακόμα και φανταστικά πλάσματα, ενώ σε κάποιες περιπτώσεις εικονικοί πράκτορες μπορούν να είναι και μηχανήματα με πολύπλοκη συμπεριφορά, όπως αυτοκίνητα, ρομπότ, κλπ.

Οι εικονικοί πράκτορες παρουσιάζουν μεγάλη χρησιμότητα στα περισσότερα από τα πεδία εφαρμογής των εικονικών περιβαλλόντων, καθώς η παρουσία αυτόνομων χαρακτήρων σε τρισδιάστατους κόσμους ενισχύει την αληθοφάνειά τους αλλά και τις δυνατότητες αλληλεπίδρασης με τον χρήστη. Στο χώρο της μηχανολογίας, του σχεδιασμού και της συντήρησης προϊόντων αξιοποιούνται οι εικονικοί άνθρωποι για την ανάλυση και προσομοίωση εικονικών πρωτοτύπων και το σχεδιασμό βασισμένο στην προσομοίωση. Μοντέλα εικονικών πρακτόρων χρησιμοποιούνται ακόμα σε τηλεδιασκέψεις με τη χρήση ρεαλιστικών αναπαραστάσεων των χρηστών για τη μείωση του εύρους της επικοινωνίας. Στο χώρο της εκπαίδευσης έχει γίνει σημαντική έρευνα στη χρήση εικονικών πρακτόρων για την αλληλεπιδραστική καθοδήγηση του χρήστη στην πραγματοποίηση διαφόρων ενεργειών, όπως ο χειρισμός πολύπλοκων μηχανημάτων. Επιπλέον, εικονικοί πράκτορες χρησιμοποιούνται ως παρουσιαστές σε εκπαιδευτικές ή επιστημονικές εφαρμογές, ως ξεναγοί σε χώρους ειδικού ενδιαφέροντος αλλά και για άλλες μορφές επικοινωνίας, όπως η απόδοση κειμένου στη νοηματική γλώσσα των κωφαλάλων. Τέλος, η μεγαλύτερη χρήση συνθετικών χαρακτήρων εντοπίζεται σήμερα στο χώρο τις διασκέδασης και ιδιαίτερα στα παιχνίδια και στην αλληλεπιδραστική αφήγηση ιστοριών.

Αν και η υπάρχουσα έρευνα στους εικονικούς πράκτορες φαίνεται να βρίσκεται σε πρωταρχικά στάδια, καθώς υπάρχουν πολλά προβλήματα που δεν έχουν ακόμα επιλυθεί, μια πρώτη προσπάθεια ταξινόμησης των υπαρχόντων συστημάτων καταλήγει σε ένα φάσμα πρακτόρων. Στη μία άκρη του φάσματος τοποθετούνται οι φυσικοί πράκτορες (physical agents), δηλαδή αυτοί στους οποίους η έμφαση έχει δοθεί στην αληθοφανή φυσική συμπεριφορά. Στο χώρο αυτό η έρευνα επικεντρώνεται σε εικονικούς ανθρώπους ή και ζώα και τα θέματά της περιλαμβάνουν τη ρεαλιστική κίνηση και τη φυσική αλληλεπίδρασή τους με το περιβάλλον, την ομιλία, τις χειρονομίες και τις εκφράσεις προσώπου. Τέτοιοι πράκτορες συνήθως αλληλεπιδρούν με το περιβάλλον μέσω εικονικών αισθητήρων και επιδραστών που δε λειτουργούν σε συμβολικό επίπεδο αλλά προσομοιώνουν το φυσικό.

Στην άλλη άκρη του φάσματος αυτού τοποθετούνται οι γνωσιακοί πράκτορες(cognitive agents), δηλαδή αυτοί στους οποίους η έμφαση έχει δοθεί στην ανθρώπινη γνωσιακή συμπεριφορά, καθώς και στη γνωσιακή αλληλεπίδραση με τους χρήστες του συστήματος. Τα περισσότερα θέματα στο χώρο αυτό αφορούν τη φυσική γλώσσα, καθώς και τις διαδικασίες απόφασης, όπως είναι ο σχεδιασμός ενεργειών (planning). Τέτοιοι πράκτορες συνήθως λαμβάνουν μέσω των αισθήσεών τους συμβολική πληροφορία απευθείας από το περιβάλλον, με αποτέλεσμα να αμφισβητείται το κατά πόσο η διαδικασία αίσθησης είναι πραγματικά αυτόνομη.

Τα προβλήματα της έλλειψης εργαλείων ανάπτυξης στο χώρο των εικονικών περιβαλλόντων αντικατοπτρίζονται, όπως είναι φυσικό, και στην εξειδικευμένη περιοχή των εικονικών πρακτόρων. Έτσι, ενώ τα τελευταία χρόνια, κυρίως λόγω της εκτενούς χρήσης των τρισδιάστατων γραφικών στα παιχνίδια με υπολογιστή, έχουν κατασκευαστεί κάποιες μηχανές τρισδιάστατων γραφικών με δυνατότητα ενσωμάτωσης συνθετικών χαρακτήρων, είναι ευθύνη του σχεδιαστή να προγραμματίσει τη συμπεριφορά τους και να τους κάνει να

συμπεριφέρονται με αληθοφανή τρόπο. Η μηχανή από μόνη της δεν εγγυάται ότι οι συνθετικοί χαρακτήρες συμπεριφέρονται όπως θα έπρεπε και δεν παρέχει κάποιο μηχανισμό, για να προγραμματιστεί και να ρυθμιστεί εύκολα η συμπεριφορά τους. Το γεγονός αυτό δε βοηθάει ιδιαίτερα στην επαναχρησιμοποίηση των πρακτόρων, καθώς θα χρειαστεί αρκετός επιπλέον προγραμματισμός, για να συνδεθούν τμήματα ενός πράκτορα σε κάποια άλλη εφαρμογή. Επιπροσθέτως, υπάρχουν ορισμένα χαρακτηριστικά και διαδικασίες των εικονικών πρακτόρων, όπως η χρήση κάποιας εσωτερικής μνήμης, η αντίληψη και η δυνατότητα πραγματοποίησης σύνθετων ενεργειών, που φαίνεται να είναι απαραίτητες σε οποιοδήποτε περιβάλλον απαιτεί σύνθετη συμπεριφορά και αληθοφάνεια. Τα παραπάνω χαρακτηριστικά θα έπρεπε λοιπόν να αποτελούν μέρος της λειτουργικότητας του πράκτορα και όχι να προστίθενται κατά την διάρκεια ανάπτυξης της εφαρμογής.

Εκτός από την έλλειψη εργαλείων ανάπτυξης στην υπάρχουσα έρευνα στο χώρο των ευφυών εικονικών πρακτόρων είναι χαρακτηριστική επίσης και η έλλειψη κοινών μεθοδολογιών και αρχιτεκτονικών για την κατασκευή τέτοιων χαρακτήρων και την ενσωμάτωσή τους σε ευρύτερες εφαρμογές. Οι δυνατότητες, η λειτουργικότητα και ο τρόπος μοντελοποίησης τόσο των εικονικών περιβαλλόντων, όσο και των πρακτόρων που κατοικούν σε αυτά, διαφέρει από εφαρμογή σε εφαρμογή στην τρέχουσα βιβλιογραφία, με αποτέλεσμα να είναι ιδιαίτερα δύσκολο να γίνει συντονισμένη έρευνα προς μια κοινή κατεύθυνση. Η σημερινή απουσία κοινών πρωτοκόλλων και τυποποιήσεων δυσκολεύει αφάνταστα την αξιοποίηση και ενσωμάτωση τμημάτων κώδικα από άλλες προσεγγίσεις, με αποτέλεσμα να επιβαρύνεται ιδιαίτερα η διαδικασία ανάπτυξης.

Τέλος, η ανάπτυξη ενός μοντέλου πράκτορα με ικανότητες τόσο σε φυσικό, όσο και σε γνωσιακό επίπεδο, παρουσιάζει ιδιαίτερα προβλήματα λόγω της δυσκολίας διασύνδεσης μεταξύ των δύο αυτών επιπέδων. Η δυσκολία έγκειται στο γεγονός ότι οι ερευνητικές θεωρίες σε κάθε ένα από τα δύο αυτά επίπεδα βασίζονται σε διαφορετικούς τρόπους αναπαράστασης του κόσμου και των αντικειμένων του. Στους φυσικούς πράκτορες ο κόσμος αναπαρίσταται με μεγάλη λεπτομέρεια και χρησιμοποιούνται τεχνικές ελέγχου παρόμοιες με αυτές του χώρου της ρομποτικής. Αντίθετα, οι γνωσιακοί πράκτορες χρησιμοποιούν συμβολική αναπαράσταση του κόσμου, η οποία εμπεριέχει υψηλότερου επιπέδου πληροφορίες, και βασίζονται σε πολύπλοκα μοντέλα συμπεριφοράς αξιοποιώντας τις θεωρίες των ευφυών πρακτόρων, της λογικής και της συμβολικής τεχνητής νοημοσύνης. Έτσι, σε μια κοινή αρχιτεκτονική παρουσιάζεται το γνωστό από τους ευφυείς πράκτορες πρόβλημα της μετάφρασης (transduction problem), δηλαδή πώς θα μεταφραστεί η λεπτομερής αναπαράσταση του κόσμου σε μια επαρκή και ακριβή συμβολική περιγραφή.

### 2.1.3. Ευφυή Εικονικά Περιβάλλοντα

Ευφυή Εικονικά Περιβάλλοντα λέμε τα εικονικά περιβάλλοντα που περιέχουν ευφυείς εικονικούς πράκτορες [Aylett2001]. Πρόκειται για τον τομέα που προκύπτει όταν οι τεχνολογίες των εικονικών περιβαλλόντων και της Τεχνητής Νοημοσύνης συγκλίνουν. Η ώθηση για αυτό το τομέα προέρχεται από έναν αριθμό κατευθύνσεων. Καταρχάς, καθώς η ποσότητα της επεξεργαστικής ισχύος που χρησιμοποιείται για το rendering έχει αυξηθεί, έχει γίνει πλέον εφικτό να αφιερώνουμε μέρος αυτής της ισχύος σε θέματα πέρα από τον οπτικό ρεαλισμό. Όσο οπτικά ελκυστικό και να είναι ένα ευφυές περιβάλλον, αν είναι στατικό και απουσιάζει η αλλαγή και η συμπεριφορά, η τελική εμπειρία είναι περιορισμένου ενδιαφέροντος. Μια πόλη χωρίς ανθρώπους, δεν μπορεί να κάνει το χρήστη να αποκτήσει την αίσθηση της παρουσίας. Καθώς το animation μπορεί να δημιουργήσει δυναμικό ενδιαφέρον, η προδιαγεγραμμένη του φύση λειτουργεί ενάντια στην διαδραστική ελευθερία που προσφέρει ένα ευφυές περιβάλλον και κρατάει το ενδιαφέρον των χρηστών για περιορισμένο χρόνο.

Η ανάπτυξη του ηλεκτρονικού εμπορίου στο Διαδίκτυο παράγει μια αντίστοιχη ανάγκη για υποστήριξη. Καθώς το τρισδιάστατο περιεχόμενο είναι ακόμα σπάνιο, είναι προφανές πως οι χρήστες θα αναζητούν ευφυή βοήθεια σε πολλές περιπτώσεις και οι πράκτορες δισδιάστατης διεπαφής αρχίζουν να δίνουν τη θέση τους σε τρισδιάστατα ομιλούντα πρόσωπα τα οποία επίσης επεκτείνονται σε νέα μέσα όπως η διαδραστική ψηφιακή τηλεόραση. Με τον ίδιο τρόπο, μεγάλης κλίμακας περιβάλλοντα για chat κινούνται προς τα τρισδιάστατα γραφικά και την εύρεση της κατάλληλης ευφυΐας που θα διαχειρίζεται τον πληθυσμό τους.

Η αύξηση της επεξεργαστικής ισχύος έχει επίσης επηρεάσει και τη βιομηχανία των ηλεκτρονικών παιχνιδιών, κάνοντας την πολύ πιο δύσκολη στο να ανταγωνίζεται μόνο στον οπτικό ρεαλισμό. Ως αποτέλεσμα η ενσωμάτωση της τεχνολογίας Τεχνητής Νοημοσύνης μέσα στα ηλεκτρονικά παιχνίδια είναι πλέον διευρυμένη με παιχνίδια όπως το *The Sims* (2000) και

*Spore* (2008) να είναι εξολοκλήρου βασισμένα στην Τεχνητή Νοημοσύνη. Με δεδομένη την ώθηση που τα ηλεκτρονικά παιχνίδια έχουνε δώσει στην ανάπτυξη τρισδιάστατων γραφικών γενικά, θα είναι πολύ ενδιαφέρον να παρατηρηθεί αν η βιομηχανία θα προκαλέσει την ίδια επιταχυντική επίδραση στο συνδυασμό Τεχνητής Νοημοσύνης και τρισδιάστατων γραφικών.

**Εικόνα 2.2 - The Sims**

**Εικόνα 2.3 - Spore**

Η ενσωμάτωση εικονικών ανθρώπων συγκεκριμένα επίσης ανοίγει το δρόμο στη χρήση εικονικών περιβαλλόντων για εφαρμογές που μέχρι τώρα χρησιμοποιούσαν δισδιάστατους εξομοιωτές. Η εκκένωση κτηρίων σε καταστάσεις έκτακτης ανάγκης και ο έλεγχος του κόσμου σε αστικές περιοχές είναι προφανή παραδείγματα, ενώ η εξομοίωση της κίνησης των δρόμων επίσης κινείται προς τρισδιάστατα και ευφυή εικονικά αυτοκίνητα.

Από την πλευρά της τεχνητής νοημοσύνης, ομάδες ερευνητών αναγνωρίζουν τα εικονικά περιβάλλοντα ως ένα ισχυρό πειραματικό χώρο για τις τεχνολογίες τους. Η αλληλεπίδραση πραγματικού χρόνου σε ένα εικονικό κόσμο αποτελεί πιο σημαντικό κίνητρο από την αλληλεπίδραση βασισμένη σε κείμενο καθώς επίσης και πιο βοηθητική.

Τα εικονικά περιβάλλοντα επίσης επιτρέπουν το πειραματισμό με πολύπλοκους ενσωματωμένους πράκτορες χωρίς όλα τα προβλήματα των περιορισμένης διάρκειας μπαταριών, υγρών επαφών, και γενικά όλα τα προβλήματα που απαντώνται στα πειράματα του πραγματικού κόσμου. Με λιγότερη προσπάθεια που απαιτείται για την ανάπτυξη βασικών αρχιτεκτονικών ελέγχου, γίνεται εφικτό με τους ευφυείς εικονικούς πράκτορες ή με συνθετικούς χαρακτήρες, να ερευνηθεί ο μοντελισμός των συναισθημάτων. Η εργασία μπορεί να διεκπεραιωθεί και στο επίπεδο συμπεριφοράς, ενσωματώνοντας γλώσσα σώματος και χειρονομίες, και στο νοητικό επίπεδο, χρησιμοποιώντας σχεδίαση και παρέμβαση βασισμένη στα συναισθήματα.

Οι τεχνολογίες Τεχνητής Νοημοσύνης χρησιμοποιούνται επίσης σε μερικές περιπτώσεις και για να διαχειρίζονται την πολυπλοκότητα ή τον υπερβολικό επεξεργαστικό φόρτο. Μόλις η φυσική ενσωματωθεί σε ένα εικονικό περιβάλλον, ο φόρτος που απαιτείται για να επιλυθούν μεγάλα σύνολα εξισώσεων με αναλυτικές μεθόδους μπορεί να επηρεάσει σοβαρά το rendering. Συνθετικοί χαρακτήρες με πολύπλοκές σκελετικές δομές μπορούν να χρησιμοποιήσουν τεχνολογία Τεχνητής Νοημοσύνης ικανή να μαθαίνει, ώστε να αποκτούνε σχετικές ικανότητες κίνηση, οι οποίες θα ήταν πολύ δύσκολο να προγραμματιστούν αναλυτικά.

## 2.2. Τα παιχνίδια σήμερα

Τα ηλεκτρονικά παιχνίδια έχουνε αποκτήσει σημαντικό μερίδιο στην καθημερινή ζωή πολλών ανθρώπων. Δεν είναι πλέον ασχολία μόνο των μικρών παιδιών, όπως συνέβαινε παλαιότερα. Ο χαρακτήρας των παιχνιδιών γίνεται πιο ενήλικος, με σκοπό την προσέλκυση νέων ηλικιακών ομάδων. Ο αριθμός των παιχτών ανά τον κόσμο ολοένα και αυξάνεται (το παιχνίδι World of Warcraft έχει τον απίστευτο αριθμό των 11.5 εκατομμυρίων εγγραφών). Ακόμα, πολλά πανεπιστήμια εντάσσουνε το μάθημα της δημιουργίας παιχνιδιών στην ύλη των μαθημάτων τους, ή ιδρύουνε μεταπτυχιακά προγράμματα για αυτό.

Τα ηλεκτρονικά παιχνίδια, είναι μια σύγχρονη μορφή ψυχαγωγίας, και θα συνεχίσουν να αποτελούν τέτοια για πολλά χρόνια ακόμη.

### Η βιομηχανία των παιχνιδιών

Αυτή η αύξηση της ζήτησης για ηλεκτρονικά παιχνίδια, εκφράζεται και στην οικονομία, όπου η θέση που κατέχουνε, μόνο αμελητέα δεν είναι. Τα ηλεκτρονικά παιχνίδια στη σημερινή εποχή, αποτελούν μία από τις ισχυρότερες και εμπορικότερες βιομηχανίες της σύγχρονης ζωής. Το λογισμικό ψυχαγωγίας είναι μάλιστα μια από τις πιο γρήγορα αναπτυσσόμενες βιομηχανίες στην αμερικάνικη οικονομία. [esa2009]. Από τη χρονιά 2005-2006 ο πραγματικός ρυθμός ανάπτυξης της βιομηχανίας στην Αμερική ήτανε περισσότερο από διπλάσιος από το πραγματικό ρυθμό ανάπτυξης ολόκληρης της οικονομίας. Είναι εντυπωσιακό ότι αυτή η ανάπτυξη δεν μειώθηκε ούτε εν καιρώ οικονομικής κρίσης [Guzder2008].

#### Οι πωλήσεις της βιομηχανίας

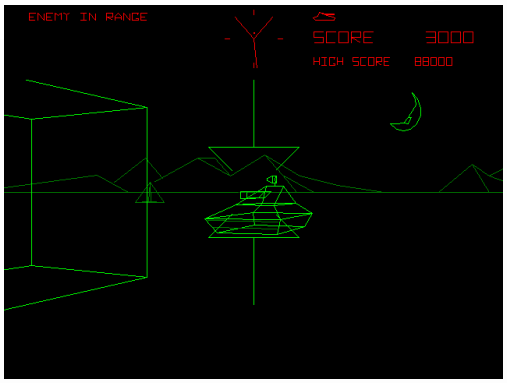
Το 1996, οι αμερικάνικες εταιρίες παραγωγής λογισμικού ψυχαγωγίας πούλησαν μόλις 74.1 εκατομμύρια μονάδες με συνολικό κέρδος 2.6 δισεκατομμύρια δολάρια. Δώδεκα χρόνια μετά, οι εταιρίες υπολογιστών και ηλεκτρονικών παιχνιδιών πούλησαν 298 εκατομμύρια μονάδες, οδηγούμενες σε ένα εκπληκτικό νούμερο εσόδων των 11.7 δισεκατομμυρίων δολαρίων από την παραγωγή λογισμικού, και 22 δισεκατομμύρια δολάρια συνολικά.

Η βιομηχανία των ηλεκτρονικών παιχνιδιών επίσης επηρεάζει θετικά και τις πωλήσεις συμπληρωματικών προϊόντων που έχουνε έσοδα περίπου 6.1 δισεκατομμύρια δολάρια. Για παράδειγμα, τα ηλεκτρονικά παιχνίδια υποκινούν και τη ζήτηση για HDTV προϊόντα. Περίπου 73 εκατομμύρια δολάρια από πωλήσεις HDTV προϊόντων μπορούνε να αποδοθούν άμεσα στην κονσόλα XBox 360 και μόνο. Επιπλέον, η βιομηχανία λογισμικού ψυχαγωγίας επίσης συνεχίζει να λειτουργεί ως μια ζωτική πηγή εργασίας. Οι εταιρίες υπολογιστών και ηλεκτρονικών παιχνιδιών απασχολούν, στην Αμερική μόνο, άμεσα και έμμεσα περισσότερους απόν 80.000 εργαζομένους. Ο μέσος όρος εισοδήματος για τους άμεσα εργαζόμενους είναι 93.000 δολάρια, ενώ όλοι οι εργαζόμενοι αθροιστικά αντιστοιχούν σε 2.2 δισεκατομμύρια δολάρια. Επιπλέον η συνεχής εξέλιξη των ηλεκτρονικών παιχνιδιών προωθεί και την γενικότερα τεχνολογική εξέλιξη, καθώς οι απαιτήσεις των παιχνιδιών σε υλικό, αυξάνονται. Αυτό ίσως να δυσαρεστεί τους καταναλωτές, αλλά σίγουρα προάγει σε κάποια βαθμό την τεχνολογία.

## 2.3. Παιχνίδια με 3D γραφικά

Η γνώση τριγωνομετρίας σε συνδυασμό με απλά μαθηματικά είναι αρκετά ώστε να μπορέσουν να αναπαρασταθούν σε ένα χαρτί, ή σε μία οθόνη, γεωμετρικά σχήματα [Edge2008]. Κάθε σχήμα αναπαρίσταται από μια σειρά σημείων, με κάθε ένα από τα σημεία να περιγράφεται από δύο αριθμούς: την οριζόντια θέση, x, και την κατακόρυφη θέση, y. Στα μαθηματικά, τα σημεία αυτά ονομάζονται δισδιάστατα διανύσματα. Τα σχήματα δημιουργούνται ενώνοντας αυτά τα διανύσματα.

Αν τα γραφικά με δισδιάστατα διανύσματα αναπαρίστανται με δύο αριθμούς για κάθε σημείο επίπεδο, τότε τα τρισδιάστατα γραφικά απαιτούν τρεις, με έναν επιπλέον αριθμό να αναπαριστά το βάθος (τη z θέση). Τα μαθηματικά που απαιτούνται ώστε να μπορέσουνε να προβληθούνε τρισδιάστατα γραφικά σε ένα δισδιάστατο επίπεδο υπάρχουν εδώ και αιώνες, άρα ήταν αναπόφευκτο ότι κάποια στιγμή οι υπολογιστές θα προβάλανε τρισδιάστατα γραφικά. Και αυτό τα γινόταν όταν το υλικό γινόταν τόσο δυνατό ώστε να μπορέσει να διαχειριστεί και να προβάλει αρκετά τρισδιάστατα σημεία ώστε να αναπαριστούν ένα κόσμο παιχνιδιού. Και έτσι, το 1980, το *Battlezone* της Atari ήταν το πρώτο αληθινά τρισδιάστατο παιχνίδι.




**Εικόνα 2.4 - Battlezone, το πρώτο τρισδιάστατο παιχνίδι**

Τα αντικείμενα αναπαρίστανται με απλά πράσινα περιγράμματα των διανυσμάτων που τα αποτελούσαν. Αν και υπήρχε ειδικός co-processor για τις μαθηματικές πράξεις, αυτός είχε περιορισμό στο πόσα σημεία μπορούσε να χειριστεί σε κάθε frame.

## 2.4. Παιχνίδια στρατηγικής

### Διαφορά μεταξύ real-time και turn-based

Τα ηλεκτρονικά παιχνίδια στρατηγικής διακρίνονται σε δύο είδη: Τα παιχνίδια στρατηγικής πραγματικού χρόνου (RTS, real-time strategy games) και παιχνίδια στρατηγικής βασισμένα σε σειρά (TBS, turn-based strategy games).

Η βασική διαφορά βρίσκεται στο κατά πόσο ο τρόπος παιχνιδιού είναι συνεχής. Στα real-time παιχνίδια στρατηγικής ο παίκτης δεν χρειάζεται να περιμένει να τελειώσει η σειρά του αντιπάλου για να παίξει, αλλά και αυτός και ο αντίπαλός του παίζουν ταυτόχρονα και σε πραγματικό χρόνο. Αντίθετα στα turn-based παιχνίδια στρατηγικής κάθε παίκτης παίζει μόνο όταν έρθει η σειρά του, και οι παίκτες δεν μπορούν να παίξουν ταυτόχρονα. Αυτή διαφορά γεννά και άλλες διαφορές στα δύο αυτά είδη.

Μια βασική διαφορά είναι στην στρατηγική που θα πρέπει να ακολουθήσει ένας παίκτης κατά το παιχνίδι [Walker2002]. Στα real-time παιχνίδια στρατηγικής, ο παίκτης σκέφτεται πράγματι την στρατηγική που θέλει να εφαρμόσει, και θα πρέπει να έχει μια πιο γενική οπτική. Για παράδειγμα, στο παιχνίδι *Civilization III* ο τελικός στόχος, επηρεάζει την στρατηγική. Ο παίκτης μπορεί να κατακτήσει τον κόσμο χτίζοντας έναν ισχυρό και πολυάριθμο στρατό, ή μπορεί να επενδύσει σε τομείς όπως η τεχνολογία. Στα turn-based παιχνίδια στρατηγικής, αυτό που ο παίκτης έχει στο μυαλό του δεν είναι τόσο η γενική στρατηγική, όσο η τακτική. Τον ενδιαφέρει πώς θα μπορέσει να νικήσει κάθε μάχη ξεχωριστά. Σε γενικές γραμμές, καθώς υπάρχουν και εξαιρέσεις, ισχύει ότι τα real-time παιχνίδια στρατηγικής δίνουν περισσότερο έμφαση στην στρατηγική παρά στη τακτική, ενώ το αντίστροφο ισχύει με τα turn-based παιχνίδια στρατηγικής. Ίσως ο λόγος που τα παιχνίδια επικεντρώνονται λιγότερα στην τακτική, είναι η τεχνητή νοημοσύνη που έχουν.

Ο επεξεργαστής του μηχανήματος που τρέχει ένα real-time παιχνίδι στρατηγικής, πρέπει να εκτελέσει αρκετές διεργασίες. Η οθόνη χρειάζεται συνεχή ανανέωση, οι ήχοι θα πρέπει να επεξεργαστούν, και να παρθούν αποφάσεις σχετική με την θέση, την κίνηση, την κατασκευή, την επίθεση κ.ά. κάθε μονάδας, σε πραγματικό χρόνο. Δεν υπάρχει παύση προτού παίξει ο επεξεργαστής. Γι' αυτό απαιτείται αρκετή υπολογιστική ισχύς. Από την άλλη μεριά, τα turn-based παιχνίδια στρατηγικής είναι αρκετά ευκολότερα να τα διαχειριστεί ένας υπολογιστής. Οπότε είναι λογικό αυτού του είδους τα παιχνίδια να έχουνε και καλύτερη τεχνητή νοημοσύνη. Με άλλα λόγια, ένας υπολογιστής που τρέχει ένα turn-based παιχνίδι, έχει περισσότερο χρόνο να «σκεφτεί». Επιπλέον, στα real-time παιχνίδια ο τομέας των γραφικών αποτελεί σημαντικό παράγοντα πώλησης, και γι' αυτό ο τομέας της τεχνητής νοημοσύνης δεν έχει την υψηλότερη προτεραιότητα. Για τους παραπάνω λόγους, αν κάποιος θέλει ένα παιχνίδι στρατηγικής με καλύτερη τεχνητή νοημοσύνη, θα πρέπει να επιλέξει κάποιο από την κατηγορία των turn-based παιχνιδιών στρατηγικής. Εξάλλου για αυτό το λόγο υπάρχει και αυτός ο διαχωρισμός στον τρόπο παιχνιδιού. Ένας άνθρωπος δεν θα μπορούσε να χρησιμοποιεί το mouse για να κινεί μονάδες ή να παίρνει αποφάσεις τόσο γρήγορα και να συναγωνίζεται ένα γρήγορο υπολογιστή με καλή τεχνητή νοημοσύνη.

## Ιστορία των παιχνιδιών στρατηγικής

Τα πρώτα turn-based παιχνίδια στρατηγικής έχουνε τις ρίζες τους σε επιτραπέζια παιχνίδια, όπως το Risk. Η αρχή έγινε το 1980 με το Ambush στο οποίο λάμβαναν χώρα μάχες σώμα με σώμα σε σκηνικό β' παγκοσμίου πολέμου. Τα περισσότερα παιχνίδια στρατηγικής που ακολούθησαν κατά τη δεκαετία του '80 έχουνε παρόμοιο θέμα. Η άνοδος της βιομηχανίας των παιχνιδιών στη δεκαετία του '90 επηρέασε και τα turn-based παιχνίδια στρατηγικής. Το 1991 έκανε την εμφάνισή του το *Civilization*, μια σειρά παιχνιδιών που συνεχίζεται ακόμα και είναι από τις κορυφαίες στο είδος. Δύο άλλα σπουδαία παιχνίδια του είδους, το Heroes of Might and Magic (1995) και *Panzer General* (1994), έκαναν χρήση εξάγωνου χάρτη. Ειδική μνεία θα πρέπει να γίνει στο παιχνίδι *Battle for* Wesnoth (http://www.wesnoth.org), ένα παιχνίδι ανοιχτού κώδικα που αναπτύσσεται από το 2003 και συμπεριλαμβάνει και διαδυκτιακές μάχες.

**Εικόνα 2.5 - Battle for Wesnoth**


**Εικόνα 2.6 - Dune II**

Για την κατηγορία των real-time παιχνιδιών στρατηγικής, την αρχή έκανε το *Dune II* (1992) [TDA2008]. Έπειτα συνέχισαν τα *Warcraft* (1994) και *Command & Conquer* (1995). Το Warcraft II (1995) διέδωσε ακόμα περισσότερο το είδος έχοντας και πολύ γραφικά και multiplayer επιλογή. Το 1998, η Blizzard κυκοφόρησε το, κατά γενική ομολογία, καλύτερο παιχνίδι του είδους, *Starcraft*, το οποίο ακόμα και σήμερα θεωρείται αξεπέραστο. Το *Black & White* (2001) υπήρξε επαναστατικό για το είδος έχοντας πολύ καλή τεχνητή νοημοσύνη.

Σήμερα, τα παιχνίδια στρατηγικής έχουνε μεγάλο μερίδιο στις πωλήσεις των ηλεκτρονικών παιχνιδιών, με τα real-time παιχνίδια να υπερισχύουνε των turn-based παιχνιδιών. Για την ακρίβεια, τα real-time παιχνίδια στρατηγικής πουλάνε περίπου τέσσερις φορές περισσότερο από τα αντίστοιχα turn-based παιχνίδια.

# 3. Λογισμικό για τη δημιουργία παιχνιδιών

## 3.1. Συμβολή από την πλευρά των 3D γραφικών

### Game Engine

Όποιος παρακολουθεί την βιομηχανία παιχνιδιών, θα έχει ακούσει πολλές φορές τον όρο game engine. Το τί ακριβώς είμαι μια game engine, δεν είναι απολύτως σαφές. Και αυτό γιατί υπάρχει μια ασαφής γραμμή ανάμεσα στο που τελειώνει η game engine ενός παιχνιδιού και που αρχίζει το περιεχόμενο του παιχνιδιού. Όπως είναι και ασαφές το αν ο κλιματισμός ενός αυτοκινήτου, αποτελεί μέρος του. Γενικά πάντως η έννοια μιας game engine είναι σχετικά απλή: Υπάρχει για να αφαιρεί τις λεπτομέρειες των κοινών λειτουργιών που σχετίζονται με τα παιχνίδια, όπως το rendering, η φυσική, το feedback από τις συσκευές, έτσι ώστε οι δημιουργεί παιχνιδιών να μπορούν να επικεντρωθούν στις λεπτομέρειες που κάνουν το παιχνίδι τους μοναδικό [Ward2008].

Οι μηχανές αυτές προσφέρουν συστατικά το οποία μπορούν να χρησιμοποιηθούν πολλές φορές και μπορεί να διαχειριστούνε έτσι ώστε να ζωντανέψουνε ένα παιχνίδι. Μερικά από αυτά τα συστατικά θα μπορούσανε να είναι η φόρτωση, η κίνηση και το rendering μοντέλων, ανίχνευση σύγκρουσης (collision detection) ανάμεσα σε αντικείμενα, φυσική, είσοδος από το χρήστη, GUI, ακόμα και κομμάτια της τεχνικής νοημοσύνης ενός παιχνιδιού. Αντίθετα, τα ίδια τα μοντέλα και τα textures, το νόημα πίσω από το collision detection και την είσοδο, το πώς τα αντικείμενα αλληλεπιδρούν μεταξύ τους, είναι αυτά που κάνουνε το πραγματικό παιχνίδι.

Μερικές διάσημες σύγχρονες μηχανές παιχνιδιών είναι οι Unreal Engine, id Tech, Cry Engine και Source.

### Graphic Engines

Συχνά ο όρος game engine συγχέεται με τις graphics engines. Η διαφορά είναι ότι μια graphic engine ασχολείται αποκλειστικά με το rendering και την παρουσίαση στην οθόνη, ενώ η game engine περιέχει graphic engine ανάμεσα στα συστατικά της. Μερικές open source graphic engines είναι οι Ogre, Irrlicht και JME.

### Βιβλιοθήκες

Ένα παιχνίδι, ειδικά όταν πρόκειται για λιγότερο εμπορικό, πιθανότατα να μην χρησιμοποιεί κάποια έτοιμη μηχανή, αλλά να κάνει χρήση βιβλιοθηκών και APIs. API (Application Programming Interface), στην βιομηχανία των παιχνιδιών αλλά και γενικά φυσικά, είναι το πρόγραμμα διεπαφής που οι διάφορες βιβλιοθήκες παρέχουν ώστε να μπορούν οι προγραμματιστές να χρησιμοποιήσουν τις λειτουργίες τους. Υπάρχει μια πληθώρα βιβλιοθηκών που μπορούν να χρησιμοποιηθούν στον τομέα, όπως βιβλιοθήκες που χειρίζονται επεξεργασία ήχου, βιβλιοθήκες για γραφικά, για τεχνητή νοημοσύνη κ.α.

Τα graphic APIs είναι ίσως τα πιο σημαντικά σε αυτή τη κατηγορία. Πρόκειται για τα APIs που χρησιμοποιούν τα παιχνίδια για να ζωγραφίσουν οτιδήποτε στην οθόνη. Άρα αυτά καθορίζουν και την εμφάνιση των παιχνιδιών. Τα δύο μεγάλα αντίπαλα δέη σε αυτή την κατηγορία, είναι η OpenGL και το Direct3D.

Η OpenGL είναι ένα ανοιχτού στάνταρντ API που παρέχει μια πληθώρα συναρτήσεων για rendering 2D και 3D γραφικών και είναι διαθέσιμη στα περισσότερα λειτουργικά συστήματα. Το Direct3D, πρόκειται για το API που κάνει χρήση της βιβλιοθήκης DirectX. Αναπτύσσεται από τη Microsoft και είναι συμβατό μόνο με Microsoft Windows.

### 3.1.1. OpenGL

Η OpenGL ορίζεται αυστηρά ως "ένα πρόγραμμα διεπαφής στο υλικό(hardware) των γραφικών" [Wright2007]. Ουσιαστικά, πρόκειται για μια βιβλιοθήκη 3D γραφικών και μοντέλων, άκρως μεταφέρσιμη και πολύ γρήγορη. Χρησιμοποιώντας την OpenGL μπορούμε να δημιουργήσουμε όμορφα 3D γραφικά με εξαιρετική οπτική ποιότητα. Το κυριότερο πλεονέκτημα της χρήσης της OpenGL, είναι ότι είναι πολύ πιο γρήγορη από ένα ray tracer ή ένα rendering πρόγραμμα. Αρχικά, η OpenGL χρησιμοποιούσε αλγόριθμους προσεχτικά ανεπτυγμένους και βελτιστοποιημένους από την Silicon Graphics, Inc. (SGI), ένας αναγνωρισμένος παγκόσμιος ηγέτης στον τομέα των γραφικών και του animation. Με τον καιρό, η OpenGL έχει εξελιχθεί καθώς επιπλέον πωλητές έχουνε συνεισφέρει τη γνώση τους

και την πνευματική τους ιδιοκτησία ώστε να αναπτύξουν δικές του υψηλής απόδοσης εφαρμογές.

Η OpenGL δεν είναι μια γλώσσα προγραμματισμού όπως η C ή η C++. Μοιάζει πιο πολύ με runtime βιβλιοθήκη της C, η οποία παρέχει κάποια prepackaged λειτουργικότητα. Στην πραγματικότητα δεν υπάρχει αυτό που μπορεί να λέμε ως «πρόγραμμα OpenGL» (εκτός ίσως στην περίπτωση των shaders, που θα εξηγηθεί αργότερα) αλλά στην ουσία πρόκειται για ένα πρόγραμμα που έγραψε κάποιος προγραμματιστής και «τυχαίνει» να χρησιμοποιεί την OpenGL ως ένα από τα APIs του. Μπορούμε να χρησιμοποιούμε τη C runtime βιβλιοθήκη για να έχουμε πρόσβαση σε ένα αρχείο, και μπορούμε να χρησιμοποιούμε την OpenGL για να δημιουργήσουμε 3D γραφικά πραγματικού χρόνου. Η OpenGL προορίζεται για τη χρήση υλικού υπολογιστή που είναι σχεδιασμένο και βελτιστοποιημένο για την εμφάνιση και μεταχείριση 3D γραφικών. Υλοποιήσεις OpenGL προγραμμάτων είναι επίσης εφικτές, και σε αυτή τη κατηγορία ανήκουν οι παλιότερες υλοποιήσεις της Microsoft και της Mesa3D (www.mesa3d.org). Η Apple επίσης φτιάχνει ένα τέτοιο πρόγραμμα, διαθέσιμο στο OS X. Με αυτές τις εφαρμογές, το rendering μπορεί να μην εκτελείται τόσο γρήγορα, και μερικά προηγμένα ειδικά εφέ μπορεί να μην είναι καν διαθέσιμα. Ωστόσο, κάνοντας χρήση μιας υλοποίησης προγράμματος σημαίνει ότι το τελικό πρόγραμμα θα τρέχει δυνητικά σε μια ποικιλία από συστήματα υπολογιστών που μπορεί να μην έχουνε κάποια κάρτα επιτάχυνσης 3D γραφικών εγκατεστημένη.


**Εικόνα 3.1 - Ένας απλός κύβος με texture και φωτισμό, φτιαγμένος με OpenGL**

Η OpenGl χρησιμοποιείται για διάφορους σκοπούς, από την κατασκευή CAD και αρχιτεκτονικών εφαρμογών, μέχρι προγράμματα κατασκευής μοντέλων που χρησιμοποιούνται για να δημιουργούν τέρατα σε ταινίες. Η εισαγωγή ενός 3D API βιομηχανικού στάνταρ σε ευρείας αγοράς λειτουργικά συστήματα όπως το Microsoft Windows και το Macintosh OS X έχει ενδιαφέρον αντίκτυπο. Με την επιτάχυνση από το υλικό του υπολογιστή και τους γρήγορους PC μικροεπεξεργαστές να γίνονται όλο και πιο συνήθεις, τα 3D γραφικά είναι πλέον τυπικά χαρακτηριστικά των καταναλωτικών και επιχειρηματικών εφαρμογών, και όχι αποκλειστικά παιχνίδια και επιστημονικές εφαρμογές.

## Η εξέλιξη ενός στάνταρ

Ο προπομπός της OpenGL ήταν η IRIS GL, της Silicon Graphics. Αρχικά μια βιβλιοθήκη 2D γραφικών, εξελίχθηκε σε ένα 3D προγραμματιστικό API για τους τελευταίας τεχνολογίας IRIS σταθμούς εργασίας γραφικών της ίδιας της εταιρίας. Αυτοί οι υπολογιστές είναι κάτι παραπάνω από απλοί γενικής χρήσης υπολογιστές. Είχανε ειδικό υλικό, βελτιστοποιημένο για την εμφάνιση υπερσύγχρονων γραφικών. Το υλικό παρείχε υπερβολικά γρήγορους μετασχηματισμούς πινάκων(ένα προαπαιτούμενο για τα 3D γραφικά), υλικό για υποστήριξη depth buffering, και άλλα χαρακτηριστικά. Μερικές φορές ωστόσο, η εξέλιξη της τεχνολογίας παρακωλύεται από την ανάγκη υποστήριξης παλαιών συστημάτων. η IRIS GL δεν είχε σχεδιαστεί από την αρχή να έχει διεπαφή vertex-style γεωμετρικής επεξεργασίας, και έγινε εμφανές ότι για να προχωρήσει μπροστά, η SGI χρειαζόταν να κάνει ένα καινούριο, καθαρό ξεκίνημα.

Η OpenGL, είναι το αποτέλεσμα των προσπαθειών της SGI, να εξελίξει και να βελτιώσει τη

μεταφερσιμότητα της IRIS GL. Το νέο API γραφικών θα προσέφερε τη δύναμη της GL αλλά θα ήταν ένα ανοιχτό στάνταρ, με εισαγωγή από άλλους πωλητές υλικών γραφικών, και θα επέτρεπε ευκολότερη προσαρμογή σε άλλες πλατφόρμες υλικών και λειτουργικά συστήματα. Η OpenGL θα σχεδιαζόταν από την αρχή για 3D γεωμετρική επεξεργασία.

### Το OpenGL ARB

Ένα ανοιχτό στάνταρ δεν είναι πραγματικά ανοιχτό αν μόνο ένας πωλητής το ελέγχει. Η ασχολία της SGI εκείνη την εποχή ήταν τα γραφικά τελευταίας τεχνολογίας υπολογιστών. Όταν βρίσκεσαι στην κορυφή, καταλαβαίνεις πως οι δυνατότητες για επέκταση είναι σχετικά περιορισμένες. Η SGI συνειδητοποίησε ότι θα ήταν επίσης θετικό για την εταιρία να κάνει κάτι καλό για την βιομηχανία ώστε να βοηθήσει να επεκταθεί η αγορά για υλικά τελευταίας τεχνολογίας γραφικών. Ένα πραγματικά ανοιχτό στάνταρ αποδεκτό από έναν αριθμό πωλητών θα το έκανε πιο εύκολο για τους προγραμματιστές να δημιουργήσουν εφαρμογές και περιεχόμενο που είναι διαθέσιμο σε μια μεγαλύτερη ποικιλία πλατφόρμων. Τα προγράμματα είναι στην ουσία αυτά που πουλάνε τους υπολογιστές, και αν η SGI ήθελε να πουλήσει περισσότερους υπολογιστές, χρειαζόταν περισσότερε προγράμματα που θα έτρεχαν στους υπολογιστές της. Και άλλοι πωλητές το κατάλαβαν αυτό, και έτσι το OpenGL Architecture Review Board (ARB) γεννήθηκε.

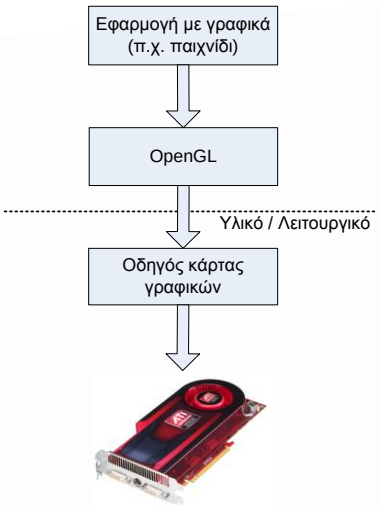
Παρόλο που η SGI αρχικά έλεγχε την αδειοδότηση του OpenGL API, τα ιδρυθέντα μέλη του OpenGL ARB ήτανε οι SGI, Digital Equipment Corporation, IBM, Intel, και Microsoft. Την 1η Ιουλίου 1992, η έκδοση 1.0 των τεχνικών προδιαγραφών της OpenGL παρουσιάστηκε. Με τον καιρό, το ARB μεγάλωνε και περιλάμβανε όλο και περισσότερα μέλη, πολλά από την κοινότητα του υλικού PC, και συναντιόταν τέσσερις φορές το χρόνο για να συντηρήσει και να βελτιώσει τις προδιαγραφές και να κάνει σχέδια για την προώθηση του OpenGL στάνταρ.

Με την πάροδο του χρόνου, η περιουσία της επιχείρησης της SGI ολοένα και μειωνόταν, για διάφορους λόγους. Το 2006, μια ουσιαστικά χρεοκοπημένη SGI μετέφερε τον έλεγχο του OpenGL στάνταρ από το ARB σε μια νέα ομάδα εργασία στο The Khronos Group (www.khronos.org). Το Khronos Group είναι ένας βιομηχανικός συνεταιρισμός ιδρυμένος από διάφορα μέλη, επιφορτισμένο με την δημιουργία και τη συντήρηση στάνταρ ανοιχτών μέσων. Τα περισσότερα μέλη του ARB ήταν ήδη μέλη του Khronos, και η μετάβαση ήταν ουσιαστικά ανώδυνη. Σήμερα, το Khronos Group συνεχίζει να εξελίσσεται και να προωθεί την OpenGL και το αδερφικό του API, OpenGL ES. Η τελευταία έκδοση OpenGL τη στιγμή που γράφεται αυτή η εργασία δημοσιεύτηκε στις August 3, 2009 και είναι η OpenGL 3.2.

Η OpenGL υπάρχει σε δύο μορφές. Το βιομηχανικό στάνταρ κωδικοποιείται μέσα στις προδιαγραφές του OpenGL. Οι προδιαγραφές αυτές περιγράφουν το OpenGL με πολύ ολοκληρωμένους και συγκεκριμένους όρους. Το API είναι πλήρως ορισμένο, όπως και ολόκληρη η μηχανή καταστάσεων(state machine) και το πώς τα διάφορα χαρακτηριστικά λειτουργούν και συνεργάζονται μεταξύ τους. Οι πωλητές υλικών, όπως η ATI, η NVIDIA ή η Apple παίρνουνε έπειτα αυτές τις προδιαγραφές και τις υλοποιούν. Αυτή η υλοποίηση λοιπόν, είναι η ενσωμάτωση του OpenGL σε ένα τύπο που οι προγραμματιστές και οι πελάτες χρησιμοποιούν ώστε να παράγουν γραφικά πραγματικού χρόνου. Για παράδειγμα, ένας πρόγραμμα οδηγού(software driver) και μια κάρτα γραφικών σε έναν υπολογιστή συνιστούν μαζί μια OpenGL υλοποίηση.

### Πώς λειτουργεί η OpenGL

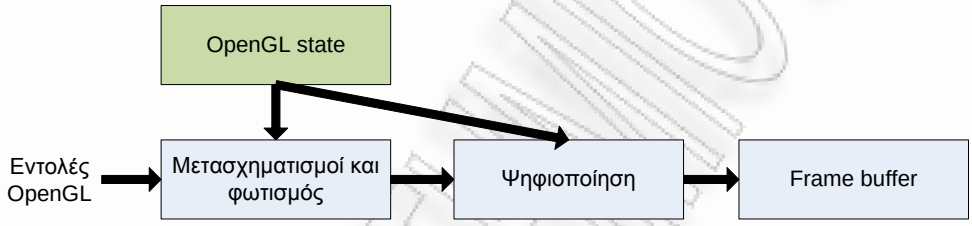
Η OpenGL προσφέρει πάνω από 300 εντολές για δημιουργία γραφικών και δρα σαν ένα ενδιάμεσο στρώμα ανάμεσα στην εφαρμογή και στη κάρτα γραφικών που θα αναλάβει τα απεικονίσει τα γραφικά στην οθόνη, κρύβοντας λεπτομέρειες υλοποίησης του υλικού και των οδηγών του [Αναγνώστου2008].




**Εικόνα 3.2 - Η OpenGL ως ενδιάμεσο στρώμα ανάμεσα στην εφαρμογή και την κάρτα γραφικών**

### OpenGL graphics pipeline

Το παρακάτω σχήμα περιγράφει αρκετά απλοϊκά τα βήματα που ακολουθεί η OpenGL από τη στιγμή που θα πάρει τις εντολές από την εφαρμογή μέχρι την δημιουργία της τελικής εικόνας (rendered image). Η ακολουθία αυτή ονομάζεται Graphics (ή Rendering) Pipeline.




**Εικόνα 3.3 – OpenGL Rendering Pipeline**

Η OpenGL χρησιμοποιεί μια μηχανή καταστάσεων (state machine) για να επικοινωνεί με την εφαρμογή. Σε αυτή την μηχανή καταστάσεων η OpenGL παραμένει διαρκώς σε μια κατάσταση μέχρι να αλλάξει η εφαρμογή την κατάσταση. Για παράδειγμα, αν θέσουμε το χρώμα που θα χρησιμοποιεί η OpenGL για να ζωγραφίσει ένα μοντέλο, το χρώμα θα παραμείνει στη μνήμη και θα χρησιμοποιείται μέχρι να το αλλάξουμε ή να κλείσουμε την εφαρμογή.

Εφόσον λοιπόν η εφαρμογή καθορίσει το περιβάλλον που θα χρησιμοποιήσει η OpenGL για να απεικονίσει το μοντέλο μας (χρώματα, υφές, πηγές φωτός, κάμερα κλπ), περνάει στο πρώτο στάδιο κατά το οποίο θα μετασχηματίσει και θα φωτίσει (transform and lighting) τα σημεία(vertices) του μοντέλου.

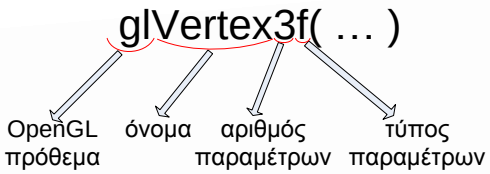
Στην συνέχεια η OpenGL περνά στο στάδιο της ψηφιοποίησης το οποίο λαμβάνει όλες τις πληροφορίες και την γεωμετρία από το προηγούμενο στάδιο (του μετασχηματισμού) και παράγει την τελική, ψηφιακή, εικόνα. Η εικόνα αντιγράφεται στην μνήμη του frame buffer και φτάνει έτσι στην οθόνη του υπολογιστή.

### Εντολές OpenGL/GLUT

*Τι είναι το GLUT*

Η OpenGL περιέχει εντολές για rendering, αλλά είναι σχεδιασμένη ώστε να είναι ανεξάρτητη από οποιοδήποτε λειτουργικό σύστημα, ή διαχειριστή παραθύρων(window manager). Κατά συνέπεια, δεν περιέχει εντολές που να ανοίγουν παράθυρα ή να διαβάζουνε την είσοδο από το ποντίκι ή το πληκτρολόγιο. Όμως είναι αδύνατον να γραφτεί ένα πλήρες πρόγραμμα γραφικών χωρίς τουλάχιστον να ανοιχτεί ένα παράθυρο, μέσα στο οποίο θα παρουσιαστούν τα γραφικά. Για το λόγο αυτό η OpenGL περιέχει και τη βιβλιοθήκη GLUT, που αναλαμβάνει αυτές τις λειτουργίες.

Μια τυπική OpenGL εντολή έχει την παρακάτω μορφή:




**Εικόνα 3.4 - Τυπική OpenGL εντολή**

### Μετασχηματισμοί

Η OpenGL υποστηρίζει μια σειρά μετασχηματισμών τους οποίους μπορούμε να χρησιμοποιήσουμε για να τοποθετήσουμε τα αντικείμενα μας στην οθόνη, να τα περιστρέψουμε, να τα μεγεθύνουμε ή να τα μικρύνουμε.

Ο μετασχηματισμός που τοποθετεί την κάμερα στην κατάλληλη θέση ώστε να «φωτογραφήσουμε» την σκηνή λέγεται μετασχηματισμός κάμερας (viewing transformation).

Οι μετασχηματισμοί που τοποθετούν τα αντικείμενα της σκηνής στην επιθυμητή τους θέση, τα μεγεθύνουνε/σμικρύνουνε και τα περιστρέφουνε ονομάζονται μετασχηματισμοί μοντέλου(modeling transformations). Στην ορολογία της OpenGL, ο μετασχηματισμός μοντέλου είναι ο μετασχηματισμός που τοποθετεί ένα αντικείμενο στο σύστημα συντεταγμένων της σκηνής (κόσμου). Στην γενική ορολογία γραφικών αυτό το μετασχηματισμό το ονομάζουμε μετασχηματισμό κόσμου. Ο μετασχηματισμός που καθορίζει τι είναι ορατό στο σύστημα συντεταγμένων κάμερας (view space) και τι είδους προβολή επιθυμούμε (με προοπτική ή όχι) ονομάζεται μετασχηματισμός προβολής (projection transformation).

Η OpenGL συνδυάζει του πίνακες μετασχηματισμού μοντέλου και μετασχηματισμού κάμερας σε έναν μόνο πίνακα, τον μετασχηματισμό Μοντέλου-Κάμερας (ModelView matrix). Με αυτό το μετασχηματισμό μπορούμε να τοποθετήσουμε ένα αντικείμενο κατευθείαν στο χώρο της κάμερας (χωρίς να περάσουμε από το σύστημα συντεταγμένων κόσμου). Μπορούμε να κατασκευάσουμε τον μετασχηματισμό Μοντέλου-Κάμερας με τέτοιο τρόπο ώστε να περιέχει κάθε περιστροφή, μετακίνηση και κλίμακα του αντικειμένου.

### Μετακίνηση

Η μετακίνηση (translation) ενός αντικειμένου γίνεται με την εντολή

```
glTranslatef(GLfloat X, GLfloat Y, GLfloat Z);
```

Η εντολή αυτή παίρνει ως παραμέτρους την μετακίνηση που επιθυμούμε κατά άξονα, κατασκευάζει ένα μετασχηματισμό μετακίνησης και τον εφαρμόζει (πολ/ζει) με τον μετασχηματισμό Μοντέλου-Κάμερας.

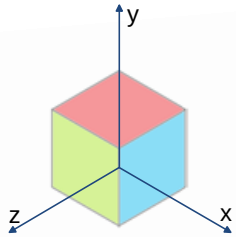
### Περιστροφή

Η περιστροφή (rotation) ενός αντικειμένου γίνεται με την εντολή

```
glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
```

Η εντολή αυτή κατασκευάζει ένα μετασχηματισμό περιστροφής γύρω από το διάνυσμα που ορίζεται από την τριάδα [x, y, z], κατά γωνία angle (μετριέται σε μοίρες 0-360) με φορά αντίθετη των δεικτών του ρολογιού. Αν μας ενδιαφέρει απλά να περιστρέψουμε το αντικείμενο γύρω από ένα άξονα (X, Y ή Z) μπορούμε να ορίσουμε το διάνυσμα ως [1, 0, 0] για το περιστροφή γύρω από τον X, ως [0, 1, 0] για το περιστροφή γύρω από τον Y και ως [0, 0, 1] για το περιστροφή γύρω από τον Z.

Γίνεται προφανές ότι η σειρά με την οποία εκτελούμε τους παραπάνω μετασχηματισμούς, επηρεάζει το τελικό αποτέλεσμα. Αυτό θα γίνει πιο σαφές με το παρακάτω παράδειγμα. Έστω ένα αντικείμενο, με το παρακάτω σύστημα συντεταγμένων:



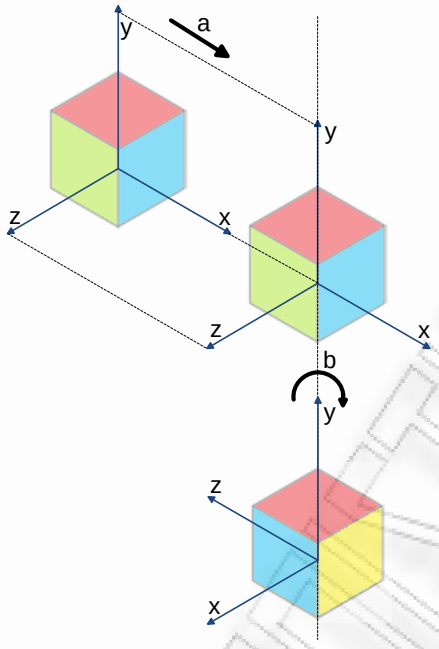

**Εικόνα 3.5 - Αντικείμενο πριν τους μετασχηματισμούς**

Οι μετασχηματισμοί που θα εκτελέσουμε είναι μια μετακίνηση στον x άξονα, και μια περιστροφή στον y άξονα κατά 90º. Δηλαδή:

Σε περίπτωση που πρώτα εκτελέσουμε μετακίνηση και μετά περιστροφή δηλαδή:

```
glTranslatef(100.0, 0.0, 0.0);
glRotatef(-90.0, 0,0, 1.0, 0.0);
```

το αποτέλεσμα θα είναι το εξής:



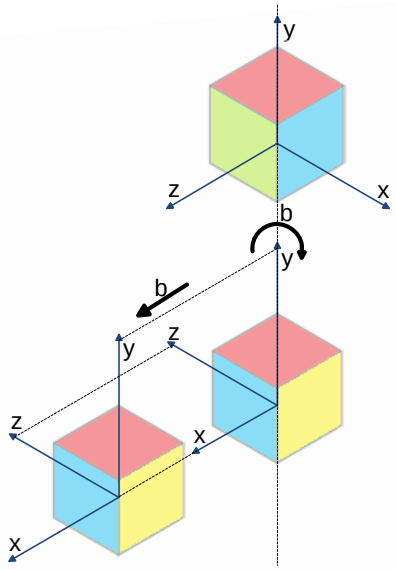

**Εικόνα 3.6 - Αντικείμενο που μετακινείται και περιστρέφεται**

Σε πρώτη φάση(a) το αντικείμενο(ή το τοπικό σύστημα συντεταγμένων) μετακινείται δεξιά 100 pixels, και στη δεύτερη φάση(b) περιστρέφουμε το αντικείμενο δεξιόστροφα κατά 90º.

Στη δεύτερη περίπτωση, οι μετασχηματισμοί εκτελούνται με αντίστροφο τρόπο. Δηλαδή:

```
glRotatef(-90.0, 0,0, 1.0, 0.0);
glTranslatef(100.0, 0.0, 0.0);
```

το αποτέλεσμα θα είναι το εξής:




**Εικόνα 3.7- Αντικείμενο που περιστρέφεται και μετακινείται**

Είναι φανερό ότι η τελική θέση του αντικειμένου διαφέρει στην κάθε περίπτωση.

**Κλίμακα**

Η μεγέθυνση/σμίκρυνση (scaling) ενός αντικειμένου γίνεται με την εντολή

```
glScalef(GLfloat x, GLfloat y, GLfloat z);
```

όπου x, y, z η κλίμακα του αντικειμένου ανά άξονα.

**Τοποθέτηση κάμερας**

Η OpenGL μας δίνει την δυνατότητα να τοποθετήσουμε την κάμερα στην σκηνή και να θέσουμε κατεύθυνση της με την χρήση μόνο μιας εντολής:

```
gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
GLdouble centerx, GLdouble centery, GLdouble centerz,
GLdouble  upx, GLdouble upy, GLdouble upz);
```

Η εντολή αυτή παίρνει ως παραμέτρους 3 διανύσματα το [eyex, eyey, eyez] που ορίζει την θέση της κάμερας στην σκηνή, το [centerx, centery, centery] το οποίο καθορίζει την κατεύθυνση της κάμερας (το σημείο στο οποίο δείχνει δηλαδή) και το διάνυσμα [upx, upy, upz] που καθορίζει ποια είναι η «πάνω» κατεύθυνση της κάμερας.

Για παράδειγμα αν θέλω να ορίσω μια κάμερα που βρίσκεται στο σημείο [10,10,10], «δείχνει/βλέπει» στην αρχή των αξόνων [0,0,0] και η πάνω κατεύθυνση της είναι ο Y άξονας [0 1 0] θα καλέσω την εντολή ως:

```
gluLookAt(10,10,10,
0,0,0
0,1,0);
```

Αυτή η εντολή είναι κυρίως βοηθητική, καθώς η μετακίνηση της κάμερας θα μπορούσε να επιτευχθεί χωρίς αυτήν την εντολή. Αντί να μετακινείται η κάμερα σε σχέση με ένα αντικείμενο, μπορούν να μετακινούνται τα αντικείμενα(το τοπικό σύστημα συντεταγμένων) σε σχέση με αυτήν. Για παράδειγμα, αν θέλουμε να δούμε από πιο κοντά ένα αντικείμενο, αντί να πλησιάσουμε εμείς(η κάμερα) προς το αντικείμενο, φέρνουμε το τοπικό σύστημα συντεταγμένων πιο κοντά σε μας μέσω των μετασχηματισμών.

**Εφαρμογή των μετασχηματισμών στην OpenGL**

Αναφέραμε ότι η OpenGL χρησιμοποιεί μόνο ένα πίνακα, τον ModelView, για να εφαρμόσει όλους τους μετασχηματισμούς μετακίνησης, περιστροφής και κλίμακας που είδαμε. Ο πίνακας αυτός αποθηκεύεται στην μνήμη της OpenGL και παραμένει ο ίδιος μέχρι να τον αλλάξουμε με μια άλλη εντολή. Το σύνολο όλων των μεταβλητών/παραμέτρων που διατηρεί η OpenGL για την λειτουργία της ονομάζεται μηχανή καταστάσεων (state machine). Με αυτό τον τρόπο λειτουργίας μια τιμή που θέτουμε σε μια μεταβλητή παραμένει σε ισχύ μέχρι να την αλλάξουμε ξανά.

Επειδή η OpenGL χρησιμοποιεί πάνω από ένα πίνακες για να ορίσει μετασχηματισμούς πριν αρχίσουμε να κατασκευάσουμε το ModelView μετασχηματισμό πρέπει να τον ενεργοποιήσουμε. Αυτό γίνεται με την εντολή

```
glMatrixMode(GL_MODELVIEW);
```

Η παράμετρος GL_MODELVIEW ορίζει ότι θέλουμε να ενεργοποιήσουμε τον πίνακα ModelView (μια άλλη παράμετρος που θα χρησιμοποιήσουμε είναι η GL_PROJECTION που ορίζει ως ενεργό τον πίνακα προβολής)

Από την στιγμή που ενεργοποιήσουμε τον ModelView πίνακα, κάθε μετασχηματισμός που κατασκευάζουμε με τις εντολές glTranslation, glRotation και glScale εφαρμόζονται, με πολλαπλασιασμό, με τον πίνακα που έχει κάθε στιγμή ο ModelView.

Για να αρχικοποιήσουμε τον ModelView πίνακα χρησιμοποιούμε την εντολή

```
glLoadIdentity();
```

Εξορισμού (αν δεν έχουμε εφαρμόσει κάποιο άλλο μετασχηματισμό) η κάμερα είναι τοποθετημένη στην αρχή των αξόνων και βλέπει προς τον άξονα -Z οποία του φορτώνει τον μοναδιαίο πίνακα. Παράδειγμα:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(10,10,10, 0,0,0, 0,1,0);
glTranslate(0, 0, -6);
glRotatef(45, 0, 1, 0);
DrawOurModel();
```

Αρχικά ενεργοποιούμε τον πίνακα ModelView, τον αρχικοποιούμε με τον μοναδιαίο πίνακα I, κατασκευάζουμε και εφαρμόζουμε τον μετασχηματισμό κάμερας View (gluLookAt), κατασκευάζουμε και εφαρμόζουμε μια μετακίνηση T κατά -6 κατά μήκος του άξονα Z και μια περιστροφή R, 45 μοίρες γύρω από τον άξονα Y.

Ο τελικός μετασχηματισμός ModelView θα είναι :

```
ModelView = R * T * View
```

Σημαντική παρατήρηση : Η OpenGL εφαρμόζει τους μετασχηματισμούς στο αντικείμενο με αντίστροφη σειρά από ότι τους δηλώνουμε στο πρόγραμμα. Στο παραπάνω παράδειγμα το αντικείμενο πρώτα θα περιστραφεί, μετά θα μετακινηθεί και μετά θα μεταφερθεί στο χώρο της κάμερας.

Αν θεωρήσουμε ένα σημείο P του αντικειμένου, τότε ο μετασχηματισμός αυτός θα εφαρμοστεί ως:

```
P' = P * ModelView = P * R * T * View
```

**Στοίβα πινάκων**

Είπαμε ότι ό,τι μετασχηματισμό κατασκευάζουμε με τις εντολές glTranslation, glRotation και glScale συσσωρεύεται (με πολλ/μό) στο πίνακα ModelView, ο οποίος με την σειρά του μετασχηματίζει όλα τα αντικείμενα της σκηνής. Υπάρχουν περιπτώσεις που πριν «προσθέσουμε» ένα νέο μετασχηματισμό στο ModelView (μια περιστροφή που ίσως αφορά ένα μόνο αντικείμενο και όχι όλη τη σκηνή) θέλουμε να τον αποθηκεύσουμε κάπου προσωρινά ώστε μετά να τον ανακτήσουμε αναλλοίωτο από τον νέο μετασχηματισμό. Η OpenGL προσφέρει ένα τέτοιο μηχανισμό και ονομάζεται στοίβα πινάκων (matrix stack). Η στοίβα είναι μια δομή δεδομένων που μας επιτρέπει μέσω εντολών Push και Pop να προσθέσουμε και να αφαιρέσουμε ένα στοιχείο από την κορυφή της στοίβας (και μόνο).

Στην OpenGL οι εντολές που αποθηκεύουν και ανακτούν τον ModelView στην κορυφή της στοίβας είναι οι

```
glPushMatrix();
glPopMatrix();
```

Παράδειγμα, αν θέλουμε να ζωγραφίσουμε 2 αντικείμενα αφού πρώτα τα μετακινήσουμε το πρώτο κατά -6 κατά μήκος του άξονα Z και το δεύτερο κατά 10 κατά τον άξονα Y τότε

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(10,10,10, 0,0,0, 0,1,0);
glPushMatrix();
```

```
glTranslate(0, 0, -6);
DrawModel_1();
glPopMatrix();
glTranslate(0, 10, 0);
DrawModel_2();
```

Στο παράδειγμα αφού ορίσουμε τον μετασχηματισμό κάμερας με την εντολή gluLookAt() αποθηκεύουμε τον ModelView στην στοίβα με την εντολή glPushMatrix(). Έπειτα μετακινούμε το πρώτο αντικείμενο κατά -6 κατά μήκος του άξονα Z και το ζωγραφίζουμε. Στην συνέχεια ανακτούμε από την στοίβα τον ModelView (που δεν περιέχει την μετακίνηση κατά -6) με την εντολή glPopMatrix(). Τέλος μετακινούμε το δεύτερο αντικείμενο κατά 10 κατά τον άξονα Y και το ζωγραφίζουμε.

Αν υποθέσουμε ότι δεν χρησιμοποιούμε την στοίβα για να αποθηκεύσουμε τον ModelView:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(10,10,10, 0,0,0, 0,1,0);
glTranslate(0, 0, -6);
DrawModel_1();
glTranslate(0, 10, 0);
DrawModel_2();
```

τότε το πρώτο αντικείμενο θα μετακινούνταν σωστά κατά -6 κατά μήκος του άξονα Z όμως το δεύτερο αντικείμενο θα μετακινούταν πρώτα κατά -6 κατά μήκος του άξονα Z και έπειτα κατά 10 κατά τον άξονα Y.

Θα εφαρμόζαμε δηλαδή και τις 2 μετακινήσεις στο δεύτερο αντικείμενο αφού όπως είπαμε όλοι οι μετασχηματισμοί που κατασκευάζουμε “συσσωρεύονται” στο πίνακα ModelView.

Η χρήση της στοίβας είναι βασική λοιπόν για την σωστή τοποθέτηση των αντικειμένων στην σκηνή.

**Μετασχηματισμός προβολής**

Με τον μετασχηματισμό προβολής (projection transform) η OpenGL προβάλλει τα 3διάστατα αντικείμενα στις 2 διαστάσεις ώστε να δημιουργήσει την τελική διδιάστατη εικόνα. Η OpenGL υποστηρίζει 2 ειδών μετασχηματισμούς προβολών την ορθογραφική προβολή (orthographic projection) και την προβολή με προοπτική (perspective projection). Μια ορθογραφική προβολή δημιουργείται με την χρήση της εντολής:

```
glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top, GLdouble near, GLdouble far );
```

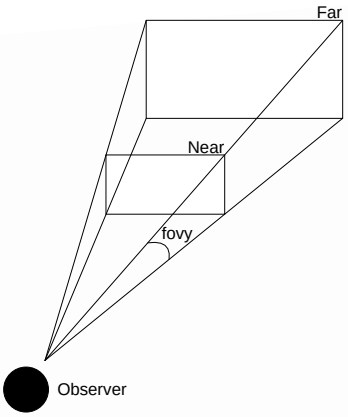
Η εντολή αυτή ορίζει ένα κύβο αποκοπής παρέχοντας τις συντεταγμένες 2 διαγώνιων κορυφών.

Μια προβολή με προοπτική δημιουργείται με την χρήση της εντολής :

```
gluPerspective(GLdouble fovy, GLdouble aspect,
GLdouble zNear, GLdouble zFar);
```

Η παράμετρος fovy ορίζει την γωνία οπτικού πεδίου στην κάθετη κατεύθυνση, η aspect καθορίζει τον αναλογία μήκους/πλάτους του κοντινού πεδίου αποκοπής και οι zNear και zFar την απόσταση του κοντινού (near) και μακρινού (far) πεδίου αποκοπής κατά τον άξονα Z.

Σχηματικά η προβολή με προοπτική ορίζεται ως:




**Εικόνα 3.8- Προβολή με προοπτική**

Η OpenGL αποθηκεύει τον πίνακα προβολής στον πίνακα GL_PROJECTION. Για να τον ενεργοποιήσουμε χρησιμοποιούμε και πάλι την εντολή glMatrixMode()

Παράδειγμα:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45, 4/3, 1, 400);
```

**Ζωγραφίζοντας στην OpenGL**

Για να μπορέσουμε να ζωγραφίσουμε οτιδήποτε, αυτό γίνεται στέλνοντας στην OpenGL τα σημεία της γεωμετρίας μας με την εντολή

```
glVertex3f(GLfloat x, GLfloat y, GLfloat z);
```

Για να ζωγραφίσει τα σημεία αυτά η OpenGL χρειάζεται ακόμα μια πληροφορία: πώς θα τα ενώσει μεταξύ τους ώστε να σχηματιστεί το αντικείμενο. Υπάρχουν πολλές επιλογές: GL_POINTS για σημεία, GL_LINES για ευθύγραμμα τμήματα, GL_TRIANGLES για τρίγωνα και άλλες.

## 3.2. Συμβολή από την πλευρά της Τεχνητής Νοημοσύνης

### 3.2.1. Τεχνητή Νοημοσύνη στα παιχνίδια

Οι γλώσσες και τα εργαλεία που χρησιμοποιούνται για την τεχνητή νοημοσύνη στα παιχνίδια, ποικίλουν. Καταρχάς υπάρχουνε οι αμιγώς AI γλώσσες όπως οι LISP και PROLOG. Κατά πάσα πιθανότητα, παιχνίδια που δημιουργούνται με αυτές τις γλώσσες δεν προορίζονται για εμπορική χρήση. Με πιο συμβατικές γλώσσες, όπως οι C++, και Java, οι AI τεχνικές που θα χρησιμοποιηθούν κατασκευάζονται με αλγόριθμους. Η νοημοσύνη εδώ βρίσκεται στον ίδιο τον αλγόριθμο, και όχι στην γλώσσα. Άλλος τρόπος, είναι με τις γλώσσες 4$^{ης}$ γενιάς. Είναι γλώσσες υψηλού επιπέδου, όπως η Smalltalk, που χρησιμοποιούνε φυσική γλώσσα ή κάποιες πιο αφηρημένες δομές. Σε αυτή τη περίπτωση οι αλγόριθμοι ή οι δομές δεδομένων μπορεί να επιλεχθούν από τον compiler. Τελευταία περίπτωση είναι οι διάφορες βιβλιοθήκες ή μηχανές (AI engines), που χρησιμοποιούνται παράλληλα με κάποια γλώσσα ή εργαλείο.

### Τύποι Τεχνητής Νοημοσύνης

Η χρήση τεχνητής νοημοσύνης σε ένα παιχνίδι, μπορεί να γίνει με διάφορους τρόπους. Παρακάτω παρουσιάζονται μερικές προσεγγίσεις.

**Σταθερές αντιδράσεις**

Αναφέρεται στη χρήσης μιας Μηχανής Διακριτών Καταστάσεων (Finite State Machine, FSM). Θα μπορούσαμε να παρομοιάσουμε το FSM με το διάγραμμα ροής. Το διάγραμμα ροής απεικονίζει ένα σύνολο καταστάσεων που ακολουθούνε κάποιο μονοπάτι. Μια κατάσταση μπορεί να μεταβεί σε άλλες καταστάσεις, οι οποίες προκαλούνται από γεγονότα μέσα στο σε μια κατάσταση [Meyer2003]. Για παράδειγμα, στο κλασσικό παιχνίδι Space Invaders, οι εξωγήινοι κινούνται σε μια σταθερή τροχιά, και πετούνε βόμβες τυχαία. Σε άλλο κλασσικό παιχνίδι, το Pac-man, τα φαντάσματα κινούνται πάντα εναντίον του παίχτη, σύμφωνα με τους

περιορισμούς που θέτουν οι τοίχοι. Η τεχνητή νοημοσύνη του υπολογιστή είτε δεν λαμβάνει καθόλου υπόψη τις ενέργειες του παίχτη (Space Invaders), είτε τον αντιμετωπίζει με πολύ απλοϊκό τρόπο (Pac-man). Αυτό καθιστά και ευκολότερο τον προγραμματισμό, καθώς έννοιες όπως μνήμη (όσον αφορά στις κινήσεις του παίκτη), ή συλλογή δεδομένων, είναι σχεδόν ανύπαρκτες. Αν και πρόκειται για πολύ παλιά παιχνίδια, η τεχνική του FSM χρησιμοποιείται ακόμα και σε σύγχρονα παιχνίδια.

Το προφανές πρόβλημα με αυτές τις μεθόδους, είναι φυσικά ότι δεν είναι αρκετά ικανοποιητικές για τον παίχτη. Ο μόνος τρόπος για να μπορέσει ο υπολογιστής να νικήσει τον παίχτη, είναι είτε να «καλέσει» περισσότερους αντιπάλους, είτε να αυξήσει την ταχύτητα του παιχνιδιού, ή γενικά να προσπαθήσει να εξαντλήσει τον παίχτη, με την ίδια πάντα απλή τεχνητή νοημοσύνη. Αυτό μπορεί να κάνει ένα παιχνίδι ενδιαφέρον ακόμα και σήμερα, αλλά σε καμία περίπτωση δεν κάνει χρήση κάποια ιδιαίτερης «νοημοσύνης».

**Σταθεροί κανόνες**

Το επόμενο επίπεδο πολυπλοκότητας, είναι να δώσουμε στον υπολογιστή «κρίση». Εξυπακούεται ότι δεν πρόκειται για πραγματική, έξυπνη κρίση, αλλά για αποφάσεις που βασίζονται σε κάποιον αλγόριθμο. Ένα τέτοιο παράδειγμα, είναι η απόφαση που μπορεί να πάρει ένα διαστημόπλοιο σε κάποιο παιχνίδι, σχετικά με τον τύπο του όπλου που θα χρησιμοποιήσει, ανάλογα με την απόστασή του από κάποιο αντίπαλο διαστημόπλοιο. Θα μπορούσε ακόμα αυτό το διαστημόπλοιο να αποφασίσει να φύγει μακριά, αποφεύγοντας τη μάχη. Η απόφαση για το ποιό τύπο όπλου θα επιλέξει το διαστημόπλοιο, μπορεί να παρθεί από πίνακες που παράγονται από τον προγραμματιστή, βασιζόμενος στην δική του εμπειρία μέσα στο παιχνίδι, ή σε μαθηματικούς υπολογισμούς, ή σε δική του διαισθητική επιλογή σχετικά με την καταλληλότερη αντίδραση σε διαφορετικές καταστάσεις.

Γενικότερα, ο υπολογιστής θα μπορούσε να επιλέξει ανάμεσα σε εναλλακτικές τακτικές χρησιμοποιώντας έναν αυθαίρετα πολύπλοκο διακριτό αλγόριθμο και μια αυθαίρετα μεγάλη βάση δεδομένων. Αυτό οδηγεί σε μια καλή εξομοίωση νοημοσύνης που θα ικανοποιήσει τους περισσότερους παίκτες. Για παράδειγμα, ένας υπολογιστής που παίζει τάβλι αποφασίζει την επόμενή του κίνηση βασιζόμενος στην τρέχουσα θέση των πουλιών και τις τιμές των ζαριών. Παρομοίως, ένα πρόγραμμα σκάκι χρησιμοποιεί έναν αλγόριθμο που αναθέτει τιμές στα πιόνια και στις θέσεις. Ο υπολογιστής θα καθορίσει την τιμή της κάθε θέσης για μερικές κινήσεις κάθε παίχτη υποθέτοντας (σε αυτό το επίπεδο) τέλειο παιχνίδι από τον άνθρωπο και θα επιλέξει την κίνηση με την μεγαλύτερη τιμή.

Αυτό δεν μπορεί να αποκαλεστεί «νοημοσύνη» από την μεριά του υπολογιστή επειδή ο υπολογιστής απλά υπακούει σύνολα κανόνων όπως αυτά γράφτηκαν από τον προγραμματιστή. Επίσης ο υπολογιστής δεν αναλύει τον τρόπο παιχνιδιού του παίχτη. Η τεχνητή νοημοσύνη ενός υπολογιστή θα μπορούσε, υποθετικά, να αναλύσει τον τρόπο παιχνιδιού του ανθρώπου και να τον χρησιμοποιήσει ως άλλη μια μεταβλητή σε ένα σταθερό αλγόριθμο ή σε lookup πίνακες. Αυτό όμως θα οδηγούσε σε τρομακτική αύξηση του μεγέθους του πίνακα.

Ένας εναλλακτικός τρόπος χρήσης αυτής της μεθόδου είναι με διάφορους αντιπάλους που χειρίζεται ο υπολογιστής, καθένας από τους οποίους έχει διαφορετικό τρόπο παιχνιδιού και πιθανότατα διαφορετικές ικανότητες. Η επιλογή ενός από αυτούς τους αντιπάλους μπορεί να γίνεται από τον άνθρωπο ή τυχαία.

**Lookup πίνακες παραγμένοι από τεχνητή νοημοσύνη**

Όταν η μέθοδος «σταθεροί κανόνες» χαρακτηρίστηκε ως μη έξυπνη, δεν είχε σκοπό να υποβαθμίσει την ποσότητα δουλειάς που χρειάζεται για την ανάπτυξη αλγορίθμων ή lookup πινάκων. Στην περίπτωση του τάβλι για παράδειγμα, ο υπολογιστής έχει να επιλέξει ανάμεσα σε μερικές από δεκάδες πιθανές κινήσεις καθώς και να επιλέξει αν θα πρέπει να παραιτηθεί ή όχι. Ο προγραμματιστής θα μπορούσε να γράψει ένα lookup πίνακα για κάθε πιθανή θέση ενός πουλιού και ρίξιμο των ζαριών, αλλά αυτό απαιτεί μεγάλο αποθηκευτικό χώρο.

Ένα λειτουργικό πρόγραμμα τάβλι αναθέτει βάρη σε διαφορετικούς συνδυασμούς θέσεων και υπολογίζει την τιμή της κάθε θέση στην οποία μπορεί να πάει το πούλι από την τρέχουσα θέση και το ρίξιμο των ζαριών. Ο προγραμματιστής μπορεί να αναπτύξει ένα σύστημα βαρών που να αναθέτει βάρη ανάλογα με την αξία κάθε πιθανής θέσης αλλά αυτό είναι μια πολύ δύσκολα υλοποιήσιμη πρόταση, ακόμα και για παιχνίδια όπως το τάβλι ή το σκάκι. Σε ένα παιχνίδι στρατηγικής με 100 μονάδες σε μια μάχη, με κάθε μονάδα να έχει να δικά της χαρακτηριστικά, με την ιδιαίτερη μορφολογία του χάρτη και τυχαίους παράγοντες να

περιπλέκουν ακόμα πιο πολύ το θέμα, το ιδανικό σύστημα βαρών δεν μπορεί να καθοριστεί από ένα άτομο, ούτε σε λογικά χρονικά πλαίσια και από πολλούς υπολογιστές που δουλεύουνε ταυτόχρονα πάνω σε αυτό.

Η λύση είναι γίνει κάποια συμβιβασμός προς το «αρκετά καλό», παρά προς το «καλύτερο». Αυτό θα μπορούσε να χαρακτηριστεί ως μια μηχανή καταστάσεων Ασαφούς Λογικής (Fuzzy Logic), σε αντίθεση με την Μηχανή Διακριτών Καταστάσεων. Ένας από τους καλύτερους τρόπους να βρεθούν τα «αρκετά καλά» βάρη και αλγόριθμοι είναι να ρυθμιστεί ο υπολογιστής έτσι ώστε να παίζει και με τις δύο πλευρές σε ένα παιχνίδι, με αρκετή ποικιλία στους αλγόριθμους και στα βάρη. Έτσι οι δύο παίχτες που χειρίζεται ο υπολογιστής αναμετρώνται και παρατηρείται ποιος παίχτης κερδίζει πιο συχνά. Έπειτα οι αλγόριθμοι αλλάζουν και το πείραμα συνεχίζεται μέχρις ότου προκύψει το ικανοποιητικό αποτέλεσμα.

Μια μεγαλύτερη βελτίωση της παραπάνω μεθόδου είναι να επιτραπεί στον υπολογιστή να κάνει τις δικές του αλλαγές ώστε να βρει την νικητήρια τεχνική. Ένας βολικός τρόπος για αυτό, είναι με κάποιο πρόγραμμα υψηλότερου επιπέδου από τα άλλα, τα οποία αφορούν τους παίχτες, που προσπαθεί να αναπτύξει ένα σύνολο από νικηφόρων αλγορίθμων και τιμές. Το πρόγραμμα αυτό «πειράζει» τακτικά τα προγράμματα των παικτών για να βρει το βέλτιστο. Αν ο προγραμματιστής πιστεύει πως οι αλγόριθμοι είναι αρκετά καλοί, τότε χρειάζεται να βελτιστοποιηθούνε μόνο τα βάρη που ανατίθενται στην εκάστοτε μονάδα, πούλι η οτιδήποτε. Επιλέγοντας τα βάρη τυχαία δεν είναι ο καλύτερος τρόπος να βρεθεί το καλύτερο σύνολο, αλλά έχει την πιο εύκολο υλοποίηση. Η ανάπτυξη αλγορίθμων και η πιο εκλεπτυσμένη ανάθεση βαρών μπορεί να γίνει με τη χρήση γενετικών αλγορίθμων.

Ο προγραμματιστής ρυθμίζει το πρόγραμμα το οποίο θα προσδιορίσει ένα καλό σύνολο παραμέτρων και το αφήνει να τρέξει για λίγο. Όταν επιστρέψει, κρατάει τους καλύτερους αλγόριθμους και βάρη που παράγονται, τα βάζει στο πρόγραμμα ως

Αυτή η μέθοδος της «Τεχνητής Νοημοσύνης κατά την Ανάπτυξη» έχει τα καλύτερα και τα χειρότερα από τους δύο κόσμους. Στην «χειρότερη» πλευρά της ζυγαριάς, ο προγραμματιστής θα πρέπει να μάθει και να χρησιμοποιήσει πραγματικές AI τεχνικές κατά την ανάπτυξη, αλλά έπειτα δίνει μια σταθερή ρουτίνα για το πραγματικό παιχνίδι. Στην «καλύτερη» πλευρά, ένα πρόγραμμα που έχει γραφτεί χρησιμοποιώντας διαδικαστική γλώσσα και αλγόριθμους θα τρέχει πολύ πιο γρήγορα και χρησιμοποιεί λιγότερους πόρους από ένα πρόγραμμα που χρησιμοποιεί πραγματικές AI τεχνικές.

**Ευέλικτοι αλγόριθμοι**

Σε αυτή τη κατηγορία υπάγονται οι προσαρμοστικοί αλγόριθμοι και πίνακες δεδομένων για παίχτη του υπολογιστή. Αυτές οι μέθοδοι χρησιμοποιούνε αλγόριθμους και βάρη τα οποία προσαρμόζονται όταν τρέχει το πρόγραμμα ώστε να υπάρξουν καλύτερα αποτελέσματα. Για παράδειγμα, σε ένα πολύπλοκο παιχνίδι στρατηγικής ο υπολογιστής μπορεί να ξεκινήσει να χρησιμοποιεί μια συγκεκριμένη μέθοδο μετακίνησης των μονάδων και επίθεσης υπό διαφορετικές συνθήκες. Η ακόμα καλύτερα, αρχικά χρησιμοποιεί μια ποικιλία τεχνικών. Το πρόγραμμα παρακολουθεί την αποτελεσματικότητα του κάθε αλγόριθμου και σύνολο βαρών και σταδιακά αποκλείει τα λιγότερο αποτελεσματικά.

Αυτή η μέθοδος μοιάζει πολύ με αυτήν που περιγράφτηκε παραπάνω, με τη διαφορά ότι η προσαρμογή των αλγορίθμων και βαρών συμβαίνει ενώ το παιχνίδι τρέχει και όχι κατά την ανάπτυξη.

Η χρήση γενετικών αλγόριθμων και άλλων μεθόδων αυτορρύθμισης του προγράμματος θεωρείται από αρκετούς τεχνητή νοημοσύνη.

**Ανάλυση των ανθρώπινων ενεργειών**

Το επόμενο βήμα είναι η ανάλυση των ενεργειών του ανθρώπινου παίχτη. Με αυτήν την έννοια, ανάλυση σημαίνει αναγνώριση, μελέτη και κατηγοριοποίηση των στοιχείων του ανθρώπινου τρόπου παιχνιδιού. Σε αυτό το σημείο, είναι που συμβαίνει η πραγματική έρευνα για τεχνητή νοημοσύνη. Το λιγότερο που χρειάζεται ένα πρόγραμμα σε αυτό το επίπεδο, είναι μεγάλες δυνατότητες αναγνώρισης προτύπων.

Αν και έχει γίνει σημαντική πρόοδος τελευταία στην αναγνώριση προτύπων, αυτή δεν πλησιάζει το επίπεδο του ανθρώπου ή ενός κατοικίδιου. Για παράδειγμα, σε ένα shooting παιχνίδι, αν ο αντίπαλος που χειρίζεται ο υπολογιστής πάντα αποφεύγει τα πυρρά του ανθρώπου από τα αριστερά, ο άνθρωπος λογικά θα το καταλάβει γρήγορα αυτό και θα σημαδεύει γρήγορα ξανά αριστερά. Ένας υπολογιστής από την άλλη μεριά, μπορεί να δει ότι

ο άνθρωπος τις μισές περίπου φορές αριστερά και τις άλλες μισές περίπου δεξιά, γεγονός που δεν δείχνει κάποιο πρότυπο.

Η αναγνώριση προτύπων είναι πολύ πιο αποτελεσματική καθώς ο αριθμός των περιπτώσεων αυξάνει. Σε ένα νευρωνικό δίκτυο, η διαδικασία αυτή μπορεί να συμβαίνει για πάντα, ακόμα και αν η ανάγκη για αποτελέσματα είναι πολύ νωρίτερα από μία αιωνιότητα.

**Επιλογή υποστόχου**

Ο τελικός στόχος χωρίζεται σε επιμέρους υποστόχους, που όταν επιτευχθούν αυτοί, επιτυγχάνεται και αυτός. Για παράδειγμα, ας υποθέσουμε ότι στο διαστημικό shoot-em-up παιχνίδι Space Invaders που αναφέρθηκε και προηγουμένως, ο υπολογιστής ελέγχει τα επίπεδα ενέργειας και τις βασικές ικανότητες του ανθρώπου, και αποφασίζει ότι ο άνθρωπος έχει ένα πλεονέκτημα επειδή έχει περισσότερη ενέργεια αλλά κατά τα άλλα είναι ίσοι. Ο υπολογιστής θα μπορούσε να αποφασίσει τον υποστόχο της εξάντλησης της ενέργειας του ανθρώπου χωρίς να παραχωρήσει κάποιο πλεονέκτημα. Για να το κάνει αυτό, ο υπολογιστής θα μπορούσε να αποφασίσει να επιτεθεί, να προσελκύσει τα πυρρά και να φύγει χωρις να δεχθεί χτύπημα. Οι ενέργειες που θα εκτελούσε ο υπολογιστής για αυτό το σενάριο, θα μπορούσαν να έχουνε προγραμματιστεί προηγουμένως ώστε να συμβούν υπό τις κατάλληλες συνθήκες, θεωρώντας ότι ο προγραμματιστής μπορεί να προβλέψει όλες τις πιθανές περιπτώσεις. Σε ένα κατάλληλο σύστημα AI, ο υπολογιστής θα αναγνώριζε κάπως μια ανάγκη, θα έψαχνε ανάμεσα σε ένα πλήθος πιθανών στόχων και δράσεων, και θα επέλεγε αυτή με τη μεγαλύτερη πιθανότητα επιτυχίας.

# 4. Ανάλυση Απαιτήσεων και Σχεδιασμός του παιχνιδιού

## 4.1. Προδιαγραφές

Οι προδιαγραφές, όπως αυτές ορίστηκαν πριν ξεκινήσει η σχεδίαση του παιχνιδιού, είναι οι παρακάτω:

1. Το παιχνίδι θα πρέπει να είναι ολοκληρωμένο και πλήρως λειτουργικό. Είναι προτιμότερο να προκύψει κάτι ολοκληρωμένο και απλό, από κάτι εξειδικευμένο και ημιτελές.
2. Ο συνολικός χρόνος της ανάπτυξης του παιχνιδιού δεν θα πρέπει να ξεπεράσει το ένα έτος.
3. Θα πρέπει να τρέχει σε λειτουργικά συστήματα Linux και Windows.
4. Οι ελάχιστες απαιτήσεις του παιχνιδιού θα πρέπει να είναι σχετικά χαμηλές.
5. Να υπάρχει δυνατότητα αναβάθμισης.

## 4.2. Ανάλυση απαιτήσεων

### 4.2.1. Επιλογή του παιχνιδιού

Λόγω της προδιαγραφής 2, δηλαδή του σχετικά περιορισμένου χρόνου υλοποίησης και της ανεπαρκής γνώσης της απαιτούμενης τεχνολογίας, δεν ήταν δυνατόν να δημιουργηθεί ένα παιχνίδι με εξειδικευμένες απαιτήσεις σε γραφικά, φυσική, τεχνητή νοημοσύνη κ.ά. Είναι πιο εφικτό και προτιμότερο ένα παιχνίδι με όσο το δυνατόν πιο στατικά γραφικά, χωρίς υπερβολικά εφέ, ελάχιστη φυσική(π.χ. όχι χρήση collision detection), ελάχιστες απαιτήσεις σε ήχο, και απλή αλλά λειτουργική τεχνητή νοημοσύνη. Το είδος των turn-based παιχνιδιών στρατηγικής, καλύπτει τα παραπάνω.

Μετά την επιλογή του είδους του παιχνιδιού, επόμενο βήμα είναι η επιλογή του βασικού θέματος του παιχνιδιού. Πάλι, λόγω της επιθυμητής «στατικότητας» του παιχνιδιού, το θέμα προέκυψε αναλόγως. Στα turn-based παιχνίδια στρατηγικής αναπαριστώνται μονάδες που τοποθετούνται σε ένα χάρτη. Στα πιο σύγχρονα παιχνίδια, οι μονάδες κινούνται όταν αλλάζουνε πάνω σε αυτό το χάρτη, η επιτίθενται σε αντίπαλες μονάδες. Αυτό στη φάση της ανάλυσης φάνταζε αρκετά εξεζητημένο, οπότε και οι μονάδες θα έπρεπε να είναι στατικές. Γι' αυτό οι μονάδες του παιχνιδιού είναι παιδικά παιχνίδια που βρίσκονται σε ένα παιδικό δωμάτιο. Το παιχνίδι «βαπτίστηκε» ανάλογα, με το όνομα “toyWars”.

### 4.2.2. Ανάλυση λοιπών απαιτήσεων

Η προδιαγραφή 3, αναφέρει ότι το παιχνίδι θα πρέπει να τρέχει σε (τουλάχιστον) δύο λειτουργικά συστήματα, Linux και Windows. Αυτό επηρεάζει τη γλώσσα που θα προτιμηθεί για

την υλοποίηση, η οποία θα είναι η Java. Η προδιαγραφή 4 αναφέρεται στις ελάχιστες απαιτήσεις του προγράμματος. Καθώς ένα από τα δύο μηχανήματα στα οποία έγινε η ανάπτυξη ήταν ένας φορητός υπολογιστής μοντέλο του 2006, θεωρήθηκε ότι εφόσον τρέχει ικανοποιητικά σε αυτό το μηχάνημα, το παιχνίδι πληροί αυτή τη προδιαγραφή. Η προδιαγραφή 5 αναφέρεται στον τρόπο υλοποίησης του παιχνιδιού. Θα πρέπει να γίνει με τέτοιο τρόπο ώστε να επιτρέπονται οι αναβαθμίσεις και οι παραδόσεις νέας έκδοσης με σχετικά εύκολο τρόπο.

## 4.3. Σχεδίαση του παιχνιδιού

### 4.3.1. Τρόπος παιχνιδιού

Ο τρόπος παιχνιδιού είναι όμοιος με τον τρόπο παιχνιδιού των περισσότερων turn-based παιχνιδιών στρατηγικής. Αυτός αναλύεται παρακάτω.

#### Οι οντότητες του παιχνιδιού

Το παιχνίδι παίζεται από δύο παίχτες. Κάθε παίκτης, έχει μια ομάδα παιχνιδιών, τα οποία τοποθετούνται πάνω σε έναν εξάγωνο χάρτη, αντικριστά μεταξύ τους ανά ομάδα. Οι ομάδες αποτελούνται από διάφορα παιχνίδια, τα οποία σε όρους του παιχνιδιού στρατηγικής θα ονομάζουμε μονάδες.

Η κάθε μονάδα μια ομάδας, έχει κάποια χαρακτηριστικά ανάλογα με τον τύπο στον οποίο ανήκει. Υπάρχουνε τρεις τύποι μονάδων, αυτές που επιτίθενται από κοντά(melee), αυτές που επιτίθενται από μακριά(ranged) και οι ιπτάμενες(flying). Τα χαρακτηριστικά αυτών των τύπων παρουσιάζονται στον παρακάτω πίνακα:

| type/attributes | Health | Movement | Attack Damage | Attack Range |
|---|---|---|---|---|
| **Melee** | 10 | 2 | 4 | 1 |
| **Ranged** | 8 | 1 | 3 | 3 |
| **Flying** | 6 | 3 | 3 | 2 |

Δηλαδή μια melee μονάδα έχει 10 πόντους ζωής, μπορεί να κινηθεί μέχρι 2 πλακίδια, προκαλεί απώλεια 4 πόντων ζωής όταν επιτίθεται και το βεληνεκές της επίθεσής της είναι 1 πλακίδιο.

#### Πώς παίζεται

Καθώς δεν πρόκειται για παιχνίδι πραγματικού χρόνου αλλά για παιχνίδι που βασίζεσαι στην σειρά των παικτών, κάθε ομάδα παίζει ανά σειρά. Θα πρέπει να παίξουνε όλες οι μονάδες ενός παίκτη, για να για να μπορέσει να παίξει ο άλλος παίκτης. Η σειρά με την οποία παίζουν οι μονάδες δεν έχει σημασία, θα πρέπει όμως να παίξουνε όλες, ακόμα και δεν κουνηθούν ή δεν επιτεθούν καν.

Μια μονάδα, μπορεί να κάνει τα εξής κατά τη σειρά της. Αφού επιλεγεί, μπορεί να μετακινηθεί σε ένα πλακίδιο από αυτά που βρίσκονται στην εμβέλεια κίνησής της, με την προϋπόθεση φυσικά ότι εκεί δεν υπάρχει άλλη μονάδα. Αφού μετακινηθεί, μπορεί να επιτεθεί στις αντίπαλες μονάδες κάποια από αυτές βρίσκεται μέσα στην εμβέλεια επίθεσής της. Όταν επιτεθεί, η προσπεράσει το δικαίωμα επίθεσης, χάνει τη σειρά της, και η σειρά περνάει σε όποια άλλη μονάδα του παίχτη δεν έχει παίξει. Υπάρχει δυνατότητα μια μονάδα ακυρώσει την μετακίνηση που έκανε, αν ο παίκτης μετανιώσει για αυτή τη κίνησή του. Απλά την μετακινεί στο αρχικό πλακίδιο, και επιλέγει νέο.

Σκοπός του παιχνιδιού είναι να καταστραφούνε όλες οι μονάδες του αντιπάλου.

### 4.3.2. Ο χάρτης του παιχνιδιού

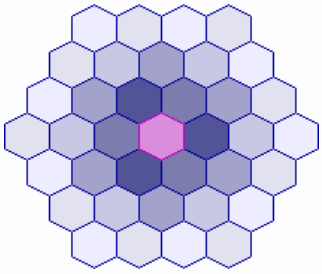
Πολλά παιχνίδια του είδους έχουνε τετράγωνα πλακίδια στο χάρτη για την κίνηση των μονάδων. Άλλα παιχνίδια έχουνε εξάγωνα. Λόγω αρχικά εμφάνισης, αλλά και μεγαλύτερου ενδιαφέροντος στη συνέχεια, προτιμήθηκε ο εξάγωνος χάρτης. Η σχεδίαση του εξάγωνου χάρτη αναφέρεται εδώ ξεχωριστά, καθώς πρόκειται για ένα από τα πιο ενδιαφέρον τμήματα της διπλωματικής διατριβής.

Το θέμα του εξάγωνου χάρτη, προκαλεί αρκετό ενδιαφέρον, όπως προκύπτει και από τη βιβλιογραφία [Patel2006]. Πρώτα θα παρουστούν τα πιθανά είδη τέτοιων χαρτών που μπορούν να υπάρξουν [Haugeland2004].

## Προσανατολισμός

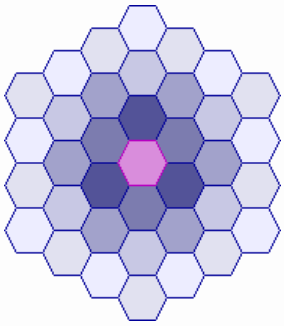
Ο προσανατολισμός ενός εξάγωνου χάρτη, εξαρτάται από τον προσανατολισμό των εξάγωνων που τον αποτελούνε.

### Οριζόντια και κατακόρυφα στοιχισμένοι χάρτες



**Εικόνα 4.1 - Οριζόντια στοιχισμένος χάρτης**

Οριζόντια στοιχημένο χάρτη, εννοούμε αυτόν που περιέχει οριζόντια στοιχισμένα εξάγωνα. Αριστερά και δεξιά από το κέντρο του πλέγματος έχουμε γωνίες.

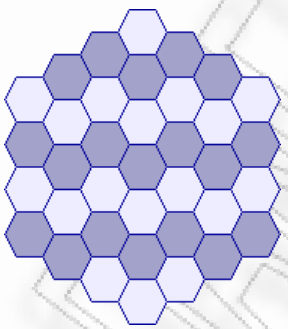


**Εικόνα 4.2 - Κατακόρυφα στοιχισμένος χάρτης**

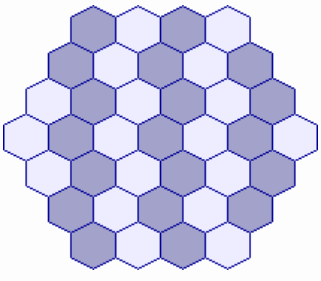
Κατακόρυφα στοιχισμένος χάρτης είναι αυτός που περιέχει κατακόρυφα στοιχισμένα εξάγωνα. Αριστερά και δεξιά από το κέντρο του πλέγματος υπάρχουνε δύο κατακόρυφες πλευρές. Αυτός είναι και ο προσανατολισμός που θα χρησιμοποιηθεί και για το παιχνίδι.

## Συστήματα συντεταγμένων

### Κυμαινόμενο σύστημα



**Εικόνα 4.3 - Οριζόντια κυμαινόμενο σύστημα**



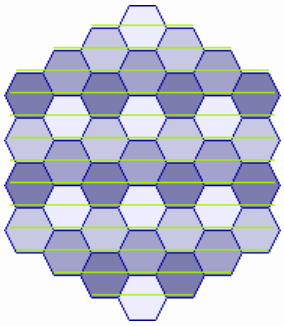
**Εικόνα 4.4 – Κατακόρυφα κυμαινόμενο σύστημα**

Κυμαινόμενο θα ονομάσω αυτό το σύστημα στο οποίο η σειρά των πλακιδίων πάνω στους άξονας πηγαίνει όπως ακριβώς είναι τοποθετημένα στο πλέγμα, τοποθετημένα πότε μισή θέση κάτω, πότε μισή θέση πάνω. Είναι το πιο εύκολο να παρατηρηθεί σύστημα πάνω σε ένα τέτοιο πλέγμα.

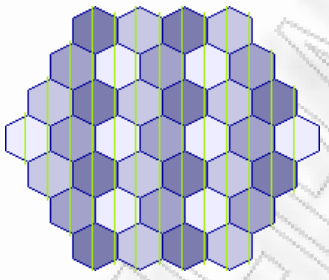
Ενώ όμως θεωρητικά είναι απλό, στην πράξη έχει αρκετές δυσκολίες. Φυσικά δεν ισχύουν οι κανόνες που ισχύουν σε ένα ορθοκανονικό σύστημα αξόνων. Εδώ έχουμε να κάνουμε με εξάγωνα, που δεν είναι στοιχισμένα το ένα ακριβώς δίπλα στο άλλο. Θα πρέπει να εμπλακεί ένα είδος λογικής, δηλαδή κάποια συνάρτηση ή ταίριασμα (mapping) στις συντεταγμένες

αυτές, ώστε να μπορέσουμε να τις χρησιμοποιήσουμε. Ενώ για απλά παιχνίδια αυτό δεν είναι σοβαρό πρόβλημα, για παιχνίδια στο οποία γίνεται χρήση αλγορίθμων όπως ο A* ή ο αλγόριθμος του Dijkstra ο βαθμός πολυπλοκότητα ανεβαίνει αισθητά.

**Διαπλεκόμενο**



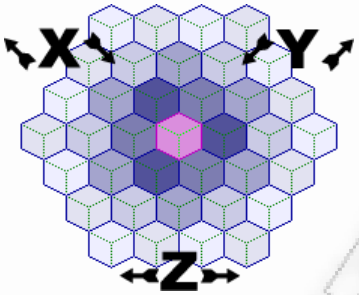
**Εικόνα 4.5 - Οριζόντια διαπλεκόμενο σύστημα**



**Εικόνα 4.6 - Κατακόρυφα διαπλεκόμενο σύστημα**

Διαπλεκόμενο λέμε το σύστημα στο οποία τα εξάγωνα είναι μεν ευθυγραμμισμένα, αλλά 1 προς 1. Με αυτόν τον τρόπο διπλασιάζονται οι τιμές πάνω στο σύστημα, προκαλώντας περισσότερο φόρτο στην όποια διεργασία μας. Μάλλον άβολο και σίγουρα δεν θα προτιμηθεί για το toyWars.

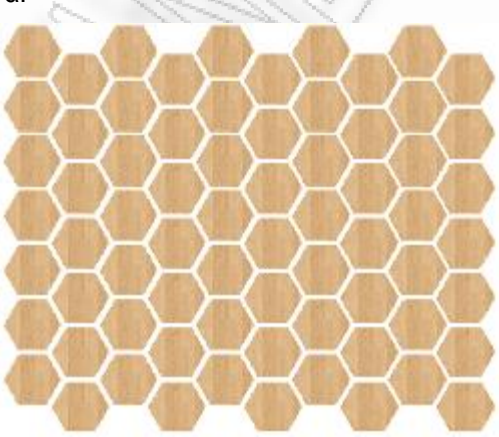
**Επίπεδο 3D**




**Εικόνα 4.7 - Επίπεδο τρισδιάστατο σύστημα**

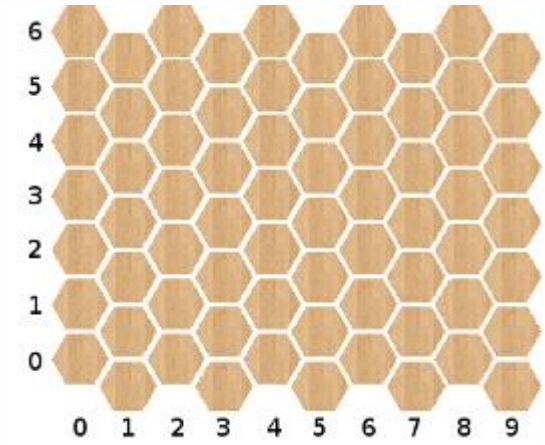
Κάθε εξάγωνο, μπορεί να αναπαρασταθεί ως ένα κύβος προσθέτοντας 3 εικονικές γραμμές. Έτσι, το σύστημα αυτό αποτελείται από τρεις άξονες. Και αυτό δεν θα προτιμηθεί, γιατί παρά το γεγονός ότι το σύστημα που προκύπτει είναι κανονικό (και οι τρεις άξονες είναι μετρημένοι στην ίδια μονάδα μέτρησης), η χρήση τριών διαστάσεων περιπλέκει την επεξεργασία των πλακιδίων αρκετά.

Με βάση τις παραπάνω επιλογές, αυτή που προτιμήθηκε είναι το οριζόντια κυμαινόμενο σύστημα συντεταγμένων. Ο χάρτη του παιχνιδιού θα πρέπει να δείχνει όπως στην παρακάτω εικόνα.



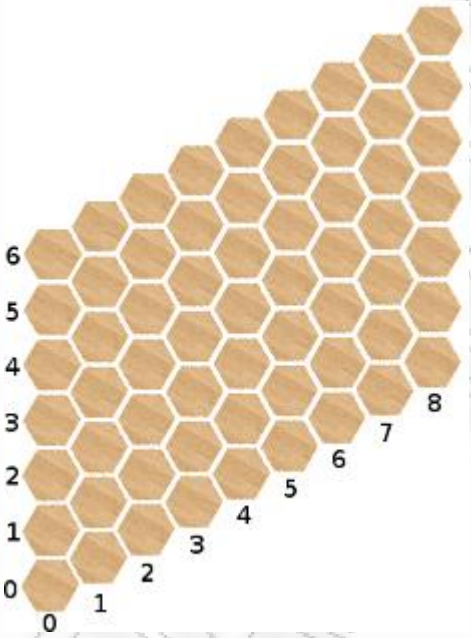
**Εικόνα 4.8 - Χάρτης του παιχνιδιού**

Με βάση την παραπάνω εικόνα, το ιδανικό θα ήταν το σύστημα των συντεταγμένων να οριζόταν κάπως έτσι:




**Εικόνα 4.9 - Λάθος σύστημα συντεταγμένων χάρτη**

Αυτό όμως δεν είναι εφικτό, καθώς τα πλακίδια μας δεν είναι τετράγωνα, αλλά εξάγωνα, και αυτά δεν είναι τοποθετημένα ευθυγραμμισμένα το ένα δίπλα στο άλλο, αλλά υπάρχει πάντα μια απόκλιση μισού εξαγώνου. Σε αυτό το σύστημα, ενώ ίσως είναι εφικτό να προσδιορίσεις ένα σημείο (πλακίδιο), αν εννοηθεί αυθαίρετα πχ ότι ως y έχουμε όλες τις κατακόρυφες γραμμές από εξάγωνα(οι οποίες έχουνε και στοιχισμένα κατακόρυφα εξάγωνα) και x το τον αριθμό των πλακιδίων που αυτές οι γραμμές περιέχουν. Θα ήταν όμως αδύνατον να μπορέσεις να προσδιορίσεις την απόσταση μεταξύ δύο πλακιδίων. Φυσικά ο τύπος Manhattan distance δεν ισχύει εδώ.




**Εικόνα 4.10 - Προτεινόμενο σύστημα συντεταγμένων**

Αυτό είναι το σύστημα συντεταγμένων που προτιμήθηκε. Το μεγαλύτερο μέρος της βιβλιογραφίας προτείνει ένα τέτοια σύστημα για τη δημιουργία εξάγωνου χάρτη. Μοιάζει με το ορθοκανονικό σύστημα αξόνων, χωρίς όμως το πρώτο συνθετικό, δεν είναι δηλαδή ορθογώνιο, οι άξονες x και y δεν είναι κάθετοι ο ένας στον άλλο.

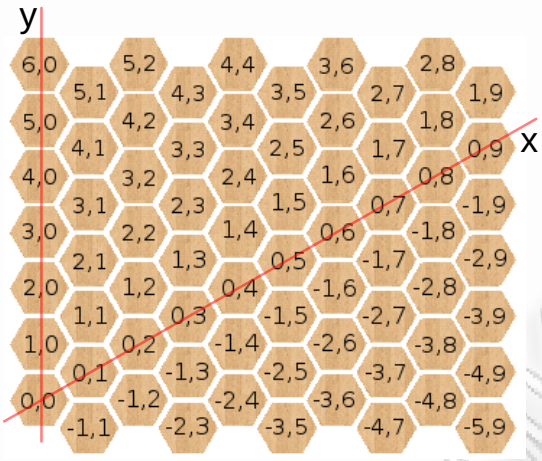
Φυσικά ο χάρτης δεν μπορεί να χρησιμοποιηθεί σε αυτή τη μορφή, καθώς εμείς θέλουμε ένα παραλληλόγραμμο εξάγωνο χάρτη(εικόνα 4.8). Θα πρέπει δηλαδή με κάποιο τρόπο ο χάρτης της εικόνας 4.8 να φτιαχτεί έχοντας τις συντεταγμένες του παραπάνω χάρτη (εικόνα 4.10)

Στο τελευταίο σχήμα, μπορούμε να παρατηρήσουμε το εξής. Ας ονομάσουμε τον κατακόρυφο άξονα y, ενώ τον πλάγιο x. Καθώς μετακινούμαστε οριζόντια προς τα δεξιά (πηγαίνοντας μία

πάνω και μία κάτω), η συμπεριφορά του x άξονα είναι η ίδια με τον χάρτη του πίνακα (τάδε), δηλαδή αυξάνεται σταθερά κατά τη μετακίνησή μας, και με τον ίδιο τρόπο και στους 2 χάρτες. Αντίθετα, αυτό δεν συμβαίνει με τον άξονα y. Μετακινούμενοι σταθερά προς τα δεξιά το y δεν είναι ίδιο και στους δύο χάρτες. Για την ακρίβεια, το y εδώ μειώνεται.

Άρα λοιπόν, θα πρέπει να κάνουμε μια ειδική αντιστοίχηση (mapping) στις συντεταγμένες, ώστε να μπορέσουμε να πάρουμε το επιθυμητό ορθογώνιο σχήμα. Η αντιστοίχηση αυτή, όπως θα φανεί στην υλοποίηση αργότερα, μπορεί να γίνει με τη βοήθεια μιας μεταβλητής, η οποία θα αυξάνεται κατά ένα σε περίπτωση μονής στήλης(x), και θα αφαιρείται από καθώς y συντεταγμένη, καθώς μετακινούμαστε προς τα δεξιά.

Έτσι, το σύστημα συντεταγμένων που προκύπτει θα δείχνει όπως στην παρακάτω εικόνα:




**Εικόνα 4.11 - Τελικό σύστημα συντεταγμένων**

Ο υπολογισμός της απόστασης, σε πλακίδια, μεταξύ δύο πλακιδίων υπολογίζεται από τον παρακάτω αλγόριθμο [Verbrugge1997]:

Για δύο σημεία A και B υπολογίζουμε τις διαφορές των x και y:
dx = Bx - Ax
dy = By - Ay
Εάν τα dx και dy είναι ταυτόσημα,
  Η απόσταση είναι η διαφορά των απόλυτων τιμών τους: distance = max( [dx], [dy] )
αλλιώς
  Η απόσταση είναι το άθροισμα των απόλυτων τιμών τους: distance = [dx] + [dy]

# 5. Υλοποίηση

Παρακάτω θα αναλυθεί η υλοποίηση του παιχνιδιού. Μέρος της τεκμηρίωσης έχει αναρτηθεί και στο blog http://toywars.wordpress.com/ που φτιάχτηκε για αυτό το σκοπό.

## 5.1. Τεχνολογίες που χρησιμοποιήθηκαν

Η γλώσσα που επιλέχθηκε για τη δημιουργία του παιχνιδιού είναι η Java. Με τη Java, το παιχνίδι μπορεί να τρέχει σε παραπάνω από ένα λειτουργικά συστήματα, χωρίς κάποια αλλαγή στον κώδικα. Επιπλέον θεωρήθηκε ως μία πολύ καλή ευκαιρία για να αποκτηθούν παραπάνω γνώσεις από τις πολύ βασικές που υπήρχαν πριν την ενασχόληση με τη μεταπτυχιακή διατριβή. Ακόμα, ο συνδυασμός Java και OpenGL (C++) για την δημιουργία ενός παιχνιδιού φαινόταν αρκετά ενδιαφέρον, και δεν συνηθίζεται.

Για τη δημιουργία των τρισδιάστατων γραφικών, χρησιμοποιήθηκε το OpenGL API. Η OpenGL είναι μια δωρεάν και ανοιχτού κώδικα βιβλιοθήκη, που μπορεί οποιοσδήποτε να κατεβάσει και να δοκιμάσει. Προηγήθηκε έρευνα σε άλλες εναλλακτικές, δωρεάν, επιλογές, όπως κάποιες μηχανές γραφικών με παραδείγματα τις Ogre (ogre4j) και jMonkeyEngine (jME). Το μεγαλύτερο ενδιαφέρον για μία πιο low-level υλοποίηση και η σχετικά μικρή υποστήριξη για τα παραπάνω έκαναν την OpenGL πιο ελκυστική.

Για να μπορέσει να γίνει το πάντρεμα αυτών των δύο, χρειάζεται μια ενδιάμεση βιβλιοθήκη. Εδώ οι λύσεις είναι δύο, η JOGL και η LWJGL. Αφού δοκιμάστηκαν και οι δύο, προτιμήθηκε η δεύτερη. Η LWJGL είναι μια δωρεάν, ανοιχτού κώδικα βιβλιοθήκη που παρέχει στους προγραμματιστές το interface ώστε να χρησιμοποιούν OpenGL και OpenAL εντολές, καθώς και πρόσβαση σε ελεγκτές για συσκευές όπως ποντίκι, πληκτρολόγιο και χειριστήριο.

## 5.2. Τεκμηρίωση του κώδικα

Εδώ θα παρουσιαστεί και θα εξηγηθεί το μεγαλύτερο μέρος του κώδικα του παιχνιδιού. Δεν θα γίνει εκτενής ανάλυση, αλλά θα αναφερθούνε τα βασικότερα σημεία κάθε κλάσης, τα πιο ενδιαφέροντα από πλευράς υλοποίησης και αυτά που είναι σχετικά δυσκολότερα να κατανοηθούν από τα υπόλοιπα. Οι κλάσεις θα αναφερθούνε ανά πακέτο στο οποίο ανήκουν, με σειρά που εξυπηρετεί την καλύτερη επεξήγηση του κώδικα.

### gr.jmanji.toywars.state

Το πακέτο αυτό είναι από τα βασικότερα του παιχνιδιού καθώς αναλαμβάνει να δημιουργήσει την «οθόνη» του παιχνιδιού, να δημιουργήσει το OpenGL πλαίσιο και στη συνέχεια ξεκινάει και τρέχει το παιχνίδι μέχρις ότου ζητηθεί να σταματήσει.

#### State

Το State είναι ένα interface που ορίζει τις μεθόδους που θα πρέπει να υλοποιήσουνε τα states (καταστάσεις) του παιχνιδιού. Θεωρούμε ότι ο τρόπος που ζωγραφίζεται το παιχνίδι, χωρίζεται σε 2 βασικά μέρη. Το ένα είναι η δισδιάστατη απεικόνιση του μενού και γενικά των δισδιάστατων εικόνων, και το άλλο είναι το κύριο μέρος του παιχνιδιού που ζωγραφίζεται σε τρισδιάστατη απεικόνιση. Αυτός ο διαχωρισμός ισχύει σε νοηματικό επίπεδο, καθώς οι λειτουργίες των καταστάσεων είναι τελείως διαφορετικές, αλλά και σε επίπεδο υλοποίησης. Για το λόγο αυτό κρίθηκε χρήσιμο να υπάρχουνε διαφορετικές κλάσεις που χρησιμοποιούνται σε κάθε κατάσταση. Το interface State είναι το κοινό σημείο αυτών των κλάσεων.

Οι σημαντικότερες μέθοδοι που ορίζει το interface είναι οι εξής:

```
public void init() throws IOException;
public void enter();
public void update();
public void render();
```

Η init() αρχικοποιεί τα resources που απαιτούνται για την κλάση. Καλείται μια φορά μόνο για κάθε κατάσταση, μετά την δημιουργία του αντικειμένου. Η enter() επαναπροσδιορίζει τις διάφορες μεταβλητές και (OpenGL) καταστάσεις κάθε φορά που εισερχόμαστε σε μια κατάσταση. Οι update() και render() είναι ονόματα μεθόδων που χρησιμοποιούνται κατά κόρον στο παιχνίδι, με την ίδια σημασία.

*update και render*

Η render() αφορά αποκλειστικά και μόνο το rendering όλων τον αντικειμένων του παιχνιδιού. Οτιδήποτε εκτελείται σε μια τέτοια μέθοδο, θα έχει πάντα σκοπό να ζωγραφίσει στην οθόνη.

Η update() από την άλλη, αναλαμβάνει τη «λογική» πίσω από το παιχνίδι. Είναι υπεύθυνη δηλαδή για την αλληλεπίδραση του παιχνιδιού με το χρήστη, τον υπολογισμό των νέων θέσεων των αντικειμένων στο χώρο, την τεχνητή νοημοσύνη κ.ά. Κάθε κατάσταση αλλά και κάθε αντικείμενο του παιχνιδιού συμμορφώνεται με αυτή τη λογική. Έτσι, αρχικά εκτελούνται οι υπολογισμοί που αφορούν τη λογική του παιχνιδιού και ύστερα η OpenGL ζωγραφίζει το αποτέλεσμα αυτών των υπολογισμών.

**GUI2D**

Είναι μια αφηρημένη (abstract) κλάση που υλοποιεί το interface State. Παρέχει τις βασικές μεταβλητές και μεθόδους που θα χρησιμοποιήσουν όσες κλάσεις θέλουν να ζωγραφίσουν στην οθόνη 2D γραφικά.

Η render() εδώ ζωγραφίζει ένα κινούμενο λογότυπο του παιχνιδιού, ενώ στο κάτω μέρος της οθόνης ζωγραφίζει την έκδοση του παιχνιδιού και τον δημιουργό του. Οι συμβολοσειρές αυτές στο κάτω μέρος της οθόνης καθώς και οποιοδήποτε κείμενο του παιχνιδιού, ζωγραφίζονται με την κλάση gr.jmanji.toywars.font.TrueTypeFont που αναφέρεται παρακάτω. Ο τρόπος που πετυχαίνεται το κινούμενο λογότυπο, θα εξηγηθεί στην κλάση gr.jmanji.toywars.object.Room. Η ιδιαιτερότητα της κλάσης αυτής, αλλά και αυτών που την επεκτείνουνε, είναι ότι ζωγραφίζουνε 2D γραφικά. Αυτό αποτελεί στην ουσία μια «εξαίρεση» για το παιχνίδι, καθώς το κύριο μέρος του είναι 3D. Για τη μετάβαση από 3D σε 2D και αντίστροφα, έχουνε δημιουργηθεί δύο static μέθοδοι στην κλάση GameWindow που θα αναλυθεί παρακάτω.

**Menu**

Ζωγραφίζει ένα μενού με κάποιες επιλογές που μπορεί να κάνει ο χρήστης. Πατώντας την επιλογή “Play”, θα βγει από αυτήν την κατάσταση, και θα εισέλθει στην κατάσταση Game. Πατώντας την επιλογή “Help”, θα εισέλθει στην κατάσταση Help, ενώ πατώντας την επιλογή “Exit”, το παιχνίδι θα κλείσει, καταστρέφοντας όλα τα resources και ελευθερώνοντας την δεσμευμένη μνήμη.

Οι μέθοδοι:

```
public void update()
public void render()
```

ανανεώνουν και ζωγραφίζουν το παιχνίδι. Η update() θα πρέπει να «πιάσει» το input από το χρήστη, είτε αυτό είναι το ποντίκι είτε το πληκτρολόγιο. Αυτό ο έλεγχος των περιφερικών από το παιχνίδι γίνεται με τη βοήθεια βιβλιοθηκών που παρέχει η LWJGL και συγκεκριμένα με τα πακέτα org.lwjgl.input.Mouse και org.lwjgl.input.Keyboard. Στην περίπτωση του πληκτρολογίου, ελέγχεται το πόσες φορές έχουν πατηθεί τα πλήκτρα πάνω και κάτω, και αντίστοιχα «φωτίζονται» οι επιλογές. Αν το input είναι το ποντίκι, τότε η επιλογή του χρήστη καθορίζεται μέσω της διαδικασίας picking, η οποία θα αναλυθεί στη κλάση gr.jmanji.toywars.mouse.Picker.

Η εικόνα του μενού εμφανίζεται και κατά την εισαγωγή του παιχνιδιού, αλλά και κατά της διάρκειά του αν πατηθεί το πλήκτρο escape. Η διαφορά είναι ότι στην δεύτερη περίπτωση υπάρχει ως φόντο μια θολή εικόνα με την κατάσταση του παιχνιδιού την στιγμή που πατήθηκε το πλήκτρο, σαν μια θολή φωτογραφία. Η υλοποίηση αυτού του εφέ, έγινε με τον παρακάτω τρόπο. Μέσω της gr.jmanji.toywars.texture.Screenshot μπορούμε να έχουμε ένα texture με την τελευταία εικόνα του παιχνιδιού, όπως αυτό είχε ζωγραφιστεί ακριβώς πριν πατηθεί το πλήκτρο escape από το χρήστη. Για να θολώσουμε αυτή την εικόνα, την ζωγραφίζουμε πολλές φορές τη μία πάνω στην άλλη, με λίγο διαφορετική θέση κάθε φορά. Έτσι, εφόσον είναι ενεργοποιημένο το GL_BLEND, η πάνω εικόνα δεν θα επικαλύπτει πλήρως τις από κάτω, αλλά θα φαίνονται όλες μαζί σχηματίζοντας ένα θολό φόντο.

**Help**

Η κλάση Help επεκτείνει την GUI2D ζωγραφίζοντας τα ονόματα των πλήκτρων που μπορεί να πατήσει ο χρήστης, μαζί με μια περιγραφή των λειτουργιών τους. Ο τρόπος σχεδίασής της είναι παρόμοιος με τη κλάση Menu.

**GameWindow**

Η κύρια κλάση του παιχνιδιού, που περιέχει και τη συνάρτηση main(), είναι η GameWindow. Αρχικά αναζητά ένα κατάλληλο αντικείμενο Display με τα χαρακτηριστικά που επιστρέφει η gr.jmanji.toywars.config.ConfigReader για την οθόνη στην οποία εκτελείται το πρόγραμμα. Το αντικείμενο αυτό αποτελεί την οθόνη του παιχνιδιού, μέσα στο οποίο θα τρέχει. Τη στιγμή που θα εκτελεστεί η μέθοδος create του αντικειμένου Display, το πρόγραμμα είναι έτοιμο να δεχτεί

OpenGL εντολές. Στο νεοδημιουργηθέν OpenGL πλαίσιο (context) λαμβάνει χώρα η αρχικοποίηση κάποιων βασικών OpenGL καταστάσεων τις οποίες θα χρησιμοποιεί όλο το παιχνίδι (εκτός αν αλλάξουν κατά τη διάρκεια) όπως η ενεργοποίηση φωτισμού, textures, blending κ.ά.

Η GameWindow περιέχει τον βασικό βρόγχο του προγράμματος, μέσα στην gameLoop(). Η λογική που διέπει αυτόν τον βρόγχο είναι η εξής:

Ενώ τρέχει το παιχνίδι:

1. Ανανέωσε την κατάσταση (state.update())
2. Ζωγράφισε το την κατάσταση (state.render())
3. Ανανέωσε το Display αντικείμενο(display.update())

Η τρίτη εντολή σχετίζεται αποκλειστικά με την OpenGL. Είναι αναγκαία και τίποτα δεν θα εμφανιστεί στην οθόνη μας, αν δεν την καλέσουμε.

Με τον διαχωρισμό των καταστάσεων, είναι πλέον πιο εύκολη η μετάβαση από τη μία στην άλλη. Η GameWindow περιέχει ένα HashMap αντικείμενο με όλες τις καταστάσεις. Ανάλογα με την κατάσταση που θέλουμε να έχουμε κάθε φορά, εξάγουμε την αντίστοιχη κατάσταση από τη λίστα.

**Game**

Είναι η κλάση στην οποία ανανεώνεται(update) και ζωγραφίζεται το κυρίως παιχνίδι.

Στη μέθοδο:

```
public void update()
```

όπως είναι αναμενόμενο, συμβαίνουν πολλές σημαντικές λειτουργίες. Η κάμερα του παιχνιδιού δέχεται το input από το χρήστη, λαμβάνονται υπόψη τα πλήκτρα που πατάει ο χρήστης(π.χ. f για fullscreen), παίζουνε οι παίχτες και ορίζεται η σειρά τους και το mode του παιχνιδιού. Ανάλογα με το mode, ο χάρτης θα χρωματίσει ανάλογα τα πλακίδια. Επιπλέον εδώ συμβαίνει και ένας ειδικός έλεγχος. Όταν μια μονάδα εκτελεί μια κίνηση, δεν πρέπει παράλληλα να μπορεί να χρήστης να ελέγξει άλλη μονάδα, ούτε η AI να παίξει άλλη μονάδα. Μέχρι να τελειώσει η κίνηση μιας μονάδας, δεν λαμβάνονται υπόψη οι παραπάνω λειτουργίες.

Στην μέθοδο:

```
public void render()
```

ζωγραφίζεται όλο το παιχνίδι, δηλαδή το δωμάτιο, ο χάρτης και οι μονάδες καλώντας τις αντίστοιχες μεθόδους για κάθε ένα από αυτά. Στο κάτω μέρος της οθόνης, υπάρχει μια περιοχή στην οποία παρουσιάζονται διάφορα μηνύματα. Για να μπορέσει να αποκλειστεί αυτή η περιοχή του viewport και να μην ζωγραφίζεται, χρησιμοποιήθηκε το scissor test της OpenGL. Αρχικά, ορίζεται μια παραλληλόγραμμη περιοχή στο παράθυρο. Όταν ενεργοποιούμε το GL_SCISSOR_TEST αυτή η περιοχή δεν λαμβάνεται υπόψη στο rendering και άρα τότε ζωγραφίζουμε όλο το παιχνίδι, αφήνοντας αυτή τη περιοχή με το χρώμα που έχει οριστεί για το καθάρισμα της οθόνης. Όταν σε αυτή τη περιοχή αργότερα θέλουμε να ζωγραφίσουμε τα μηνύματα, ενεργοποιούμε το GL_SCISSOR_TEST.

### gr.jmanji.toywars.shape

Θεωρήθηκε σωστό κάποια απλά γεωμετρικά σχήματα που χρησιμοποιεί το παιχνίδι να ενσωματωθούνε μέσα σε αντίστοιχες κλάσεις. Έτσι μπορούν να χρησιμοποιηθούν πολλές φορές χωρίς να επαναλαμβάνεται ο ίδιος κώδικας.

**Rectangle**

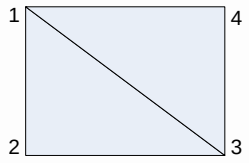
Η κλάση Rectangle σχεδιάζει ένα ορθογώνιο παραλληλόγραμμο. Πρόκειται για ένα βασικό σχήμα, που θα χρησιμοποιηθεί πολλές φορές στη σχεδίαση του δωματίου. Σύμφωνα με τον κατασκευαστή της κλάσης:

```
public Rectangle(float width, float height, float factor)
```

ορίζουμε το μήκος και το πλάτος του παραλληλογράμμου. Επιπλέον ορίζουμε και μια μεταβλητή factor, η οποία υποδεικνύει το πόσες φορές θα γίνει mapped ένα texture πάνω στο παραλληλόγραμμο αυτό. Πόσες φορές δηλαδή θα ταιριάξει το texture πάνω στο παραλληλόγραμμο. Σε περίπτωση που είναι μικρότερο της μονάδας, θα χρησιμοποιηθεί μέρος του texture, ενώ αν είναι μεγαλύτερο, το texture θα επαναλαμβάνεται.

```
public void render()
```

Η render() θα ζωγραφίσει το παραλληλόγραμμο σύμφωνα με τις παραπάνω μεταβλητές, ως μια ένωση δύο τριγώνων (η OpenGL σε πολλές κάρτες γραφικών αποδίδει καλύτερα όταν ζωγραφίζει τρίγωνα απ' ότι τετράγωνα).




**Εικόνα 5.1 – Σχεδίαση παραλληλογράμμου με GL_TRIANGLE_STRIP**

Έχει σημασία η σειρά με την οποία σχεδιάζουμε τις κορυφές. Η φορά αυτή καθορίζει ποιό είναι το «πρόσωπο» της σχήματός μας. Στην OpenGL, το default πρόσωπο ενός σχήματος είναι η πλευρά στην οποία οι κορυφές της έχουνε σχεδιαστεί αντίστροφα από τη φορά του ρολογιού.

**Hexagon**

Η κλάση Hexagon σχεδιάζει ένα κανονικό εξάγωνο. Το σχήμα αυτό θα χρειαστεί για την κατασκευή του πλακιδίου.

Ο κατασκευαστής της κλάσης:

```
public Hexagon(float size)
```

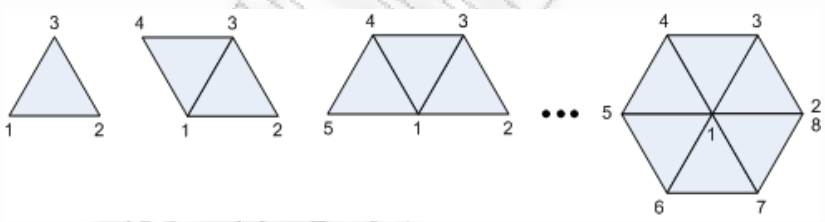
απαιτεί να ορισθεί ένα μέγεθος (size) για το σχήμα αυτό, που είναι η διάμετρος του νοητού κύκλου που περικλείει από αυτό το εξάγωνο.

Αφού έχουμε ορίσει αρχικά ως βήμα (STEP) το 1/6 του 2π, η σχεδίαση του εξαγώνου στη μέθοδο:

```
public void render()
```

γίνεται ως εξής:

1. Θέσε τη γωνία (angle) ίση με 0 και σημείωσε το σημείο (0, 0).
2. Για τις 6 κορυφές του εξαγώνου κάνε:
    a. Υπολόγισε το x = συν(γωνία).
    b. Υπολόγισε το y = ημ(γωνία).
    c. Σημείωσε το σημείο (x, y).
    d. Αύξησε την γωνία κατά το βήμα.




**Εικόνα 5.2 – Σχεδίαση εξαγώνου με GL_TRIANGLE_FAN**

Η ακριβής υλοποίηση του παραπάνω αλγορίθμου είναι όπως βρίσκεται στην κλάση:

```
public void render() {
    float angle = 0.0f; // angle(in radians) for each vertex
    float x, y;

    GL11.glBegin(GL11.GL_TRIANGLE_FAN);
        GL11.glNormal3f(0.0f, 0.0f, 1.0f);
        GL11.glTexCoord3f(0.5f, 0.5f, 0.0f);
        GL11.glVertex3f(0.0f, 0.0f, 0.0f);

        // Divide the circle up into 6 sections(starting from top right)
        for (int numVertices = 0; numVertices < 6; numVertices++){
            x = (float)Math.cos(angle);
```

```
            y = (float)Math.sin(angle);
            angle += STEP;

            // Map the hexagon coordinates to image coordinates(0...1)
            GL11.glTexCoord3f((x+1)/2.0f, (y+1)/2.0f, 0.0f);
            GL11.glVertex3f(radius*x, radius*y, 0.0f);
        }
        // Last vertex closes the fan (defined explicitely for best
precision)
        GL11.glTexCoord3f(1.0f, 0.5f, 0.0f);
        GL11.glVertex3f(radius, 0.0f, 0.0f);
    GL11.glEnd();
}
```

Μια ιδιαιτερότητα σε αυτή τη μέθοδο είναι η γραμμή:

```
GL11.glTexCoord3f((x+1)/2.0f, (y+1)/2.0f, 0.0f);
```

Στην OpenGL, μπορούμε να ζωγραφίζουμε σε οποιοσδήποτε συντεταγμένες, αλλά πάνω σε ένα texture, θα πρέπει να περιοριστούμε στο διάστημα (0, 1). Με τον παραπάνω τρόπο πετυχαίνεται αυτή η αντιστοίχιση ώστε από ένα τετράγωνο texture, να παίρνουμε πάντα το εξάγωνο που αντιστοιχεί στα x και y.

**DoubleCone**

Η κλάση αυτή ζωγραφίζει μια πράσινη πυραμίδα πάνω από την εκάστοτε τρέχουσα μονάδα.

### gr.jmanji.toywars.mouse

Το πακέτο αυτό περιέχει κλάσεις που αφορούν στην αλληλεπίδραση με το ποντίκι.

**MousePointer**

Πρόκειται για μια κλάση που ζωγραφίζει ένα δικό μας mouse pointer, διαφορετικό από αυτό του λειτουργικού συστήματος που χρησιμοποιούμε. Η μέθοδος:

```
public void render()
```

αλλάζει προσωρινά το σύστημά μας σε ορθοκανονικό, για να σχεδιάσει ένα τετράγωνο, 32x32 pixels, και να τοποθετήσει επάνω του ένα texture ίδιων διαστάσεων με σχέδιο ένα βέλος δικής μας προτίμησης. Αυτό θα είναι το pointer του παιχνιδιού. Ο τύπος του αρχείου εικόνας θα πρέπει φυσικά να υποστηρίζει transparency (π.χ. png) και εκτός της περιοχής όπου είναι ζωγραφισμένο το βέλος, να είναι διαφανές. Έτσι, εφόσον την στιγμή που ζωγραφίζουμε το βέλος έχουμε ενεργοποιημένο το mode GL_BLEND και έχουμε ήδη ζωγραφίσει όλο το υπόλοιπο παιχνίδι, θα δούμε το δικό μας pointer να μετακινείται όταν μετακινούμε το ποντίκι.

**Picker**

Πρόκειται για μια πολύ σημαντική κλάση, που είναι υπεύθυνη για την επιλογή των αντικειμένων που επιθυμούμε στο χώρο με το ποντίκι. Λόγω της πολυπλοκότητάς και σπουδαιότητά της στο παιχνίδι, θα εξετασθεί αναλυτικότερα.

Στο μεγαλύτερο μέρος της δημιουργίας του παιχνιδιού, αλλά και γενικά σε όλες τις εφαρμογές OpenGL και 3D γραφικών, ο γενικός τρόπος λειτουργία είναι κοινός. Θα πρέπει 3D συντεταγμένες που έχουμε σχεδιάσει σε κάποιο πρόγραμμα, να αναπαρασταθούνε (λόγω του περιορισμού των υλικών μέσων) σε μια 2D οθόνη. Αυτό στην OpenGL το αναλαμβάνει το viewport. Σε περίπτωση όμως που θέλουμε να έχουμε αλληλεπίδραση με το mouse σε ένα 3D περιβάλλον, ο τρόπος σκέψης είναι ακριβώς ο αντίθετος. Θα πρέπει δηλαδή οι 2D συντεταγμένες του σημείου που βρίσκεται ο pointer του ποντικιού στην οθόνη, να μεταφραστούνε στις 3D συντεταγμένες του σημείου στο χώρο που θέλουμε να επιλέξουμε. Αν ακούγεται απλό, δεν είναι.

Ευτυχώς η OpenGL διαθέτει έναν ενδιαφέρον και εύκολο μηχανισμό για αυτή τη διαδικασία, που ονομάζεται selection. Ο γενικός αλγόριθμος που ακολουθεί, είναι:

1. Βρες τις συντεταγμένες του ποντικιού στο παράθυρο.
2. Μπες σε selection mode.
3. Επαναπροσδιόρισε το viewing volume ως μια μικρή περιοχή της οθόνης γύρω από τον βελάκι του ποντικιού (picking).
4. Ζωγράφισε την σκηνή, είτε ολόκληρη είτε μόνο τα αντικείμενα που είναι σχετικά με το

selection.

5. Βγες από το selection mode και αναγνώρισε τα αντικείμενα που ζωγραφίζονται στο καινούριο προσωρινό viewing volume.

Για να μπορέσει να γίνει η αναγνώριση των αντικειμένων που θέλουμε να είναι διαθέσιμα προς επιλογή, πρέπει κάπως να τα ονομάσουμε. Γι' αυτό υπάρχει μια στοίβα ονομάτων (name stack) στην οποία εισάγουμε τα ονόματα που θέλουμε κάθε φορά να ονομάσουμε. Η OpenGL παρέχει επίσης ένα selection buffer στο οποία αποθηκεύονται τα hits κατά τη διάρκεια του selection mode. Μια εγγραφή hit ονομάζουμε την εγγραφή που εισάγεται στο selection buffer σε περίπτωση που ένα αντικείμενο που ζωγραφίζεται τέμνει το καινούριο viewing volume. Αυτή θα περιέχει τα ονόματα που βρίσκονται στο name stack καθώς και το ελάχιστο και μέγιστο βάθος του αντικειμένου.

Ένα παράδειγμα με δύο hit εγγραφές, θα μπορούσε να είναι το παρακάτω:

| Περιεχόμενα hit εγγραφής | Περιγραφή |
|---|---|
| 1 | Αριθμός ονομάτων για το πρώτο hit |
| (7.3) | Ελάχιστο βάθος για το πρώτο hit |
| (7.5) | Μέγιστο βάθος για το πρώτο hit |
| 7 | Όνομα πρώτου hit |
| 2 | Αριθμός ονομάτων για το δεύτερο hit |
| (4.2) | Ελάχιστο βάθος για το πρώτο hit |
| (5.9) | Μέγιστο βάθος για το πρώτο hit |
| 14 | Πρώτο όνομα για δεύτερο hit |
| 15 | Δεύτερο όνομα για δεύτερο hit |

**Εικόνα 5.3 – Πίνακας με τα περιεχόμενα του selection buffer**

Συνήθως όταν γίνεται ένα αριστερό κλικ μας ενδιαφέρει να βρούμε ποιό αντικείμενο επέλεξε ο χρήστης. Άρα ψάχνουμε αν καταρχάς καταγράφηκε κάποια hit εγγραφή στο selection buffer. Αν ναι και αυτή είναι μια, τότε έχουμε βρει το όνομα του αντικειμένου. Αν είναι παραπάνω από μία, τότε θέλουμε να μάθουμε ποιό από τα αντικείμενα ήτανε πιο κοντά στο viewport. Θα πρέπει να χρησιμοποιηθεί η πληροφορία που υπάρχει για τα βάθη των αντικειμένων. Στο παραπάνω παράδειγμα, που αναφέρεται σε τέτοια περίπτωση, το αντικείμενο του δεύτερου hit, με τα δύο ονόματα (14, 15), είναι αυτό που βρίσκεται πιο κοντά στο viewport, και άρα αυτό θεωρούμε ότι επέλεξε ο χρήστης.

Η υλοποίηση για το selection βρίσκεται στις εξής βασικές μεθόδους:

```
private void startPickingGeneric(int xMouse, int yMouse)
public void startPicking3D(int xMouse, int yMouse)
public void startPicking2D(int xMouse, int yMouse)
public void stopPicking()
```

Η startPickingGeneric() είναι γενική μέθοδος που χρησιμοποιείται από τις startPicking3D() και startPicking2D(). Αυτό που κάνει είναι:

1. Αποθήκευσε το τρέχον viewing volume.
2. Ενεργοποίησε το selection mode.
3. Αρχικοποίησε το name stack.
4. Αποθήκευσε το Projection Matrix.
5. Επαναπροσδιόρισε το viewing volume σε ένα 5χ5 pixels τετράγωνο γύρω από τη θέση του mouse (picking).

Έτσι, οι startPicking3D() ή startPicking2D (ανάλογα με το αν θέλουμε picking για 2 ή 3 διαστάσεις αντίστοιχα), μαζί με την startPickingGeneric(), καταστούν το παιχνίδι έτοιμο για picking. Οτιδήποτε θα ζωγραφίζεται από εδώ και πέρα μέχρι να καλεστεί η stopPicking(), δεν

θα φαίνεται στο παράθυρο, ενώ τα ονόματα και τα βάθη των αντικειμένων, θα αποθηκεύονται στο selection buffer. Η stopPicking() ξαναφέρνει το viewing volume στην κατάσταση που ήταν.

Έτσι, το interface που παρέχει η κλάση Picker προς τις άλλες κλάσεις, χρησιμοποιείται ως εξής:

```
startPicking3D();
renderSelection();
stopPicking3D();
```

Οι μέθοδοι:

```
public TileCoords getSelectedTile()
public int getSelectedUnit()
```

ψάχνουν μέσα στο selection buffer, για να επιστρέψουν αν και ποιό πλακίδιο ή μονάδα επιλέχτηκε κατά τη διάρκεια του selection. Η ανάγκη ύπαρξης δύο διαφορετικών μεθόδων έγκειται στο γεγονός ότι χρειάζεται διαφορετική διαχείριση του selection buffer κατά περίπτωση, καθώς το κάθε πλακίδιο του χάρτη του παιχνιδιού έχει δύο ονόματα, ενώ η μονάδα έχει ένα.

### gr.jmanji.toywars.object.tile

Το πακέτο αυτό διαχειρίζεται ό,τι αφορά το πλακίδιο του χάρτη του παιχνιδιού.

#### Tile

Η κλάση Tile αποτελεί όπως δηλώνει το πλακίδιο του χάρτη. Πάνω σε αυτό θα βρίσκονται οι μονάδες του παιχνιδιού. Ο κατασκευαστής:

```
public Tile(Texture baseTexture, Texture sideTexture, float size)
```

απαιτεί δύο textures (ένα για τη βάση και ένα για τις πλάγιες πλευρές) και μια τιμή μεγέθους που ορίζει το μέγεθος του πλακιδίου. Στην ουσία το τρισδιάστατο πλακίδιο αποτελείται από ένα εξάγωνο και πέντε παραλληλόγραμμα κάθετα σε αυτό. Ο αλγόριθμος για το σχεδιασμό των κάθετων πλευρών είναι ίδιος με τον αλγόριθμο της σχεδίασης του εξαγώνου. Πρώτα ζωγραφίζουμε το εξάγωνο. Έπειτα κάνουμε μια περιστροφή 90 μοιρών στον άξονα x ώστε να ζωγραφίσουμε τα παραλληλόγραμμα κάθετα στο εξάγωνο. Αρχίζουμε από το πρώτο, κάνοντας κάθε φορά μια περιστροφή στον y άξονα κατά την γωνία.

#### TileCoords

Πρόκειται για μια βοηθητική κλάση που επιτρέπει την αποθήκευση της θέσης ενός πλακιδίου στο χάρτη, μέσα σε ένα αντικείμενο. Έτσι αποθηκεύεται σε ένα αντικείμενο η θέση (x, y) στο χάρτη.

### gr.jmanji.toywars.object

Το πακέτο αυτό περιέχει κλάσεις που αποτελούνε αντικείμενα του παιχνιδιού, όπως το δωμάτιο και ο χάρτης.

#### Map

Η κλάση αυτή ζωγραφίζει τον χάρτη του παιχνιδιού, που αποτελείται από τρισδιάστατα πλακίδια και τα χρωματίζει ανάλογα. Καθώς πρόκειται για ένα από τα πιο δύσκολα κομμάτια της διπλωματικής, θα αναλυθεί λεπτομερώς.

Όπως δηλώνεται και από τον κατασκευαστή της,

```
public Map(int xDim, int yDim)
```

το μέγεθος του χάρτη δεν είναι στατικό, αλλά μπορεί να πάρει τις διαστάσεις που επιθυμούμε. Ορίζουμε τον αριθμό των γραμμών και στηλών από πλακίδια που θέλουμε, και ένας χάρτης με γραμμές x στήλες πλακίδια θα κατασκευαστεί.

Οι μέθοδοι

```
public void render(ArrayList colorList, Mode mode)
public void renderSelection()
```

ζωγραφίζουν το χάρτη για εμφάνιση και επιλογή αντίστοιχα. Η render() δέχεται δύο παραμέτρους. Το colorList είναι η λίστα με τις συντεταγμένες των πλακιδίων που πρέπει να χρωματιστούν, και mode το mode στο οποίο βρίσκεται το παιχνίδι και από το οποίο θα εξαρτηθεί το χρώμα των πλακιδίων στη λίστα.

Οι δύο μέθοδοι μοιάζουνε αρκετά. Η renderSelection() φυσικά δεν θα εμφανίσει τίποτα στην

οθόνη, καθώς την καλούμε κατά το picking, και γι' αυτό πρόκειται για μια απλοποιημένη έκδοση της render(). Ας την εξετάσουμε αναλυτικά.

```
public void renderSelection(){
    int coordOffset = 0;

    GL11.glPushMatrix();
    GL11.glTranslatef(-getWidth()/2, -getHeight()/2, 0.0f);
    GL11.glTranslatef(0.0f, 0.0f, tile.getHeight());

    // render first by column (for each row)
    for (int i=0; i<xDim; i++){
        // push the x coordinate of the unit for picking
        GL11.glPushName(i);
            if (i%2!=0)
                coordOffset++;

         GL11.glPushMatrix();
            // render a column
            for (int j=-coordOffset; j<yDim-coordOffset; j++){
                // push the y coordinate of the unit for picking
                GL11.glPushName(j);
                    //  render  only  hexagons  in  selection  mode,  for
efficiency
                    // GL11.glCallList(tileID);
                    tile.renderBase();
                GL11.glPopName();

                // move to the next row
                GL11.glTranslatef(0.0f, vDistance, 0.0f);
            }
        GL11.glPopName();

        GL11.glPopMatrix();

        // move to the next column
        GL11.glTranslatef(hDistance, 0.0f, 0.0f);

        // add the offset
        if (i%2!=0)
            GL11.glTranslatef(0.0f, tileWidth/2.0f + OFFSET/2.0f, 0.0f);
        else
            GL11.glTranslatef(0.0f, -tileWidth/2.0f - OFFSET/2.0f, 0.0f);
    }

    GL11.glPopMatrix();
}
```

Αυτό που κάνει ο παραπάνω μέθοδος για να ζωγραφίσει το χάρτη είναι:

- Επανέλαβε για όλες τις στήλες:
    - Ζωγράφισε μια στήλη.
    - Μετακινήσου δεξιά.
    - Αν πρόκειται για μονή στήλη, πρόσθεσε μια κατακόρυφη απόκλιση.

Όσον αφορά το selection, κάθε φορά που αλλάζουμε γραμμή η στήλη, εισάγουμε το αντίστοιχο όνομα στην name stack. Έτσι, το κάθε πλακίδιο θα έχει διπλό όνομα, ένα για τη x συντεταγμένη και ένα για την y συντεταγμένη. Αυτές οι συντεταγμένες, το προσδιορίζουν ως μοναδικό, και δεν χρειάζεται να παράγουμε εμείς ένα όνομα για αυτή τη διαδικασία. Με το τέλος του selection, σε περίπτωση που έχει υπάρξει hit, το selection buffer θα έχει για το πλακίδιο που επιλέχτηκε και τα δύο ονόματα που αντιστοιχούνε σε αυτό. Αυτή η διαδικασία είναι πολύ βολική, καθώς το ίδιο το όνομα του πλακιδίου, προσδιορίζει και την θέση του στο χάρτη, και δεν απαιτείται επιπλέον προσπάθεια για να βρεθεί αυτή.

Ο έλεγχος για τον δεύτερο βρόγχο είναι

```
for (int j=-coordOffset; j<yDim-coordOffset; j++)
```

Το coordOffset ξεκινάει από 0 και αυξάνεται κάθε φορά που πρόκειται για μονή στήλη. Έτσι πετυχαίνεται το σύστημα συντεταγμένων που παρουσιάζεται στην εικόνα ...

**Room**

Η κλάση αυτή όπως λέει και το όνομά της, είναι το δωμάτιο στο οποίο θα εξελίσσεται το παιχνίδι. Συγκεκριμένα, ζωγραφίζει το δωμάτιο ως χώρο, τέσσερις τοίχοι το πάτωμα και το ταβάνι με τα αντίστοιχα textures, και όποιο άλλο αντικείμενο υπάρχει μέσα σε αυτό, εκτός του χάρτη και των μονάδων.

Ο κατασκευαστής της κλάσης απαιτεί μήκος, πλάτος και ύψος του ορθογώνιου παραλληλεπίπεδου που αποτελεί το δωμάτιο.

```
public Room(float length, float width, float height)
```

Το μέγεθος είναι δυναμικό, και αλλάζει ανάλογα με τις προτιμήσεις μας, αλλάζοντας αναλογικά και τις θέσεις των αντικειμένων στο δωμάτιο.

Η μέθοδος

```
public void render()
```

αναλαμβάνει να ζωγραφίσει το δωμάτιο. Το δωμάτιο αποτελείται από στατικά και κινούμενα αντικείμενα. Για το λόγο αυτό, τοποθετούμε το στατικό μέρος σε μια display list η οποία καλείται από τη render(). Έπειτα, ζωγραφίζονται τα κινούμενα μέρη, που είναι τα μπαλόνια στο ταβάνι του δωματίου, και το κινούμενο αλογάκι. Αυτά, επειδή έχουνε μεταβλητές που αλλάζουνε σε κάθε ανανέωση της οθόνης, δεν μπορούμε να τα βάλουμε σε κάποια display list, οπότε τα ζωγραφίζουμε ξεχωριστά. Τα περισσότερα αντικείμενα του δωματίου, αποτελούνε φορτωμένα μοντέλα μέσω της κλάσης BaseModel. Ένα παράδειγμα που δείχνει πώς ανακτούμε και ζωγραφίζουμε ένα μοντέλο, είναι το εξής:

```
ModelContainer.get("BED").render();
```

Αυτή η εντολή θα ανακτήσει από τη ModelContainer κλάση το μοντέλο που αντιστοιχεί στο όνομα “BED” και θα την ζωγραφίσει στο σημείο που βρισκόμαστε στη δεδομένη στιγμή.

Τα κινούμενα αντικείμενα, υλοποιήθηκαν ως εξής. Η κίνηση των αντικειμένων είναι μια αέναη επαναλαμβανόμενη κίνηση ανάμεσα σε δύο αντίθετες κατευθύνσεις. Μια τέτοια κίνηση, μπορεί να επιτευχθεί μέσω της συνάρτησης του ημίτονου. Αν θεωρήσουμε ότι το x είναι η γωνία/κλίση του μοντέλου μπαλονιού, και αυτό αυξάνεται σταθερά , τότε το y θα παίρνει συνεχώς τιμές μεταξύ -yMax και yMax, και μάλιστα με τις τιμές να του y να συμπυκνώνονται κοντά στα -yMax και yMax. Αυτός ο μηχανισμός μας παρέχει μια μεταβλητή που παίρνει συνεχώς τιμές ανάμεσα σε ένα διάστημα, με τις τιμές να αυξάνονται γρηγορότερα προς το κέντρο του διαστήματος, και αργότερα στα άκρα του διαστήματος. Ταιριάζει απόλυτα με το εφφέ κίνησης που θέλουμε να πετύχουμε, και γι' αυτό χρησιμοποιούμε αυτή τη μεταβλητή για να περιστρέφουμε κάθε φορά το αντικείμενο, σύμφωνα με τη νέα τιμή. Η ίδια η μέθοδος, παρουσιάζεται μερικώς παρακάτω:

```
private void renderBalloons(){
    xBalloonAngle += BALLOON_MOVEMENT_SPEED;
    if (xBalloonAngle > 360.0f){
        xBalloonAngle = 360.0f - xBalloonAngle;
    }
    float xBalloonPosition = (float) Math.sin(Math.toRadians(xBalloonAngle))
* BALLOON_ANGLE;

. . .

        GL11.glRotatef(xBalloonPosition, 1.0f, 0.0f, 0.0f);
        ModelContainer.get("BALLOON_BLUE").render();

. . .
}
```

Ένα άλλο ενδιαφέρον σημείο εδώ, είναι η υλοποίηση του παραθύρου. Υπάρχουνε αρκετοί τρόποι για να μπορέσει να επιτευχθεί κάτι τέτοιο. Ένας είναι να φτιαχτεί το παραλληλόγραμμο του τοίχου με έχοντας μια παραλληλόγραμμη τρύπα στο σημείο που θέλουμε. Είναι ο πιο εύκολος τρόπος όσον αφορά στη σχεδίαση του παραλληλογράμμου, αλλά όχι όσον αφορά στο texture mapping. Θα έπρεπε να μπορέσει να αντιστοιχηθεί έπειτα το ίδιο παραλληλόγραμμο αρχείο εικόνας σε ένα τέτοιο σχήμα. Άλλος τρόπος είναι σχεδιαστεί

κανονικά το παραλληλόγραμμο του τοίχου, αλλά το texture το οποίο θα τοποθετηθεί πάνω, να περιέχει μια παραλληλόγραμμη τρύπα στο σημείο που θέλουμε. Όταν λέμε τρύπα εδώ, εννοούμε να είναι διαφανές το σημείο αυτό. Όμως έτσι είναι τελείως ανεξάρτητα το texture από το σημείο του παραθύρου στο τοίχο, και μια τυχόν αλλαγή του ενός, θα απαιτεί αλλαγή και του άλλου. Ο τρίτος τρόπος, που είναι και ο πιο ενδιαφέρον και προτιμήθηκε, είναι με τη χρήση του stencil buffer. Το stenceil test αποφασίζει για το αν ένα τμήμα θα ζωγραφιστεί ή όχι ανάλογα με τη σύγκριση μεταξύ stencil buffer και την reference τιμή.

Αρχικά, στη init() της GameWindow ενεργοποιούμε το STENCIL_TEST, ορίζουμε GL_REPLACE ως η την λειτουργία που θα εκτελεστεί σε περίπτωση που το stencil test αποτύχει, και ορίζουμε ότι το stencil buffer θα γεμίζει με μηδενικά όποτε το καθαρίζουμε.

```
// Enable Stencil testing
GL11.glEnable(GL11.GL_STENCIL_TEST);
GL11.glStencilOp(GL11.GL_REPLACE, GL11.GL_REPLACE, GL11.GL_REPLACE);
GL11.glClearStencil(0x0);
```

Μέσα στην render() σχεδιάζουμε το παράθυρο, με τη χρήση της εσωτερικής κλάσης Window:

```
public void renderWindow()
    GL11.glStencilFunc(GL11.GL_EQUAL, 1, 1);

    // Render the "blank" window
    Window win = new Window(-4.0f, 39.0f, -3.5f, 20.7f);
    GL11.glBegin(GL11.GL_QUADS);
       win.render();
    GL11.glEnd();

    GL11.glStencilFunc(GL11.GL_NOTEQUAL, 1, 1);
```

Λέμε ότι η σύγκριση που θα γίνεται θα είναι GL_EQUAL. Δηλαδή το stencil test θα επιτυχγάνει αν το (ref & mask) = (stencil & mask ). Σχεδιάζουμε το παράθυρο μέσω της inner class Window. Στη συγκεκριμένη το test αποτυγχάνει. Και άρα, τα pixels αυτά δεν ζωγραφίζονται. Έπειτα ορίζουμε τη σύγκριση ως GL_NOTEQUAL, και άρα το test θα επιτυγχάνει, και άρα οτιδήποτε σχεδιάζεται από δω και πέρα, θα ζωγραφίζεται στην οθόνη.

## gr.jmanji.toywars.model

### BaseModel

Η κλάση αυτή φορτώνει και ζωγραφίζει 3D μοντέλα. Χρησιμοποιεί το ModelWorldEngine API (http://jerome.jouvie.free.fr/OpenGl/Projects/ModelLoader.php). Βασίζεται σε παραδείγματα που αναφέρονται στο forum του site. Το API αυτό μπορεί να φορτώσει στη μνήμη διαφόρων τύπων μοντέλα όπως 3DS, LWO, OBJ κ.ά.. Έπειτα τα ζωγραφίζει στο υπάρχον OpenGL πλαίσιο. Το toyWars χρησιμοποιεί OBJ μοντέλα, ένα ανοιχτού τύπου format που έχει αναπτυχθεί από την Wavefront Technologies. Τα μοντέλα έχουνε όλα βρεθεί ελεύθερα στο διαδίκτυο και έχουνε προσαρμοστεί ώστε να ταιριάξουνε με τις απαιτήσεις του παιχνιδιού (μέγεθος, προσανατολισμός, αφαίρεση αντικειμένων κ.τ.λ.). Οι μέθοδοι:

```
private void loadForRendering()
private void loadForSelection()
```

φορτώνουν το μοντέλο δύο φορές. Η πρώτη μέθοδος το κάνει για το κανονικό rendering, με τις αντίστοιχες ρυθμίσεις που απαιτούνται (πχ texture loading) ενώ η δεύτερη φορτώνει το μοντέλο με πιο “ελαφριές” ρυθμίσεις, καθώς σε αυτή τη περίπτωση δεν μας ενδιαφέρει η εμφάνιση του μοντέλου. Αντίστοιχα, οι μέθοδοι:

```
public void render()
public void renderSelection()
```

ζωγραφίζουνε το μοντέλο ανάλογα με την περίσταση. Αν πρόκειται για κανονικό rendering ζωγραφίζουμε το μοντέλο που φορτώθηκε από τη loadForRendering() ενώ αν πρόκειται για selection, ζωγραφίζουμε το μοντέλο που φορτώθηκε από τη loadForSelection().

### ModelContainer

Πρόκειται για την “αποθήκη” των μοντέλων. Με τη μέθοδο

```
public static void loadModels
```

τοποθετούνται όλα τα μοντέλα σε ένα hash table, σύμφωνα με ένα όνομα που τους δίνεται

κατά την εισαγωγή. Αυτό θα είναι και το όνομα που θα τα προσδιορίζει από εδώ και πέρα. Αφού εισαχθούνε το hash table, έπειτα φορτώνονται ένα προς ένα. Πρόκειται για μια σχετικά χρονοβόρα διαδικασία, η οποία συμβαίνει κατά την εκκίνηση του παιχνιδιού. Επιπλέον, με την public static BaseModel get(String name) μπορούμε να ανακτούμε όποιο μοντέλο θέλουμε, δίνοντας το όνομά του.

### gr.jmanji.toywars.unit

#### Unit

Πρόκειται για το interface που ακολουθεί η αφηρημένη κλάση AbstractUnit. Οτιδήποτε μπορεί να κάνει μια μονάδα, βρίσκεται σε αυτό το interface.

#### AbstractUnit

Είναι μια αφηρημένη κλάση που υλοποιεί το Unit interface. Αυτή η κλάση κρατάει όλα τα χαρακτηριστικά μιας μονάδας, και εκτελεί όλες της τις λειτουργίες. Ο κατασκευαστής της κλάσης είναι:

```
public AbstractUnit(Player player, BaseModel model, int xTile, int yTile, Map map)
```

player είναι ο παίχτης που θα χειρίζεται αυτή το μονάδα, model το μοντέλο που αντιστοιχεί σε αυτή τη μονάδα, xTile και yTile οι αρχικές του συντεταγμένες στο χάρτη, και map ο ίδιος ο χάρτης. Κατά την αρχικοποίηση του AbstractUnit αντικειμένου, δημιουργούνται δύο display lists, μία για το κανονικό rendering, και μία για το selection.

Στη μέθοδο:

```
public void render(boolean renderHealth)
```

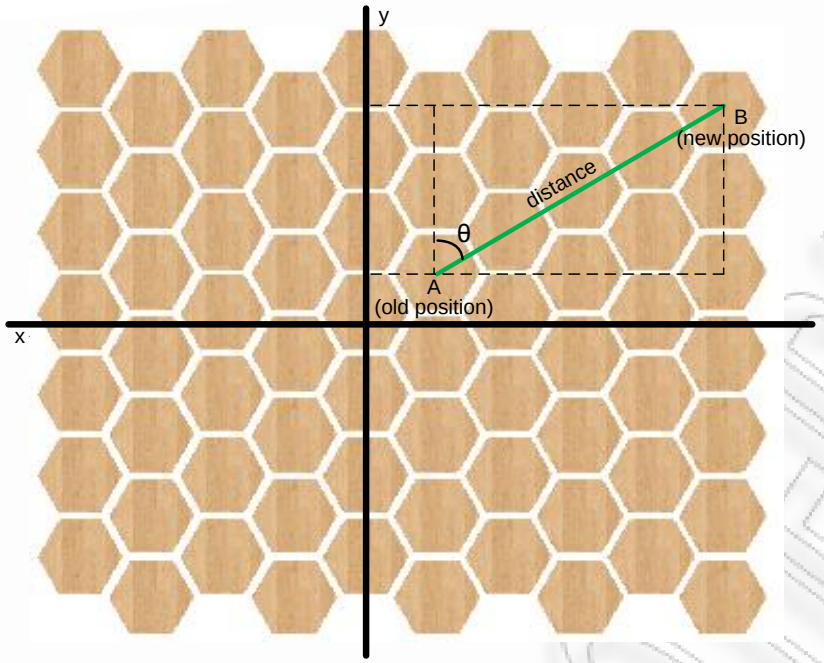
Αρχικά μεταφερόμαστε στο σημείο του χάρτη που θα ζωγραφιστεί η μονάδα. Αν χρειάζεται να εμφανιστούν οι πόντοι ζωής, ζωγραφίζονται σε μια θέση πάνω από το μοντέλο. Επίσης, αν η μονάδα έχει επιλεχθεί από το χρήση ή τον υπολογιστή(AI) τότε θα εμφανιστεί πάνω από το μοντέλο και ένας κέρσορας. Έπειτα αν το μοντέλο πρέπει να στρίψει, εκτελείται η περιστροφή. Τέλος καλείται η display list που έχει φτιαχτεί για rendering.

Η μέθοδος:

```
public void move(int xTileNew, int yTileNew, boolean moveAnimation)
```

μετακινεί τη μονάδα στις x και y συντεταγμένες του χάρτη, αφού πρώτα αποθηκεύσει τις παλιές. Υπάρχει επιλογή για το αν θα υπάρξει εφέ κίνηση κατά τη μετακίνηση της μονάδας ή θα μεταφερθεί στο καινούριο πλακίδιο ακαριαία. Αυτή η επιλογή είναι χρήσιμη για την ακύρωση μιας κίνησης της μονάδας, που κατά την περίπτωση αυτή δεν εκτελείται κάποιο εφέ κίνησης της μονάδας προς τα πίσω.

Το παρακάτω σχήμα βοηθάει στην κατανόηση του υπολογισμού της κίνησης της μονάδας.




**Εικόνα 5.4**

Θεωρούμε ένα σύστημα συντεταγμένων του χάρτη στη οθόνη, όπως αυτό του σχήματος. Για λόγους ευκολίας η z συντεταγμένη δεν λαμβάνεται υπόψη. Όπως φαίνεται στον ορισμό της move(), η νέα θέση της μονάδας είναι οι νέες συντεταγμένες του χάρτη. Για να ζωγραφιστεί όμως η μονάδα στο χάρτη, χρειάζονται οι πραγματικές συντεταγμένες της οθόνης μας, όχι οι θέση του πλακιδίου στο χάρτη. Για μια μονάδα που θέλει να κινηθεί από ένα πλακίδιο A σε ένα πλακίδιο B, οι νέες θέσεις υπολογίζονται ως εξής:

```
xNewPosition = xTile*map.getHDistance();
yNewPosition = (xTile/2 + yTile)*map.getVDistance();
if (xTile%2 != 0 ){
    yNewPosition = yNewPosition + map.getVDistance()/2;
}
```

Αν δεν θέλαμε να δείξουμε ότι η μονάδα όντος κινείται προς ένα νέο πλακίδιο, τότε αυτές οι τιμές είναι αρκετές. Απλά ζωγραφίζουμε τη μονάδα κατευθείαν στις νέες αυτές συντεταγμένες της οθόνης.

Η απόσταση στο χώρο μεταξύ της παλιάς και της καινούριας θέσης, δηλαδή μεταξύ A και B, υπολογίζεται από τον γνωστό τύπο της απόστασης δύο σημείων:

```
distance = (float) Math.sqrt((xNewPosition-xOldPosition)*(xNewPosition-
xOldPosition) + (yNewPosition-yOldPosition)*(yNewPosition-yOldPosition));
```

Στη συνέχεια υπολογίζουμε τη γωνία κατά την οποία μετατοπίζεται η μονάδα. Αυτή θα μας χρειαστεί για μπορέσουμε να δημιουργήσουμε και το εφέ της κίνησης, αλλά και αυτό της περιστροφή της μονάδας, προτού αυτή μετακινηθεί. Θεωρούμε ότι η μονάδα του σχήματος 1-4 κοιτάζει προς το +y. Όταν λοιπόν θέλει να μετακινηθεί προς ένα πλακίδιο και χρειαστεί να στρίψει, θα στρίψει κατά την γωνία που σχηματίζεται από τον κατακόρυφο άξονα και την γραμμή μεταξύ της παλιάς και νέας θέσης. Στο σχήμα η γωνία αυτή είναι η θ. Εφόσον γνωρίζουμε τις θέσεις A και B, υπολογίζουμε την γωνία θ, και έπειτα τα ημίτονα και συνημίτονα της γωνίας που θα χρειαστούν.

Για την περιστροφή, υπολογίζουμε την φορά, και ξεκινώντας από 0, σε κάθε ανανέωση της οθόνης προσθέτουμε ή αφαιρούμε ανάλογα μια μικρή τιμή. Συνεχίζουμε μέχρι να φτάσουμε στη γωνία θ.

```
currentRotationAngle = currentRotationAngle - ROTATION_SPEED;
```

Αφού ολοκληρωθεί η περιστροφή εκτελείται το εφέ της κίνησης. Σε κάθε ανανέωση της μονάδας υπολογίζουμε τη νέα θέση(πάνω στην πράσινη γραμμή του σχήματος), και συνεχίζουμε μέχρι να φτάσουμε στη νέα τελική θέση.

```
xCurrentPosition = xCurrentPosition + distance * cosa * MOVEMENT_SPEED;
yCurrentPosition = yCurrentPosition + distance * sina * MOVEMENT_SPEED;
```

Η κλάση αυτή υλοποιεί πολλές μεθόδους που αφορούνε τη λογική του παιχνιδιού. Ελέγχει αν μια μονάδα μπορεί να κινηθεί ή να επιτεθεί, κινεί η κάνει την μονάδα να επιτεθεί, ελέγχει για θάνατο της μονάδας, ψάχνει για κοντινούς εχθρούς κ.ά. Αυτό το κάνει κυρίως με τη δημιουργία και επιστροφή λιστών με την αντίστοιχη πληροφορία μέσα. Για παράδειγμα, θα δούμε μια τέτοια μέθοδο:

```
public ArrayList getEnemiesList(UnitManager unitManager){
    ArrayList <Unit> unitsList = unitManager.getUnitList();
    ArrayList enemiesList = new ArrayList();
    for(Unit unit:unitsList){
        if (this.isEnemy(unit)){
            enemiesList.add(unit);
        }
    }
    return enemiesList;
}
```

Αυτή η μέθοδος κάνει κάτι πολύ απλό, επιστρέφει μια λίστα με όλους τους εχθρούς της μονάδας. Όσες μονάδες δηλαδή ανήκουνε στον αντίπαλο παίχτη, προστίθενται στην λίστα η οποία έπειτα επιστρέφεται.

### gr.jmanji.toywars.player

Αυτό το πακέτο περιέχει σχεδόν όλη τη λογική που αφορά το ίδιο το παιχνίδι στρατηγικής. Ως player εννοούμε την οντότητα που χειρίζεται μια ομάδα από παιχνίδια(μονάδες), είτε αυτός είναι ένας κανονικός άνθρωπος, είτε ο υπολογιστής.

#### Player

Η αφηρημένη κλάση Player έχει τις βασικές μεταβλητές και μεθόδους που χρησιμοποιούνε οι παίχτες του παιχνιδιού. Επειδή είναι abstract, δεν μπορεί να υπάρξει τέτοιο αντικείμενο, καθώς θα πρέπει να ορισθεί αν πρόκειται για κανονικό παίχτη, ή παίχτη που χειρίζεται η A.I. του παιχνιδιού. Βασική μέθοδος εδώ που θα πρέπει να γίνει override από τις κλάσεις που την επεκτείνουν, είναι η:

```
public void play()
```

η οποία όπως λέει, επιτρέπει στον παίκτη να παίξει. Γίνεται override από τις κλάσεις που την επεκτείνουν, καθώς ο τρόπος που παίζει ο ανθρώπινος παίκτης και ο υπολογιστής, είναι τελείως διαφορετικός.

```
public Player(Map map, UnitManager unitManager, Direction direction)
```

Κατά τη δημιουργία μας κλάσης παίκτη, χρειάζεται να δοθεί ο χάρτης του παιχνιδιού, ο unit manager, και προς τα ποια κατεύθυνση θα κοιτάνε οι μονάδες του παίχτη.

#### HumanPlayer

Πρόκειται για τον ανθρώπινο παίκτη του παιχνιδιού, αυτόν δηλαδή που χειρίζεται ο χρήστης. Η play() είναι σχετικά πολύπλοκη, εδώ θα αναφερθεί η γενική ιδέα. Όσον αφορά τη κατάσταση του ίδιου του παιχνιδιού, αυτή χωρίζεται σε 3 modes, normal, move και attack. Στο κανονικό mode δεν συμβαίνει τίποτα, το παιχνίδι περιμένει τον χρήστη να επιλέξει μια μονάδα. Μόλις επιλέξει κάποια από τις μονάδες του, τότε το mode μετατρέπεται σε move, και μπορεί να κινήσει την μονάδα που επέλεξε. Αφού τη μετακινήσει, το παιχίδι εισέρχεται σε mode επίθεσης, όπου μπορεί να επιλέξει αν και σε ποιά μονάδα θα επιτεθεί. Πατώντας το Backspace, μπορεί να μεταβεί κάθε φορά στο προηγούμενο mode, εκτός αν είναι στο κανονικό. Όλα αυτα υλοποιούνται με μια σειρά από ελέγχους, και μεθόδους της AbstractUnit.

#### AIPlayer

Είναι η κλάση που αναλαμβάνει την τεχνητή νοημοσύνη του παιχνιδιού. Πρόκειται για τον παίχτη που χειρίζεται ο υπολογιστής. Δεν χρησιμοποιείται κάποιος γνωστός αλγόριθμος, αλλά μια απλή και προβλέψιμη στρατηγική, αρκετή ώστε να είναι επιθετική και νικηφόρα για κάποιον άπειρο παίχτη. Ο αλγόριθμος που ακολουθεί κάθε μονάδα κατά τη σειρά της στη play(), είναι ο εξής:

- Όσο είναι η σειρά της μονάδας να παίξει:

- Αν το mode είναι normal:
  - Αν υπάρχει εχθρική μονάδα στην εμβέλεια επίθεσης, γύρισε σε attack mode.
  - Αν όχι, γύρισε σε move mode.
- Αν το mode είναι move:
  - Βρες την κοντινότερη εχθρική μονάδα.
  - Φτιάξε μια ταξινομημένη λίστα με τις πιθανές θέσεις μετακίνησης της τρέχουσας μονάδας, από την πιο κοντινή στην πιο μακρινή σε σχέση με την κοντινότερη εχθρική μονάδα.
  - Μέσα στη λίστα, ψάξε για μια θέση προς μετακίνηση, με προτίμηση μια θέση όπου η απόστασή της από την εχθρική μονάδα να ισούται με το βεληνεκές επίθεσης της τρέχουσας μονάδας.
  - Μετακίνησε τη μονάδα σε αυτή τη θέση και γύρισε το mode σε move.
- Αν το mode είναι attack:
  - Κάνε επίθεση στην κοντινότερη εχθρική μονάδα.

### gr.jmanji.toywars.texture

Το πακέτο αυτό περιέχει που αποθηκεύουν, φορτώνουν και επιστρέφουν textures. Κάνει χρήση του org.newdawn.spaceinvaders.lwjgl, ένα πακέτο τρίτου που παρέχει όλο το υπόβαθρο για αυτές τις διεργασίες.

#### Textures

Αυτή η αφηρημένη (abstract) περιέχει τις ονομασίες όλων των textures που χρησιμοποιεί το παιχνίδι. Αν κάποια κλάση θα θέλει να φορτώσει μια εικόνα, θα πρέπει να υπάρχει εδώ.

#### TextureLoader

Φορτώνει κάθε texture που έχει δηλωθεί στην κλάση Textures, με την χρήση της getTexture() της κλάσης org.newdawn.spaceinvaders.lwjgl.TextureLoader.

#### Screenshot

Η κλάση αυτή χρησιμοποιείται για την παραγωγή ενός εφέ που απαιτεί η κλάση gr.jmanji.toywars.state.Menu. Αυτό που κάνει είναι να μεταφέρει όλα τα pixels της τρέχουσας οθόνης, σε ένα αντικείμενο Texture.

Οι μέθοδοι που ορίζει η κλάση αυτή είναι:

```
public void take()
private BufferedImage screenShotImage(int width, int height)
public void bind()
```

Η πρώτη μέθοδος καλείται κάθε φορά που θέλουμε να πάρουμε μια «φωτογραφία» της οθόνης μας. Αυτή καλεί την screenShotImage η οποία αντιγράφει στη μνήμη σε ένα αντικείμενο της κλάσης BufferedImage (της βιβλιοθήκης awt) τον αριθμό των pixels θα ζητηθεί, μέσω της glReadPixels(). Τα pixels αυτά θα πρέπει να αντιστραφούν στον κατακόρυφο άξονα λόγω διαφορετικής υλοποίησης OpenGl και Java. Έπειτα το BufferedImage αντικείμενο χρησιμοποιείται για να φτιαχτεί ένα texture. Η bind() τέλος, δεσμεύει αυτό το texture για mapping.

### gr.jmanji.toywars.sound

#### Sounds

Περιέχει απλά τα ονόματα των μεταβλητών που χρησιμοποιούνται για τους ήχους του παιχνιδιού.

### gr.jmanji.toywars.config

#### ConfigReader

Η κλάση αυτή μπορεί να διαβάζει τις ρυθμίσεις που χρειάζονται για το Display αντικείμενο. Οι ρυθμίσεις αυτές βρίσκονται σε ένα xml αρχείο, στο ίδιο directory στο οποίο βρίσκεται και το τελικό εκτελέσιμο jar. Η μορφή του xml αρχείου, είναι η εξής:

```
<?xml version="1.0"?>
```

```
<display>
    <width>1024</width>
    <height>768</height>
    <fullscreen>false</fullscreen>
    <vsync>false</vsync>
</display>
```

Έτσι ο χρήστης μπορεί να εισάγει τις ρυθμίσεις που επιθυμεί σε αυτό το αρχείο, πρωτού τρέξει το παιχνίδι. Οι τιμές που μπορούν να πάρουνε δεν είναι φυσικά αυθαίρετες, αλλά καθορίζονται από την κάρτα γραφικών και οθόνη του μηχανήματος στο οποίο τρέχει το παιχνίδι. Αν οι επιλογές μας σε αυτό το αρχείο δεν συμβαδίζουν με αυτές που υποστηρίζει ο οδηγός της κάρτας γραφικών, το παιχνίδι δεν θα τρέξει και εμφανιστεί ένα μήνυμα λάθους.

## gr.jmanji.toywars.font

### TrueTypeFont

Με την κλάση αυτή μπορούμε να γράφουμε γράμματα σε δισδιάστατη ή τρισδιάστατη απεικόνιση. Οι γραμματοσειρές που υποστηρίζονται εξαρτώνται από το σύστημα στο οποίο τρέχει το πρόγραμμα, και συγκεκριμένα από το πίνακα Font[] που επιστρέφει η μέθοδος

GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts() της awt βιβλιοθήκης. Ο κώδικας βρέθηκε διαθέσιμος στο forum της lwjgl, και χρησιμοποιήθηκε απαράλλαχτος.

## gr.jmanji.toywars

### Camera

Η κλάση αυτή όπως δηλώνει και το όνομά της αποτελεί την κάμερα του παιχνιδιού. Αλλάζει την θέση και γωνία λήψης ανάλογα με το input που δέχεται από το χρήστη.

Ο κατασκευαστής:

```
public Camera(Room room, Map map)
```

απαιτεί το δωμάτιο και το χάρτη του παιχνιδιού, ώστε να μπορέσει η κάμερα να γνωρίζει τα όρια μέσα στο οποία θα κινείται. Κατά το κανονικό mode, αυτά θα είναι τα όρια του χάρτη, ενώ στο ελεύθερο mode(που θα εξηγηθεί παρακάτω), αυτά είναι τα όρια του δωματίου.

Πρόκειται για ένα από τα δυσκολότερα μέρη της διπλωματικής, και απαιτεί γνώση των τριγωνομετρικών συναρτήσεων και το μετασχηματισμών της OpenGL.

Αρχικά θα πρέπει να υπάρξει ένα input από το χρήστη. Η μέθοδος:

```
public void getInput()
```

αναλαμβάνει να πιάσει και την είσοδο από το πληκτρολόγιο, και από το ποντίκι. Στην περίπτωση που υπάρχει μια κίνηση του ποντικιού ενώ είναι πατημένο το δεξί κλικ, με τις Mouse.getX() και Mouse.gety() ελέγχουμε τις νέες θέσεις του ποντικιού, και υπολογίζουμε τις διαφορές:

```
xCurrent = Mouse.getX();
yCurrent = Mouse.getY();
xDiff = xCurrent - xPrevious;
yDiff = yCurrent - yPrevious;
xPrevious = xCurrent;
yPrevious = yCurrent;
xAngle = xAngle + xDiff*MOUSE_SENSITIVITY;

...

yAngle = yAngle - yDiff*MOUSE_SENSITIVITY;
```

Οι xAngle και yAngle είναι οι γωνίες κατά τις οποίες θα περιστραφεί όλη η σκηνή αργότερα.

Σε περίπτωση που πατηθεί κάποιο πλήκτρο κατεύθυνσης, θα πρέπει να υπολογιστούν οι νέες θέσεις της κάμερα, σε σχέση με την κίνησή της πάνω στο χάρτη. Για παράδειγμα, αν πατηθεί το πλήκτρο «πάνω», τότε αυτές υπολογίζονται ως εξής:

```
if (Keyboard.isKeyDown(Keyboard.KEY_UP)) {
    xPos += (float)(Math.sin(Math.toRadians(xAngle))*KEY_SENSITIVITY);
    yPos += (float)(Math.cos(Math.toRadians(xAngle))*KEY_SENSITIVITY);
    zPos += (float)(Math.cos(Math.toRadians(yAngle))*KEY_SENSITIVITY);
```

```
}
```

Επιπλέον, έχει προστεθεί και ένα εφέ το οποίο εξομαλύνει την μετάβαση κατά το zoom, όταν αυτό τελειώνει. Στα περισσότερα παιχνίδια του είδους, όταν πατηθεί το πλήκτρο για το zoom, και μετα ο χρήστης το αφήσει, η κίνηση της κάμερας δεν σταματάει ακαριαία, αλλά υπάρχει μια περίοδος που κινείται όλο και πιο αργά, μέχρι να σταματήσει τελείως. Αυτό υλοποιείται ως εξής:

Αν κάνουμε zoom in και αφεθεί το πλήκτρο θα μετακινηθεί η κάμερα μέχρι το σημείο που θέλουμε Έπειτα, δίνουμε ένα περιθώριο στο τελικό zoom

```
finalZoom = zoom - ZOOM_OFFSET;
```

Το zoom θα μειώνεται όλο και περισσότερο, μέχρι να γίνει ίδιο με το τελικό.

```
zoom -= ( (zoom - finalZoom) * ZOOM_OFFSET_SPEED);
```

Τέλος, σε κάθε ανανέωση της οθόνης καλείται η

```
public void move()
```

η οποία ανάλογα με το mode της κάμερας καλεί τις

```
public void moveNormal()
private void moveFree()
```

αντίστοιχα. Μέσα σε αυτές συμβαίνουν οι περιστροφές και οι μετακινήσεις που θα αλλάξουν τη θέση και τη γωνία της κάμερας. Φυσικά, η σειρά με την οποία συμβαίνουνε αυτά, έχει τεράστια σημασία. Δύο ιδιαίτερες μέθοδοι εδώ είναι οι:

```
public static void moveBack{
public static void moveBackWithZoom()
```

οι οποίες υλοποιούν το εφέ που κάνουν ένα αντικείμενο να «κοιτάζει» μονίμως την οθόνη. Αυτό χρησιμοποιείται στις συμβολοσειρές με του πόντους ζωής που έχει μια μονάδα, και στις πόνους ζωής που χάνει μετά από κάποια επίθεση. Μάλιστα, η moveBackWithZoom() προσθέτει και ένα επιπλέον zoom όσο απομακρυνόμαστε από την μονάδα.

**Messenger**

Η κλάση αυτή έχει φτιαχτεί για να μεταφέρει τα διάφορα μηνύματα στο παιχνίδι. Κύρια χρήση είναι για την εμφάνιση των μηνυμάτων στο κάτω μέρος της οθόνης, κάθε φορά που μια μονάδα επιτίθεται, ή όταν τελειώνει το παιχνίδι.

**Constants**

Η κλάση αυτή περιέχει διάφορες σταθερές (enum) που χρησιμοποιούνται σε διάφορα σημεία του κώδικα.

# 6. Μελέτη Περίπτωσης

## 6.1. Πώς τρέχει το παιχνίδι

Ο φάκελος του παιχνιδιού περιέχει τα εξής:

Φάκελος lib: Οι βιβλιοθήκες που χρησιμοποιεί το παιχνίδι.
XML αρχείο config.xml: xml αρχείο ρυθμίσεων.
Αρχείο κειμένου README.txt: Κείμενο με οδηγίες και ιστορικό bug fixes
Εκτελέσιμο αρχείο run.bat: Εκτελέσιμο αρχείο του παιχνίδιού, σε Windows.
Εκτελέσιμο αρχείο run.sh: Εκτελέσιμο αρχείο του παιχνίδιού, σε Linux.
jar αρχείο toyWars_<έκδοση>.jar: Το παραχθέν jar.

Το αρχείο config.xml περιέχει κάποιες βασικές ρυθμίσεις για το παράθυρο του παιχνιδιού. Μπορούμε να το επεξεργαστούμε, για να αλλάξουμε την ανάλυση, να επιλέξουμε αν θέλουμε fullscreen και vertical sync. Για να τρέξουμε το παιχνίδι απλά τρέχουμε το αντίστοιχο εκτελέσιμο αρχείο. Σε περιβάλλον Windows κάνουμε διπλό κλικ στο start.bat. Σε περιβάλλον Linux, μεταφερόμαστε στον κατάλογο του παιχνιδιού, και τρέχουμε σε κονσόλα “sh start.sh” (για bash). Περιμένουμε λίγο να φορτώσει το παιχνίδι, και ύστερα μπορούμε να παίξουμε.

## 6.2. Παράδειγμα χρήσης του παιχνιδιού

Η πρώτη εικόνα που θα αντικρίσουμε αφού φορτώσει το παιχνίδι, είναι η παρακάτω:


**Εικόνα 6.1 - Εισαγωγική οθόνη**

Οι βασικές επιλογές, επιλογές «Play», «Help» και «Exit» γίνονται highlight με κίτρινο χρώμα όταν το βελάκι του ποντικιού πηγαίνει πάνω σε κάποια από αυτές ή όταν πλοηγηθούμε σε κάποια με τα βελάκια του πληκτρολογίου. Με αριστερό κλικ ή το πλήκτρο «Enter» μπορούμε να επιλέξουμε κάποια. Επιλέγοντας το «Exit», βγαίνουμε από το παιχνίδι. Επιλέγοντας το «Help», πηγαίνουμε στην παρακάτω οθόνη:


**Εικόνα 6.2 - Οθόνη Help**

Εδώ περιγράφεται συνοπτικά ο χειρισμός του παιχνιδιού. Επιλέγοντας το «Back», Επιστρέφουμε στην προηγούμενη εισαγωγική οθόνη. Αν στην οθόνη της εικόνας 6.1 επιλέξουμε «Play», τότε ξεκινάει το παιχνίδι.

**Εικόνα 6.3 - Το παιχνίδι ξεκίνησε**

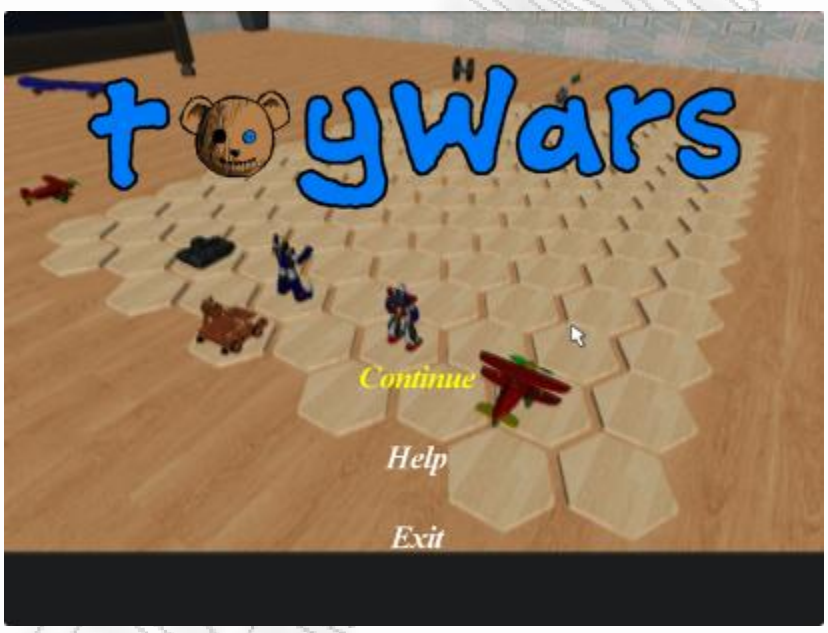
Εδώ βλέπουμε τον χάρτη του παιχνιδιού μέσα στο δωμάτιο, με τις δύο αντιμέτωπες ομάδες παιχνιδιών. Η δική μας ομάδα είναι αυτή που βρίσκεται πιο κοντά στην κάμερα. Μπορούμε να αλλάξουμε τη θέση και τη γωνία της κάμερας ανά πάσα στιγμή. Με τα βελάκια του πληκτρολογίου μετακινούμαστε πάνω στο χάρτη. Έχοντας πατημένο το δεξί κλικ και μετακινώντας το ποντίκι, αλλάζουμε την γωνία της κάμερας. Αν θέλουμε να έχουμε μια καλύτερη οπτική του δωματίου, πατώντας το «c», αλλάζει ο τύπος κάμερας από «κανονικός» σε «ελεύθερος». Με τα βελάκια του πληκτρολογίου μεταφερόμαστε όπου θέλουμε στο δωμάτιο, και κοιτάμε προς όποια κατεύθυνση θέλουμε με το δεξί κλικ πατημένο και κινώντας το ποντίκι.

**Εικόνα 6.4 – «Ελεύθερη» κάμερα**

Με το πλήκτρο «c» επιστρέφουμε στο κανονικό τύπο κάμερας. Ανά πάσα στιγμή, μπορούμε να δούμε πόσους πόντους ζωής έχουνε όλες οι μονάδες που βρίσκονται στο χάρτη πατώντας το πλήκτρο «h».

**Εικόνα 6.5 - Εμφάνιση πόντων ζωής**

Επιπλέον επιλογές που έχει ο παίχτης, είναι να μεταβεί σε fullscreen πατώντας το πλήκτρο «f», να σταματήσει ή να συνεχίσει τη μουσική πατώντας «m», και να αντιστρέψει την κάθετη κίνηση του ποντικιού πατώντας το «i». Αν ο παίχτης θέλει να «παγώσει» το παιχνίδι, τότε μπορεί να πατήσει το πλήκτρο «Escape». Η μουσική σταματάει, και η επόμενη εικόνα θα παρουσιαστεί, με φόντο το εκάστοτε στιγμιότυπο του παιχνιδιού.




**Εικόνα 6.6 - Οθόνη Pause**

Οι επιλογές εδώ μοιάζουν με αυτές τις αρχική εισαγωγικής οθόνης. Για να επιστρέψουμε στο παιχνίδι επιλέγουμε με τον ίδιο τρόπο το «Continue», είτε πατάμε απλά πάλι το πλήκτρο «Escape».

Στην παρτίδα αυτή, ο ανθρώπινος παίχτης παίζει πρώτος. Μπορούμε να επιλέξουμε όποια από τις μονάδες μας θέλουμε. Μόλις το κάνουμε αυτό, θα δούμε τα πλακίδια στα οποία αυτή μπορεί να μετακινηθεί.

**Εικόνα 6.7 - Μετακίνηση μονάδας**

Τα πλακίδια στα οποία μπορεί να μετακινηθεί η μονάδα αναβοσβήνουν με χρώμα μπλε. Αν θέλουμε να ακυρώσουμε την επιλογή αυτής της μονάδας, μπορούμε είτε να πατήσουμε το πλήκτρο «Backspace», ή απλά να επιλέξουμε με το ποντίκι κατευθείαν κάποια άλλη από τις μονάδες μας που δεν έχει ακόμα παίξει. Αν θέλουμε μην μετακινηθούμε, τότε κάνουμε skip αυτό το mode πατώντας το πλήκτρο «Enter».

Όταν επιλέξουμε με το ποντίκι κάποιο από τα μπλε πλακίδια, τότε η μονάδα μετακινείται προς αυτό. Είτε μετακινηθούμε είτε πατήσουμε «Enter», η μονάδα αμέσως μπαίνει σε mode επίθεσης.

**Εικόνα 6.8 - Επίθεση μονάδας**

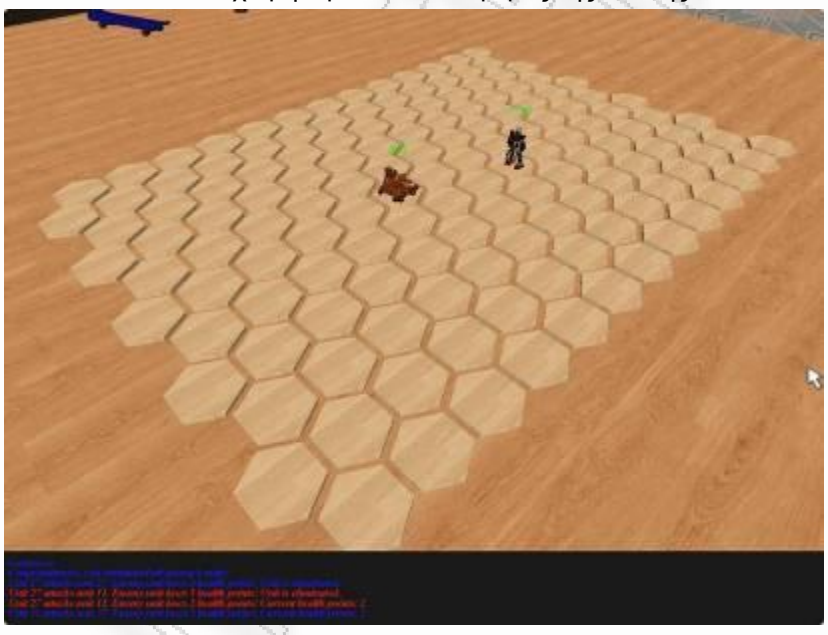
Σε αυτό το mode, τα κόκκινα πλακίδια που αναβοσβήνουν δείχνουν σε ποια πλακίδια μπορεί να επιτεθεί η μονάδα μας. Αν θέλουμε να ακυρώσουμε αυτή τη μετακίνηση και μεταβίβαση σε mode επίθεσης, τότε πατάμε το πλήκτρο «Backspace» και η μονάδα τοποθετείται αμέσως στο τελευταίο πλακίδιο που ήταν, σε mode κίνησης. Αν είμαστε ικανοποιημένοι με την επιλογή μας και θέλουμε να επιτεθούμε σε κάποια αντίπαλη μονάδα, θα πρέπει αυτή να βρίσκεται πάνω σε κάποιο από τα κόκκινα πλακίδια, και να την επιλέξουμε με το ποντίκι. Αν δεν υπάρχει

τρόπος να επιτεθούμε σε κάποια αντίπαλη μονάδα, ή δεν θέλουμε να επιτεθούμε, κάνουμε skip αυτό το mode, και άρα τελειώνει και η σειρά της μονάδας, πατώντας το πλήκτρο «Enter».

Καθώς προχωράει το παιχνίδι, στο κάτω μέρος της οθόνης θα εμφανίζονται μηνύματα, με τα χτυπήματα που δέχονται οι μονάδες. Με μπλε είναι τα χτυπήματα που κάνουν οι δικές μας μονάδες, και με κόκκινο τα χτυπήματα των αντιπάλων μονάδων.

**Εικόνα 6.9 - Εμφάνιση μηνυμάτων**

Το παιχνίδι τελειώνει όταν κάποια από τις ομάδες καταστρέψει όλες τις μονάδες της αντίπαλης μονάδας. Τότε στο χάρτη θα έχουνε μείνει μόνο μονάδας της νικητήριας ομάδας και θα τυπωθεί το αντίστοιχο μήνυμα στο κάτω μέρος της οθόνης.



**Εικόνα 6.10 - Τερματισμός παιχνιδιού**

# 7. Συμπεράσματα

## 7.1. Σύνοψη της διπλωματικής

Στην παρούσα διπλωματική διατριβή, ολοκληρώθηκε η ανάπτυξη ενός τρισδιάστατου παιχνιδιού στρατηγικής. Η δημιουργία του παιχνιδιού ξεκίνησε με προσδιορισμό των απαιτήσεων και της ταυτότητας του παιχνιδιού. Αφού επιλέχθηκε το είδος του παιχνιδιού και έγινε ένας σχεδιασμός σχετικά με τον τρόπο παιχνιδιού, ακολούθησε έρευνα πάνω στις δυνατές τεχνολογίες που μπορούν να χρησιμοποιηθούν.

Η υλοποίηση έγινε τελικά με το OpenGL API, και για γλώσσα προγραμματισμού χρησιμοποιήθηκε η Java. Τα προβλήματα που προέκυψαν κατά την υλοποίηση ήταν πολλά, αλλά μετά από συστηματική ενασχόληση και τη βοήθεια της κοινότητας όποτε αυτή χρειάστηκε, ξεπεράστηκαν.

Το παραδοτέο παιχνίδι που προέκυψε, είναι ηλεκτρονικό παιχνίδι λειτουργικό από την αρχή έως το τέλος, με πολύ απλά γραφικά και νοημοσύνη, και σίγουρα υπάρχει χώρος για πολύ περισσότερη εξέλιξη.

## 7.2. Περαιτέρω εξέλιξη

Η υπάρχουσα υλοποίηση μπορεί να διευρυνθεί με ποικίλους τρόπους. Καταρχήν το ίδιο το παιχνίδι δεν θεωρείται σε καμία περίπτωση ότι έχει φτάσει στη τελική παραδοτέα έκδοση, παρά μόνο βρίσκεται σε δοκιμαστική. Πολλοί τομείς μπορούν να υποστούνε σημαντική βελτίωση. Η παρουσίαση και τα γραφικά θα μπορούσαν να γίνουν πιο ευχάριστα, με εισαγωγή περισσότερων μοντέλων και χρήση διαφόρων εφέ. Επιπλέον η ύπαρξη σκιάς, μια πολύ ενδιαφέρουσα τεχνική στην OpenGL η οποία δεν πρόλαβε να υλοποιηθεί, θα έκανε την παρουσίαση του παιχνιδιού ρεαλιστικότερη. Ακόμα, η διεπαφή του χρήστη στην παρούσα φάση δεν είναι αρκετή εύχρηστη. Θα πρέπει ο χρήστης να ξέρει από πριν τις λειτουργίες του πληκτρολογίου και του ποντικιού, ενώ για να μπορέσει να παίξει το παιχνίδι θα πρέπει να συνδυάσει οπωσδήποτε και τα δύο, πράγμα όχι απαραίτητο. Ο ήχος του παιχνιδιού σε αυτή τη φάση είναι υποτυπώδης. Υπάρχει όλο το υπόβαθρο για προσθήκη επιπλέον μουσικής και ήχων.

Στον τομέα της τεχνητής νοημοσύνης, το παιχνίδι μπορεί να δεχτεί πολλές βελτιώσεις. Η τωρινή νοημοσύνη του υπολογιστή είναι πολύ βασική. Σε γενικές γραμμές, απλά ψάχνει για την κοντινότερη αντίπαλη μονάδα και αν μπορεί την πλησιάζει και της επιτίθεται. Σίγουρα μπορούν να εφαρμοστούν πολύ πιο εκλεπτυσμένοι αλγόριθμοι που να παίρνουν αποφάσεις για την επόμενη κίνηση του υπολογιστή, όπως γνωστοί αλγόριθμοι αναζήτησης σαν τον A*.

Μια πολύ ενδιαφέρουσα προσθήκη θα ήτανε η επιλογή online παιχνιδιού. Αντί να παίζει ένας άνθρωπος με τον υπολογιστή, να παρέχεται η δυνατότητα να μπορούν να παίξουν δύο ή και παραπάνω άνθρωποι μέσω δικτύου.

Ο κώδικας του παιχνιδιού μπορεί να χρησιμοποιηθεί σαν πλαίσιο για άλλα παιχνίδια. Θα μπορούσε για παράδειγμα να γίνει χρήση μόνο του κώδικα που αναφέρεται στο rendering δισδιάστατων εικόνων. Επίσης, αρκετές φορές κατά την ανάπτυξη, παρατηρήθηκε ότι η μετάβαση από αυτό το είδος σε κάποιο άλλο, γίνεται πιο εύκολα χρησιμοποιώντας τον ίδιο κώδικα που χρησιμοποιεί το toyWars. Η ελεύθερη κάμερα μοιάζει αρκετά με κάμερα ενός παιχνιδιού πρώτου προσώπου, και με μια μικρή αλλαγή μπορεί να λειτουργήσει εύκολα ως τέτοια.

# 8. Αναφορές


Astle, Dave, Kevin Hawkins, *Beginning OpenGL Game Programming,* Premier Press, 2004

Aylett, Ruth, Marc Cavazza, *Intelligent Virtual Environments – A State-of-the-art Report,* United Kingdom, 2001

Aylett, Ruth, Michael Luck, *Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments.* Applied Artificial Intelligence, Volume 14, 2000

Badler, N, C Phillips, B Webber, *Simulating Humans: Computer Graphics Animation and Control,* Oxford University Press, 1993

Bourg, David M., Glenn Seeman, *AI for Game Developers,* O'Reilly, 2004

Burbage, David J, *comp.ai.games FAQ,* 2000, http://www.gamedev.net/reference/articles/article200.asp

Clark Verbrugge, Hex Grids, 1997, http://www-cs-students.stanford.edu/~amitp/Articles/HexLOS.html

Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis, *OpenGL Programming Guide (6th edition)*, Addison-Wesley, 2007

Edge, *A Brief History of 3D,* 2008, http://www.next-gen.biz/magazine/a-brief-history-3d

Ellis, S.R., M. Kaiser, A.J. Grunwald, *Pictoral Communication in virtual and real environments,* Taylor and Francis, 1993.

Esa, *Video Games & the Economy,* 2009, http://www.theesa.com/gamesindailylife/economy.asp

Gigante, *Virtual Reality: definitions, history and applications,* Academic-Press, 1993

Guzder, Deena, *Video Games: Still Booming in a Bad Economy,* 2008, http://www.time.com/time/business/article/0,8599,1868630,00.html

Haugeland, John. *Hex Grids,* 2004, http://sc.tri-bit.com/Hex_Grids

Loeffler, T. Anderson, *The Virtual Reality Casebook,* New York: Nostrand Reinhold, 1994

LWJGL, *LWJGL Documentation,* 2009, http://lwjgl.org/documentation.php

Meyer, Nathaniel, *Finite State Machine Tutoria,.* 2003, http://www.generation5.org/content/2003/FSM_Tutorial.asp

Patel, Amit, *Hexagonal Grids.* 2006, http://www-cs-students.stanford.edu/~amitp/gameprog.html#hex.

Russell, Stuart, Peter Norvig, *Artificial Intelligence (3rd Edition),* Prentice Hall, 2009

TDA, *The History of Real Time Strategy,* 2008, http://www.gamereplays.org/portals.php?show=page&name=the_history_of_real_time_strategy_pt1

Thalmann, Nadia Magnenat, Daniel Thalmann, *Artificial Life and Virtual Reality,* Chichester: John Wiley & Sons, 1994

Walker, Mark H, *Strategy Gaming,* http://archive.gamespy.com/articles/february02/strategy1/index.shtm.

Ward, Jeff, *What is a Game Engine?* 2008

Wrigh, Richard S., Benjamin Lipchak, Nicholas Haemel, *OpenGL SuperBible (4rth edition),* Addison-Wesley, 2007

Zeltzer, David, *Autonomy, interaction, and presence,* Cambridge, MIT Press, 1992

Αναγνώστου, Κώστας, *Εισαγωγή στην OpenGL.* 2008, http://videogameslab.wordpress.com/2008/11/05/opengl_intro1/