ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΜΑΛΚΟΓΙΑΝΝΗΣ

«ΑΝΑΠΤΥΞΗ ΤΡΙΣΔΙΑΣΤΑΤΟΥ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΒΕΛΤΙΣΤΟΠΟΙΗΜΕΝΟ ΣΧΕΔΙΑΣΜΟ ΔΙΑΔΡΟΜΩΝ»

Εξεταστική επιτροπή

Επ. Καθ. Αικατερίνη Μανιά (επιβλέπουσα)
Επ. Καθ. Μιχαήλ Λαγουδάκης
Επ. Καθ. Βασίλειος Σαμολαδάς

Χανιά 2010

ΠΕΡΙΛΗΨΗ

Η εργασία αυτή παρουσιάζει την ανάπτυξη ενός ολοκληρωμένου τρισδιάστατου παιχνιδιού για υπολογιστές με χρήση της μηχανής παιχνιδιών Unreal Development Kit (UDK). Το παιχνίδι ανήκει στην κατηγορία παιχνιδιών που είναι γνωστά ως πρώτου-τρίτου προσώπου (first-third person shooters). Συνοπτικά, το σενάριο του παιχνιδιού κάνει λόγο για έναν τρελό επιστήμονα που δημιουργεί τερατάκια στο εργαστήριο του με σκοπό να επανδρώσει το στρατό του. Τα τερατάκια απελευθερώνονται λόγω κάποιας βλάβης στο σύστημα της φυλακής και αρχίζουν να σκοτώνουν όλους τους αντιπάλους τους, συμπεριλαμβανομένου και του επιστήμονα. Έτσι, ο τελευταίος αναγκάζεται να τα εξοντώσει με τα ειδικά όπλα του. Ο παίκτης διαχειρίζεται τον χαρακτήρα του επιστήμονα και έχει σκοπό να συγκεντρώσει όσο περισσότερους πόντους μπορεί συλλέγοντας νομίσματα, ενώ ταυτόχρονα αντιμετωπίζει τους αντιπάλους του.

Η υλοποίηση ενός σύγχρονου παιχνιδιού τρισδιάστατων γραφικών είναι μια επίπονη και απαιτητική εργασία, για την οποία σε επαγγελματικό επίπεδο εργάζονται εκατοντάδες άτομα για να την φέρουν εις πέρας. Οι μεγάλες εταιρίες του χώρου διαθέτουν ανθρώπους με διάφορες γνώσεις, όπως προγραμματιστές, level designers, animators και πολλούς άλλους, οι οποίοι συνθέτουν τα επιμέρους τμήματα ενός ολοκληρωμένου παιχνιδιού.

Για την ολοκλήρωση του παιχνιδιού υλοποιήσαμε τη γραφική διεπαφή χρήστη, τον τρισδιάστατο χώρο, την κίνηση του παίκτη, τα όπλα που χρησιμοποιεί, την συμπεριφορά των αντιπάλων βασισμένη σε τεχνητή νοημοσύνη, τα βοηθητικά αντικείμενα και τις δυνατότητες που δίνουν. Αρχικά, επικεντρωθήκαμε στην εκμάθηση της πλατφόρμας και των εργαλείων. Κατόπιν, ορίσαμε την κάμερα και τον χαρακτήρα, προσθέσαμε την δυνατότητα συλλογής νομισμάτων και μια αρχικά απλή γραφική διεπαφή. Στη συνέχεια προσθέσαμε αντιπάλους χωρίς ιδιαίτερη νοημοσύνη και ξεκινήσαμε την δημιουργία του εικονικού κόσμου, στον οποίο λαμβάνει χώρα το παιχνίδι. Προσθέσαμε ειδικές δυνάμεις στον χαρακτήρα, όπως την δυνατότητα να γίνεται αόρατος, ανίκητος και να τρέχει γρηγορότερα για ορισμένο χρόνο και δημιουργήσαμε κατάλληλα όπλα για να αντιμετωπίζει τους εχθρούς. Επίσης, προσθέσαμε και ένα σύστημα αποθήκευσης αντικειμένων (inventory system) για τα αντικείμενα που διαθέτει ο χαρακτήρας και δεν έχει χρησιμοποιήσει ακόμα. Υλοποιήσαμε έναν αλγόριθμο τεχνητής νοημοσύνης για τους αντιπάλους, έτσι ώστε να κινούνται μέσα στον εικονικό κόσμο με τον βέλτιστο τρόπο και δώσαμε στον χαρακτήρα την δυνατότητα να χρησιμοποιεί αυτό τον αλγόριθμο για να κινηθεί προς ένα σταθερό σημείο. Τέλος, τελειοποιήσαμε την γραφική διεπαφή, προσθέσαμε τις κινήσεις των παικτών (animations) και βελτιώσαμε τον αλγόριθμο τεχνητής νοημοσύνης χρησιμοποιώντας δυαδικό σωρό (binary heap) για ακόμα μεγαλύτερη ταχύτητα.

Η εφαρμογή αξιολογήθηκε από χρήστες με τη χρήση ερωτηματολογίου (Questionnaire for User Interaction Satisfaction - QUIS), καθώς και μέσα από γραπτές παρατηρήσεις με σκοπό την διόρθωση ορισμένων ατελειών και την μεγαλύτερη ευχαρίστηση του παίκτη.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω ιδιαίτερα την επιβλέπουσα καθηγήτρια μου κ. Αικ. Μανιά για την πολύτιμη βοήθεια και καθοδήγηση που μου παρείχε καθ' όλη την διάρκεια ανάπτυξης και υλοποίησης της διπλωματικής μου εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τους καθηγητές κ. Μ. Λαγουδάκη και κ. Β. Σαμολαδά τόσο για την βοήθεια όσο και για τον χρόνο που θα αφιερώσουν στην ανάγνωση της διπλωματικής μου εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την στήριξη και την αγάπη τους, καθώς και όλους τους ανθρώπους που συνέβαλαν με οποιοδήποτε τρόπο όλα αυτά τα χρόνια.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ

1.1 Εισαγωγή

Στόχος της διπλωματικής είναι ο σχεδιασμός και η υλοποίηση ενός τρισδιάστατου παιχνιδιού για υπολογιστές με την βοήθεια της μηχανής παιχνιδιών Unreal Development Kit (UDK) [1]. Ο τίτλος της εργασίας "Ανάπτυξη τρισδιάστατου παιχνιδιού με βελτιστοποιημένο σχεδιασμό διαδρομών" αναφέρεται στην ενσωμάτωση στο τρισδιάστατο παιχνίδι ενός αλγορίθμου τεχνητής νοημοσύνης που έχει στόχο την εύρεση των βέλτιστων διαδρομών μεταξύ δύο προκαθορισμένων σημείων. Το παιχνίδι που επιλέξαμε να υλοποιήσουμε ανήκει στην κατηγορία "βολών πρώτου προσώπου" (First-Person-Shooter, FPS) [2], όπου έχουμε ενσωματώσει την δυνατότητα στο χρήστη να το μετατρέψει σε παιχνίδι κατηγορίας "βολών τρίτου προσώπου" (Third-Person-Shooter, TPS) [3]. Και οι δύο κατηγορίες πραγματεύονται μάχες με τη χρήση κάποιου όπλου. Στα παιχνίδια βολών πρώτου προσώπου ο χρήστης βλέπει την δράση μέσα από τα μάτια του χαρακτήρα που διαχειρίζεται. Αντίθετα στα παιχνίδια βολών τρίτου προσώπου ο χρήστης βλέπει την δράση από ένα πιο ψηλό σημείο, που μπορεί να βρίσκεται πίσω ή πλάγια από τον χαρακτήρα, επιτρέποντας στον χρήστη να έχει μια πιο γενική άποψη του χώρου, αλλά και της θέσης του χαρακτήρα.

1.2 Περιγραφή παιχνιδιού

Στα πλαίσια της διπλωματικής δημιουργήσαμε το σενάριο για το παιχνίδι "Mad World". Το σενάριο κάνει λόγο για έναν τρελό επιστήμονα που ονομάζεται Johann, ο οποίος επιθυμεί να κατακτήσει την χώρα και για να το καταφέρει αυτό, επανδρώνει τον στρατό του με τα τερατάκια που δημιουργεί. Τα τερατάκια απελευθερώνονται και αρχίζουν να εξοντώνουν όποιον δουν στο οπτικό τους πεδίο, ακόμα και τον ίδιο. Συνεπώς, ο Johann αναγκάζεται να τα εξοντώσει με τα μαγικά όπλα του για να σωθεί.

Ο χρήστης ελέγχει τον χαρακτήρα του Johann, αρχικά χωρίς κάποιο όπλο και πρέπει να τον διατηρήσει σώο, μαζεύοντας όσο περισσότερους βαθμούς γίνεται. Οι τρόποι για να μαζέψει βαθμούς είναι τρεις. Ο πρώτος και βασικότερος είναι μαζεύοντας τα νομίσματα που είναι διάσπαρτα στην πίστα. Ο δεύτερος τρόπος για να συλλέξει περισσότερους βαθμούς είναι να σκοτώσει τους εχθρούς του και ο τρίτος είναι να τερματίσει το παιχνίδι σε όσο το δυνατόν λιγότερο χρόνο μπορεί. Τερματίζοντας το σε λιγότερο χρόνο από τον προκαθορισμένο επιβραβεύεται με επιπλέον βαθμούς. Οι βαθμοί του προστίθενται και αν η βαθμολογία του είναι μεγάλη, καταγράφεται στον πίνακα με τις υψηλότερες βαθμολογίες.

Η δράση του παιχνιδιού εξελίσσεται σε ένα τρισδιάστατο χώρο (πίστα), τον οποίο δημιουργήσαμε έτσι ώστε να καλύπτει τις ανάγκες του σεναρίου του παιχνιδιού, αλλά και να προκαλεί το ενδιαφέρον του χρήστη. Μέσα στην πίστα έχουμε εισάγει διάφορα αντικείμενα τα οποία είναι συλλέξιμα. Εκτός από τα νομίσματα που προαναφέραμε, υπάρχουν τα όπλα και διάφορα φίλτρα, τα οποία δίνουν κάποιες δυνατότητες στον χαρακτήρα και θα αναλυθούν εκτενέστερα στην συνέχεια.

Οι αντίπαλοι του χρήστη χωρίζονται σε 2 κατηγορίες με διαφορετική συμπεριφορά. Στην πρώτη κατηγορία ανήκουν οι αντίπαλοι που προστατεύουν ένα συγκεκριμένο σημείο του χώρου, δεν κινούνται και περιμένουν να εμφανιστεί στην κοντινή τους περιοχή ο χαρακτήρας, ώστε να τον κυνηγήσουν και να τον εξοντώσουν. Στην περίπτωση που ο χαρακτήρας κρυφτεί, οι συγκεκριμένοι αντίπαλοι είναι προγραμματισμένοι να επιστρέφουν στο αρχικό σημείο που προστατεύουν και να περιμένουν εκ νέου. Οι αντίπαλοι της δεύτερης κατηγορίας σε αντίθεση με τους πρώτους, δεν περιμένουν σε ένα σημείο, αλλά περιφέρονται σε μια προκαθορισμένη περιοχή. Για να επιτευχθεί αυτό, στον τρισδιάστατο χώρο έχουν εισαχθεί και ομαδοποιηθεί κόμβοι (θα αναλυθεί αναλυτικά σε επόμενο κεφάλαιο). Σε κάθε αντίπαλο αυτής της κατηγορίας έχει ανατεθεί μια περιοχή μέσω μιας ομάδας κόμβων. Από την στιγμή που ο αντίπαλος αναγνωρίσει τον χαρακτήρα, η συμπεριφορά του είναι αντίστοιχη με τους αντιπάλους της κατηγορίας 1. Αν χαθεί από το ορατό τους πεδίο ο χαρακτήρας επιστρέφουν σε κατάσταση περιήγησης στην περιοχή που έχει καθοριστεί.

Ο αλγόριθμος τεχνητής νοημοσύνης, εκτός από τη χρήση που γίνεται στις μετακινήσεις των αντιπάλων, χρησιμοποιείται και για τον χαρακτήρα του χρήστη. Μέσα στα πλαίσια του παιχνιδιού, ο χρήστης μπορεί να επιλέξει την ημι-αυτόνομη (semi-autonomous) μετακίνηση του χαρακτήρα στην πίστα για μια προκαθορισμένη διαδρομή. Με αυτόν τον τρόπο επιτυγχάνεται η συγχώνευση του αλγορίθμου στην συμπεριφορά του χαρακτήρα που διαχειρίζεται ο χρήστης.

Για τον τερματισμό του παιχνιδιού ο χαρακτήρας πρέπει να συλλέξει όλα τα νομίσματα και να μεταβεί σε ένα συγκεκριμένο σημείο της πίστας, ενώ πρέπει να αποφύγει τους αντιπάλους και να βρει τρόπους να υπερνικήσει τα εμπόδια. Για παράδειγμα, θα πρέπει να αναζητήσει τον τρόπο να περάσει την λίμνη, στην οποία αν πέσει μέσα θα σκοτωθεί, ξεκινώντας το παιχνίδι από την αρχή.

1.3 Δομή της εργασίας

Στο δεύτερο κεφάλαιο παρουσιάζουμε το θεωρητικό υπόβαθρο της εργασίας. Δίνουμε επιγραμματικά την ιστορία των βιντεοπαιχνιδιών και τις γενικές κατηγορίες τους. Στη συνέχεια περιγράφουμε τις μηχανές παιχνιδιών που μας απασχόλησαν στην φάση της υλοποίησης της εφαρμογής και τέλος αναλύουμε τις αρχές του αλγορίθμου τεχνητής νοημοσύνης που χρησιμοποιήσαμε. Για την βελτίωση του αλγορίθμου προσθέσαμε δυαδική ουρά, που προκαλεί ταχύτερη εκτέλεση του.

Στο τρίτο κεφάλαιο περιγράφουμε την τεχνολογική βάση της εργασίας. Αναλύουμε με μεγαλύτερη λεπτομέρεια την μηχανή παιχνιδιών που χρησιμοποιήσαμε και τα επί μέρους βασικά κομμάτια που την απαρτίζουν, καθώς και τα εργαλεία που την συνοδεύουν. Ιδιαίτερη μνεία κάνουμε στην γλώσσα προγραμματισμού, στην οποία υλοποιήσαμε την εφαρμογή και στα χαρακτηρίστηκα της.

Στο τέταρτο κεφάλαιο αναλύουμε το σενάριο του παιχνιδιού, στο οποίο στηρίξαμε την υλοποίηση που κάναμε. Στη συνέχεια, χρησιμοποιώντας το σενάριο περιγράφουμε τον τρόπο που σχεδιάσαμε το παιχνίδι (τον τρισδιάστατο χώρο, τον χαρακτήρα του παίκτη, τους αντιπάλους, τα αντικείμενα, κτλ.) και τέλος παρουσιάζουμε τη γραφική διεπαφή του χρήστη. Σε αυτή συμπεριλαμβάνουμε εκτός από τα διάφορα

μενού και την κύρια διεπαφή, που εμφανίζεται κατά την πλοήγηση του παίκτη στον χώρο.

Στο πέμπτο κεφάλαιο δείχνουμε τον τρόπο που υλοποιήσαμε τα βασικότερα μέρη του παιχνιδιού. Ξεκινώντας από τις κάμερες, συνεχίσαμε στον τρόπο που υλοποιήσαμε την συμπεριφορά του χαρακτήρα (ταχύτερο τρέξιμο, σύστημα αποθήκευσης αντικειμένων, όπλα) και των αντιπάλων. Στη συνέχεια, αναφερόμαστε στον τρόπο υλοποίησης της γραφικής διεπαφής και του συστήματος της βαθμολογίας και τέλος στα σημαντικότερα κομμάτια του αλγορίθμου A* και του δυαδικού σωρού που χρησιμοποιήσαμε.

Στο έκτο κεφάλαιο παρουσιάζουμε τα αποτελέσματα δοκιμών του παιχνιδιού, τα οποία βοήθησαν στην εξαγωγή χρήσιμων συμπερασμάτων σχετικά με την ευχρηστία του καθώς επίσης και στην διόρθωση ορισμένων ατελειών. Παραθέτουμε το ερωτηματολόγιο που χρησιμοποιήθηκε, που δημιουργήθηκε από το πανεπιστήμιο του Maryland.

Στο έβδομο και τελευταίο κεφάλαιο γίνεται μια ανακεφαλαίωση της διπλωματικής διατριβής, αναφέρουμε τα συμπεράσματα της εργασίας και παρουσιάζονται μελλοντικές επεκτάσεις.

ΚΕΦΑΛΑΙΟ 2
ΣΧΕΤΙΚΗ ΕΡΕΥΝΑ

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα κάνουμε μια σύντομη αναδρομή στην ιστορία των βιντεοπαιχνιδιών, καθώς και στα είδη στα οποία αυτά διαχωρίζονται. Στη συνέχεια, θα περιγράψουμε ορισμένες μηχανές παιχνιδιών που μας απασχόλησαν στην έναρξη της παρούσης εργασίας και τέλος θα αναλύσουμε θεωρητικά τον αλγόριθμο τεχνητής νοημοσύνης A* και τους δυαδικούς σωρούς που χρησιμοποιήσαμε για την βελτίωση του αλγορίθμου.

2.2 Ιστορία βιντεοπαιχνιδιών

Η προέλευση των βιντεοπαιχνιδιών βρίσκεται στα αμυντικά συστήματα του στρατού, που ήταν βασισμένα σε σωλήνες καθοδικών ακτίνων γύρω στο 1940. Με το πέρασμα του χρόνου, δημιουργήθηκαν όλο και περισσότερα παιχνίδια, πιο εξελιγμένα και με αυξημένη πολυπλοκότητα Το πρώτο βιώσιμο εμπορικά παιχνίδι ήταν το “Computer Space” του 1971, που έθεσε τις βάσεις για την νέα βιομηχανία ψυχαγωγίας στα τέλη της δεκαετίας του 1970 στις Ηνωμένες Πολιτείες, την Ιαπωνία και την Ευρώπη. Η χρυσή εποχή των βιντεοπαιχνιδιών arcade έφθασε στο αποκορύφωμά της τη δεκαετία του 1980. Δημιουργήθηκαν παιχνίδια που έκαναν μεγάλη αίσθηση στο κοινό, όπως το Pac-Man (1980), Donkey Kong (1981), Mario Bros. (1983), Prince of Persia (1989), κ.α., πολλά από τα οποία συνεχίζονται ακόμα και σήμερα. Μετά την επιτυχία των “Apple II” και “Commodore PET”, εμφανίστηκαν νέες πλατφόρμες, όπως “Atari ST”, “Amstrad CPC”, “Nintendo”, κ.α. και η αγορά των βιντεοπαιχνιδιών χωρίστηκε σε δύο μέρη. Η βόρεια Αμερική και η Ευρώπη διατήρησαν το μερίδιο των παιχνιδιών σε υπολογιστικά

Εικόνα 2.1: Computer Space, 1971

Εικόνα 2.2: Amstrad CPC464, 1984

συστήματα (PC games), ενώ η Ιαπωνία έγινε κυρίαρχος των παιχνιδιών σε κονσόλες (Console games). Σύντομα, τα παιχνίδια από δυσδιάστατα μετατράπηκαν σε τριών διαστάσεων και χωρίστηκαν σε μεγάλες κατηγορίες ανάλογα με το περιεχόμενο τους (πρώτου προσώπου, τρίτου προσώπου, στρατηγικής, κτλ.). [4]

Σήμερα, τα τρισδιάστατα παιχνίδια γραφικών έχουν να επιδείξουν τεράστιες δυνατότητες κυρίως λόγω της ισχύος που είναι διαθέσιμη από τις κονσόλες έβδομης γενιάς και τα νέα υπολογιστικά συστήματα. Η επεξεργαστική ισχύς έχει αυξηθεί κατά δισεκατομμύρια φορές από τα πρώτα χρόνια εμφάνισης των παιχνιδιών, όπως και η χωρητικότητες της μνήμης και των δίσκων. Οι κατασκευάστριες εταιρίες παιχνιδιών χρησιμοποιούν τα πλεονεκτήματα αυτά και δημιουργούν τίτλους με περισσότερα σύγχρονα στοιχεία.

Εικόνα 2.3: Sony Playstation 3

Εικόνα 2.4: Nintendo Wii

Η τεχνολογία παρέχει κ άλλες δυνατότητες πέρα από την απεικόνιση υψηλής ποιότητας εικόνας και ήχου. Τα σύγχρονα συστήματα διαθέτουν σύνδεση δικτύου, έτσι ώστε ο χρήστης εκτός από την πλοήγηση στο διαδίκτυο, να μπορεί να επικοινωνεί ταυτόχρονα με άλλους παίκτες, λαμβάνοντας μέρος σε παιχνίδια πολλών παικτών (στη συνέχεια, θα αναφερθούμε σε αυτή την κατηγορία παιχνιδιών). Επιπρόσθετα, μπορεί να χρησιμοποιήσει νέους τρόπους επικοινωνίας με το σύστημα, όπως τα νέου τύπου χειριστήρια. Με αυτά, επιτυγχάνεται πιο άμεσος έλεγχος του παιχνιδιού μέσω φυσικών κινήσεων, που μεγεθύνουν την αίσθηση συμμετοχής στο παιχνίδι.

Ενώ τα παλαιότερα παιχνίδια ήταν δημιούργημα ενός ή λίγων ανθρώπων, τη σύγχρονη εποχή κυρίαρχες στο χώρο είναι μεγάλες εταιρίες πολλών ατόμων, που χωρίζονται σε ομάδες και ασχολούνται στα επί μέρους κομμάτια ενός παιχνιδιού. Ο λόγος που συμβαίνει αυτό είναι ότι δίνεται η δυνατότητα για μεγάλη εξειδίκευση στον σχεδιασμό και στην υλοποίηση, πράγμα που συνεπάγεται καλύτερη ποιότητα στο τελικό αποτέλεσμα. Πλέον, είναι δυνατό βλέποντας μια σκηνή ενός παιχνιδιού να γίνει σύγχυση μεταξύ πραγματικού και φανταστικού.

Τα τελευταία χρόνια παρατηρείται η αύξηση των χρηστών που παίζουν κάποιο είδος παιχνιδιού διαδικτυακά. Αυτά τα παιχνίδια είναι ενσωματωμένα σε σελίδες κοινωνικής δικτύωσης και αριθμούν πάνω από 500 εκατομμύρια χρηστών.

2.3 Είδη βιντεοπαιχνιδιών

Τα είδη (genre) των βιντεοπαιχνιδιών χρησιμοποιούνται για την κατηγοριοποίηση των παιχνιδιών, χρησιμοποιώντας ως βάση την αλληλεπίδραση με τον χρήστη (gameplay) και όχι τις οπτικές διαφορές ή τις διαφορές στην αφήγηση. Σε αντίθεση με άλλα έργα μυθοπλασίας, όπως οι ταινίες ή τα βιβλία, το περιεχόμενο και το περιβάλλον δεν χαρακτηρίζουν το παιχνίδι. Για παράδειγμα, ένα παιχνίδι δράσης εξακολουθεί να είναι ένα παιχνίδι δράσης, ανεξάρτητα από το αν λαμβάνει χώρα σε έναν φανταστικό κόσμο ή στο διάστημα. Δυστυχώς, στην κατηγοριοποίηση των παιχνιδιών υπάρχει έλλειψη συναίνεσης όσων αφορά τον επίσημο αποδεκτό χαρακτηρισμό ενός παιχνιδιού. Όπως σε κάθε είδους ταξινόμηση, έτσι και στην ταξινόμηση των παιχνιδιών είναι αναγκαίο να χρησιμοποιηθούν ορισμένες σταθερές για να εξαχθούν συμπεράσματα. Ένας τρόπος για να ομαδοποιηθούν τα παιχνίδια είναι η παρατήρηση του τρόπου ολοκλήρωσης των εμποδίων. Τα παιχνίδια που ο τρόπος ολοκλήρωσης είναι παρόμοιος δημιουργούν ένα κοινό είδος (genre).

Στη συνέχεια θα παραθέσουμε συγκεντρωτικά τις βασικές και δευτερεύουσες κατηγορίες βιντεοπαιχνιδιών. Ένα σημαντικό στοιχείο που πρέπει να σημειώσουμε είναι ότι οι ταξινομήσεις γίνονται παρωχημένες σε πολύ μικρό χρονικό διάστημα ανάλογα με την εξέλιξη της τεχνολογίας, των ιδεών και του σχεδιασμού των παιχνιδιών. Επιπρόσθετα, ορισμένες φορές τα σύνορα μεταξύ των κατηγοριών για κάποια παιχνίδια είναι δυσδιάκριτα. Αυτό συμβαίνει διότι κάποια βιντεοπαιχνίδια διαθέτουν χαρακτηριστικά που επιτρέπει να τα συμπεριλάβουμε σε παραπάνω από μια κατηγορίες ταυτόχρονα. Συνεπώς, σε αυτή την ενότητα θα κάνουμε μια περιγραφή των κύριων κατηγοριών με σκοπό να δημιουργηθεί μια πρώτη εικόνα για το πλήθος των κατηγοριών και των βασικών χαρακτηριστικών που διαθέτουν τα σύγχρονα παιχνίδια.

1 Action game
- 1.1 Beat 'em up
- 1.2 Fighting game
- 1.3 Platform game
- 1.4 Shooter game
  - 1.4.1 First-person shooter
  - 1.4.2 Third-person shooter
  - 1.4.3 Shoot 'em up
  - 1.4.4 Light gun shooter
  - 1.4.5 Rail shooter
  - 1.4.6 Tactical shooter

2 Adventure game
- 2.1 Graphic adventure game
- 2.2 Interactive fiction
- 2.3 Interactive movie
- 2.4 Visual novel

3 Action-adventure game
- 3.1 Stealth game
- 3.2 Survival horror game

4 Role-playing video game
5 Music and rhythm game
6 Simulation game
- 6.1 Career simulation
- 6.2 Life simulation
- 6.3 Vehicle simulation

7 Sports game
- 7.1 Sports management game

8 Racing game
9 Strategy game
- 9.1 Turn-based Strategy Game
- 9.2 Real-time strategy game

10 Party game
11 Puzzle game
12 Traditional game
13 Educational game
14 Adult game
15 Massively multiplayer online game

Εικόνα 2.5: Είδη βιντεοπαιχνιδιών [5]

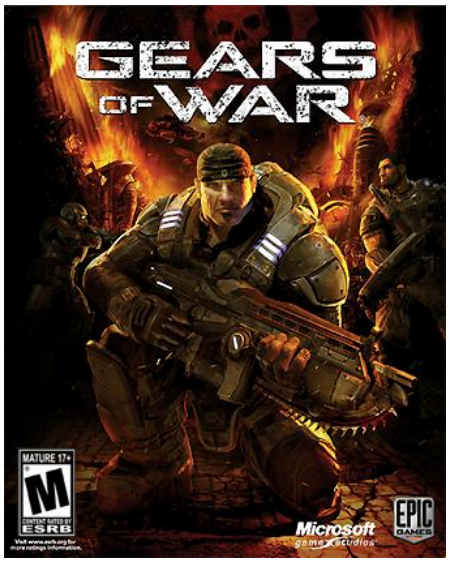
2.3.1 Δράσης (Action)

Τα παιχνίδια δράσης είναι μια μεγάλη κατηγορία των βιντεοπαιχνιδιών και διαχωρίζεται σε αρκετές υπό-κατηγορίες. Πρόκειται για παιχνίδια που δίνουν έμφαση στις σωματικές προκλήσεις για τον χρήστη, συμπεριλαμβανομένου του συντονισμού χεριού-ματιών καθώς και στην επίδραση του χρόνου. Λύση γρίφων ή εξερευνητικά στοιχεία δεν περιλαμβάνονται σε αυτή την κατηγορία.

Ο χρήστης συνήθως ελέγχει κάποιο χαρακτήρα, περιηγείται σε κάποιο-α επίπεδα, συλλέγει αντικείμενα, αποφεύγει εμπόδια και μάχεται με τους εχθρούς. Στο τέλος κάποιου επιπέδου ο παίκτης αντιμετωπίζει κάποιον αντίπαλο μεγαλύτερο και πιο δύσκολο από τους κοινούς αντιπάλους. Οι εχθροί και τα εμπόδια καταστρέφουν την υγεία και τις ζωές του χαρακτήρα και το παιχνίδι τελειώνει όταν οι ζωές του μηδενιστούν. Εναλλακτικά, ο παίκτης κερδίζει τερματίζοντας τα διάφορα επίπεδα του παιχνιδιού.

Όπως αναφέραμε, στην κατηγορία δράσης ανήκουν αρκετές υπό-κατηγορίες. Βασικότερες είναι οι κατηγορίες αγώνων (fighting), πλατφόρμας (platform) και βολών. Η τελευταία διαχωρίζεται σε περισσότερες υπό-κατηγορίες, με βασικότερες τις πρώτου και τρίτου προσώπου (first/third person shooters). Το παιχνίδι “Mad World” που έχουμε υλοποιήσει στα πλαίσια της διπλωματικής εργασίας είναι μια μίξη παιχνιδιού βολών πρώτου και τρίτου προσώπου.

Εικόνα 2.6: Super Mario Galaxy 2
Είδος: Platform

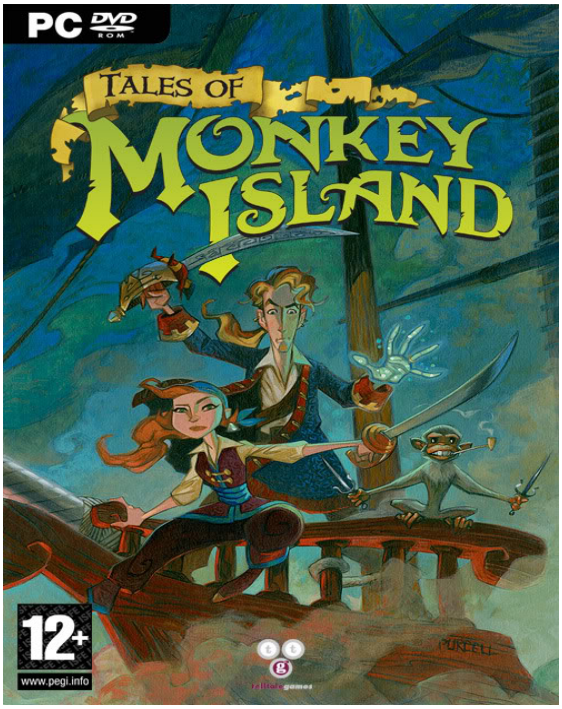


Εικόνα 2.7: Gears of war
Είδος: Third person shooter

2.3.2 Περιπέτειας (Adventure)

Ένα παιχνίδι περιπέτειας είναι ένα παιχνίδι στο οποίο ο παίκτης αναλαμβάνει το ρόλο του πρωταγωνιστή σε μια δια-δραστική ιστορία, που προχωρά μέσα από την εξερεύνηση και την επίλυση γρίφων. Η εστίαση του είδους στην ιστορία επιτρέπει την

άντληση ιδεών από άλλα αφηγηματικά είδη και μέσα, όπως η λογοτεχνία και ο κινηματογράφος. Σχεδόν όλα τα παιχνίδια περιπέτειας είναι σχεδιασμένα για έναν παίκτη, αφού η έμφαση στην ιστορία και τον χαρακτήρα καθιστά την σχεδίαση για πολλούς παίκτες πρακτικά αδύνατη. Τα παιχνίδια περιπέτειας με τη σειρά τους χωρίζονται σε υπό-κατηγορίες, όπως παιχνίδια δια-δραστικής ταινίας (interactive movie), δια-δραστικής μυθοπλασίας (interactive fiction), οπτικού μυθιστορήματος (visual novel) και γραφικής περιπέτειας (graphic adventure).



Εικόνα 2.8: Tales of monkey island
Είδος: Graphic adventure

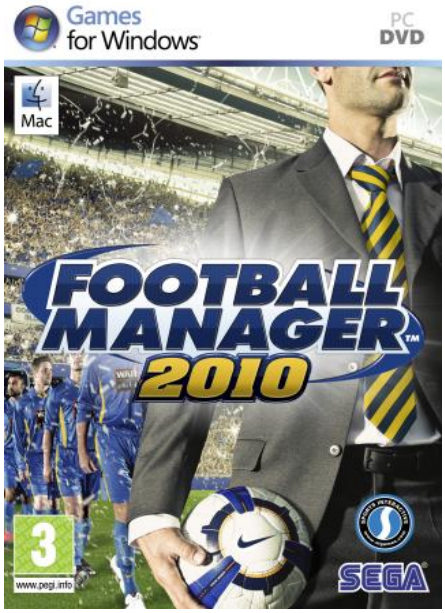
2.3.3 Ρόλου (Role-playing game – RPG)

Τα παιχνίδια ρόλου είναι ένα είδος που στηρίζεται στην αφήγηση μιας ιστορίας και στην ανάπτυξη του χαρακτήρα. Ο παίκτης ελέγχει κάποιον χαρακτήρα με ειδικές δεξιότητες ή ικανότητες. Οι μάχες γίνονται χρησιμοποιώντας εντολές, όπως «επίθεση» ή «χρησιμοποίησε το χ όπλο» και δεν υπάρχουν περίπλοκοι συνδυασμοί πλήκτρων επίθεσης, όπως στα παιχνίδια δράσης. Το βασικότερο χαρακτηριστικό αυτής της κατηγορίας είναι ότι καθώς ο παίκτης προχωρά επίπεδα στο παιχνίδι, ο χαρακτήρας αποκτά περισσότερες ικανότητες και εξελίσσεται ανάλογα με την ιστορία. Συχνά, η χρήση που κάνει ο παίκτης προκαλεί αλλαγές στον χαρακτήρα, προκαλώντας μεγάλη ποικιλία στις δυνατές καταστάσεις του παιχνιδιού. Τα παιχνίδια αυτής της κατηγορίας μπορούν να παιχτούν από έναν παίκτη, όμως πολύ συχνά γίνεται χρήση του διαδικτύου, κάνοντας δυνατή την ταυτόχρονη σύνδεση εκατοντάδων ή και χιλιάδων παικτών.

2.3.4 Προσομοίωσης (Simulation)

Τα παιχνίδια προσομοίωσης είναι σχεδιασμένα να προσομοιώνουν κάποιο συγκεκριμένο γεγονός με διάφορους βαθμούς ρεαλισμού. Το αντικείμενο της προσομοίωσης μπορεί να είναι ένα κάποιο αληθινό γεγονός ή κάποια ιδέα επιστημονικής φαντασίας. Μεγάλη επιτυχία έχουν δημιουργήσει τα παιχνίδια που προσομοιώνουν την καθημερινότητα, όπως το "The Sims", αλλά και τα παιχνίδια που προσομοιώνουν αθλητικές δραστηριότητες, όπως το "Football manager". Επιπρόσθετα, υπάρχουν και άλλες υπό-κατηγορίες, όπως προσομοιώσεις οχημάτων (αυτοκίνητα, φορτηγά, αεροπλάνα), κατασκευές κτηρίων, προσομοιώσεις καριέρας, κτλ.

Εικόνα 2.9: The sims 3
Είδος: Life simulation

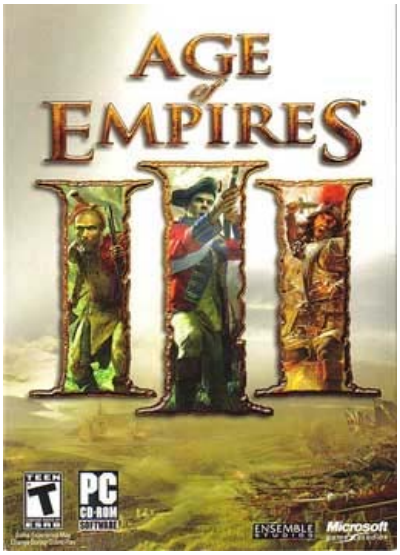


Εικόνα 2.10: Football manager 2010
Είδος: Sport simulation

2.3.5 Στρατηγικής (Strategy)

Τα παιχνίδια στρατηγικής δίνουν έμφαση στον επιδέξιο τρόπο σκέψης και στον σχεδιασμό για να επιτευχθεί η νίκη. Παρέχουν στρατηγικές, τακτικές και ορισμένες φορές υλικοτεχνικές προκλήσεις. Πολλές φορές προσφέρουν οικονομικές προκλήσεις και εξερευνήσεις. Συνήθως, ο χρήστης καλείται να δημιουργήσει και να ελέγξει διάφορες μονάδες (λχ. στρατό) για να ολοκληρώσει το παιχνίδι. Γενικά, τα παιχνίδια στρατηγικής χωρίζονται σε τέσσερις τύπους ανάλογα με το αν είναι πραγματικού χρόνου (real time) ή εκ περιτροπής (turn based) και αν εστιάζουν στην τακτική ή στην στρατηγική. Τα παιχνίδια εκ περιτροπής είναι αυτά που ο χρήστης διαθέτει το χρόνο για να κάνει μια κίνηση και κατόπιν έρχεται η σειρά του επόμενου, ενώ τα παιχνίδια πραγματικού χρόνου είναι αυτά που οι παίκτες παίζουν ταυτόχρονα, χωρίς ο χρόνος να διαιρείται σε μέρη.

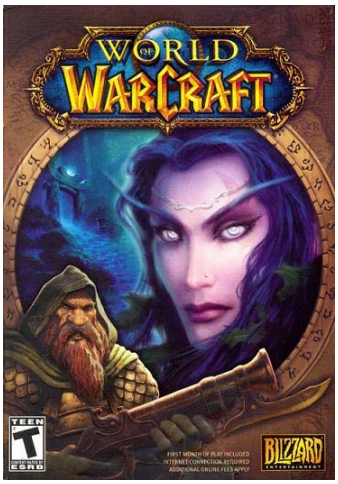
Εικόνα 2.11: Worms-A space oddity
Είδος: Turn-based strategy



Εικόνα 2.12: Age of empires 3
Είδος: Real time strategy

2.3.6 Πολυπληθές διαδικτυακό (MMO)

Ένα πολυπληθές διαδικτυακό παιχνίδι είναι ικανό να υποστηρίξει εκατοντάδες ή χιλιάδες παίκτες ταυτόχρονα. Αναγκαία, πρέπει να υπάρχει σύνδεση στο διαδίκτυο και ένας σταθερός τρισδιάστατος κόσμος. Τα παιχνίδια αυτού του τύπου επιτρέπουν στους παίκτες να συνεργαστούν και να ανταγωνιστούν μεταξύ τους σε μεγάλη κλίμακα και ορισμένες φορές να αλληλεπιδρούν ουσιαστικά, για να ολοκληρώσουν κάποια αποστολή. Σε αυτή την κατηγορία περιλαμβάνονται σχεδόν όλες οι υπόλοιπες κατηγορίες με επιπρόσθετο χαρακτηριστικό την διαδικτυακή επικοινωνία. Σήμερα, εκατομμύρια χρήστες ανά τον κόσμο συνδέονται μεταξύ τους με χαρακτηριστικό παράδειγμα τα 12 εκατομμύρια συνδρομητές του παιχνιδιού “World of Warcraft”.



Εικόνα 2.13: World of Warcraft
Είδος: MMO RPG

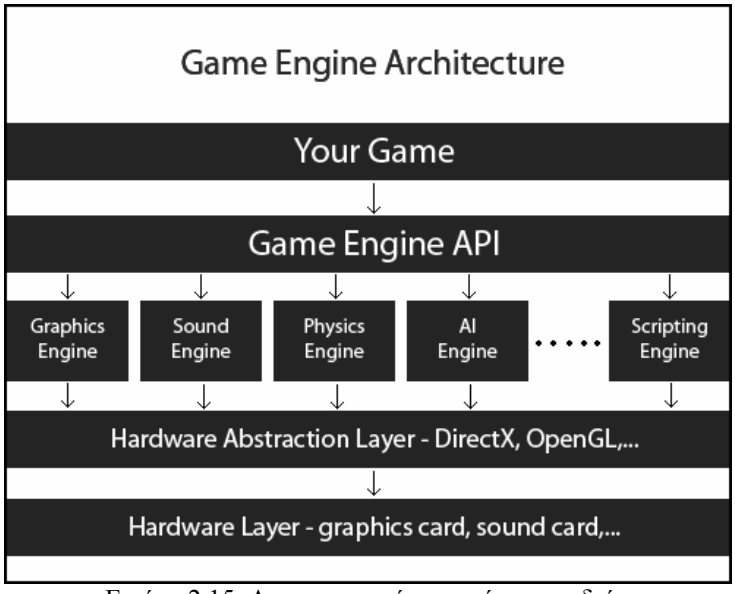
Εικόνα 2.14: NFS World
Είδος: MMO Racing

2.4 Μηχανές παιχνιδιών

Για την ανάπτυξη ενός παιχνιδιού τρισδιάστατων γραφικών είναι απαραίτητο το κατάλληλο λογισμικό. Στη σύγχρονη εποχή, οι εταιρίες που κατασκευάζουν παιχνίδια συνήθως πρώτα δημιουργούν το λογισμικό αυτό, το οποίο τους επιτρέπει την ευκολότερη σχεδίαση και υλοποίηση. Το λογισμικό αυτό δεν χρησιμοποιείται για την ανάπτυξη ενός παιχνιδιού μόνο, αλλά για μια σειρά από παιχνίδια της εταιρίας, προτού η εξέλιξη της τεχνολογίας το καταστήσει απαρχαιωμένο και δώσει τη θέση του σε μια νέα έκδοση.

Στην πλήρη μορφή το λογισμικό που δημιουργούν οι μεγάλες εταιρίες είναι μια μηχανή παιχνιδιών. Μια μηχανή παιχνιδιών προσφέρει την διασύνδεση (Application Programming Interface - API) που χρειάζεται ο προγραμματιστής για να επικοινωνήσει με το υλικό (hardware). Στο εσωτερικό της μηχανής υπάρχουν υλοποιημένα αυτόνομα συστήματα που διαχειρίζονται μια εξειδικευμένη διαδικασία, όπως τα συστήματα γραφικών, ήχου, φυσικής, τεχνητής νοημοσύνης, κτλ.

Το σύστημα γραφικών είναι επιβαρυμένο με την απεικόνιση των γραφικών στην οθόνη. Περιέχει τις βιβλιοθήκες και τις συναρτήσεις που καλούνται από την μηχανή παιχνιδιών για την αναπαράσταση όλων των οντοτήτων. Η μηχανή φυσικής αντίστοιχα, δημιουργεί τον ρεαλισμό στα παιχνίδια, εισάγοντας τους νόμους της φυσικής σε αυτά. Για την εξομοίωση χρησιμοποιείται η βαρύτητα, η ανίχνευση συγκρούσεων, αλλά και άλλα στοιχεία για να αποδοθεί το αποτέλεσμα. Ο προγραμματιστής μπορεί να αυξήσει ή να μειώσει τον βαθμό ρεαλισμού, ώστε να επιτύχει το επιθυμητό αποτέλεσμα σε σχέση με το είδος του παιχνιδιού που δημιουργεί. Όλες οι αλληλεπιδράσεις μεταξύ των οντοτήτων σε ένα παιχνίδι, η κίνηση των χαρακτήρων, η δυναμική των υγρών επεξεργάζονται από αυτό το σύστημα.



Εικόνα 2.15: Αρχιτεκτονική μηχανών παιχνιδιών

Το σύστημα τεχνητής νοημοσύνης είναι υπεύθυνο για την νοημοσύνη που παρουσιάζουν οι χαρακτήρες που δεν ελέγχονται από τον παίκτη. Χωρίς τις λειτουργίες

που παρέχει, το αποτέλεσμα των κινήσεων των αντιπάλων θα ήταν αφύσικο, ως και ενοχλητικό για τους παίκτες. Τέλος, μια πλήρης μηχανή παιχνιδιών διαθέτει και άλλα συστήματα, για τον ήχο, συγγραφή κώδικα (scripting), κτλ.

Στην παρούσα εργασία επιλέξαμε να εργασθούμε χρησιμοποιώντας μια μηχανή παιχνιδιών, καθώς παρέχουν ευελιξία, περιέχουν κάποιο εύχρηστο γραφικό περιβάλλον με εργαλεία για την ανάπτυξη του παιχνιδιού και οτιδήποτε άλλο χρειάζεται για να δημιουργηθεί ένα παιχνίδι γραφικών. Στη συνέχεια, θα παρουσιάσουμε τα βασικά στοιχεία από τις μηχανές παιχνιδιών που μας απασχόλησαν περισσότερο και την τελική επιλογή της μηχανής UDK.

2.4.1 Unity 3D

Η Unity 3D είναι ένα ολοκληρωμένο περιβάλλον δημιουργίας τρισδιάστατων παιχνιδιών ή άλλων εφαρμογών (π.χ. αρχιτεκτονική απεικόνιση, τρισδιάστατα σχέδια πραγματικού χρόνου, κτλ.) [6]. Παρέχει πληθώρα εργαλείων για την επεξεργασία της γεωμετρίας, των υφών, του φωτισμού και των ήχων, καθώς και την μηχανή φυσικής Ageia της εταιρίας nVidia. Στον τομέα των γραφικών η Unity έχει μια βελτιστοποιημένη διοχέτευση γραφικών για DirectX και OpenGL. Επίσης διαθέτει κινούμενα δικτυώματα, συστήματα σωματιδίων και προχωρημένο σύστημα για τον φωτισμό των χώρων και των σκιών.

Στον τομέα του προγραμματισμού, η Unity υποστηρίζει τρεις γλώσσες, την JavaScript, την C# και Boo, που είναι μια παραλλαγή της Python. Και οι τρεις γλώσσες είναι εξίσου γρήγορες και μπορούν να αλληλοσυνδεθούν. Η λογική του παιχνιδιού τρέχει στην ανοιχτού κώδικα πλατφόρμα "Mono", κάτι το οποίο δίνει πλήρη ταχύτητα και ευελιξία. Επίσης, για την ανάπτυξη του παιχνιδιού παρέχεται πρόγραμμα απασφαλμάτωσης (debugger), επιτρέποντας την παύση του παιχνιδιού και εκτέλεση του προγράμματος γραμμή-γραμμή.

Η Unity 3D είναι μια πάρα πολύ διαδεδομένη μηχανή παιχνιδιών, με μεγάλο κοινό και φόρουμ για βοήθεια. Παρέχεται δωρεάν για μη εμπορική χρήση και δίνει όλα τα εργαλεία για την ανάπτυξη ενός παιχνιδιού, όπως αυτό που δημιουργήσαμε στα πλαίσια της παρούσης διπλωματικής. Πρόκειται για μια εύκολη στην εκμάθηση μηχανή με πολύ καλά αποτελέσματα. Δίνει την δυνατότητα για δημιουργία παιχνιδιών που τρέχουν σε διάφορες πλατφόρμες (PC, Mac, Android, Xbox, PS3, iOS), αλλά και σε φυλλομετρητή διαδικτύου μέσω ειδικού plug-in.

2.4.2 Torque 3D

Η μηχανή παιχνιδιών Torque 3D είναι μια μηχανή με πλούσια χαρακτηριστικά, υψηλού επιπέδου διαδικτυακό σύστημα, ποιοτική μηχανή γραφικών, εργαλείο δημιουργίας γραφικών διεπαφών, συντάκτη τρισδιάστατου χώρου και μια γλώσσα προγραμματισμού παρόμοια με την C [7]. Η μηχανή δεν διατίθεται δωρεάν και το κόστος αυτής ανέρχεται από τα 100$ για απλή άδεια, χωρίς την παροχή του πηγαίου κώδικα, έως 1200$ για ομάδες ανάπτυξης εμπορικών παιχνιδιών, με την παροχή πλήρους υποστήριξης.

Ο προγραμματισμός των παιχνιδιών γίνεται με την γλώσσα “TorqueScript”, που μοιάζει με την C++. Τα πλεονεκτήματα της γλώσσας είναι οι γρήγορες μαθηματικές πράξεις, που χρειάζονται πάρα πολύ σε παιχνίδια γραφικών, αντικειμενοστραφής λογική και άρτια τεκμηρίωση.

Η μηχανή τρέχει σε πλατφόρμες “Windows” και “MacOS” και μπορεί να δημιουργήσει ακόμα και διαδικτυακές εφαρμογές. Διαθέτει όλα τα βασικά χαρακτηριστικά μιας σύγχρονης μηχανής παιχνιδιών, όπως σύστημα φωτισμού και σύστημα φυσικής, όμως το βασικό μειονέκτημα της μηχανής είναι το κόστος, δεδομένου του γεγονότος ότι για την εργασία αναζητήσαμε μηχανές χωρίς συνδρομή.

2.4.3 Unreal Engine

Πρόκειται για μια από τις καλύτερες, αν όχι η καλύτερη μηχανή παιχνιδιών αυτή την στιγμή. Στο τέλος του 2009 ξεκίνησε να διατίθεται δωρεάν για μη εμπορική χρήση, συνεπώς αποτέλεσε μια πάρα πολύ καλή λύση για τα πλαίσια αυτής της εργασίας. Η μηχανή αυτή χρησιμοποιείται από τα μεγαλύτερα studios του κόσμου, έχει χιλιάδες μέλη στην κοινότητα που παρέχουν βοήθεια και χαρακτηριστικά που δύσκολα βρίσκει κανείς σε άλλες μηχανές παιχνιδιών. Η έκδοση που παρέχεται δωρεάν δεν συμπεριλαμβάνει τον πηγαίο κώδικα, κάτι που όμως δεν είναι ιδιαίτερα περιοριστικό για εφαρμογές όπως η συγκεκριμένη.

Ο πυρήνας της μηχανής είναι γραμμένος στην γλώσσα C++, πράγμα που την κάνει πολύ γρήγορη, ενώ ο προγραμματιστής μπορεί να συγγράψει σε C++ ή σε Unrealscript, που δίνουν πολύ μεγάλες δυνατότητες. Περιέχει όλα τα υπό-συστήματα που είναι απαραίτητα και είναι αρκετά απαιτητική από το σύστημα, λόγω της υψηλής ποιότητας του αποτελέσματος. Στο επόμενο κεφάλαιο θα αναλύσουμε εκτενέστερα την μηχανή και τα χαρακτηριστικά της.

Η εκμάθηση της μηχανής είναι πάρα πολύ δύσκολη, όμως την επιλέξαμε για δύο βασικούς λόγους. Ο πρώτος είναι ότι διατίθεται δωρεάν και ο δεύτερος ότι θελήσαμε να διαχειριστούμε και να μάθουμε ένα επαγγελματικό εργαλείο, το οποίο ουσιαστικά δίνει ατελείωτες δυνατότητες στην ανάπτυξη εφαρμογών τριών διαστάσεων.

Εικόνα 2.16: Unreal Development Kit
Η δωρεάν έκδοση της μηχανής Unreal

2.5 Τεχνητή νοημοσύνη

Ο όρος τεχνητή νοημοσύνη αναφέρεται στον κλάδο της επιστήμης υπολογιστών, ο οποίος ασχολείται με τη σχεδίαση και την υλοποίηση υπολογιστικών συστημάτων που μιμούνται στοιχεία της ανθρώπινης συμπεριφοράς, τα οποία υπονοούν έστω και στοιχειώδη ευφυΐα: μάθηση, προσαρμοστικότητα, εξαγωγή συμπερασμάτων, κατανόηση από συμφραζόμενα, επίλυση προβλημάτων κλπ. Η τεχνητή νοημοσύνη αποτελεί σημείο τομής μεταξύ πολλών πεδίων όπως της επιστήμης υπολογιστών, της ψυχολογίας, της φιλοσοφίας, της νευρολογίας, της γλωσσολογίας και της επιστήμης μηχανικών, με στόχο τη σύνθεση ευφυούς συμπεριφοράς με στοιχεία συλλογιστικής, μάθησης και προσαρμογής στο περιβάλλον, ενώ συνήθως εφαρμόζεται σε μηχανές ή υπολογιστές ειδικής κατασκευής.

Η τεχνητή νοημοσύνη είναι παρούσα σχεδόν σε όλα τα ηλεκτρονικά παιχνίδια. Κάθε οντότητα που δεν διαχειρίζεται από κάποιον χρήστη πρέπει να ελέγχεται από κάποιου είδους τεχνητή νοημοσύνη. Κατά τη διάρκεια των ετών αναπτύχθηκαν διάφορες τεχνικές για την αποδοτικότερη και ανταγωνιστικότερη συμπεριφορά των αντιπάλων. Στο παιχνίδι που δημιουργήσαμε γίνεται χρήση τεχνητής νοημοσύνης στους αντιπάλους κατά τη διάρκεια των μετακινήσεων. Αυτό συμβαίνει γιατί σε έναν τρισδιάστατο χώρο υπάρχουν εμπόδια και επιθυμούμε οι αντίπαλοι να μεταβαίνουν από ένα σημείο σε ένα άλλο με τον βέλτιστο τρόπο, αποφεύγοντας τα εμπόδια αυτά. Στη συνέχεια, θα παρουσιάσουμε θεωρητικά τον αλγόριθμο και την δομή δεδομένων που χρησιμοποιήσαμε.

2.5.1 Ο αλγόριθμος A*

Η ευρύτερα γνωστή μορφή αναζήτησης πρώτα στο καλύτερο (Best First Search - BFS) ονομάζεται αναζήτηση A* [8]. Η αναζήτηση αυτή αξιολογεί τους κόμβους συνδυάζοντας το *g(n)*, το κόστος της μετάβασης στον κόμβο *n*, και το *h(n)*, το κόστος της μετάβασης από τον κόμβο *n* στον στόχο.

$$f(n) = g(n) + h(n).$$

Η *g(n)* μας δίνει το κόστος της διαδρομής από τον αρχικό κόμβο στον κόμβο *n* και *h(n)* είναι το εκτιμώμενο κόστος της φθηνότερης διαδρομής από τον κόμβο *n* στον στόχο. Συνεπώς έχουμε:

f(n) = εκτιμώμενο κόστος της φθηνότερης λύσης μέσω του κόμβου *n*

Επομένως, αν επιδιώκουμε να βρούμε τη φθηνότερη λύση, είναι λογικό να δοκιμάσουμε πρώτα τον κόμβο με τη μικρότερη τιμή *g(n) + h(n).*

Εφόσον η ευρετική συνάρτηση *h(n)* ικανοποιεί ορισμένες συνθήκες, η αναζήτηση A* είναι πλήρης και βέλτιστη. Η προϋπόθεση για να είναι πλήρης και βέλτιστη η αναζήτηση είναι η *h(n)* να είναι παραδεκτός ευρετικός μηχανισμός (admissible heuristic). Αυτό σημαίνει ότι η *h(n)* ποτέ δεν υπερεκτιμά το κόστος επίτευξης του στόχου και συνεπώς (αφού η *g(n)* είναι το ακριβές κόστος της μετάβασης στον κόμβο *n)* η *f(n)* δεν

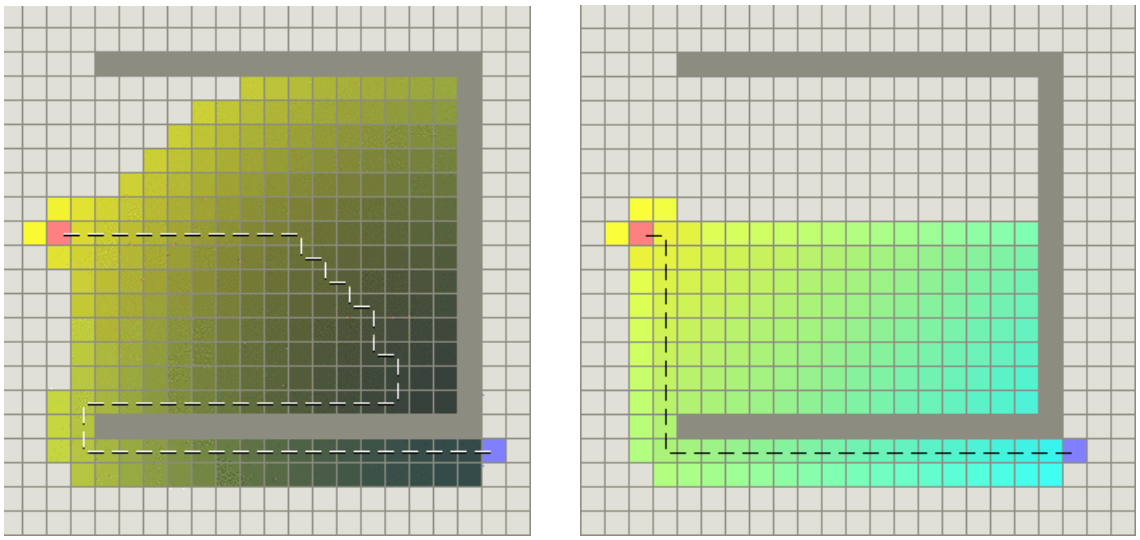
υπερεκτιμά ποτέ το κόστος μιας λύσης μέσω του *n*. Ο ευρετικός μηχανισμός που χρησιμοποιήσαμε στην υλοποίηση του αλγορίθμου είναι η ευκλείδεια απόσταση, που εγγυάται ότι δεν υπερεκτιμά το κόστος για τον στόχο, αφού η ευθεία γραμμή μεταξύ δυο σημείων είναι πάντα ο συντομότερος δρόμος.

Η χρονική πολυπλοκότητα του αλγορίθμου εξαρτάται από την ευρετική συνάρτηση. Στην χειρότερη περίπτωση ο αριθμός τον κόμβων που επεκτείνονται είναι εκθετικός, όμως είναι πολυωνυμικός, εάν ισχύουν τα εξής: ο χώρος αναζήτησης είναι ένα δένδρο, υπάρχει μόνο μια κατάσταση τερματισμού και ο ευρετική συνάρτηση ακολουθεί την συνθήκη:

$$| h(x) - h^{*}(x) | = O(\log h^{*}(x))$$

όπου h^{*} είναι η βέλτιστη ευρετική συνάρτηση, δηλαδή το ακριβές κόστος της μετακίνησης από το *x* στο στόχο.

Η λογική του αλγορίθμου είναι να διασχίσει τον γράφο ακολουθώντας το μονοπάτι με το μικρότερο κόστος, κρατώντας μια ταξινομημένη ουρά προτεραιότητας με τα εναλλακτικά μονοπάτια. Εάν σε κάποιο σημείο το κομμάτι του μονοπατιού στο οποίο βρίσκεται έχει μεγαλύτερο κόστος από κάποιο άλλο κομμάτι που συνάντησε, τότε εγκαταλείπει το παρόν κομμάτι και συνεχίζει να διασχίζει από το κομμάτι με το μικρότερο κόστος. Αυτό συνεχίζεται μέχρι να φτάσει στο στόχο του. Στην επόμενη εικόνα φαίνεται πόσο καλύτερο αποτέλεσμα δίνει ο αλγόριθμος A*, σε σχέση με τον αλγόριθμο αναζήτηση πρώτα στο καλύτερο (Best First Search):



Εικόνα 2.17: Αριστερά Best First Search, Δεξιά A* [9]

Στην αριστερή εικόνα φαίνεται ξεκάθαρα το πρόβλημα του αλγορίθμου BFS. Προσπαθεί να κινηθεί προς το στόχο, ακόμα και αν δεν είναι το σωστό μονοπάτι. Αφού δεν κρατάει πληροφορία για το κόστος του μονοπατιού ως το τρέχον σημείο, αλλά μόνο το κόστος ως το στόχο, το μονοπάτι μεγαλώνει ανεξέλεγκτα. Αντίθετα, δεξιά φαίνεται το πλεονέκτημα του αλγορίθμου A*.

Ο αλγόριθμος σε μορφή ψευδοκώδικα δίνεται στη συνέχεια [10]:

```
function A*(start,goal)
     closedset := the empty set
     openset := set containing the initial node
     came_from := the empty map
     g_score[start] := 0
     h_score[start] := heuristic_estimate_of_distance(start, goal)
     f_score[start] := h_score[start]

     while openset is not empty
         x := the node in openset having the lowest f_score[] value
         if x = goal
             return reconstruct_path(came_from, came_from[goal])
         remove x from openset
         add x to closedset
         foreach y in neighbor_nodes(x)
             if y in closedset
                 continue
             tentative_g_score := g_score[x] + dist_between(x,y)

             if y not in openset
                 add y to openset
                 tentative_is_better := true
             elseif tentative_g_score < g_score[y]
                 tentative_is_better := true
             else
                 tentative_is_better := false

             if tentative_is_better = true
                 came_from[y] := x
                 g_score[y] := tentative_g_score
                 h_score[y] := heuristic_estimate_of_distance(y, goal)
                 f_score[y] := g_score[y] + h_score[y]
     return failure

function reconstruct_path(came_from, current_node)
     if came_from[current_node] is set
         p = reconstruct_path(came_from, came_from[current_node])
         return (p + current_node)
     else
         return current_node
```

Ο αλγόριθμος A* χρησιμοποιεί δύο λίστες για να υπολογίσει το συντομότερο μονοπάτι. Η πρώτη λίστα (ανοιχτή λίστα) είναι αυτή που περιέχει το σύνολο των προσωρινών κόμβων που πρέπει να αξιολογηθούν. Η δεύτερη λίστα (κλειστή λίστα) είναι αυτή που περιέχει το σύνολο των κόμβων που ήδη αξιολογήθηκαν. Συνεπώς, ο αλγόριθμος δεν αξιολογεί τον ίδιο κόμβο περισσότερες από μια φορές. Στην επόμενη υπό-ενότητα θα αναλύσουμε τον δυαδικό σωρό που χρησιμοποιείται για την βελτίωση της ταχύτητας προσπέλασης του πρώτου στοιχείου στην ανοιχτή λίστα. Για την κλειστή λίστα χρησιμοποιούμε έναν πίνακα, τον οποίο προσπελαύνουμε με γραμμική αναζήτηση.

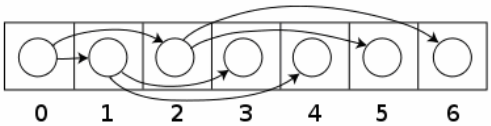
2.5.2 Δυαδικός σωρός (Binary heap)

Συνηθισμένη βελτίωση του A* είναι η χρησιμοποίηση μιας δομής δεδομένων για την ανοιχτή λίστα, αντί για απλό πίνακα, που να κάνει ταχύτερη την προσπέλαση του στοιχείου με το μικρότερο κόστος. Η δομή δεδομένων που χρησιμοποιείται είναι ο δυαδικός σωρός. Ο δυαδικός σωρός είναι ένα αντικείμενο που αποθηκεύεται σαν πίνακας, αλλά μπορούμε να το σκεφτούμε σαν ένα σχεδόν πλήρες δυαδικό δέντρο [11]. Όλα τα επίπεδα του δέντρου είναι γεμάτα, εκτός ίσως από τα τελευταία. Το πρώτο στοιχείο του δέντρου αποθηκεύεται στην πρώτη θέση του πίνακα και αν έχουμε το δείκτη της θέσης, μπορούμε να προσπελάσουμε τον πατέρα ή τα παιδιά ενός κόμβου, μέσω των συναρτήσεων:

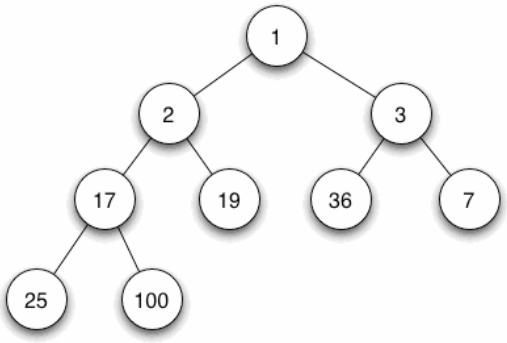
```
Parent(i)
    Return Floor (i/2)

Left(i)
    Return 2i

Right(i)
    Return 2i + 1
```

Σε έναν δυαδικό σωρό ελάχιστου, ο πατέρας έχει πάντα τιμή μικρότερη από τα δυο παιδιά, οπότε εύκολα μπορούμε να προσπελάσουμε τον κόμβο με το μικρότερο κόστος και με μικρή χρονική πολυπλοκότητα μπορούμε να εισάγουμε ένα νέο στοιχείο. Η χρονική πολυπλοκότητα της αναζήτησης είναι O(1) και της εισαγωγής και διαγραφής O(log n).

Όταν εισάγουμε ένα νέο στοιχείο το θέτουμε στην τελευταία θέση του δέντρου και συγκρίνοντας το επαναληπτικά με τον πατέρα του, το μεταθέτουμε στη σωστή θέση. Αυτό μπορεί να το οδηγήσει μέχρι την πρώτη θέση του δέντρου και το κόστος είναι λογαριθμικό. Αντίστοιχα, κατά την διαγραφή του πρώτου στοιχείου, μεταφέρουμε στην πρώτη θέση το τελευταίο στοιχείο και μέσω των συγκρίσεων με τα παιδιά, το μεταθέτουμε στην σωστή θέση.



Εικόνα 2.18: Δυαδικός σωρός σε πίνακα [12]



Εικόνα 2.19: Δυαδικός σωρός ελάχιστου στοιχείου [12]

ΚΕΦΑΛΑΙΟ 3
ΤΕΧΝΟΛΟΓΙΚΗ ΒΑΣΗ

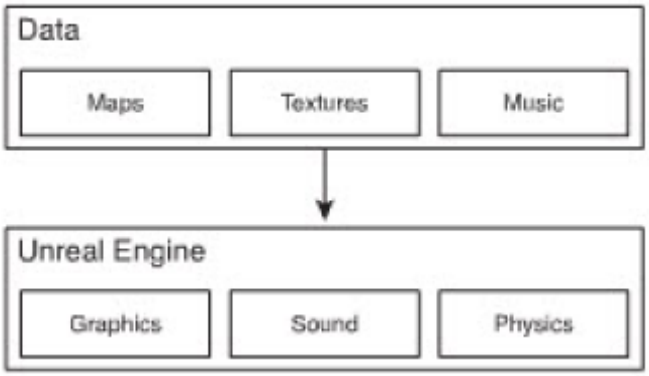
3.1 Εισαγωγή

Για την ανάπτυξη του παιχνιδιού που πραγματεύεται η παρούσα διπλωματική απαιτήθηκε η χρήση μιας μηχανής παιχνιδιών (Game engine) [13]. Στα πρώτα χρόνια της ανάπτυξης παιχνιδιών τα παιχνίδια δημιουργούνταν από το μηδέν, χωρίς τη χρήση κάποιας πλατφόρμας. Στη σύγχρονη εποχή, η χρήση λογισμικού και εργαλείων είναι απαραίτητη για την βελτίωση του αποτελέσματος, αλλά και την μείωση του χρόνου υλοποίησης. Στο συγκεκριμένο κεφάλαιο θα κάνουμε μια περιγραφή της μηχανής παιχνιδιών που χρησιμοποιήσαμε, αναλύοντας τα επί μέρους βασικά τμήματα της.

Το Unreal Development Kit είναι η δωρεάν (για ακαδημαϊκούς σκοπούς) έκδοση της μηχανής Unreal (Unreal engine) της εταιρίας Epic Games [14]. Στο δωρεάν πακέτο περιλαμβάνονται όλα τα εργαλεία και η διεπαφή προγραμματισμού εφαρμογών (Application Programming Interface – API), αλλά όχι ο πηγαίος κώδικας της μηχανής, ο οποίος είναι διαθέσιμος μόνο με την αγορά της μηχανής. Η εταιρία προσφέρει την UDK δωρεάν και για ανάπτυξη παιχνιδιών, συλλέγοντας ένα ποσοστό των κερδών στην περίπτωση που το παιχνίδι δίνεται προς πώληση.

3.2 Unreal Development Kit (UDK)

Όπως προαναφέραμε, η UDK είναι μια ελαφριά παραλλαγή της μηχανής Unreal, με τη διαφορά ότι ο χρήστης δεν έχει πρόσβαση στον πηγαίο κώδικα. Όμως, δίνονται όλες οι άλλες δυνατότητες και τα εργαλεία. Συνεπώς, η UDK είναι ταυτόχρονα και μηχανή παιχνιδιών και σουίτα εφαρμογών. Σε αυτή περιλαμβάνονται τα τμήματα της μηχανής (πυρήνας, γραφικά, ήχος, φυσική,…), οι editors, η γλώσσα προγραμματισμού (Unrealscript). Στην εικόνα 3.1 δίνεται μια απλουστευμένη άποψη της ροής των δεδομένων, έτσι όπως δομούνται για να συγκροτήσουν ένα παιχνίδι. Στις επόμενες ενότητες παρουσιάζουμε τα τμήματα της UDK με σκοπό να δείξουμε την βάση στην οποία δουλέψαμε για την υλοποίηση του παιχνιδιού.



Εικόνα 3.1: Η ροή των δεδομένων στην Unreal Engine [15]

3.3 Πυρήνας της UDK

Στις επόμενες υπό-ενότητες περιγράφουμε συνοπτικά τα τμήματα της μηχανής που βρίσκονται στον πυρήνα [15]. Στην εισαγωγή του κεφαλαίου αναφέραμε ότι στην έκδοση που δίνεται δωρεάν δεν περιλαμβάνεται ο κώδικας του πυρήνα, συνεπώς δεν γνωρίζουμε τίποτα ιδιαίτερο σχετικά με την υλοποίηση, εκτός από ό,τι γράφεται στα σχετικά άρθρα στο διαδίκτυο. Τα βασικά μέρη του πυρήνα είναι:

3.3.1 Μηχανή γραφικών

Η μηχανή γραφικών ελέγχει οτιδήποτε εμφανίζεται στην οθόνη κατά τη διάρκεια του παιχνιδιού. Είναι υπεύθυνη για τους υπολογισμούς, για τους οποίους ο παίκτης σπάνια αναρωτιέται. Για παράδειγμα, καθορίζει ποια αντικείμενα θα εμφανιστούν στην οθόνη μπροστά από άλλα ή ποια θα μείνουν κρυμμένα. Στην Unreal, τα κρυμμένα αντικείμενα δεν σχεδιάζονται καθόλου, έτσι ώστε να επιταχυνθεί η διαδικασία του rendering (η απόδοση των γραφικών στην οθόνη) [16].

Επιπλέον, ένα πολύ σημαντικό χαρακτηριστικό της μηχανής είναι το Level Streaming. Για να γίνει κατανοητό, μπορούμε να σκεφτούμε ένα τρισδιάστατο χώρο με αρκετούς μικρότερους χώρους. Όταν ο χαρακτήρας περάσει από ένα χώρο σε ένα άλλο και πλέον δεν υπάρχει ορατότητα, τότε η μηχανή σταματά το rendering του μη ορατού χώρου και εξοικονομείται έτσι επεξεργαστική ισχύς. Αυτό δίνει τη δυνατότητα στον σχεδιαστή να δημιουργήσει πολύ μεγάλους και σύνθετους χώρους, που αυξάνουν την ευχαρίστηση στον χρήστη. Η μηχανή γραφικών είναι υπεύθυνη και για την απόδοση των σκιαστών (shaders) και των υλικών (materials), τα οποία εφαρμόζονται σε όλα τα αντικείμενα.

3.3.2 Μηχανή ήχου

Ένα πολύ σημαντικό στοιχείο των παιχνιδιών είναι ο ήχος. Για να καταλάβουμε τη σημασία αυτή, αρκεί να προσπαθήσουμε να παίξουμε ένα παιχνίδι χωρίς ήχο. Η διαφορά στην διασκέδαση είναι πολύ μεγάλη. Η μηχανή ήχου δίνει την δυνατότητα στον σχεδιαστή να προσθέσει ήχους στο περπάτημα του χαρακτήρα, στο σπάσιμο ενός γυαλιού, στο νερό που κυλάει σε ένα ποτάμι, στην εκπυρσοκρότηση του όπλου αλλά και στο θέμα της μουσικής που μπορεί να αλλάζει ανάλογα με το σημείο ή την κατάσταση στην οποία βρίσκεται ο χαρακτήρας (αν για παράδειγμα δέχεται επίθεση).

Στην Unreal η μηχανή ήχου αναλαμβάνει ό,τι έχει να κάνει με τους ήχους. Ο σχεδιαστής εισάγει τους καταγεγραμμένους ήχους, δημιουργεί τα δεδομένα για το πώς θα παιχτεί ο ήχος και η αναπαραγωγή γίνεται αυτόματα, σύμφωνα με τις καθορισμένες ρυθμίσεις. Η μηχανή προωθεί τον ήχο στο hardware του συστήματος και από εκεί στα ηχεία.

3.3.3 Μηχανή φυσικής

Στον πραγματικό κόσμο κάθε αντικείμενο αντιδρά στα ερεθίσματα τα οποία λαμβάνει. Όλα τα αντικείμενα υπόκεινται στους νόμους της φυσικής, και συμπεριφέρονται σύμφωνα με αυτούς. Η μηχανή φυσικής είναι υπεύθυνη να διαχειρίζεται όλες τις συγκρούσεις των αντικειμένων καθώς και τις εκρήξεις. Ο προγραμματισμός των κανόνων φυσικής σε ένα παιχνίδι είναι πολύ απαιτητικός, ιδιαίτερα στη σύγχρονη εποχή. Ο σχεδιαστής μπορεί να χρησιμοποιήσει την μηχανή PhysX της εταιρίας nVidia.Η PhysX υποστηρίζει μια πληθώρα φυσικών προσομοιώσεων για στερεά σώματα, τους χαρακτήρες ή ακόμα και τα ρούχα.

3.3.4 Υποδομή δικτύου (Network infrastructure)

Κατά τη διάρκεια ενός παιχνιδιού πολλών ατόμων (multiplayer) κάθε υπολογιστής (client) είναι συνεχώς σε επικοινωνία με έναν υπολογιστή που συμπεριφέρεται σαν εξυπηρετητής (server). Ο server μπορεί να λειτουργήσει ταυτόχρονα και σαν client, επιτρέποντας στον χρήστη να παίξει το παιχνίδι μαζί με τους υπόλοιπους παίκτες (non-dedicated server). Το κλειδί στην καλύτερη απόδοση ενός διαδικτυακού παιχνιδιού είναι η ελαχιστοποίηση των δεδομένων που στέλνονται μεταξύ των υπολογιστών. Η μηχανή είναι προγραμματισμένη να στέλνει μόνο τα απολύτως απαραίτητα δεδομένα, όπως είναι η θέση των χαρακτήρων και η αλληλεπίδραση με το περιβάλλον. Με ελάχιστο κόπο, και τη χρήση αυτής της υποδομής ο σχεδιαστής μπορεί να δημιουργήσει multiplayer παιχνίδια.

3.3.5 Διαχειριστής εισόδου (Input manager)

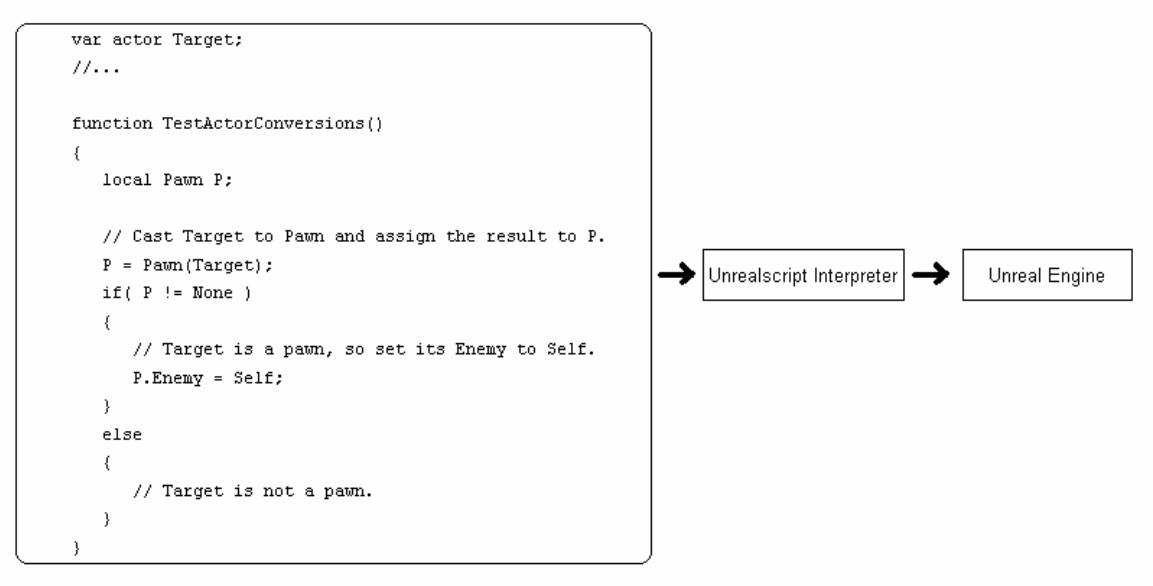
Ένα πολύ σημαντικό κομμάτι κάθε παιχνιδιού είναι η είσοδος που προέρχεται από τον χρήστη. Ο χρήστης χρησιμοποιεί το πληκτρολόγιο, το ποντίκι ή κάποιο χειριστήριο για να ορίσει στο παιχνίδι την επιθυμία του. Ο διαχειριστής πρέπει να αναγνωρίσει τις εισόδους αυτές και να τις μετατρέψει σε εντολές, τις οποίες το παιχνίδι καταλαβαίνει και αντιδρά κατάλληλα.

Για να γίνει πιο κατανοητή η διαδικασία ας περιγράψουμε την περίπτωση που ο χρήστης επιλέξει να πατήσει το πλήκτρο άλματος (jump). Ο διαχειριστής στέλνει το σήμα στον πυρήνα της μηχανής, ο οποίος με τη σειρά του τρέχει τη ρουτίνα εκτέλεσης του γραφικού άλματος. Για να παρατηρήσουμε το τελικό αποτέλεσμα λαμβάνει χώρα το rendering μέσω της μηχανής γραφικών, αλλά και η εκτέλεση του κατάλληλου ήχου μέσω της μηχανής ήχου. Όλα αυτά βέβαια συμβαίνουν σε ελάχιστο χρόνο και ο χρήστης δεν καταλαβαίνει την καθυστέρηση.

3.3.6 Διερμηνευτής της Unrealscript (Unrealscript interpreter)

Η UDK περιλαμβάνει, εκτός των άλλων και μια γλώσσα προγραμματισμού,

ειδικά διαμορφωμένη για τις ανάγκες της ανάπτυξης παιχνιδιών. Η γλώσσα αυτή θα περιγραφεί αναλυτικά σε επόμενη ενότητα. Στον πυρήνα της μηχανής υπάρχει ο διερμηνευτής της γλώσσας προγραμματισμού, ο οποίος είναι υπεύθυνος για την μετατροπή του κώδικα που γράφει ο σχεδιαστής (σε Unrealscript), σε γλώσσα μηχανής.



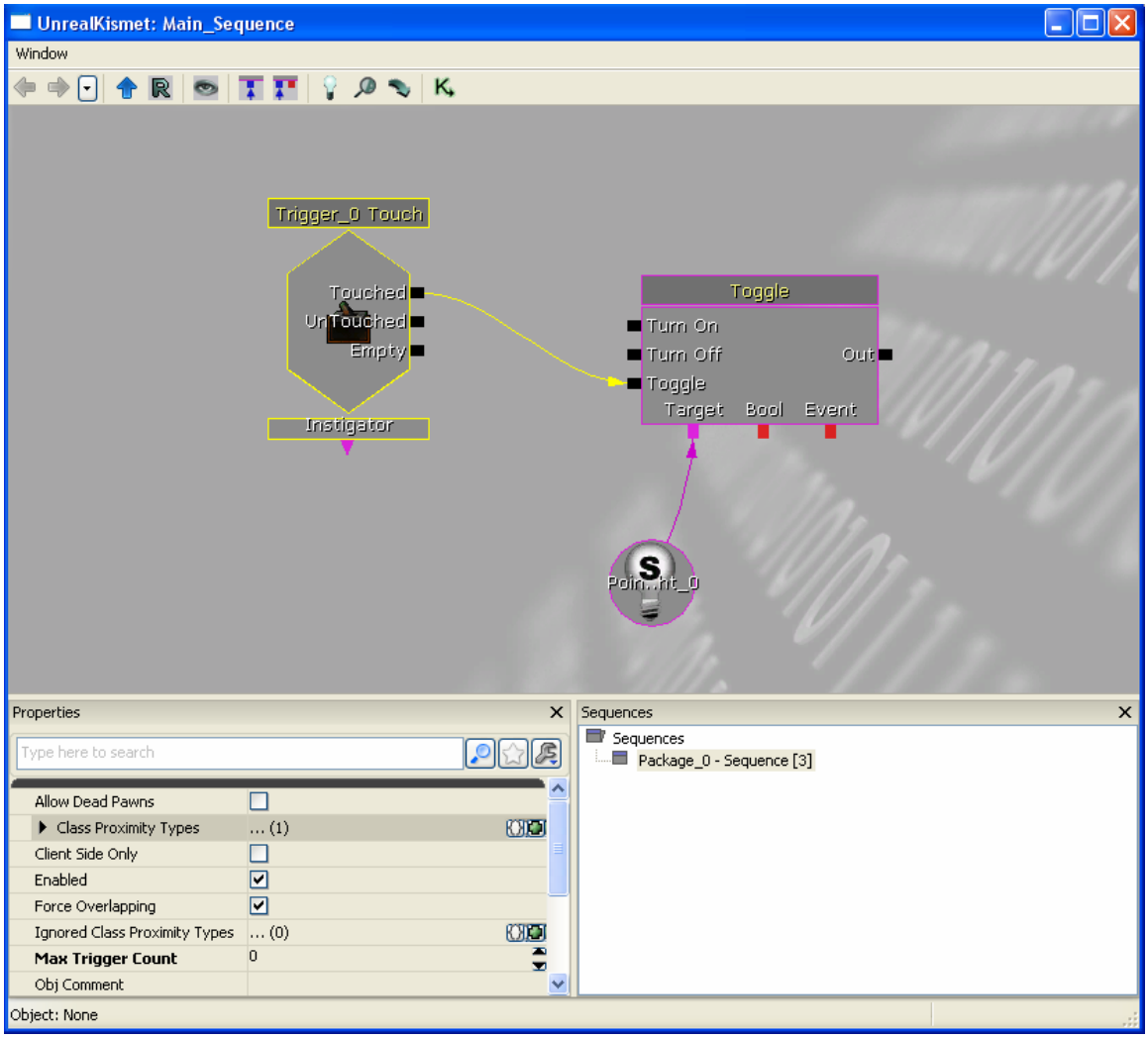
Εικόνα 3.2 Ο διερμηνευτής της Unrealscript

3.4 Unreal editor

Σε αυτήν την ενότητα θα περιγράψουμε τα βασικότερα εργαλεία του Unreal editor. Κάθε υπό-ενότητα θα εξηγήσει ένα απλό εργαλείο ή έννοια, τα οποία μαζί σχηματίζουν τον editor [17].

3.4.1 Kismet

Το Kismet είναι ένα γραφικό σύστημα της UDK, το οποίο χρησιμοποιείται κυρίως από μη προγραμματιστές, όπως level designers ή καλλιτέχνες για τον σχεδιασμό της λογικής του παιχνιδιού. Το σύστημα αυτό είναι βασισμένο σε κόμβους, οι οποίοι συνδέονται μεταξύ τους κατάλληλα και δημιουργούν κάποιο αποτέλεσμα. Τα scripts του Kismet αποθηκεύονται σε κάθε εικονικό χάρτη (map), οπότε δεν είναι δυνατόν να χρησιμοποιηθούν για τον ορισμό καθολικών κανόνων του παιχνιδιού. Στην εικόνα δίνεται ένα παράδειγμα του γραφικού, όπου φαίνεται ο χώρος στον οποίο ο χρήστης μπορεί να εισάγει μεταβλητές ή γεγονότα (events). Η λογική προκύπτει από τη σύνδεση των κόμβων με βέλη. Το Kismet μπορεί να κάνει σχεδόν τα πάντα (σε σχέση με τις δυνατότητες της Unrealscript) και αυτό προκύπτει από το γεγονός ότι υποστηρίζει μαθηματικές πράξεις, σχεσιακή λογική, διαχείριση γεγονότων και δράσεων. Το βασικότερο πλεονέκτημα του Kismet είναι ότι ακόμα και η ομάδα προγραμματιστών ενός παιχνιδιού δεν χρειάζεται να αναλωθεί στην συγγραφή κώδικα για απλά πράγματα, αλλά να επικεντρωθεί στον σχεδιασμό σύνθετων script.

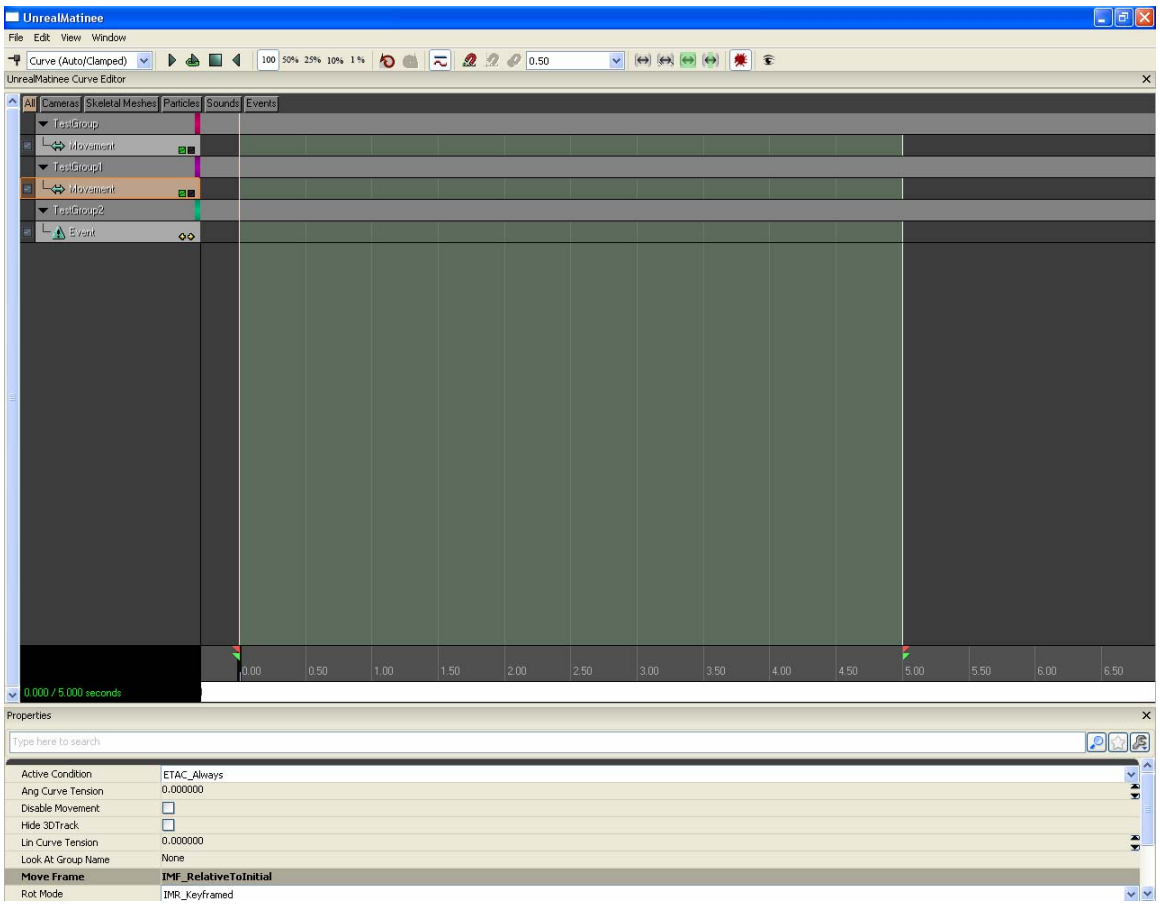


Εικόνα 3.3 Παράδειγμα οθόνης Kismet

Στο παραπάνω απλό παράδειγμα παρουσιάζουμε πώς ο χαρακτήρας μπορεί να ανάψει και να σβήσει κάποιο φως. Ο κόμβος στα αριστερά είναι τύπου trigger. Αυτό σημαίνει ότι όταν ο χαρακτήρας εισέλθει σε ένα καθορισμένο σημείο της πίστας, τότε ο κόμβος trigger δίνει έξοδο και ενεργοποιεί τον κόμβο toggle. Ο κόμβος αυτός αλλάζει την κατάσταση του αντικειμένου που είναι συνδεδεμένο στην παράμετρο target. Με αυτόν τον τρόπο, κάθε φορά που ο χαρακτήρας εισέρχεται στο καθορισμένο σημείο, ανάβει ή σβήνει το φως.

3.4.2 Matinee

Το Matinee είναι ένα εργαλείο της UDK με το οποίο μπορεί να γίνει αλλαγή των ιδιοτήτων ενός αντικειμένου κατά τη διάρκεια του χρόνου. Η λειτουργικότητα του ποικίλει από την δημιουργία animation για κινούμενες πόρτες ή ασανσέρ, μέχρι σχεδιασμό cut-scene με χρήση σύνθετων εφφέ. Το Matinee είναι άμεσα συνδεδεμένο με το Kismet, καθώς η εκκίνηση μιας ακολουθίας γίνεται μέσα από το δεύτερο. Παρακάτω δίνεται ένα παράδειγμα του εργαλείου.



Εικόνα 3.4 Παράδειγμα οθόνης Matinee

Το Matinee είναι δομημένο σε groups. Κάθε group περιέχει ένα αριθμό από tracks, τα οποία ελέγχουν τις ιδιότητες ενός αντικειμένου. Για να γίνει αλλαγή μιας ιδιότητας αρκεί να εισαχθεί ένα κλειδί, ώστε να αποθηκευτεί η πληροφορία της τιμής στο χρόνο. Οι τιμές μιας ιδιότητας μεταξύ δυο κλειδιών καθορίζονται από το χρήστη με τη χρήση σχημάτων παρεμβολής (interpolation schemes). Για την αλλαγή των διαφόρων ιδιοτήτων των αντικειμένων υπάρχουν διαφορετικά tracks. Επιγραμματικά, τα βασικότερα είναι:

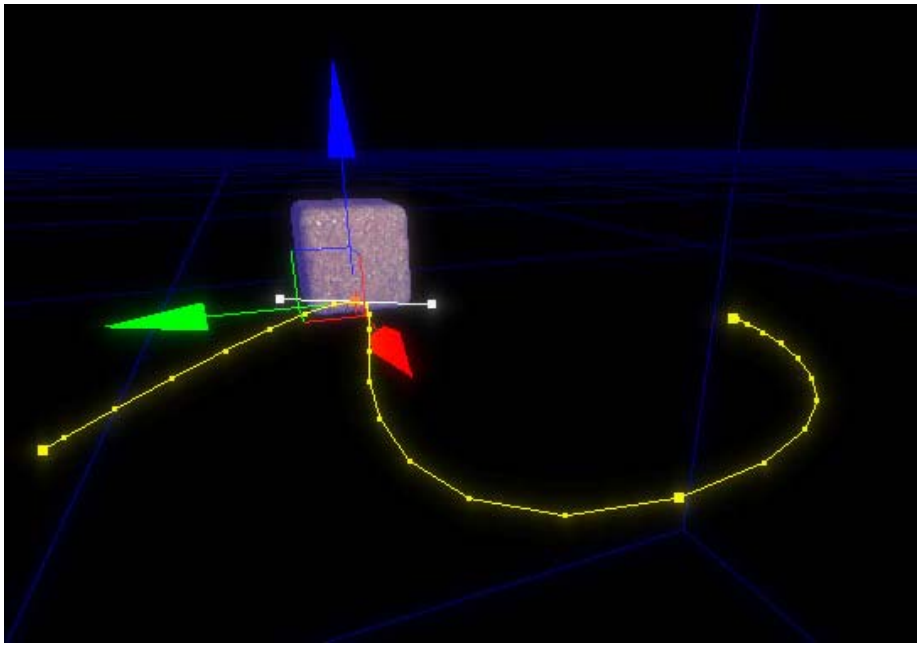
Movement track: Είναι ένα από τα βασικότερα tracks και χρησιμοποιείται για την μετακίνηση ενός αντικειμένου. Στην παρακάτω εικόνα φαίνεται η καμπύλη μετακίνησης μιας πέτρας.

Float, vector, color track: Ελέγχουν τις ιδιότητες για τα διάφορα ήδη μεταβλητών. Π.χ. Με το color track μπορεί να αλλαχθεί η φωτεινότητα ενός φωτός.

Anim control track: Αυτός ο τύπος χρησιμοποιείται για να καθοριστεί ποιο animation να παιχτεί σε διαφορετικά διαστήματα μιας ακολουθίας.

ParticleSystem Toggle track: Χρησιμοποιείται από τον χρήστη για την ενεργοποίηση και απενεργοποίηση των σωματιδίων (particles). Για παράδειγμα, ενεργοποιεί το εφφέ της έκρηξης, όταν το βλήμα του όπλου χτυπήσει μια περιοχή.

Slomo track: Ελέγχει την ταχύτητα όλων των ενεργειών, της φυσικής, των εφφέ, του ήχου, κτλ.



Εικόνα 3.5 Καμπύλη του movement track

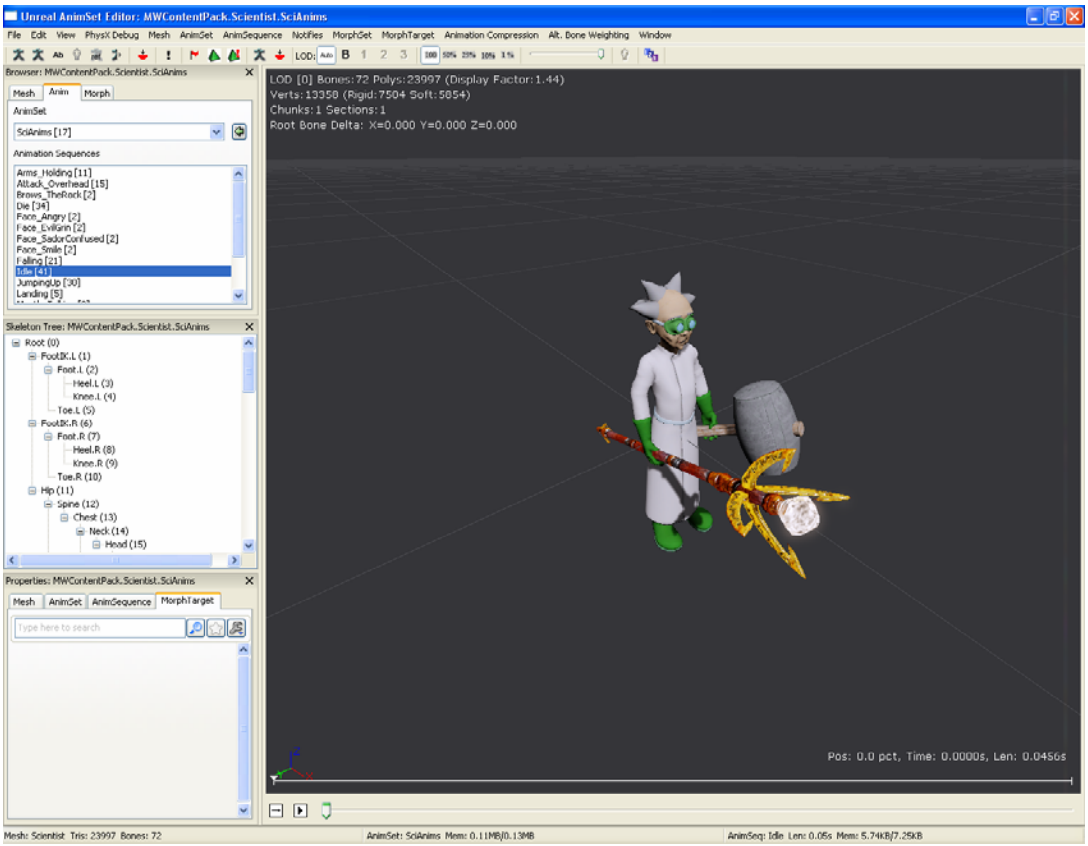
3.4.3 AnimSet editor

Ο AnimSet editor της UDK είναι το βασικό εργαλείο για την οργάνωση και προβολή των animations των skeletal meshes. Σε αυτό μπορούμε να διαχειριστούμε τις περισσότερες πληροφορίες που σχετίζονται με τα skeletal meshes, όπως τη θέση τους στο χώρο, την περιστροφή τους ή τα ρούχα τους. Η γραφική διεπαφή του editor δίνεται στην επόμενη εικόνα. Ο συγκεκριμένος editor έχει διάφορα χαρακτηριστικά, από τα οποία τα σημαντικότερα είναι:

Sockets: Τα sockets είναι σημεία που είναι τοποθετημένα σε συγκεκριμένα άκρα του skeletal mesh. Όταν το skeletal mesh μετακινείται, μαζί του μετακινείται και το socket. Το ενδιαφέρον με τα sockets είναι ότι μπορούμε να συνδέσουμε κάποιο όπλο για παράδειγμα, έτσι ώστε να κινείται και αυτό μαζί με την κίνηση των χεριών του χαρακτήρα. Με τη χρήση των sockets μπορούμε να συνδέουμε σε κάθε χαρακτήρα ένα αντικείμενο, αντί να έχουμε διαφορετικούς χαρακτήρες για κάθε περίπτωση.

Mirror tables: Τα mirror tables εκμεταλλεύονται την παρατήρηση ότι τα περισσότερα skeletal meshes είναι συμμετρικά. Κάθε bone έχει το συμμετρικό του και γνωρίζοντας αυτό, ο σχεδιαστής μπορεί να μειώσει την εργασία του στο μισό.

Cloth physics: Υπάρχουν μέρη ενός skeletal mesh τα οποία πιθανώς θέλουμε να καθορίζονται από την μηχανή φυσικής και όχι από κάποιο animation. Τέτοια μέρη μπορεί να είναι κάποιο ρούχο, μια σημαία ή ένα μαντήλι. Η κίνηση αυτών καθορίζεται από τα cloth physics.



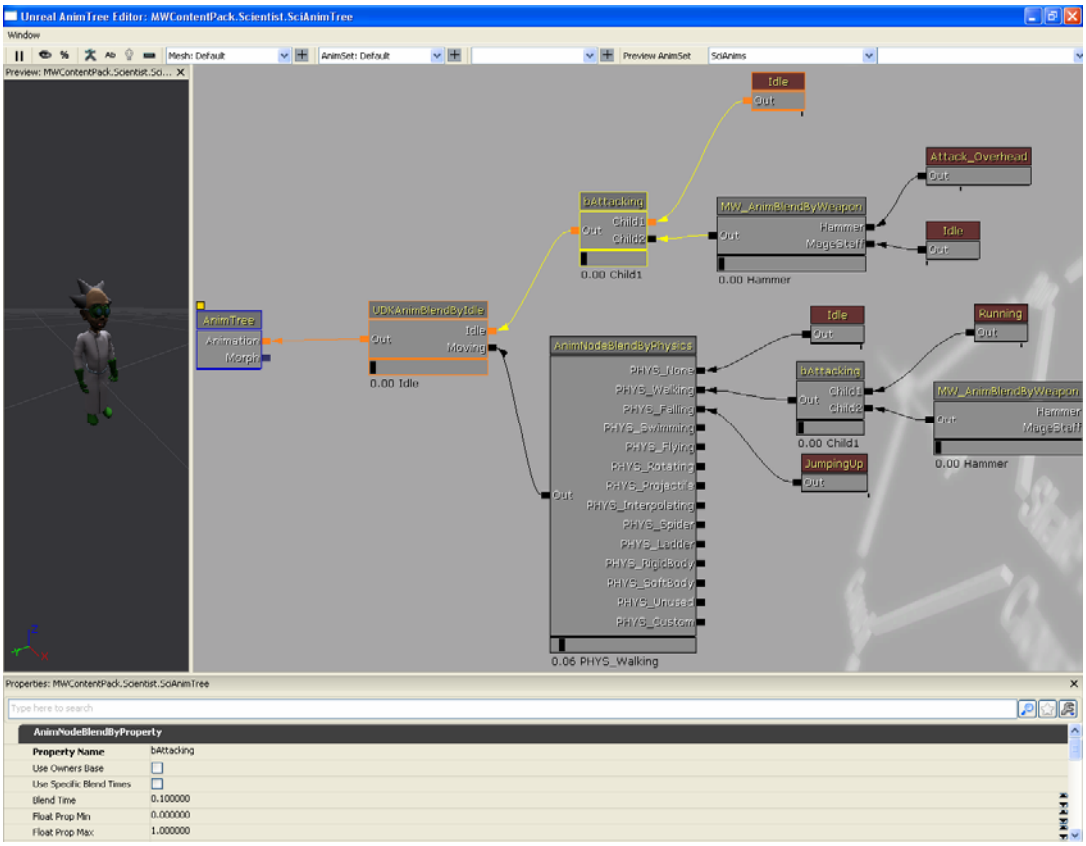
Εικόνα 3.6 Γραφική διεπαφή AnimSet editor

3.4.4 AnimTree editor

Ο AnimTree editor είναι ένα γραφικό εργαλείο για την δημιουργία animation δέντρων (animation trees). Δίνεται η δυνατότητα στον σχεδιαστή να καθορίσει πως και ποια μέρη ενός 3D μοντέλου να κινηθούν. Για να καταλάβουμε καλύτερα, θα δώσουμε ένα παράδειγμα.

Ένα πολύ σημαντικό χαρακτηριστικό είναι το animation blending. Αυτό σημαίνει ότι ο σχεδιαστής μπορεί να συνδυάσει 2 ή περισσότερα animations. Για παράδειγμα, στην περίπτωση που ο χαρακτήρας τρέχει και ταυτόχρονα κάνει επίθεση με τα χέρια, μπορούμε να συνδυάσουμε το animation του τρεξίματος με το animation της επίθεσης και να φαίνεται το αντίστοιχο αποτέλεσμα στον χρήστη.

Έστω ότι έχουμε ένα χαρακτήρα ενός επιστήμονα, όπως στο παιχνίδι που υλοποιήσαμε. Θέλουμε όταν βρίσκεται ακίνητος να παίζεται ένα animation ακινησίας, όταν τρέχει αντίστοιχα ένα animation τρεξίματος, όταν επιτίθεται να κινεί τα χέρια του ανάλογα και ούτω καθεξής. Για να συμβούν όλα αυτά θα πρέπει να συνδέσουμε κατάλληλα τους κόμβους, όπως φαίνεται στην παρακάτω εικόνα. Η ιδέα είναι ότι χρησιμοποιώντας π.χ. ένα κόμβο AnimBlendByIdle μπορούμε να ορίσουμε διαφορετικά animation ανάλογα με το αν ο χαρακτήρας κινείται ή μένει σταθερός. Αντίστοιχα, με ένα κόμβο AnimBlendByProperty μπορούμε να ορίσουμε διαφορετικά animations, ανάλογα με το αν ο χαρακτήρας επιτίθεται ή όχι.



Εικόνα 3.7 Γραφική διεπαφή AnimTree editor

3.4.5 Lightmass

Το Lightmass είναι το σύστημα φωτισμού της UDK. Αποδίδει υψηλής ποιότητας στατικό φωτισμό και χάρτες σκίασης (shadow maps) χρησιμοποιώντας τεχνικές που είναι πολύ σύνθετες για να τρέξουν σε πραγματικό χρόνο σε έναν υπολογιστή. Για αυτό το λόγο το Lightmass τρέχει κατά τη διάρκεια σχεδίασης της πίστας, από τον level designer. Διαθέτει πολλά σύγχρονα χαρακτηριστικά όπως φωτισμό περιοχών και σκιές με την εκπομπή του φωτός να γίνεται από διαφορετικά είδη πηγών, φωτισμό χαρακτήρων, έμμεσο φωτισμό, διαφανείς σκιές, κτλ. Ακόμα, το Lightmass μας δίνει την δυνατότητα να επιλέξουμε ανάμεσα σε τέσσερις επιλογές ποιότητας του αποτελέσματος, από την απλούστερη ποιότητα (γρηγορότερη) μέχρι την ποιότητα παραγωγής.

Επειδή πρόκειται για μια χρονοβόρα διαδικασία, η UDK προσφέρει ένα επιπλέον εργαλείο (Unreal Swarm), το οποίο επιτρέπει τον παράλληλο υπολογισμό σε πολλαπλούς υπολογιστές. Τέτοιες διαδικασίες χρησιμοποιούνται και στις σύγχρονες ταινίες και στην τηλεόραση, όπου «φάρμες» υπολογιστών συνθέτουν το αποτέλεσμα του φωτισμού των σκηνών.

3.5 Unrealscript

Η Unrealscript είναι η γλώσσα προγραμματισμού που χρησιμοποιείται στην μηχανή Unreal 3 και συνεπώς και στην UDK [18]. Η γλώσσα δημιουργήθηκε για την πρώτη μηχανή Unreal από τον Tim Sweeney, τον ιδρυτή της εταιρίας Epic. Στις επόμενες εκδόσεις επεκτάθηκε και ενισχύθηκε με πολλές δυνατότητες. Ακολουθεί τις ίδιες αντικειμενοστραφείς αρχές (object-oriented), όπως η Java και η C++, όμως έχει και επιπλέον στοιχεία, τα οποία δεν μπορεί να βρει κάποιος στις παραδοσιακές γλώσσες προγραμματισμού. Τα στοιχεία αυτά στοχεύουν στον εύκολο προγραμματισμό παιχνιδιών και περιλαμβάνουν:

Time – Οι συναρτήσεις που παίρνουν κάποιο χρόνο για να ολοκληρωθούν μπορεί να είναι ιδιαίτερα δύσκολες στο να προγραμματιστούν σε γλώσσες όπως η C++. Συχνά απαιτούν την χρήση κάποιου επιπρόσθετου νήματος ή κάποιο άλλο τρόπο, ώστε να αποφευχθεί το κόλλημα του υπολογιστή. Η Unrealscript λύνει αυτό το πρόβλημα με τη χρήση λανθανόντων (latent) συναρτήσεων. Οι λανθάνουσες συναρτήσεις τρέχουν στο «πίσω» μέρος του υπολογιστή χωρίς να παρεμβαίνουν με το υπόλοιπο σύστημα, απλοποιώντας με αυτό τον τρόπο την αλυσίδα των γεγονότων που περιλαμβάνουν κάποιο πέρασμα του χρόνου.

State – Η λογική των παιχνιδιών και συνήθως ειδικά η τεχνητή νοημοσύνη (AI) έχουν άμεση σχέση με συμπεριφορές που επηρεάζονται από την κατάσταση που βρίσκεται ένα αντικείμενο. Η Unrealscript παρέχει ένα σύστημα καταστάσεων (states), το οποίο δίνει την δυνατότητα να υπάρχουν στην ίδια κλάση πολλαπλές εκδόσεις της ίδιας συνάρτησης. Με αυτό τον τρόπο, ανάλογα με την κατάσταση στην οποία βρίσκεται ένα αντικείμενο μπορεί να τρέχει η ανάλογη έκδοση της συνάρτησης.

Network replication – Να διατηρηθεί η συνέπεια μεταξύ του εξυπηρετητή (server) και του πελάτη (client) μπορεί να είναι τρομερά δύσκολη υπόθεση για τους προγραμματιστές. Στην Unrealscript συμπεριλαμβάνονται τα στοιχεία αυτά που κρατούν συνεπή τα δεδομένα και στις δυο πλευρές. Όμως, λόγω του περιορισμένου εύρους στις συνδέσεις δικτύου είναι σημαντικό μόνο τα απαραίτητα δεδομένα να στέλνονται προς τον εξυπηρετητή και αντιστρόφως.

Τα βασικά στοιχεία στην Unrealscript που προέρχονται από την Java είναι το περιβάλλον χωρίς δείκτες (pointers), ο αυτόματος συλλέκτης «σκουπιδιών» (automatic garbage collector), απλή ιεραρχία κλάσεων (υπάρχει μια αρχική κλάση και όλες οι υπόλοιπες κλάσεις πηγάζουν από αυτή) και έλεγχος λαθών κατά την σύνταξη (compilation).

3.5.1 Ιεραρχία αντικειμένων

Ιστορικά, τα περισσότερα παιχνίδια τρισδιάστατων γραφικών σχεδιάστηκαν μονολιθικά, με την βασική λειτουργικότητα να είναι μη επεκτάσιμη και τα δεδομένα εισόδου να είναι ενσωματωμένα στον πηγαίο κώδικα. Το βασικό πλεονέκτημα της

Unreal είναι η δυνατότητα να προστεθεί λειτουργικότητα με την επέκταση των ήδη υπαρχόντων κλάσεων με νέες, αντί της μετατροπής κάποιου έτοιμου κώδικα.

Η ιεραρχία της Unreal είναι 100% αντικειμενοστραφής και υποστηρίζει υψηλού επιπέδου αντικειμενοστραφείς έννοιες, όπως γράφους αντικειμένων, σειριοποίηση και πολυμορφισμό. Κάθε κλάση στην Unreal παράγεται χρησιμοποιώντας μια υπέρ-κλάση. Η βασικότερες κλάσεις από τις οποίες όλες οι υπόλοιπες παράγονται είναι οι εξής:

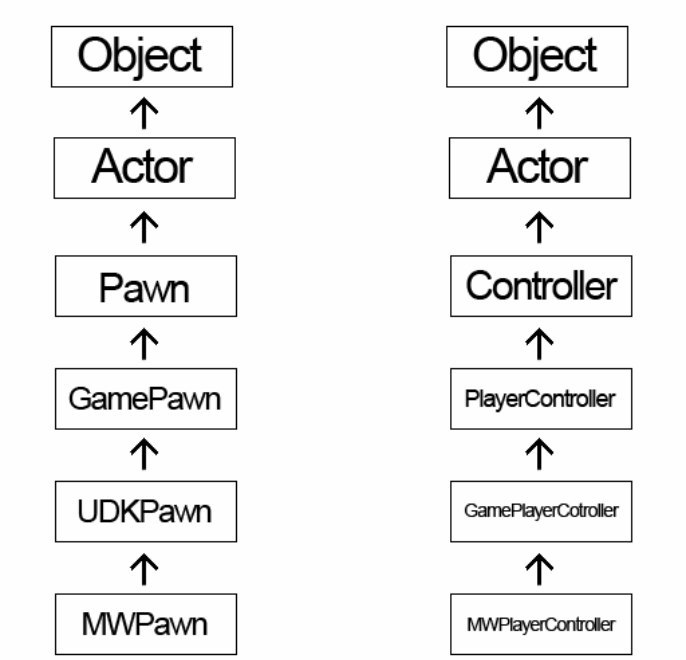
Object – Είναι η γονική κλάση όλων των κλάσεων στην Unreal. Όλες οι συναρτήσεις στην κλάση Object είναι προσπελάσιμες από παντού γιατί όλες οι κλάσεις παράγονται από αυτή. Η Object είναι μια αφηρημένη κλάση, υπό την έννοια ότι δεν κάνει τίποτα χρήσιμο. Όλη η λειτουργικότητα προέρχεται από τις υπό-κλάσεις, όμως σε αυτή βρίσκονται δεκάδες μεταβλητές, δομές και συναρτήσεις που είναι χρήσιμες σε όλες τις υπόλοιπες κλάσεις.

Actor – Προέρχεται από την κλάση Object και είναι η γονική κλάση όλων των αυτόνομων αντικειμένων στην Unreal. Η κλάση Actor περιέχει όλη την λειτουργικότητα που χρειάζεται ένα αντικείμενο (actor) για να κινηθεί, να αλληλεπιδράσει με άλλα αντικείμενα, να επηρεάσει το περιβάλλον και να κάνει όλα τα χρήσιμα σχετικά με το παιχνίδι πράγματα.

Pawn – Προέρχεται από την κλάση Actor και είναι η γονική κλάση όλων των παιχτών και πλασμάτων, τα οποία είναι ικανά για υψηλού επιπέδου τεχνητή νοημοσύνη ή έλεγχο από τον παίχτη.

Class – Προέρχεται από την κλάση Object και είναι ένα ειδικού είδους αντικείμενο, το οποίο περιγράφει την κλάση ενός αντικειμένου. Αν και φαίνεται αρχικά κάπως μπερδεμένο, μια κλάση είναι ένα αντικείμενο και η κλάση περιγράφει ορισμένα αντικείμενα. Για παράδειγμα, όταν δημιουργούμε ένα νέο αντικείμενο (actor), μπορούμε να ορίσουμε την κλάση του αντικειμένου χρησιμοποιώντας τον τύπο Class.

Για να αντιληφθούμε καλύτερα την λογική των κλάσεων της Unreal, αρκεί να δούμε ποιες βασικές κλάσεις είναι απαραίτητες για τον χαρακτήρα που χειρίζεται ο παίκτης. Σχεδόν κάθε αντικείμενο που βρίσκεται στον χώρο είναι τύπου Actor. Η οπτική αναπαράσταση του παίκτη στο παιχνίδι ονομάζεται πιόνι (Pawn) και προέρχεται από αυτή την κλάση Actor. Δεν υπάρχει τίποτα στην κλάση Pawn που να ορίζει πώς ελέγχεται ένα πιόνι, παρότι στην πραγματικότητα μπορεί να ελέγχεται από ένα άνθρωπο ή τον υπολογιστή. Η συγκεκριμένη κλάση ορίζει ποιο 3D μοντέλο να χρησιμοποιηθεί, ποιες συναρτήσεις για την κίνηση, επίθεση και οτιδήποτε άλλο το πιόνι πρέπει να είναι ικανό να κάνει. Το «μυαλό» του πιονιού βρίσκεται στην κλάση Controller. Στην περίπτωση που το πιόνι ελέγχεται από έναν άνθρωπο ο ελεγκτής θα είναι μια κλάση PlayerController (η οποία θα δέχεται εισόδους από το πληκτρολόγιο και το ποντίκι), ενώ αν ελέγχεται από τον υπολογιστή ο ελεγκτής θα είναι μια κλάση AIController. Στην επόμενη εικόνα δίνεται η ιεραρχία των κλάσεων μέχρι τις κλάσεις Pawn και Controller του παιχνιδιού Mad World που υλοποιήσαμε.



Εικόνα 3.8 Ιεραρχία κλάσεων μέχρι τις κλάσεις Pawn κ PlayerController του παιχνιδιού Mad World

3.5.2 Κλάσεις

Κάθε αρχείο της Unrealscript περιέχει μια κλάση και ξεκινά δηλώνοντας την κλάση, την υπέρ-κλάση και κάθε άλλη πληροφορία που είναι σχετική με την κλάση αυτή. Η απλούστερη μορφή είναι:

```
class MyClass extends MyParentClass;
```

Στο παραπάνω παράδειγμα δηλώνουμε μια καινούργια κλάση με όνομα “MyClass”, η οποία κληρονομεί την λειτουργικότητα της “MyParentClass”. Κάθε κλάση κληρονομεί όλες τις μεταβλητές, τις συναρτήσεις και τις καταστάσεις (states) από την γονική κλάση. Κατόπιν, σε αυτή μπορούν να προστεθούν νέες μεταβλητές, νέες συναρτήσεις ή να γίνει παράκαμψη (override) μιας υπάρχουσας και νέες καταστάσεις (states) ή να προστεθεί λειτουργικότητα σε κάποια που βρίσκεται στην γονική κλάση.

Μια τυπική προσέγγιση στο σχεδιασμό κλάσεων στην Unrealscript είναι να δημιουργούμε μια νέα κλάση, η οποία να έχει ως υπέρ-κλάση, την κλάση που έχει την περισσότερη λειτουργικότητα σε σχέση με αυτή που χρειαζόμαστε. Με αυτή την προσέγγιση δεν χρειάζεται να επανεφεύρουμε τον τροχό για κάθε τι νέο που θέλουμε να δημιουργήσουμε. Απλά, προσθέτουμε την νέα λειτουργικότητα κρατώντας οτιδήποτε από την υπάρχουσα που δεν θέλουμε να προσαρμόσουμε. Για παράδειγμα, στον προγραμματισμό τεχνητής νοημοσύνης, μπορεί κανείς να χρησιμοποιήσει αρκετές συναρτήσεις ως βασική λειτουργικότητα και να προσαρμόσει μόνο αυτές που ενδιαφέρεται. Ο ορισμός της κλάσης μπορεί να πάρει αρκετούς προσδιοριστές που επιδρούν στην κλάση με διαφορετικούς τρόπους. Κάποιοι από αυτούς είναι οι λέξεις κλειδιά: native, abstract, transient, config, placeable, κτλ.

3.5.3 Μεταβλητές

Οι μεταβλητές μπορούν να εμφανιστούν σε δυο μέρη στην Unrealscript. Οι καθολικές μεταβλητές εμφανίζονται αμέσως μετά την δήλωση της κλάσης, ενώ οι τοπικές μεταβλητές στην αρχή των συναρτήσεων ή μέσα στις δομές. Προφανώς, οι τοπικές μεταβλητές είναι ενεργές μόνο κατά τη διάρκεια της εκτέλεσης της συνάρτησης.

```
var float pi;
local string myName;
```

Οι τύποι μεταβλητών που υποστηρίζονται από την Unrealscript είναι οι εξής:

- **byte**: Μια τιμή ενός byte που κυμαίνεται από 0 ως 255
- **int**: Ένας 32-bit ακέραιος αριθμός
- **bool**: Μια λογική τιμή. True/false
- **float**: Ένας 32-bit αριθμός κινητής υποδιαστολής
- **string**: Μια σειρά χαρακτήρων
- **constant**: Μια μεταβλητή που δεν μπορεί να τροποποιηθεί
- **array<Type>**: Ένας πίνακας του τύπου "Type"
- **enumeration**: Μια μεταβλητή που μπορεί να πάρει μια σειρά από ακέραιες τιμές προκαθορισμένου ονόματος.
- **struct**: Όμοια με τις δομές στην γλώσσα C, οι δομές στην Unrealscript επιτρέπουν την δημιουργία νέων τύπων μεταβλητών, που περιέχουν άλλες μεταβλητές. Για παράδειγμα, δυο πολύ χρησιμοποιούμενες δομές στην Unreal είναι οι δομές vector (περιέχει τις μεταβλητές X,Y,Z) και rotator.
- **name**: Χρησιμοποιείται για την αποθήκευση του ονόματος ενός αντικειμένου στην Unreal, όπως το όνομα μιας συνάρτησης, κατάστασης (state), κλάσης, κτλ. Αποθηκεύονται σε έναν καθολικό πίνακα ως δείκτες για ταχύτερη προσπέλαση.
- **object and Actor references**: Μεταβλητή που αναφέρεται σε ένα άλλο αντικείμενο στον τρισδιάστατο χώρο. Μέσω αυτών των μεταβλητών μπορούμε να προσπελάσουμε τις μεταβλητές ενός άλλου αντικειμένου. Μπορούν να πάρουν την ειδική τιμή "none", που είναι αντίστοιχη με τιμή "null" στην C.
- **delegate**: Τα delegates είναι αναφορές σε κάποια συνάρτηση μέσα σε ένα στιγμιότυπο ενός αντικειμένου. Πρόκειται για τον συνδυασμό δυο προγραμματιστικών ιδεών, των συναρτήσεων και των μεταβλητών. Οι μεταβλητές διατηρούν κάποιες τιμές που μπορούν κατά τη διάρκεια της εκτέλεσης του προγράμματος να αλλάξουν. Υπό κάποια έννοια τα delegates είναι μεταβλητές που κρατούν κάποιες τιμές, μόνο που αυτές οι τιμές είναι άλλες συναρτήσεις μέσα στην κλάση. Επίσης, εκτελούνται σαν συναρτήσεις και για αυτό είναι ένα τόσο ισχυρό εργαλείο στον προγραμματισμό παιχνιδιών.

3.5.4 Εκφράσεις

Οι εκφράσεις είναι ένα βασικό στοιχείο κάθε γλώσσας προγραμματισμού. Στην Unrealscript για να θέσουμε μια τιμή σε μία μεταβλητή χρησιμοποιούμε τον τελεστή "=".

```
function Test()
{
   local int i;
   local string s;
   local vector v, q;

   i = 10;
   s = "Hello!";
   v = q;
}
```

Ένα χαρακτηριστικό της Unrealscript είναι η αυτόματη μετατροπή ενός τύπου σε έναν άλλο, όταν σε μια έκφραση ορίσουμε διαφορετικούς τύπους στις δυο πλευρές. Για παράδειγμα, αν θέσουμε μια ακέραια τιμή σε μια πραγματική, ο μεταγλωττιστής κάνει αυτόματα την μετατροπή της σε πραγματική για να γίνει η πράξη.

```
function Test()
{
   local int a;
   local float b;

   a = 1;
   b = a + 2.5;
}
```

Τέτοιες μετατροπές μπορούν να γίνουν και σε άλλους τύπους μεταβλητών, όμως σε ορισμένες περιπτώσεις ο προγραμματιστής πρέπει να ορίσει επ' ακριβώς την μετατροπή που επιθυμεί. Αντίστοιχα με τις απλές μετατροπές που αναφέραμε, γίνονται μετατροπές και σε πιο σύνθετες, όπως οι αναφορές σε αντικείμενα και Actors.

3.5.5 Συναρτήσεις

Στην Unrealscript μπορούμε να ορίσουμε νέες συναρτήσεις ή να γράψουμε νέες εκδόσεις από υπάρχουσες συναρτήσεις. Αν και οι περισσότερες συναρτήσεις είναι γραμμένες κατευθείαν σε Unrealscript, μπορούμε να γράψουμε συναρτήσεις σε C++ και να κληθούν από οποιοδήποτε σημείο. Η μορφή μιας απλής συνάρτησης είναι:

```
// Επιστρέφει το μέγεθος ενός διανύσματος
function float VectorSize( vector V )
{
   return sqrt( V.X * V.X + V.Y * V.Y + V.Z * V.Z );
}
```

Επιπρόσθετα, με την ειδική λέξη “out” μπορούμε να εξάγουμε παραπάνω από μια μεταβλητές μέσω των ορισμάτων μιας συνάρτησης. Με την λέξη “optional” μπορούμε να κάνουμε ορισμένες παραμέτρους μιας συνάρτησης προαιρετικές.

Όπως αναφέραμε, μπορούμε να γράψουμε νέες εκδόσεις μια συνάρτησης σε κάποια υπό-κλάση. Αν για παράδειγμα θέλουμε να δημιουργήσουμε την κλάση του αντιπάλου, αρκεί να βρούμε την σωστή υπέρ-κλάση, να την θέσουμε στην κλάση που δημιουργούμε και να κάνουμε παράκαμψη (override) μόνο σε αυτές τις συναρτήσεις που θέλουμε να έχουν διαφορετική λειτουργικότητα από την υπάρχουσα. Και στις συναρτήσεις, όπως και στις κλάσεις, υπάρχουν λέξεις που επιδρούν ανάλογα προσδίδοντας χαρακτηριστικά. Τέτοιες είναι οι static, latent, final, iterator, simulated, event, κτλ.

3.5.6 Δομές ελέγχου

Η Unrealscript υποστηρίζει όλες τις βασικές δηλώσεις ελέγχου, όπως στις γλώσσες C/C++/Java. Υποστηρίζει βρόχους “for”, “do…until”, “while”, και δηλώσεις “if”, “switch” και “goto”.

```
//Βρόχος For
for( i=0; i<4; i++ )
{
   log( "The value of i is " $ i );
   if( i == 2 )
      continue;
}

//Βρόχος do…until
do
{
   log( "The value of i is " $ i );
   i = i + 1;
} until( i == 4 );

//Βρόχος while
while( i < 4 )
{
   log( "The value of i is " $ i );
   i = i + 1;
   if( i == 3 )
      break;
}

//Δήλωση if
if( i < 10)
{}
else if ( i = 10)
{}
else
{}
```

3.5.7 Παράδειγμα κώδικα σε Unrealscript

Η ενότητα σχετικά με την Unrealscript τελειώνει με ένα παράδειγμα, του οποίου σκοπός είναι να δώσει μια καλύτερη εικόνα στον τρόπο που ορίζεται μια απλή κλάση. Προφανώς, η ενότητα αυτή δεν αντικαθιστά το εγχειρίδιο της γλώσσας, παρά μόνο δίνει μια αρχική εικόνα για τα βασικά στοιχεία και χαρακτηριστικά της.

Στο παράδειγμα, ο αντίπαλος μόλις δημιουργηθεί, βρίσκεται σε κατάσταση περιήγησης (λόγω της λέξης "auto", που ορίζει την αρχική κατάσταση) και αναζητά τον εχθρό του κάθε τρία δευτερόλεπτα. Όση ώρα δεν τον βρίσκει, απλά περπατά τυχαία μέσα στην πίστα, ενώ μόλις τον εντοπίσει (μέσω του γεγονότος "SeePlayer"), περνά σε κατάσταση επίθεσης. Εκεί, συνεχίζει να επιτίθεται για όση ώρα ο αντίπαλος του είναι στο πεδίο όρασης του. Μόλις χαθεί, επανέρχεται στην κατάσταση περιήγησης.

```
class Creature extends UDKBot;

auto state Wandering
{
	event SeePlayer(Pawn SeenPlayer)
	{
		GoToState('Attack');
	}

	function Walk()
	{
		//Walk randomly
	}

	Begin:
		Sleep(3);
		if(!SpottedPlayer())
			Walk();
		Goto('Begin');
}

state Attack
{
	event EnemyNotVisible()
	{
		GoToState('Wandering');
	}

	function AttackEnemy()
	{
		//Fire...
	}

	Begin:
		AttackEnemy();
		goto('Begin');
}
```

ΚΕΦΑΛΑΙΟ 4
ΣΧΕΔΙΑΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ ΚΑΙ ΓΡΑΦΙΚΗ ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ

4.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο εξετάσαμε την τεχνολογία, την οποία χρησιμοποιήσαμε για να υλοποιήσουμε το παιχνίδι της παρούσης διπλωματικής. Σε αυτό το κεφάλαιο θα παρουσιάσουμε το σενάριο στο οποίο βασιστήκαμε, τον σχεδιασμό του παιχνιδιού και την γραφική διεπαφή χρήστη.

4.2 Σενάριο παιχνιδιού

Ο Johann είναι ένας τρελός επιστήμονας που ζει και εργάζεται στο εργαστήριο του σε μια ερημική περιοχή. Εκεί, κλεισμένος μέρα και νύχτα δημιουργεί, άλλες φορές με επιτυχία, άλλες με αποτυχία, διάφορα αντικείμενα ή όντα. Μέχρι τώρα έχει δημιουργήσει έναν αριθμό από όπλα, κάποια μαγικά φίλτρα και κάποια τέρατα, τα οποία προορίζει για τον στρατό που προσπαθεί να συγκεντρώσει. Στόχος του είναι να καταλάβει τη χώρα, την οποία θέλει να κυβερνήσει.

Ως αυτή τη στιγμή έχει ολοκληρώσει ένα μέρος της δουλειάς του (κάποια μαγικά φίλτρα και κάποια όπλα), όμως δεν έχει "προγραμματίσει" σωστά τα τέρατα. Τα τέρατα του δεν έχουν κάποια νόηση, εκτός από το να θέλουν να σκοτώσουν οποιονδήποτε, ή να προστατέψουν κάποιες περιοχές που θεωρούν δικές τους. Γι' αυτό το λόγο ο Johann τα κρατάει κλεισμένα σε φυλακές για να μην κινδυνεύει έως ότου ολοκληρώσει το έργο του και να μπορεί να έχει τον απόλυτο έλεγχο στις πράξεις τους.

Μια νύχτα όμως, σαν όλες τις άλλες, που ο Johann δούλευε ως αργά, ένας ανεμοστρόβιλος έπληξε το εργοστάσιο της εταιρίας ηλεκτρισμού, γεγονός που είχε ως αποτέλεσμα τα περισσότερα κελιά στα οποία ήταν κλεισμένα τα τέρατα να ξεκλειδώσουν. Τα τέρατα ελευθερώθηκαν και δραπέτευσαν από το εργαστήριο. Ο Johann που κινδυνεύει και ο ίδιος, κατάλαβε ότι το σχέδιο του ναυάγησε και ότι πρέπει να τα κυνηγήσει και να τα φυλακίσει. Σε κάποιες περιπτώσεις που είναι αδύνατο να τα επιστρέψει στα κελιά, θα πρέπει να τα σκοτώσει με κάποιο από τα όπλα του. Ο λόγος είναι ότι τα τέρατα δεν σκοτώνονται με συνηθισμένους τρόπους, αλλά μόνο με τα προηγμένα όπλα του Johann ή με το ειδικό σφυρί του.

4.3 Σχεδιασμός παιχνιδιού

Το πρώτο βήμα για τον σχεδιασμό ενός τρισδιάστατου παιχνιδιού γραφικών είναι η μελέτη αντίστοιχων παιχνιδιών που ήδη κυκλοφορούν στο εμπόριο. Είναι σίγουρο ότι ένα άτομο δεν είναι σε θέση να παράγει ένα παιχνίδι με παρόμοια αποτελέσματα, όπως τα μεγάλα studios, όμως μέσα από αυτή τη διαδικασία μπορεί κανείς να εξάγει χρήσιμα συμπεράσματα και ιδέες, που μπορεί να χρησιμοποιήσει.

Αφού δημιουργήσουμε ένα επαρκές σενάριο, θα πρέπει να κατασκευάσουμε τις βασικές οντότητες και αντικείμενα που χρειάζονται για να προσδώσουν στο παιχνίδι την

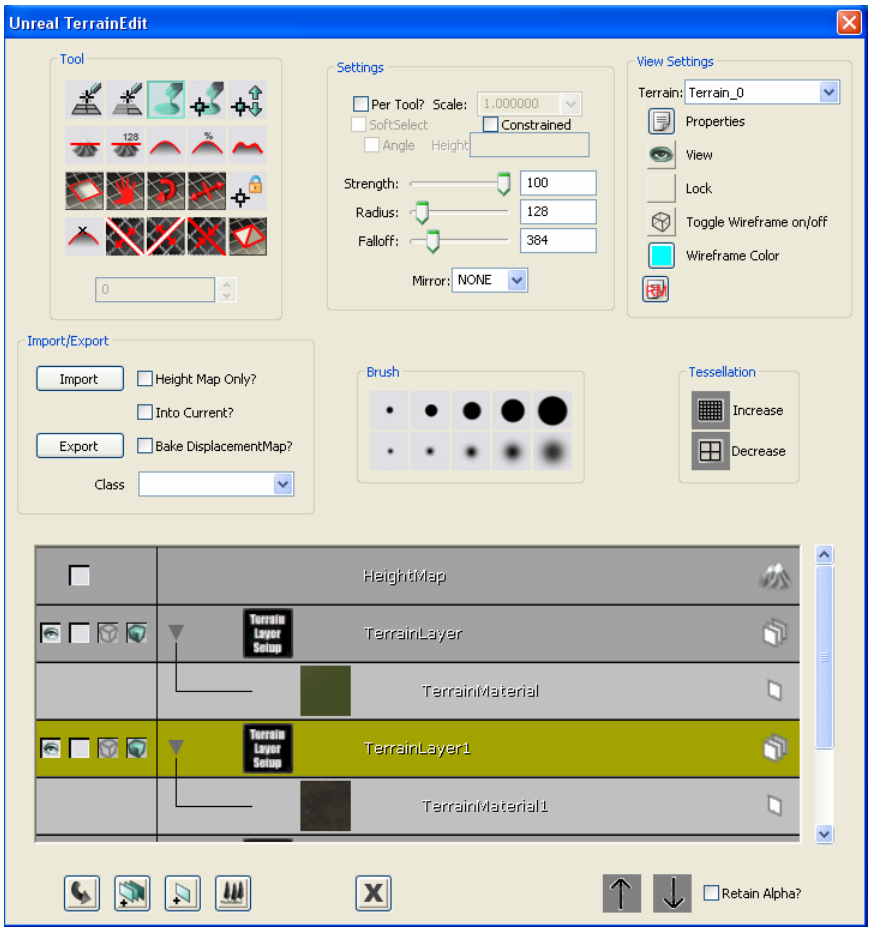
ανάλογη ατμόσφαιρα. Πρέπει να αναφέρουμε ότι η δημιουργία της πίστας, των χαρακτήρων και διάφορων ακόμα αντικειμένων δεν ήταν το βασικό αντικείμενο της εργασίας, παρ' όλα αυτά θεωρήσαμε πιο σωστό να χρησιμοποιήσουμε δικά μας ή να βρούμε στο διαδίκτυο έτοιμα, που να συμβαδίζουν με την ατμόσφαιρα που θέλαμε να δώσουμε στο παιχνίδι μας. Στην συνέχεια αυτής της ενότητας θα αναφέρουμε με έναν πιο τεχνικό τρόπο τα βήματα που ακολουθήσαμε για την δημιουργία της πίστας, των χαρακτήρων και της κίνησης τους και διαφόρων αντικειμένων που είναι απαραίτητα.

4.3.1 Σχεδιασμός πίστας

Σύμφωνα με το σενάριο του παιχνιδιού κατασκευάσαμε ένα τρισδιάστατο χώρο στον οποίον λαμβάνουν χώρα όλα τα γεγονότα [19]. Ο τρελός επιστήμονας της ιστορίας μας εργάζεται στο εργαστήριο του σε μια ερημική περιοχή. Η ερημική αυτή περιοχή είναι το δάσος και το εργαστήριο είναι χτισμένο μέσα στο βουνό. Για να είναι εύκολη η πλοήγηση του παίκτη πρέπει τα μονοπάτια να είναι φανερά, καθώς και τα όρια της πίστας. Επιπλέον, χωρίσαμε τον χώρο σε δύο μεγάλες νοητές περιοχές και στη μέση φτιάξαμε ένα ποτάμι, το οποίο ο χαρακτήρας δεν μπορεί να περάσει. Για να φτάσει στον τερματισμό της πίστας πρέπει να περάσει μέσα από το μονοπάτι και από εκεί στα αρχαία

Εικόνα 4.1: Πανοραμική άποψη της πίστας

ερείπια, αφού πρώτα ξεπεράσει την θανατηφόρα λίμνη. Για την δημιουργία του εδάφους χρησιμοποιήσαμε τον συντάκτη εδάφους (Terrain editor) της UDK. Με αυτό το εργαλείο ανυψώσαμε ορισμένες περιοχές της πίστας, δημιουργώντας βουνά και χαμηλώσαμε άλλες για το ποτάμι. Στη συνέχεια, προσθέσαμε υφές, ώστε τα υψηλότερα μέρη της πίστας να μοιάζουν ότι είναι φτιαγμένα από χώμα, και τα χαμηλότερα από γρασίδι. Επίσης ορίσαμε και διαφορετική υφή για το μονοπάτι.



Εικόνα 4.2: Ο συντάκτης εδάφους (Terrain editor) της UDK

Τέλος, για την ολοκλήρωση της πίστας τοποθετήσαμε συγκρουστές (colliders), που δεν επιτρέπουν στον χαρακτήρα να φτάσει σε σημεία του χώρου που δεν θέλουμε, πηγής ανέμου για την φυσική κίνηση των φύλλων των δέντρων, καθολικά φώτα (που προσομοιώνουν τον ήλιο), φώτα για τα δωμάτια του εργαστηρίου, καθώς και τόμων (volumes) που στόχο έχουν την οριοθέτηση χώρων με ειδικά χαρακτηριστικά. Τέτοιοι χώροι είναι για παράδειγμα η λίμνη, στην οποία όταν πέφτει ο χαρακτήρας θέλουμε να πεθαίνει. Αντίστοιχος τόμος βρίσκεται στο ποτάμι, όπου δημιουργείται εφφέ θολώματος του νερού, όταν η θέση του χαρακτήρα είναι χαμηλότερα από την επιφάνεια του νερού.

4.3.2 Χαρακτήρας και αντίπαλοι

Το σενάριο του παιχνιδιού κάνει λόγο για έναν τρελό επιστήμονα, τον Johann, ο οποίος κατασκευάζει περίεργα τερατάκια για να επανδρώσει τον στρατό του. Η UDK δίνει δωρεάν δύο, τρεις χαρακτήρες, που όμως δεν ταιριάζουν εμφανισιακά με την ατμόσφαιρα του παιχνιδιού μας. Για αυτό το λόγο, θεωρήσαμε σωστό να κατασκευάσουμε ένα χαρακτήρα που να μοιάζει με επιστήμονα. Επειδή όμως η δημιουργία ενός τρισδιάστατου μοντέλου δεν ήταν στα πλαίσια της παρούσης εργασίας αναζητήσαμε στο διαδίκτυο κάποιο μοντέλο με ανάλογη μορφή [20].

Εικόνα 4.3 Το τρισδιάστατο μοντέλο του Johann

Με το ειδικό πρόγραμμα 3ds Max [21], επεξεργαστήκαμε κατάλληλα το μοντέλο, μειώσαμε τα πολύγωνα για να γίνει λιγότερο απαιτητικό για το υπολογιστικό σύστημα που θα τρέχει το παιχνίδι, προσθέσαμε ένα σύστημα δίποδου και το εισάγαμε στην βιβλιοθήκη αντικειμένων της UDK. Στο μοντέλο αυτό προσθέσαμε τις κατάλληλες βασικές κινήσεις (animations) του τρεξίματος, άλματος και χτυπήματος με το ειδικό σφυρί. Η δημιουργία ενός αρχείου κίνησης είναι μια επίπονη εργασία που στόχο έχει να δώσει στο στατικό μοντέλο την αίσθηση της κίνησης. Για να συμβεί αυτό, πρέπει να ορίσουμε τις διαδοχικές καταστάσεις του μοντέλου κατά τη διάρκεια της κίνησης.

Εικόνα 4.4 Διαδοχικές φάσεις της κίνησης του χτυπήματος

Στην προηγούμενη εικόνα δίνουμε μια ιδέα για τις βασικές θέσεις που πρέπει να δημιουργηθούν ώστε να ολοκληρωθεί η κίνηση του χτυπήματος με το σφυρί. Ξεκινώντας από το πρώτο βήμα ορίζουμε το μοντέλο να έχει όρθια στάση με τα χέρια στην ανάταση και πηγαίνοντας στο τέταρτο μετατοπίζουμε το σώμα με τέτοιο τρόπο, ώστε να δίνουμε την εντύπωση της ανάλογης κίνησης. Φυσικά, δεν αρκούν μόνο τέσσερις διακριτές φάσεις για την ολοκλήρωση του σχεδίου κίνησης (animation), αλλά δουλεύοντας με τον ίδιο τρόπο και μεθοδικά δημιουργήσαμε τις απαραίτητες κινήσεις για το παιχνίδι.

Για τους αντιπάλους αναζητήσαμε στο διαδίκτυο το κατάλληλο μοντέλο το οποίο να περιλαμβάνει και τις ανάλογες κινήσεις. Στην ιστοσελίδα της UDK δίνεται για χρήση δωρεάν το παιχνίδι Dungeon Defense [22] μαζί με τον πηγαίο κώδικα και τη βιβλιοθήκη αντικειμένων. Χρησιμοποιήσαμε το μοντέλο του αντιπάλου που δίνεται, ως το μοντέλο στο δικό μας παιχνίδι, όπως αυτό φαίνεται στην επόμενη εικόνα. Ο αντίπαλος δεν διαθέτει περισσότερα από ένα όπλα, οπότε απλά προσθέσαμε μόνιμα το ρόπαλο της εικόνας και εισάγαμε τις κινήσεις του τρεξίματος, της επίθεσης, του άλματος και του θανάτου.

Εικόνα 4.5 Το τρισδιάστατο μοντέλο των αντιπάλων

4.3.3 Αντικείμενα και ήχοι

Για τον εμπλουτισμό της πίστας με αντικείμενα δημιουργήσαμε εκ του μηδενός δικά μας, ή βρήκαμε κάποια έτοιμα. Η κατασκευή ενός αντικειμένου με πολλές λεπτομέρειες είναι εξίσου δύσκολη με την κατασκευή ενός χαρακτήρα. Απαιτείται η σχεδίαση του αντικειμένου σε πρόγραμμα μοντελοποίησης και ύστερα η δημιουργία κατάλληλων υφών (textures) και υλικών (materials). Οι τεχνικές δημιουργίας των υφών και των υλικών αναλύονται σε διάφορα βιβλία και διαδικτυακά μαθήματα (tutorials) και δεν είναι αντικείμενο της εργασίας.

Για τον επιπρόσθετο εμπλουτισμό της ψυχαγωγίας του παίκτη έχουμε προσθέσει και ορισμένα ηχητικά εφφέ. Με αυτόν τον τρόπο επιτυγχάνουμε την καλύτερη κατανόηση των γεγονότων κατά τη διάρκεια του παιχνιδιού και της περιήγησης στα μενού. Τέτοια εφφέ χρησιμοποιήσαμε όταν ο αντίπαλος χτυπάει τον χαρακτήρα και όταν ο τελευταίος συλλέγει νομίσματα.

Εικόνα 4.6 Διάφορα αντικείμενα

4.3.4 Τερματισμός και βαθμολογία

Όταν ο παίκτης μαζέψει όλα τα απαιτούμενα νομίσματα και βρει τρόπο να φτάσει στο κρυφό δωμάτιο στην άλλη μεριά της πίστας, τότε το παιχνίδι τερματίζει. Για να τα καταφέρει, πρέπει να μείνει ζωντανός και να σκοτώσει όσο περισσότερους εχθρούς μπορεί, γιατί με αυτόν τον τρόπο μαζεύει επιπρόσθετους πόντους. Οι βαθμοί που μπορεί να συλλέξει ο παίκτης είναι οργανωμένοι ως εξής:

Αντίπαλος	50 πόντοι
Νόμισμα	100 πόντοι
Χρόνος Bonus	1 πόντος/sec

Η ελάχιστη βαθμολογία που μπορεί να συγκεντρώσει ένας παίκτης τερματίζοντας την πίστα είναι ίση με τον αριθμό των νομισμάτων που έχουμε εισάγει στην πίστα επί 100 πόντους που αξίζει κάθε νόμισμα. Για κάθε αντίπαλο παίρνουμε 50 πόντους και κάθε δευτερόλεπτο που έχει απομείνει στο χρονόμετρο μας δίνει επιπλέον 1 πόντο. Με αυτόν τον τρόπο αθροίζουμε τους πόντους και σε περίπτωση που η βαθμολογία μας αξίζει να καταγραφεί στην πρώτη δεκάδα, αποθηκεύεται σε αρχείο στον υπολογιστή.

4.4 Γραφική διεπαφή χρήστη

Στις επόμενες υπό-ενότητες θα περιγράψουμε την γραφική διεπαφή του χρήστη (GUI – Graphical User Interface), η οποία είναι υπεύθυνη για την ενημέρωση του σχετικά με οτιδήποτε συμβαίνει στο παιχνίδι.

4.4.1 Αρχικό μενού

Στην επόμενη εικόνα παρουσιάζεται η αρχική σελίδα της εφαρμογής. Σε αυτή, ο χρήστης μπορεί να επιλέξει να ξεκινήσει ένα νέο παιχνίδι (Play The Game), να δει τις μεγαλύτερες βαθμολογίες (View Highscores) ή να εγκαταλείψει την εφαρμογή (Exit).

Εικόνα 4.7 Αρχικό μενού

4.4.2 Κύριο γραφικό περιβάλλον

Όταν ο παίκτης ξεκινήσει το νέο παιχνίδι, στην οθόνη εμφανίζεται το λεγόμενο Heads-Up Display (HUD). Με τον όρο HUD αναφερόμαστε στην διεπαφή χρήστη (user interface) που παρέχει όλες εκείνες τις πληροφορίες που χρειάζεται ένας παίκτης για να παίξει ένα τρισδιάστατο παιχνίδι. Εκτός των άλλων, η διεπαφή εμφανίζει πληροφορίες σχετικά με τη ζωή του χαρακτήρα, τα αντικείμενα που διαθέτει και την πρόοδο του παιχνιδιού (π.χ. τη βαθμολογία ή/και το χρόνο).

Στο παιχνίδι Mad World το HUD περιέχει τις παρακάτω πληροφορίες:

Εικόνα 4.8 Heads-Up Display

Σταυρός: Στο κέντρο της οθόνης φαίνεται ένας σταυρός που είναι ο στόχος με τον οποίο ο χαρακτήρας επιτίθεται στους αντιπάλους.

Μετρητής νομισμάτων: Στην πάνω αριστερά περιοχή της οθόνης εμφανίζεται ένας κουμπαράς, στον οποίο αθροίζονται τα κέρματα που ο χαρακτήρας μαζεύει κατά τη διάρκεια της περιπλάνησης του στο χώρο.

Χρονόμετρο: Όπως εξηγήσαμε και νωρίτερα, στο παιχνίδι υπάρχει ένα χρονόμετρο των πέντε λεπτών, που δίνει επιπρόσθετη βαθμολογία στο τέλος του παιχνιδιού.

Βαθμολογία: Η τρέχουσα βαθμολογία του παίκτη εμφανίζεται στα δεξιά της οθόνης.

Όπλα: Ο χαρακτήρας μπορεί να διαθέτει ως δύο όπλα (σε αυτή την έκδοση του παιχνιδιού τα όπλα είναι ένα σφυρί και ένα μαγικό ραβδί). Στη οθόνη εμφανίζεται πληροφορία στα αριστερά σχετικά με το ποιο όπλο κρατά ο χαρακτήρας.

Υγεία: Σε σχήμα μπαταρίας εμφανίζεται το ποσοστό της υγείας που έχει ο χαρακτήρας ανά πάσα στιγμή. Η υγεία αυξάνεται λαμβάνοντας το κατάλληλο μαγικό φίλτρο.

Ενέργεια: Αντίστοιχα με την υγεία ο χρήστης όταν τρέχει καταναλώνει ενέργεια, που φαίνεται στο ανάλογο εικονίδιο. Όταν ο παίκτης σταματήσει να τρέχει, η ενέργεια αναπληρώνεται.

Αόρατος: Στην πίστα βρίσκονται διάσπαρτα κάποια μαγικά φίλτρα που κάνουν τον χαρακτήρα αόρατο για ορισμένα δευτερόλεπτα. Η πληροφορία αυτή φαίνεται στο ανάλογο εικονίδιο. Ο παίκτης δεν μπορεί να χρησιμοποιήσει το ίδιο φίλτρο άμεσα. Για αυτό, σε περίπτωση που προσπαθήσει να το χρησιμοποιήσει πολύ σύντομα, εμφανίζεται ανάλογο μήνυμα που τον ενημερώνει.

Ανίκητος: Η λογική είναι αντίστοιχη με την περίπτωση από πάνω, με μόνη διαφορά ότι ο χαρακτήρας γίνεται ανίκητος για ορισμένα δευτερόλεπτα.

Εικόνα 4.9 Σύστημα αποθήκευσης αντικειμένων

Καθ' όλη την διάρκεια του παιχνιδιού ο χρήστης μπορεί να επιλέξει το πλήκτρο "I" και να ανοίξει το σύστημα αποθήκευσης αντικειμένων (Εικ 4.9). Εκεί αποθηκεύονται όλα τα μαγικά φίλτρα για μελλοντική χρήση. Η σχεδίαση που έχουμε κάνει είναι τέτοια που δεν θα επιτρέπει στο χρήστη να διαθέτει δύο ίδια μαγικά φίλτρα την ίδια στιγμή.

Εικόνα 4.10 Γραφική διεπαφή για την ημιαυτόματη περιήγηση

Έχουμε προσθέσει στο παιχνίδι την δυνατότητα ο χρήστης να μπορεί να περιηγηθεί από το εργαστήριο του μέχρι τα ερείπια στην άλλη πλευρά της πίστας με ημιαυτόματο τρόπο (Εικ. 4.10). Ο τρόπος αυτός κάνει χρήση του αλγορίθμου τεχνητής νοημοσύνης A*, που αναλύεται θεωρητικά στο 2ο κεφάλαιο και χρησιμοποιείται από τους αντιπάλους κατά τις δικές τους περιηγήσεις. Στην συγκεκριμένη γραφική διεπαφή, ο χρήστης καλείται απλά να πατήσει πάνω στο κουμπί με την ένδειξη "X" και ο χαρακτήρας τρέχει προς το σημείο που τον προστάξαμε.

Τέλος, έχουμε προσθέσει ένα μενού παύσης κατά τη διάρκεια του παιχνιδιού. Ο χρήστης μπορεί να επιλέξει να ξεκινήσει το παιχνίδι από την αρχή, ή να εγκαταλείψει την εφαρμογή ανά πάσα στιγμή.

4.4.3 Μενού βαθμολογίας

Τερματίζοντας το παιχνίδι ο χρήστης είναι πιθανό να έχει μαζέψει μεγάλη βαθμολογία, η οποία είναι αρκετή για να εισαχθεί στις δέκα καλύτερες. Σε αυτή την περίπτωση καλείται να εισάγει το όνομα του στην κατάλληλη γραφική διεπαφή (Εικόνα 4.11). Τέλος, στη εικόνα 4.12 εμφανίζονται οι υψηλότερες βαθμολογίες που έχουν επιτευχθεί. Το μενού αυτό μπορεί να το δει ο χρήστης και κατά την έναρξη του παιχνιδιού.

Εικόνα 4.11 Εισαγωγή ονόματος

Εικόνα 4.12 Υψηλότερες βαθμολογίες

4.4.4 Μενού οδηγιών

Για να μπορεί οποιοσδήποτε παίκτης να χειριστεί το παιχνίδι, δημιουργήσαμε δυο οθόνες με σχετικές πληροφορίες. Στης πρώτη εμφανίζονται οι βασικές οδηγίες για τους σκοπούς του παιχνιδιού, ώστε ο χρήστης να πάρει μια ιδέα για το τι πρέπει να επιτύχει στο παιχνίδι. Στην δεύτερη γραφική διεπαφή εμφανίζονται όλες οι αναθέσεις των πλήκτρων σε ενέργειες. Τα μενού αυτά εμφανίζονται κατά τη διάρκεια του παιχνιδιού ακριβώς δίπλα στο σημείο εκκίνησης.

Εικόνα 4.13 Οδηγίες παιχνιδιού

Εικόνα 4.14 Χειρισμός παιχνιδιού

ΚΕΦΑΛΑΙΟ 5
ΥΛΟΠΟΙΗΣΗ ΠΑΙΧΝΙΔΙΟΥ

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναλύσουμε και θα παρουσιάσουμε ορισμένα μέρη της υλοποίησης του παιχνιδιού Mad World. Η Unrealscript, όπως έχουμε εξηγήσει, είναι μια αντικειμενοστραφής γλώσσα, όπου πολύ μεγάλο ρόλο παίζει η ιεραρχία των κλάσεων. Ο προγραμματιστής, αφού κατανοήσει την ιεραρχία αυτή καθώς και τις δυνατότητες που ήδη παρέχονται, καλείται να προσθέσει την λειτουργικότητα που τον ενδιαφέρει.

Η Unrealscript είναι δημιούργημα μιας μεγάλης ομάδας ανθρώπων και υπό κανονικές συνθήκες ένας νέος προγραμματιστής καλείται να μάθει τις δυνατότητες που του δίνονται σχετικά με το αντικείμενο για το οποίο προσελήφθηκε. Αυτό σημαίνει ότι ένας προγραμματιστής τεχνητής νοημοσύνης μαθαίνει κυρίως το κομμάτι της γλώσσας που έχει να κάνει με την τεχνητή νοημοσύνη. Αντίστοιχα, ένας προγραμματιστής φυσικής μαθαίνει κυρίως τις κλάσεις φυσικής και ούτω καθεξής. Στην παρούσα εργασία εργασθήκαμε στα περισσότερα τμήματα της γλώσσας και επιλέξαμε τις κλάσεις από τις οποίες θα κληρονομήσουμε τη λειτουργικότητα. Η Unrealscript παρουσιάζει τις μεγαλύτερες δυνατότητες στην ομάδα κλάσεων “UT”. Αυτές οι κλάσεις περιέχουν τη λειτουργικότητα που υπάρχει στο παιχνίδι “Unreal Tournament” της εταιρίας. Εμείς επιλέξαμε να εργαστούμε ένα επίπεδο παραπάνω, θέλοντας να ξεφύγουμε από τα χαρακτηριστικά αυτού του παιχνιδιού, να προσδώσουμε διαφορετικό ύφος και να παράγουμε κώδικα όχι τόσο σύνθετο.

5.2 Κάμερες

Το παιχνίδι της εργασίας ανήκει στα παιχνίδια βολών πρώτου-τρίτου προσώπου. Αυτό σημαίνει ότι ο παίκτης είτε θα βλέπει το παιχνίδι μέσα από τα μάτια του χαρακτήρα, είτε από ένα μακρύτερο σημείο πίσω από τον χαρακτήρα, ο οποίος σε αυτή την περίπτωση θα είναι ορατός. Στην δική μας υλοποίηση δίνουμε στον παίκτη την δυνατότητα να επιλέξει μεταξύ των καμερών χρησιμοποιώντας την ροδέλα του ποντικιού. Οι παρακάτω συναρτήσεις ενημερώνουν το σύστημα για το ποια κάμερα θέλει να επιλέξει:

```
//Zoom In
simulated exec function ZoomInCamera()
{
	ZoomCamPoint--;

	if(ZoomCamPoint<=0)
		ZoomCamPoint=0;
}

//Zoom Out
simulated exec function ZoomOutCamera()
{
	ZoomCamPoint++;
```

```
        if(ZoomCamPoint>=3)
                ZoomCamPoint=3;
}
```

Οι συναρτήσεις αυτές καλούνται όταν ο χρήστης μετακινήσει την ροδέλα του ποντικιού και ο σκοπός τους είναι να αλλάξουν την μεταβλητή “ZoomCamPoint”. Μέσω αυτής της μεταβλητής γίνεται η επιλογή της απόστασης στην οποία τοποθετούμε την κάμερα. Παρακάτω δίνουμε την υλοποίηση της μετακίνησης της κάμερας, ανάλογα με το ποια κάμερα είναι ενεργοποιημένη.

Εικόνα 5.1: Κάμερα πρώτου και τρίτου προσώπου

```
simulated function bool CalcCamera( float fDeltaTime, out vector
        out_CamLoc, out rotator out_CamRot, out float out_FOV )
{
        local MWPlayerController CPC;
        CPC=MWPlayerController(Owner);

        if(CPC!=none)
        {
                if(CPC.ZoomCamPoint==0) //Handle BehindView
                {
                        GetActorEyesViewPoint(out_CamLoc, out_CamRot);
                        RemoveMeshVisibility(true);
                }
                else
                {
                        CalcThirdPersonCamera(fDeltaTime, out_CamLoc,
                                out_CamRot, out_FOV, CPC.ZoomCamPoint);
                }
                return true;
        }
}
```

Η παραπάνω συνάρτηση διαβάζει την μεταβλητή “ZoomCamPoint” και ανάλογα με αυτή είτε επιστρέφει τις συντεταγμένες από το ύψος των ματιών του χαρακτήρα ενώ

ταυτόχρονα αποκρύπτει το μοντέλο, είτε καλεί την παρακάτω συνάρτηση που υπολογίζει την κάμερα του τρίτου προσώπου. Αρχικά, υπολογίζεται η απόσταση της κάμερας σύμφωνα με τις τιμές που έχουμε θέσει στις μεταβλητές "ThirdPersonCameraOffset". Στη συνέχεια κάνουμε έναν έλεγχο μεταξύ του χαρακτήρα και της κάμερας. Αν παρεμβάλλεται ανάμεσα τους κάποιος τοίχος, τότε επιστρέφουμε άμεσα σε κάμερα πρώτου προσώπου. Αυτό συμβαίνει όταν έχουμε κάποια κάμερα τρίτου προσώπου και ταυτόχρονα ο χαρακτήρας πλησιάσει κοντά έναν τοίχο ή ένα εμπόδιο. Τότε υπάρχει περίπτωση η κάμερα να διαπερνά το αντικείμενο κάνοντας την θέαση του παιχνιδιού αδύνατη.

```
simulated function bool CalcThirdPersonCamera( float fDeltaTime, out
      vector out_CamLoc, out rotator out_CamRot, out float out_FOV,
      optional int CamZoom )
{
      local vector CamLoc, HitLocation, HitNormal, CamX, CamY, CamZ;

      CamLoc=Location;
      GetAxes(out_CamRot, CamX, CamY, CamZ);

      switch(CamZoom)
      {
            case 1:
                  out_CamLoc = CamLoc - CamX*ThirdPersonCamOffset1.X +
                        ThirdPersonCamOffset1.Y*CamY +
                        ThirdPersonCamOffset1.Z*CamZ;
                  break;
            case 2:
                  out_CamLoc = CamLoc - CamX*ThirdPersonCamOffset2.X +
                        ThirdPersonCamOffset2.Y*CamY +
                        ThirdPersonCamOffset2.Z*CamZ;
                  break;
            case 3:
                  out_CamLoc = CamLoc - CamX*ThirdPersonCamOffset3.X +
                        ThirdPersonCamOffset3.Y*CamY +
                        ThirdPersonCamOffset3.Z*CamZ;
                  break;
            default:
                  break;
      }

      //Used to cause collision for camera
      if(Trace(HitLocation, HitNormal, out_CamLoc, CamLoc, false,
            vect(12,12,12))!=none)
      {
            //Go to first person view if cornered
            GetActorEyesViewPoint(out_CamLoc, out_CamRot);
            RemoveMeshVisibility(true);     //Set Mesh to Invisible
            return false;
      }
      else if(Mesh.bOwnerNoSee==true) //if true lets turn it back off
      {
            RemoveMeshVisibility(false);
      }
      return true;
}
```

5.3 Συμπεριφορά χαρακτήρα

Σε αυτή την ενότητα θα παρουσιάσουμε πώς υλοποιήθηκαν ορισμένες δυνατότητες του χαρακτήρα.

5.3.1 Ταχύτερο τρέξιμο

Ο χαρακτήρας έχει την δυνατότητα να τρέξει για ορισμένο χρόνο ταχύτερα από το κανονικό. Ο χρήστης όταν επιθυμεί να τρέξει γρηγορότερα ο χαρακτήρας, πατά το κατάλληλο πλήκτρο. Στη συνέχεια τρέχουν οι παρακάτω συναρτήσεις και ενεργοποιείται (ή αντίστοιχα απενεργοποιείται όταν δεν πατιέται το πλήκτρο) το χρονόμετρο (timer) “CheckSprinting”.

```
/When the Sprint Button is pressed
simulated exec function SprintOn()
{
	setTimer(0.05, true, 'CheckSprinting');
	bIsSprinting=true;
}

//When the Sprint Button is released
simulated exec function SprintOff()
{
	setTimer(0.05, true, 'CheckSprinting');
	bIsSprinting=false;
}
```

Η συνάρτηση καλείται σε πολύ τακτά χρονικά διαστήματα και μετράει σε μια μεταβλητή το χρόνο που περνά με πατημένο το πλήκτρο. Για όση ώρα είναι δυνατό, ο χαρακτήρας τρέχει ταχύτερα, ενώ αν τρέχει για μεγαλύτερο διάστημα από αυτό που έχουμε ορίσει, αυτόματα επιστρέφει στην αργή κατάσταση. Επιπρόσθετα, έχουμε υλοποιήσει μια λειτουργία επαναφοράς του χρονομέτρου παρόμοια με την φάση ξεκούρασης ενός ανθρώπου μετά από τρέξιμο. Σε αυτή τη φάση η ενέργεια αυξάνεται μέχρις ότου ο χαρακτήρας να «ξεκουραστεί» πλήρως.

```
simulated function CheckSprinting()
{
	//Sprinting
	if(bIsSprinting && TimeSprinting<=TimeAllowSprint && bCanSprint)
	{
		TimeSprinting++;
		Pawn.GroundSpeed=SprintSpeed;	//SprintSpeed

		//I sprinted too long, automatic turn off
		if(TimeSprinting > TimeAllowSprint)
		{
			bIsSprinting=false;
			Pawn.GroundSpeed=Pawn.default.GroundSpeed;
		}
	}
```

```
	//No longer sprinting
	if(!bIsSprinting && TimeSprinting>0)
	{
		TimeSprinting--;
		Pawn.GroundSpeed=Pawn.default.GroundSpeed;
	}

	//No longer sprinting and I have caught my breath, turn off timer
	if(!bIsSprinting && TimeSprinting==0)
	{
		ClearTimer('CheckSprinting');
	}
}
```

5.3.2 Σύστημα αποθήκευσης αντικειμένων

Η υλοποίηση του συστήματος αποθήκευσης αντικειμένων χωρίστηκε σε δυο τμήματα. Στο πρώτο τμήμα υλοποιήσαμε την εμφάνιση του συστήματος και στο δεύτερο τμήμα την επιλογή και χρήση ενός αντικειμένου. Ο παίκτης πατά το πλήκτρο “I” και εμφανίζεται μια σκηνή με τα αντικείμενα που έχει μαζέψει. Κατόπιν, επιλέγει κάποιο αντικείμενο και γίνεται χρήση αυτού. Στη συνέχεια παρουσιάζουμε τα βασικά μέρη της υλοποίησης, ξεκινώντας από την εμφάνιση της σκηνής. Στην συνάρτηση “CreateButtonScreen” πρώτα βρίσκουμε ποια αντικείμενα έχει ο χαρακτήρας στη διάθεση του και ύστερα καλούμε για κάθε ένα την συνάρτηση “AddButton” για να προστεθούν στην σκηνή.

```
//Creates the Screen of Buttons
function CreateButtonScreen()
{
	local int InvNum;
	local MWInventoryManager IMGR;
	local MWInventory_Potions MI;
	local array<MWInventory> InvAr;

	IMGR=MWInventoryManager(MWPC.Pawn.InvManager);

	if(ButtonScreen!=none)
	{
	     foreach IMGR.InventoryActors(class'MWInventory_Potions', MI)
	     {
	          InvAr.AddItem(MI);
	     }

	      //Creates an array of buttons based off of the
	            inventoryAllowed

	      for(InvNum=0; InvNum<MWPC.PlayerInfo.MaxInventoryAllowed;
	            InvNum++)
	      {
	            ButtonScreen.AddButton(InvNum, UseInventoryItem,
	                  MWPC.PlayerInfo.MaxInventoryAllowed, InvAr);
	      }
	}
}
```

Σε αυτή τη συνάρτηση περνάμε σαν παραμέτρους τα διάφορα αντικείμενα μέσω του πίνακα "InvTmp" και τα τοποθετούμε στην σκηνή ως εικονίδια. Το δεύτερο σημαντικότερο στοιχείο αυτής της συνάρτησης είναι η delegate συνάρτηση που δίνεται σαν όρισμα. Με αυτή τον τρόπο μπορούμε να καλέσουμε για κάθε αντικείμενο την συνάρτηση που θα κάνει χρήση του αντικειμένου. Στο κεφάλαιο τρία έχουμε κάνει μια αναφορά στις συναρτήσεις delegate, που είναι χαρακτηριστικό της Unrealscript.

```
//Adds the Button and its Information to the Screen
function AddButton(int InvNum, delegate<UIObject.OnClicked>
      BtnDelegate, int InvMax, array<MWInventory>InvTmp)
{
      MWButtons[InvNum]=new class'MadWorld.MWGame_Button'; //Creates
                                                            new Button
      self.InsertChild(MWButtons[InvNum]); //Place it on the screen
      MWButtons[InvNum].ArrayNum=InvNum;   //Set its location

      MWButtons[InvNum].SetPosition(0.1,UIFace_Left,
                                    EVALPOS_PercentageOwner);
      MWButtons[InvNum].SetPosition(0.1,UIFace_Right,
                                    EVALPOS_PercentageOwner);
      MWButtons[InvNUm].SetPosition(0,  UIFace_Top,
                                    EVALPOS_PercentageOwner);
      MWButtons[InvNum].SetPosition(0.1, UIFace_Bottom,
                                    EVALPOS_PercentageOwner);

      if(InvNum > 0)
            MWButtons[InvNum].SetDockParameters(UIFACE_LEFT,
            MWButtons[InvNum-1], UIFACE_Right, ButtonScreen_Spacing);
      else
            MWButtons[InvNum].SetDockParameters(UIFACE_LEFT, self,
            UIFACE_Left, 0);

      MWButtons[InvNum].SetImage(EmptyImage);
      if(InvNum<InvTmp.length && InvTmp[InvNum]!=none)
            MWButtons[InvNum].SetImage(InvTmp[InvNum].InventoryIcon);

    MWButtons[InvNum].SetEnabled(true);//Enable the button
    MWButtons[InvNum].OnClicked=BtnDelegate; //Sets delegate
    //Make sure it can't be changed once docked
    MWButtons[InvNum].DockTargets.bLockWidthWhenDocked = true;
    MWButtons[InvNum].DockTargets.bLockHeightWhenDocked = true;
}
```

Εικόνα 5.2: Επιλέγοντας αντικείμενο

Σε αυτό το σημείο ο χρήστης έχει πατήσει το πλήκτρο "I" και έχει εμφανιστεί η σκηνή του συστήματος μαζί με όσα αντικείμενα έχει στη διάθεση του. Στη συνέχεια μπορεί να επιλέξει ένα αντικείμενο για να το χρησιμοποιήσει. Για να υλοποιηθεί αυτή η δυνατότητα καλείται η παρακάτω συνάρτηση, η οποία πρώτα βρίσκει ποιο αντικείμενο επιλέχθηκε και ύστερα δίνει στον χαρακτήρα την επιπρόσθετη δυνατότητα.

```
function UseItem(Inventory Inv)
{
	local MWInventory_Potions MWIP;
	MWIP=MWInventory_Potions(Inv);

	if(MWIP!=none)
	{
		if(Inv.IsA('MWInventory_Potion_Green'))//Health Potion case
		{
			PlayerInfo.bHasPotion_Health=false;

			Pawn.Health+=30;
			if(Pawn.Health>100)
				Pawn.Health=100;	//Maximum health is 100
		}
		else if(Inv.IsA('MWInventory_Potion_Red'))  //Invisibility
		{
			if(bCanTurnInvisible)
			{
				PlayerInfo.bHasPotion_Invisibility=false;

				MWPawn(Pawn).bInvisible=true;
				bCanTurnInvisible=false;
				SetTimer(1.0, true, 'InvisibleTimerOn');
			}
			else
			{
				InvisibleWarningTimer=0;
				InvisibleWarningMoment=InvisibleTimer;
				bInvisibleWarning=true;
				SetTimer(1.0, true, 'InvisibleWarningTimerOn');
				return;
			}
		}
		else if(Inv.IsA('MWInventory_Potion_Blue')) //Invincibility
			//Like Invisibility. Deleted for space reasons.

		RemoveItemUsed(Inv);
	}
}
```

Πιο συγκεκριμένα, αν ο χρήστης επιλέξει το κόκκινο φίλτρο, που κάνει τον παίκτη αόρατο, τότε η συνάρτηση αλλάζει κάποιες μεταβλητές και ενεργοποιεί το χρονόμετρο, που μετρά πόση ώρα ο χαρακτήρας θα βρίσκεται σε αυτή την κατάσταση. Μόλις γίνει πάλι ορατός, υπάρχει η λειτουργία που δεν επιτρέπει άμεσα τη χρήση αυτού του φίλτρου για $2^{η}$ φορά και μας ενημερώνει ότι πρέπει να περιμένουμε λίγο ακόμα. Για λόγους χώρου, έχουμε αποκρύψει τα λιγότερο ενδιαφέροντα κομμάτια κώδικα της υλοποίησης.

5.3.3 Όπλα

Σημαντικό κομμάτι του παιχνιδιού είναι τα όπλα που δίνουν την δυνατότητα στον παίκτη να εξοντώνει τους αντιπάλους του. Η Unrealscript δίνει αρκετή λειτουργικότητα για τα όπλα, εκτός από την περίπτωση όπου θέλουμε το όπλο να έχει τελείως διαφορετική συμπεριφορά. Οι προγραμματιστές της μηχανής έχουν ορίσει τρεις διαφορετικούς τύπους επίθεσης. Αυτοί είναι η επίθεση από κοντινή απόσταση (InstantHit), η επίθεση με όπλο που ρίχνει κάποιο είδος βλήματος (Projectile) και ένας ειδικός τύπος που προγραμματίζεται από εμάς, σε περίπτωση που οι άλλοι δύο τύποι δεν μας καλύπτουν (Custom). Στο παιχνίδι έχουμε προσθέσει δυο όπλα με σκοπό να δώσουμε περισσότερο ενδιαφέρον και να προσθέσουμε λειτουργικότητα.

Αρχικά θα δείξουμε πώς έχουμε υλοποιήσει τη σύνδεση ενός οποιουδήποτε όπλου με το χέρι του χαρακτήρα. Για να το επιτύχουμε αυτό καλούμε την παρακάτω συνάρτηση στην οποία δίνουμε σαν ορίσματα το μοντέλο στο οποίο θέλουμε να συνδέσουμε το όπλο (δηλαδή στην περίπτωση μας το χαρακτήρα) και το όνομα του socket που θα γίνει η σύνδεση. Στο κεφάλαιο τρία έχουμε κάνει μια αναφορά στα sockets που στόχο έχουν να συνδέουν ένα αντικείμενο σε ένα άλλο μοντέλο.

```
simulated function AttachWeaponTo(SkeletalMeshComponent MeshCpnt,
      optional Name SocketName)
{
      local MWPawn P;
      P = MWPawn(Instigator);

      //Share light environment
      Mesh.SetLightEnvironment(P.LightEnvironment);
      //And the shadows
      Mesh.SetShadowParent(P.Mesh);

      if(SocketName != 'none')
      {
            //Base our actor on the pawn
            SetBase(P, , P.Mesh, SocketName);
            //Attach our skeletalmesh component to our pawn's socket
            P.Mesh.AttachComponentToSocket(Mesh, SocketName);
      }
      else
            SetBase(P);   //Or just base on it

      SetHidden(false);   //Unhide the weapon when active

      //Set the right animation to the animTree
      if(P.Weapon.Class==class'MWWeaponHammer')
            P.SetWeaponAnimType(MWT_Hammer);
      else if(P.Weapon.Class==class'MWWeaponMageStaff')
            P.SetWeaponAnimType(MWT_MageStaff);
}
```

Στην συνάρτηση “AttachWeaponTo” αναθέτουμε στο socket του χαρακτήρα το όπλο και μοιράζουμε το φωτισμό και τις σκιές. Στο τελευταίο μέρος ορίζουμε ποιο animation να χρησιμοποιείται, ανάλογα με το όπλο που κρατά ο χαρακτήρας.

Ο χρήστης όταν επιθυμεί να κάνει επίθεση πατά το αριστερό κουμπί του ποντικιού, οπότε και ξεκινά η επιθετική κίνηση του χαρακτήρα. Ταυτόχρονα με την κίνηση αρχίζει να ενημερώνεται η μηχανή ότι ο χαρακτήρας επιτίθεται και αν ο στόχος βρίσκεται πάνω σε κάποιον αντίπαλο, τότε καλείται η συνάρτηση που προκαλεί μείωση της ζωής του αντιπάλου. Η συνάρτηση αυτή καλείται αυτόματα και είναι η παρακάτω:

```
event TakeDamage(int Damage, Controller InstigatedBy, vector
      HitLocation, vector Momentum, class<DamageType> DamageType,
      optional TraceHitInfo HitInfo, optional Actor DamageCauser)
{
      local MWPlayerController MWPC;
      local MWBotPlayers MWBP;

      MWPC=MWPlayerController(Controller);
      MWBP=MWBotPlayers(Controller);

      if(MWPC==none)     //Damage enemies
      {
            MWPawn(MWBP.Pawn).bHurt=true;
            SetTimer(0.5, true, 'HurtTimer');

            Damage=20;
            super.TakeDamage(Damage, InstigatedBy, HitLocation,
                  Momentum, DamageType, HitInfo, DamageCauser);
      }
}
```

Στην συνάρτηση έχουμε προσθέσει και ένα χρονόμετρο που σκοπό έχει να μην μπορεί ο αντίπαλος να κάνει ζημιά στο χαρακτήρα για ένα μικρό χρονικό διάστημα κατά το οποίο δέχεται ο ίδιος επίθεση. Αυτό είναι λογικό αν σκεφτούμε ότι όταν δεχόμαστε επίθεση είναι τουλάχιστον πάρα πολύ δύσκολο να απαντήσουμε τη στιγμή της επίθεσης ή για λίγο ακόμα.

Εικόνα 5.3: Επίθεση με το σφυρί

Τέλος, για αυτό το όπλο δεν χρειάστηκε να γράψουμε κάποιο πολύπλοκο κώδικα εκτός από την αρχικοποίηση κάποιων μεταβλητών, ώστε να γνωρίζει η μηχανή διάφορες πληροφορίες για το συγκεκριμένο όπλο. Τέτοιες πληροφορίες είναι για παράδειγμα, η μέγιστη απόσταση από την οποία μπορεί να βλάψει ο χαρακτήρας τον αντίπαλο, η περίοδος της επίθεσης (κάθε πόσο προκαλείται ένα νέο χτύπημα) και το είδος του όπλου.

Στη συνέχεια, θεωρήσαμε σωστό να προσθέσουμε ένα πιο ιδιαίτερο όπλο, που προκαλεί χτυπήματα από μεγαλύτερη απόσταση. Το όπλο αυτό είναι ένα μαγικό ραβδί, που εκτοξεύει κάποια παράξενα βλήματα. Η λειτουργία αυτού του όπλου είναι πολύ διαφορετική από το σφυρί. Ο χρήστης μπορεί να πατήσει το αριστερό πλήκτρο του ποντικιού και για όση ώρα είναι πατημένο, αυξάνεται η ενέργεια του όπλου. Όταν το πλήκτρο αφεθεί, τότε εκτοξεύεται ένα βλήμα ανάλογου μεγέθους με την ενέργεια που έχει συγκεντρωθεί. Αυτό δίνει μεγαλύτερες πιθανότητες επιτυχίας στον παίκτη να επιτύχει τον αντίπαλο. Στην επόμενη εικόνα φαίνεται στον τοίχο το αποτέλεσμα την επίθεσης με το μαγικό ραβδί.

Εικόνα 5.4: Αποτέλεσμα της επίθεσης με το μαγικό ραβδί

Προγραμματιστικά, το όπλο αυτό μπορεί να βρίσκεται σε δύο καταστάσεις, όση ώρα το κρατάει ο χαρακτήρας. Η πρώτη κατάσταση είναι η «Ενεργή» κατάσταση και βρισκόμαστε σε αυτή όσο ο χρήστης δεν επιτίθεται και απλά κρατά το όπλο. Μόλις ο χρήστης πατήσει το πλήκτρο της επίθεσης, το όπλο περνά στην κατάσταση «Βολή όπλου». Σε αυτή την κατάσταση, όπως προαναφέραμε, βρισκόμαστε καθ' όλη τη διάρκεια που το πλήκτρο της επίθεσης είναι πατημένο και το όπλο φορτίζει με ενέργεια. Μόλις ο χρήστης αφήσει το πλήκτρο, εκτοξεύεται το βλήμα και επιστρέφουμε στην «Ενεργή» κατάσταση. Στην συνέχεια παραθέτουμε τον κώδικα της κατάστασης «Βολή όπλου».

```
simulated state WeaponFiring
{
	simulated event BeginState( Name PreviousStateName )
	{
		TimeWeaponFiring( CurrentFireMode );

		//initialize our charge effects
		StoredChargePercent=0;
		ChargeEffect.SetScale(MinChargeEffectScale);
		ChargeEffect.ActivateSystem();
		ChargeLight.SetLightProperties(0);
		ChargeLight.SetEnabled(true);
		MyAudioComponent1.FadeIn(1,1);
	}

	simulated function RefireCheckTimer()
	{
		//if switching to another weapon, abort firing and put
		//down right away
		if( bWeaponPutDown )
		{
			PutDownWeapon();
			return;
		}
		return;
	}

	event Tick(float DeltaTime)
	{
		super.Tick(DeltaTime);

		//increment our charge value and effects every frame while
		//holding down the fire button
		if(StoredChargePercent < 1)
		{
			StoredChargePercent = FMin(StoredChargePercent +
                                   (DeltaTime/FullChargeTime),1);

			ChargeEffect.SetScale(MinChargeEffectScale +
			(MaxChargeEffectScale-MinChargeEffectScale) *
			StoredChargePercent);

			ChargeLight.SetLightProperties
			  (ChargeLightOriginalBrightness*StoredChargePercent);

			if(StoredChargePercent == 1)
				ReachedFullCharge();
		}
	}

	/** Activates full-charge effects */
	function ReachedFullCharge()
	{
		MyAudioComponent2.FadeIn(1,1);
		MyAudioComponent1.FadeOut(1,0);
		FullyChargedEffect.ActivateSystem();
	}
```

```
	/** Released weapon fire */
	simulated event EndState( Name NextStateName )
	{
		//shoot it!
		FireAmmunition();
		StoredChargePercent=0;
		LastReleasedTime=WorldInfo.TimeSeconds;

		//stop all the charge effects
		if(MyAudioComponent1.IsPlaying())
			MyAudioComponent1.FadeOut(0.5,0);

		if(MyAudioComponent2.IsPlaying())
			MyAudioComponent2.FadeOut(0.5,0);

		ChargeEffect.DeactivateSystem();
		ChargeLight.SetEnabled(false);
		FullyChargedEffect.DeactivateSystem();

		super.EndState(NextStateName);
	}
}
```

Το πιο ενδιαφέρον μέρος της κατάστασης «Βολή όπλου» είναι η συνάρτηση “Tick”. Αυτή καλείται σε τακτά χρονικά διαστήματα και αυξάνει το μέγεθος του βλήματος που θα εκτοξευθεί. Μόλις ο χρήστης αφήσει το πλήκτρο τότε καλείται η συνάρτηση “EndState”, που εκτοξεύει το βλήμα. Επί της ουσίας, καλείται μια συνάρτηση που υπολογίζει την αρχή και το τέλος του διανύσματος που θα κολουθήσει το βλήμα, και κατόπιν το δημιουργεί θέτοντας το μέγεθος που πρέπει να έχει και το στόχο που πρέπει να πετύχει.

5.4 Συμπεριφορά αντιπάλων

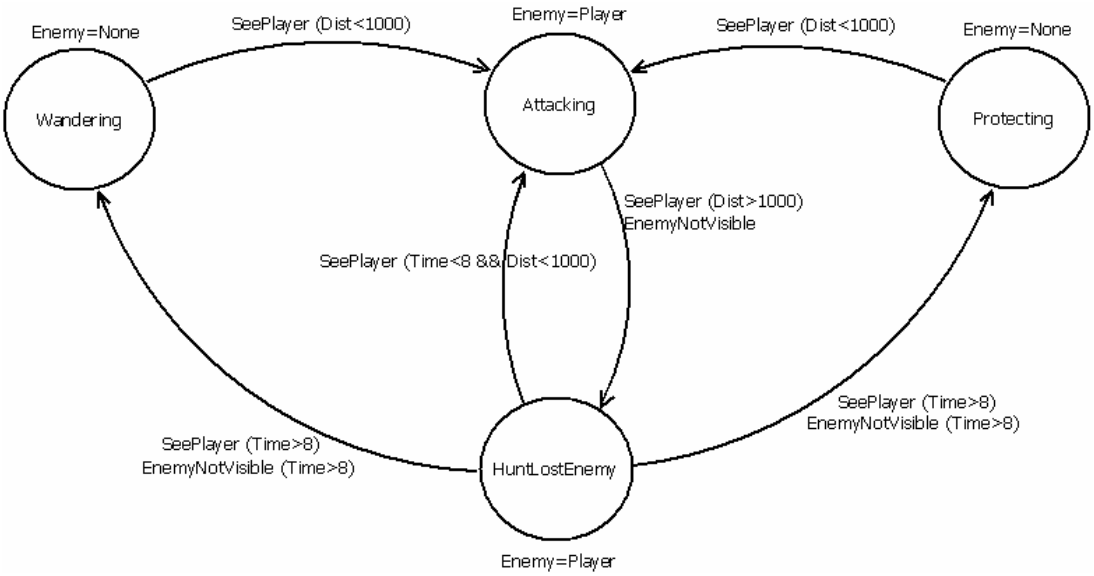
Στη προηγούμενη ενότητα ασχοληθήκαμε με την συμπεριφορά του χαρακτήρα που χειρίζεται ο παίκτης. Σε αυτή την ενότητα θα ασχοληθούμε με την συμπεριφορά των αντιπάλων που υπάρχουν στο παιχνίδι “Mad World”. Οι αντίπαλοι που δημιουργήσαμε είτε περιφέρονται στην πίστα με τρόπο που θα εξηγήσουμε σε επόμενη ενότητα (τύπος Α), ή περιμένουν μόνιμα σε ένα σημείο να βρουν τον χαρακτήρα και να του επιτεθούν (τύπος Β).

Έχουμε οργανώσει τις συμπεριφορές των αντιπάλων σε καταστάσεις (states) και με αυτό τον τρόπο ανάλογα με τα ερεθίσματα που δέχονται, συμπεριφέρονται κατάλληλα. Όταν δημιουργείται ένας νέος αντίπαλος (έστω τύπου Α), πηγαίνει αυτόματα στην κατάσταση «Περιήγηση». Σε αυτή την κατάσταση κάνει χρήση του αλγορίθμου τεχνητής νοημοσύνης που θα περιγράψουμε παρακάτω για να κινείται στην πίστα και περιμένει μέχρι να βρεθεί στον οπτικό του πεδίο ο χαρακτήρας για να τον κυνηγήσει. Η πίστα είναι ένας μεγάλος τρισδιάστατος χώρος, όπου έχουμε ορίσει χειροκίνητα περίπου 100 κόμβους και σε αυτή συνυπάρχουν ταυτόχρονα αρκετοί αντίπαλοι. Όμως, επειδή δεν θέλουμε να περιηγείται ο καθένας από αυτούς σε ολόκληρο τον χώρο, έχουμε ομαδοποιήσει τους κόμβους που μπορούν να περιηγηθούν σε μικρότερες ομάδες. Αυτό

το έχουμε επιτύχει θέτοντας μια μεταβλητή με έναν αριθμό (τον αριθμό της ομάδας) στον κάθε κόμβο. Στη συνέχεια, έχουμε ορίσει στον αλγόριθμο να αναζητά κόμβους της ομάδας στην οποία ανήκει ο αντίπαλος και βρίσκει τη διαδρομή με το μικρότερο κόστος. Η συνάρτηση που καλείται όταν ο αντίπαλος εντοπίσει στο πεδίο του τον χαρακτήρα είναι η εξής:

```
function SeePlayer(Pawn SeenPlayer)
{
	super.SeePlayer(SeenPlayer);

	if(VSize(SeenPlayer.Location-Pawn.Location)<SeenPlayerDistance)
	{
		//If pawn is INVISIBLE, bots can't attack him
		if(MWPawn(SeenPlayer).bInvisible==true)
			return;

		Enemy=SeenPlayer;	//Enemy spoted
		GoToState('Attacking');
	}
}
```

Σε αυτή υπάρχουν δύο έλεγχοι: πρώτα υπολογίζεται η απόσταση του αντιπάλου από τον χαρακτήρα και αν είναι μικρότερη από την απόσταση που έχουμε ορίσει τότε προχωρούμε στο δεύτερο έλεγχο. Αν ο χαρακτήρας έχει πάρει το μαγικό φίλτρο που τον κάνει αόρατο, τότε δεν περνάμε στην φάση της επίθεσης και επιστρέφουμε από την συνάρτηση. Σε αντίθετη περίπτωση ο αντίπαλος αντιλαμβάνεται τον χαρακτήρα σαν τον εχθρό του και περνά σε κατάσταση επίθεσης. Στην εικόνα 5.5 φαίνετα διαγραμματικά ο τρόπος που γίνεται μεταγωγή των καταστάσεων ανάλογα με τον τύπο του αντιπάλου και των ερεθισμάτων που δέχεται.



Εικόνα 5.5: Μεταγωγές καταστάσεων αντιπάλων

Οι αντίπαλοι και των δύο τύπων, εφόσον εντοπίσουν τον χαρακτήρα, περνούν σε κατάσταση επίθεσης. Σε αυτή την κατάσταση ο αντίπαλος ακολουθεί κατά πόδας τον χαρακτήρα και του επιτίθεται με το ρόπαλο που διαθέτει. Έχουμε προσθέσει έναν έλεγχο, ο οποίος σταματά τον αντίπαλο σε μια ορισμένη απόσταση από τον χαρακτήρα, όση χρειάζεται για να φτάνει με το ρόπαλο, αλλά όχι μακρύτερα. Έτσι, όταν βρίσκεται κοντά προκαλεί μείωση της ζωής του χαρακτήρα και ταυτόχρονα εκτέλεση ενός εφφέ χτυπήματος μέσω της συνάρτησης:

```
event TakeDamage(int Damage, Controller InstigatedBy, vector
      HitLocation, vector Momentum, class<DamageType> DamageType,
      optional TraceHitInfo HitInfo, optional Actor DamageCauser)
{
      local MWPlayerController MWPC;
      local MWBotPlayers MWBP;
      local MWBotPlayers MWBP_By;

      MWPC=MWPlayerController(Controller);
      MWBP=MWBotPlayers(Controller);
      MWBP_By=MWBotPlayers(InstigatedBy);

      if(MWPC!=none)    //Damage Player
      {
      //If player is invincible or the enemy is hurting don't do damage
            if(!bInvincible && !MWPawn(MWBP_By.Pawn).bHurt)
            {
                  Damage=15;
                  PlayTakeHitEffects();
                  super.TakeDamage(Damage, InstigatedBy, HitLocation,
                        Momentum, DamageType, HitInfo, DamageCauser);
            }
      }
}
```

Αν σε κάποια στιγμή ο αντίπαλος χάσει από το οπτικό του πεδίο τον χαρακτήρα, τότε σταματά την επίθεση και περνά στην κατάσταση «Εξαφάνιση χαρακτήρα». Μόλις εισέλθει σε αυτή την κατάσταση πηγαίνει στην τελευταία γνωστή θέση του χαρακτήρα και σταματά περιμένοντας για ένα μικρό χρονικό διάστημα, μήπως εμφανιστεί εκ νέου. Όμως, επειδή ο αντίπαλος δεν έχει και ιδιαίτερα μεγάλη νοημοσύνη, μετά από αυτό το διάστημα απλά επιστρέφει στην περιοχή που βρισκόταν πριν δει τον χαρακτήρα, εκτός και εάν τον εντοπίσει, οπότε του επιτίθεται.

Νωρίτερα εξηγήσαμε πως έχουμε γεμίσει τον χώρο με κόμβους και τους έχουμε ομαδοποιήσει. Αυτό μας δίνει την ευελιξία να μπορούμε να προγραμματίσουμε τον αντίπαλο να επιστρέφει στην αρχική περιοχή από οποιαδήποτε θέση και αν βρισκόταν τη στιγμή που έχασε τον χαρακτήρα από το οπτικό του πεδίο. Μόλις ο αντίπαλος επιστρέψει στην αρχική περιοχή αρχίζει να περιηγείται ή μένει σταθερός, ανάλογα με τον τύπο στον οποίο ανήκει.

Ως αυτό το σημείο περιγράψαμε όλη την συμπεριφορά των αντιπάλων, εκτός από τον τρόπο που μετακινούνται μέσα στην πίστα, όταν δεν κυνηγούν τον χαρακτήρα. Ο αλγόριθμος της τεχνητής νοημοσύνης θα περιγραφεί σε επόμενη ενότητα αυτού του κεφαλαίου.

5.5 Heads-Up Display (HUD)

Στην εικόνα 4.8 φαίνεται η γραφική διεπαφή του χρήστη για το παιχνίδι “Mad World”. Για να δημιουργηθεί αυτή η διεπαφή καταρχήν δημιουργήσαμε την γραμματοσειρά που μας ενδιέφερε και σε πρόγραμμα επεξεργασίας εικόνων τις κατάλληλες εικόνες, τις οποίες εισάγαμε στη βιβλιοθήκη της μηχανής για χρήση. Η διεπαφή χωρίζεται σε δυο βασικές υποπεριπτώσεις. Η πρώτη είναι ο χαρακτήρας να είναι ζωντανός, οπότε εμφανίζονται όλα αυτά τα στοιχεία που δίνουν στον χρήστη πληροφόρηση για τα όπλα, τη ζωή, την ενέργεια, κτλ. Η δεύτερη περίπτωση είναι ο χαρακτήρας να μην είναι ζωντανός, πράγμα που σημαίνει ότι είτε τώρα ξεκινά το παιχνίδι, είτε τελείωσε. Σε αυτή την ενότητα θα δείξουμε κάποια μέρη της υλοποίησης της γραφικής διεπαφής, ενόσω ο χαρακτήρας είναι ζωντανός. Ξεκινώντας, παραθέτουμε τον κώδικα που εμφανίζει το χρονόμετρο που βρίσκεται στο πάνω μέρος της οθόνης.

```
function DrawBonusTime()
{
  local string TimeLeftSeconds;
  local string TimeLeftMinutes;
  local string Temp;

  Canvas.SetPos(BonusPos.X, BonusPos.Y);
  TimeLeftSeconds=string(MWGame(WorldInfo.Game).RemainingBonusSeconds);
  if(MWGame(WorldInfo.Game).RemainingBonusSeconds<10)
  {
       Temp="0"$TimeLeftSeconds;
       TimeLeftSeconds=Temp;
  }
  TimeLeftMinutes=string(MWGame(WorldInfo.Game).RemainingBonusMinutes);
  Canvas.DrawTextCentered("Bonus Time
       Remainings:"$TimeLeftMinutes$":"$TimeLeftSeconds, , 0.75, 0.75);
}
```

Αρχικά, ορίζουμε το μέρος της οθόνης στο οποίο θα εμφανίζεται το χρονόμετρο. Ύστερα, μετασχηματίζουμε τους ακέραιους αριθμούς σε σειρές γραμμάτων και στην ειδική περίπτωση που τα δευτερόλεπτα είναι μονοψήφια, προσθέτουμε τον χαρακτήρα “0”. Τέλος, σχεδιάζουμε στην οθόνη το αποτέλεσμα, το οποίο ανανεώνεται σε τακτά χρονικά διαστήματα, ώστε να μοιάζει με πραγματικό χρονόμετρο.

Ένα δεύτερο στοιχείο της διεπαφής είναι ο σταυρός στο κέντρο της οθόνης, που χρησιμοποιείται για την στόχευση των αντιπάλων. Ο τρόπος που είναι υλοποιημένος είναι ο εξής:

```
function DisplayCrosshair()
{
      local float CrosshairSize;
      CrosshairSize = 14;
      Canvas.SetDrawColor(255,0,0,255);
      Canvas.SetPos(CenterX - CrosshairSize, CenterY);
      Canvas.DrawRect(2*CrosshairSize + 1, 1);
      Canvas.SetPos(CenterX, CenterY - CrosshairSize);
      Canvas.DrawRect(1, 2*CrosshairSize + 1);
}
```

Η λογική είναι να σχεδιάσουμε δυο κάθετα μεταξύ τους ορθογώνια, που το καθένα έχει κάποιο μήκος και ένα ελάχιστο πλάτος. Αφού επιλέξουμε το χρώμα και το μήκος των ορθογωνίων, τα σχεδιάζουμε χρησιμοποιώντας τις κατάλληλες συναρτήσεις που δίνονται έτοιμες στην κλάση "Canvas".

Μέχρι αυτό το σημείο έχουμε δείξει τον τρόπο με τον οποίο εμφανίζουμε είτε κάποια μεταβλητή του παιχνιδιού, όπως είναι ο χρόνος, είτε πώς σχεδιάζουμε ένα σχήμα στην οθόνη. Ένα ακόμα βασικό στοιχείο που εμφανίζουμε στην διεπαφή είναι οι εικόνες. Ο χαρακτήρας μπορεί να έχει στη κατοχή του κάποιο ή κάποια όπλα, ή να μην έχει καθόλου. Με τον παρακάτω κώδικα ενημερώνουμε τον χρήστη για το όπλο που μπορεί να έχει ο χαρακτήρας στην κατοχή του.

```
function DisplayWeaponSlot()
{
	Canvas.DrawColor=MWWhiteColor;
	Canvas.SetPos(WeaponSlotLoc.X, WeaponSlotLoc.Y);

	if(MWPawnPlayer.Weapon==none)
		Canvas.DrawTexture(WeaponSlot1, 0.3);
	else
	{
		if(MWPawnPlayer.Weapon.Class==class'MWWeaponHammer')
			Canvas.DrawTexture(WeaponSlot2, 0.3);
		else if(MWPawnPlayer.Weapon.Class==class'MWWeaponMageStaff')
			Canvas.DrawTexture(WeaponSlot3, 0.3);
	}
}
```

Στην συγκεκριμένη συνάρτηση ορίζουμε το σημείο στο οποίο θέλουμε να εμφανιστούν οι εικόνες, και κατόπιν ελέγχουμε αν ο χαρακτήρας έχει στην κατοχή του κάποιο όπλο. Αν δεν έχει, τότε εμφανίζουμε την εικόνα 1, η οποία δείχνει ένα άδειο «οπλοστάσιο». Σε περίπτωση που έχει κάποιο όπλο, τότε γίνεται ένας έλεγχος για το είδος του όπλου, και εμφανίζεται η κατάλληλη εικόνα.

Όλες οι υπόλοιπες πληροφορίες που φαίνονται στην γραφική διεπαφή σχεδιάζονται με τον ίδιο ή κάποιο παρόμοιο τρόπο.

5.6 Βαθμολογία

Στο παιχνίδι έχουμε προσθέσει την λειτουργία της αποθήκευσης των μεγαλύτερων βαθμολογιών. Μόλις ο παίκτης τερματίσει το παιχνίδι, καλείται να εισάγει το όνομα που επιθυμεί να φαίνεται στον πίνακα με τις υψηλότερες βαθμολογίες. Έχουμε δημιουργήσει μια δομή (αντίστοιχα με τις δομές στη C) που κρατά το όνομα του παίκτη και τη βαθμολογία του. Η δομή είναι:

```
struct HighScoresEntry
{
	var string PlayerName;
	var int Score;
};
```

Επίσης, για την αποθήκευση των στοιχείων έχουμε δημιουργήσει ένα πίνακα που δέχεται τέτοιες δομές. Κάθε φορά που τερματίζουμε το παιχνίδι καλείται η συνάρτηση που ελέγχει αν η νέα βαθμολογία είναι ικανή να βρίσκεται στον πίνακα και ταξινομεί κατάλληλα τις εγγραφές. Λόγω του περιορισμένου αριθμού των εγγραφών θεωρήσαμε σωστό να μην βελτιστοποιήσουμε τον αλγόριθμο ταξινόμησης και διατηρήσαμε τον αλγόριθμο γραμμικού χρόνου που φαίνεται παρακάτω:

```
function AddHighScore(HighScoresEntry entry)
{
   local int ctr;
   local int i;
   local HighScoresEntry temp;

   if(HighScoresEntries.Length==0)
   {
      for(i=0;i<10;i++)
            HighScoresEntries.InsertItem(i, temp);
   }

   if(entry.Score >= HighScoresEntries[9].Score)
   {
      HighScoresEntries[9]=entry;

      for(ctr=9;ctr>0;ctr--)
      {
            if (HighScoresEntries[ctr].Score >=
                  HighScoresEntries[ctr-1].Score)
            {
                  temp=HighScoresEntries[ctr];
                  HighScoresEntries[ctr]=HighScoresEntries[ctr-1];
                  HighScoresEntries[ctr-1]=temp;
            }
      }
   }

   SaveConfig();
}
```

5.7 Γραφική διεπαφή χρήστη

Για να δημιουργηθούν οι διεπαφές πρέπει να συμβούν δυο πράγματα. Πρώτον, να κατασκευάσουμε τις απαραίτητες κλάσεις της Unrealscript και δεύτερον να σχεδιάσουμε τις διεπαφές χρησιμοποιώντας ως βάση τις αντίστοιχες κλάσεις και ως εργαλείο τον συντάκτη (editor) της UDK.

Ο συντάκτης είναι ένα εργαλείο που δίνει την δυνατότητα να σχεδιάσουμε την διεπαφή από το μηδέν. Μπορούμε να προσθέσουμε εικόνες, κουμπιά, μενού, κείμενο και ό,τι άλλο επιθυμούμε, για να κάνουμε την σκηνή ενδιαφέρουσα και εύκολη στη χρήση. Η αλληλεπίδραση της σκηνής με τον χρήστη γίνεται μέσω της Unrealscript. Αυτό σημαίνει ότι για να μετατραπεί το πάτημα ενός κουμπιού σε εντολή, πρέπει να συνδέσουμε μέσω κώδικα το κουμπί με την ανάλογη εντολή. Στη συνέχεια παραθέτουμε ενδεικτικά μια κλάση:

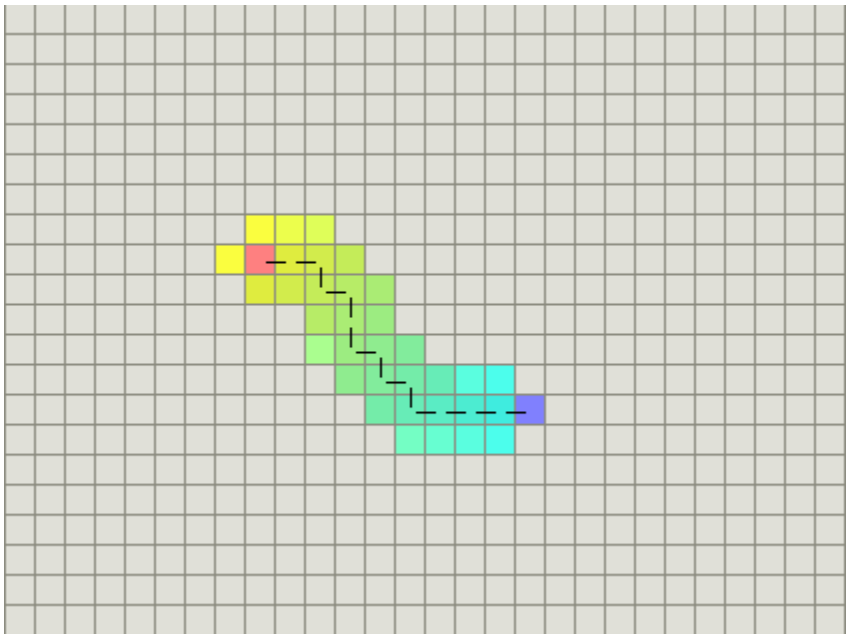
```
class MWUIScene_HighScoresMenu extends MWUIScene_StartMenu;

var transient MWGame_Button ButtonBack;

var UILabel NameLabel[10];
var UILabel ScoreLabel[10];

event PostInitialize()
{
   local int ctr;
   local string temp1;
   local string temp2;

   Super.PostInitialize();
   ButtonBack=MWGame_Button(FindChild('Button_Back',true));

   for(ctr=0;ctr<10;ctr++)
   {
      temp1="Name_"$string(ctr);
      temp2="Score_"$string(ctr);

      NameLabel[ctr]=UILabel(FindChild(name(temp1), true));
      ScoreLabel[ctr]=UILabel(FindChild(name(temp2), true));

   }
}

event SceneActivated(bool bInitialActivation)
{
   local int ctr;

   super.SceneActivated(bInitialActivation);
   ButtonBack.OnClicked=ButtonBackPressed;

   for(ctr=0;ctr<10;ctr++)
   {
      NameLabel[ctr].SetValue(MWGame(GetWorldInfo().Game).HighScores.Hi
                              ghScoresEntries[ctr].PlayerName);
      ScoreLabel[ctr].SetValue(string(MWGame(GetWorldInfo().Game).HighS
                              cores.HighScoresEntries[ctr].Score));
   }
}

function bool ButtonBackPressed(UIScreenObject EventObject,
                                int PlayerIndex)
{
   CloseScene(self);
   return true;
}

DefaultProperties
{
   SceneSkin=UISkin'MWContentPack.Menus.CGameSkin'
}
```

5.8 Ο αλγόριθμος A*

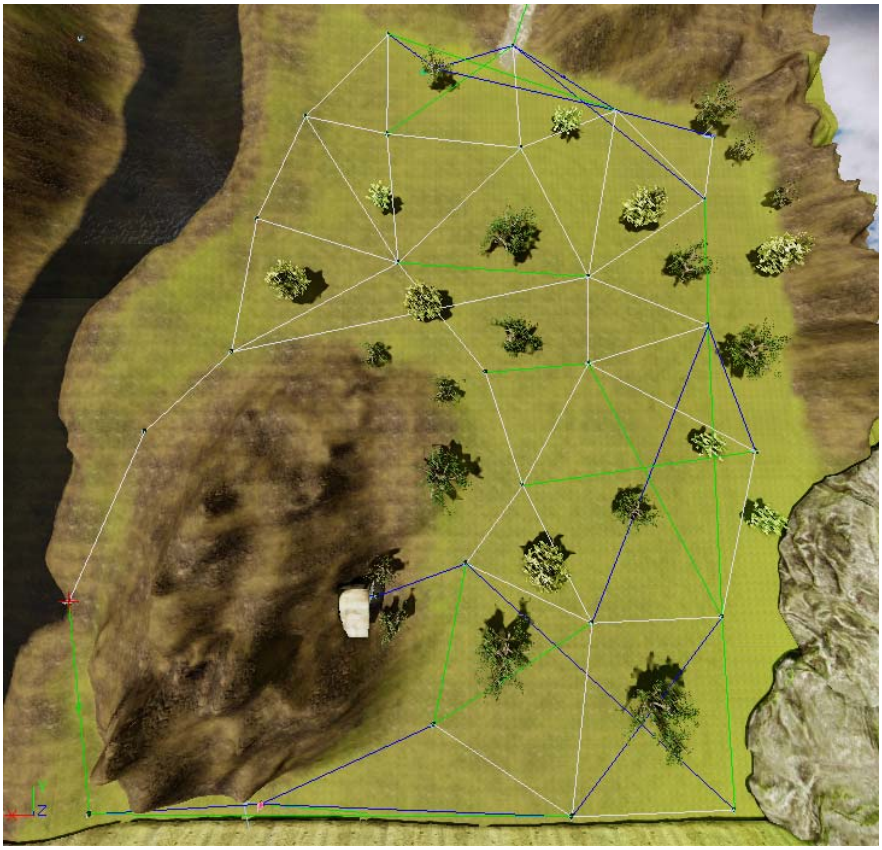
Το πιο ενδιαφέρον μέρος της υλοποίησης του παιχνιδιού ήταν ο αλγόριθμος τεχνητής νοημοσύνης. Ο τίτλος της διπλωματικής παραπέμπει σε κάποιου είδους βελτιστοποίηση στον τρόπο κίνησης των χαρακτήρων. Έτσι λοιπόν, χρησιμοποιήσαμε τον αλγόριθμο τεχνητής νοημοσύνης A* σε δυο περιπτώσεις. Η βασικότερη χρήση του A* γίνεται στον τρόπο που περιηγούνται οι αντίπαλοι στην πίστα (εκτός από τα διαστήματα που κυνηγούν τον χαρακτήρα) και δεύτερον στην αυτοματοποιημένη περιήγηση του χαρακτήρα μεταξύ δύο σημείων της πίστας που έχουμε προκαθορίσει. Και στις δύο περιπτώσεις ο αλγόριθμος είναι ο ίδιος, όμως είναι διαφορετικός ο τρόπος χρήσης του. Στη συνέχεια θα αναφερθούμε στην υλοποίηση του αλγορίθμου στην περίπτωση χρήσης του από τους αντιπάλους.

5.8.1 Εύρεση συντομότερου μονοπατιού

Το πρώτο πρόβλημα που αντιμετωπίσαμε ξεκινώντας την υλοποίηση ήταν ότι ο τρισδιάστατος χώρος παρότι έχει αρκετούς κόμβους δεν είναι ένα περιβάλλον τύπου «σκακιέρας». Κοιτώντας την εικόνα 5.6 καταλαβαίνουμε ότι όλος ο χώρος είναι χωρισμένος σε «κουτάκια» και με αυτό το διαχωρισμό είναι εύκολο να ορίσουμε το σημείο εκκίνησης, τερματισμού και την ενδιάμεση διαδρομή. Σε ένα παιχνίδι τα πράγματα δεν είναι ακριβώς έτσι. Ο τρισδιάστατος χώρος δεν χωρίζεται σε «κουτάκια» και ο αντίπαλος δεν μπορεί να βρίσκεται πάντα ακριβώς σε ένα κόμβο. Στην εικόνα 5.7 δείχνουμε ένα μέρος της πίστας του παιχνιδιού μαζί με ένα διάγραμμα της σύνδεσης των κόμβων.



Εικόνα 5.6: Περιβάλλον τύπου σκακιέρας



Εικόνα 5.7: Οι κόμβοι κ η διασύνδεση τους

Όπως φαίνεται ξεκάθαρα ο αντίπαλος μπορεί να βρεθεί σε οποιοδήποτε σημείο, οπότε ο αλγόριθμος δεν μπορεί να λειτουργήσει αν δεν του δώσουμε ως ορίσματα τους κόμβους εκκίνησης και τερματισμού. Για να λύσουμε αυτό το πρόβλημα χρησιμοποιούμε την παρακάτω συνάρτηση που εντοπίζει τον κοντινότερο κόμβο.

```
function int GetClosestVisibleNP()
{
   local NavigationPoint NP;
   local int RetNP;
   local float MinDist;
   local float TempDist;

   MinDist=1000000;
   RetNP=-1;

   foreach VisibleActors(class'NavigationPoint',NP,2000, Pawn.Location)
   {
      TempDist=VSize(Pawn.Location - NP.Location);
      if(TempDist<MinDist)
      {
            MinDist=TempDist;
            RetNP=GetIdByNavPoint(NP);
      }
   }
```

```
   return RetNP;
}
```

Με αυτή την συνάρτηση αναζητούμε μεταξύ όλων των κόμβων αυτόν που βρίσκεται σε απόσταση μικρότερη ή ίση με 2000 και είναι ορατός. Μόλις βρεθεί, η συνάρτηση επιστρέφει και εμείς μετακινούμε τον αντίπαλο από το σημείο που βρίσκεται στον κόμβο. Σε αυτό το σημείο έχουμε τον κόμβο εκκίνησης, οπότε μπορούμε να βρούμε τυχαία (στην περίπτωση περιήγησης που εξετάζουμε) έναν κόμβο τερματισμού, αρκεί ο κόμβος αυτός να ανήκει στην ομάδα κόμβων που έχουμε ορίσει να περιηγείται ο συγκεκριμένος αντίπαλος. Στη συνέχεια καλούμε την συνάρτηση του αλγορίθμου για τους δεδομένους κόμβους, την οποία παρουσιάζουμε εδώ:

```
function bool A_Star_Algorithm(int Start, int End)
{
   ClosedSet.Length=0;
   OpenSet.AddItem(Start);
   CameFromArray.Length=0;
   SetScoreArray(Start, 0, HeuristicEstimation(Start, End));

   while(OpenSet.Length>0)
   {
      NodeX=OpenSet[0];   //The node with the lowest F score

      if(NodeX==End)
      {
         Path.AddItem(End);
         CreatePath(End);
         return true;
      }

      RemoveFirstFromOpenset();   //Removes NodeX from OpenSet
      ClosedSet.AddItem(NodeX);

      SetVisibleNodes(NodeX);

      //For each neighbor of NodeX
      for(Actr=0;Actr<VisibleNodes.Length;Actr++)
      {
         NodeY=VisibleNodes[Actr];

         if(IsIdInClosedSet(NodeY)==true)
            continue;
         Temp_G_Score = ScoreArray[NodeX].G_Score +
                        HeuristicEstimation(NodeX,NodeY);

         if(IsIdInOpenSet(NodeY)==false)
         {
            OpenSet.AddItem(NodeY);
            bTempIsBetter=true;
            Flag=1;
         }
         else if(Temp_G_Score<ScoreArray[NodeY].G_Score)
         {
            bTempIsBetter=true;
            Flag=2;
```

```
            }
            else
               bTempIsBetter=false;

            if(bTempIsBetter==true)
            {
               CameFromArray[NodeY]=NodeX;
               SetScoreArray(NodeY, Temp_G_Score,
                             HeuristicEstimation(NodeY, End));

               //Sort node to the right position
               SortNodeInOpenSet(NodeY, Flag);
            }
         }
      }
      `log("ERROR");
      return false;
}
```

Ο αλγόριθμος έχει την μορφή του ψευδοκώδικα που παρουσιάσαμε στο κεφάλαιο 2. Έτσι, αρχικοποιούμε τις λίστες "OpenSet" και "ClosedSet". Η πρώτη είναι μια λίστα που περιέχει τους προσωρινούς κόμβους που πρέπει να αξιολογηθούν. Στην παρούσα εργασία έχουμε χρησιμοποιήσει δυαδικό σωρό (binary heap) για την υλοποίηση αυτής. Στην επόμενη υπό-ενότητα περιγράφουμε πως έχει γίνει αυτή η υλοποίηση. Η πρώτη καταχώρηση σε αυτή τη λίστα είναι ο κόμβος εκκίνησης. Η δεύτερη λίστα, για την οποία έχουμε χρησιμοποιήσει πίνακα για την αποθήκευση των στοιχείων, είναι αυτή με τους κόμβους που έχουν ήδη αξιολογηθεί. Αφού γίνει αυτή η αρχικοποίηση, συνεχίζουμε με τον βασικό βρόχο του αλγορίθμου. Ο βρόχος συνεχίζει για όσο διάστημα η λίστα "OpenSet" έχει κάποιο κόμβο. Αν δεν υπάρχει άλλος κόμβος, τότε επιστρέφει χωρίς επιτυχία, που σημαίνει ότι δεν υπάρχει διαδρομή από τον κόμβο εκκίνησης, σε αυτόν του τερματισμού.

Εφόσον η λίστα "OpenSet" δεν είναι κενή, ψάχνουμε σε αυτή για τον κόμβο με το χαμηλότερο κόστος. Σε αυτό το σημείο πρέπει να αναφέρουμε το κέρδος της χρήσης δυαδικού σωρού για την υλοποίησης της λίστας. Επειδή σε αυτή τη λίστα μας ενδιαφέρει να εξάγουμε μόνο τον κόμβο με το μικρότερο κόστος, ο δυαδικός σωρός είναι η βέλτιστη δομή αποθήκευσης. Η χρονική πολυπλοκότητα της αναζήτησης είναι O(1) και της εισαγωγής και διαγραφής O(log n). Αντίθετα η χρονική πολυπλοκότητα για την λίστα είναι για την αναζήτηση και την διαγραφή ενός στοιχείου O(n) και της εισαγωγής O(1). Φυσικά οι πολυπλοκότητες που αναφέραμε για τον σωρό ισχύουν όταν μας ενδιαφέρει ο κόμβος με το μικρότερο κόστος και όχι οι υπόλοιποι. Σε αντίθετη περίπτωση οι πολυπλοκότητες είναι διαφορετικές.

Μόλις εντοπίσουμε τον πρώτο κόμβο στην λίστα "OpenSet", ελέγχουμε αν είναι ο κόμβος τερματισμού. Αν είναι, καλούμε μια αναδρομική συνάρτηση που δημιουργεί τον πίνακα με τους κόμβους που πρέπει να διαπεράσουμε για να ολοκληρώσουμε την διαδρομή. Η αναδρομική συνάρτηση είναι η εξής:

```
function CreatePath(int curr)
{
      local int i;
      if(curr==StartNode)
            return;
```

```
	for(i=0;i<CameFromArray.Length;i++)
	{
		if(i==curr)
		{
			Path.AddItem(CameFromArray[i]);
			CreatePath(CameFromArray[i]);
			break;
		}
	}
}
```

Αν δεν είναι ο κόμβος τερματισμού, τον εξάγουμε από την "OpenSet" και τον εισάγουμε στην "ClosedSet". Στη συνέχεια, αναζητούμε στον τρισδιάστατο χώρο για όλους τους κόμβους που είναι ορατοί και προσπελάσιμοι από το σημείο που εξερευνούμε αυτή τη στιγμή. Για κάθε κόμβο που είναι ορατός ελέγχουμε αν βρίσκεται ήδη στην "ClosedSet" και αν βρίσκεται τον προσπερνούμε, αφού τον κόμβο αυτό τον έχουμε ήδη επεξεργαστεί. Για τον έλεγχο κάνουμε γραμμική αναζήτηση στη λίστα με χρονική πολυπλοκότητα O(n). Εφόσον δεν βρίσκεται, υπολογίζουμε το κόστος G για την πρόσβαση σε αυτόν, που ισούται με το άθροισμα του κόστους G του «πατέρα» συν το κόστος της απόστασης μεταξύ του «πατέρα» και του κόμβου που επεξεργαζόμαστε. Το κόστος της απόστασης υπολογίζεται από την παρακάτω συνάρτηση:

```
function float HeuristicEstimation(int Start, int End)
{
	local float Distance;
	local NavigationPoint StartNP;
	local NavigationPoint EndNP;

	StartNP=MappingArray[Start]; //Set the starting NavigationPoint
	EndNP=MappingArray[End];     //Set the ending NavigationPoint

	//Returns the euclidean size of the vector (the square root of
	the sum of the components squared).
	Distance=VSize(StartNP.Location - EndNP.Location);

	return Distance;
}
```

Στην παραπάνω συνάρτηση βλέπουμε ότι η απόσταση μεταξύ δύο κόμβων υπολογίζεται χρησιμοποιώντας τα διανύσματα θέσης τους. Τα διανύσματα αυτά περιέχουν πληροφορία και στους τρεις άξονες X, Y, Z και με αυτό τον τρόπο λαμβάνουν υπ' όψιν και τις διάφορες υψομετρικές διαφορές που παρατηρούνται στην πίστα. Συνεπώς, παρότι ο χώρος ενός παιχνιδιού είναι τρισδιάστατος, οι μετακινήσεις γίνονται σε υποχώρο δύο διαστάσεων, με την τρίτη να επηρεάζει μόνο το κόστος.

Συνεχίζοντας, ελέγχουμε αν ο κόμβος που επεξεργαζόμαστε βρίσκεται στην "OpenSet". Αν δεν βρίσκεται ήδη, αυτό σημαίνει ότι πρέπει να τον προσθέσουμε. Γνωρίζουμε τον «πατέρα», οπότε τον αποθηκεύουμε για να ξέρουμε πως φτάσαμε ως εδώ. Επίσης αποθηκεύουμε το κόστος G, το κόστος H (το κόστος από τον κόμβο αυτό ως τον τερματικό κόμβο) και το συνολικό κόστος F. Φυσικά, πρέπει να ταξινομήσουμε την λίστα "OpenSet" μετά την νέα εισαγωγή, αλλά θα περιγράψουμε τον τρόπο στην επόμενη υπό-ενότητα.

Στην περίπτωση που ο κόμβος που επεξεργαζόμαστε βρίσκεται στην λίστα "OpenSet", υπάρχουν δυο υπό-περιπτώσεις. Η πρώτη είναι το προσωρινό κόστος G να είναι μικρότερο του προηγούμενου. Εφόσον είναι μικρότερο, τότε πρέπει να ανανεώσουμε τα κόστη G,H και F, να αποθηκεύσουμε το νέο «πατέρα» και να κάνουμε την ταξινόμηση της λίστας. Η δεύτερη περίπτωση (όπου το προσωρινό κόστος G δεν είναι μικρότερο του προηγούμενου) μας ενημερώνει ότι δεν χρειάζεται να επεξεργαστούμε εκ νέου τα στοιχεία.

Σε αυτό το σημείο έχουμε κάνει όλους τους ελέγχους και αν όλα έχουν πάει καλά ο αλγόριθμος θα επιστρέψει το συντομότερο μονοπάτι από τον κόμβο εκκίνησης στον τερματικό. Στην επόμενη υπό-ενότητα θα περιγράψουμε τον τρόπο που υλοποιήσαμε τον δυαδικό σωρό που βελτιστοποιεί τον αλγόριθμο και τον κάνει ταχύτερο.

5.8.2 Βελτίωση A* με δυαδικό σωρό (binary heap)

Η πρώτη έκδοση του αλγορίθμου στο παιχνίδι ήταν χρησιμοποιώντας απλούς πίνακες για τις λίστες. Στη συνέχεια θελήσαμε να επιταχύνουμε την εκτέλεση του αλγορίθμου αντικαθιστώντας τον πίνακα "OpenSet" με δυαδικό σωρό. Οι συναρτήσεις που έχουμε υλοποιήσει είναι εξειδικευμένες καθώς η χρήση που κάνουμε στον δυαδικό σωρό είναι ιδιαίτερη. Ο αλγόριθμος A* απαιτεί την αναζήτηση και διαγραφή μόνο του πρώτου στοιχείου (και όχι οποιουδήποτε στοιχείου της λίστας) και την ταξινόμηση αυτής είτε μετά από μια εισαγωγή ενός νέου στοιχείου, είτε μετά από ανανέωση του κόστους μιας υπάρχουσας καταχώρησης.

Η πρώτη ενέργεια που κάνουμε στον αλγόριθμο A* είναι η αναζήτηση του πρώτου στοιχείου της λίστας "OpenSet", ο οποίος έχει το μικρότερο κόστος F. Όπως έχουμε αναφέρει, η λίστα είναι πάντα ταξινομημένη με τέτοιο τρόπο, ώστε το πρώτο της στοιχείο να έχει το μικρότερο κόστος F. Άρα, μπορούμε να διαβάσουμε το πρώτο στοιχείο εύκολα και με χρονική πολυπλοκότητα O(1):

```
NodeX=OpenSet[0];
```

Στη συνέχεια καλούμαστε να διαγράψουμε το πρώτο στοιχείο της λίστας, αλλά να διατηρήσουμε στην πρώτη θέση το στοιχείο με το μικρότερο κόστος F. Για την υλοποίηση της διαγραφής του πρώτου στοιχείου της λίστας εργασθήκαμε σύμφωνα με το άρθρο [23] όπου περιγράφει επακριβώς σε ψευδοκώδικα την διαδικασία:

```
function RemoveFirstFromOpenset()
{
   local int u;
   local int v;
   local int temp;

   //Replace the first with the last item
   OpenSet[0]=OpenSet[OpenSet.Length-1];
   //Reduce the number of items in the list by one
   OpenSet.Length=OpenSet.Length-1;

   v=0;
   while(true)
```

```
    {
        u=v;

        if(2*(u+1) <= OpenSet.Length-1)     //If both children exist
        {
            //Select the lowest of the two children
            if(ScoreArray[OpenSet[2*u+1]].F_Score <=
                ScoreArray[OpenSet[2*(u+1)]].F_Score)
            {
                if(ScoreArray[OpenSet[u]].F_Score >
                    ScoreArray[OpenSet[2*u+1]].F_Score)
                        v=2*u+1;
            }
            else
            {
                if(ScoreArray[OpenSet[u]].F_Score >
                    ScoreArray[OpenSet[2*(u+1)]].F_Score)
                        v=2*(u+1);
            }
        }
        else if(2*(u+1)-1 <= OpenSet.Length-1) //If only child #1 exists
        {
            //Check if the F cost is greater than the child
            if(ScoreArray[OpenSet[u]].F_Score >=
                ScoreArray[OpenSet[2*u+1]].F_Score)
                v=2*u+1;
        }

        if(u != v)  //If parent's F > one or both of its children,
                    //swap them
        {
            temp=OpenSet[u];
            OpenSet[u]=OpenSet[v];
            OpenSet[v]=temp;
        }
        else
            break;
    }
}
```

Η λογική της συνάρτησης είναι αφού διαγράψουμε το πρώτο στοιχείο να το αντικαταστήσουμε με το τελευταίο στοιχείο της λίστας. Έπειτα, κάνουμε διαδοχικές συγκρίσεις του κόστους F αυτού του στοιχείου και των παιδιών του και όποτε χρειάζεται ανταλλάσουμε τις θέσεις τους. Η συνάρτηση επιστρέφει όταν το στοιχείο βρει την κατάλληλη θέση μέσα στη λίστα, που σημαίνει ότι όλοι οι γονείς έχουν κόστος μικρότερο από τα παιδιά τους. Συνεπώς το πρώτο στοιχείο της λίστας μετά την ταξινόμηση θα έχει το μικρότερο κόστος F.

Η τελευταία διαδικασία που συμβαίνει στην λίστα “OpenSet” είναι η ταξινόμηση της ύστερα από μια νέα εισαγωγή στοιχείου, ή μετά από μια ανανέωση του κόστους ενός υπάρχοντος στοιχείου. Η συγκεκριμένη διαδικασία προϋποθέτει να γνωρίζουμε σε ποια από τις δύο περιπτώσεις βρισκόμαστε. Για αυτό το λόγο χρησιμοποιούμε μια μεταβλητή, ώστε να γνωρίζουμε αν το στοιχείο που θα ταξινομήσουμε βρίσκεται στην τελευταία θέση της λίστας, ή σε κάποιο σημείο οπουδήποτε μέσα σε αυτή.

Αν το στοιχείο βρίσκεται στην τελευταία θέση της λίστας, γνωρίζουμε άμεσα τη θέση του και το συγκρίνουμε με τον «πατέρα» του. Αν το κόστος του είναι μικρότερο του «πατέρα» τα αλλάζουμε θέσεις. Συνεχίζουμε την διαδικασία μέχρι να φτάσει στην θέση που του αντιστοιχεί. Αν σε κάποιο στοιχείο γίνει ανανέωση στο κόστος και βρίσκεται ήδη στην ανοιχτή λίστα, πρώτα πρέπει να το εντοπίσουμε με γραμμική αναζήτηση. Η γραμμική αναζήτηση κοστίζει, αλλά οι κόμβοι είναι σχετικά λίγοι, οπότε το κόστος δεν μας επηρεάζει ιδιαίτερα. Μόλις το εντοπίσουμε, μπορούμε να συνεχίσουμε την διαδικασία με τις ανταλλαγές όπως προηγουμένως, ανεβάζοντας το όσο πιο ψηλά γίνεται. Σημειώνουμε ότι το ανανεωμένο κόστος μπορεί να είναι μόνο μικρότερο από το παλιό και για αυτό το στοιχείο δεν μπορεί να μετακινηθεί κατώτερα στη λίστα, παρά μόνο προς τη ρίζα. Στη συνέχεια παραθέτουμε την υλοποίηση της συνάρτησης:

```
function SortNodeInOpenSet(int id, int modeFlag)
{
	local int m;
	local int temp;
	local int i;

	if(modeFlag==1)	//Case 1: Node inserted for first time. Its
	{			//position is at the end of the OpenSet array
		m=OpenSet.Length;
	}
	else if(modeFlag==2) //Case 2: Node is already in the OpenSet.
				//So, first find it linearly
	{
		for(i=0;i<OpenSet.Length;i++)
		{
			if(OpenSet[i]==id)
				break;
		}
		m=i+1;
	}
	else
	{
		`log("Error in SortNodeInOpenSet()");
		return;
	}
	//Main loop that sets the node to the right position
	while(m != 1)
	{
		//Compare the added element with its parent; if they are in
		//the correct order, stop.
		if(ScoreArray[OpenSet[m-1]].F_Score <=
		   ScoreArray[OpenSet[FFloor(m/2)-1]].F_Score)
		{
			temp=OpenSet[FFloor(m/2)-1];
			OpenSet[FFloor(m/2)-1]=OpenSet[m-1];
			OpenSet[m-1]=temp;
			m=FFloor(m/2);
		}
		else
			break;
	}
}
```

ΚΕΦΑΛΑΙΟ 6
ΑΞΙΟΛΟΓΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

6.1 Εισαγωγή

Στο κεφάλαιο αυτό θα παρουσιάσουμε μια μέθοδο αξιολόγησης και τα αποτελέσματα αυτής σχετικά με το παιχνίδι που δημιουργήσαμε στα πλαίσια της διπλωματικής εργασίας.

6.2 Μέθοδος αξιολόγησης

Για την αξιολόγηση του συστήματος χρησιμοποιήσαμε το ερωτηματολόγιο για την ικανοποίηση της αλληλεπίδρασης του χρήστη (Questionnaire for User Interaction Satisfaction - QUIS) [24]. Το ερωτηματολόγιο περιλαμβάνει ερωτήσεις που αφορούν την γενική εντύπωση του χρήστη, την επικοινωνία με το σύστημα, την ευκολία εκμάθησης, την εγκατάσταση της εφαρμογής, κτλ.

Στην διαδικασία αξιολόγησης συμμετείχαν δέκα χρήστες διαφορετικών ηλικιών και γνώσεων σχετικά με υπολογιστές και τρισδιάστατα παιχνίδια γραφικών. Το ερωτηματολόγιο περιλαμβάνει 39 ερωτήσεις, χωρισμένες σε επτά ενότητες. Η κλίμακα είναι από ένα ως εννιά καθώς επίσης και απάντηση «δεν γνωρίζω/δεν απαντώ». Στην συνέχεια παραθέτουμε το ερωτηματολόγιο:

Ερωτηματολόγιο Αξιολόγησης Διαλογικής Χρήσης Συστήματος (QUIS - Πανεπιστήμιο του Maryland, 1997)

Αριθμός ερ/γίου: _____
Σύστημα: _____
Ηλικία: _____
Φύλο: άρρεν ___ θήλυ___

ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη

Παρακαλώ κυκλώστε τους αριθμούς που αντικατοπτρίζουν ακριβέστερα τις εντυπώσεις σας από τη χρήση αυτού του υπολογιστικού συστήματος. Δεν ξέρω/δεν απαντώ = NA.

1.1. Γενική αντίδραση του Συστήματος:

απαράδεκτη υπέροχη
1 2 3 4 5 6 7 8 9 NA

1.2. Γενική αντίδραση του Συστήματος:

μπερδεύει ικανοποιεί
1 2 3 4 5 6 7 8 9 NA

1.3. Γενική αντίδραση του Συστήματος :

χαζό ευχάριστο
1 2 3 4 5 6 7 8 9 NA

1.4. Γενική αντίδραση του Συστήματος :

δύσκολο εύκολο
1 2 3 4 5 6 7 8 9 NA

1.5. Γενική αντίδραση του Συστήματος :

άκαμπτο ευέλικτο
1 2 3 4 5 6 7 8 9 NA

ΜΕΡΟΣ 2: Οθόνη

2.1. Ο σχεδιασμός της οθόνης βοήθησε:

2.1.1. Ο σχεδιασμός της οθόνης βοήθησε: ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

2.1.2. Ποσότητα πληροφορίας που εμφανίζονταν στην οθόνη :

ανεπαρκής επαρκής
1 2 3 4 5 6 7 8 9 NA

2.1.3. Δόμηση της πληροφορίας που εμφανίζονταν στην οθόνη :

χαοτική οργανωμένη
1 2 3 4 5 6 7 8 9 NA

2.2. Αλληλουχία οθονών :

2.2.1. Αλληλουχία οθονών : μπερδεμένη σαφής
1 2 3 4 5 6 7 8 9 NA

2.2.2. Επόμενη οθόνη στη σειρά : απρόβλεπτη προβλέψιμη
1 2 3 4 5 6 7 8 9 NA

2.2.3. Επιστροφή στην προηγούμενη οθόνη : αδύνατη εύκολη
1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλια σας για τη δομή των οθονών εδώ :

_____
_____
_____

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

3.1. Τα μηνύματα εμφανίζονται στην οθόνη με:

3.1.1. Τα μηνύματα εμφανίζονται στην οθόνη με :

ασυνέπεια συνέπεια
1 2 3 4 5 6 7 8 9 NA

3.2. Τα μηνύματα που εμφανίζονται στην οθόνη είναι:

3.2.1. Τα μηνύματα που εμφανίζονται στην οθόνη είναι :

μπερδεμένα σαφή
1 2 3 4 5 6 7 8 9 NA

3.3. Ο υπολογιστής σας ενημερώνει για το τι κάνει:

3.3.1. Ο υπολογιστής σας ενημερώνει για το τι κάνει:

ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

3.3.2. Εκτελώντας μια κίνηση οδηγούμαστε σε προβλέψιμο αποτέλεσμα :

ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

3.3.3. Ο έλεγχος της ποσότητας της ανάδρασης του Συστήματος:

αδύνατος εύκολος
1 2 3 4 5 6 7 8 9 NA

3.3.4 Καθυστερήσεις : απαράδεκτες παραδεκτές
1 2 3 4 5 6 7 8 9 NA

3.4. Μηνύματα λαθών :

3.4.1.Μηνύματα λαθών : δεν βοηθούν πολύ βοηθητικά
1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για την ορολογία και την επικοινωνία με το Σύστημα εδώ :

_____
_____
_____

ΜΕΡΟΣ 4: Εκμάθηση χρήσης

4.1. Η εκμάθηση χρήσης του συστήματος είναι:

4.1.1. Η εκμάθηση χρήσης του συστήματος είναι:

δύσκολη εύκολη
1 2 3 4 5 6 7 8 9 NA

4.1.2. Χρόνος για την εκμάθηση χρήσης του συστήματος :

πολύς λίγος
1 2 3 4 5 6 7 8 9 NA

4.2. Η εξερεύνηση των δυνατοτήτων με τη μέθοδο "προσπάθειας και λάθους (trial and error)":

4.2.1. Η εξερεύνηση των δυνατοτήτων με τη μέθοδο "προσπάθειας και λάθους (trial and error)":

απογοητεύει αποδίδει
1 2 3 4 5 6 7 8 9 NA

4.3. Οι εργασίες γίνονται σε λογική αλληλουχία:

4.3.1. Οι εργασίες γίνονται σε λογική αλληλουχία:

ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

4.3.2. Τα βήματα για την ολοκλήρωση της εργασίας ακολουθούν μια λογική σειρά

ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

4.3.3 Ανάδραση όταν ολοκληρωθεί η εργασία:

ασαφής σαφής
1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για την εκμάθηση της χρήσης εδώ :

_____
_____
_____

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

5.1. Η ταχύτητα του Συστήματος είναι:

5.1.1. Η ταχύτητα του Συστήματος είναι: πολύ αργή ικανοποιητική
1 2 3 4 5 6 7 8 9 NA

5.2. Το Σύστημα είναι σταθερό :

5.2.1. Το Σύστημα είναι σταθερό: ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

5.2.2. Χειρισμοί – Λειτουργίες αναξιόπιστα αξιόπιστα
1 2 3 4 5 6 7 8 9 NA

5.3. Η ευκολία χειρισμού εξαρτάται από την εμπειρία του χρήστη:

5.3.1. Η ευκολία χειρισμού εξαρτάται από την εμπειρία του χρήστη:
ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για τις δυνατότητες του Συστήματος εδώ :
_____
_____
_____

ΜΕΡΟΣ 6: Πολυμέσα

6.1. Η ποιότητα των εικόνων/ φωτογραφιών είναι:
κακή καλή
1 2 3 4 5 6 7 8 9 NA

6.1.1.Εικόνες/ Φωτογραφίες :
θολές καθαρές
1 2 3 4 5 6 7 8 9 NA

6.1.2. Η φωτεινότητα της εικόνας/ φωτογραφίας :
σκοτεινή φωτεινή
1 2 3 4 5 6 7 8 9 NA

6.2. Η ηχητική απόδοση είναι :

6.2.1. Η ηχητική απόδοση είναι :
ασταθής εύρυθμη
1 2 3 4 5 6 7 8 9 NA

6.2.2. Η ηχητική απόδοση είναι:
αλλοιωμένη ξεκάθαρη
1 2 3 4 5 6 7 8 9 NA

6.3. Τα χρώματα που χρησιμοποιούνται είναι :

αφύσικα φυσικά
1 2 3 4 5 6 7 8 9 NA

6.3.1. Η ποσότητα των χρωμάτων που είναι διαθέσιμη είναι :

ανεπαρκής επαρκής
1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για τα πολυμέσα εδώ :

_____
_____
_____

ΜΕΡΟΣ 7: Εγκατάσταση συστήματος

7.1. Η ταχύτητα του εγκατάστασης του συστήματος είναι:

αργή γρήγορη
1 2 3 4 5 6 7 8 9 NA

7.2. Η εξατομίκευση είναι:

δύσκολη εύκολη
1 2 3 4 5 6 7 8 9 NA

7.3. Πληροφορείται ο χρήστης για την πρόοδό της :

ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

7.4. Δίνει εποικοδομητικές εξηγήσεις όταν συμβαίνουν αποτυχίες :

ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

Παρακαλώ γράψτε τα σχόλιά σας για την εγκατάσταση του συστήματος εδώ :

_____
_____
_____

6.3 Αποτελέσματα ερωτηματολογίου

Στη συνέχεια, παραθέτουμε συγκεντρωτικά τα αποτελέσματα, τα οποία μας βοήθησαν να βελτιστοποιήσουμε την εφαρμογή, μεγεθύνοντας την ευχαρίστηση του παίκτη. Σημειώνουμε ότι λόγω της φύσης της εφαρμογή (τρισδιάστατο παιχνίδι γραφικών) ορισμένες ερωτήσεις δεν είναι δυνατόν να απαντηθούν και ως εκ τούτου τα αποτελέσματα σημειώνονται με "-".

ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη

Ερώτηση	Μ.Ο. Βαθμολογίας
1.1	8.1
1.2	7.3
1.3	7.7
1.4	8.8
1.5	6.8

ΜΕΡΟΣ 2: Οθόνη

Ερώτηση	Μ.Ο. Βαθμολογίας
2.1.1	7.9
2.1.2	7.4
2.1.3	9
2.2.1	8.5
2.2.2	7.7
2.2.3	7.8

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

Ερώτηση	Μ.Ο. Βαθμολογίας
3.1.1	8.5
3.2.1	7.7
3.3.1	5.9
3.3.2	7.4
3.3.3	-
3.3.4	9
3.4.1	-

ΜΕΡΟΣ 4: Εκμάθηση χρήσης

Ερώτηση	Μ.Ο. Βαθμολογίας
4.1.1	7.1
4.1.2	8
4.2.1	8.7
4.3.1	7.4
4.3.2	7.7
4.3.3	8.1

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

Ερώτηση	Μ.Ο. Βαθμολογίας
5.1.1	8.3
5.2.1	8.8
5.2.2	7.8
5.3.1	6.8

ΜΕΡΟΣ 6: Πολυμέσα

Ερώτηση	Μ.Ο. Βαθμολογίας
6.1	9
6.1.1	8.9
6.1.2	8.7
6.2.1	8
6.2.2	7.8
6.3	8.6
6.3.1	8.1

ΜΕΡΟΣ 7: Εγκατάσταση συστήματος

Ερώτηση	Μ.Ο. Βαθμολογίας
7.1	5.6
7.2	7.8
7.3	8.9
7.4	6.6

6.4 Συμπεράσματα

Στη συνέχεια παρουσιάζουμε τα συμπεράσματα που προέκυψαν από την κριτική των χρηστών που απάντησαν στο ερωτηματολόγιο, ομαδοποιημένα κατά ομάδες ερωτήσεων.

ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη

Η γενική εντύπωση των χρηστών είναι ικανοποιητική από την συνολική αντίδραση της εφαρμογής. Το παιχνίδι προσφέρει ένα ευχάριστο διάλλειμα στο χρήστη, αποτέλεσμα που προκύπτει από το γεγονός ότι είναι ένα σχετικά εύκολο παιχνίδι γραφικών σε σχέση με το επίπεδο των σύγχρονων παιχνιδιών. Φυσικά, η κριτική στην εφαρμογή ποικίλει μεταξύ των χρηστών, ανάλογα με το βαθμό εξοικείωσης του καθενός με τα παιχνίδια, αλλά και με τους υπολογιστές γενικότερα.

ΜΕΡΟΣ 2: Οθόνη

Σύμφωνα με τις απαντήσεις των χρηστών υπάρχουν κάποια προβλήματα στον τρόπο που εμφανίζονται οι πληροφορίες στην οθόνη, όπως για παράδειγμα η μη ενημέρωση από το σύστημα, όταν αυτό βρίσκεται σε κατάσταση φόρτωσης (loading). Παρά όλα αυτά, μετά την απαραίτητη εξοικείωση όλοι οι χρήστες δεν αντιμετώπισαν κανένα πρόβλημα και συνέχισαν τη χρήση της εφαρμογής ομαλά.

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

Καθυστερήσεις και λάθη στα μηνύματα λαθών δεν παρατηρήθηκαν. Ένα στοιχείο το οποίο αναφέρθηκε ήταν η μέτρια ενημέρωση του χρήστη κατά την επίθεση του στον αντίπαλο. Συνεπώς, επιβάλλεται η προσθήκη ενός μηνύματος που να ενημερώνει σε ποιο επίπεδο βρίσκεται η υγεία του αντιπάλου.

ΜΕΡΟΣ 4: Εκμάθηση χρήσης

Η εκμάθηση του παιχνιδιού, όπως αναφέραμε ποικίλει ανάλογα με το επίπεδο του κάθε χρήστη. Η βαθμολογία που συγκεντρώσαμε σε αυτή την ομάδα ερωτήσεων είναι αρκετά πάνω του μετρίου, κυρίως λόγω της απλής μορφής του παιχνιδιού και των εύκολα κατανοητών γραφικών διεπαφών. Η λογική της μεθόδου “προσπάθεια και λάθος (trial and error)” κρίνεται αρκετά ικανοποιητική από τους χρήστες.

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

Το σύστημα, σύμφωνα με τις γνώμες των χρηστών αποδείχτηκε πολύ σταθερό,

λόγω της μηχανής παιχνιδιών που χρησιμοποιήσαμε. Όπως έχουμε ήδη αναφέρει, η UDK είναι μια μηχανή επαγγελματικού επιπέδου και με δεδομένο ότι δημιουργήσαμε ένα παιχνίδι χωρίς κάποια bugs, η εφαρμογή δεν τερματίστηκε ξαφνικά σχεδόν ποτέ. Ακόμα και κατά την περίοδο δημιουργίας του παιχνιδιού, ξαφνικός τερματισμός παρατηρήθηκε ελάχιστες φορές. Ακόμα, η ταχύτητα του παιχνιδιού χαρακτηρίστηκε ικανοποιητική, λόγω της μορφής του παιχνιδιού, το οποίο είναι ελαφρύ εκ κατασκευής.

ΜΕΡΟΣ 6: Πολυμέσα

Η ποιότητα των εικόνων άφησε πάρα πολύ καλές εντυπώσεις, τόσο με την ευκρίνεια τους, όσο και με την φωτεινότητα. Επίσης, τα χρώματα θεωρήθηκαν πολύ ικανοποιητικά και δεν δημιούργησαν κάποιο πρόβλημα στην παρακολούθηση του παιχνιδιού. Η ποιότητα των ήχων αναφέρθηκε ως ιδανική, όμως είναι απαραίτητη η προσθήκη περισσοτέρων ηχητικών εφφέ και κομματιών.

ΜΕΡΟΣ 7: Εγκατάσταση συστήματος

Η εγκατάσταση της εφαρμογής είναι προϊόν μιας ειδικής εφαρμογής της UDK. Αυτό σημαίνει ότι το πρόγραμμα εγκατάστασης δημιουργείται αυτόματα από την μηχανή, παρέχοντας όλα τα πακέτα που χρειάζονται, μαζί με ένα πλήρως αυτοματοποιημένο πρόγραμμα. Συνεπώς, παρέχονται όλες οι πληροφορίες που χρειάζονται και γενικά δεν παρατηρούνται προβλήματα κατά την εγκατάσταση.

ΚΕΦΑΛΑΙΟ 7
ΑΠΟΤΕΛΕΣΜΑΤΑ Κ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

7.1 Στόχος και αποτελέσματα

Στόχος της παρούσης εργασίας ήταν η δημιουργία ενός τρισδιάστατου παιχνιδιού γραφικών με τη χρήση μιας μηχανής παιχνιδιών. Σχεδιάσαμε και υλοποιήσαμε τα διαφορετικά μέρη ενός παιχνιδιού, τα οποία χρειάζονται για να λειτουργεί αυτόνομα σαν μια εφαρμογή του εμπορίου.

Αρχικά, ξεκινήσαμε αναζητώντας την κατάλληλη μηχανή παιχνιδιών που θα μας παρείχε τα βασικά εργαλεία για την δημιουργία του παιχνιδιού. Μέσα από διάφορες επιλογές, επιλέξαμε αυτή που δίνει τις περισσότερες δυνατότητες και είναι αυτό που λέμε πρότυπο (standard) στη σημερινή εποχή. Επινοήσαμε ένα σενάριο, ικανό να προσδώσει ένα ενδιαφέρον και ξεκινήσαμε βήμα-βήμα την υλοποίηση. Χρησιμοποιώντας στην αρχή κάποια έτοιμα στοιχεία, όπως τρισδιάστατο χώρο και μοντέλα χαρακτήρων, προσθέσαμε λειτουργικότητα στον χαρακτήρα που χειρίζεται ο παίκτης και στους αντιπάλους. Σύντομα καταλάβαμε ότι οι αντίπαλοι έμοιαζαν να μην είναι καθόλου ευφυείς. Έτσι, δημιουργήσαμε τις διάφορες καταστάσεις στην συμπεριφορά τους. Άλλοτε περιπλανιόνται στον χώρο, άλλοτε κυνηγούν τον αντίπαλο και αν τον χάσουν επιστρέφουν στη θέση τους. Ως εδώ, οι αντίπαλοι προσπαθούσαν να εξοντώσουν τον χαρακτήρα του παίκτη, όμως συχνά συγκρούονταν με εμπόδια στην πίστα. Έτσι, αποφασίσαμε να χρησιμοποιήσουμε τεχνητή νοημοσύνη για τις μετακινήσεις τους.

Παράλληλα, ξεκινήσαμε να σχεδιάζουμε τα μοντέλα, τα αντικείμενα και τον τρισδιάστατο χώρο του παιχνιδιού. Μια διαδικασία αρκετά επίπονη, για κάποιον που δεν είναι σχεδιαστής πίστας ή κινουμένων σχεδίων (level designer ή animator). Με βοήθεια από το διαδίκτυο, από βιβλία και έτοιμα μοντέλα, ο τρισδιάστατος χώρος απέκτησε κατάλληλο ύφος σχετικό με το σενάριο.

Στο σημείο αυτό, το παιχνίδι δεν έμοιαζε καθόλου ολοκληρωμένο. Έτσι, προσθέσαμε λειτουργικότητα για να αποκτήσει ενδιαφέρον. Ο χαρακτήρας μπορούσε πλέον να συλλέξει νομίσματα, μαγικά φίλτρα που του δίνουν επιπλέον δυνάμεις, να ανοίξει πόρτες, βρίσκοντας τον κρυμμένο μηχανισμό. Στη συνέχεια, υλοποιήσαμε ένα τρόπο ώστε να μπορεί να τρέχει γρηγορότερα και να περνά μια λίμνη πάνω σε ένα κινούμενο βράχο. Όμως ακόμα δεν μπορούσε να κάνει κακό στους αντιπάλους του. Για αυτό το λόγο προσθέσαμε στην αρχή ένα όπλο κοντινής απόστασης και μόλις είδαμε ότι αυτό δεν είναι αρκετό προσθέσαμε και ένα όπλο μακρινής απόστασης.

Έπειτα, για να ενημερώνεται ο χρήστης, συμπεριλάβαμε στο παιχνίδι μια γραφική διεπαφή. Ένα αρχικό μενού, ένα ενδιάμεσο για να μπορεί να σταματήσει την εφαρμογή κατά τη διάρκεια εκτέλεσης και ένα σύστημα αποθήκευσης αντικειμένων για να αποθηκεύονται τα αντικείμενα που συλλέγει. Μόλις το παιχνίδι πήρε αυτή τη σχεδόν ολοκληρωμένη μορφή, αποφασίσαμε να ασχοληθούμε με την υλοποίηση του πιο ενδιαφέροντος υπό-συστήματος της εφαρμογής. Ο αλγόριθμος τεχνητής νοημοσύνης θα «μάθαινε» στους αντιπάλους να μετακινούνται μέσα στον τρισδιάστατο χώρο δείχνοντας σημάδια ευφυΐας. Πρώτα, δημιουργήσαμε ένα πλέγμα από κόμβους και ξεκινήσαμε την υλοποίηση. Μόλις καταφέραμε να μετακινούνται οι αντίπαλοι με βέλτιστο τρόπο, αποφασίσαμε να βελτιστοποιήσουμε τον αλγόριθμο προσθέτοντας έναν δυαδικό σωρό

(binary heap). Η προσθήκη τεχνητής νοημοσύνης σε ένα παιχνίδι γραφικών ήταν μια επίπονη, αλλά πολύ ενδιαφέρουσα διαδικασία. Τέλος, χρησιμοποιήσαμε τον βελτιστοποιημένο αλγόριθμο και στην πλοήγηση του χαρακτήρα αυτοματοποιώντας μια μετακίνηση του στον χώρο.

7.2 Μελλοντικές επεκτάσεις και βελτιώσεις

Το παιχνίδι που δημιουργήσαμε μπορεί να βελτιωθεί με διάφορους τρόπους. Αρχικά, θα μπορούσαμε να δημιουργήσουμε περισσότερες πίστες, όπου θα διαδραματίζονται νέα περιστατικά προκαλώντας το ενδιαφέρον του παίκτη. Φυσικά, η οργάνωση του παιχνιδιού θα μπορούσε να γίνει σε αποστολές, έτσι ώστε να χρειάζεται να συμβεί κάτι πριν ο παίκτης προχωρήσει σε επόμενο επίπεδο. Άλλο ένα στοιχείο είναι η προσθήκη του χαρακτηριστικού των επιπέδων στον χαρακτήρα του παίκτη και των αντιπάλων. Αυτό σημαίνει ότι, όσο περνάει ο χρόνος και ο παίκτης τερματίζει τις πίστες, ο χαρακτήρας θα λαμβάνει νέα όπλα με περισσότερες δυνατότητες, οι αντίπαλοι θα γίνονται πιο δύσκολοι και η εξόντωση τους θα δίνει μεγαλύτερη βαθμολογία και νέα κρυμμένα αντικείμενα.

Προγραμματιστικά, η τεχνητή νοημοσύνη θα μπορούσε να γίνει ακόμα πιο βέλτιστη με τη χρήση ουράς Fibonacci, η οποία δίνει την δυνατότητα ταχύτερης λειτουργίας των εισαγωγών και διαγραφών των στοιχείων στη λίστα. Ακόμα, ο αλγόριθμος θα μπορούσε να χρησιμοποιηθεί και κατά τη διάρκεια της επίθεσης του αντιπάλου στο χαρακτήρα, ώστε να γίνεται αποφυγή εμποδίων και αντικειμένων. Φυσικά, το παιχνίδι θα μπορούσε να μετατραπεί σε διαδικτυακό, δίνοντας την δυνατότητα σε ένα παίκτη να χειρίζεται τον χαρακτήρα του επιστήμονα και σε κάποιον άλλον να λαμβάνει μέρος ως αντίπαλος σε μια μάχη μεταξύ των δύο.

7.3 Επίλογος

Η εργασία ολοκληρώθηκε και το αποτέλεσμα είναι παραπάνω από συναρπαστικό. Από την παιδική μας ηλικία μέχρι την αρχή της διπλωματικής εργασίας αναρωτιόμασταν πώς δημιουργούνται τα παιχνίδια υπολογιστών. Μέσα σε ένα σύντομο χρονικό διάστημα ξεκινώντας από το μηδέν δημιουργήσαμε ένα παιχνίδι τρισδιάστατων γραφικών με ολοκληρωμένο σενάριο και χρηστικότητα ικανή να μας ικανοποιήσει και να μας ευχαριστήσει. Σίγουρα, δεν μπορεί να συγκριθεί το αποτέλεσμα με τα σύγχρονα παιχνίδια, όμως η διαδικασία έγινε κατανοητή και η γνώση αποκτήθηκε. Χρησιμοποιήσαμε τα ίδια προγράμματα με αυτά που χρησιμοποιούν μεγάλες εταιρίες και επαγγελματίες του χώρου και μάθαμε ότι τα γεγονότα που συμβαίνουν σε ένα παιχνίδι δεν είναι τίποτα άλλο από τα αποτελέσματα της μελέτης των επιστημόνων, των μηχανικών και των καλλιτεχνών που εργάσθηκαν για αυτό, συνδυασμένα με ευφυή τρόπο. Τα κέρδη της βιομηχανίας παιχνιδιών πρόκειται να ξεπεράσουν το έτος 2012 τα 60 δις δολάρια παγκοσμίως, οπότε το μέλλον ανήκει σίγουρα στα γραφικά.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Unreal Development Kit (UDK): http://www.udk.com/

[2] First person shooter: http://en.wikipedia.org/wiki/First-person_shooter

[3] Third person shooter: http://en.wikipedia.org/wiki/Third-person_shooter

[4] History of video games: http://en.wikipedia.org/wiki/History_of_video_games

[5] Video game genres: http://en.citizendium.org/wiki/Video_game_genres

[6] Unity 3D: http://unity3d.com/

[7] Torque 3D: http://www.torquepowered.com/

[8] Russell S., Norvig P. (2005) Τεχνητή νοημοσύνη - Μια σύγχρονη προσέγγιση

[9] Amit's A* pages: http://theory.stanford.edu/~amitp/GameProgramming/

[10] A* search algorithm: http://en.wikipedia.org/wiki/A*_search_algorithm

[11] Cormen T., et al. (2001) Introduction to algorithms, Second edition

[12] Binary heap: http://en.wikipedia.org/wiki/Binary_heap

[13] Μηχανή παιχνιδιού (Game engine): http://en.wikipedia.org/wiki/Game_engine

[14] Epic Games: http://www.epicgames.com/technology/

[15] Busby J., Parrish Z., Wilson J. (2009) Mastering Unreal Technology, Volume I Introduction to Level Design with Unreal Engine 3

[16] Rendering: http://en.wikipedia.org/wiki/Rendering_%28computer_graphics%29

[17] UDK documentation: http://udn.epicgames.com/Three/WebHome.html

[18] Unrealscript reference: http://udn.epicgames.com/Three/UnrealScriptReference.html

[19] Ed. Byrne, (2005) Game level design

[20] DAZ 3D models: http://www.daz3d.com/i/3d_models/0?_m=d

[21] Autodesk 3ds Max: http://usa.autodesk.com/

[22] Dungeon Defense: http://www.udk.com/showcase-dungeon-defense

[23] Using Binary Heaps in A* Pathfinding: http://www.policyalmanac.org/games/binaryHeaps.htm

[24] Questionnaire for User Interaction Satisfaction (QUIS): http://lap.umd.edu/quis/