ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ
ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

3D COMPUTER GAMING USING A GAME ENGINE

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΣΟΥΡΗΣ ΙΣΙΔΩΡΟΣ

Επιβλέπων: Αικατερίνη Μανιά,
Επ. Καθ. Πολυτεχνείου Κρήτης

Χανιά 2011

3D Computer Gaming using a Game Engine

Τσουρής Ισίδωρος

Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

Επιβλέπων: Αικατερίνη Μανιά,
Επικ. Καθ. Πολυτεχνείου Κρήτης

Εξεταστική επιτροπή:

Μανιά Α.	Λαγουδάκης Μ.	Δεληγιαννάκης Α.
Επίκ Καθ.	Επίκ Καθ.	Επίκ Καθ

Χανιά 2011

Περίληψη

Στην παρούσα εργασία αναπτύχθηκε ένα διαδραστικό, τρισδιάστατο παιχνίδι, ενός και μόνο παίκτη (single-player), για προσωπικό υπολογιστή. Χρησιμοποιήθηκε η μηχανή παιχνιδιών Torque η οποία στηρίζεται στην αρχιτεκτονική πελάτη-εξυπηρετητή και επέτρεψε αντικειμενοστρεφή σχεδίαση της εφαρμογής. Το παιχνίδι φέρει τον τίτλο "WaterWar" και ανήκει στην κατηγορία "πρώτου προσώπου βολής " (FPS games), με στοιχεία από παιχνίδια περιπέτειας (adventure games). Ο παίκτης περιπλανιέται στον εικονικό κόσμο του παιχνιδιού και μάχεται χαρακτήρες ελεγχόμενους από τον υπολογιστή. Για την ολοκλήρωση του παιχνιδιού απαιτήθηκε ο σχεδιασμός των αντικειμένων και των κτιριακών δομών της σκηνής του παιχνιδιού, η δημιουργία του εικονικού κόσμου, η μοντελοποίηση των χαρακτήρων και ο σχεδιασμός των ακολουθιών κίνησής τους, η υλοποίηση των γραφικών διασυνδέσεων που επιτρέπουν την επικοινωνία του χρήστη με την εφαρμογή και η ανάπτυξη της τεχνητής νοημοσύνης. Για τον προγραμματισμό της τεχνητής νοημοσύνης χρησιμοποιήθηκε η τεχνική της μηχανής πεπερασμένων καταστάσεων όπου κάθε αντικείμενο το οποίο χειρίζεται ο υπολογιστής ξεκινάει με μία αρχική κατάσταση και ανάλογα με τις συνθήκες μεταβαίνει σε κάποια άλλη. Τέλος αναπτύχθηκε, σε Matlab, ένα βοηθητικό πρόγραμμα με το οποίο παράγονται υψομετρικοί χάρτες. Οι τελευταίοι χρησιμοποιούνται για τη δημιουργία των εδαφών των αποστολών του παιχνιδιού.

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου στην κ. Αικατερίνη Μανιά, επιβλέπουσα καθηγήτρια, για την καθοδήγησή της και τη βοήθεια στην εκπόνηση της παρούσας διπλωματικής. Ευχαριστώ επίσης τους κυρίους καθηγητές της επιτροπής για το χρόνο που αφιέρωσαν και τις παρατηρήσεις τους σχετικά με την εργασία.

Ευχαριστώ επίσης όλους τους καθηγητές και βοηθούς του τμήματος των Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών, του Πολυτεχνείου Κρήτης, που με ενέπνευσαν και που με έμαθαν να σκέφτομαι σαν μηχανικός, ο οποίος για όλη του τη ζωή θα προσπαθεί να βρίσκεται μπροστά από τις τεχνολογικές εξελίξεις.

Τέλος δεν γίνεται να μην ευχαριστήσω τους συμφοιτητές μου εκείνους με τους οποίους μοιράστηκα όλες τις σκέψεις μου, τις στεναχώριες και τις χαρές, κατά τη διάρκεια της διαμονής μου στα Χανιά. Για την Άλκηστη λοιπόν, το Λεωνίδα, το Σωτήρη και το Γιάννη!

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Τα ηλεκτρονικά παιχνίδια σήμερα απέχουν πολύ από την εικόνα που έχει σχηματίσει γι΄ αυτά ο μέσος άνθρωπος. Πρόκειται πλέον για μια ολοκληρωμένη βιομηχανία η οποία συναγωνίζεται σε εισπράξεις και σε δημοτικότητα τις βιομηχανίες κινηματογράφου και μουσικής. Πρόκειται για μια βιομηχανία με προϋπολογισμούς ανάπτυξης παιχνιδιών που φτάνουν τα εκατό εκατομμύρια δολάρια και με τεχνολογία που είναι ικανή να παράγει απέραντους και αλληλεπιδραστικούς τρισδιάστατους κόσμους.

Ακόμα και την περίοδο αυτή, όπου η οικονομική κρίση μαστίζει ολόκληρο τον πλανήτη, η βιομηχανία των βιντεοπαιχνιδιών προχωρά με σταθερά ανοδική πορεία και καταφέρνει να έχει κέρδη ρεκόρ. Το παράδοξο αυτό φαινόμενο οφείλεται στο γεγονός ότι σε αντίθεση με την υπόλοιπη βιομηχανία λογισμικού, η βιομηχανία βιντεοπαιχνιδιών θεωρείται βιομηχανία ψυχαγωγίας! Τα ηλεκτρονικά παιχνίδια αποτελούν σήμερα μέσο ψυχαγωγίας για όλες τις ηλικίες και των δύο φύλων, με σχεδόν ολοκληρωτική διείσδυση στο νεανικό κοινό και τεράστια απήχηση στους ενήλικες.

Έρευνες σε Ευρώπη (ISFE-Interactive Software Federation of Europe) και ΗΠΑ (ESA-Entertainment Software Association) ανατρέπουν πολλά στερεότυπα σχετικά με τον παίκτη βιντεοπαιχνιδιών. Το πρώτο στερεότυπο που διαψεύδεται είναι ότι ο παίκτης βιντεοπαιχνιδιών είναι άνδρας νεαρής ηλικίας. Όλες οι ηλικίες και των δύο φύλων παίζουν παιχνίδια, σε όλες τις πλατφόρμες παιχνιδιών. Φυσικά υπάρχουν προτιμήσεις ανάλογα με το φύλο και την ηλικία.

Πέρα όμως από τη ψυχαγωγική του υπόσταση, ένα ηλεκτρονικό παιχνίδι είναι και ένα προϊόν τεχνολογίας. Απαιτεί εξειδικευμένες γνώσεις προγραμματισμού και καλή κατανόηση της αρχιτεκτονικής της πλατφόρμας παιχνιδιού και των σύνθετων αλγορίθμων και τεχνικών ανάπτυξης λογισμικού.

Στις αρχές της βιομηχανίας βιντεοπαιχνιδιών, η ανάπτυξή του ήταν σχετικά απλή υπόθεση. Για την ανάπτυξη ενός εμπορικά επιτυχημένου βιντεοπαιχνιδιού αρκούσαν η έμπνευση και οι τεχνικές γνώσεις μιας μικρής ομάδας προγραμματιστών. Χαρακτηριστικό είναι το παράδειγμα του παιχνιδιού "*Space Invaders*" (Taito, 1978), το οποίο αναπτύχθηκε εξολοκλήρου από ένα άτομο, σε μια περίοδο 12 μηνών, και έγινε τεράστια επιτυχία εδραιώνοντας τις ιαπωνικές εταιρίες ανάπτυξης βιντεοπαιχνιδιών στην παγκόσμια αγορά.

Τα σύγχρονα παιχνίδια αποτελούν πλέον συλλογική προσπάθεια μιας μεγάλης ομάδας ατόμων με διακριτούς ρόλους και ειδικότητες. Για την ανάπτυξή τους δεν αρκεί το ταλέντο ενός προγραμματιστή. Πλέον μιλάμε για ομάδες ανάπτυξης που αποτελούνται

από σεναριογράφους, καλλιτέχνες τρισδιάστατων μοντέλων, υφών, συνθέτες μουσικής, προγραμματιστές διεπαφών, εργαλείων και γενικά ειδικότητες που θα συμβάλουν στο σχεδιασμό του παιχνιδιού, τη δημιουργία του περιεχομένου του και την ανάπτυξη του κώδικα.

Οι σελίδες που ακολουθούν έρχονται για να συμπληρώσουν το προγραμματιστικό κομμάτι της διπλωματικής εργασίας με θέμα "**3D Computer Gaming using a Game Engine**", όπου πραγματεύεται την ανάπτυξη ενός τρισδιάστατου ηλεκτρονικού παιχνιδιού με χρήση της μηχανής παιχνιδιών Torque.

Το παιχνίδι φέρει τον τίτλο "WaterWar" και βασίζεται στην ταινία μικρού μήκους "World Water War". Βρισκόμαστε σε ένα όχι και τόσο μακρινό μέλλον όπου ένας νέος παγκόσμιος πόλεμος έχει ξεσπάσει. Στα βάθη της αφρικανικής ηπείρου ένοπλες ομάδες έχουν καταφέρει να αποκτήσουν το πολυτιμότερο πλέον αγαθό του πλανήτη, για το οποίο και γίνεται ο πόλεμος: ένα μπουκάλι νερό! Η μόνη ελπίδα για παγκόσμια ειρήνη είναι ο Jack, ένας πρώην γιατρός που έχει εξελιχθεί σε έναν νέο "John Rambo" και καλείται να εντοπίσει τη θέση όπου φυλάσσεται το "πακέτο".

Το πρώτο κεφάλαιο της εργασίας, όπως φάνηκε παραπάνω, περιέχει πληροφορίες για τη βιομηχανία ανάπτυξης των ηλεκτρονικών παιχνιδιών και αποτελεί εισαγωγικό κεφάλαιο για την εργασία.

Το δεύτερο κεφάλαιο ασχολείται με τις μηχανές παιχνιδιών, το πλέον απαραίτητο λογισμικό για την ανάπτυξη ενός βιντεοπαιχνιδιού. Περιγράφονται τα υποσυστήματα των μηχανών παιχνιδιών, παραθέτονται οι σημαντικότερες από αυτές και αναλύεται ο λόγος για τον οποίο χρησιμοποιήθηκε η "Torque Game Engine" στην παρούσα διπλωματική.

Το τρίτο κεφάλαιο έχει να κάνει αποκλειστικά με τη μηχανή "Torque Game Engine". Γίνεται μία πρώτη επαφή με τη μηχανή αλλά και τη γλώσσα σεναρίων Torque.

Στο τέταρτο κεφάλαιο περιγράφεται το παιχνίδι που αναπτύχθηκε. Πρόκειται για τη φάση της σχεδίασης όπου καταγράφονται αναλυτικά με χαρτί και μολύβι όλα όσα πρέπει να αναπτύξει ο προγραμματιστής.

Το πέμπτο κεφάλαιο είναι καθαρά καλλιτεχνικό και έχει να κάνει με την ανάπτυξη των τρισδιάστατων μοντέλων, και όχι μόνο, που αποτελούν τα δεδομένα του παιχνιδιού. Στο ίδιο κεφάλαιο ξεχωρίζει η ανάπτυξη του "TerrainGenarator", ενός βοηθητικού προγράμματος που αποσκοπεί στην εύκολη δημιουργία εδαφών για το παιχνίδι.

Το έκτο κεφάλαιο είναι η ανάπτυξη του κώδικα της εφαρμογής. Εκεί αναλύονται θέματα όπως η αρχιτεκτονική "πελάτη-εξυπηρετητή", ο προγραμματισμός των διεπαφών, του συστήματος καταγραφής αλλά και της τεχνική νοημοσύνης.

Το έβδομο κεφάλαιο είναι καταγραφή των αποτελεσμάτων, των παρατηρήσεων κατά την ανάπτυξη του παιχνιδιού και των μελλοντικών προεκτάσεων της εφαρμογής.

Κεφάλαιο 2

Μηχανές Παιχνιδιών

Στο κεφάλαιο αυτό θα γίνει μια γενική παρουσίαση των μηχανών παιχνιδιών. Γίνεται αναφορά στο πως φτάσαμε να τις χρησιμοποιούμε για τη δημιουργία παιχνιδιών, για το τι είναι μηχανή παιχνιδιού, τι σχέση έχει με τα γνωστά στους περισσότερους DirectX/OpenGL, με ποια κριτήρια την επιλέγει κανείς και ποια είναι η μηχανή που χρησιμοποιείται στην παρούσα διπλωματική εργασία.

2.1 Ανακαλύπτοντας τον "τροχό"

Το 1961 έκανε την εμφάνισή του το "Spacewar". Αποτέλεσε τον προπομπό για αυτό που καλούμε σήμερα ηλεκτρονικό παιχνίδι. Ο δημιουργός του ακούει στο όνομα Steve Russel, φοιτητής του M.I.T. Αυτό που σκέφτηκε δεν ήτανε τίποτε άλλο από το να "τυπώνει" στην οθόνη απλούς χαρακτήρες ASCII, με κατάλληλη διάταξη και συχνότητα, ώστε να δίνουν την εντύπωση αντικειμένων σε κίνηση.

Σχήμα 2.1: Το Spacewar ενώ τρέχει σε έναν PDP-1

Λίγα χρόνια αργότερα, τα ηλεκτρονικά παιχνίδια θα γίνουν γνωστά στον περισσότερο κόσμο χάρη στην πρώτη "παιχνιδομηχανή": πρόκειται για την κονσόλα "Atari" που κατασκευάστηκε το 1972 . Την ίδια χρονιά στην κυκλοφορία δίνεται το σπουδαιότερο δείγμα απλού παιχνιδιού στην ιστορία, το "Pong". Το "Pong" ήταν το πρώτο παιχνίδι του οποίου τα δικαιώματα αγοράστηκαν από εταιρεία κονσόλας και επίσης το πρώτο παιχνίδι που όρισε βασικούς αλγορίθμους γύρω από την κίνηση των γραφικών.

Φτάνουμε στην δεκαετία του '80. Εκεί συναντάμε το πιο διαχρονικό παιχνίδι όλων των εποχών! Όπως σωστά μαντέψατε πρόκειται για το "Pac-Man", μια ιδέα του Tohru Iwatani εμπνευσμένη από ένα κομμάτι που έλειπε από την πίτσα που είχε μπροστά του! Το "Pac-Man" έχει παιχτεί πάνω από 10 δισεκατομμύρια φορές στην εικοσάχρονη πορεία του, ενώ κυκλοφόρησαν και συνεχίζουν να κυκλοφορούν διάφορες εκδόσεις του.

Η δεκαετία του '80 είναι η εποχή όπου αρχίζουν να εξελίσσονται οι παιχνιδομηχανές αλλά και οι ηλεκτρονικοί υπολογιστές. Φυσικό επακόλουθο και η εξέλιξη των παιχνιδιών. Δελεαστικοί και πολύχρωμοι τίτλοι έκαναν την εμφάνιση τους σε υπολογιστές όπως "Commodore", "Amstrad", "Amiga". Εξακολουθεί όμως η ανάπτυξη ενός

παιχνιδιού να είναι "επώδυνη" διαδικασία για τους προγραμματιστές. Πέρα από τους περιορισμούς λόγω υλικού, ο κώδικας (που και στα πιο απλά παιχνίδια ήτανε πολύπλοκος) έχανε την αξία του μετά την έκδοση. Οι νεότερες διανομές ακόμα και του ίδιου του παιχνιδιού, λόγω της ανάπτυξης της τεχνολογίας, χρησιμοποιούσαν νέες τεχνικές σχεδίασης και έκαναν χρήση επιπλέον πόρων.

"Το νερό όμως έχει μπει στο αυλάκι" και στα μέσα του 1990 αρχίζουμε να μιλάμε για πέμπτη γενιά[1] παιχνιδιών. Το 1993 η "id Software" δίνει στην κυκλοφορία το "Doom". Πρόκειται για ένα παιχνίδι σταθμό στην κατηγορία των " Πρώτου Προσώπου Βολής " (FPS-First Person Shooting). Τα τρισδιάστατα γραφικά και η δυνατότητα δικτύωσης του ήταν αυτά που ξεχώρισαν. Για όλους όμως όσους ασχολούνται σήμερα με την ανάπτυξη παιχνιδιών, το Doom χαρακτηρίζεται ως πρωτοποριακό για έναν ακόμα λόγο. Εισήγαγε μια νέα μέθοδο δημιουργίας παιχνιδιών: τη μηχανή παιχνιδιών ("Game Engine")! Για την ιστορία, η μηχανή που χρησιμοποιήθηκε ήταν η Doom Engine.

Σχήμα 2.2: Doom από την id Software

Τί το επαναστατικό όμως έφερε ή νέα μέθοδος! Προγραμματιστές αλλά και γενικά όσοι ασχολούνταν με παιχνίδια υπολογιστών, μπορούσαν να "πειράξουν" τον πυρήνα του "Doom" και να δημιουργήσουν ένα νέο παιχνίδι με δικά τους μοντέλα, σκηνικά, μουσική. Οι εταιρείες άρχισαν να γράφουν μηχανές ή να στηρίζονται στις ήδη υπάρχουσες για την ανάπτυξη των νέων τίτλων. Ενδεικτικά τα "Unreal Tournament", "Tom Clancy's Splinter Cell", "BioShock" έχουν ως βάσης τους την Unreal Engine της Epic Games.

Σήμερα δεν νοείται ανάπτυξη ηλεκτρονικού παιχνιδιού χωρίς χρήση κάποιας μηχανής. Αυτό σε καμία περίπτωση δεν σημαίνει ότι πρέπει να γράψουμε τη δική μας για να δημιουργήσουμε το πρώτο μας παιχνίδι. Δεν χρειάζεται να "ανακαλύψουμε και πάλι τον τροχό"! Στην ιστοσελίδα
http://wiki.gamedev.net/index.php/Game_Engines υπάρχει μία λίστα με μηχανές παιχνιδιών. Τη στιγμή που ξεκίνησε το παρόν κείμενο υπήρχαν καταγεγραμμένες παραπάνω από 100!

2.2 Τί είναι μία μηχανή παιχνιδιού (Game Engine)

Μηχανή παιχνιδιού είναι ο πυρήνας (από τη σκοπιά του λογισμικού) ενός ηλεκτρονικού παιχνιδιού, που επιτρέπει την εκτέλεσή του. Βασικά πρόκειται για την συλλογή τμημάτων κώδικα (modules) που συνεργάζονται για να "τρέξει" το παιχνίδι.

Πού σταματάει όμως η μηχανή και πού ξεκινάει το παιχνίδι; Πολλοί είναι αυτοί που ταυτίζουν τις δύο έννοιες. Η μηχανή είναι αυτό που λέμε η "καρδιά και το μυαλό". Είναι αυτό που σου επιτρέπει να κινείς το χαρακτήρα σου, να βλέπεις, να ακούς, να αλληλεπιδράς με αντικείμενα. Περικλείει κώδικα και συναρτήσεις για να διευκολύνει την ανάπτυξη του παιχνιδιού. Ας φανταστούμε το εξής απλό σενάριο: χειριζόμαστε μια

[1] 1η γενιά:1972-1977, 2η γενιά:1976-1984, 3η γενιά:1983-1992, 4η γενιά:1987-1996, 5η γενιά:1993-2002, 6η γενιά:1998-2006, 7η γενιά:2004-...

φιγούρα που εισέρχεται σε μια σπηλιά και συγκεντρώνει χρυσά νομίσματα που βρίσκει. Το παιχνίδι, σε αυτήν τη περίπτωση, είναι η ιδέα για το τι πρέπει να κάνουμε, τι θα γίνει αφού συγκεντρώσουμε τα νομίσματα, τι χαρακτήρες υπάρχουν σε αυτό το στάδιο καθώς και ό,τι άλλο βλέπουμε όπως δάδες φωτιάς στη σπηλιά, ο χρόνος που απομένει, ο μετρητής νομισμάτων. Στο ίδιο σενάριο, η μηχανή είναι οι κρυμμένοι εκείνοι μηχανισμοί που σου δίνουν τη δυνατότητα να χρησιμοποιήσεις το πληκτρολόγιο για να κινήσεις το χαρακτήρα, που ελέγχουν ότι δεν μπορείς να περάσεις μέσα από εμπόδια που βρίσκονται στο δρόμο σου, που σχεδιάζουν τη σκηνή έτσι ώστε οι ακτίνες του ήλιου να ανταναμλούν στη λιμνούλα δίπλα στην είσοδο της σπηλιάς!

Καταλάβαμε λοιπόν ότι μηχανή παιχνιδιού και παιχνίδι δεν είναι το ίδιο και το αυτό. Ας δούμε όμως τώρα το λόγο που η πρώτη αποτελεί αναπόσπαστο κρίκο στην αλυσίδα ανάπτυξης ενός ηλεκτρονικού παιχνιδιού. Από μόνο του ένα παιχνίδι δεν είναι τίποτα περισσότερο από ένα πρόγραμμα (πιο εξειδικευμένης μορφής βέβαια), γραμμένο σε κάποια γλώσσα προγραμματισμού. Αυτό σημαίνει ότι μπορούμε να "δουλέψουμε" πάνω του όπως θα κάναμε σε κάποια κοινή εφαρμογή. Ωστόσο ορισμένες από τις εργασίες που πρέπει να γίνουν, θα τις ξανασυναντήσουμε σε κάθε απόπειρα δημιουργίας ενός νέου παιχνιδιού. Για παράδειγμα θα αναγκαστούμε να συμπεριλάβουμε κώδικα σχετικό με βασικές λειτουργίες του λειτουργικού συστήματος ή κώδικα που θα υλοποιεί ένα μοντέλο "πελάτη-εξυπηρετητή" και θα επιτρέπει τη δικτύωση του. Γιατί να προγραμματίζουμε τα ίδια πράγματα σε κάθε νέα σειρά του τίτλου μας! Αυτό είναι και το μεγάλο πλεονέκτημα της ανάπτυξης παιχνιδιών με τη βοήθεια μηχανής. Συγκεντρώνουμε όλα τα κοινά, λειτουργικά στοιχεία των παιχνιδιών μας, τα "γράφουμε" μία φορά και τα "κρύβουμε" σε ένα μαύρο, για το παιχνίδι, κουτί. Όμως εξίσου σημαντικό θεωρείται και το παρακάτω. Οι περισσότερες μηχανές επιτρέπουν στα παιχνίδια να εκτελεστούν σε οποιαδήποτε πλατφόρμα! Με τον όρο πλατφόρμα εννοούμε είτε κάποια κονσόλα (π.χ. XBox) είτε προσωπικούς υπολογιστές και κατά επέκταση ένα από τα γνωστά λειτουργικά συστήματα (MS Windows, Linux, Mac OS X).

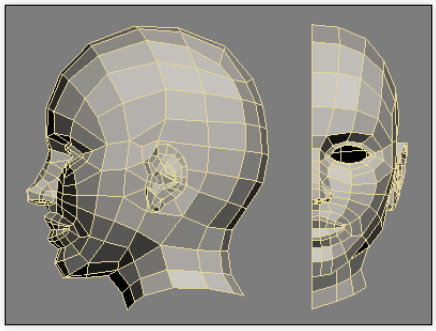
Καιρός να "συμμαζέψουμε" όλα τα παραπάνω, και να κρατήσουμε ότι:
μία Μηχανή Παιχνιδιού είναι η οργάνωση κώδικα που επιτρέπει το διαχωρισμό των γενικών λειτουργιών ενός παιχνιδιού, από τις πιο εξειδικευμένες λειτουργίες του. Σήμερα οι μεγάλες εταιρείες του χώρου χρησιμοποιούν τη δική τους μηχανή, την οποία και έχουν αναπτύξει σταδιακά και αφού έχουν εντοπίσει τα βασικά γνωρίσματα των εφαρμογών τους. Μάλιστα πολλές από αυτές έχουν κατοχυρώσει τα δικαιώματά τους πάνω στον κώδικα, έτσι ώστε όσοι θέλουν να χρησιμοποιήσουν τη μηχανή να πρέπει να καταβάλουν κάποιο αντίτιμο. Ως παράδειγμα αναφέρουμε την "Epic Games", όπου κοστολογεί την "Unreal Engine 2" στο ποσό των τριακοσίων πενήντα χιλιάδων δολαρίων! Από την άλλη υπάρχουν μηχανές "ανοιχτού κώδικα". Ο κώδικας τους είναι διαθέσιμος στο διαδίκτυο και ο καθένας είναι ελεύθερος να τον προσαρμόσει στις ανάγκες του, χωρίς κανένα κόστος.

2.3 Τα βασικά μέρη μίας μηχανής παιχνιδιού

Μία τυπική μηχανή αποτελείται από τρία τμήματα, τρία υποσυστήματα: τη "**μηχανή απόδοσης**" ("Renderer"), τη "**φυσική μηχανή**" ("Physical Engine") και το "**γράφο σκηνής**" ("Scene Graph").

2.3.1 Μηχανή απόδοσης (Renderer)

Η μηχανή απόδοσης είναι εκείνη όπου υπολογίζει πως εμφανίζεται η σκηνή στον παίκτη/παρατηρητή και πως η απαραίτητη πληροφορία για αυτό αποθηκεύεται στη μνήμη.



Σχήμα 2.3: Τρισδιάστατο μοντέλο σχεδιασμένο με μικρό αριθμό πολυγώνων

Για να γίνει κατανοητός ο ορισμός, ας φέρουμε στο μυαλό μας το πιο απλό τρισδιάστατο μοντέλο, τον κύβο. Ο κύβος αποτελείται από οκτώ σημεία (vertices), ένα σε κάθε μία από τις οκτώ γωνίες του. Τα σημεία ενώνονται, ανά δύο, με ευθείες γραμμές (edges) και σχηματίζονται έξι επιφάνειες (surfaces). Κάθε επιφάνεια διαιρείται σε δύο πολύγωνα (σε δύο τρίγωνα για την ακρίβεια). Επιπλέον μπορεί ο κύβος να έχει τη μία του επιφάνεια "διαφανή" ή να έχει κάποια συγκεκριμένη "ύφή" (texture). Όλα τα παραπάνω καταγράφονται και αποθηκεύονται από τη μηχανή απόδοσης. Χρησιμοποιώντας την πληροφορία αυτή και με τη βοήθεια της κάρτας γραφικών, όπως θα δούμε παρακάτω, το εν λόγω υποσύστημα δημιουργεί πολύγωνα (βλ. σχήμα 2.3) και σχεδιάζει τελικά το τρισδιάστατο μοντέλο του κύβου.

Με τον ίδιο τρόπο σχηματίζονται όλα τα μοντέλα της σκηνής. Στη συνέχεια η μηχανή απόδοσης κάνει υπολογισμούς για το φως που διαχέεται πάνω τους, για το αν υπάρχουν αντανακλάσεις στις επιφάνειες, σε ποια αντικείμενα θα εμφανίζεται σκιά και πού, με βάση τις φωτεινές πηγές, και ολοκληρώνει το πολυδιάστατο έργο της με την διαδικασία της διαλογής (culling).

Διαλογή (Culling)

Οι μοντέρνες κάρτες γραφικών είναι σχεδιασμένες ώστε να χειρίζονται και να εμφανίζουν χιλιάδες πολύγωνα το δευτερόλεπτο. Όμως για όλα υπάρχει κάποιος περιορισμός.

Έστω λοιπόν ένας τρισδιάστατος εικονικός κόσμος αποτελούμενος από χιλιάδες πολύγωνα. Σε κάποιο σημείο του βρίσκεται ο χαρακτήρας "πρώτου προσώπου" που χειριζόμαστε και παρατηρεί προς μία κατεύθυνση. Στην κατεύθυνση αυτή υπάρχουν μοντέλα (δηλαδή πολύγωνα) που είναι ορατά αλλά και μοντέλα που δεν φαίνονται (για παράδειγμα ένας άλλος χαρακτήρας που βρίσκεται πίσω από κάποιο πέτρινο τοίχο). **Με τη διαδικασία της διαλογής, επιλέγονται και σχεδίαζονται μόνο τα ορατά, για τον παρατηρητή, πολύγωνα**! Επιτυγχάνεται έτσι καλύτερη ταχύτητα στο παιχνίδι και υψηλός βαθμός καρέ[2] (frame rate).

2.3.2 Φυσική μηχανή (Physical Engine)

Η φυσική μηχανή είναι υπεύθυνη για τις περισσότερες λειτουργίες της μηχανής παιχνιδιού. Είναι το υποσύστημα εκείνο που χειρίζεται τις συγκρούσεις, τη δικτύωση, τον ήχο

[2]Πολλοί θεωρούν ότι ένας βαθμός καρέ κοντά στα 30 fps είναι ικανοποιητικός για κάποιο παιχνίδι. Ωστόσο αυτό είναι σχετικό. Για παράδειγμα τα γραφικά του "Call of Duty 4: Modern Warfare" αναδεικνύονται όταν το μηχάνημά μας έχει τη δυνατότητα να επιτύχει 60 fps.

και τα κινούμενα γραφικά, τη γλώσσα δέσμης ενεργειών και τη τεχνική νοημοσύνη.

Ανίχνευση συγκρούσεων (Collision Detection)

Πρόκειται για τη διαδικασία με την οποία ελέγχουμε αν μέσα στον κόσμο του παιχνιδιού, δύο ή παραπάνω αντικείμενα έρχονται σε επαφή. Η σύγκρουση συνεπάγεται κάποιου είδους διαδραστικότητας αντικειμένων στο χώρο αυτό.

Ήχος (Sound)

Ο ήχος παίζει σημαντικό ρόλο στα μοντέρνα παιχνίδια μιας και συμβάλει στη δημιουργία της ατμόσφαιρας που θέλει να πετύχει ο δημιουργός. Η φυσική μηχανή συγκεντρώνει και αποθηκεύει ήχους και καθορίζει στο αντίστοιχο υλικό πότε να τους αναπαράγει.

Ωστόσο η ενέργεια αυτή δεν είναι τόσο απλή όσο ακούγεται. Όπως συμβαίνει και με το φως, ο ήχος επηρεάζεται από το περιβάλλον. Έτσι αν θέλουμε να έχουμε ένα ρεαλιστικό παιχνίδι πρέπει, για παράδειγμα, η μηχανή να παράγει έναν αμυδρό ήχο που προέρχεται από το βάθος ενός σκοτεινού τούνελ και όσο ο χαρακτήρας πλησιάζει προς την πηγή του να γίνεται πιο καθαρός. Επιπλέον τα πετρώματα που υπάρχουν στο τούνελ, θα κάνουν τον ήχο να ακούγεται διαφορετικός. Για τα παραπάνω άλλα και για ενέργειες όπως η συμπίεση και η αποσυμπίεση των αρχείων ήχου είναι υπεύθυνο το εν λόγω υποσύστημα.

Γλώσσα δέσμης ενεργειών (Scripting Language)

Μία "γλώσσα δέσμης ενεργειών" είναι και αυτή μία γλώσσα προγραμματισμού όπως η C, C++, με τη βασική διαφορά ότι δεν μεταφράζεται (not compiled). Για το λόγο αυτό τα τμήματα ενός παιχνιδιού που έχουν προγραμματιστεί με "γλώσσα δέσμης ενεργειών" είναι πιο αργά από τα παραδοσιακά τμήματα που έχουν γραφτεί, για παράδειγμα, με C++. Από την άλλη όμως ο προγραμματιστής μπορεί να τροποποιεί ένα "σενάριο" (script) όσες φορές θέλει χωρίς να είναι αναγκασμένος να περιμένει το "μεταφραστή" (compiler) να ολοκληρώσει τη μετάφραση. Επίσης θα μπορούσε να υποστηρίξει κάποιος ότι οι "γλώσσες σεναρίων" είναι αρκετά ευέλικτες μιας και δεν χρειάζεται να δηλώνονται τύποι δεδομένων, να αρχικοποιούνται μεταβλητές ή να λαμβάνεται υπόψιν ο χώρος που πρέπει να δεσμευτεί στη μνήμη για κάποιο αντικείμενο.

Στα σημερινά παιχνίδια η χρήση "γλωσσών σεναρίων" είναι σχεδόν κανόνας. **Τα "σενάρια" χρησιμοποιούνται για να ενώσουν τα διαφορετικά τμήματα μιας μηχανής έτσι ώστε να έχουμε ένα πλήρες λειτουργικό παιχνίδι**. Για παράδειγμα με "σενάρια" υλοποιείται ο έλεγχος και η συμπεριφορά του χαρακτήρα, ο χρόνος που απομένει για να ολοκληρωθεί το παιχνίδι, οι βαθμοί που κερδίζει ο παίκτης.

Κινούμενα γραφικά σχέδια (Animations)

Ένα τρισδιάστατο μοντέλο μπορεί να είναι σχεδιασμένο έτσι ώστε να μην ξεχωρίζει από το πραγματικό. Ωστόσο αν η κίνηση του είναι "φτωχή", τότε όλη η δουλειά στη σχεδίαση είναι άδικος κόπος!

Στα παιχνίδια υπολογιστών έχουμε δύο κατηγορίες κινουμένων γραφικών: τα "βασισμένα στο πλέγμα[3]" κινούμενα γραφικά (mesh based animation) και τα "σκελετώδη" (skeletal animation). Η πρώτη κατηγορία είναι ο παραδοσιακός τρόπος κινουμένων σχεδίων. Ο σχεδιαστής καθορίζει για κάθε καρέ (frame) τη νέα θέση των σημείων των πολυγώνων. Εκτός το ότι η διαδικασία αυτή απαιτεί πολύ χρόνο, τα εν λόγω γραφικά δεν ενδείκνυνται για "φυσικά"/"ζωντανά" γραφικά! Σε αυτήν την περίπτωση χρησιμοποιούμε τα "σκελετώδη" κινούμενα γραφικά, όπου σε κάθε μοντέλο δίνουμε έναν αριθμό από "όστά" (όπως σε έναν πραγματικό σκελετό). Κάθε φορά που κινείται ένα από αυτά, η μηχανή υπολογίζει πως επηρεάζονται τα υπόλοιπα.

Δικτύωση (Networking)

Σήμερα τα περισσότερα παιχνίδια δίνουν τη δυνατότητα σε πολλούς χρήστες να συνδεθούν σε κάποιο δίκτυο και να παίξουν μεταξύ τους. Έτσι δεν είναι λίγες οι μηχανές που φέρουν τα απαραίτητα πρωτόκολλα για το σκοπό αυτό (η πλειοψηφία χρησιμοποιεί τα TCP- Transmission Control Protocol και UDP-User Datagram Protocol). Μάλιστα οι περισσότερες υλοποιούν το μοντέλο "**πελάτη-εξυπηρετητή**" (**Client-Server architecture**). Ο πελάτης και ο εξυπηρετητής μπορεί να βρίσκονται είτε στο ίδιο μηχάνημα, είτε σε διαφορετικά μηχανήματα συνδεδεμένα σε δίκτυο. Το συγκεκριμένο μοντέλο επιτρέπει:

- την αναμενόμενη δικτύωση πολλαπλών χρηστών (multiplayers),
- την ανάπτυξη παιχνιδιών και για ένα άτομο (single-player) και για παραπάνω, (multiplayers) χωρίς επιπλέον προγραμματισμό στον κώδικα του παιχνιδιού,
- τη σωστή οργάνωση των λειτουργιών της μηχανής (για παράδειγμα η μηχανή παιχνιδιών Torque, της GarageGames, διαχειρίζεται στον εξυπηρετητή τα "συστατικά" όπως χαρακτήρες, οχήματα, και αφήνει στον πελάτη τις διασυνδέσεις),
- την αποφυγή κάποιος παίκτης να έχει πρόσβαση σε κώδικα που τροποποιώντας τον να καταφέρει να "κλέψει".

Τεχνητή νοημοσύνη (Artificial Intelligence)

Για μία μηχανή παιχνιδιού, ο όρος τεχνητή νοημοσύνη αναφέρεται στις τεχνικές εκείνες που δημιουργούν την ψευδαίσθηση ότι οι χαρακτήρες που δεν χειρίζονται από τον παίκτη έχουν την ευφυΐα να προβούν σε κάποιες ενέργειες. Συνήθως οι τεχνικές αυτές προέρχονται από την επιστήμη της τεχνητής νοημοσύνης. Ωστόσο χρησιμοποιούνται και αλγόριθμοι από θεωρία ελέγχου, ρομποτική και επιστήμη υπολογιστών γενικώς.

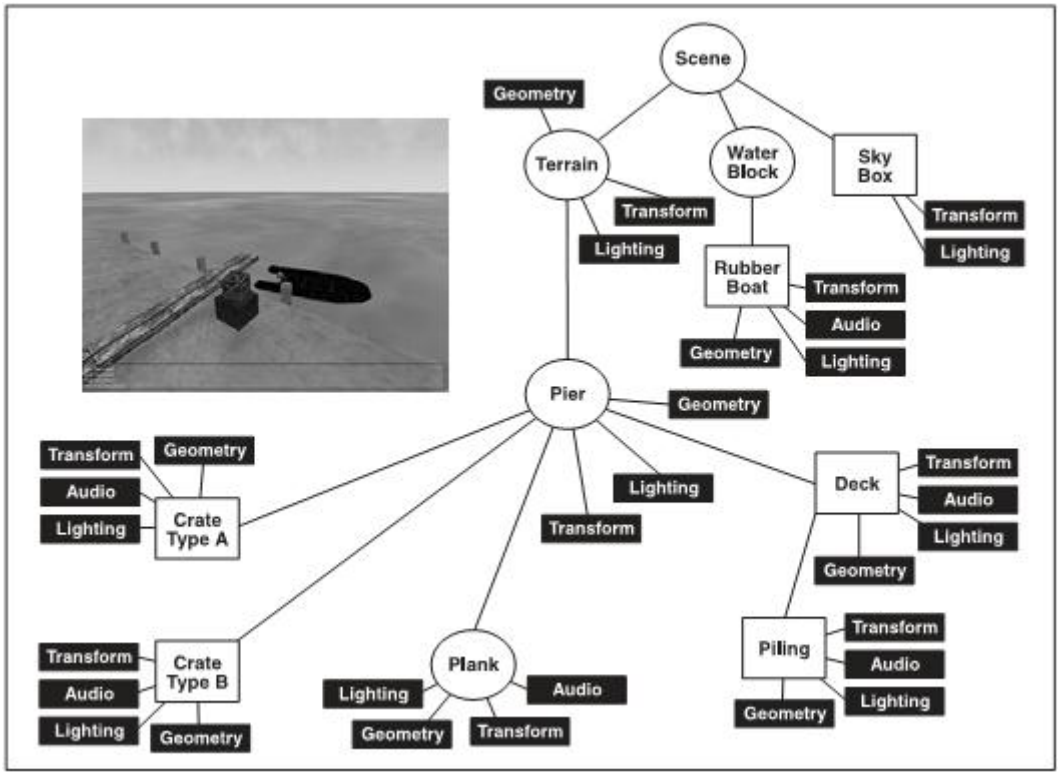
Αυτό που πρέπει να τονίσουμε είναι ότι η τεχνητή νοημοσύνη των παιχνιδιών, ηθελημένα δεν προσεγγίζει την ανθρώπινη λογική. Στόχος των παιχνιδιών είναι η διασκέδαση, οπότε γίνονται κάποιες υποχωρήσεις ως προς την "έξομοίωση" της ανθρώπινης συμπεριφοράς από τη φυσική μηχανή.

[3]Πλέγμα είναι η συλλογή των σημείων, των ακμών και των επιφανειών όπου καθορίζουν ένα πολυεδρικό αντικείμενο.

2.3.3 Γράφος σκηνής (Scene Graph)

Για να μπορέσει η μηχανή να σχεδιάσει τρισδιάστατα αντικείμενα πρέπει να ξέρει τη διάταξή τους στον εικονικό κόσμο. Χρειάζεται επίσης να κρατάει πληροφορίες όπως αλλαγές στην κατάσταση των μοντέλων και στον προσανατολισμό τους.

Ο γράφος σκηνής είναι ο μηχανισμός εκείνος που περιέχει την παραπάνω πληροφορία. **Είναι μια ειδική μορφή κατευθυνόμενου γράφου που κρατάει τα δεδομένα για όλες τις οντότητες του εικονικού κόσμου, σε δομές που καλούνται κόμβοι (nodes)**. Ως οντότητα εννοούμε οποιοδήποτε συστατικό του παιχνιδιού με πιο συνηθισμένες τα τρισδιάστατα σχήματα, τους ήχους και τα φώτα.



Σχήμα 2.4: Παράδειγμα γράφου σκηνής

Όταν φτάσει η στιγμή της σχεδίασης η μηχανή κάνει προσπέλαση του γράφου, χρησιμοποιώντας δείκτες, για να μετακινηθεί στον επόμενο προς σχεδίαση κόμβο κάθε φορά. Εκεί εφαρμόζει τις ενέργειες που πρέπει να γίνουν και σταδιακά σχηματίζεται ο κόσμος του παιχνιδιού.

2.4 OpenGL, DirectX και η σχέση τους με μία μηχανή παιχνιδιού

Καλούμε διεπαφή προγραμματισμού εφαρμογών (Application Programming Interface-API) τη διασύνδεση των προγραμματιστικών διαδικα-

σιών που ένα λειτουργικό σύστημα, βιβλιοθήκη ή εφαρμογή παρέχει προκειμένου να επιτρέπει να γίνονται προς αυτό αιτήσεις από άλλα προγράμματα και/ή ανταλλαγή δεδομένων.

Κάποιες φορές ένας ορισμός από μόνος του δημιουργεί περισσότερα ερωτήματα από αυτά που προσπαθεί να εξηγήσει. Κάτι τέτοιο συμβαίνει και στην περίπτωση της διεπαφής προγραμματισμού εφαρμογών (διεπαφή για συντομία). Μία σχετική με τον όρο έρευνα, στις κοινότητες του διαδικτύου, αναδεικνύει πλήθος χρηστών που ζητάνε "χειροπιαστή" ερμηνεία!

Έστω λοιπόν η γλώσσα προγραμματισμού Java. Με την Java γράφουμε τον πηγαίο μας κώδικα (καθορίζει τους βασικούς τύπους δεδομένων, με ποιες εντολές κάνουμε επανάληψη, πως επιστρέφουμε μία τιμή κλπ.). Έτσι όμως υλοποιούμε μόνο βασικές λειτουργίες! Τί γίνεται αν θέλουμε να δημιουργήσουμε ένα γραφικό περιβάλλον με παράθυρα; Θα πρέπει κάποια τμήματα του κώδικά μας να αφορούν το λειτουργικό σύστημα, οπότε μιλάμε για μεγαλύτερο κώδικα και πολυπλοκότητα. Για αυτό το λόγο όλες οι σύγχρονες υλοποιήσεις γλωσσών μας παρέχουν έτοιμα τμήματα κώδικα με τη μορφή βιβλιοθηκών/εργαλείων. Τί σχέση έχουν τώρα όλα αυτά με τον όρο διεπαφή; Η διεπαφή είναι στην ουσία ο τρόπος με τον οποίο επικοινωνούμε με τις βιβλιοθήκες, δηλαδή οι συναρτήσεις, οι μέθοδοι, τα αντικείμενα που μπορούμε να καλέσουμε ώστε να χρησιμοποιήσουμε τις δυνατότητες που μας παρέχουν οι βιβλιοθήκες.

Αυτό ακριβώς είναι και τα DirectX και OpenGL. Συλλογές εντολών που παρέχουν το μέσο εκείνο με το οποίο μία μηχανή παιχνιδιού (γενικώς κάποια εφαρμογή) θα αποκτήσει πρόσβαση στην κάρτα γραφικών. Η μηχανή απόδοσης, μέσω της διεπαφής, στέλνει τις απαραίτητες πληροφορίες (θέσεις σημείων, συναρτήσεις ευθειών και κυρτών γραμμών κ.α.) στην κάρτα έτσι ώστε η τελευταία να ξέρει με ποιο τρόπο να ενώσει σημεία για να σχεδιάσει τριγωνικά πολύγωνα και να προκύψει το τελικό τρισδιάστατο μοντέλο.

Σήμερα, στο χώρο ανάπτυξης παιχνιδιών έχουν επικρατήσει οι δύο διεπαφές που προαναφέραμε. Η DirectX αναπτύχθηκε από την Microsoft και υποστηρίζεται μόνο από τα λειτουργικά συστήματά της (όχι μόνο των προσωπικών υπολογιστών αλλά και των παιχνιδομηχανών Xbox και Xbox 360). Αντιθέτως η OpenGL της Silicon Graphics, είναι διαθέσιμη σε πολλές διαφορετικές πλατφόρμες. Ωστόσο άλλες σημαντικές διαφορές μεταξύ των δύο δεν υπάρχουν. Αξίζει όμως να σημειωθεί ότι ενώ αρχικά η DirectX υστερούσε ως προς την έτερη διεπαφή, στις μέρες μας λόγω της συνεργασίας της Microsoft με εταιρείες παραγωγής καρτών γραφικών φαίνεται να είναι πιο εξελίξιμη από την OpenGL!

2.5 Κριτήρια επιλογής μηχανής

Η δημιουργία μηχανής παιχνιδιού είναι μία πολύπλοκη διαδικασία. Απαιτεί ιδιαίτερες προγραμματιστικές ικανότητες και γνώσεις, πολύ χρόνο (μέσα στον οποίο συνεχώς θα πρέπει να λαμβάνονται υπόψιν οι αλλαγές σε υλικό και διεπαφές) και εκτός αυτού το τελικό αποτέλεσμα ενδέχεται να μην είναι το επιθυμητό. Ακόμα και οι μεγαλύτερες εταιρείες ανάπτυξης παιχνιδιών χρησιμοποιούν ήδη υπάρχουσες μηχανές. Οπότε δεν θα πρέπει να υπάρχουν ενδοιασμοί από τους νέους στο χώρο για τη χρήση αυτών. Το μόνο για το οποίο θα πρέπει να ανησυχούν είναι η σωστή επιλογή της μηχανής. Ακολουθούν τα βασικότερα κριτήρια για την επιλογή μίας μηχανής παιχνιδιών.

2.5.1 Άδεια χρήσης και κόστος

Στην αρχή του κεφαλαίου δόθηκε ο ιστοχώρος http://wiki.gamedev.net/index.php/Game_Engines όπου περιέχει μία λίστα με μηχανές παιχνιδιών. Για κάθε μηχανή υπάρχουν πληροφορίες όπως η γλώσσα προγραμματισμού, η διεπαφή που χρησιμοποιεί, η υποστήριξη δικτύωσης και η άδεια χρήσης.

Μελετώντας την τελευταία αυτή στήλη θα βρούμε μηχανές με το χαρακτηριστικό "free" ή "open source". Αυτό σημαίνει ότι μπορεί οποιοσδήποτε να τις αποκτήσει χωρίς κόστος, να χρησιμοποιήσει αυτούσιο τον κώδικα ή να τον τροποποιήσει ώστε να καλύπτει τις ανάγκες του. Δεν έχει κανέναν περιορισμό. Ωστόσο αν θέλει να ακολουθήσει το πνεύμα του 'ανοιχτού λογισμικού" καλό θα είναι οι παρατηρήσεις του, οι βελτιώσεις στον κώδικα να κοινοποιούνται στην κοινότητα της μηχανής!

Κάποιες άλλες μηχανές βρίσκονται υπό την προστασία αδειών τύπου GPL/LPGL. Οι άδειες αυτές υποχρεώνουν τη δημοσίευση του κώδικα της εφαρμογής που αναπτύχθηκε με τη μηχανή. Επίσης τα παιχνίδια που στηρίζονται σε αυτές μπορούν να αντιγράφονται χωρίς την άδεια του δημιουργού.

Τέλος έχουμε τις μηχανές για εμπορικούς σκοπούς. Το κόστος αυτών δεν είναι σταθερό και εξαρτάται από αρκετούς παράγοντες. Για παράδειγμα μπορεί:

- να είναι ελεύθερες για χρήση αλλά η πρόσβαση στον κώδικα να απαιτεί κάποιο αντίτιμο,
- να έχουν σχετικά χαμηλή τιμή αλλά να εξαρτώνται από ακριβά προγράμματα σχεδίασης μοντέλων,
- να απαιτούν επί πλέον χρήματα για τις ενημερώσεις,
- να χρειάζεται η αγορά αδειών για κάθε προγραμματιστή μιας ομάδας,
- να υπάρχει άλλη τιμή για μεγάλες εταιρείες παραγωγής παιχνιδιών και άλλη για ανεξάρτητους προγραμματιστές.

Περισσότερες πάντως πληροφορίες για τα δικαιώματα και τις υποχρεώσεις από τη χρήση μιας μηχανής, υπάρχουν στο εκάστοτε συμφωνητικό αδείας που μπορούμε να βρούμε στις επίσημες ιστοσελίδες αυτών.

2.5.2 Σχεδίαση παιχνιδιού

Οι απαιτήσεις σε σχεδίαση που έχουμε για το παιχνίδι επηρεάζουν και την επιλογή της μηχανής. Αν για παράδειγμα στόχος μας είναι ένα παιχνίδι με αυτοκίνητα τότε η μηχανή θα πρέπει να υποστηρίζει εξομοίωση κίνησης οχημάτων ή τουλάχιστον να μπορεί να ενσωματώσει κάποιο σχετικό μοντέλο φυσικής. Μπορεί να θέλουμε πολλοί χρήστες να παίζουν αναμεταξύ τους οπότε χρήση της "Torque" για παράδειγμα, που υποστηρίζει δικτύωση, αποτελεί σοφή κίνηση!

2.5.3 Περιβάλλον-πλατφόρμες

Θα δημιουργήσουμε μια εφαρμογή για προσωπικό υπολογιστή ή για κάποια παιχνιδομηχανή; Στην πρώτη περίπτωση τί λειτουργικό θα έχουμε; Υπάρχουν μηχανές για ανάπτυξη σε Windows, Linux, PS2 κτλ, αλλά και μηχανές ανεξαρτήτως πλατφόρμας (για παράδειγμα η "UnrealEngine2").

Η επιλογή περιβάλλοντος ανάπτυξης ωστόσο δεν είναι τόσο εύκολη. Κάποιος ανεξάρτητος δημιουργός ίσως σκεφτεί ότι ένας τίτλος για Linux θα γνωρίσει μεγαλύτερη ζήτηση από ότι για Windows, όπου στην τελευταία περίπτωση η αγορά έχει κατά κάποιο τρόπο "κορεστεί". Προκύπτουν όμως άλλα θέματα.

Όταν ξεκίνησε η παρούσα εργασία έγινε μια απόπειρα για δημιουργία παιχνιδιού αποκλειστικά για λειτουργικό Linux με στόχο να καταρριφθεί ο μύθος των Windows ως το επικρατέστερο λειτουργικό για την ανάπτυξη παιχνιδιών. Μετά από έρευνα επιλέχθηκε η "Delta3D" (http://www.delta3d.org/). Πρόκειται για μηχανή "ανοικτού κώδικα" που αναπτύχθηκε από το Αμερικάνικο ναυτικό και χρησιμοποιήθηκε αρχικά ως μηχανή εξομοίωσης. Σήμερα βρίσκει εφαρμογή και σε άλλους τομείς όπως η διασκέδαση.

Είναι γραμμένη σε γλώσσα C++, παράγει τρισδιάστατα γραφικά, υποστηρίζει ήχο και δικτύωση και είναι διαθέσιμη για πλατφόρμες Linux, Windows και MacOSX. Η εγκατάσταση έγινε μέσω προγράμματος διαχείρισης πακέτων ("synaptic"). Με τον τρόπο αυτό μεταγλωττίζεται ο κώδικας κάποιας εφαρμογής αλλά και όλες οι απαραίτητες βιβλιοθήκες. Δεν άργησαν όμως να παρουσιαστούν τα πρώτα προβλήματα. Κατά τη δοκιμή της μηχανής (και ενώ η εγκατάσταση είχε κυλήσει ομαλά) εμφανίστηκαν ασυμβατότητες με άλλα πακέτα (συγκεκριμένα με το εργαλείο "OpenSceneGraph" που χρησιμοποιεί η μηχανή για τα γραφικά). Η κοινότητα της Delta3D αμέσως ξεκίνησε να ψάχνει για λύση χωρίς όμως κάποιο άμεσο αποτέλεσμα. Εγκαταλείφθηκε έτσι η αρχική ιδέα.

2.5.4 Προϊστορία μηχανής

Χρήσιμη πληροφορία μπορεί να αποδειχτεί το "παρελθόν" της μηχανής. Αν έχει ήδη χρησιμοποιηθεί στην ανάπτυξη παιχνιδιών, εμπορικών και μη, σημαίνει ότι έχει δοκιμαστεί και δεν θα βρεθούμε στη δυσάρεστη θέση να είμαστε εμείς που θα βρούμε πρώτοι τα λάθη της.

2.6 Μηχανές άξιες προσοχής

Στην παρούσα ενότητα παρουσιάζονται δέκα εμπορικές και δέκα ελεύθερες μηχανές. Η επιλογή έγινε με βάση τις κρητικές που συγκεντρώνουν στην ιστοσελίδα DevMaster.net. Πρόκειται για μία από τις μεγαλύτερες πηγές πληροφόρησης πάνω στην ανάπτυξη παιχνιδιών κυρίως για ανεξάρτητους προγραμματιστές. Από την λίστα απουσιάζουν οι πιο γνωστές εμπορικές μηχανές όπως η "Unreal", μιας και για κάποιον εκτός μεγάλης εταιρείας είναι σχεδόν αδύνατον να έχει δουλέψει με μία τέτοια!

2.6.1 Η "δεκάδα" των εμπορικών μηχανών

Μηχανή	Γλώσσα	Πλατφόρμα	Διεπαφή	Ήχος	Δικτύωση	Γλώσσα σεναρίων
Torque Game Engine	C++	Windows, Linux, MacOS	DirectX, OpenGL	2D Sound, 3D Sound, Streaming Sound	Client-Server	Torque Script

- ΥΠΕΡ: από τις πιο οικονομικές εμπορικές μηχανές με διαθέσιμο τον κώδικα, μεγάλη κοινότητα, υποστήριξη πολλών τρισδιάστατων μοντέλων
- ΚΑΤΑ: δεν υποστηρίζει σκιαστές (Shaders)

TV3D SDK 6.5	C++, .Net, VB, Delphi	Windows	DirectX	2D Sound, 3D Sound, Streaming Sound	Αναμένεται	VBscript, Python, Java Script

- ΥΠΕΡ: εύκολη στη χρήση
- ΚΑΤΑ: δεν διατίθεται ο κώδικας

A7 (3D GameStudio)	C++, Delphi	Windows	DirectX	3D Sound	Client-Server, Master Server	Lite-c

- ΥΠΕΡ: εύκολη στη χρήση και για όσους δεν γνωρίζουν προγραμματισμό
- ΚΑΤΑ: δεν διατίθεται ο κώδικας, περιορισμένη σε δυνατότητες γλώσσα σεναρίων

C4 Engine	C/C++	Windows, MacOS, PS3	OpenGL	2D sound, 3D Sound, Streaming Sound	Client-Server	Visual Scripting

- ΥΠΕΡ: καλή υποστήριξη προερχόμενη από τον ίδιο το δημιουργό της μηχανής, εξελίσσεται συνεχώς
- ΚΑΤΑ: στην πραγματικότητα δεν υπάρχει γλώσσα σεναρίων. Πρόκειται για ένα παράθυρο όπου τα σενάρια διορθώνονται γραφικά για βοήθεια στους σχεδιαστές

Unity	C#	Windows, MacOS	DirectX, OpenGL	2D Sound, 3D Sound, Streaming Sound	Client-Server	JavaScript, C#, Boo

- ΥΠΕΡ: πολλά εργαλεία, πλήρης βιβλιογραφία
- ΚΑΤΑ: ο κώδικας διατίθεται με ξεχωριστή άδεια, απαιτεί καλό υλικό (hardware)

Μηχανή	Γλώσσα	Πλατφόρμα	Διεπαφή	Ήχος	Δικτύωση	Γλώσσα σεναρίων
NeoAxis Engine	C/C++, C#	Windows	DirectX, OpenGL	2D Sound, 3D Sound, Streaming Sound	Όχι στην παρούσα έκδοση	C# like syntax

- ΥΠΕΡ: κατάλληλη για κάθε κατηγορία παιχνιδιών
- ΚΑΤΑ: πολύ ακριβή η άδεια για την απόκτηση του κώδικα

DX Studio	C/C++, C#, Visual Basic 6, VB.NET	Windows	DirectX	2D Sound, 3D Sound, Streaming Sound	Client-Server	JavaScript

- ΥΠΕΡ: ό,τι χρειάζεται να αναπτυχθεί για το παιχνίδι μπορεί να γίνει μέσω σεναρίων, εύκολο περιβάλλον διασύνδεσης
- ΚΑΤΑ: προβλήματα ασυμβατότητας με κάποια αρχεία μοντέλων, δεν διατίθεται ο κώδικας

Leadwerks Engine 2	C/C++, C#, Visual Basic 6, VB.NET, Java, D, Delphi, Pascal, Basic, Ada, Fortran, Lisp, Perl, Python	Windows	OpenGL	3D Sound	Αναμένεται	Leadwerks Script

- ΥΠΕΡ: πολύ καλό υποσύστημα για φωτισμούς και σκιές
- ΚΑΤΑ: όχι καλή βιβλιογραφία, απαιτητική σε υλικό (hardware), δεν διατίθεται ο κώδικας

Μηχανή	Γλώσσα	Πλατφόρμα	Διεπαφή	Ήχος	Δικτύωση	Γλώσσα σεναρίων
Visual3D .NET Game Engine	C#	Windows, Xbox360	DirectX	3D Sound, Streaming Sound	Client-Server, Peer-to-Peer	any .NET CLR language, C#, VB.NET, IronPython, Lua.NET

- ΥΠΕΡ: έχει από πίσω της μία μεγάλη εταιρεία (Microsoft), αν η τιμή είναι αυτή που εκτιμάται τότε μιλάμε για μια φτηνή μηχανή με βάση τις δυνατότητές της
- ΚΑΤΑ: απαιτητική σε υλικό (hardware)

3Impact	C/C++, Delphi, Basic	Windows	DirectX	2D Sound, 3D Sound	Client-Server	Όχι

- ΥΠΕΡ: "ελαφριά" μηχανή
- ΚΑΤΑ: ακριβή για αυτά που παρέχει, εμφανή λάθη (bugs), μόνο on-line βοήθεια, δεν διατίθεται κώδικας

2.6.2 Η "δεκάδα" των ελεύθερων μηχανών

Μηχανή	Γλώσσα	Πλατφόρμα	Διεπαφή	Ήχος	Δικτύωση	Γλώσσα σεναρίων
OGRE	C/C++	Windows, Linux, MacOS	DirectX, OpenGL	Όχι	Όχι	Όχι

- ΥΠΕΡ: μεγάλη κοινότητα, καλή βιβλιογραφία, εφαρμογή σε πολλά παιχνίδια
- ΚΑΤΑ: στην πραγματικότητα πρόκειται για μηχανή γραφικών και όχι για μηχανή παιχνιδιών

Irrlicht	C/C++, C#, VB.NET	Windows, Linux, MacOS	DirectX, OpenGL, Software	Όχι	Όχι	Lua

- ΥΠΕΡ: μεγάλη κοινότητα, καλή βιβλιογραφία, αναπτύσσεται συνεχώς
- ΚΑΤΑ: δεν υποστηρίζει πολλά είδη αρχείων

Μηχανή	Γλώσσα	Πλατφόρμα	Διεπαφή	Ήχος	Δικτύωση	Γλώσσα σεναρίων
Crystal Space	C/C++	Windows, Linux, MacOS	OpenGL, Software	2D Sound, 3D Sound	Όχι	Java, Perl και Python αλλά σαν plugins

- ΥΠΕΡ: μπορεί να χρησιμοποιηθεί σε κάθε τρισδιάστατη εφαρμογή και όχι μόνο για παιχνίδια

- ΚΑΤΑ: κάποια παραδείγματα και βοηθήματα δεν είναι ενημερωμένα, όχι καλό σύστημα ανίχνευσης συγκρούσεων, δεν συνιστάται σε κάποιον χωρίς εμπειρία

Panda3D	C/C++, Python	Windows, Linux, SunOS	DirectX, OpenGL	2D Sound, 3D Sound, Streaming Sound	Client-Server	Python

- ΥΠΕΡ: εύκολη στη χρήση, έχει χρησιμοποιηθεί στην ToonTown της Disney
- ΚΑΤΑ: όχι μεγάλη κοινότητα, τα βοηθήματα είναι κυρίως για Python και όχι για ανάπτυξη με C++

jMonkey Engine	Java	Windows, Linux, MacOS	OpenGL	3D Sound	JGN, jme	jMonkey Scripting Framework

- ΥΠΕΡ: πιο γρήγορη στη μετάφραση (compiling, linking) από μηχανές με κώδικα σε C++, υποστήριξη σκιών και φωτισμού με ελάχιστες προσθήκες κώδικα
- ΚΑΤΑ: υστερεί από άλλες μηχανές που έχουν αναπτυχθεί, για παράδειγμα με C++, μιας και οι προγραμματιστές μηχανής σε Java είναι λίγοι για να συμβάλουν στη βελτίωση του κώδικα

Reality Factory	C/C++	Windows	DirectX, OpenGL	3D Sound, Streaming Sound	Peer-to-Peer	Integrated scripting engine (SimKin), cinematic and in-game sequence scripting

- ΥΠΕΡ: κατάλληλη για όσους δεν γνωρίζουν καθόλου προγραμματισμό
- ΚΑΤΑ: στηρίζεται σε ξεπερασμένη για την εποχή μηχανή (Genesis 3D) με αποτέλεσμα να "κρεμάει"

Μηχανή	Γλώσσα	Πλατφόρμα	Διεπαφή	Ήχος	Δικτύωση	Γλώσσα σεναρίων
Blender Game Engine	C/C++, Python	Windows, Linux, MacOS, Solaris, FreeBSD, Irix	OpenGL	2D Sound, 3D Sound	Client-Server	Python

- ΥΠΕΡ: υποστηρίζει όλα σχεδόν τα λειτουργικά συστήματα
- ΚΑΤΑ: δύσκολη στη χρήση για τους νέους στο χώρο, ίσως περισσότερο εργαλείο μοντελοποίησης παρά μηχανή

The Nebula Device 2	C/C++	Windows	DirectX	2D Sound, 3D Sound, Streaming Sound	Client-Server, Peer-to-Peer	TCL, Lua, Python, Ruby

- ΥΠΕΡ: έχει χρησιμοποιηθεί στην ανάπτυξη εμπορικών τίτλων παιχνιδιών
- ΚΑΤΑ: δύσκολη στη χρήση

Realm-Forge	C/C++, C#, D, Delphi, Ada, For-tran, Perl, Python, VB.NET	Windows, Linux, MacOS, Solaris, HP/UX, FreeBSD	DirectX, OpenGL	2D Sound, 3D Sound, Streaming Sound	Client-Server, Peer-to-Peer, Master Server	C#, JScript.net

- ΥΠΕΡ: πολλά χαρακτηριστικά (features) για ελεύθερη μηχανή
- ΚΑΤΑ: βρίσκεται ακόμα στο στάδιο ανίχνευσης σφαλμάτων (alpha stage)

Open-Scene-Graph	C/C++	Windows, Linux, MacOS, Solaris, SunOS, FreeBSD, Irix, Playstation	OpenGL	2D Sound, 3D Sound, Streaming Sound	Όχι	Lua

- ΥΠΕΡ: υποστηρίζει όλα σχεδόν τα λειτουργικά συστήματα, μπορεί να χρησιμοποιηθεί γενικά σε εφαρμογές γραφικών, καλό υποσύστημα διαλογής (culling)
- ΚΑΤΑ: "φτωχή" βιβλιογραφία, όχι ουσιαστική βοήθεια από την κοινότητα

2.7 Ποιά επιλέγουμε τελικά

Ποιά είναι λοιπόν η μηχανή που θα χρησιμοποιήσουμε; Πριν την αποκαλύψουμε ας καταγράψουμε τις σκέψεις μας.

Ξεκινάμε με το δεδομένο ότι θα αναπτύξουμε μία τρισδιάστατη εφαρμογή. Προηγούμενη εμπειρία σε ανάπτυξη ηλεκτρονικού παιχνιδιού δεν υπάρχει οπότε η πιο σοφή κίνηση, για πρώτο παιχνίδι, είναι η δημιουργία ενός "πρώτου προσώπου βολής " παιχνιδιού ή παιχνιδιού αυτοκινητιστικών αγώνων. Ό,τι και να επιλέξουμε θα περιέχει κάποιου είδους τεχνητή νοημοσύνη ενώ καλό είναι να υποστηρίζει και δυνατότητα δικτύωσης πολλών χρηστών. Έτσι έχουμε μία πρώτη ιδέα για το τι χαρακτηριστικά μηχανής ψάχνουμε.

Δεν πρέπει να ξεχνάμε την ιδιότητα του προγραμματιστή, με την οποία εκπονούμε την εργασία. Δεν είμαστε σχεδιαστές ή γραφίστες! Έτσι δεν μπορούμε να διαλέξουμε μηχανή που όλα γίνονται με γραφικά περιβάλλοντα. Καλό επίσης θα ήτανε η τελική μας επιλογή να μην μας περιορίσει αλλά αντίθετα να μας ανοίξει και άλλους δρόμους. Τί θέλουμε να πούμε με αυτό: μηχανή που διατίθεται με τον κώδικα, δίνει τη δυνατότητα να μελετήσουμε τα διάφορα τμήματά της (όπως αυτά περιγράφτηκαν παραπάνω) αλλά και να προσθέσουμε νέες λειτουργίες στον πυρήνα του παιχνιδιού. Το τελευταίο προϋποθέτει ότι γνωρίζουμε τη γλώσσα προγραμματισμού της ή τουλάχιστον έχουμε μια μεγαλύτερη εξοικείωση σε σχέση με άλλες γλώσσες. Στην παρούσα εργασία, το κριτήριο αυτό συνεπάγεται κώδικα γραμμένο σε C/C++.

Κάπου εδώ φτάνουμε στην αποκάλυψη του ονόματος της μηχανής που επιλέχτηκε. Η μηχανή που συγκεντρώνει όλα τα στοιχεία που παρουσιάστηκαν είναι η **Torque Game Engine** από την **GarageGames**. Μία πρώτη γνωριμία (όπως το μικρό της κόστος) έγινε στο παρόν κεφάλαιο αλλά εκτενέστερη αναφορά θα ακολουθήσει στο αμέσως επόμενο.

Κεφάλαιο 3

Η μηχανή παιχνιδιών Torque

Η Torque είναι μία μηχανή τρισδιάστατων γραφικών η οποία διατίθεται σε επαγγελματίες και ανεξάρτητους προγραμματιστές από την **GarageGames** (http://www.garagegames.com). Πρόκειται για μία τροποποιημένη έκδοση της μηχανής που χρησιμοποίησε η Dynamix για το παιχνίδι "Tribes 2", κυκλοφορίας 2001. Κάποια από τα μέλη της εταιρείας αποχώρησαν την ίδια εποχή και ίδρυσαν την GarageGames εξαγοράζοντας τη μηχανή που χρησιμοποιήθηκε στο εν λόγω παιχνίδι. Το αρχικό όνομα που δόθηκε ήτανε V12 αλλά μετά από κάποια νομικά "κολλήματα" πήρε το σημερινό, **Torque Game Engine (TGE)**!

Στο παρόν κεφάλαιο θα αναλυθούν τα κύρια χαρακτηριστικά της μηχανής, τα σημαντικότερα υποσυστήματά της και θα γίνει μία πρώτη παρουσίαση της γλώσσας σεναρίων, "TorqueScript".

3.1 Βασικά χαρακτηριστικά

Κάποιες πληροφορίες για τη μηχανή "Torque" (για συντομία "TGE") παρουσιάστηκαν στο προηγούμενο κεφάλαιο, στους συγκριτικούς πίνακες των μηχανών. Εδώ καταγράφονται τα χαρακτηριστικά της μηχανής που την κάνουν να ξεχωρίζουν και αποτελούν το δυνατό της σημείο.

3.1.1 Αρχιτεκτονική "πελάτη-εξυπηρετητή" ("client-server" architecture)

Η TGE είναι κατάλληλη για ανάπτυξη παιχνιδιών "ενός παίκτη" (**single-player** games) αλλά και πολλών παικτών ταυτόχρονα (**multiplayer**). Αυτό το τελευταίο επιτυγχάνεται χάριν του γεγονότος ότι σχεδιάστηκε εξ' αρχής έτσι ώστε να διευκολύνει την ανάπτυξη δικτυακών εφαρμογών.

Κάθε εφαρμογή υλοποιημένη στην Torque χωρίζεται σε δύο μέρη: **τον πελάτη (client) και τον εξυπηρετητή (server)**. Αυτό ισχύει ακόμα και όταν η εφαρμογή εκτελείται σε έναν μόνο υπολογιστή (τοπική σύνδεση, ενώ σε αντίθετη περίπτωση έχουμε απομακρυσμένη σύνδεση). Ο κώδικας στο τμήμα "εξυπηρετητής" είναι ανεξάρτητος από τον κώδικα στο τμήμα "πελάτης". Ο εξυπηρετητής έχει τις δικές του μεταβλητές, συναρτήσεις και αντικείμενα, και αντίστοιχα για τον πελάτη. Ο εξυπηρετητής δεν μπορεί να προσπελάσει άμεσα τις μεταβλητές ή τα αντικείμενα του πελάτη ούτε να καλέσει τις

συναρτήσεις που ορίζονται στον κώδικα του πελάτη. Τα αντίστοιχα ισχύουν για τον πελάτη[1].

Γίνεται λοιπόν αντιληπτό ότι έχουμε διάκριση λειτουργιών. Ένα καλά σχεδιασμένο δικτυακό παιχνίδι θα πρέπει να αφήνει στην πλευρά του εξυπηρετητή όσον το δυνατόν περισσότερες αρμοδιότητες. Συνηθίζεται ο πελάτης να είναι υπεύθυνος μόνο για ό,τι έχει να κάνει με τη διασύνδεση του χρήστη με την εφαρμογή (**GUI**[2]).

Αφήνοντας όμως όλη τη λειτουργικότητα στην πλευρά "εξυπηρετητής", αυτόματα προκύπτει ένα σοβαρό θέμα! Όλο και περισσότερη πληροφορία θα πρέπει να μεταφερθεί από τον εξυπηρετητή προς τους πελάτες. Τί σημαίνει αυτό; Μείωση του εύρους ζώνης (bandwidth) του δικτύου! Κάτι τέτοιο οδηγεί στα λεγόμενα "κολλήματα" (lags) κατά τη διάρκεια του παιχνιδιού. Οι ενέργειες ενός παίκτη (πελάτης) αργούν να βρουν ανταπόκριση από το παιχνίδι (εξυπηρετητής). Αναφέρθηκε όμως αρχικά ότι η TGE διευκολύνει τη δικτύωση και λύνει αποτελεσματικά τέτοιου είδους προβλήματα. Κάποιες από τις στρατηγικές που χρησιμοποιούνται είναι:

- διάκριση ενημερώσεων σε σημαντικές και λιγότερο σημαντικές, έτσι ώστε οι πρώτες να αποστέλλονται στους πελάτες με μεγαλύτερη συχνότητα από ότι οι τελευταίες,
- αποστέλλονται μόνο τα απαραίτητα bits πληροφορίας,
- οι συμβολοσειρές αποστέλλονται μία μόνο φορά ολόκληρες και σε κάθε άλλη περίπτωση μεταφορά τους, από τον εξυπηρετητή προς τους πελάτες, χρησιμοποιείται ένας αριθμός/ετικέτα.

Πέρα όμως αυτού (δηλαδή τη διαχείριση του εύρους ζώνης) το σύστημα δικτύωσης της TGE φροντίζει και για θέματα που έχουν να κάνουν με την απώλεια πακέτων [εύρεση λαθών-κώδικας κυκλικού πλεονασμού (CRC)] αλλά και το συγχρονισμό μεταξύ των ενεργειών πελάτη και εξυπηρετητή.

Γενικά ο διαχωρισμός σε "πελάτη-εξυπηρετητή" βοηθά στην απομόνωση κρίσιμων για την ομαλή λειτουργία του κόσμου λειτουργιών, στην πλευρά του εξυπηρετητή. Ο τελευταίος αναλαμβάνει να παρέχει σε όλους τους πελάτες, που είναι συνδεδεμένοι μαζί του, την ίδια, κοινή εικόνα του κόσμου και να διαχειρίζεται την αλληλεπίδρασή τους με αυτόν και μεταξύ τους με συνεπή τρόπο.

3.1.2 Από γεγονότα καθοδηγούμενος εξομοιωτής (event-driven simulator)

Η Torque είναι ένας "καθοδηγούμενος από γεγονότα εξομοιωτής", **δηλαδή κάθε ενέργεια της μηχανής προκαλείται από κάποιου είδους γεγονός**.

[1] Υπάρχει μια εξαίρεση σε αυτόν τον κανόνα! Όταν "πελάτης" και "εξυπηρετητής" "τρέχουν" στο ίδιο μηχάνημα, είναι δυνατόν να έχουν πρόσβαση ο ένας στον κώδικα του άλλου. Φυσικά κώδικας που στηρίζεται στην παραπάνω εξαίρεση πρέπει να αποφεύγεται, καθώς δεν πρόκειται να λειτουργήσει σε δικτυακό περιβάλλον.

[2] **Graphical User Interface (Γραφική Διεπαφή Χρήστη):** διασύνδεση γραφικών με το χρήστη. Τύπος διεπαφής υπολογιστή με την οποία η επικοινωνία μεταξύ ανθρώπου και μηχανής γίνεται με εκτεταμένη χρήση οπτικών προτροπών και υποβοηθήσεων της μνήμης (του ανθρώπου) με τη μορφή εικονιδίων, αντικειμένων, ετικετών, συρόμενων μενού και παραθύρων, παρά με κώδικα και εντολές κειμένου.

Μέσα στον κόσμο του παιχνιδιού, όλες οι οντότητες (για παράδειγμα το έδαφος, ένα δένδρο, ο χαρακτήρας που χειρίζεται ο παίκτης) θεωρούνται αντικείμενα (**objects**). Τα αντικείμενα αλληλεπιδρούν μεταξύ τους και τότε έχουμε ένα γεγονός (**event**). Σε αναλογία με τον πραγματικό κόσμο ας φέρουμε στο μυαλό μας το πρωινό ξύπνημα. Ξυπνάμε απότομα, ενώ ονειρευόμαστε ότι είμαστε πτυχιούχοι, εξαιτίας του ήχου από το ξυπνητήρι (γεγονός). Αυτό προκαλεί (**trigger**) μία ενέργεια, που είναι να σηκώσουμε το χέρι μας και να πατήσουμε το κουμπί που κλείνει το ξυπνητήρι [είσοδος (**input**)]. Η ενέργειά μας αυτή είναι ένα γεγονός για το αντικείμενο "ξυπνητήρι" το οποίο μπαίνει σε κατάσταση αναμονής για πέντε λεπτά. Όταν ο χρόνος αυτός περάσει προκαλείται ένα νέο γεγονός που είναι να ηχήσει ξανά το ρολόι.

Όμοια και με την TGE ορίζουμε τα αντικείμενα του παιχνιδιού και υλοποιούμε τις συναρτήσεις εκείνες που θα χειριστούν τα γεγονότα που θα προκύψουν [αυτές οι συναρτήσεις αναφέρονται και ως "κλήσεις " (**Callbacks**)].

3.1.3 Διαθέσιμος κώδικας σε C++ και "TorqueScript"

Σε αντίθεση με άλλες μηχανές, η TGE διατίθεται μαζί με τον κώδικά της. Μας δίνεται δηλαδή η δυνατότητα να τροποποιήσουμε την ίδια τη μηχανή προσθέτοντας νέες λειτουργίες. Αυτό σε συνδυασμό με τη γλώσσα σεναρίων "TorqueScript" (θα παρουσιαστεί παρακάτω) επιτρέπει την ανάπτυξη εφαρμογών (και στην περίπτωσή μας ενός ηλεκτρονικού παιχνιδιού) σε τρία επίπεδα:

- **διαχείρισης συστημάτων**,
- **υλοποίησης αντικειμένων** και
- **υλοποίησης του παιχνιδιού**.

Στο πρώτο επίπεδο η TGE κάνει όλη τη δουλειά για εμάς! Ο πηγαίος της κώδικας εκτελεί πλήθος βασικών λειτουργιών στο παρασκήνιο, επιτρέποντας στον προγραμματιστή να μην αναλώνεται στην ανάπτυξη κώδικα για θέματα που αφορούν το λειτουργικό σύστημα, την αξιοποίηση του υλικού του μηχανήματος και οτιδήποτε άλλο συναντάει ξανά και ξανά σε μία εφαρμογή. Πολύ σπάνια ο προγραμματιστής (εκτός και αν πρόκειται για αυτόν που αναπτύσσει τη μηχανή) θα χρειαστεί να ασχοληθεί με αυτό το επίπεδο!

Αν και το επίπεδο "διαχείρισης συστημάτων" είναι υπεύθυνο για τις ιδιότητες των αντικειμένων που υπάρχουν στη σκηνή του παιχνιδιού [για παράδειγμα ο χαρακτήρας που ελέγχεται από τον παίκτη (avatar) έχει βάρος, ταχύτητα κίνησης, πλέγμα συγκρούσεων (**"bounding box"** ή αλλιώς **"collision boxes"**)], στο δεύτερο επίπεδο ("υλοποίησης αντικειμένων") μπορούμε να προσθέσουμε επιπλέον δυνατότητες. Για παράδειγμα θέλουμε οι χαρακτήρες που ελέγχονται από τον υπολογιστή (**"bot"** ή **"NPCs"**) να χρησιμοποιούν τον αλγόριθμο A* για "εύρεση μονοπατιού". Υλοποιούμε τον αλγόριθμο σε γλώσσα C++ και τον ενσωματώνουμε στον "πυρήνα" της μηχανής.

Τέλος η "υλοποίηση του παιχνιδιού" είναι το επίπεδο στο οποίο ο προγραμματιστής εργάζεται (κυρίως) με τη γλώσσα σεναρίων της TGE και προγραμματίζει το παιχνίδι. Είναι η φάση όπου ο παίκτης περιηγείται στη σκηνή του παιχνιδιού, εκτελεί αποστολές (tasks), κερδίζει βαθμούς, αντιλαμβάνεται τις επιπτώσεις που έχει κάποια ενέργειά του. Για να μην υπάρξει κάποια παρεξήγηση, στο ίδιο αυτό επίπεδο ο προγραμματιστής καθορίζει και τις ενέργειες των "εχθρών" (bots). Δηλαδή ναι μεν μπορεί να χρησιμοποιούν

για τη μετακίνηση τους τον A* αλγόριθμο αλλά πρέπει να ορίσουμε πότε να ξεκινήσουν την κίνησή τους, πώς να δράσουν αν εντοπίσουν κάποιον συνδεδεμένο παίκτη (εννοείται το avatar) και πολύ περισσότερο πώς να εντοπίσουν έναν παίκτη!

3.1.4 Σύστημα απόδοσης (3D Rendering)

Η Torque είναι μία ολοκληρωμένη μηχανή και όχι απλώς μία μηχανή απόδοσης (rendering engine). Μέσω των συναρτήσεων και μεθόδων της επιτρέπει την ανάπτυξη παιχνιδιών για διάφορες πλατφόρμες (mac, pc, linux), τη δυνατότητα δικτύωσης (όπως αναφέραμε παραπάνω), υποστηρίζει ανίχνευση συγκρούσεων, φυσική για κίνηση "σκελετωδών" αντικειμένων ("rigid body") και οχημάτων, διαχειρίζεται τον ήχο, τις γραφικές διασυνδέσεις με το χρήστη. Επιπλέον προσφέρει τη δυνατότητα δημιουργίας βασικών οντοτήτων για το παιχνίδι όπως το έδαφος, ο ουρανός, το νερό. Καλύπτει επίσης και θέματα φωτισμού, σκιών αλλά δεν υποστηρίζει τεχνολογία "σκιαστών" (**shaders**[3]).

Όλα αυτά τα τμήματα κώδικα της μηχανής δουλεύουν αρμονικά για να πάρουμε το τελικό αποτέλεσμα στην οθόνη. Έχοντας αποκατασταθεί μία σύνδεση μεταξύ πελάτη και εξυπηρετητή, τη δράση αναλαμβάνουν οι βιβλιοθήκες που είναι υπεύθυνες για τη διασύνδεση του χρήστη με το παιχνίδι (GUI library). Το αντικείμενο "Canvas" είναι η "βάση" πάνω στην οποία θα στηθεί όλη η εφαρμογή. Είναι ο "καμβάς " πάνω στον οποίο θα "ζωγραφίσουμε" (οι όροι προέρχονται από τον τρόπο που εργάζεται ένας ζωγράφος) το μενού του παιχνιδιού μας, θα παρουσιάσουμε πληροφορίες για τα αντικείμενα που συγκεντρώνουμε, θα αποτελέσει τα "μάτια" του χαρακτήρα με τον οποίο ο παίκτης περιπλανιέται στον κόσμο του παιχνιδιού. Σε αυτόν η μηχανή απόδοσης, μέσω της κάρτας γραφικών θα παράγει την τρισδιάστατη σκηνή του παιχνιδιού.

Αρχικά το σύστημα απόδοσης της Torque ορίζει τον προσανατολισμό της κάμερας και τη γωνία θέασης [field of view (fov)] και στη συνέχεια σχεδιάζει τη σκηνή με χρήση εντολών OpenGL. Προσοχή εδώ! Ο χρήστης δεν είναι σε θέση να δει τον τρισδιάστατο κόσμο, προς το παρόν! Για να γίνει αυτό απαιτείται το λεγόμενο "**άντικείμενο ελέγχου**" (**control object**). Πρόκειται για το αντικείμενο εκείνο που ελέγχεται (όπως προδίδει και το όνομα του) από το χρήστη και συνήθως είναι αντικείμενο της κλάσης "Player" ή της κλάσης "Camera". Για να μην μπερδευτούμε μπορούμε να πούμε απλά ότι το "άντικείμενο ελέγχου" είναι το "avatar"! Στην ουσία το αντικείμενο ελέγχου καθορίζει την κάμερα του παίκτη για τον εικονικό κόσμο, δηλαδή πού βρίσκεται σε σχέση με το "avatar" [πρώτου προσώπου (first person) ή τρίτου προσώπου (third person) σημείο θέασης (point of view)] και ποια είναι η μέγιστη γωνία θέασης. Η μηχανή συγκεντρώνει τις πληροφορίες αυτές. Τα παραπάνω εξελίσσονται στην πλευρά του εξυπηρετητή.

Από την πλευρά του εξυπηρετητή μεταφερόμαστε στην πλευρά του πελάτη όπου η TGE καλεί το γράφο σκηνής (του πελάτη). Η βιβλιοθήκη αυτή είναι υπεύθυνη για να καθορίσει ποια από τα αντικείμενα της σκηνής πρέπει να σχεδιαστούν με βάση τη θέση της κάμερας του παίκτη. Ο πελάτης ενημερώνει τον εξυπηρετητή και ο τελευταίος αποστέλλει τα απαραίτητα αντικείμενα. Έτσι σε κάθε πελάτη σχεδιάζεται μόνο ό,τι

[3]Πρόκειται για προγράμματα με τα οποία η κάρτα γραφικών αναλαμβάνει την απεικόνιση ειδικών εφέ όπως αντανακλάσεις, διαφάνειες, διαθλάσεις, λάμψεις. Υπάρχουν δύο είδη σκιαστών: vertex και pixel. Χωρίς τη χρήση τους, η διαδικασία φωτισμού είναι καθορισμένη (flat shaded ή phong ή gouraud shading models).

βλέπει ο χρήστης [διαλογή (culling)[4]].

3.1.5 Ενσωματωμένα εργαλεία επεξεργασίας

Η Torque περιέχει δύο βοηθητικά εργαλεία: το "**World Editor**" και το "**GUI Editor**".

Το τελευταίο από τα δύο, επιτρέπει την επεξεργασία μιας ήδη υπάρχουσας διεπαφής (για παράδειγμα το βασικό μενού του παιχνιδιού που εμφανίζεται κατά την εκκίνηση) ή τη δημιουργία νέας. Είναι ένα σχετικά απλό εργαλείο όπου στηρίζεται στη λειτουργία "μεταφορά και απόθεση" ("drag-and-drop").

Από την άλλη ο "World Editor" είναι ένας πιο σύνθετος "συντάκτης - επεξεργαστής ", όπου διαχειρίζεται τη σκηνή. Στην πραγματικότητα δεν είναι ένα αλλά οκτώ διαφορετικά εργαλεία:

Εργαλείο	**Περιγραφή**
World Editor Manipulator	Επιτρέπει τη μετατόπιση (translate), περιστροφή (rotate) και διαβάθμιση (scale) αντικειμένων που ήδη έχουν τοποθετηθεί στη σκηνή.
World Editor Inspector	Επιτρέπει τον έλεγχο και την επεξεργασία ιδιοτήτων των αντικειμένων των αποστολών (missions).
World Editor Creator	Επιτρέπει την εισαγωγή νέων αντικειμένων στη σκηνή.
Mission Area Editor	Επιτρέπει τον καθορισμό των ορίων/συνόρων της σκηνής.
Terrain Editor	Επιτρέπει την απευθείας επεξεργασία του εδάφους (terrain) χρησιμοποιώντας το ποντίκι ως "πινέλο".
Terrain Terraform Editor	Επιτρέπει τη φόρτωση εικόνων ως αρχεία εδάφους (terrain files) και την εφαρμογή αλγοριθμικών γεννητριών (algorithmic generators) και φίλτρων σε αυτό.
Terrain Texture Editor	Επιτρέπει την επιλογή οποιουδήποτε αριθμού υφών (textures) και την εφαρμογή τους στη σκηνή αφού πρώτα έχει γίνει χρήση αλγορίθμων για μύξη και τοποθέτηση.
Terrain Texture Painter	Επιτρέπει την επιλογή έως και έξι υφών (textures) και την εφαρμογή τους στο έδαφος.

Πίνακας 3.1: Τα υπο-εργαλεία του "World Editor"

Στην παρούσα εργασία τα μόνα που θα χρησιμοποιηθούν είναι το "Terrain Terraform Editor" και το "Terrain Texture Painter".

[4]Η TGE χρησιμοποιεί **"Portals"**. Η δεύτερη μέθοδος διαλογής είναι τα "BSP trees".

3.2 Τύποι αρχείων

Ακολουθούν οι βασικοί τύποι αρχείων τους οποίους υποστηρίζει η μηχανή παιχνιδιών Torque:

- .cs: αρχεία πηγαίου κώδικα σε γλώσσα σεναρίων "TorqueScript",
- .gui: αρχεία πηγαίου κώδικα σε "TorqueScript", όπως και τα .cs, με κώδικα σχετικό για διεπαφές (GUI),
- .mis: αρχεία πηγαίου κώδικα σε "TorqueScript", όπως και τα .cs, με κώδικα σχετικό με τη δομή κάποιας αποστολής και τη δημιουργία αντικειμένων σε αυτή,
- .dso: δυαδικά αρχεία που προκύπτουν από τη μετάφραση των παραπάνω αρχείων,
- .jpg και .png: επιτρεπτοί τύποι αρχείων για εικόνες,
- .wav και .ogg: επιτρεπτοί τύποι αρχείων για ήχους,
- .dts: τύποι αρχείων για τα τρισδιάστατα μοντέλα που εισάγονται στη σκηνή,
- .dsq: αρχεία που περιέχουν τις ακολουθίες κίνησης (animation sequences) για κάποιο μοντέλο, όταν αυτές δεν είναι ενσωματωμένες στο ίδιο το ".dts" [παρέχουν μεγαλύτερη ευελιξία και δυνατότητα για ταυτόχρονη εκτέλεση περισσοτέρων από ενός κινουμένων γραφικών (merge animations)] και
- .dif: αρχεία για "δομικές κατασκευές ", όπως κτήρια, και οι οποίες επιτρέπουν σε ένα αντικείμενο να βρεθεί στο εσωτερικό τους.

3.3 Η γλώσσα σεναρίων "TorqueScript"

Πλέον είναι κανόνας η χρήση γλώσσας σεναρίων στην ανάπτυξη βιντεοπαιχνιδιών. Κάποιες μηχανές χρησιμοποιούν ήδη υπάρχουσες γλώσσες σεναρίων, όπως Python και Lua, ενώ άλλες έρχονται με ένα δικό τους σύστημα για "σενάρια". Στις τελευταίες ανήκει και η TGE και η γλώσσα σεναρίων η οποία χρησιμοποιείται είναι η **"TorqueScript"**.

Η TorqueScript είναι μία δυνατή και ευέλικτη γλώσσα που συντακτικά είναι παρόμοια με τη γλώσσα προγραμματισμού C++. Τα κύρια χαρακτηριστικά της είναι:

- βασικά γνωρίσματα γλωσσών προγραμματισμού. Χρησιμοποιεί τύπους δεδομένων (συμβολοσειρές, αριθμούς και booleans), δομές ελέγχου, συναρτήσεις, πράξεις πάνω σε δεδομένα,
- πρόσβαση σε συστήματα της μηχανής. Επιτρέπει την πρόσβαση σε υποσυστήματα της TGE όπως σε αυτό της απόδοσης (renderer), του ήχου, της διαχείρισης εισόδου/εξόδου, καθώς και τη δημιουργία/διαγραφή αντικειμένων ή τον ορισμό νέων συναρτήσεων/λειτουργιών,

- στοιχεία οντοκεντρικών προγραμμάτων. Δίνεται η δυνατότητα μέσα από τα "σενάρια" για αυτονομία οντοτήτων ("έγκλεισμός "), απόκρυψη δεδομένων, κληρονομικότητα (δημιουργία νέων αντικειμένων από ήδη υπάρχοντα) και πολυμορφισμό (υπερκάλυψη των υπαρχόντων ιδιοτήτων ενός αντικειμένου),
- χώροι ονομάτων (namespaces). Μερικές φορές μπορεί κάποια συνάρτηση/μεταβλητή ενός πεδίου δράσης να έρχεται σε σύγκρουση με άλλη συνάρτηση/μεταβλητή με το ίδιο όνομα σε διαφορετικό πεδίο δράσης. Με τη μέθοδο των χώρων ονομάτων μπορούμε να λύσουμε αυτό το πρόβλημα,
- κατά επιλογή φόρτωση. Η TorqueScript επιτρέπει δυναμικό φόρτωμα τμημάτων κώδικα χωρίς να χρειάζεται να φορτώσουμε εξαρχής ολόκληρο τον κώδικα στη μνήμη,
- μετάφραση και εκτέλεση. Τα "σενάρια" μεταφράζονται και εκτελούνται όπως συμβαίνει και στις γλώσσες προγραμματισμού. Σε περίπτωση λάθους υποδεικνύεται το σημείο, μέσα στον κώδικα, όπου πιθανότατα έχει συμβεί.

Ωστόσο δεν παύει να είναι μία γλώσσα σεναρίων. Αυτό συνεπάγεται ότι υπάρχουν διαφορές σε σχέση με τις γλώσσες προγραμματισμού. Ακολουθεί λοιπόν μία σύντομη αναφορά στις σημαντικότερες από αυτές.

3.3.1 Μη ευαισθησία σε τύπους δεδομένων (type insensitive)

Η TorqueScript επιτρέπει, πέρα από booleans, δύο τύπους δεδομένων: αριθμούς (numbers) και συμβολοσειρές (strings). Η μετατροπή από τον έναν τύπο στον άλλο γίνεται άμεσα. Έτσι ο τύπος των μεταβλητών ποτέ δεν δηλώνεται. Η ίδια η μεταβλητή μπορεί να χρησιμοποιηθεί άλλοτε σαν μεταβλητή αριθμών και άλλοτε σαν μεταβλητή συμβολοσειρών.

```
// Script #1: Test type-insensitive

if (12=="12")
   echo("Type Insensitive");
else
   echo("Type Sensitive");
```

Εκτελώντας το παραπάνω "σενάριο" θα εμφανιστεί στην κονσόλα[5] το μήνυμα "*Type Insensitive*".

3.3.2 Μη ευαισθησία σε πεζούς και κεφαλαίους χαρακτήρες (case insensitive)

Η TorqueScript δεν κάνει διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων σε ονόματα μεταβλητών ή συναρτήσεων.

```
// Script #2: Test case-insensitive

%a="Torque Game Engine";    //%a is a local variable
%pub="Garage Games";        //%pub is a local variable
```

[5] Είναι μία εφαρμογή η οποία έρχεται μαζί με τη μηχανή και επιτρέπει την εκτέλεση "σεναρίων".

```
echo(%a);       //output: Torque Game Engine
echo(%A);       //output: Torque Game Engine
echo(%pUB,"**",%PUb);    //output: Garage Games**Garage Games
```

Δηλαδή, οι μεταβλητές "a" και "A", στο "σενάριο" παραπάνω, έχουν την ίδια τιμή ("Torque Game Engine"). Το ίδιο συμβαίνει και με τις μεταβλητές "pub","pUB" και "PUB". Δεν είναι τρεις διαφορετικές αλλά μία μεταβλητή με τιμή "GarageGames".

3.3.3 Μεταβλητές και εμβέλεια (scope)

Η γλώσσα σεναρίων της Torque χρησιμοποιεί:

- τοπικές μεταβλητές (**local**), που διακρίνονται από το χαρακτήρα '%' πριν το όνομα της μεταβλητής (για παράδειγμα η μεταβλητή "pub" στο προηγούμενο τμήμα κώδικα) και
- καθολικές μεταβλητές (**global**), οι οποίες δηλώνονται με το χαρακτήρα '**$**', αντί του '%' (για παράδειγμα "$localVar" είναι μία καθολική μεταβλητή).

Μία τοπική μεταβλητή δηλώνεται μέσα σε ένα μπλοκ. Όπως συμβαίνει και με τις πιο ευρέως χρησιμοποιούμενες γλώσσες υψηλού επιπέδου, μία τοπική μεταβλητή μπορεί να αναφερθεί μόνο στο μπλοκ στο οποίο ορίζεται και στα μπλοκ που είναι ένθετα σε αυτό. Αντιθέτως μία καθολική μεταβλητή είναι προσβάσιμη και έγκυρη παντού στον κώδικα, αλλά και σε όλα τα αρχεία σεναρίων (αρχεία ".cs").

3.3.4 Συμβολοσειρές (strings)

Δουλεύοντας με TorqueScript θα συναντήσουμε δύο ειδών συμβολοσειρές:

- μέσα σε διπλά "άυτάκια" (**double-quoted strings**). Είναι η κλασσική μορφή συμβολοσειράς που συναντάμε και στη C++ και
- μέσα σε μονά "άυτάκια" (**single-quoted ή tagged strings**). Είναι μία ειδική κατηγορία για τη μεταφορά συμβολοσειρών μέσω μίας σύνδεσης. Η συμβολοσειρά την πρώτη φορά στέλνεται ως έχει. Από εκεί και πέρα, οποτεδήποτε χρειαστεί να σταλθεί η ίδια συμβολοσειρά, αποστέλλεται ένας αριθμός που αντιστοιχεί σε αυτήν (tag). Είναι ένας τρόπος με τον οποίο η μηχανή πετυχαίνει αποδοτικότερες τιμές σε εύρος ζώνης δικτύου.

```
// Script #3: Double-quoted VS single-quoted strings

%a="This is a standard string";
%b='This is a tagged string';
echo(%a);       //output: This is a standard string
echo(%b);       //output: 9
```

3.3.5 Πίνακες (arrays)

Η TorqueScript υποστηρίζει πίνακες μίας διάστασης αλλά και περισσοτέρων. Πρέπει όμως να δώσουμε προσοχή στα κάτωθι:

- υπάρχει διαχωρισμός μεταξύ μεταβλητής και πίνακα με το ίδιο όνομα. Δεν ισχύει ότι και στη C++, όπου το όνομα του πίνακα είναι δείκτης στο πρώτο του στοιχείο (δεν υπάρχουν δείκτες έτσι όπως τους ξέρουμε στη C++),
- η αναφορά σε ένα στοιχείο του πίνακα μπορεί να γίνει είτε με το συνηθισμένο από τη C, τρόπο [δηλαδή όνομα του πίνακα και ο αριθμός της θέσης του στοιχείου μέσα σε αγκύλες (για παράδειγμα %myArray[4])], είτε απαλείφοντας τις αγκύλες (%myArray4) και
- όπως και πριν, σε πίνακες πολλών διαστάσεων οι αγκύλες μπορούν να απομακρυνθούν ενώ το κόμμα μπορεί να αντικατασταθεί από "κάτω υπογράμμιση" [για παράδειγμα η αναφορά στο στοιχείο που βρίσκεται στη θέση (1,2) του τοπικού πίνακα "myArray" μπορεί να γίνει με %myArray[1,2] ή %myArray1_2).

3.3.6 Διανύσματα (vectors)

Κατά την ανάπτυξη του παιχνιδιού θα υπάρξουν περιπτώσεις που θα πρέπει να γίνει επεξεργασία σε μία ομάδα τιμών. Για παράδειγμα η θέση ενός αντικειμένου στη σκηνή ορίζεται από την τριάδα συντεταγμένων (x,y,z). Η πληροφορία αυτή αποθηκεύεται με τη βοήθεια "διανυσμάτων".

Το πιο συνηθισμένο λάθος, όταν χρησιμοποιούνται διανύσματα, είναι η χρήση αριθμητικών τελεστών για αριθμητικές πράξεις (πρόσθεση, αφαίρεση κτλ.). Αν αναλογιστούμε ότι κατά τη μεταγλώττιση δεν υπάρχει προειδοποίηση για αυτό, καταλαβαίνουμε ότι καταλήγουμε με τελείως λάθος δεδομένα (δείτε το παράδειγμα). Για πράξεις με διανύσματα χρησιμοποιούνται μόνο οι μαθηματικές "συναρτήσεις κονσόλας"!

```
// Script #4: Vectors

// Define vectors "start" and "end"
%start="0.0 5.0 -3.2";
%end  ="1.0 2.0 0.2";

// VectorAdd(vecA, vecB)
//Function to add two vectors of up to 3 elements
//each to each other
echo( VectorAdd(%start,%end) );  //output: 1 7 -3

//Bad Vector Math !!
echo( %start + %end );  //output: 1
```

3.3.7 Κλάσεις και αντικείμενα

Βασικό χαρακτηριστικό της γλώσσας σεναρίων Torque είναι ο **αντικειμενοστρεφής** (και όχι αντικειμενοστραφής[6]) χαρακτήρας (**object oriented**).

[6] Αντικειμενοστρεφής = αντικείμενο + στρέφομαι: Μιλάμε για αντικειμενοστρεφή προγραμματισμό δηλαδή τον προγραμματισμό που στρέφεται προς τα αντικείμενα. Παρόμοια λέμε εσωστρεφής,

Στο φυσικό κόσμο, παντού υπάρχουν αντικείμενα. Ο άνθρωπος σκέπτεται με βάση τα αντικείμενα. Έχει την ικανότητα της αφαίρεσης, που επιτρέπει να βλέπει τις εικόνες στην οθόνη ως αντικείμενα, όπως αυτοκίνητα, βουνά, δένδρα, αντί ως ξεχωριστές κουκκίδες με χρώματα. Μπορεί, αν θέλει, να σκέπτεται δάση αντί για δένδρα και σπίτια αντί για τούβλα.

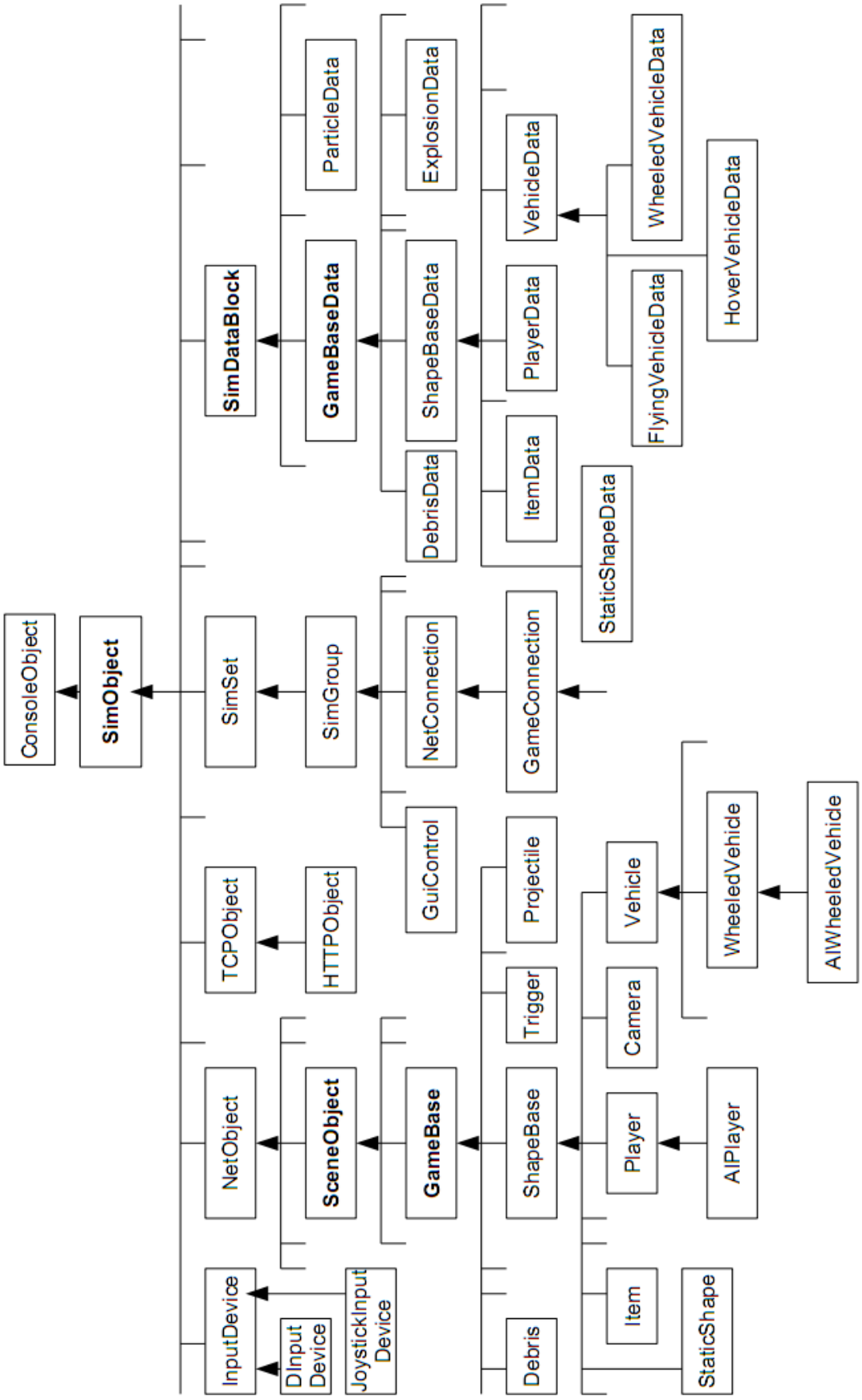
Τα αντικείμενα μπορούν να διαιρεθούν σε δυο κατηγορίες, δηλαδή στα κινούμενα και στα ακίνητα αντικείμενα. Τα κινούμενα αντικείμενα είναι "ζωντανά" με κάποια έννοια. Μπορούν να κινούνται στον κόσμο και να κάνουν "πράγματα". Τα ακίνητα δεν φαίνεται να κάνουν κάτι. Απλώς υπάρχουν! Όλα αυτά τα αντικείμενα ωστόσο, έχουν κάποια κοινά. Έχουν όλα ιδιότητες ή κατάσταση, όπως μέγεθος, σχήμα, χρώμα, που τα περιγράφουν. Όλα έχουν συμπεριφορές ή λειτουργίες που καθορίζουν τι κάνουν. Διάφορα αντικείμενα μπορεί να έχουν παρόμοιες ιδιότητες και παρόμοιες συμπεριφορές.

Ο αντικειμενοστρεφής προγραμματισμός μοντελοποιεί τα πραγματικά αντικείμενα με αντίστοιχα αντικείμενα σε προγράμματα. Εκμεταλλεύεται τις σχέσεις των κλάσεων, όπου τα αντικείμενα μιας κλάσης έχουν παρόμοιες ιδιότητες. Εκμεταλλεύεται την κληρονομικότητα των σχέσεων όπου νέες κλάσεις αντικειμένων παράγονται από ιδιότητες και λειτουργίες υπαρχουσών κλάσεων και προσθήκη μοναδικών χαρακτηριστικών. Δίνει την δυνατότητα για επικοινωνία μεταξύ αντικειμένων όπως συμβαίνει και στον πραγματικό κόσμο (οι άνθρωποι στέλνουν μηνύματα μεταξύ τους).

Η Torque υλοποιεί μία μεγάλη ιεραρχία κλάσεων, παρέχοντας βασική λειτουργικότητα σε τομείς που εκτείνονται από το χειρισμό και τον καθορισμό της συμπεριφοράς των αντικειμένων του κόσμου έως τη δικτύωση. Για παράδειγμα, η κλάση (class) "**ShapeBase**" αποτελεί τη βάση για όλες τις κλάσεις των αντικειμένων (objects) που έχουν κάποιο σχήμα και είναι ορατά στον κόσμο, ορίζοντας τις ιδιότητες (fields) και τις μεθόδους (methods) που είναι κοινές για τις κλάσεις αυτών των αντικειμένων. Η κλάση "**Player**" είναι υποκλάση (subclass) της "ShapeBase" και χρησιμοποιείται για τη δημιουργία αντικειμένων για τους παίκτες και την παρουσία αυτών μέσα στον κόσμο του παιχνιδιού (avatars). Κληρονομεί όλη τη λειτουργικότητα της "ShapeBase", και επιπλέον ορίζει εξειδικευμένες ιδιότητες και μεθόδους χρήσιμες για αντικείμενα αυτής της κατηγορίας. Ο χρήστης μπορεί να ορίσει επιπλέον ιδιότητες και μεθόδους.

Πριν συνεχίσουμε, στην επόμενη σελίδα υπάρχει μία μερική παρουσίαση της ιεραρχίας (δένδρο κληρονομικότητας) των κλάσεων που υποστηρίζει η μηχανή. Το πλήθος των υλοποιημένων κλάσεων καθιστά σχεδόν αδύνατη την πλήρη καταγραφή αυτών και των μεταξύ τους σχέσεων σε ένα και μόνο διάγραμμα!

εξωστρεφής και όχι εσωστραφής κτλ.


ConsoleObject
SimObject
SimDataBlock
SimSet
SimGroup
NetConnection
GameConnection
GameBaseData
ShapeBaseData
ParticleData
ExplosionData
DebrisData
VehicleData
PlayerData
ItemData
StaticShapeData
WheeledVehicleData
FlyingVehicleData
HoverVehicleData
TCPObject
HTTPObject
GuiControl
NetObject
SceneObject
GameBase
ShapeBase
Projectile
Trigger
Vehicle
Camera
Player
AIPlayer
WheeledVehicle
AIWheeledVehicle
InputDevice
DInput Device
JoystickInput Device
Debris
Item
StaticShape

Γενικά για τη δημιουργία ενός αντικείμενου, με τη γλώσσα σεναρίων Torque, ακολουθούμε την παρακάτω σύνταξη: όπου

%newObject = new τύποςΚλάσης (Όνομα : ΑντΠηγή, όρισμα0,...,όρισμαN) {

[datablock = ΌνομαDB;]

[υπάρχονΠεδίο0=αρχικήΤιμή0;]
...

[υπάρχονΠεδίοM=αρχικήΤιμήM;]

[απόΧρήστηΠεδίο0=αρχικήΤιμή0;]
...

[απόΧρήστηΠεδίοN=αρχικήΤιμήN;]
};

- **τύποςΚλάσης**: μία δηλωμένη στη μηχανή κλάση, η οποία προέρχεται απευθείας από την "SimObject" ή από υποκλάση της,
- **Όνομα** (προαιρετικά): είναι το όνομα του αντικείμενου. Μπορεί να μην είναι μοναδικό μιας και κατά τη δημιουργία του αντικειμένου αυτόματα η Torque αναθέτει σε αυτό έναν μοναδικό αριθμό-αναγνωριστικό (ID-handle),
- **ΑντΠηγή** (προαιρετικά): το όνομα κάποιου αντικειμένου που έχει οριστεί νωρίτερα και από το οποίο κληρονομεί ιδιότητες το νέο αντικείμενο,
- **όρισμα0...όρισμαN** (προαιρετικά): τα ορίσματα της συνάρτησης δημιουργίας (constructor) της κλάσης στην οποία ανήκει το αντικείμενο,
- **datablock** (προεραιτικά): το όνομα ενός ήδη υπάρχοντος και σχετικού με την κλάση datablock (βλ. παρακάτω).

Τα υπόλοιπα έχουν να κάνουν με αρχικοποίηση ιδιοτήτων-πεδίων που είτε η μηχανή έχει αντιστοιχήσει στην κλάση (και άρα είναι ιδιότητες του αντικειμένου), είτε ο προγραμματιστής θέλει να προσθέσει στο νέο αντικείμενο.

```
// Script #5: Create a new static object

$newObjectId=new StaticShape() {

   dataBlock="Rock";

   // SceneObject features-staticShapes objects derives from
   //SceneObject objects
   position="0 0 0";
   rotation="1 0 0 0";
   scale="1 1 1";
};
```

3.3.8 Datablocks

Μελετώντας το δένδρο κληρονομικότητας θα παρατηρήσουμε ότι για ορισμένες κλάσεις υπάρχουν και κάποιες συμπληρωματικές τους με την κατάληξη "*Data*". Αυτές οι συμπληρωματικές κλάσεις καλούνται "**datablocks**". Λέμε ότι στην κλάση, για παράδειγμα, "Player" αντιστοιχεί το datablock "PlayerData".

Τα "datablocks" ίσως είναι η πιο ιδιαίτερη δομή στην TorqueScript. **Πρόκειται για μία συλλογή ιδιοτήτων που έρχεται να συμπληρώσει τις ιδιότητες (αν αυτές υπάρχουν) της κλάσης στην οποία αντιστοιχεί το datablock**. Έτσι στο τελευταίο σενάριο (Script #5), το αντικείμενο τύπου "StaticShape" που δημιουργήσαμε δεν χαρακτηρίζεται μόνο από τα πεδία "position", "rotation" και "scale" αλλά και από τα πεδία-ιδιότητες που ορίζονται στο αντικείμενο "Rock". Το τελευταίο είναι το datablock για το στιγμιότυπο της κλάσης StaticShape (δηλαδή για το νέο μας αντικείμενο ή αλλιώς για το αντικείμενο $newObjectId μιας και φροντίσαμε να αποθηκεύσουμε, κατά τη δημιουργία, το μοναδικό αριθμό ID).

Τί ιδιότητες όμως ορίζει και αρχικοποιεί το datablock "Rock"; Αρχικά ας φανταστούμε ότι το αντικείμενο τύπου "StaticShape" που δημιουργήσαμε είναι ένας βράχος κάπου στον κόσμο του παιχνιδιού μας. Γίνεται αντιληπτό ότι το που ακριβώς "ζει" καθορίζεται από την τιμή του πεδίου "position". Στον πραγματικό κόσμο ένας βράχος όμως έχει και μάζα ή ιδιότητες όπως η τριβή. Αυτά θα μπορούσαν να αποτελέσουν πεδία για το datablock "Rock"!

```
// Script #6: Create the datablock for the object
//of script #5

datablock StaticShapeData( Rock ){

   shapeFile="~/data/items/rock.dts";
   mass=85;
   density=10;
   friction = 0.4;
};
```

Όπως φαίνεται η **δημιουργία αντικειμένων από datablocks** είναι παρόμοια με τη δημιουργία αντικειμένων από κλάσεις (ενότητα 3.3.7), με τις εξής όμως διαφορές:

- δεν μπορούμε να πάρουμε μία μεταβλητή με το αναγνωριστικό (ID-handle) του αντικειμένου ενός datablock και
- η δεσμευμένη λέξη "new" έχει αντικατασταθεί από τη δεσμευμένη λέξη "datablock".

Γενικώς στις κλάσεις παρατηρεί κανείς ελάχιστες έως καθόλου ιδιότητες και κυρίως μεθόδους, και στα datablocks το αντίθετο. Οι λίγες ιδιότητες που εμφανίζονται στις κλάσεις έχει νόημα να ορίζονται για κάθε αντικείμενο κλάσης ξεχωριστά και δεν θα είχε νόημα να τις χρησιμοποιήσουμε ομαδικά για πολλά αντικείμενα κλάσεων, όπως μπορούμε να κάνουμε με τις ιδιότητες που ορίζονται στα datablocks. Φαίνεται δηλαδή ότι στην TorqueScript όπου ήταν δυνατό οι ιδιότητες και οι μέθοδοι διαχωρίστηκαν επίτηδες. Τα datablocks ενθυλακώνουν τις ιδιότητες που είναι σχετικές με τα αντικείμενα της αντίστοιχης κλάσης και το ίδιο datablock μπορεί να χρησιμοποιηθεί από πολλά διαφορετικά αντικείμενα αυτής της κλάσης, με αποτέλεσμα να αποφεύγεται η επανάληψη της

αρχικοποίησης των ιδιοτήτων για κάθε αντικείμενο με παρόμοια λειτουργικότητα. Η διάκριση δεν είναι εντελώς ακριβής, και σίγουρα δεν είναι δεσμευτική για της ιδιότητες και μεθόδους που ορίζει ο χρήστης.

3.3.9 Συναρτήσεις και μέθοδοι

Γενικά στον προγραμματισμό οι συναρτήσεις, όπως γνωρίζουμε, είναι μικρότερα τμήματα κώδικα που διευκολύνουν την ανάπτυξη και συντήρηση μεγαλύτερων προγραμμάτων. Το ίδιο ισχύει και στην TorqueScript. Αντίστοιχα μιλάμε για μεθόδους όταν αναφερόμαστε σε συναρτήσεις που ανήκουν σε κάποια κλάση.

Η δήλωση μίας συνάρτησης ακολουθεί την παρακάτω σύνταξη:

```
function όνομαΣυνάρτησης(παράμετρος1,...,παράμετροςN){

//σώμα συνάρτησης
}
```

ενώ για τις μεθόδους:

```
function όνομαΚλάσης :: όνομαΜεθόδου(%this, παράμετρος1,...,παράμετροςN){

//σώμα μεθόδου
}
```

Στην TorqueScript αυτό που πρέπει να προσέξουμε είναι η πρώτη παράμετρος των μεθόδων (%this). **Το πρώτο όρισμα σε μια μέθοδο είναι η αναφορά στο συγκεκριμένο στιγμιότυπο - αντικείμενο της κλάσης για το οποίο κλήθηκε η μέθοδος, την οποία περνάει αυτόματα η μηχανή στις μεθόδους των κλάσεων όταν αυτές καλούνται.** Δεν έχει σημασία πώς ονομάζουμε το πρώτο όρισμα, αρκεί να ξέρουμε ότι πάντα θα είναι αυτή η αναφορά και ότι τα ορίσματα που θα περάσουμε εμείς στη μέθοδο ξεκινάνε από το δεύτερο όρισμα και μετά!

Στην περίπτωση που έχουμε μέθοδο ενός datablock (δεν πρέπει να ξεχνάμε ότι και τα datablocks είναι κλάσεις) είναι πιθανό να συναντήσουμε τη μεταβλητή %db αντί της %this. Δεν πρόκειται για κάτι διαφορετικό απλά μας θυμίζει ότι η αναφορά γίνεται σε ένα αντικείμενο datablock.

Ακολουθεί ένα παράδειγμα κλήσης μεθόδου για να γίνει κατανοητή η παραπάνω ιδιαιτερότητα της γλώσσας σεναρίων της TGE:

```
// Script #7: Method call

//a method of class Player
function Player::playFootStep(%this,%sound)
{
   if(%this.footStepsOn)
   {
      serverPlay3D(%sound,%this.getTransform());
```

```
      %this.schedule(250,playFootStep,%sound);
   }
}
...
...
$player = new Player() //create a new player object
  {
     dataBlock=MaleAvatar;
     inventory[HealthKit]=0;
     inventory[MP5]=0;
     inventory[MP5Ammo]=0;

  };
...
...
//Call playFootStep method
$player.playFootStep($footStepSound1);
//Same as
//$player.playFootStep($player,$footStepSound1);
...
...
```

Ολοκληρώνοντας την ενότητα, μία τελευταία παρατήρηση-προειδοποίηση: **στην TorqueScript δεν ισχύει ο πολυμορφισμός και η υπερφόρτωση συναρτήσεων**:

- συνάρτηση με ίδιο όνομα με προηγούμενη, ακόμα και αν έχουν διαφορετικό αριθμό παραμέτρων, διαγράφει την προηγούμενη συνάρτηση,
- αν κληθεί μία συνάρτηση με λιγότερες παραμέτρους από ότι είναι στη δήλωση, τότε η παράμετρος που αγνοείται συμπληρώνεται με κενή συμβολοσειρά (¨). Αντιθέτως αν χρησιμοποιηθούν περισσότερες παράμετροι τότε παραλείπονται οι επιπρόσθετες.

3.4 Συνοψίζοντας

Η "Torque" είναι μία πλήρης μηχανή παιχνιδιών και όχι απλά μία μηχανή απόδοσης. Στηρίζεται στην αρχιτεκτονική πελάτη-εξυπηρετητή κάτι που επιτρέπει την ανάπτυξη παιχνιδιών ˙ένός παίκτη" αλλά και περισσότερων με την ίδια ευκολία. Ο κώδικάς της είναι γραμμένος σε γλώσσα C++ και διατίθεται στους προγραμματιστές για να τον παραμετροποιήσουν σύμφωνα με τις ανάγκες τους. Στα μείον της μηχανής είναι η χρήση των αρχείων ".dts" και ".dif" για τα τρισδιάστατα μοντέλα. Πολύ δύσκολα θα βρει κανείς έτοιμα μοντέλα, σε αυτές τις μορφές, για να τα εντάξει στις εφαρμογές του. Επίσης τα βοηθητικά εργαλεία για τη σχεδίαση της σκηνής ή των γραφικών διασυνδέσεων δεν είναι καθόλου εύχρηστα. Αυτό το τελευταίο όμως δεν πρέπει να μας προβληματίζει μιας και τα πάντα θεωρούνται αντικείμενα (objects) και μπορούν να δημιουργηθούν με τη βοήθεια σεναρίων. Η "TorqueScript" είναι μία απλοποιημένη γλώσσα προγραμματισμού υψηλού επιπέδου, που ανήκει στην κατηγορία των λεγόμενων γλωσσών σεναρίων. Έχει χαρακτηριστικά της C++ με την εξής σημαντική διαφορά: στην "TorqueScript", δεν υπάρχει κανένας τρόπος να εντάξει ο χρήστης νέες κλάσεις

(αυτό μπορεί να γίνει μόνο αν αλλάξει τον κώδικα της μηχανής). Δεν υπάρχει κάτι αντίστοιχο του

```
class όνομαΚλάσης {

//ιδιότητες και μέθοδοι
};
```

αλλά μόνο η δυνατότητα δημιουργίας αντικειμένων από τις υλοποιημένες κλάσεις του πυρήνα της TGE.

Κεφάλαιο 4

Σχεδίαση και οργάνωση του παιχνιδιού

Το πρώτο στάδιο της ανάπτυξης ενός ηλεκτρονικού παιχνιδιού είναι αυτό της σχεδίασης του παιχνιδιού. Πρόκειται για το στάδιο όπου καθορίζεται το είδος του παιχνιδιού που πρέπει να υλοποιηθεί, η πλατφόρμα ανάπτυξης, το αγοραστικό κοινό στο οποίο θα απευθύνεται αλλά και θέματα του ίδιου του παιχνιδιού όπως σενάριο, επίπεδα-πίστες, όπλα που χρησιμοποιούνται κτλ.

4.1 Σχεδίαση του παιχνιδιού "Water War"

Ο σχεδιασμός ενός παιχνιδιού είναι από τις σημαντικότερες φάσεις ανάπτυξης, ίσως και η σημαντικότερη και είναι καλό να ξεκινάμε από αυτήν, κατά την δημιουργία ενός νέου παιχνιδιού. Βασικά είναι μία καταγραφή σε χαρτί του τι θα υλοποιηθεί στο παιχνίδι.

4.1.1 Σύντομη περιγραφή

Το παιχνίδι θα φέρει τον τίτλο "**Water War**". Ο παίκτης αναλαμβάνει τον ρόλο ενός στρατιώτη που προσπαθεί να επιβιώσει σε έναν νέο Παγκόσμιο Πόλεμο που έχει ξεσπάσει στη Γη. Οι μάχες διεξάγονται στα βάθη της αφρικάνικης ηπείρου.

4.1.2 Σενάριο

Βρισκόμαστε εν έτη 2037 και ο κόσμος συγκλονίζεται, για τρίτη φορά, από έναν Παγκόσμιο Πόλεμο. Ο Τζακ Σέπαρντ είναι πρώην γιατρός που "πέταξε" την ιατρική στολή και ντύθηκε στα χακί. Τώρα σαν ένας νέος Τζον Ράμπο καθοδηγείται από τη "Φωνή" και το μόνο που τον ενδιαφέρει είναι πως θα επιβιώσει στη νέα κατάσταση που έχει διαμορφωθεί. Αυτή τη φορά όλοι πολεμάνε με όλους, με μόνο στόχο να ζήσουν!

Η υπόθεση του παιχνιδιού είναι εμπνευσμένη από την ταινία "World War III" η οποία προβλήθηκε στη Σύνοδο του ΟΗΕ για το κλίμα, στην Κοπεγχάγη. Πρόκειται για μία μικρού-μήκους ταινία των φοιτητών Στέλιου Αλεξανδράκη και Μενέλαου Παμπουκίδη, του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης, η οποία κέρδισε το πρώτο βραβείο στο διεθνή διαγωνισμό με τίτλο "One minute to Save the World", το 2009 (http://www.1minutetosavetheworld.com/).

4.1.3 Είδος παιχνιδιού

Το "Water War" θα ανήκει στην κατηγορία των παιχνιδιών πρώτου-προσώπου-βολής [FPS (=first person shooter)] αλλά θα περιέχει και στοιχεία από παιχνίδια περιπέτειας (adventure games). Θα πρέπει δηλαδή ο χρήστης, πέρα από τις μάχες, να βρει κάποιες πληροφορίες ώστε να μπορέσει να ολοκληρώσει την αποστολή του.

4.1.4 Επιρροές

Δεν υπάρχει κάποιο συγκεκριμένο παιχνίδι από το οποίο να αντλήθηκαν στοιχεία για το "Water War". Θα ακολουθεί τη "βασική συνταγή" των παιχνιδιών τύπου "FPS": συνεχής δράση και πολλές εκρήξεις! Ίσως η μόνη επιρροή έχει να κάνει με την οθόνη όπου θα δηλώνει ότι ο παίκτης πληγώθηκε. Η ιδέα που έχει εφαρμοστεί στο "Modern Warfare 2" της "Activision", με τις κηλίδες αίματος που κάνουν πιο δύσκολη την περιήγηση στο χώρο, είναι κάτι το διαφορετικό (σε σύγκριση με τις παραδοσιακές υλοποιήσεις όπου στην οθόνη ακαριαία εμφανίζεται μία κόκκινη λάμψη) που θα ενταχθεί στην εφαρμογή.

4.1.5 Πλατφόρμα παιχνιδιού

Το παιχνίδι θα τρέχει και θα δοκιμαστεί σε Windows XP. Για την ανάπτυξη του θα χρησιμοποιηθεί η μηχανή παιχνιδιών Torque 1.5.2.

4.1.6 Περιβάλλον παιχνιδιού

Σύμφωνα με το σενάριο, η δράση διαδραματίζεται κάπου στην αφρικάνικη ήπειρο. Για το λόγο αυτό η σκηνή του παιχνιδιού θα δίνει την αίσθηση μίας ερημικής έκτασης, χωρίς ιδιαίτερη βλάστηση. Τα κτήρια που θα συνθέτουν τη σκηνή θα είναι λιτά και σε σκούρες αποχρώσεις έτσι ώστε να γίνεται αντιληπτό ότι βρισκόμαστε σε στρατιωτική περιοχή.

Ο ήρωας θα εισαχθεί στη σκηνή από τον ουρανό εκτελώντας πτώση με αλεξίπτωτο. Μία εξωτερική κάμερα θα δείχνει την πτώση του. Στη συνέχεια θα γίνει αλλαγή σε κάμερα πρώτου προσώπου, δηλαδή πλέον ο παίκτης θα βλέπει μέσα από τα μάτια του Τζακ. Ταυτόχρονα θα διαμορφωθεί και η βασική οθόνη περιήγησης [HUD (=Heads Up Display)] ώστε να εμφανίζει πληροφορίες για την ενέργεια, τα όπλα και τα αντικείμενα που θα έχει στη διάθεσή του ο παίκτης. Δεν θα υπάρχει κανένας περιορισμός κίνησης στη σκηνή του παιχνιδιού. Ο χρήστης θα είναι ελεύθερος να κινηθεί προς κάθε κατεύθυνση. Ωστόσο η θέση των οντοτήτων της σκηνής (για παράδειγμα λόφοι, κτήρια) θα είναι τέτοια που να αντιλαμβάνεται ποια πορεία πρέπει να ακολουθήσει. Σε αυτό θα συμβάλουν και οι πληροφορίες που θα συγκεντρώνει "μιλώντας " με κάποιον φυλακισμένο και παρατηρώντας χάρτες μέσα στις εγκαταστάσεις του εχθρού.

Όσον αφορά τους εχθρούς, αυτοί θα τοποθετηθούν σε συγκεκριμένες θέσεις δημιουργώντας πέντε χώρους δράσης. Δεν θα είναι στάσιμοι αλλά ανά τακτά χρονικά διαστήματα θα περιπολούν γύρω από το αρχικό σημείο.

4.1.7 Ροή παιχνιδιού

Η εφαρμογή θα ξεκινάει με μία εισαγωγική οθόνη με τον τίτλο του παιχνιδιού και θα ακολουθεί η οθόνη με το λογότυπο της "GarageGames". Στη συνέχεια ο χρήστης θα

βρεθεί στο βασικό μενού όπου και θα υπάρχει επιλογή για τερματισμό ή για εκκίνηση του "Water War".

Κατά τη διάρκεια του παιχνιδιού θα μπορεί να γίνει τερματισμός και επιστροφή στο περιβάλλον του λειτουργικού με το πλήκτρο "Esc". Αν ο παίκτης φτάσει στο στόχο τότε αυτόματα θα τερματίσει το παιχνίδι και πάλι.

4.1.8 Βασικές οντότητες του παιχνιδιού

Με τον όρο οντότητες εννοούμε τους παίκτες, τους εχθρούς, τα κτήρια και όποιο άλλο αντικείμενο βρίσκεται στη σκηνή του παιχνιδιού.

Ήρωας

Η βασική φιγούρα στο παιχνίδι θα είναι ο Τζακ. Θα είναι σχεδιασμένος έτσι ώστε να παραπέμπει σε στρατιώτη. Στο μεγαλύτερο μέρος του παιχνιδιού ο παίκτης θα βλέπει τον κόσμο μέσα από τα μάτια του ήρωα. Για το λόγο αυτό η σχεδίαση του θα είναι τέτοια ώστε να φαίνεται μόνο το χέρι που στηρίζει το όπλο (και φυσικά μέρος του όπλου). Η κάμερα πρώτου προσώπου θα έχει μία μέγιστη και μία ελάχιστη γωνία κατακόρυφης θέασης (look angle). Δεν πρέπει ο παίκτης καθώς κοιτάζει το έδαφος να φτάσει σε σημείο να βλέπει εσωτερικά του μοντέλου (η θέση της κάμερας θα βρίσκεται στο ύψος των ματιών του μοντέλου) ή όταν κοιτάζει προς τα πάνω να βλέπει σε γωνιά μεγαλύτερη των ενενήντα μοιρών.

Ωστόσο θα υπάρχει και επιλογή για αλλαγή σε κάμερα τρίτου προσώπου όπου θα παρατηρούμε τον ήρωα από πίσω. Για το λόγο αυτό θα υπάρχει ένα πλήρης μοντέλο (και όχι για παράδειγμα μόνο ένα χέρι που κρατάει το όπλο, όπως συμβαίνει σε κάποια παιχνίδια ίδιου τύπου). Σε αυτήν την περίπτωση θα φαίνονται και οι κινήσεις. Πρέπει λοιπόν να αναπτυχθούν τα γραφικά (animations) για κίνηση προς τα εμπρός, πίσω, στα πλάγια και άλμα εν στάση ή εν κινήσει.

Ο ήρωας θα ξεκινάει με μία μέγιστη ενέργεια (εκατό μονάδες). Κάθε φορά που θα δέχεται μία σφαίρα, η ενέργεια θα μειώνεται και μάλιστα "ή ζημιά" θα είναι ανάλογη με το σημείο που τραυματίστηκε (κεφάλι, θώρακας, πόδια). Εκτός από τις σφαίρες, μείωση ενέργειας θα έχουμε και κατά την πτώση από μεγάλα ύψη ή ανάλογα με την απόσταση του χαρακτήρα από σημείο που εκδηλώνεται έκρηξη (για παράδειγμα ο Τζακ πατάει κάποια νάρκη).

Όταν η ενέργεια μηδενίσει θα πρέπει να υπάρξει κάποιο κινούμενο γραφικό για το θάνατο του παίκτη. Ο παίκτης θα μπορεί να "ἀναγεννηθεί" κάπου μέσα στη σκηνή. Θα καθοριστούν τρία σημεία "ἀναγέννησης " (spawn points) και κάθε φορά θα επιλέγεται ένα με βάση τη θέση που προηγουμένως βρισκόταν. Δεν θα υπάρχει όριο για το πόσες φορές θα ξαναμπεί στο παιχνίδι.

Θα δίνεται βέβαια η δυνατότητα, στον παίκτη, να ανακτήσει μέρος της ενέργειάς του. Αυτό προϋποθέτει, κατά την περιήγησή του, να έχει βρει και να έχει συγκεντρώσει πακέτα ενέργειας (health kits). Οποιαδήποτε στιγμή με ένα πλήκτρο θα μπορεί να "γιατρευτεί". Κάθε πακέτο ενέργειας θα προσθέτει πενήντα μονάδες ενέργειας. Θα πρέπει λοιπόν ο παίκτης να χρησιμοποιεί τα πακέτα με σύνεση (για παράδειγμα αν έχει υποστεί "ζημιά" πέντε μονάδων και χρησιμοποιήσει το πακέτο τότε θα βρεθεί σε πολύ δύσκολη θέση όταν η ενέργειά του, αργότερα, θα έχει πέσει πολύ χαμηλά και δεν θα έχει κάποιο άλλο πακέτο).

Όσον αφορά τον οπλισμό, αυτός θα είναι ένα όπλο τύπου AK-47. Το όπλο αυτό είναι αυτόματο. Για τη χρήση του θα πρέπει να υπάρχουν διαθέσιμες σφαίρες. Σε διάφορες θέσεις στο έδαφος θα βρίσκονται πακέτα που θα περιέχουν πενήντα σφαίρες το καθένα. Ο παίκτης δεν θα μπορεί να έχει στη διάθεση του (να μαζέψει) πάνω από έξι τέτοια πακέτα (δηλαδή να έχει παραπάνω από τριακόσιες σφαίρες). Κατά τη διάρκεια της μάχης τα πολεμοφόδια θα μειώνονται και έτσι θα μπορούν να συγκεντρωθούν νέα πακέτα και πάλι.

Τέλος κάπου μέσα στα κτήρια θα υπάρξει μία μάσκα αερίων. Ο παίκτης από τη στιγμή που τη μαζεύει θα μπορεί να τη χρησιμοποιήσει με κάποιο πλήκτρο. Χωρίς αυτήν θα είναι αδύνατη η προσπέλαση ενός συγκεκριμένου δωματίου.

Εχθροί

Για τους εχθρούς θα χρησιμοποιηθεί ένα και μόνο τρισδιάστατο μοντέλο (διαφορετικό από του ήρωα) αλλά με δύο είδη υφών (textures). Κάποιοι θα είναι με καλυμμένα πρόσωπα και άλλοι όχι. Τα κινούμενα γραφικά θα είναι ίδια με αυτά του ήρωα.

Εκτός από την εμφάνιση, οι εχθροί θα διακρίνονται και ως προς τις αρμοδιότητές τους. Οι περισσότεροι από αυτούς με το που θα εντοπίσουν τον παίκτη θα του επιτεθούν. Δύο θα έχουν το ρόλο του να σημάνουν το συναγερμό όταν αντιληφθούν τον εισβολέα. Στη συνέχεια θα αρχίσουν να συμπεριφέρονται όπως και οι υπόλοιποι.

Η αρχική ενέργεια για κάθε ελεγχόμενο από τον υπολογιστή χαρακτήρα, θα οριστεί στις εκατό μονάδες και πάλι. Αυτή τη φορά όμως η "ζημιά" από τις σφαίρες θα είναι μεγαλύτερη σε σχέση με αυτή του παίκτη. Επίσης δεν θα υπάρχει η δυνατότητα ανάκτησης ενέργειας και από τη στιγμή που θα μηδενίσει ο εχθρός δεν θα επανεμφανιστεί.

Για όπλο θα έχουν ένα τύπου MP-5, με απεριόριστο αριθμό από σφαίρες. Ο προγραμματισμός θα πρέπει να είναι τέτοιος ώστε να υπάρξει μία μικρή παύση μετά από συνεχόμενες ριπές.

Φυλακισμένος

Θα δημιουργηθεί ένας χαρακτήρας φυλακισμένου με τον οποίο ο παίκτης θα μπορέσει να "μιλήσει" και να συγκεντρώσει πληροφορίες. Θα τοποθετηθεί σε κτήριο κοντά στο σημείο πτώσης του ήρωα. Αν τον ελευθερώσει τότε ο φυλακισμένος θα τρέξει μακριά από το σημείο που εξελίσσεται η δράση. Το τρισδιάστατο μοντέλο που θα χρησιμοποιηθεί θα είναι του ήρωα, με διαφορετική υφή.

Κτήρια

Κατά κύριο λόγο τα κτήρια θα είναι ενός δωματίου, χωρίς πολλές λεπτομέρειες, με ανοίγματα για είσοδο και παράθυρα. Στα κτήρια θα βρίσκονται αντικείμενα τα οποία και θα μπορεί να συλλέγει ο παίκτης ή να χειρίζεται.

Ένα από αυτά θα είναι δύο ορόφων και θα υπάρχει υπολογιστής στον οποίο και θα πρέπει να αποκτήσει πρόσβαση για να ανακτήσει έναν κωδικό. Αυτός ο κωδικός θα χρειαστεί για να ανοίξει η συρόμενη πόρτα ενός βιοχημικού εργοστασίου, που θα βρίσκεται προς το τέλος της αποστολής. Για να εισέλθει ο παίκτης σε αυτό θα πρέπει να διαθέτει ειδική μάσκα. Φορώντας τη μάσκα θα ψάξει για το διακόπτη που θα ανοίγει την πύλη που οδηγεί στο "στόχο".

Ο "στόχος " θα είναι ένα χρηματοκιβώτιο το οποίο εδώ δεν θα αποκαλύψουμε τι περιέχει. Η θέση του θα είναι μέσα σε πέτρινο κατασκεύασμα, ιδιαίτερα ψηλό και με πολλά σημεία από τα οποία θα μπορούν να "πεταχτούν" εχθροί. Γύρω από το οικοδόμημα θα υπάρχουν τείχη και για να περάσουμε από αυτά θα πρέπει, όπως είπαμε, να έχουμε επισκεφτεί το βιοχημικό εργοστάσιο.

Δένδρα

Στη σκηνή του παιχνιδιού θα υπάρχουν κάκτοι τύπου "Saguaro". Αν ο παίκτης έρθει σε επαφή (collide) με τον κάκτο θα εμφανιστούν πληροφορίες, σε παράθυρο μηνυμάτων, για το συγκεκριμένο είδος.

Ιδιαίτερης σημασίας θα είναι το "δένδρο της Tenere"[1] κάπου στη σκηνή. Πάλι με επαφή θα αντλεί ο ήρωας πληροφορίες που η προσεκτική ανάγνωσή τους θα αποτελεί "κλειδί" για την εξέλιξη του παιχνιδιού. Πιο συγκεκριμένα οι πληροφορίες θα επιτρέψουν στον παίκτη να μαντέψει τον κωδικό πρόσβασης του ηλεκτρονικού υπολογιστή, που θα βρίσκεται μέσα στο διώροφο κτήριο.

Οχήματα

Θα υπάρξουν μοντέλα στρατιωτικών οχημάτων όπως άρματα μάχης και τζιπ για να προσδώσουμε μεγαλύτερο ενδιαφέρον στη σκηνή. Ο ήρωας δεν θα επιτρέπεται να τα οδηγήσει.

Θα προσθεθεί όμως ένα πυροβόλο (κανόνι) το οποίο και θα χρησιμοποιείται όταν ο παίκτης θα βρίσκεται κοντά σε αυτό. Μήνυμα στην οθόνη θα τον ειδοποιεί ότι, για παράδειγμα, το πυροβόλο είναι οπλισμένο και με συγκεκριμένο πλήκτρο θα πραγματοποιεί βολή.

4.1.9 Συνθήκη τερματισμού

Το παιχνίδι θα τερματίζει όταν ο παίκτης φτάσει στο "στόχο".

4.1.10 Χειρισμός

Ο χειρισμός θα γίνεται με το πληκτρολόγιο και το ποντίκι. Για την κίνηση θα χρησιμοποιηθούν τα πλήκτρα **"WASD"** (w: κίνηση προς τα εμπρός, a: κίνηση προς τα αριστερά, s: κίνηση προς τα πίσω, d: κίνηση προς τα δεξιά) ενώ με το ποντίκι ο παίκτης θα ελέγχει την κάμερα και θα πυροβολεί. Στην πορεία θα καθοριστούν τα πλήκτρα με τα οποία θα αλληλεπιδρούμε με οντότητες του παιχνιδιού (συνηθίζεται το πλήκτρο "F"), θα χρησιμοποιούμε τα πακέτα ενέργειας και τη μάσκα.

4.2 Οργάνωση αρχείων

Όλα τα αρχεία βρίσκονται στο φάκελο "Water War". Ο φάκελος περιέχει περίπου 500 αρχεία. Τα 250 από αυτά αποτελούν τα δεδομένα του παιχνιδιού. Πρόκειται για

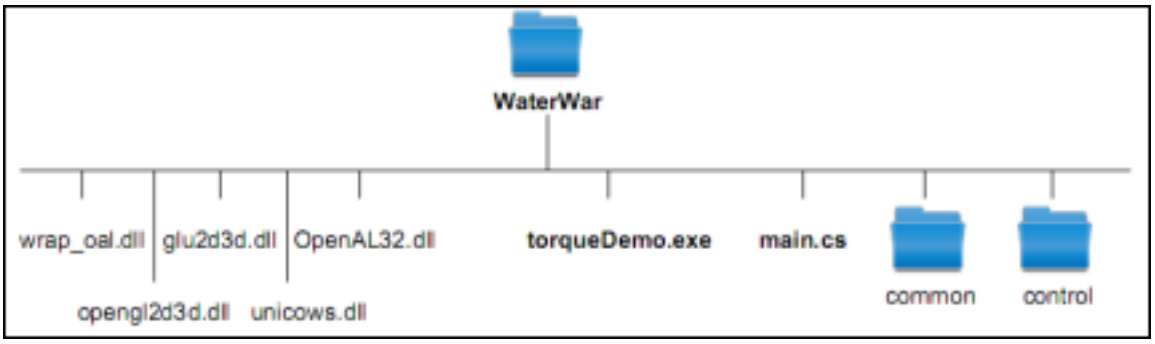
[1] Το δέντρο της Tenere ήταν το πιο απομονωμένο δέντρο στον κόσμο. Μια απόμερη ακακία, που μεγάλωσε στην έρημο της Σαχάρας, στο Νίγηρα. Δεν υπήρχε άλλο δέντρο σε ακτίνα 400 χλμ. Το δέντρο δεν υπάρχει πια καθώς το 1973 ένας μεθυσμένος, Λίβυος οδηγός φορτηγού έπεσε πάνω του!

τρισδιάστατα μοντέλα, εικόνες και αρχεία ήχου. Τα υπόλοιπα είναι τα τμήματα κώδικα που αναπτύχθηκαν για τη δημιουργία του παιχνιδιού.

4.2.1 "Φάκελος ρίζα" (Root Directory)

Δεν υπάρχει σχεδόν κανένας περιορισμός στο πως θα οργανωθούν τα αρχεία, σε παιχνίδια που αναπτύσσονται με την TGE. Το μόνο που απαιτείται είναι ένας βασικός φάκελος, μέσα στον οποίο θα βρίσκεται **το εκτελέσιμο αρχείο της εφαρμογής, το αρχείο "main.cs" (the root main module)** και φυσικά όλα τα υπόλοιπα αρχεία. Ο φάκελος αυτός καλείται "φάκελος ρίζα".

Στην εφαρμογής μας, "φάκελος ρίζα" είναι ο "WaterWar". Το "torqueDemo.exe"



Σχήμα 4.1: Ο "φάκελος ρίζα" του "Water War"

είναι το εκτελέσιμο αρχείο. Βλέπουμε ότι ο φάκελος ρίζα περιέχει και τις δυναμικές βιβλιοθήκες (.dll) για τη χρήση των απαραίτητων συναρτήσεων και υπορουτινών που σχετίζονται με τις βιβλιοθήκες "OpenGL" και "OpenAL". Να σημειωθεί και πάλι ότι το αρχείο "main.cs" είναι ιδιαίτερα σημαντικό και χωρίς αυτό (στο φάκελο όπου είναι και το εκτελέσιμο) το παιχνίδι δεν θα μπορεί να "τρέξει"!

Οι διαδρομές (paths) όλων των αρχείων που δίνονται σε αυτήν την εργασία είναι σχετικές (relative) και ξεκινάνε πάντα από τον κατάλογο-ρίζα. Για παράδειγμα, η διαδρομή "/control/main.cs" δηλώνει το αρχείο "main.cs" που βρίσκεται στον κατάλογο "control", ο οποίος με τη σειρά του βρίσκεται στον κατάλογο "WaterWar".

4.2.2 Το βασικό αρχείο "main.cs" (the root main module)

Κατά την εκκίνησή της η μηχανή ψάχνει για το αρχείο "main.cs", που πρέπει να βρίσκεται στον ίδιο κατάλογο με το εκτελέσιμο (βλ. Σχήμα 4.1). Συνεπώς το σημείο εισόδου της εφαρμογής είναι αυτό το αρχείο.

Σε αυτό γίνονται οι πρώτες αρχικοποιήσεις, δηλώνονται τα ορίσματα γραμμής (δεν έχουμε για τη δική μας εφαρμογή), καθορίζονται οι διαδρομές των αρχείων. Επίσης μπορούμε να ζητήσουμε κατά την μετάφραση του κώδικα να προστεθούν επιπλέον πληροφορίες, όπου θα βοηθήσουν στην αποσφαλμάτωσή του.

Τέλος είναι το αρχείο όπου ξεκινάνε και φορτώνονται στη μνήμη τα πρώτα αρχεία σεναρίων (εντολή **"exec"**). Συνηθίζεται τα αρχεία αυτά να ονομάζονται επίσης "main.cs". Έτσι, μέσα στο φάκελο-ρίζα, έχουμε τους φακέλους **"control"** και **"common"** όπου ο καθένας περιέχει το δικό του "main.cs" αρχείο. Αυτά με τη σειρά τους

φορτώνουν στη μνήμη άλλα αρχεία κ.ο.κ. με αποτέλεσμα σταδιακά όλος ο κώδικας να μεταφράζεται και να εκτελείται.

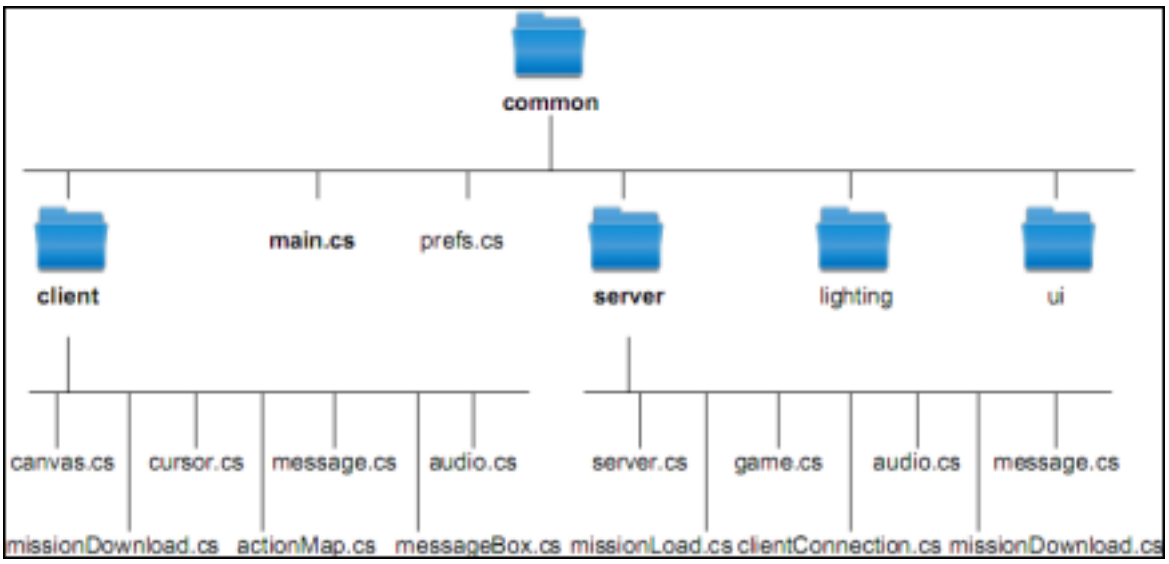
```
//==================================
//  ./main.cs
//
//  root main module for Water War
//
// The root script!
//==================================
$traceMode=false;
$winConsole=true;


trace($traceMode);   //add extra commentary in the log file
EnableWinConsole($winConsole);

$logModeEnabled=true;   //create a log file
SetLogMode(2); //overwrites existing log file and close log
   file at exit

//**********************************
//*OnExit is called directly from C++ code, whereas onStart is
//*invoked at the end of this file.
//**********************************
function OnExit(){}

//*************************
//*Default startup function
//*************************
function OnStart(){}

// --> Notify engine about required folder paths
$pathList="common;control";
SetModPaths($pathList);
// <--

// --> In-line program statements
exec("common/main.cs"); //load into memory the common/main.cs
   module
exec("control/main.cs"); //load into memory the control/main.cs
    module
// <--

OnStart(); //call all versions of OnStart() that appear in the
   pathList
```

4.2.3 Φάκελος “common”

Ο φάκελος “common” περιλαμβάνει συναρτήσεις, μεθόδους και ορισμούς κλάσεων και datablock που καθορίζουν στοιχειώδεις γενικές λειτουργίες της εφαρμογής.

Συγκεκριμένα ο “common” κατάλογος περιλαμβάνει τους καταλόγους “server”,

“client”, “lighting”, “ui” και το αρχείο καθορισμένων/προεπιλεγμένων τιμών “prefs.cs”. Οι βασικές λειτουργίες των αρχείων που περικλείονται στα παραπάνω είναι η κατασκευή του εξυπηρετητή, η αρχικοποίησή του και η καταστροφή του, όταν αυτή απαιτείται, η δημιουργία σύνδεσης μεταξύ του τελευταίου και των πελατών και η αποσύνδεση όταν αυτή είναι επιθυμητή. Επίσης καθορίζονται οι τρεις φάσεις φόρτωσης της εφαρμογής (θα τις αναλύσουμε παρακάτω), η υλοποίηση συναρτήσεων που διευκολύνουν την επικοινωνία μεταξύ πελατών και εξυπηρετητή, η αρχικοποίηση του γραφικού συστήματος (δημιουργία “Canvas” στον πελάτη) και θέματα που έχουν να κάνουν με τη γραφική διασύνδεση του χρήστη όπως η χρήση κέρσορα και τα παράθυρα μηνυμάτων για τερματισμό της εφαρμογής.



Σχήμα 4.2: Τα σημαντικότερα αρχεία του “common”

4.2.4 Φάκελος “control”

Λόγω της οργάνωσης της εφαρμογής σύμφωνα με την αρχιτεκτονική πελάτη - εξυπηρετητή, ο κώδικάς του φακέλου “control” χωρίζεται στο τμήμα του πελάτη (“client”), στο τμήμα του εξυπηρετητή (“server”) και στο τμήμα των δεδομένων (“data”).

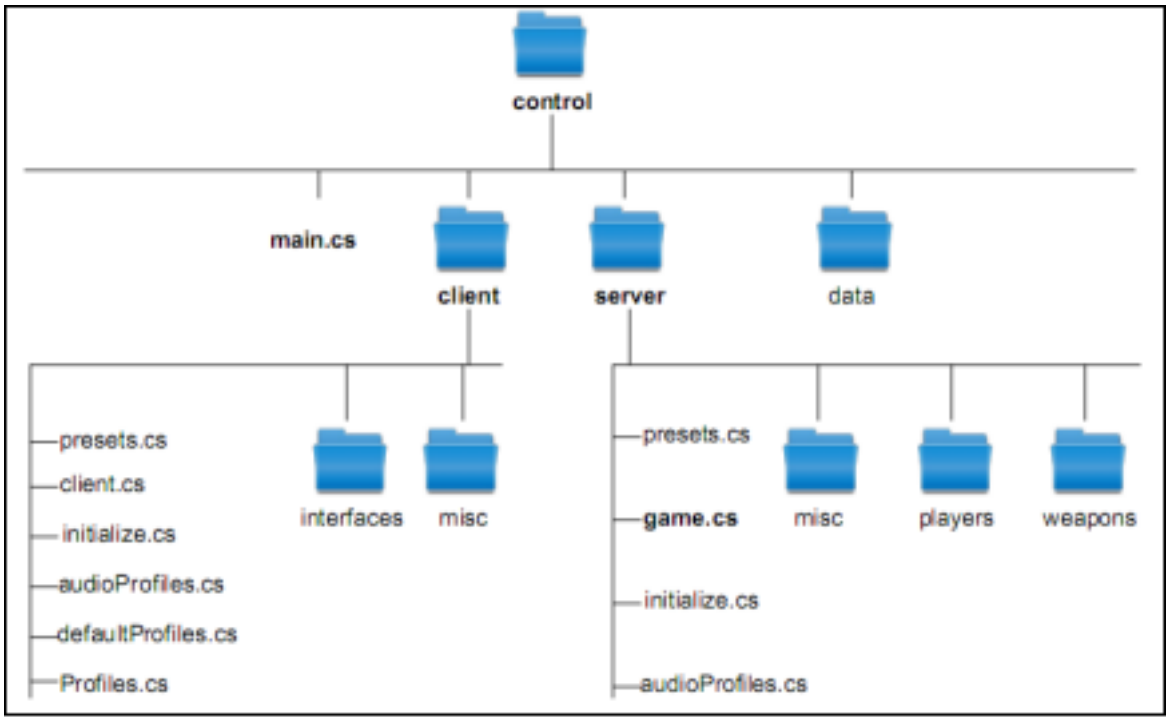
Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο, η οργάνωση αυτή διευκολύνει τον καταμερισμό εργασίας, την αποδοτικότερη διαχείριση των δεδομένων και την απόκρυψη πληροφοριών από το ένα μέρος στο άλλο όταν αυτό απαιτείται. Επίσης διευκολύνει τη μελλοντική επέκταση της εφαρμογής ώστε να εξυπηρετεί περισσότερους πελάτες αν κάτι τέτοιο είναι επιθυμητό. Σε αυτήν την περίπτωση, ο κώδικας των πελατών και του εξυπηρετητή μπορούν να εκτελούνται σε διαφορετικούς υπολογιστές και να επικοινωνούν μέσω δικτύου.

Το τμήμα του πελάτη είναι υπεύθυνο κυρίως για τη σύνδεση με τον εξυπηρετητή, τον έλεγχο της εισόδου και της εξόδου και την παρουσίαση της εικόνας.

Το τμήμα του εξυπηρετητή είναι επιφορτισμένο με άλλου είδους αρμοδιότητες όπως η δημιουργία και διαγραφή αντικειμένων, η κίνηση των χαρακτήρων, η ανίχνευση

σύγκρουσης μεταξύ αντικειμένων, η υλοποίηση των "triggers" και γενικότερα ο καθορισμός των περισσότερων συναρτήσεων και μεθόδων που αφορούν στην κατασκευή και τη συμπεριφορά των αντικειμένων της εφαρμογής. Έτσι το τμήμα του κώδικα που υλοποιεί το μεγαλύτερο κομμάτι του παιχνιδιού βρίσκεται στο φάκελο "server".

Ο κατάλογος "data" δεν περιέχει κώδικα, αλλά πόρους οπτικοακουστικής φύσης (3D μοντέλα, εικόνες, αρχεία ήχου), που χρησιμοποιούνται για την αναπαράσταση του κόσμου. Ο κώδικας του πελάτη και του εξυπηρετητή είναι υπεύθυνος για το χειρισμό αυτών των πόρων. Για παράδειγμα, ο πελάτης επιλέγει ένα μουσικό κομμάτι που επαναλαμβάνεται όσο δείχνει το κεντρικό μενού ή ο εξυπηρετητής δημιουργεί ένα αντικείμενο του datablock "PlayerData" και χρησιμοποιεί κάποιο 3D μοντέλο που θα αναπαριστά το αντικείμενο αυτό στον κόσμο.



Σχήμα 4.3: Τα περιεχόμενα του καταλόγου "control"

4.3 Συνοψίζοντας

Η σχεδίαση που παρουσιάστηκε στο πρώτο μέρος του κεφαλαίου είναι πολύ πρόχειρη. Στη βιομηχανία των ηλεκτρονικών παιχνιδιών, οι σχεδιαστές δίνουν μεγάλη σημασία στη λεπτομέρεια. Για παράδειγμα θα έπρεπε να είχε αναφερθεί το πόσο "ζημιά" υφίσταται ο παίκτης όταν χτυπηθεί στο κεφάλι, ή την ακριβή θέση που θα τοποθετηθούν οι οντότητες. Ο προγραμματιστής που θα διαβάσει το σχέδιο του παιχνιδιού πρέπει να μπορεί να υλοποιήσει τα αντικείμενά του και τη συμπεριφορά τους.

Το παιχνίδι συνεχώς θα βελτιώνεται κατά τη διάρκεια της παρούσας εργασίας και μέχρι την ολοκλήρωσή της. Μετά και από τις τελευταίες δοκιμές το σίγουρο είναι ότι

θα υπάρξουν αλλαγές πάνω στη σχεδίαση. Έτσι αυτό που παρουσιάστηκε στην ουσία είναι ο βασικός κορμός με τον οποίο ξεκίνησε η διπλωματική.

Με βάση αυτόν αναπτύχθηκαν τα κομμάτια κώδικα του παιχνιδιού και οργανώθηκαν σύμφωνα με την αρχιτεκτονική πελάτη - εξυπηρετητή. Να σημειωθεί ότι η Torque έρχεται με μερικά παραδείγματα εφαρμογών, που ακολουθούν μία συγκεκριμένη οργάνωση αρχείων. Η εργασία αυτή δεν χρησιμοποιεί κανένα από αυτά τα παραδείγματα και έχει γραφτεί ξεκινώντας από το μηδέν. Βέβαια, τα παραδείγματα αυτά χρησίμευσαν σαν οδηγός και σε μεγάλο βαθμό ακολουθείται παρόμοια οργάνωση με αυτά. Η επιλογή αυτή σημαίνει ότι έχουμε μόνο την απαραίτητη λειτουργικότητα και πιο συνεκτική, αφού η πλειοψηφία των λειτουργιών προστέθηκε σταδιακά και μόνο όταν αυτή χρειάστηκε. Αυτό σε καμία περίπτωση δε σημαίνει ότι αν θέλουμε να δημιουργήσουμε ένα νέο παιχνίδι θα πρέπει να ξεκινήσουμε πάλι από την αρχή! Ο κώδικας "common" θα παραμείνει ίδιος ενώ στο "control" τμήμα, αν το παιχνίδι είναι πάλι τύπου FPS, οι αλλαγές θα είναι σε συγκεκριμένα αρχεία.

Πριν συνεχιστεί η ανάλυση της λειτουργικότητας του παιχνιδιού θα ακολουθήσει η δημιουργία των οντοτήτων του. Δυστυχώς οι τύποι αρχείων που χρησιμοποιεί η μηχανή, για τα τρισδιάστατα μοντέλα, δεν είναι τόσο συνηθισμένοι (μάλλον θεωρούνται πλέον ξεπερασμένοι) οπότε είμαστε αναγκασμένοι να μοντελοποιήσουμε μόνοι μας ή να τροποποιήσουμε κάποια μοντέλα ευρέως χρησιμοποιούμενων τύπων (για παράδειγμα ".3ds").

Κεφάλαιο 5

Δημιουργία οντοτήτων του παιχνιδιού

Τα ηλεκτρονικά παιχνίδια είναι τρόπος διασκέδασης για πολλούς. Κάποιοι σίγουρα γνωρίζουν ότι πίσω από αυτό που βλέπουν στην οθόνη τους υπάρχει κώδικας γραμμένος από μια ομάδα προγραμματιστών. Οι περισσότεροι όμως δεν μπαίνουν στη διαδικασία να αναλογιστούν πώς κινείται ο εικονικός τους χαρακτήρας, στο Sims! Στρέφουν την προσοχή τους στα αντικείμενα της σκηνής που παίζουν το ρόλο του κτιρίου, του αυτοκινήτου και οτιδήποτε άλλο υπάρχει και στον πραγματικό κόσμο. Απολύτως λογικό αυτό, μιας και έχουμε να κάνουμε με μια οπτική εφαρμογή. Τα μοντέλα λοιπόν είναι βασικά συστατικά ενός ηλεκτρονικού παιχνιδιού. Σε ένα παιχνίδι δράσης, ένας κύβος σαν βασικός χαρακτήρας σίγουρα θα προκαλέσει την ολοκληρωτική του αποτυχία στην αγορά, σε αντίθεση με ένα τρισδιάστατο μοντέλο άντρα στρατιώτη, με εμφανή τα σημάδια από τις μάχες στο πρόσωπό του.

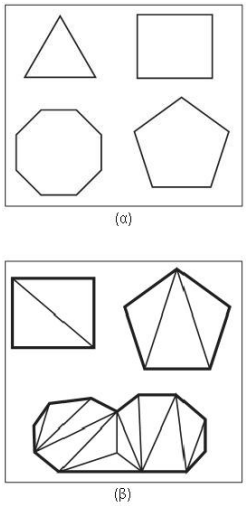
Το κεφάλαιο 5 ασχολείται με τη δημιουργία των οντοτήτων του παιχνιδιού όπως είναι οι τρισδιάστατοι χαρακτήρες και τα κτήρια. Γίνεται επίσης αναφορά στο λεγόμενο "παραλληλεπίπεδο του ουρανού" ("SkyBox") και παρουσιάζεται μία εφαρμογή που αναπτύχθηκε ώστε να παράγουμε εύκολα εδάφη (terrain) για τις αποστολές του παιχνιδιού.

5.1 Τρισδιάστατα μοντέλα (3D models)

Μιας και είναι το πρώτο κεφάλαιο το οποίο ασχολείται με τα τρισδιάστατα μοντέλα, κάποιες βασικές πληροφορίες θα δώσουν μία πιο ολοκληρωμένη εικόνα πάνω στο θέμα. Ας ξεκινήσουμε από τα αυτονόητα! Τί είναι τρισδιάστατο μοντέλο;

Στον πραγματικό κόσμο αντιλαμβανόμαστε αντικείμενα τριών διαστάσεων (ύψος, πλάτος, βάθος). Όταν θέλουμε να τα αναπαραστήσουμε σε μια οθόνη υπολογιστή, πρέπει να αναλογιστούμε ότι η τελευταία είναι μία δισδιάστατη συσκευή εξόδου (ύψος από την κορυφή της οθόνης μέχρι τη βάση και πλάτος από αριστερά της οθόνης έως δεξιά). **Το προϊόν της διαδικασίας με την οποία αποδίδουμε ένα αντικείμενο του πραγματικού κόσμου σε μια δισδιάστατη συσκευή απεικόνισης (για παράδειγμα οθόνη, χαρτί εκτύπωσης), ονομάζεται τρισδιάστατο μοντέλο (3D model)**. Το μοντέλο αυτό στη συνέχεια μπορεί να γίνει πιο ρεαλιστικό με την προσθήκη οπτικών χαρακτηριστικών όπως σκιές, υφές. **Η όλη διαδικασία υπολογισμού της εμφάνισης του τρισδιάστατου μοντέλου, δηλαδή η**

μετατροπή σε οντότητα που να μπορεί να αναπαρασταθεί σε δισδιάστατη συσκευή και η τελική εικόνα, καλείται "απόδοση" (rendering).



Σχήμα 5.1: (α) Πολύγωνα, (β) Πλέγματα

Αν ανατρέξουμε στο κεφάλαιο 2 και συγκεκριμένα στην ενότητα για τη μηχανή απόδοσης (2.3.1), θα δούμε ότι είχε γίνει μία πρώτη αναφορά στα μοντέλα τριών διαστάσεων. Θέλοντας να γενικεύσουμε το παράδειγμα του κύβου, στην εν λόγω ενότητα, μπορούμε να πούμε ότι ένα μοντέλο αναπαριστά ένα τρισδιάστατο αντικείμενο χρησιμοποιώντας μια συλλογή **σημείων (vertices)**. Τα σημεία συνδέονται με γραμμές που ονομάζονται **ακμές (edges)**. Εάν συνδέσουμε τρία ή περισσότερα σημεία με ακμές, έτσι ώστε να σχηματιστεί μια κλειστή **επιφάνεια (face/facet)**, δημιουργείται ένα **πολύγωνο (polygon)**. Το πιο απλό πολύγωνο είναι το **τρίγωνο (triangle)**.

Οι κάρτες γραφικών σήμερα είναι σχεδιασμένες ώστε να επεξεργάζονται χιλιάδες τρίγωνα το δευτερόλεπτο. Εξαιτίας αυτού, στη μοντελοποίηση χρησιμοποιούνται κατά κύριο λόγω τρίγωνα αντί άλλων πολυγώνων. Εξάλλου κάθε πολύπλοκο πολύγωνο μπορεί να διαιρεθεί σε μια συλλογή τριγώνων που καλείται **πλέγμα (mesh)**.

Η παράσταση αντικειμένων με πολύγωνα **(πολυγωνικό μοντέλο)** είναι η παλαιότερη και η πιο διαδεδομένη, ιδιαίτερα δε όταν τα αντικείμενα αυτά έχουν λίγες επίπεδες επιφάνειες. Η δυσκολία παράστασης αυξάνεται για πολύπλοκα, μη επίπεδα αντικείμενα οι επιφάνειες των οποίων πρέπει να προσεγγισθούν από ένα σύνολο πολυγώνων. Δεν υπάρχει κάποιος κανόνας για το πόσα πολύγωνα πρέπει να χρησιμοποιηθούν για αυτήν την προσέγγιση, αλλά ακόμα και διαισθητικά κάποιος θα παρατηρήσει ότι περιοχές με μεγάλη κοιλότητα θα χρειαστεί να έχουν μεγαλύτερο αριθμό πολυγώνων ανά μονάδα επιφάνειας από περισσότερο επίπεδες περιοχές!

5.2 Τεχνικές σχεδίασης τρισδιάστατων μοντέλων

Υπάρχουν αρκετές διαφορετικές τεχνικές σχεδίασης που χρησιμοποιούν όσοι ασχολούνται με τη δημιουργία μοντέλων. Η ποικιλία αυτή εξαρτάται τόσο από τα εργαλεία όσο και από τα δεδομένα τα οποία είναι διαθέσιμα σε κάποιον για τη σχεδίαση του μοντέλου. Έστω για παράδειγμα ότι έχουμε αναλάβει την τρισδιάστατη απεικόνιση κάποιας περιοχής της Ελλάδας. Φυσικά και δεν θα δουλέψουμε με εργαλεία όπως το "Autodesk 3ds Max"[1]! Πρώτα θα επισκεφτούμε τις αρμόδιες υπηρεσίες όπου θα μας δώσουν τα δεδομένα της περιοχής σε κάποια ψηφιακή μορφή (DLG-O, DEM, SDTS,...). Στη συνέχεια θα βρούμε το λογισμικό εκείνο όπου θα μπορέσει να διαβάσει τα παραπάνω δεδομένα και να μας εξάγει το μοντέλο που θέλουμε.

Στην παρούσα εργασία απαιτούνται μοντέλα για ηλεκτρονικό παιχνίδι. Ακολουθείται λοιπόν η φιλοσοφία "**του μικρού αριθμού πολυγώνων**" **(low-poly modeling)**. Από όσα δηλαδή λιγότερα πολύγωνα απαρτίζεται ένα μοντέλο, τόσο το καλύτερο για τη σχεδίαση. Με όσα περισσότερα πολύγωνα "φορτώνεται" κάποιο μοντέλο, τόσα

[1] Πρόκειται για το πλέον διαδεδομένο εμπορικό πρόγραμμα δημιουργίας τρισδιάστατων, ρεαλιστικών μοντέλων, φωτορεαλισμού, κίνησης και ειδικών εφέ.

λιγότερα θα είναι διαθέσιμα για άλλα στιγμιότυπα του ίδιου, ή για άλλα μοντέλα, σε ένα δεδομένο καρέ για κάποιο δεδομένο βαθμό καρέ.

Η σχεδίαση με ελάχιστο αριθμό πολυγώνων απαιτεί πρώτα από όλα την ένταξη των μοντέλων σε κατηγορίες. Κάποια από αυτά θα έχουν πρωτεύον ρόλο στο παιχνίδι, ενώ άλλα θα έρχονται σε δεύτερη μοίρα και μπορεί να περνάνε και απαρατήρητα. Όπως είναι λογικό τα τελευταία δεν θα μοντελοποιηθούν με μεγάλη λεπτομέρεια. Για παράδειγμα ένας βράχος στην άκρη του δρόμου δεν είναι τόσο σημαντικός και μπορεί κάλλιστα να αναπαρασταθεί από ένα πλέγμα τριάντα το πολύ πολυγώνων. Αντίθετα ένα κτήριο αμέσως θα τραβήξει την προσοχή του παρατηρητή/παίκτη και συνεπώς δεν θα μπορούσε να υλοποιηθεί με λιγότερα από τριακόσια πολύγωνα.

Στη συνέχεια ο σχεδιαστής πρέπει να υπολογίσει τι θα συμβεί αν όλα τα μοντέλα βρεθούν ταυτόχρονα στην σκηνή. Ένας χαρακτήρας από πέντε χιλιάδες πολύγωνα μπορεί να διαχειριστεί άνετα από κάποιο μηχάνημα. Τί γίνεται αν το παιχνίδι είναι δικτυακό και έχουμε δέκα παίκτες που ξεκινάνε από το ίδιο σημείο της πίστας; Θεωρούμε ότι κάθε παίκτης έχει το δικό του χαρακτήρα (**avatar**), που αναπαριστάται από το ίδιο μοντέλο των πέντε χιλιάδων πολυγώνων. Άρα ο υπολογιστής χρειάζεται να επεξεργαστεί πενήντα χιλιάδες πολύγωνα (χωρίς να λογαριάζουμε τα πολύγωνα από το έδαφος, τα κτήρια, τα όπλα)!

Τέλος συνυπολογίζεται η γενιά του επεξεργαστή και της κάρτας γραφικών. Ίσως τα πέντε χιλιάδες πολύγωνα, στην παραπάνω περίπτωση, να είναι αρνητικός παράγοντας για κάποιον υπολογιστή της προηγούμενης δεκαετίας. Είναι σοφή κίνηση ένα παιχνίδι να μπορεί να είναι εκτελέσιμο σε ένα ευρύ φάσμα χρηστών, με μια αξιοπρεπή ταχύτητα, ενώ παράλληλα να επιτρέπεται στους πιο τεχνολογικά "δυνατούς" να έχουν εξαιρετικό ρυθμό καρέ.

Στην περίπτωση της Torque Game Engine οι ενδεικτικές τιμές για να επιτευχθεί υψηλός βαθμός λεπτομέρειας είναι οι εξής:

- χαρακτήρες - 2250 πολύγωνα,
- οχήματα - 1500 πολύγωνα,
- όπλα - 500 πολύγωνα,
- υπόλοιπα αντικείμενα - έως 400 πολύγωνα.

5.2.1 Με βασικά σχήματα (Shape Primitives)

Όταν αναφερόμαστε σε βασικά σχήματα, εννοούμε παραλληλεπίπεδο (box), κύλινδρο (cylinder), σφαίρα (sphere), επιφάνεια (plane). Σχεδόν όλα τα λογισμικά για δημιουργία μοντέλων τριών διαστάσεων προσφέρουν τη δυνατότητα εύκολης σχεδίασης των παραπάνω.

Η συγκεκριμένη τεχνική είναι ο πιο γρήγορος τρόπος σχεδίασης μοντέλου με μικρό αριθμό πολυγώνων. Επιλέγεται κάποιο σχήμα (για παράδειγμα κύβος) που ταιριάζει στο τμήμα που θέλουμε να "κτίσουμε" και τροποποιούνται κατάλληλα οι ακμές του ώστε να πάρουμε το επιθυμητό αποτέλεσμα.

5.2.2 Με σταδιακή δημιουργία πολυγώνων (Incremental Polygon Construction)

Με την εν λόγω μέθοδο πρώτα τοποθετούμε τα σημεία στις κατάλληλες θέσεις. Στη συνέχεια επιλέγουμε, συνήθως ανά τριάδες,τα σημεία με τέτοιο τρόπο ώστε να δημιουργηθεί κάποια επιφάνεια του μοντέλου μας.

Στη τεχνική της "σταδιακής δημιουργίας πολυγώνων" προηγείται, τις περισσότερες φορές, η φωτογράφηση,το σκιτσάρισμα του σχήματος προς απόδοση, σε διάφορες διευθύνσεις (για κάποια ανθρώπινη μορφή για παράδειγμα, προφίλ και ανφάς). Έτσι μπορούμε να χρησιμοποιήσουμε τα τελευταία ως οδηγούς για το πού και πώς θα δημιουργηθούν τα πολύγωνα. Παρακάτω θα δούμε στην πράξη τη χρήση τέτοιων 'όδηγών".

5.2.3 Με κατά άξονα ή αυθαίρετη 'έξαγωγή'' (Axial/Arbitrary Extrusion)

Ξεκινάμε με ένα βασικό σχήμα, συνήθως παραλληλεπίπεδο, το υποδιαιρούμε και επιλέγουμε τα πολύγωνα εκείνα που θα μας δώσουν τα νέα πλέγματα. Με την "κατά άξονα εξαγωγή" το νέο πλέγμα αναπτύσσεται προς έναν μόνο άξονα σε αντίθεση με την αυθαίρετη όπου γίνεται προς όποιον άξονα χρειαστεί.

5.2.4 Συνδυασμός των παραπάνω και άλλες τεχνικές

Το ποια τεχνική θα ακολουθήσει κάποιος για τη δημιουργία του μοντέλου είναι καθαρά προσωπική επιλογή και έχει να κάνει τόσο με τα δεδομένα και τα εργαλεία που έχει στη διάθεσή του, όπως είπαμε και στην αρχή, αλλά και από την εμπειρία του πάνω στη σχεδίαση. Μπορεί ανά πάσα στιγμή να χρησιμοποιήσει έναν συνδυασμό των παραπάνω. Για παράδειγμα, ο κορμός σε ένα ανθρώπινο μοντέλο μπορεί να γίνει με την τεχνική των "βασικών σχημάτων" (κύλινδροι για πόδια και χέρια, ορθογώνιο παραλληλεπίπεδο για στήθος) ενώ το πρόσωπο με "σταδιακή δημιουργία πολυγώνων".

Οι τεχνικές που περιγράφτηκαν είναι ένα μόνο δείγμα για μοντελοποίηση σχημάτων. Κοινό χαρακτηριστικό τους είναι ότι στηρίζονται στη φιλοσοφία της σχεδίασης με μικρό αριθμό πολυγώνων. Στην ίδια κατηγορία ανήκει και η "**χαρτογράφηση τοποθεσιών**" **(topographical shape mapping)** με την οποία παίρνουμε τρισδιάστατες μορφές περιοχών/εδαφών. Βέβαια πρόκειται για μια αυτοματοποιημένη διαδικασία όπως περιγράφτηκε στην αρχή της ενότητας.

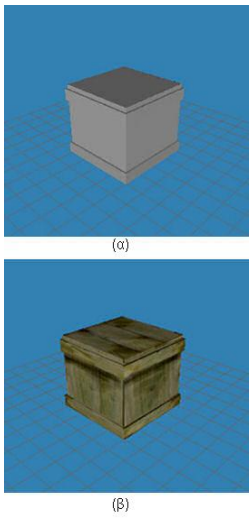
5.3 Υφές (Textures)

Οι 'ύφές" είναι οι αφανής ήρωες των παιχνιδιών τριών διαστάσεων. Βασικός στόχος τους είναι να δημιουργούν και να ενισχύουν την επιθυμητή ατμόσφαιρα στο παιχνίδι.

Ωστόσο τις περισσότερες φορές χρησιμοποιούνται για να λύσουν τα χέρια των σχεδιαστών ως προς τις λεπτομέρειες που πρέπει να προσθέσουν σε ένα μοντέλο. Για παράδειγμα ένα τείχος από πέτρες δεν χρειάζεται να μοντελοποιηθεί με διαφορετικά, μικρά, ορθογώνια παραλληλεπίπεδα, τοποθετημένα με μαεστρία το ένα πάνω στο άλλο. Αρκεί η δημιουργία ενός και μόνο παραλληλεπιπέδου στις διαστάσεις του τείχους και η χρήση της κατάλληλης υφής που θα δώσει την αίσθηση της δημιουργίας με πολλούς λίθινους όγκους.

Με την ίδια τεχνική μπορούμε να αναπαραστήσουμε και το φυσικό περιβάλλον χωρίς να χρειάζεται να κατασκευάσουμε τρισδιάστατα μοντέλα για τα μικροαντικείμενα της σκηνής όπως κλαδιά, θάμνοι, χαλίκια. Ακόμα και η παρουσία ενός καταρράκτη στη σκηνή μπορεί να γίνει με τον ίδιο τρόπο. Και δεν περιοριζόμαστε μόνο σε αυτά. Σκεφτείτε ένα κουτί (έξι πλευρές) που περικλείει τη σκηνή δράσης του παιχνιδιού μας. Η εφαρμογή των σωστών υφών σε αυτό θα μας δώσει την εντύπωση ενός θόλου, που θα αποτελεί τον ουρανό, πάνω από τον ορίζοντα ("skybox").

Πρακτικά οι υφές είναι αρχεία γραφικών: ".gif", ".jpg" και ".png"[2]. Έχει επικρατήσει να χρησιμοποιούνται τα δύο τελευταία μιας και τα ".gif" αρχεία (Graphics Interchange Format) δεν προσφέρουν κάτι περισσότερο από ότι τα κατάληξης "png". Η αλήθεια είναι ότι πρόκειται για τα μόνα αρχεία "ψηφιοκουκκίδων"[3] που επιτρέπουν κάποιου είδους κίνησης (animation). Ωστόσο στο χώρο των τρισδιάστατων εφαρμογών υπάρχουν πολλές τεχνικές για κίνηση, οπότε το δυνατό χαρτί των εν λόγω μορφών εικόνας αχρηστεύεται αν αναλογιστούμε ότι η διεπαφή DirectX (βλέπε "OpenGL, DirectX και η σχέση τους με μία μηχανή παιχνιδιού", ενότητα 2.4) δεν υποστηρίζει υφές τύπου ".gif"!


(α)

(β)

Σχήμα 5.2: Μοντέλο (α) χωρίς και (β) με υφές

Από εκεί και πέρα και οι εικόνες με κατάληξη "jpg" (Join Photo Group) αποφεύγονται στην τεχνική των υφών. Μπορεί ο "άλγόριθμος συμπίεσης JPEG" να είναι αρκετά αποδοτικός και να παράγει μικρά σε μέγεθος αρχεία, ωστόσο είναι απωλεστικός (lossy). Πετυχαίνει δηλαδή μεγάλη συμπίεση με το να μειώνει την ποιότητα της εικόνας. Αυτό συνεπάγεται ότι αν έχουμε κάποια υφή και χρειαστεί να την επεξεργαστούμε τρεις ή τέσσερις φορές τότε μετά από κάθε νέα αποθήκευση θα έχουμε μια νέα εικόνα, ποιοτικά υποδεέστερη από τις προηγούμενες. Κάτι τέτοιο βέβαια δεν είναι και τόσο αποθαρρυντικό για τη χρήση αρχείων ".jpg". Συνήθως οι υφές δημιουργούνται μία φορά και μετά απλώς υπάρχουν "ξεχασμένες " εκεί που αποθηκεύτηκαν.

Τί γίνεται όμως αν θέλουμε να μοντελοποιήσουμε υλικά όπως το γυαλί, υλικά δηλαδή τα οποία είναι διαφανή και σου επιτρέπουν να κοιτάξεις μέσα από αυτά; Η υφή που θα χρησιμοποιήσουμε σε αυτή την περίπτωση πρέπει να έχει κατάληξη "png" (Portable Network Graphics)! Σε σχέση με τα προηγούμενα αρχεία (".jpg") μόνα αυτά υποστηρίζουν διαφάνεια (transparency) και επιτρέπουν τον έλεγχό της μέσω της ρύθμισης της αδιαφάνειας (opacity).

Τέλος όσον αφορά την ανάλυση των εικόνων που θα χρησιμοποιηθούν ως υφές,

[2]Τα τρία σημαντικότερα που συναντάμε κατά κόρον στα γραφικά. Υπάρχουν και άλλα, όπως τα ".bmp", που καλό είναι να τα χρησιμοποιούμε μόνο όταν δεν έχουμε άλλη επιλογή.

[3]Τα αρχεία γραφικών (εικόνες, φωτογραφίες) είναι δύο τύπων: τα αρχεία τύπου "ψηφιοκουκκίδων" (raster) και τα "διανυσματικά" (vector). Για ευκολία θα χρησιμοποιούμε τους αγγλικούς όρους.

Raster αρχεία είναι αυτά στα οποία η εικόνα σχηματίζεται από άθροισμα κουκκίδων. Κάθε σημείο της εικόνας αποτελείται από μια κουκκίδα (pixel) που καθορίζει το χρώμα ή τον τόνο του γκρι αν πρόκειται για ασπρόμαυρη.

Vector αρχεία, είναι αυτά στα οποία η εικόνα σχηματίζεται από άθροισμα γεωμετρικών σχημάτων (ευθείες, κύκλοι, τόξα, καμπύλες και πάχη αυτών).

Raster αρχεία είναι τα ".jpg", ".gif", ".tif", ".bmp", ".pcx", ".fpx" (FlashPix), ".pcd" (PhotoCD), ".png". Vector αρχεία είναι τα ".cdr" (Corel Draw), ".dxf" (AutoCad), ".ps" kai ".eps" (postscript), ".pdf" (Acrobat), ".tif".

δεν θα πρέπει να ξεπερνάει τα 512x512 pixels. Καλό θα ήτανε να χρησιμοποιούμε τη μέγιστη, επιτρεπτή ανάλυση μόνο σε ειδικές περιπτώσεις. Υφές 256x256 pixels θα ήτανε μια σοφή επιλογή ώστε να εξοικονομήσουμε πόρους μνήμης. Μπορούν να χρησιμοποιηθούν και άλλα μεγέθη όπως 32x32, 128x128, 128x256. Αυτό που έχει σημασία είναι οι διαστάσεις να είναι κάθε φορά δύναμη του δύο για να αποφύγουμε τυχόν ασυμβατότητες με τα συστήματα σχεδίασης των καρτών γραφικών.

5.4 Τα εργαλεία

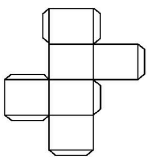
Πριν ξεκινήσει η παρουσίαση της δημιουργίας του τρισδιάστατου χαρακτήρα, θα γίνει μία σύντομη αναφορά στα εργαλεία που βοήθησαν στο έργο αυτό. Πρόκειται για λογισμικά με γραφικό περιβάλλον όπου επέτρεψαν να σχεδιαστεί το μοντέλο βήμα-βήμα, να αποκτήσει κίνηση και υφή και να εξαχθεί σε τέτοια μορφή ώστε να μπορεί να το διαχειριστεί η μηχανή παιχνιδιών Torque.

Το πρώτο εργαλείο είναι αυτό με το οποίο δημιουργήθηκε το μοντέλο των τριών διαστάσεων. Σήμερα στην αγορά κυριαρχεί η εφαρμογή "Autodesk 3ds Max". Ωστόσο το κόστος απόκτησης του προγράμματος είναι ιδιαίτερα υψηλό. Η πιο αξιόλογη, διαδεδομένη και ελεύθερη, εφαρμογή, ακούει στο όνομα Blender. Για κάποιον που πρώτη φορά ασχολείται με τον τομέα της σχεδίασης ίσως είναι η πιο καλή επιλογή. Αυτοί όμως που έχουν ήδη δουλέψει με κάποια εμπορική εφαρμογή σίγουρα θα δυσαρεστηθούν με το γραφικό περιβάλλον, μιας και δεν ακολουθεί τα πρότυπα των τελευταίων.

Στην παρούσα εργασία χρησιμοποιήθηκε το όχι τόσο γνωστό "Milkshape 3D" [19]. Πρόκειται για ένα πρόγραμμα σχεδίασης τρισδιάστατων μοντέλων, με μικρό αριθμό πολυγώνων, το οποίο δημιουργήθηκε από τον Mete Ciragan.

Τί μας προσφέρει το εν λόγω λογισμικό; Πρώτα από όλα ένα πολύ φιλικό περιβάλλον με τα πιο σημαντικά εργαλεία που συναντάμε και σε εμπορικές εφαρμογές. Τέσσερα ξεχωριστά παράθυρα για παρακολούθηση της δουλειάς μας από διαφορετικές γωνίες (front, right, top, 3D views), εύκολη δημιουργία βασικών σχημάτων (σφαίρα, γεώσφαιρα, παραλληλεπίπεδο, κύλινδρος), βασικές πράξεις (για παράδειγμα μετακίνηση, περιστροφή, κλιμάκωση), "σκελετώδη γραφικά" (βλέπε "Κινούμενα γραφικά σχέδια", ενότητα 2.3.2) και πάνω από εβδομήντα διαφορετικές μορφές αρχείων εξαγωγής.

Εκτός από τα μοντέλα, δημιουργήθηκαν και οι απαραίτητες υφές. Για το σκοπό αυτό χρειάστηκε ένα πρόγραμμα επεξεργασίας εικόνων και συγκεκριμένα το "Paint Shop Pro".



Σχήμα 5.3: "Ξετυλίγοντας " έναν κύβο

Το "Paint Shop Pro" της εταιρίας Jasc (τώρα πλέον της Corel) είναι ένα πρόγραμμα ψηφιακής επεξεργασίας εικόνας με το οποίο μπορούμε να διαχειριστούμε φωτογραφίες αλλά και να δημιουργήσουμε δικές μας εικόνες . Άλλα παρόμοια προγράμματα που κυκλοφορούν είναι : "Photoshop" (ίσως το πιο διαδεδομένο από όλα) , "Corel PhotoPaint" , "Ulead PhotoImpact", "The GIMP". Το "Paint Shop Pro" ξεχωρίζει ως προς τη μεγάλη ευκολία εκμάθησής του αλλά και στη δυνατότητα να μπορεί να διαχειρίζεται και διανυσματικά γραφικά.

Για την εφαρμογή των υφών τώρα, σε ένα μοντέλο τριών διαστάσεων, απαιτείται μία διαδικασία την οποία στα ελληνικά θα μπορούσαμε να την ονομάσουμε, με τον άκομψο όρο, "ξετύλιγμα" (UV

Unwrapping). Φανταστείτε ένα χάρτινο κύβο του οποίου "πειράζουμε" τις ακμές με τέτοιο τρόπο ώστε όλες οι έδρες του να "άπλωθούν" πάνω στην επιφάνεια του γραφείου μας. Το τελικό αποτέλεσμα θα είναι σαν αυτό της εικόνας 5.3.

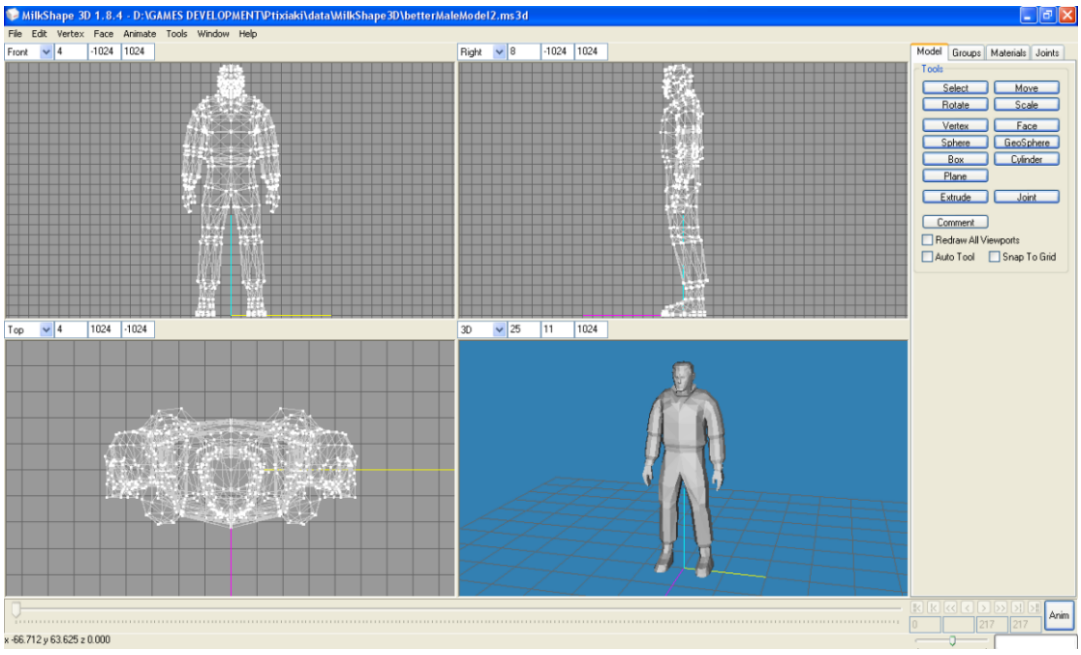
Κάτι παρόμοιο μας επιτρέπει και το ελεύθερο πρόγραμμα "LithUnwrap" [20]. Από τη στιγμή που έχουμε "μία δισδιάστατη μορφή" του μοντέλου μπορούμε να καθορίσουμε (μέσω του προγράμματος αυτού και άλλων τέτοιου τύπου) ποιο τμήμα της υφής θα εμφανίζεται στο κάθε τμήμα του μοντέλου.

5.5 Δημιουργώντας έναν χαρακτήρα

Παρακάτω περιγράφεται η διαδικασία δημιουργίας ενός από τους χαρακτήρες που θα εμφανίζονται στο παιχνίδι. Η ίδια διαδικασία μπορεί να ακολουθηθεί και για οποιοδήποτε μοντέλο του παιχνιδιού πλην των κτηρίων (όπως εξηγείται παρακάτω).

5.5.1 Μοντελοποίηση

Σε πρώτη φάση χρησιμοποιήθηκε το "MilkShape 3D" για τη δημιουργία του πλέγματος του μοντέλου. Για το βασικό κορμό των χαρακτήρων, για τα άνω και κάτω άκρα, για το λαιμό και για το πίσω τμήμα του κρανίου, εφαρμόστηκε η τεχνική των "βασικών σχημάτων". Το χέρι μοντελοποιήθηκε με "κατά άξονα εξαγωγή", ενώ το πρόσωπο με "σταδιακή δημιουργία πολυγώνων". Κάτι άλλο που πρέπει επίσης να σημειωθεί είναι



Σχήμα 5.4: Μοντελοποίηση χαρακτήρα με MilkShape 3D

ότι προτιμήθηκε "συνεχής, πλεγματοειδής μοντελοποίηση" (continuous-mesh model). Δηλαδή όλα τα επιμέρους τμήματα που απαρτίζουν το μοντέλο (για παράδειγμα στήθος,

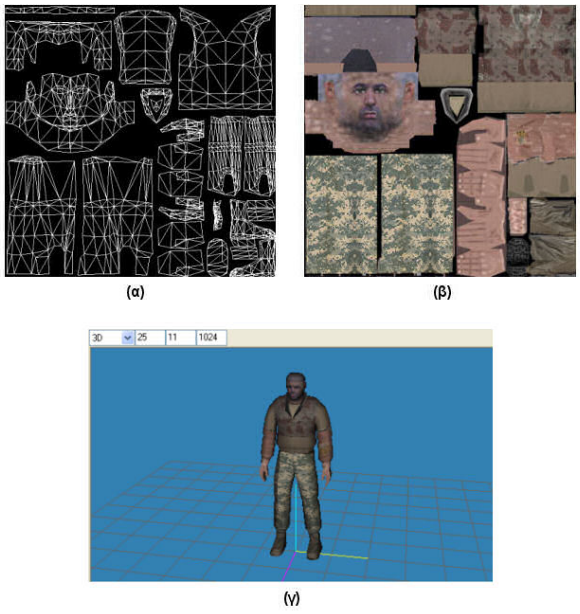
πάνω άκρα, λαιμός) συνδέονται μεταξύ τους και δεν αποτελούν επιμέρους τμήματα όπως συμβαίνει στην "κατά τμήματα, πλεγματοειδής μοντελοποίηση" (segmented-mesh model). Σύμφωνα με την τελευταία μπορούμε, για παράδειγμα, να έχουμε ξεχωριστό πλέγμα για το στήθος και ξεχωριστό για το αριστερό, άνω άκρος και να τα τοποθετήσουμε τόσο κοντά το ένα στο άλλο δίνοντας την αίσθηση ότι συνδέονται. Ωστόσο συνίσταται να αποφεύγεται η συγκεκριμένη τακτική μιας και ενδέχεται να προκληθούν σφάλματα κατά την εξαγωγή του μοντέλου αλλά και κατά την εφαρμογή υφών. Αυτό συμβαίνει γιατί τα προγράμματα που είναι υπεύθυνα για τα εν λόγω θέματα μπορεί να αναγνωρίσουν ως ένα και όχι ως δύο τα σημεία που βρίσκονται πολύ κοντά το ένα στο άλλο. Έτσι "χάνουμε" ένα σημείο το οποίο μπορεί να καταστρέψει τη γεωμετρία του μοντέλου.

5.5.2 Δημιουργία υφών

Η δημιουργία των υφών για τα μοντέλα του παιχνιδιού απαιτεί τρεις φάσεις.

Στην πρώτη χρησιμοποιείται το εργαλείο "LithUnwrap" το οποίο αποτυπώνει όλες τις επιφάνειες του τρισδιάστατου μοντέλου σε ένα σχέδιο δύο διαστάσεων. Προκύπτει ένα είδος χάρτη πάνω στο οποίο "ζωγραφίζουμε" τις υφές.

Αυτό το τελευταίο γίνεται με προγράμματα επεξεργασίας εικόνας όπως το "PaintShop Pro". Στην παρούσα εργασία όλα τα αρχεία υφών είναι σε μία από τις επιτρεπτές για τη μηχανή μορφές αρχείων: ".gif", ".jpg" και ".png".



Σχήμα 5.5: (α) Δημιουργία χάρτη υφών - (β) Σχεδίαση υφής - (γ) Εφαρμογή υφής

Η τελευταία φάση είναι αυτή της εφαρμογής της υφής στο μοντέλο. Γίνεται πάλι με το εργαλείο μοντελοποίησης.

5.5.3 Προσθήκη κίνησης (Animation)

Ένα στατικό μοντέλο όσο ρεαλιστικό και αν δείχνει δεν θα παρουσιάζει κανένα ενδιαφέρον αν δεν κινείται. Για προσθήκη κίνησης στα μοντέλα, το πρώτο που χρειάζεται είναι να αναπτυχθεί ένα είδος σκελετού.

Όπως και με το ανθρώπινο σώμα, ο σκελετός για τον χαρακτήρα μας είναι η κατασκευή εκείνη που "κρύβεται" πίσω από το δέρμα (όπου δέρμα για το μοντέλο είναι η υφή που σχεδιάσαμε στην προηγούμενη ενότητα).

Για άλλη μία φορά εργαζόμαστε με την εφαρμογή "MilkShape 3D". Απαιτείται η δημιουργία κατάλληλων κόμβων στο πλέγμα του μοντέλου(nodes/joints). Κάθε κόμβος μπορεί να συσχετιστεί με κάποιον άλλον με σχέση "γονέα-παιδιού". Κάθε κίνηση/περιστροφή του γονέα, επηρεάζει πλέον και τη θέση του παιδιού!

Στο μοντέλο του χαρακτήρα που έχουμε ως παράδειγμα, ο πρώτος κόμβος (συνηθίζεται να έχει το όνομα "Base") που δημιουργούμε είναι στη βάση του χαρακτήρα, ανάμεσα στα πόδια. Από εκεί και πέρα μπορούμε να δημιουργήσουμε όσους κόμβους κρίνουμε απαραίτητο. Μια ιδιαίτερα απλή αναλογία είναι τρεις κόμβοι για τον κορμό, ένας με δύο για το κεφάλι και δύο με τρεις κόμβοι για κάθε άκρο του χαρακτήρα.

Όταν πλέον αναπτυχθεί ο σκελετός για το μοντέλο ακολουθεί **η διαδικασία ανάθεσης (rigging). Είναι η διαδικασία με την οποία αναθέτουμε στους κόμβους του σκελετού, τμήματα του πλέγματος του μοντέλου.** Πλέον κάθε κίνηση του κόμβου θα επηρεάζει και τη θέση των σημείων που συνθέτουν το σχετικό με τον κόμβο τμήμα του πλέγματος.

Τέλος αυτό που απομένει είναι η δημιουργία της ακολουθίας των καρέ. Για να μπορέσει η μηχανή παιχνιδιών "Torque" να "δώσει" κίνηση σε κάποιο μοντέλο, για να έχουμε δηλαδή κινούμενα γραφικά σχέδια (animations), χρειάζεται ένας αριθμός από "πόζες " για το σκελετό (για παράδειγμα τρέξιμο, άλμα από στάση, βήματα προς τα πίσω). Η μηχανή στη συνέχεια υπολογίζει τη θέση του πλέγματος του μοντέλου από τη θέση και την περιστροφή των κόμβων.

5.6 Έδαφος

Σε κάποια παιχνίδια η δράση λαμβάνει χώρο μέσα σε κτήρια και σε υπόγειους διαδρόμους. Σε άλλα η σκηνή είναι ένας εξωτερικός χώρος όπως μία κοιλάδα. Τα περισσότερα όμως, όπως και το δικό μας, συνδυάζουν και τα δύο.

Στη δεύτερη περίπτωση πρέπει να δημιουργήσουμε την οντότητα για το έδαφος (terrain), όπου είναι ένας συνδυασμός γεωγραφικών δεδομένων (για παράδειγμα λόφοι και κοιλάδες) και χαρακτηριστικών του εδάφους (άμμος, γρασίδι). Για τα γεωγραφικά δεδομένα, στην ανάπτυξη παιχνιδιών, χρησιμοποιούμε τρισδιάστατα μοντέλα. Η μέθοδος των υφών, στη συνέχεια, εφαρμόζεται με στόχο να δώσουμε την αίσθηση του χωμάτινου ή πλούσιου σε χαμηλή βλάστηση εδάφους.

Οι περισσότερες μηχανές παιχνιδιών διαθέτουν δικό τους υποσύστημα ("Terrain Manager") το οποίο είναι υπεύθυνο για τη μοντελοποίηση του εδάφους. Συνήθως ο διαχειριστής εδάφους συνοδεύεται με κάποια εργαλεία όπως παραγωγή τυχαίας μορφολογίας, επεξεργασία υπαρχόντων εδαφών αλλά και δυνατότητα φόρτωσης εικόνων. Αν ανατρέξουμε σε προηγούμενο κεφάλαιο, στην παρουσίαση που έγινε για την Torque, θα καταλάβουμε για τι ακριβώς εργαλεία μιλάμε ("Terrain Editor", "Terrain Terraform Editor").

Αυτό που χρησιμοποιήθηκε στην εργασία είναι το εργαλείο για φόρτωση εικόνων. Προσέξτε ότι προς το παρόν ενδιαφερόμαστε μόνο γεωγραφικά για το πως θα δημιουργήσουμε το έδαφος του παιχνιδιού. Δηλαδή θέλουμε να δημιουργήσουμε λόφους, πεδιάδες, φαράγγια και όποια άλλη μορφολογία θα μπορούσαμε να σκεφτούμε! Άρα ποιος ο λόγος να φορτώσουμε εικόνες ή πιο σωστά τι είδους εικόνες είναι αυτές;

5.6.1 Υψομετρικοί χάρτες

Η Torque λοιπόν έχει τη δυνατότητα να παράγει εδάφη με βάση κάποιον **υψομετρικό χάρτη (heightmaps)**. **Ένας τέτοιος χάρτης περιέχει πληροφορίες για τα υψόμετρα μιας γεωγραφικής περιοχής και συνήθως αποτελείται από ισοϋψείς καμπύλες. Οι ισοϋψείς καμπύλες αντιπροσωπεύουν μια σειρά από σημεία που απέχουν εξίσου από την επιφάνεια της θάλασσας**. Ας πάρουμε ως παράδειγμα την παρακάτω εικόνα. Βλέπουμε ότι οι ισοϋψείς καμπύλες προκύπτουν αν "κόψουμε" το βουνό με κάθετα επίπεδα και τις τομές αυτές τις προβάλουμε στην επιφάνεια του χαρτιού μας. Οι ισοϋψείς καμπύλες είναι κλειστές γραμμές και σε κάποιο σημείο τους έχουν ένα νούμερο το οποίο δείχνει την απόσταση από τη θάλασσα.

Σχήμα 5.6: (α) Ένα βουνό με πέντε επίπεδα - (β) Κάθετα επίπεδα στο βουνό - (γ) Ισοϋψής καμπύλη - (δ) Τρόπος παρουσίασης υψομετρικού χάρτη με αποχρώσεις του γκρι

Πέρα από τη χρήση ισοϋψών καμπυλών, ένας υψομετρικός χάρτης μπορεί να αποδοθεί και με τη βοήθεια χρωματικής κλίμακας. Κάθε υψόμετρο, σε αυτήν την περίπτωση, αντιστοιχεί και σε διαφορετικό χρώμα. Όταν δεν υπάρχουν μεγάλες φυσικές διαφορές (μεγάλος αριθμός διαφορετικών υψομέτρων) ο χάρτης χρωματίζεται με διαφορετικές αποχρώσεις του γκρι, **με τα πιο φωτεινά μέρη να δηλώνουν μεγαλύτερο ύψος σημείου σε σχέση με τις πιο σκούρες περιοχές**. Η τελευταία αυτή απεικόνιση είναι που ενδιαφέρει εμάς.

5.6.2 Δημιουργία εδαφών με χρήση υψομετρικών χαρτών

Στην μηχανή παιχνιδιών "Torque", το έδαφος δημιουργείται από έναν άπειρα επαναλαμβανόμενο χάρτη ισοϋψών. Ο χάρτης εισάγεται ως μία 256x256 εικόνα, τύπου ".png". Η εικόνα μπορεί να είναι "πλήρης χρώματος " (full-color, 24-bit) ή και μικρότερη. Για παράδειγμα αν, όπως αναφέραμε και προηγουμένως, δεν έχουμε μεγάλες εναλλαγές υψομέτρου, τότε μία εικόνα σε αποχρώσεις του γκρι (grey-scale, 8-bit) μπορεί να μας δώσει 256 ($=2^8$) διαφορετικά επίπεδα (elevations). Στο σχήμα 5.27.δ μπορούμε να διακρίνουμε πέντε επίπεδα, με διαφορετικούς τόνους γκρι για το κάθε ένα.

Η μηχανή χρησιμοποιεί αυτήν την εικόνα (τον υψομετρικό χάρτη) ως βάση (home tile) και την εφαρμόζει επαναλαμβανόμενα σε όλη την επιφάνεια της σκηνής του παι-

χνιδιού, με ένωση-ανάμειξη των ακμών της (edge-blended). Ποιο είναι το μέγεθος της βάσης όμως σε σχέση με τον πραγματικό κόσμο; Είπαμε ότι έχουμε μια εικόνα με 256 pixels σε κάθε πλευρά. Για ευκολία ας φανταστούμε το έδαφος που θα δημιουργήσουμε ως ένα επίπεδο πλέγμα τετραγώνων. Η μηχανή "παίρνει" κάθε pixel της εικόνας και το αντιστοιχεί σε κάποιο τετράγωνο. Σαν δεδομένο, από τις προδιαγραφές της μηχανής, γνωρίζουμε ότι το τετράγωνο αυτό έχει μέγεθος 8 WUs (world units-μονάδα που χρησιμοποιείται στην Torque) ανά ακμή. Άρα όταν "σκανάρουμε" όλη την εικόνα θα έχουμε ένα σύνολο 256 τέτοιων τετραγώνων ανά πλευρά πλέγματος και συνεπώς 2048 (=256x8) WUs ανά πλευρά. Προσεγγιστιστικά αυτό αντιστοιχεί σε 2048 μέτρα ανά πλευρά, για τη βάση μας! Σε κάθε τετράγωνο πλέον αντιστοιχεί και ένα ύψος.

Να τονιστεί και πάλι ότι οι διαστάσεις αυτές αφορούν το μέγεθος ενός τμήματος του εδάφους και όχι το σύνολο του εδάφους! Αν κάποιος θεωρεί ότι τα περίπου δύο χιλιόμετρα ανά πλευρά δεν του είναι αρκετά, ως βάση, μπορεί να αλλάξει την παράμετρο "squareSize", η οποία δέχεται τιμές από δύο έως εξήντα τέσσερα. Στην ουσία πρόκειται για τον αριθμό που καθορίζει πόσα WUs αντιστοιχούν σε κάθε pixel. Συνίσταται να παραμένει στην αρχική της τιμή οκτώ. Ωστόσο πιο ενδεικτικές τιμές, σε περίπτωση αλλαγής, είναι οι δυνάμεις του δύο (π.χ. 2, 16, 32).

Καταλάβαμε πως η μηχανή χειρίζεται τους υψομετρικούς χάρτες, οπότε έφτασε η στιγμή να δούμε πως τους δημιουργούμε! Για αρχή χρειαζόμαστε δεδομένα, δηλαδή τα ύψη με βάση τα οποία θα σχεδιάσουμε το χάρτη. Αν θέλουμε να μοντελοποιήσουμε πραγματική περιοχή τότε μπορούμε να απευθυνθούμε σε κάποια γεωλογική υπηρεσία.[4] Στην συγκεκριμένη εργασία τα δεδομένα είναι τυχαία.

5.6.3 Σχεδίαση υψομετρικού χάρτη με πρόγραμμα επεξεργασίας εικόνας

Για έναν απλό, υψομετρικό χάρτη μπορούμε να δουλέψουμε με ένα πρόγραμμα επεξεργασίας εικόνας όπως το "Paint Shop Pro". Δεν πρέπει να ξεχνάμε ότι η εικόνα θα έχει κατάληξη ".png" και διαστάσεις 256x256 pixels. Στη συνέχεια χρειάζεται να γίνουν κάποιοι πρόχειροι υπολογισμοί για να βρούμε τις επιτρεπτές RGB τιμές που θα χρησιμοποιήσουμε στα εργαλεία του προγράμματος.

Έστω λοιπόν ότι έχουμε αποφασίσει να παράγουμε εικόνα με βάθος οκτώ, δηλαδή εικόνα που θα περιέχει μόνο αποχρώσεις του γκρι (greyscale-color depth=8 bit). Αποφασίζουμε και για το πλήθος των επιπέδων (στο σχήμα 5.27.α διακρίνουμε πέντε επίπεδα) και καταλήγουμε στον αριθμό έξι. Σε κάθε επίπεδο τώρα πρέπει να δώσουμε και από μία απόχρωση. Με βάθος οκτώ συνολικά έχουμε 256 (=2^8) αποχρώσεις (σε αποχρώσεις του γκρι ισχύει R=G=B). Το επίπεδο με το χαμηλότερο ύψος θα έχει χρώμα μαύρο. Κάθε επίπεδο θα διαφέρει από το άλλο με βάση τη σχέση:

πλήθος χρωμάτων / (σύνολο επιπέδων -1)

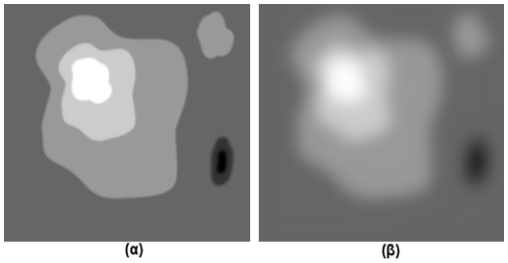
δηλαδή κατά 51 (=256/5) RGB τιμές. Παίρνουμε λοιπόν τον πίνακα 5.1. Σε αυτόν έχουμε αντιστοιχήσει και τα ύψη του κάθε επιπέδου με βάση την ελάχιστη και μέγιστη υψομετρική τιμή που θέλουμε να υπάρχει στη σχεδίασή μας.

[4] Για περιοχές της Αμερικής μπορούμε να αναζητήσουμε δεδομένα μέσω της επίσημης σελίδας της αμερικανικής, γεωλογικής υπηρεσίας (http://www.usgs.gov).

Επίπεδα (elevations)	Χρώμα (r,g,b)	Ύψος (μέτρα)
1	0, 0, 0	0
2	51, 51, 51	20
3	102, 102, 102	40
4	153, 153, 153	60
5	204, 204, 204	80
6	255, 255, 255	100

Πίνακας 5.1: Πίνακας Επιπέδων (Elevation Table)

Με χρήση κάποιου εργαλείου σχεδίασης του προγράμματος (για παράδειγμα το "Pen Tool" στο "Paint Shop Pro") και τα χρώματα της δεύτερης στήλης, σχεδιάζουμε υψομετρικούς χάρτες όπως αυτόν στην παρακάτω εικόνα. Καλό είναι να θολώσουμε την τελική εικόνα (blur) με κάποιο φίλτρο, ώστε να μην έχουμε απότομες εναλλαγές μεταξύ των επιπέδων (για το "Paint Shop Pro" υπάρχει αυτή η δυνατότητα στην καρτέλα "Adjust").



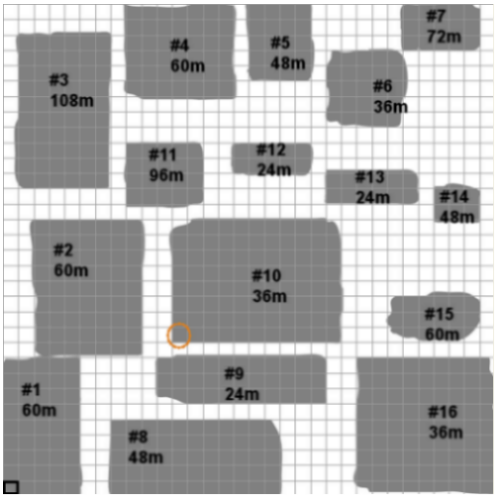
Σχήμα 5.7: (α) Παράδειγμα υψομετρικού χάρτη για τα δεδομένα του πίνακα 5.1 - (β) Ο ίδιος υψομετρικός χάρτης με προσθήκη θορύβου (θολωμένη εικόνα)

5.6.4 Σχεδίαση υψομετρικού χάρτη με την εφαρμογή "TerrainGenerator"

Προγράμματα όπως το "Paint Shop Pro" συνίστανται για απλούς υψομετρικούς χάρτες. Τί γίνεται όμως αν ο προς δημιουργία υψομετρικός χάρτης περιλαμβάνει μεγάλο αριθμό ανεξάρτητων λόφων και μάλιστα καθένας από αυτούς έχει δέκα, δεκατρία ή είκοσι επίπεδα; Αναπτύχθηκε λοιπόν μία απλή εφαρμογή η οποία αυτοματοποιεί τη διαδικασία παραγωγής υψομετρικών εικόνων. Πρόκειται για το **"TerrainGenerator"**, ένα πρόγραμμα υλοποιημένο με τη βοήθεια του πακέτου "Matlab" της "The MathWorks" (http://www.mathworks.com/). Για περισσότερα σχετικά με το "Matlab" αλλά

και τον κώδικα της εφαρμογής μπορείτε να ανατρέξετε στο παράρτημα. Η παρούσα υπο-ενότητα είναι αφιερωμένη στη λειτουργεία του "TerrainGenerator".

Η βασική ιδέα πάνω στην οποία στηρίχθηκε η εφαρμογή "TerrainGenerator" είναι ότι ξεκινάμε με μια επίπεδη περιοχή και "φυτεύουμε" λόφους σε διάφορες θέσεις. Κάθε λόφος έχει ένα σημείο εκκίνησης και καλύπτει μια επιφάνεια καθώς εκτείνεται προς τα ανατολικά και προς βόρεια από το σημείο. Προγραμματιστικά χρησιμοποιήθηκε ένας 32x32 πίνακας που παίζει το ρόλο του καμβά για την εφαρμογή. Θυμηθείτε ότι οι υψομετρικοί χάρτες είναι διαστάσεων 256x256 pixels. Άρα κάθε κυψέλη (τετράγωνο) του 32x32 πίνακα αντιστοιχεί σε 8 pixels στον τελικό υψομετρικό χάρτη. Και τι περιέχει κάθε κυψέλη; Μία τιμή ύψους για τα συγκεκριμένα pixels!



Σχήμα 5.8: Πρόχειρο σχέδιο με τις θέσεις και τα ύψη των λόφων (γκρι περιοχές)

Για παράδειγμα, το σχήμα 5.29 είναι ένα πρόχειρο σχεδιάγραμμα για το πως πρέπει να είναι η βάση του εδάφους του παιχνιδιού.[5] Στηριζόμενοι σε αυτό θα παράγουμε τον υψομετρικό χάρτη με το "TerrainGenerator".

Οι δεκαέξι (16) αριθμημένες, γκρι, περιοχές αναπαριστούν τις θέσεις για τους αντίστοιχους δεκαέξι λόφους. Αναγράφεται επίσης το μέγιστο ύψος για τον καθένα. Αν παρατηρήσουμε προσεχτικά το σχήμα θα δούμε ένα πλέγμα 32x32 τετραγώνων. Μας βοηθάει να εντοπίζουμε τη θέση κάθε οντότητας στο χώρο. Γιατί όμως 32x32; Μα φυσικά για να συμπίπτει με τον 32x32 πίνακα του "TerrainGenerator". Ας πάρουμε την αριθμημένη περιοχή δέκα (#10) και ας δοκιμάσουμε να δούμε που βρίσκεται το κάτω αριστερό άκρο της (πορτοκαλί κύκλος στο σχήμα). Οριζόντια βρίσκεται στο δωδέκατο τετράγωνο και κατακόρυφα στο ενδέκατο (ξεκινάμε και μετράμε με αφετηρία το πρώτο, κάτω, αριστερά τετράγωνο του πλέγματος - το έντονα μαύρο στο σχήμα). Και πόσο χώρο καλύπτει; Προς τα δεξιά (ή ανατολικά) έντεκα τετράγωνα και προς τα πάνω (ή βόρεια) οκτώ. Ακολουθούμε την ίδια διαδικασία για όλες τις γκρι περιοχές και έχουμε τον παρακάτω πίνακα:

[5]Με τον όρο βάση, όπως εξηγήσαμε νωρίτερα, αναφερόμαστε στον 2048x2048, σε τετραγωνικά μέτρα, χώρο που θα επαναλαμβάνεται από την μηχανή και θα δίνει την αίσθηση ενός χωρίς όρια εδάφους.

#Λόφος	Αρχικό σημείο x	Αρχικό σημείο y	Ανατολικό όριο	Βόρειο όριο	Μέγιστο ύψος (μέτρα)
1	1	1	5	9	60
2	3	10	7	9	60
3	2	21	6	10	108
4	9	27	7	6	60
5	17	28	4	5	48
6	22	25	5	5	36
7	27	30	5	3	72
8	8	1	11	5	48
9	11	7	11	3	24
10	12	11	11	8	36
11	9	20	5	4	96
12	16	22	5	2	24
13	22	20	6	2	24
14	29	19	3	2	48
15	26	11	6	3	60
16	24	1	9	9	36

Πίνακας 5.2: Θέση λόφων για το σχήμα 5.29 μετρώντας σε τετράγωνα (ως τεράγωνο αρχής θεωρούμε το πρώτο, κάτω, αριστερά τετράγωνο του πλέγματος)

Αυτά είναι τα δεδομένα με τα οποία το πρόγραμμα παράγει τον υψομετρικό χάρτη. Αρχικά, όταν εκκινήσουμε την εφαρμογή, μας ζητείται το πλήθος των λόφων (Hills#) που θέλουμε να παράγουμε και το πλήθος των επιπέδων (Elevations#). Ο αριθμός των επιπέδων θα είναι ίδιος για όλες τις οντότητες. Στη συνέχεια εισάγουμε τις τιμές του πίνακα 5.2 στη φόρμα που εμφανίζεται. Κάθε γραμμή δέχεται δεδομένα για ένα λόφο, με την ίδια ακριβώς σειρά που βρίσκονται και στον πίνακα. Το επόμενο βήμα είναι να εντοπιστεί η θέση του κάθε λόφου στον 32x32 πίνακα του "TerrainGenerator", που έστω ότι τον ονομάζουμε Z. Εμπειρικά ο εν λόγω πίνακας είναι το πλέγμα που υπάρχει στο σχήμα 5.29. Για παράδειγμα η καταχώρηση στη δωδέκατη γραμμή και στην ενδέκατη στήλη του Z ανταποκρίνεται στην κυκλωμένη περιοχή του σχεδιαγράμματος 5.29. Με βάση το μέγιστο ύψος και τον αριθμό επιπέδων, καθώς συνεχίζεται η διαδικασία, δημιουργείται ένας πίνακας επιπέδων με το χαρακτηριστικό:

διαφορά ύψους μεταξύ επιπέδων=μέγιστο ύψος / (αριθμό επιπέδων-1)

Έτσι αν έχουμε δώσει ως μέγιστο ύψος τα 10 μέτρα και αριθμό επιπέδων 6 τότε:

Από τη σχέση φαίνεται ότι το πρόγραμμα θα λειτουργήσει βέλτιστα αν τα μέγιστα ύψη είναι ακέραια πολλαπλάσια του αριθμού επιπέδων, αφού πρώτα έχει μειωθεί κατά ένα. Η μία αυτή τιμή που αφαιρέθηκε, αντιστοιχεί σε επίπεδο με ύψος μηδέν. Εξαρχής το πρόγραμμα αρχικοποιεί το χάρτη με το μηδενικό επίπεδο άρα δεν λογαριάζεται στους υπολογισμούς.

Το "TerrainGenerator" τώρα, σε κάθε κυψέλη του λόφου που επεξεργάζεται, καταχωρεί μία τυχαία επιλεγμένη, από τον πίνακα επιπέδων, τιμή ύψους. Όταν πλέον έχουν

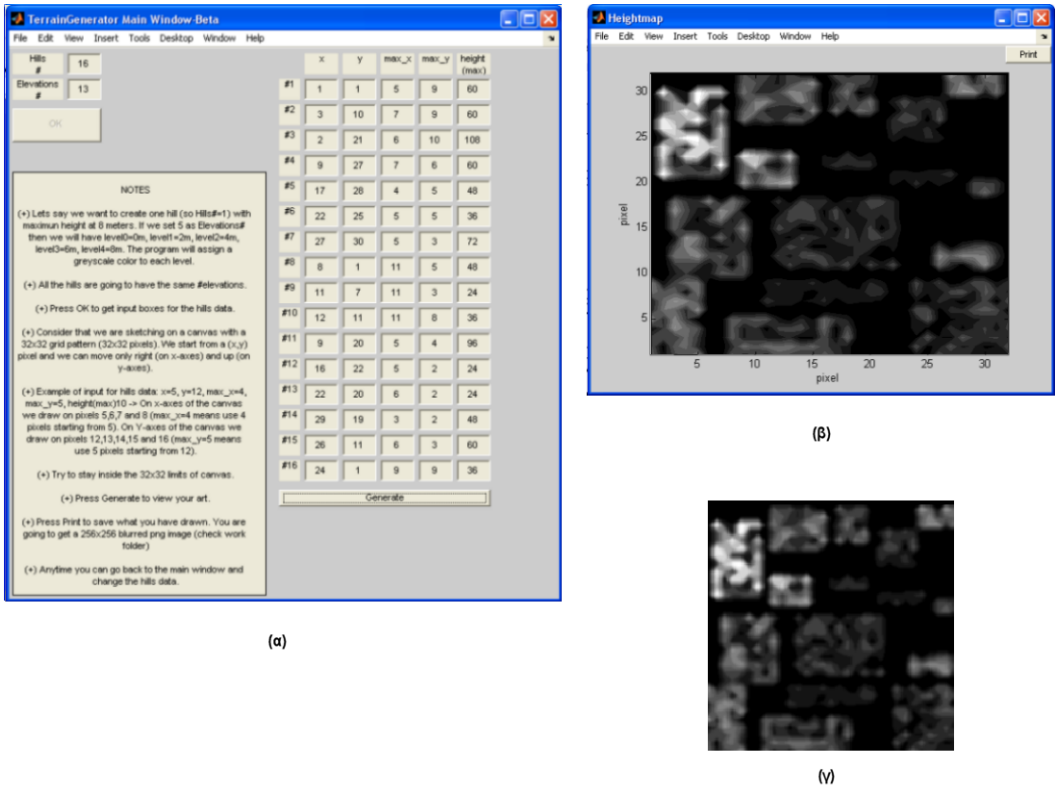
Επίπεδο #	Ύψος (m)
1	0
2	2
3	4
4	6
5	8
6	10

Πίνακας 5.3: Πίνακας επιπέδων για 6 επίπεδα με μέγιστο ύψος στα 10 μέτρα

γίνει όλες οι καταχωρήσεις, πατάμε "Generate" στη διεπαφή και εμφανίζεται στην οθόνη ο υψομετρικός χάρτης. Έχουμε τη δυνατότητα να αλλάξουμε τα δεδομένα των λόφων (όχι όμως και το πλήθος των λόφων ή των επιπέδων) ή απλώς να πατήσουμε "Generate" και πάλι. Στην τελευταία αυτή περίπτωση προκύπτει μια καινούργια εικόνα μιας και όπως αναφέραμε οι τιμές, στις κυψέλες, καταχωρούνται τυχαία.

Αν είμαστε ικανοποιημένοι με το αποτέλεσμα τότε με το κουμπί "Print" (που βρίσκεται πάνω δεξιά στο γράφημα που έχει εμφανιστεί) αποθηκεύουμε στο δίσκο τον χάρτη. Ο υψομετρικός χάρτης θα σωθεί ως εικόνα τύπου ".png", διαστάσεων 256x256, αφού πρώτα έχει θολωθεί (blurred) ώστε να έχουν ομαλοποιηθεί οι εναλλαγές από το ένα επίπεδο στο άλλο.

Για να χρησιμοποιήσει κανείς το εργαλείο "TerrainGenerator" χρειάζεται να "τρέξει", μέσα από το περιβάλλον του "Matlab" το "terrainGeneratorMain.m" αρχείο. Αν δεν διαθέτει το συγκεκριμένο μαθηματικό πακέτο, το "TerrainGenerator" υπάρχει και σαν αυτόνομο, εκτελέσιμο αρχείο ("terrainGeneratorMain.exe"). Στην τελευταία αυτή περίπτωση το γραφικό περιβάλλον θα παρουσιάζει κάποιες μικρές διαφορές σε σχέση με τις εικόνες που παρουσιάζονται παρακάτω.



Σχήμα 5.9: (α) Το γραφικό περιβάλλον του TerrainGenerator - (β) Ο καμβάς του TerrainGenerator εμφανίζει τον υψομετρικό χάρτη για τα δεδομένα του πίνακα 5.2 και για 13 επίπεδα - (γ) Η 256x256 ".png" εικόνα που σώζουμε στο δίσκο

5.6.5 Παραγωγή των ".ter" αρχείων

Το όνομα που επιλέχθηκε στην εφαρμογή ("TerrainGenerator") είναι λίγο ατυχές. Με τη βοήθειά του παράγουμε υψομετρικούς χάρτες, δηλαδή αρχεία γραφικών με κατάληξη ".png". Το αρχείο εδάφους (terrain file), όπου έδαφος είναι ο τρισδιάστατος χώρος πάνω στον οποίο κινούνται τα τρισδιάστατα μοντέλα του παιχνιδιού, έχει κατάληξη ".ter".

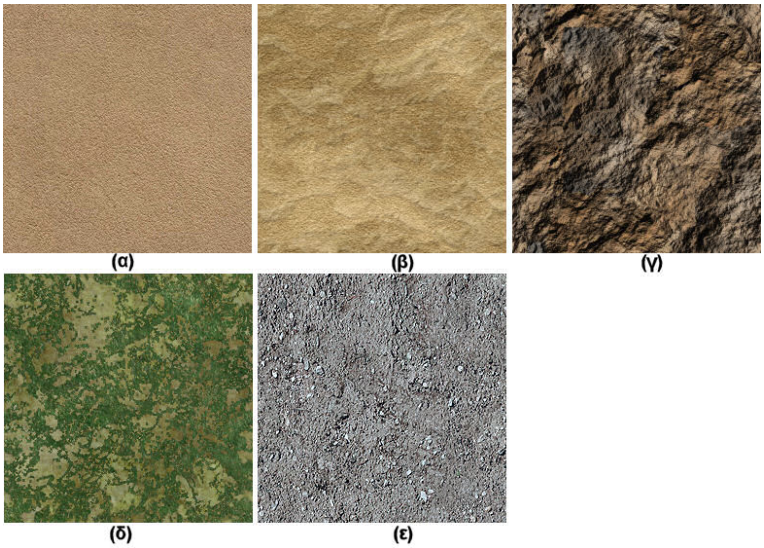
Για να μετατρέψουμε τον υψομετρικό χάρτη σε τέτοια μορφή αρχείου χρησιμοποιούμε το εργαλείο "Terrain Terraform Editor", που έρχεται μαζί με τη μηχανή "Torque". Η πρώτη επιλογή (πάνω και δεξία στο γραφικό περιβάλλον του εργαλείου) αναγράφει "Min Terrain Height". Προσοχή γιατί ίσως αυτό να ξεγελάει! Πρόκειται για τη θέση όπου εισάγουμε το μέγιστο ύψος που θα έχει το έδαφός μας. Σύμφωνα με τον πίνακα 5.2 πρέπει να δώσουμε τον αριθμό εκατό οκτώ (108). Αμέσως μετά έχουμε το εύρος ύψους ("Height Range") όπου είναι η διαφορά μεταξύ του ελάχιστου και μέγιστου ύψους του εδάφους. Το ελάχιστο ύψος, για τα δεδομένα που χρησιμοποιούμε, είναι το μηδέν όποτε πάλι εισάγουμε ως τιμή τον αριθμό εκατό οκτώ (108). Το επόμενο βήμα είναι το "φόρτωμα" του υψομετρικού χάρτη. Πατώντας το κουμπί "Operation" και επιλέγοντας "Bitmap", διαλέγουμε τον χάρτη από τη θέση όπου τον έχουμε αποθηκεύσει. Θα εμφανιστεί στο μικρότερο παράθυρο κάτω και αριστερά στο εργαλείο. Με μια πρώ-

τη ματιά μοιάζει διαφορετικός. Αυτό συμβαίνει γιατί το "Terrain Terraform Editor" χρησιμοποιεί έγχρωμη κλίμακα αναπαράστασης και όχι μαυρόασπρη. Έτσι οι επίπεδες περιοχές αναπαρίστανται με λευκό χρώμα και οι λόφοι με διακυμάνσεις του πράσινου.

Από το βασικό τώρα μενού επιλέγουμε File -> Save Mission As... και σώζουμε την αποστολή με ένα μοναδικό όνομα. Παράγεται λοιπόν το ".ter" αρχείο αλλά και ένα αρχείο με κατάληξη ".mis". Το τελευταίο περιέχει στη γλώσσα σεναρίων της "Torque" (Torque script language) πληροφορίες για την όλη σκηνή του παιχνιδιού (για παράδειγμα ένταση ήλιου, θέση ουρανού, θέσεις αντικειμένων στη σκηνή). Προς το παρόν δεν θα ασχοληθούμε με αυτό μιας και θα το αναλύσουμε αλλά και θα το τροποποιήσουμε στην πορεία της εργασίας.

5.6.6 Εφαρμογή υφών (textures) για το έδαφος

Όπως και στην περίπτωση μοντελοποίησης του χαρακτήρα, χρησιμοποιούμε υφές για να "ντύσουμε" το έδαφος. Για το πως δημιουργούμε τα γραφικά αρχεία υφών δεν χρειάζεται να αναφερθεί κάτι παραπάνω. Σημειώνεται ωστόσο ότι πρέπει να είναι εικόνες διαστάσεων 256x256.

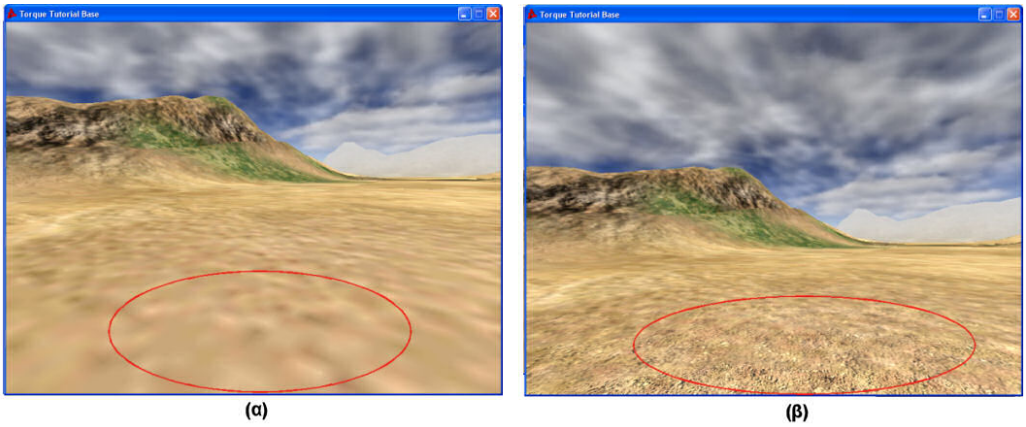


Σχήμα 5.10: Οι υφές για το έδαφος. Η τελευταία (ε) χρησιμοποιείται ως υφή λεπτομέρειας

Με το εργαλείο "Terrain Texture Painter" μας δίνεται η δυνατότητα να δουλέψουμε με έως και έξι αρχεία υφών. Το πρώτο από αυτά εφαρμόζεται σε όλο το έδαφος και τα υπόλοιπα σε περιοχές που εμείς θέλουμε, επικαλύπτοντας την αρχική υφή. Και πάλι εργαζόμαστε μόνο πάνω στη βάση, δηλαδή στην επιφάνεια η οποία επαναλαμβάνεται δίνοντας την αίσθηση του χωρίς όρια εδάφους. Για να είμαστε σίγουροι ότι επεξεργαζόμαστε τη σωστή επιφάνεια υπάρχει το εργαλείο "Mission Area Editor" που καθορίζει τα όρια της εν λόγω περιοχής.

Το τελευταίο που αξίζει να σημειωθεί για τις υφές είναι η χρήση μίας και μόνο εικόνας ως "υφής λεπτομέρειας" (detail texture). Από απόσταση το αποτέλεσμα, μετά

την εφαρμογή υφών, μπορεί να είναι αρκετά ικανοποιητικό όχι όμως και από κοντά. Παρατηρείστε τα παρακάτω στιγμιότυπα όπου στο δεύτερο γίνεται χρήση της υφής λεπτομέρειας.



Σχήμα 5.11: Εφαρμογή υφής λεπτομέρειας (α) Χωρίς - (β) Με

5.7 Ήλιος

Το αντικείμενο που αναπαριστά τον ήλιο δεν θα είναι ορατό στη σκηνή του παιχνιδιού. Σκοπός του είναι να καθορίσει τον τρόπο που φωτίζονται τα αντικείμενα που απαρτίζουν το περιβάλλον της εφαρμογής. Μέσω του "World Editor" ή προγραμματιστικά (βλ. αρχείο **"/control/data/missions/mission1.mis"**) μπορεί να καθοριστεί η θέση του αντικειμένου που αναπαριστά τον ήλιο, η κλήση του ως προς κάθε άξονα και η κλίμακα του καθώς και το χρώμα, η ένταση και η κατεύθυνση του φωτός που εκπέμπει.

Σύμφωνα με τη θέση του ήλιου δημιουργούνται οι σκιές των αντικειμένων. Επίσης οι πλευρές των αντικειμένων που δεν βρίσκονται προς την πλευρά του ήλιου είναι πιο σκουρόχρωμες. Το χρώμα και η ένταση του φωτός που επηρεάζει τα αντικείμενα και δημιουργεί τις σκιές τους καθορίζεται από την τιμή της παραμέτρου "color". Η τιμή της παραμέτρου "ambient" καθορίζει το χρώμα και την ένταση του φωτός που υπάρχει διάχυτο στο περιβάλλον της εφαρμογής.

Listing 5.1: Τμήμα του κώδικα από το αρχείο της αποστολής

```
//==================================
//   ./control/data/missions/mission1.mis
//
//   The mission
//==================================
...
...
new Sun() {
   canSaveDynamicFields = "1";
   azimuth = "0";
   elevation = "35";
```

```
   color = "0.535 0.535 0.535 1";
   ambient = "0.295 0.295 0.295 1";
   CastsShadows = "1";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   locked = "true";
   direction = "0.57735 0.57735 -0.57735";
   position = "0 0 0";
};

new TerrainBlock(Terrain) {
...
...
};

new Sky(Sky) {
   canSaveDynamicFields = "1";
   position = "-1088 -928 0";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   materialList = "~/data/skies/sky_day.dml";
   cloudHeightPer[0] = "0.349971";
   cloudHeightPer[1] = "0.25";
   cloudHeightPer[2] = "0.199973";
   cloudSpeed1 = "0.0005";
   cloudSpeed2 = "0.0002";
   cloudSpeed3 = "0.0003";
   visibleDistance = "800";
   fogDistance = "700";
   fogColor = "0.82 0.828 0.844 1";
   fogStorm1 = "0";
   fogStorm2 = "0";
   fogStorm3 = "0";
   fogVolume1 = "500 0 100";
   fogVolume2 = "0 0 0";
   fogVolume3 = "0 0 0";
   fogVolumeColor1 = "128 128 128 -2.22768e+038";
   fogVolumeColor2 = "128 128 128 0";
   fogVolumeColor3 = "128 128 128 -1.70699e+038";
   windVelocity = "0.0 1.25 0";
   windEffectPrecipitation = "1";
   SkySolidColor = "0.547 0.641 0.789 0";
   useSkyTextures = "1";
   renderBottomTexture = "0";
   noRenderBans = "0";
   locked = "true";
};
...
...
```

5.8 Ουρανός

Πρόκειται για ένα παραλληλεπίπεδο μεγάλων διαστάσεων με διαφορετικές υφές να καλύπτουν τις έξι έδρες του. Οι έδρες διατηρούν πάντα σταθερή απόσταση από τον επισκέπτη και επομένως ο επισκέπτης δεν μπορεί ποτέ να φτάσει τα όρια του παραλληλεπιπέδου. Με αυτόν τον τρόπο δημιουργείται η ψευδαίσθηση ότι ο ουρανός βρίσκεται σε άπειρη απόσταση. Η θέση του, η κλήση του ως προς κάθε άξονα και η κλίμακα του καθορίζονται από τις τιμές των παραμέτρων "position", "rotation" και "scale" αντίστοιχα. Το αρχείο "./control/data/skies/sky_day.dml" που αποτελεί τιμή της παραμέτρου "materialList" περιέχει τα ονόματα των εικόνων που θα καλύψουν κάθε έδρα του παραλληλεπιπέδου και των εικόνων που θα χρησιμοποιηθούν για την παρουσίαση των σύννεφων. Η υφή της κάτω πλευράς του παραλληλεπιπέδου δεν είναι ορατή, όπως υποδηλώνει η τιμή μηδέν (0) της παραμέτρου "renderBottomTexture". Υπάρχει δυνατότητα ο ουρανός να μην παρουσιαστεί σύμφωνα με το μοντέλο του παραλληλεπιπέδου, αν η τιμή της μεταβλητής "useSkyTextures" μετατραπεί από 1 σε 0. Στην περίπτωση αυτή ο ουρανός καλύπτεται από το χρώμα που ορίζει η παράμετρος "SkySolidColor".

Καθορίζονται τρία επίπεδα σύννεφων σε ύψη που ορίζονται από τις τιμές των παραμέτρων "cloudHeightPer[0]", "cloudHeightPer[1]" και "cloudHeightPer[2]". Σύμφωνα με την παρατήρηση ότι η ταχύτητα των σύννεφων αυξάνεται στα ψηλότερα στρώματα της ατμόσφαιρας, ορίζονται διαφορετικές ταχύτητες σε κάθε στρώμα (παράμετροι "cloudSpeed1", "cloudSpeed2", "cloudSpeed3").

Η ομίχλη μπορεί και αυτή να υλοποιηθεί σε επίπεδα διαφορετικής πυκνότητας. Στον εικονικό κόσμο της εφαρμογής χρησιμοποιείται το ένα μόνο από τα διαθέσιμα επίπεδα. Η έκταση του χώρου που θα καταλαμβάνει η ομίχλη καθορίζεται από την τιμή της παραμέτρου "fogVolume1", ενώ το χρώμα της από την τιμή της παραμέτρου "fogVolumeColor1". Εκτός από τα επίπεδα αυτά υπάρχει και η ομίχλη που εμφανίζεται πάντα σε συγκεκριμένη απόσταση από τον επισκέπτη και καθιστά μη ορατά τα αντικείμενα που βρίσκονται σε μεγαλύτερες αποστάσεις. Η απόσταση αυτή καθορίζεται από την τιμή της "fogDistance" και το χρώμα της από την παράμετρο "fogColor".

Οι παράμετροι "windVelocity" και "windEffectPrecipitation" καθορίζουν τα χαρακτηριστικά του ανέμου και βρίσκουν εφαρμογή σε περίπτωση εισαγωγής φαινομένων καταιγίδας στον εικονικό κόσμο.

Όλα αυτά φαίνονται στο τμήμα κώδικα που παραθέσαμε παραπάνω. Γενικά στο αρχείο **"./control/data/missions/mission1.mis"** περιγράφεται η αποστολή του παιχνιδιού. Όλες οι οντότητες που συνθέτουν τη σκηνή (έδαφος, ουρανός, δέντρα, κτήρια κτλ) δηλώνονται μέσα σε αυτό το αρχείο. Με βάση αυτό η μηχανή (στην πλευρά του εξυπηρετητή) δημιουργεί τον εικονικό κόσμο του παιχνιδιού.

5.9 Κτήρια

Με το πρόγραμμα σχεδίασης "MilkShape 3D" δημιουργήθηκαν οι περισσότερες οντότητες του παιχνιδιού (δένδρα, βράχους, οχήματα). Προσοχή δόθηκε κυρίως στη δημιουργία του "**πλέγματος συγκρούσεων**" **(collision mesh)**. Χωρίς αυτό θα βρισκόμασταν στη δυσάρεστη θέση να μην αλληλεπιδρούμε με τα αντικείμενα της σκηνής. Ο χαρακτήρας θα μπορούσε να περνάει και μέσα από αυτά!

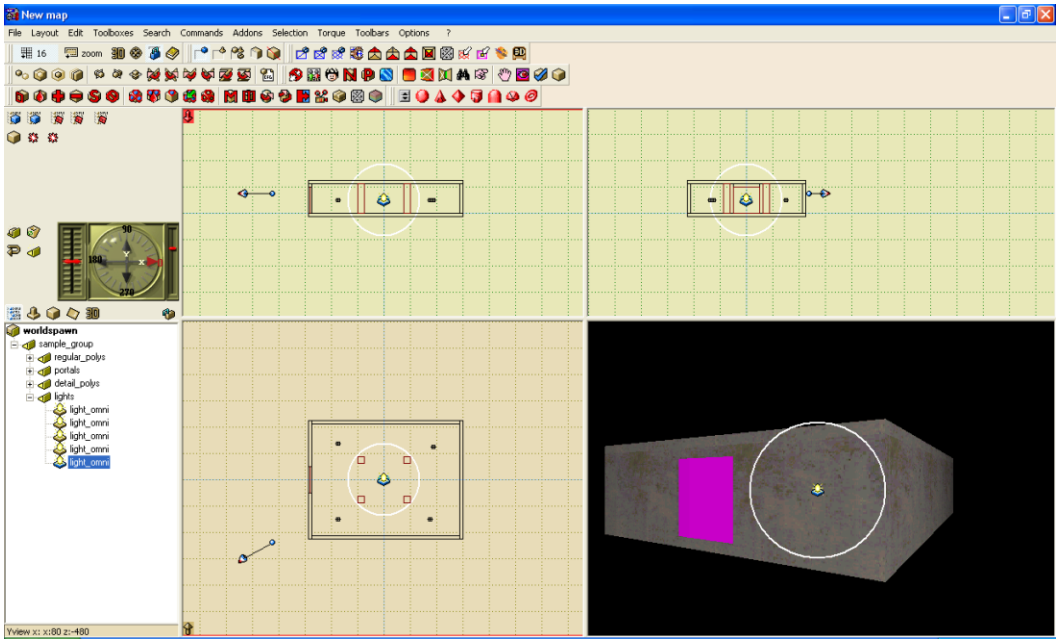
Το "πλέγμα συγκρούσεων" δεν χρειάστηκε κατά τη μοντελοποίηση του χαρακτήρα,

μιας και η μηχανή φροντίζει για αυτό σε αντικείμενα της κλάσης "Player". Για τις υπόλοιπες όμως οντότητες, στη φάση εξαγωγής του μοντέλου δίνεται η δυνατότητα να ορίσουμε το "πλέγμα συγκρούσεων". Μπορούμε όμως να το ορίσουμε και κατά τη σχεδίαση. Για παράδειγμα με το "MilkShape 3D" σχεδιάζουμε το κουτί του σχήματος 5.4. Θέλουμε τώρα έναν νέο κύβο, οπότε από τα εργαλεία επιλέγουμε "Box". Αυτός ο νέος κύβος δημιουργείται έτσι ώστε στο εσωτερικό του να βρεθεί το μοντέλο του κουτιού. **Απαραίτητα πρέπει να τον μετονομάσουμε ως "collision-1"**. Το '-1' είναι το "**επίπεδο λεπτομέρειας**" **(Level Of Detail - LOD)** και στην ουσία θα κάνει αόρατο το πλέγμα στον παίκτη. Το πλέγμα όμως θα βρίσκεται εκεί και ο ήρωας μας θα αντιλαμβάνεται τη σύγκρουση με το κιβώτιο!

Με τον ίδιο τρόπο θα μπορούσαν να μοντελοποιηθούν και οντότητες όπως τα κτήρια. Ωστόσο αυτά έχουν πολλές επιφάνειες και πάνω σε κάποιες από αυτές μπορεί να κινηθεί ο παίκτης. Χρειάζεται λοιπόν επιπλέον προσοχή στη δημιουργία του "πλέγματος συγκρούσεων", κάτι που γίνεται και χρονοβόρο αν το κτήριο έχει πολλές λεπτομέρειες.

Εκτός αυτού, τα μοντέλα ".dts" (με το "MilkShape 3D" εξάγουμε σε αυτή τη μορφή) δεν "αντιλαμβάνονται" τις έννοιες του φωτισμού και των σκιών. Αν θέλουμε να φωτίσουμε κάποιο "DTS αντικείμενο" θα πρέπει να γίνει με δημιουργία ξεχωριστών οντοτήτων.

Η λύση στα παραπάνω έρχεται με την υποστήριξη αντικειμένων τύπου ".dif". Όλες οι οντότητες μέσα στις οποίες θα μπορεί να βρεθεί ο χαρακτήρας θα πρέπει να είναι σε αυτή τη μορφή.



Σχήμα 5.12: Το περιβάλλον εργασίας του QuArK

Υπάρχουν πολλά προγράμματα που εξάγουν μοντέλα με κατάληξη ".dif". Στην παρούσα εργασία χρησιμοποιήθηκε το **"QuArK"**. Το "QuArK" παρέχει μία πληθώρα εργαλείων σχεδιασμού και πολλαπλά παράθυρα για την ταυτόχρονη επισκόπηση διαφορετικών όψεων της σχεδιαζόμενης δομής. Επίσης περιέχει το εργαλείο "material

browser" το οποίο επιτρέπει την επιλογή υφής για κάθε έδρα της δομής που σχεδιάζεται. Επιπλέον το πρόγραμμα επιτρέπει την προσθήκη αντικειμένων τύπου "light entity" που αντιστοιχούν σε φωτεινές πηγές με δυνατότητα ρύθμισης της έντασής τους. Οι δομές που σχεδιάζονται αποθηκεύονται σε αρχεία τύπου ".map", τα οποία στη συνέχεια εξάγονται (exporter "map2dif") και μετατρέπονται στα αρχεία τύπου ".dif".

5.10 Συνοψίζοντας

Στο κεφάλαιο που μόλις ολοκληρώθηκε παρουσιάστηκαν τα τρία βασικά στάδια δημιουργίας ενός τρισδιάστατου αντικειμένου:

- σχεδίαση του πλέγματος του μοντέλου,
- δημιουργία υφών και
- δημιουργία κινούμενων γραφικών.

Αυτό που τελικά δημιουργήθηκε είναι ένας τρισδιάστατος χαρακτήρας για το παιχνίδι. Στη συνέχεια έγινε αναφορά στις οντότητες του εδάφους, του ουρανού, του ήλιου και των κτηρίων, που συνθέτουν τη σκηνή του παιχνιδιού. Αξίζει να κρατήσουμε, σχετικά με τα τελευταία, ότι για τη δημιουργία του εδάφους του παιχνιδιού απαιτείται ένας χάρτης ισοϋψών. Η μηχανή με βάση αυτόν παράγει ένα φαινομενικά χωρίς όρια έδαφος. Η δημιουργία αυτών των υψομετρικών χαρτών είναι πλέον εύκολη διαδικασία μέσω της εφαρμογής "TerrainGenerator" που αναπτύχθηκε.

Κεφάλαιο 6

Υλοποίηση του παιχνιδιού

Το παρόν κεφάλαιο ασχολείται με τον κώδικα της εφαρμογής μας. Κυρίως αναλύονται θέματα που αφορούν τη δικτύωση, τις γραφικές διασυνδέσεις του χρήστη (GUIs), την τεχνική νοημοσύνη. Θα γίνει αναφορά σε έννοιες όπως "σύστημα αποθήκευσης " (inventory system), "σωματιδιακά τεχνάσματα εντυπωσιασμού" και "έκπομποί" (particles, emitters), μηχανές καταστάσεων (FSM).

6.1 Θέματα δικτύωσης

Για άλλη μία φορά να σημειωθεί ότι η εφαρμογή στηρίζεται στην αρχιτεκτονική πελάτη - εξυπηρετητή. Ο πελάτης είναι υπεύθυνος μόνο για τη διασύνδεση του παίκτη με το παιχνίδι και όλα τα υπόλοιπα συμβαίνουν στην πλευρά του εξυπηρετητή.

Η πρώτη ενέργεια του εξυπηρετητή είναι η δημιουργία της σκηνής με βάση το αρχείο της αποστολής. Σε αυτό βρίσκονται οι πληροφορίες που καθορίζουν τα αντικείμενα και τη θέση τους μέσα στη σκηνή. Οτιδήποτε εμφανίζεται μέσα στον εικονικό κόσμο δηλώνεται σε αυτό το αρχείο: αντικείμενα τα οποία θα μπορεί να συλλέξει ο παίκτης (items), παίκτες ελεγχόμενοι από τον υπολογιστή (bots), σημεία εισόδου (spawn points), triggers, οντότητες του φυσικού περιβάλλοντος όπως είδαμε σε προηγούμενο κεφάλαιο κτλ.

Όλα αυτά τα αντικείμενα δημιουργούνται στην πλευρά του εξυπηρετητή και στη συνέχεια μεταφέρονται/αντιγράφονται στον πελάτη. Έχει επικρατήσει αυτά τα αντικείμενα στον πελάτη να αποκαλούνται **"ghosts"**.

6.1.1 Δημιουργία του εξυπηρετητή

Το παιχνίδι είναι ενός και μόνο παίκτη (single-player). Έτσι ο εξυπηρετητής δημιουργείται όταν ο χρήστης πατήσει "Play" στο βασικό μενού.

Καλείται η συνάρτηση *createServer()* με ορίσματα το είδος του εξυπηρετητή και το όνομα της αποστολής που πρέπει να φορτωθεί. Στο πρώτο αυτό στάδιο δημιουργίας δεν κάνουμε τίποτα παραπάνω από το να φορτώσουμε στη μνήμη τα τμήματα κώδικα που βρίσκονται μέσα στους υποφακέλους του "./control/server/". Κατά κύριο λόγο πρόκειται για τον ορισμό των ιδιοτήτων (datablocks) και των συναρτήσεων των οντοτήτων του παιχνιδιού.

Ακολουθεί η δεύτερη φάση δημιουργίας του εξυπηρετητή. Στο τέλος αυτής, το παιχνίδι θα έχει φορτωθεί στον εξυπηρετητή και θα είναι έτοιμο να "κατέβει" στον πελάτη που θα συνδεθεί μαζί του. Οι σημαντικότερες ενέργειες εδώ είναι:

- δημιουργία του κειμένου που περιγράφει την αποστολή (*buildLoadInfo()*). Οι απαραίτητες πληροφορίες συλλέγονται από το αντικείμενο "MissionInfo" που έχει δημιουργηθεί μέσα στο αρχείο της αποστολής (./control/data/missions/mission1.mis),

- υπολογισμός του **Cyclic Redundancy Check**. Το CRC είναι γνωστό από τα δίκτυα υπολογιστών, όπου χρησιμοποιείται σαν τρόπος ανίχνευσης λαθών στα πακέτα που διακινούνται στο δίκτυο. Το CRC ενός πακέτου που μεταδίδεται υπολογίζεται από τον αποστολέα και προστίθεται στο πακέτο. Ο αποδέκτης επανα-υπολογίζει το CRC στο πακέτο που πήρε και το συγκρίνει με την τιμή που είναι αποθηκευμένη στο πακέτο. Αν το πακέτο δεν άλλαξε/αλλοιώθηκε στη διαδρομή, τα CRC θα συμπίπτουν. Εδώ χρησιμοποιείται ως τρόπος ανίχνευσης αλλαγών στην αποστολή, ώστε να ξέρουμε αν πρέπει να φωτιστεί ξανά η αποστολή (*GetFileCRC()*),

- δημιουργία της αποστολή και των αντικειμένων (*exec(missionFile)*),

- φόρτωμα των μονοπατιών (paths) που έχουν οριστεί σε εσωτερικούς χώρους (*PathOnMissionLoadDone()*),

- μεταφορά της αποστολής στους συνδεδεμένους πελάτες (*onMissionLoaded()*).

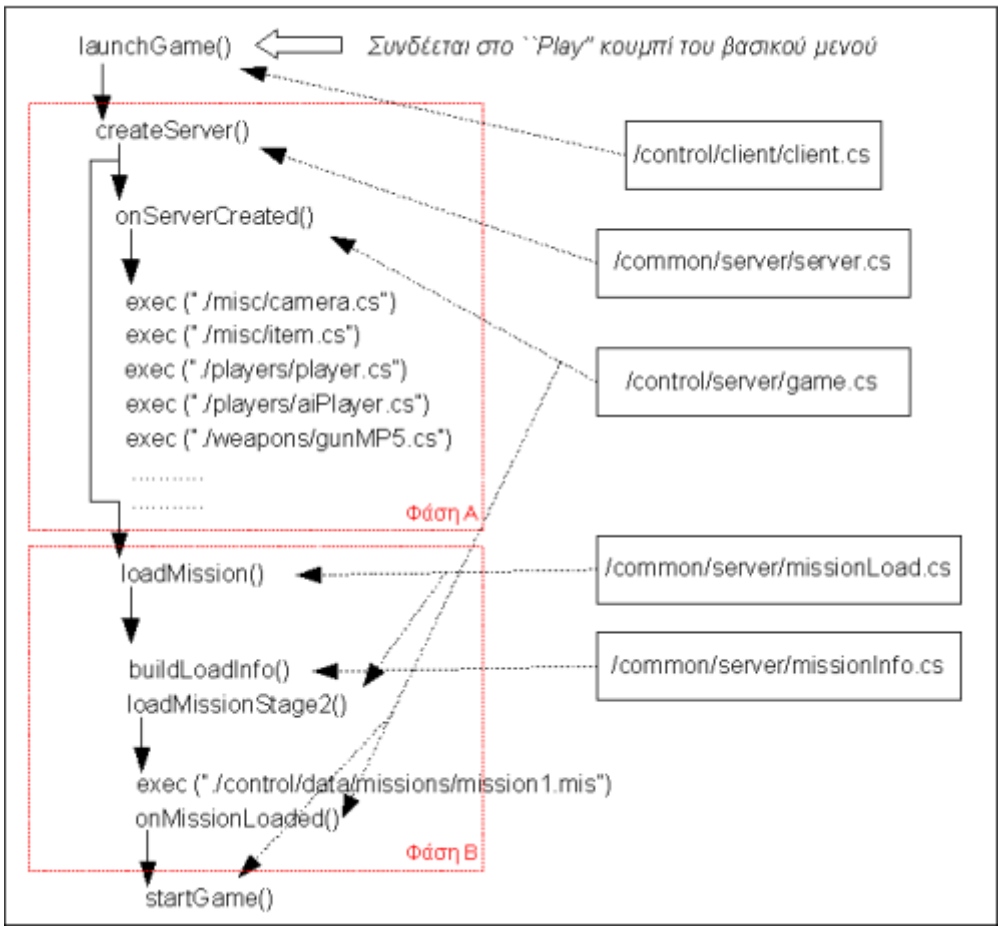
Επειδή η εφαρμογή περιέχει πλήθος σεναρίων και σε κάθε σενάριο υπάρχουν συναρτήσεις που καλούν άλλες σε διαφορετικό αρχείο, είναι δύσκολο να παρακολουθήσει κανείς τις ενέργειες που γίνονται. Για το λόγο αυτό παρατίθεται το παρακάτω διάγραμμα που βοηθάει να εντοπιστούν οι συναρτήσεις που χρησιμοποιούνται για τη δημιουργία του εξυπηρετητή και το αρχείο στο οποίο ορίζονται. Στο διάγραμμα αυτό τα αρχεία μέσα σε πλαίσιο είναι αυτά όπου υπάρχουν οι δηλώσεις των συναρτήσεων και μεθόδων, ενώ τα έντονα, μαύρα βέλη δείχνουν τη σειρά με την οποία γίνονται οι κλήσεις.

6.1.2 Σύνδεση πελάτη-εξυπηρετητή

Το πρώτο πράγμα που πρέπει να κάνει ένας πελάτης για να εισέλθει στον κόσμο και να μπορέσει να αλληλεπιδράσει με οτιδήποτε υπάρχει σε αυτόν είναι να συνδεθεί με τον εξυπηρετητή. Βασική κλάση εδώ είναι η **"GameConnection"**, που ορίζει μεθόδους και callbacks για την εγκατάσταση και το χειρισμό μιας τέτοιας σύνδεσης. Δημιουργείται λοιπόν στην πλευρά του πελάτη ένα αντικείμενο της κλάσης "GameConnection". Οι ενέργειες που γίνονται είναι:

- δημιουργία του αντικειμένου της κλάσης "GameConnection" για τη σύνδεση στον εξυπηρετητή (*new GameConnection(ServerConnection)* - βλ. αρχείο ./control/client/client.cs),

- δημιουργία τοπικής σύνδεσης (*ServerConnection.connectLocal()* - βλ. αρχείο ./control/client/client.cs),



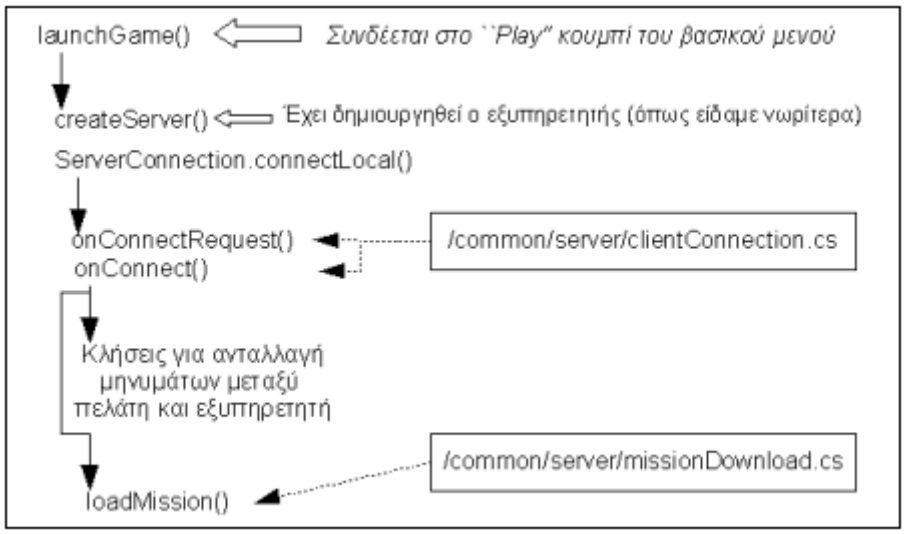
Σχήμα 6.1: Διάγραμμα βασικότερων ενεργειών κατά τη δημιουργία του εξυπηρετητή

- προσπάθεια σύνδεσης στον εξυπηρετητή (*onConnectRequest()* - ορίζεται στο αρχείο ./common/server/clientConnection.cs),
- επιτυχής σύνδεση (*onConnect()* - ορίζεται στο αρχείο ./common/server/clientConnection.cs).

Στην πλευρά του εξυπηρετητή, η μηχανή διατηρεί εσωτερικά μία λίστα με όλους τους συνδεδεμένους πελάτες. Η λίστα αυτή είναι προσβάσιμη σε επίπεδο script μέσω του αντικειμένου **"ClientGroup"** (που είναι αντικείμενο της κλάσης "SimGroup" ή "SimSet"). Το "ClientGroup" υπάρχει χωρίς να χρειάζεται να το δημιουργήσουμε. Όταν ο πελάτης συνδεθεί στον εξυπηρετητή, η μηχανή προσθέτει αυτόματα το αντικείμενο της κλάσης "GameConnection" στο "ClientGroup" του εξυπηρετητή.

6.1.3 Κατέβασμα της αποστολής στον πελάτη

Από τη στιγμή που έχει εγκατασταθεί η σύνδεση μεταξύ πελάτη-εξυπηρετητή, ο ένας μπορεί να επικοινωνήσει απ΄ ευθείας με τον άλλο με το μηχανισμό ανταλλαγής μηνυμάτων



Σχήμα 6.2: Διάγραμμα βασικότερων ενεργειών για τη σύνδεση του πελάτη με τον εξυπηρετητή

της Torque. Ο μηχανισμός αυτός βασίζεται στις συναρτήσεις κονσόλας **"commandToServer()"** και **"commandToClient()"**, και λειτουργεί ως εξής:

- **Μήνυμα από πελάτη σε εξυπηρετητή:** Στην πλευρά του πελάτη, στέλνουμε το μήνυμα στον μοναδικό εξυπηρετητή με τον οποίο είμαστε συνδεδεμένοι.

```
function commandToServer('FunctionName',arg1,arg2...) {...};
```

Στην πλευρά του εξυπηρετητή, η λήψη του μηνύματος μεταφράζεται στην κλήση μιας συνάρτησης με κατάλληλα διαμορφωμένο όνομα.

```
function serverCmdFunctionName(arg1,arg2...) {...};
```

- **Μήνυμα από εξυπηρετητή σε πελάτη:** Στην πλευρά του εξυπηρετητή επιλέγουμε έναν από τους συνδεδεμένους πελάτες και του στέλνουμε το μήνυμα.

```
function commandToClient(gameConnectionOfClient,'FunctionName',arg1,arg2...)
{...};
```

Στην πλευρά του πελάτη, η λήψη του μηνύματος μεταφράζεται στην κλήση μιας συνάρτησης με κατάλληλα διαμορφωμένο όνομα.

```
function clientCmdFunctionName(arg1,arg2...) {...};
```

Προσέξτε ότι το ενδεικτικό "FunctionName" βρίσκεται μέσα σε "μονά αυτάκια" (tagged string).

Ο πελάτης είναι πλέον έτοιμος να κατεβάσει την αποστολή. Η διαδικασία αυτή γίνεται σε τρεις φάσεις. Σε κάθε μία απαιτείται ο συγχρονισμός μεταξύ πελάτη-εξυπηρετητή ώστε η κάθε φάση να ξεκινά όταν είναι και δύο έτοιμοι και να μην προχωράμε στην επόμενη φάση αν δεν ολοκληρώσουν και οι δύο πλευρές επιτυχώς την τρέχουσα φάση. Ο συγχρονισμός επιτυγχάνεται με το μηχανισμό μηνυμάτων που περιγράψαμε.

Πρώτη φάση

Στην πρώτη φάση αποστέλλονται στον πελάτη τα datablocks των αντικειμένων της αποστολής:

- αποστέλλεται το CRC της αποστολής έτσι ώστε να μπορεί ο πελάτης να ελέγξει αν θα τη λάβει σωστά (*setMissionCRC()*),
- ξεκίνα η αποστολή των datablocks, που έχουν φορτωθεί στον εξυπηρετητή, στον πελάτη. Έτσι, ο πελάτης θα έχει όλες τις πληροφορίες που χρειάζεται για αυτά και θα μπορέσει να χειριστεί τα αντικείμενα κλάσεων που τα χρησιμοποιούν και τα οποία θα σταλούν στην επόμενη φάση. Ο πελάτης δεν έχει πρόσβαση στον κώδικα των datablocks που του έστειλε ο εξυπηρετητής, οπότε δεν μπορεί να τα αλλοιώσει (*transmitDataBlocks()*),
- ολοκληρώνεται επιτυχώς η αποστολή (*onDatablocksDone()*).

Δεύτερη φάση

Στη δεύτερη φάση αντιγράφονται στον πελάτη τα αντικείμενα που έχουν δημιουργηθεί στον εξυπηρετητή (ghosting):

- ο πελάτης καταστρέφει ό,τι ενδεχομένως έχει απομείνει από προηγούμενες αποστολές (*purgeResources()*),
- μεταφέρεται η πληροφορία για τα μονοπάτια (paths) που έχουν οριστεί (*transmitPaths()*),

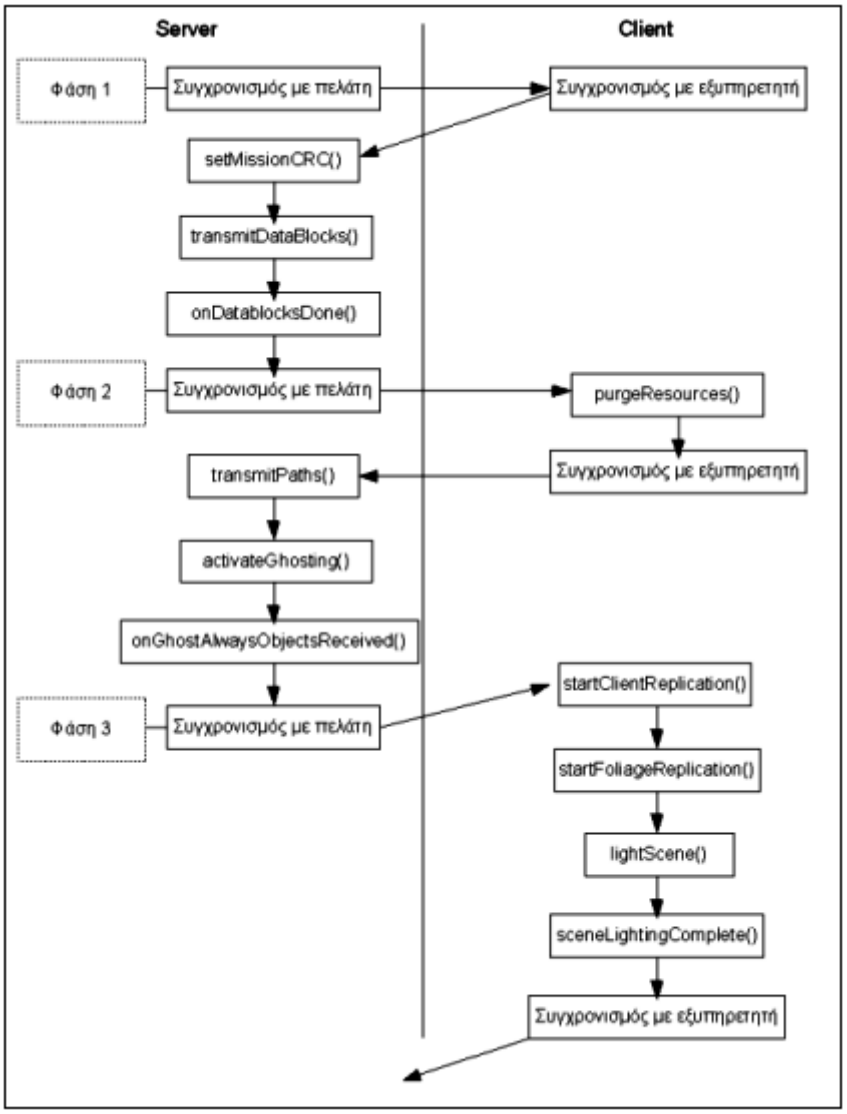
- ξεκίνα η αποστολή των αντικειμένων, που έχουν φορτωθεί στον εξυπηρετητή, στον πελάτη. Όπως και πριν ο πελάτης δεν έχει πρόσβαση στον κώδικά τους και δεν μπορεί να τα αλλοιώσει (*activateGhosting()*),
- ολοκληρώνεται επιτυχώς η αποστολή των αντικειμένων (*onGhostAlwaysObjectsReceived()*).

Τρίτη φάση

Πλέον έχει ολοκληρωθεί το κατέβασμα της αποστολής από τον εξυπηρετητή και ο πελάτης αναλαμβάνει να φωτίσει τη σκηνή:

- ενεργοποίηση του συστήματος παραγωγής πολλών όμοιων αντικειμένων από ένα πρωτότυπο (*startClientReplication()*),
- ενεργοποίηση του συστήματος παραγωγής πολλών όμοιων αντικειμένων που αναπαριστούν βλάστηση από ένα πρωτότυπο (*startFoliageReplication()*),
- φωτισμός της αποστολής (*lightScene()*),
- ολοκλήρωση της διαδικασίας φωτισμού (*sceneLightingComplete()*).

Οι τρεις αυτές φάσεις παρουσιάζονται στο παρακάτω διάγραμμα ενώ ακολουθούν και τα τμήματα κώδικα σε εξυπηρετητή και πελάτη που τις υλοποιούν.



Σχήμα 6.3: Διαδικασία φόρτωσης αποστολής από τον εξυπηρετητή στον πελάτη

```
//==================================================
//  common/server/missionDownload.cs
//
//  Mission loading FROM SERVER SIDE->The server sends
//the mission to each connected client.
//Loading Phases:
// Phase 1: Transmit Datablocks
//          Transmit targets
// Phase 2: Transmit Ghost Objects
// Phase 3: Start Game (on the client side, phase 3 is to light
    the mission!!)
//
```

```
// The server invokes the client MissionStartPhase[1-3]
   function to request
//permission to start each phase.  When a client is ready for a
    phase,
//it responds with MissionStartPhase[1-3]Ack.
//=================================================


//****************************************
//* function GameConnection::loadMission(%this)
//*Args:
//*    (1)%this: The connected client who wants to download
//*the mission.
//*Purpose:
//*    Send over the information that will display the server
   info.
//*When we learn it got there, we'll send the data blocks.
//****************************************
function GameConnection::loadMission(%this)
{
   %this.currentPhase=0;
   if(%this.isAIControlled())
      %this.onClientEnterGame();
   else
      commandToClient(%this,'MissionStartPhase1',
         $missionSequence,$Server::MissionFile,MissionGroup.
         musicTrack);
}

//--------------------------------------------------------
//                   Phase 1
//--------------------------------------------------------

//Server starts Phase 1 when gets the ACK from client
function serverCmdMissionStartPhase1Ack(%client, %seq)
{
   // Make sure to ignore calls from a previous mission load
   if (%seq!=$missionSequence||!$MissionRunning)
      return;
  if (%client.currentPhase!= 0)
    return;
  %client.currentPhase=1;

  // Set the CRC value for the current mission.
  //This allows client to determine whether they need to
  //relight a scene or not. The CRC value is used to calculate
  //the signature of a mission LIGHTING file (.ml). If the
     signature
  //does not match the current CRC, the mission will be relight
     .
  %client.setMissionCRC($missionCRC);

  // Send over the datablocks.
```

```
  // OnDataBlocksDone will get called when have confirmation
  //that they've all been received.
  %client.transmitDataBlocks($missionSequence);
}

// Torque calls onDataBlocksDone() callback when datablocks
   have
//been received from client
function GameConnection::onDataBlocksDone(%this,%
   missionSequence)
{
  //Make sure to ignore calls from a previous mission load
  if (%missionSequence!=$missionSequence)
     return;
  if (%this.currentPhase!= 1)
     return;
  %this.currentPhase=1.5;

  //Inform client that you are going to start Phase 2 and wait
     for ACK
  commandToClient(%this, 'MissionStartPhase2', $missionSequence
     , $Server::MissionFile);
}

//--------------------------------------------------------
//                    Phase 2
//--------------------------------------------------------

//Server starts Phase 2 when gets the ACK from client
function serverCmdMissionStartPhase2Ack(%client, %seq)
{
  // Make sure to ignore calls from a previous mission load
  if (%seq != $missionSequence || !$MissionRunning)
     return;
  if (%client.currentPhase != 1.5)
     return;
  %client.currentPhase = 2;
  // Update mod paths, this needs to get there before the
     objects.
  %client.transmitPaths();

  // Start ghosting objects to the client
  // GHOSTS: all objects are created on the server and then
     some
  //of these are duplicated on the client. These duplicates are
      called
  //ghosts.
  %client.activateGhosting();
}

// Torque calls onGhostAlwaysObjectsReceived callback when
//the ghosting process is complete. Now Server is ready for the
     next phase.
```

```
function GameConnection::onGhostAlwaysObjectsReceived(%client)
{
   // Ready for next phase.
   commandToClient(%client, 'MissionStartPhase3',
      $missionSequence, $Server::MissionFile);
}

//----------------------------------------------------------
//                    Phase 3
//----------------------------------------------------------

function serverCmdMissionStartPhase3Ack(%client, %seq)
{
  // Make sure to ignore calls from a previous mission load
  if(%seq != $missionSequence || !$MissionRunning)
   return;
  if(%client.currentPhase != 2)
    return;
  %client.currentPhase = 3;

  // Server is ready to drop into the game
  %client.startMission();
  %client.onClientEnterGame();
}
```

```
//==============================================
//  common/client/missionDownload.cs
//
//  Mission loading FROM CLIENT SIDE
//Loading Phases:
// Phase 1: Download Datablocks
// Phase 2: Download Ghost Objects
// Phase 3: Scene Lighting
//==============================================

//---------------------------------
// Phase 1
//---------------------------------

function clientCmdMissionStartPhase1(%seq, %missionName, %
   musicTrack)
{
   onMissionDownloadPhase1(%missionName, %musicTrack);
   commandToServer('MissionStartPhase1Ack', %seq);
}

function onDataBlockObjectReceived(%index, %total)
{
   onPhase1Progress(%index/%total);
}

//---------------------------------
```

```
// Phase 2
//---------------------------------
function clientCmdMissionStartPhase2(%seq,%missionName)
{
  onPhase1Complete();
  purgeResources();
  onMissionDownloadPhase2(%missionName);
  commandToServer('MissionStartPhase2Ack', %seq);
}

function onGhostAlwaysStarted(%ghostCount)
{
   $ghostCount = %ghostCount;
   $ghostsRecvd = 0;
}

function onGhostAlwaysObjectReceived()
{
   $ghostsRecvd++;
   onPhase2Progress($ghostsRecvd / $ghostCount);
}

//---------------------------------------
// Phase 3
//---------------------------------------

function clientCmdMissionStartPhase3(%seq,%missionName)
{
  onPhase2Complete();
  StartClientReplication();
  StartFoliageReplication();
  $MSeq = %seq;
  $Client::MissionFile = %missionName;

  // Need to light the mission before we are ready.
  // The sceneLightingComplete function will complete the
     handshake
  // once the scene lighting is done.
  lightScene("sceneLightingComplete", "");
  if (lightScene("sceneLightingComplete", ""))
  {
     schedule(1, 0, "updateLightingProgress");
     onMissionDownloadPhase3(%missionName);
     $lightingMission = true;
  }
}

function updateLightingProgress()
{
  onPhase3Progress($SceneLighting::lightingProgress);
  if ($lightingMission)
   $lightingProgressThread = schedule(1, 0, "
       updateLightingProgress");
```

```
}

function sceneLightingComplete()
{
   onPhase3Complete();
   // The is also the end of the mission load cycle.
   onMissionDownloadComplete();
   commandToServer('MissionStartPhase3Ack', $MSeq);
}
```

6.2 Γραφικές Διασυνδέσεις Χρήστη (GUIs)

Γενικά, σε εφαρμογές πολυμέσων τα γραφικά περιβάλλοντα διασύνδεσης του χρήστη είναι το μέσο με το οποίο ο τελευταίος επικοινωνεί με την εφαρμογή. Στην περίπτωση μας ο παίκτης μέσω αυτών θα πλοηγηθεί στον εικονικό κόσμο που έχουμε σχεδιάσει.

Ανατρέχοντας στο παράρτημα, στην ιεραρχία των κλάσεων της μηχανής θα εντοπίσουμε την κλάση **"GuiControl"**. Πρόκειται για τη ρίζα των υποκλάσεων GUI (δεν φαίνονται στο σχήμα). Οι κλάσεις αυτές υλοποιούν τις λειτουργίες που προσφέρει η μηχανή για τη δημιουργία των γραφικών διασυνδέσεων του χρήστη (Graphical User Interfaces).

Σε αντιστοιχία με όσα είδαμε για το διαχωρισμό σε κλάσεις και datablocks, στην περίπτωση των GUIs, έχουμε κλάσεις και **profiles**. Τα profiles είναι συλλογή ιδιοτήτων, κάτι σαν τα style sheets της HTML. Εδώ όμως δεν υπάρχει ξεχωριστή κλάση profile για κάθε GUI κλάση (θυμηθείτε ότι στην κλάση "Player" αντιστοιχεί το datablock "PlayerData", στην "ShapeBase" το "ShapeBaseData" κ.ο.κ) αλλά μόνο το **"GuiControlProfile"** από το οποίο δημιουργούνται όλα τα αντικείμενα profile (ενδεικτικά ανατρέξτε στο αρχείο "./control/client/Profiles.cs"). Ως παράδειγμα παρατίθεται το παρακάτω τμήμα κώδικα στο οποίο υπάρχει το αντικείμενο "FinalScreen1" της κλάσης "GuiBitmapCtrl" (υποκλάση της "GuiControl") και το οποίο, εκτός των άλλων, χρησιμοποιεί το profile "GuiDefaultProfile" (αντικείμενο τύπου "GuiControlProfile"):

```
//--------------------------------------------
//./control/client/interfaces/playerInterface.gui
//--------------------------------------------
//--- OBJECT WRITE BEGIN ---
new GameTSCtrl(PlayerInterface) {
   profile = "GuiContentProfile";
   horizSizing = "right";
   vertSizing = "bottom";
   position = "0 0";
   ...
   ...
   new GuiCrossHairHud() {
      profile = "GuiDefaultProfile";
      horizSizing = "center";
      ...
      ...
   };
   new GuiBitmapCtrl(FinalScreen1) {
```

```
      profile = "GuiDefaultProfile";
      horizSizing = "center";
      vertSizing = "center";
      position = "160 60";
      extent = "320 240";
      minExtent = "8 8";
      visible = "0";
      helpTag = "0";
      bitmap = "./art/lastmessage1";
      wrap = "0";
   };
   ...
   ...
};
```

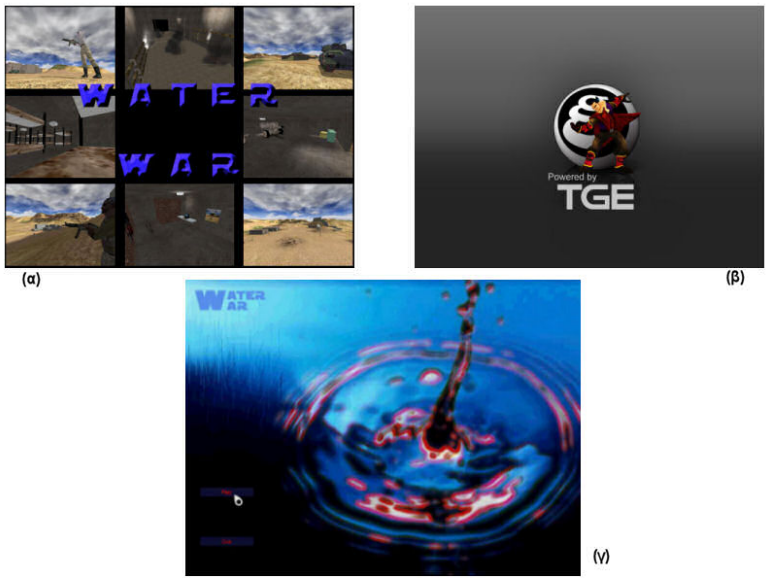
Τα GUIs βρίσκονται, όπως είδαμε, στην πλευρά του πελάτη. Όταν ο πελάτης εκτελεστεί, πρέπει κατά τις αρχικοποιήσεις του να δημιουργήσει ένα αντικείμενο της κλάσης "GuiCanvas". Αυτό δεν γίνεται με το "new" όπως για τα άλλα αντικείμενα κλάσεων, αλλά καλώντας την *createCanvas()* (βλ. αρχείο ./common/client/canvas.cs). Αφού γίνει αυτό, το αντικείμενο που κατασκεύασε εσωτερικά η μηχανή είναι διαθέσιμο με το όνομα **"Canvas"**. Ένας πελάτης πρέπει να έχει ένα και μόνο αντικείμενο της κλάσης "GuiCanvas", μέσα στο οποίο τοποθετούνται όλα τα άλλα αντικείμενα GUI κλάσεων. Συνεπώς, το αντικείμενο της κλάσης "GuiCanvas" πρέπει να δημιουργηθεί πριν από οποιοδήποτε άλλο αντικείμενο GUI κλάσεων. Η τοποθέτηση άλλων αντικειμένων GUI κλάσεων στο "Canvas" γίνεται με τις μεθόδους *setContent()* και *pushDialog()*. Η πρώτη, κάθε φορά που καλείται, απομακρύνει το προηγούμενο αντικείμενο από το Canvas. Μόνο με *pushDialog()* μας επιτρέπεται ένα GUI αντικείμενο να εμφανίζεται "πάνω" σε άλλο!

6.2.1 Οθόνες εισαγωγής και βασικό μενού

Με την έναρξη του παιχνιδιού εμφανίζονται δύο οθόνες εισαγωγής. Η πρώτη είναι μία εικόνα από στιγμιότυπα του παιχνιδιού (screenshots) και στην οποία αναγράφεται ο τίτλος του. Παραμένει ενεργή για πέντε δευτερόλεπτα, στο πρώτο από τα οποία χρησιμοποιείται εφέ σταδιακής εμφάνισης (fade in) και στο τελευταίο, εφέ σταδιακής απενεργοποίησης (fade out) (σχετικό αρχείο: ./control/client/interfaces/introScreen.cs).

Μετά τα πέντε αυτά δευτερόλεπτα, στο "Canvas" αντικείμενο τοποθετείται (*setContent()*) η οθόνη που αποτελείται από την εικόνα με το λογότυπο της TGE. Και εδώ έχουν υλοποιηθεί τα ίδια με προηγουμένως (σχετικό αρχείο: ./control/client/interfaces/splashScreen.cs). Με το πέρας των πέντε, νέων, δευτερολέπτων (δηλαδή από την έναρξη του παιχνιδιού έχουν μεσολαβήσει συνολικά δέκα δευτερόλεπτα) εμφανίζεται το βασικό μενού.

Το GUI για το βασικό μενού διαφέρει από τις οθόνες εισαγωγής. Στην τελευταία περίπτωση κάθε οθόνη προγραμματίζεται ως ένα και μόνο αντικείμενο, το οποίο παραμένει ενεργό για κάποιο χρονικό διάστημα και, το σημαντικότερο, δεν δέχεται καμία είσοδο από τον παίκτη. Για το βασικό μενού όμως, έχει υλοποιηθεί ένα αντικείμενο ("menuScreen") μέσα στο οποίο ορίζονται επιπλέον αντικείμενα. Τα επιπλέον αυτά αντικείμενα είναι δύο και αντιστοιχούν στα δύο κουμπιά που υπάρχουν στο βασικό μενού (σχετικό αρχείο: ./control/client/interfaces/menuScreen.cs).



Σχήμα 6.4: (α), (β) Οθόνες εισαγωγής - (γ) Βασικό μενού

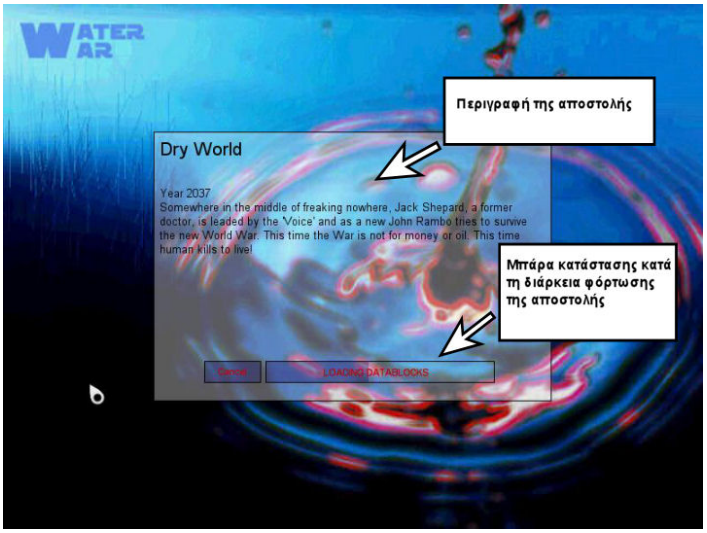
Κάθε κουμπί ("Play" και "Quit") έχει συνδεθεί και με μία συνάρτηση. Το "Play" με τη *launchGame()*, ενώ το "Quit" με τη *quit()*. Ο ρόλος της *quit()* είναι προφανής. Για τη *launchGame()* ανατρέξτε στα σχήματα 6.1 και 6.2. Είναι η συνάρτηση από την οποία αρχίζει η δημιουργία του εξυπηρετητή, η σύνδεσή του με τον πελάτη και ό,τι είδαμε στη σχετική ενότητα. Μάλιστα η διαδικασία της φόρτωσης της αποστολής από τον πελάτη, οι τρεις φάσεις δηλαδή που περιγράφτηκαν, εμφανίζονται σε νέο παράθυρο. Στο ίδιο αυτό παράθυρο παρουσιάζονται και οι πληροφορίες της αποστολής που συνελέχθησαν (σχήμα 6.5). Από εδώ ξεκινάει, στην ουσία, το παιχνίδι!

Πριν συνεχίσουμε να σημειώσουμε ότι έχει επικρατήσει, στη Torque, να μιλάμε για **controls**, όταν θέλουμε να αναφερθούμε γενικά σε αντικείμενα των Gui κλάσεων. Ένα control που περιέχει άλλα controls είναι **container** για αυτά. Για controls που ορίζονται μέσα στο ίδιο container και καταλαμβάνουν την ίδια περιοχή έχει σημασία η σειρά δημιουργίας τους (κάθε control θα καλύπτει τα προηγούμενά του).

6.2.2 Head-up Display (HUD)

Όταν ο πελάτης συνδεθεί με τον εξυπηρετητή και εισέλθει στον κόσμο, αναλαμβάνει πλέον την απεικόνιση του κόσμου με βάση τις πληροφορίες που του στέλνει ο εξυπηρετητής. Έτσι ο χρήστης μπορεί να βλέπει τι γίνεται στον κόσμο και να δρα ανάλογα.

Η απεικόνιση του κόσμου είναι μία υπόθεση διαφορετική από τις οθόνες εισαγωγής και το μενού. Εκεί τα GUIs είναι, σε τελική ανάλυση, εικόνες και μπορούν άμεσα να αντιστοιχηθούν σε pixels της οθόνης. Ο κόσμος όμως αποτελείται από μία συλλογή τρισδιάστατων αντικειμένων. Σε αυτά μπορεί να εφαρμόζονται υφές και να υπάρχει κάποιο μοντέλο φωτισμού. Απαιτείται λοιπόν μία πολύπλοκη διαδικασία για τη μετατροπή αυτής της περιγραφής του κόσμου σε μία δυσδιάστατη εικόνα που να μπορεί να παρουσιαστεί στην οθόνη. Αυτό είναι δουλειά της μηχανής και συγκεκριμένα του "renderer".



Σχήμα 6.5: Η οθόνη που εμφανίζεται όταν πατηθεί το κουμπί "Play"

Υπάρχει μία ειδική κλάση GUI που αναλαμβάνει να αποτυπώσει στην οθόνη τα αποτελέσματα του "rendering". Είναι η κλάση **GameTSCtrl**. Δημιουργούμε ένα αντικείμενο αυτής της κλάσης (σχετικό αρχείο: /control/client/interfaces/playerInterface.gui). **Έχει καθιερωθεί η εν λόγω διασύνδεση, μέσω της οποίας βλέπουμε τον κόσμο, πληροφορίες όπως η ενέργεια του ήρωα, μηνύματα σε μορφή κειμένου κλπ. να ονομάζεται HUD (Head-Up Display[1])**. Ο πελάτης θέτει το HUD σαν περιεχόμενο του Canvas (*setContent()*) και πλέον ο χρήστης μπορεί να δει τον κόσμο.

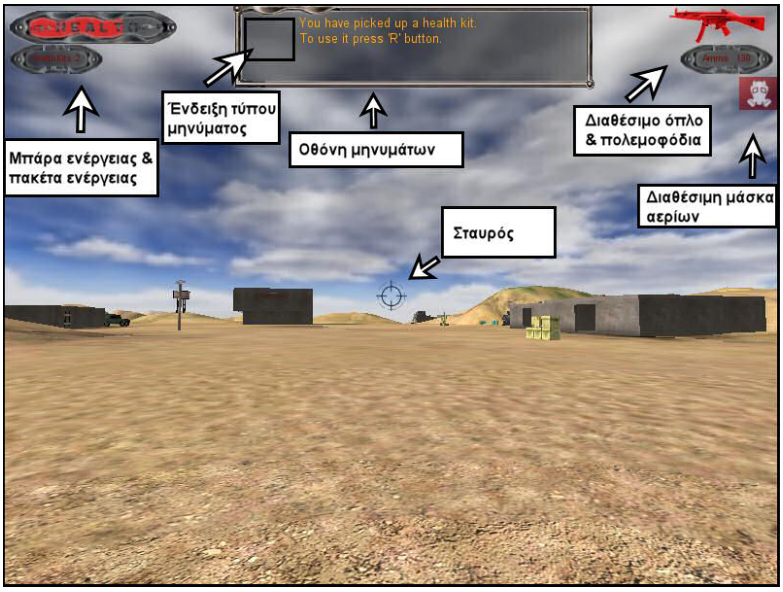
Υπάρχουν κάποιες ειδικές κλάσεις, τα αντικείμενα των οποίων προορίζονται για χρήση σε ένα HUD. Δύο από αυτές είναι οι κλάσεις "GuiCrossHairHud" και "GuiHealthBarHud". Η πρώτη μας επιτρέπει να εμφανίσουμε στο μέσο της οθόνης, το συνηθισμένο "σταυρό" (ή οτιδήποτε άλλο) που συναντάται στα παιχνίδια τύπου FPS. Ο "σταυρός " υποδεικνύει τη διεύθυνση που κοιτάει/στοχεύει ο παίκτης. Η κλάση "GuiHealthBarHud" χρησιμοποιείται για τη δημιουργία αντικειμένων που απεικονίζουν τις μπάρες ενέργειας.

"Μία εικόνα ίσον χίλιες λέξεις " λένε, οπότε στο σχήμα 6.6 παρουσιάζουμε το HUD του παιχνιδιού. Έτσι ο παίκτης θα πλοηγείται στον κόσμο του παιχνιδιού! Τα αντικείμενα που συνθέτουν το HUD επεξηγούνται στην εικόνα. Το μόνο που χρειάζεται να ξεκαθαρίσουμε είναι η "ένδειξη τύπου μηνύματος ".

Στο παιχνίδι έχουμε δύο ειδών μηνύματα τα οποία εμφανίζονται στην οθόνη μηνυμάτων:

- απλές πληροφορίες που εμφανίζονται κάθε φορά που ο παίκτης αλληλεπιδρά με κάποιο αντικείμενο (για παράδειγμα μαζεύει πακέτα ενέργειας ή πολεμοφόδια) και
- μηνύματα που δέχεται από τον "άνώτερό του".

[1]Ο όρος προέρχεται από την οθόνη που χρησιμοποιούν οι πιλότοι στην πολεμική αεροπορία και στην οποία εμφανίζονται διάφορες πληροφορίες εν ώρα πτήσης.



Σχήμα 6.6: Head-Up Display του WaterWar

Για να διακρίνουμε τις δύο αυτές κατηγορίες, εμφανίζεται κάθε φορά το αντίστοιχο εικονίδια του σχήματος 6.7, σε θέση αριστερά του μηνύματος.

Σχήμα 6.7: Εικονίδιο (α) μηνύματος από "Φωνή", (β) απλής πληροφορίας

6.2.3 Η οθόνη μηνυμάτων του HUD

Ανοίγοντας το αρχείο "./control/client/interfaces/playerInterface.gui" στο οποίο ορίζονται τα αντικείμενα που συνθέτουν το HUD, δεν θα βρούμε το GUI αντικείμενο για την οθόνη μηνυμάτων. Η οθόνη μηνυμάτων έχει οριστεί σε ξεχωριστό αρχείο και εμφανίζεται στο HUD με τη βοήθεια της μεθόδου *pushDialog()* (όπως σημειώσαμε νωρίτερα με τη μέθοδο αυτή τοποθετούμε νέα αντικείμενα GUI "πάνω" στα ήδη υπάρχοντα αντικείμενα του Canvas). Η διαδικασία αυτή φαίνεται στο τμήμα κώδικα που ακολουθεί. Η *onWake()* είναι κλήση συστήματος (**callback**) και καλείται με την ενεργοποίηση του HUD, δηλαδή με την είσοδο του παίκτη στο παιχνίδι.

Listing 6.1: Τμήμα του κώδικα του αρχείου "./control/client/misc/screens.cs"

```
function PlayerInterface::onWake(%this)
{
   $enableDirectInput="1";
   activateDirectInput();
```

```
   //Push-make visible the MessageBox
   Canvas.pushDialog(MainMessageHud);
   chatHud.attach(mainMessageVector);

   playerKeymap.push();
}
...
...
```

Το "MainMessageHud" είναι το αντικείμενο GUI που χρησιμοποιούμε για την οθόνη μηνυμάτων. Ορίζεται στο αρχείο "./control/client/interfaces/messageBox.gui". Πρόκειται για ένα "container" που ορίζει μια σειρά από "controls". Κάποια από αυτά έχουν να κάνουν με το πλαίσιο που περιβάλει την οθόνη μηνυμάτων, άλλα με το εικονίδιο ένδειξης που περιγράψαμε μόλις πριν. Το σημαντικότερο ωστόσο αντικείμενο-"control" που ορίζεται στο "MainMessageHud" είναι το "ChatHud", τύπου **"GuiMessageVectorCtrl"**.

Listing 6.2: Ορισμός του "MainMessageHud"

```
//--- OBJECT WRITE BEGIN ---
new GuiControl(MainMessageHud)
{
   profile = "GuiModelessDialogProfile";
   horizSizing = "width";
   vertSizing = "height";
   position = "0 0";
   extent = "640 480";
   minExtent = "8 8";
   visible = "1";
   helpTag = "0";
   noCursor = "1";

   new GuiControl(MainMessageHudContainer)
   {
      profile = "GuiDefaultProfile";
      horizSizing = "relative";
      vertSizing = "bottom";
      position = "190 00";
      extent = "400 250";
      minExtent = "8 8";
      visible = "1";
      helpTag = "0";

      new GuiBitmapCtrl(InfoIcon)
      {
            profile = "GuiDefaultProfile";
            horizSizing = "right";
            vertSizing = "top";
            position = "20 25";
            extent = "32 32";
            minExtent = "8 8";
            visible = "0";
            helpTag = "0";
```

```
            bitmap = "./art/info_icon";
            wrap = "0";
            };

         new GuiBitmapCtrl(VoiceIcon)
         {
            profile = "GuiDefaultProfile";
            horizSizing = "right";
            vertSizing = "top";
            position = "20 25";
            extent = "32 32";
            minExtent = "8 8";
            visible = "0";
            helpTag = "0";
            bitmap = "./art/voice_icon";
            wrap = "0";
         };

      new GuiBitmapBorderCtrl(OuterChatHud)
      {
         profile = "ChatBoxBorderProfile";
         horizSizing = "width";
         vertSizing = "bottom";
         position = "0 0";
         extent = "272 88";
         minExtent = "8 8";
         visible = "1";
         helpTag = "0";
         useVariable = "0";
         tile = "0";

         new GuiBitmapCtrl()
         {
            profile = "GuiDefaultProfile";
            horizSizing = "width";
            vertSizing = "height";
            position = "8 8";
            extent = "256 72";
            minExtent = "8 8";
            visible = "1";
            helpTag = "0";
            bitmap = "./art/hudfill.png";
            wrap = "0";
         };

         new GuiButtonCtrl(chatPageDown)
         {
            profile = "GuiButtonProfile";
            horizSizing = "left";
            vertSizing = "top";
            position = "220 58";
            extent = "36 14";
            minExtent = "8 8";
```

```
            visible = "0";
            helpTag = "0";
            text = "Dwn";
            groupNum = "-1";
            buttonType = "PushButton";
         };

         new GuiScrollCtrl(ChatScrollHud)
         {
            profile = "ChatBoxScrollProfile";
            horizSizing = "width";
            vertSizing = "height";
            position = "58 10";
            extent = "256 72";
            minExtent = "8 8";
            visible = "1";
            helpTag = "0";
            willFirstRespond = "1";
            hScrollBar = "alwaysOff";
            vScrollBar = "alwaysOff";
            constantThumbHeight = "0";
            childMargin = "0 0";

            new GuiMessageVectorCtrl(ChatHud)
            {
               profile = "ChatBoxMessageProfile";
               horizSizing = "width";
               vertSizing = "height";
               position = "1 1";
               extent = "252 16";
               minExtent = "8 8";
               visible = "1";
               helpTag = "0";
               lineSpacing = "0";
               lineContinuedIndex = "10";
               maxColorIndex = "5";
            };
         };
      };
   };
};
//--- OBJECT WRITE END ---
```

Στόχος των αντικειμένων τύπου "GuiMessageVectorCtrl" είναι να εμφανίζουν τα περιεχόμενα του **"MessageVector"** με το οποίο συνδέονται. "MessageVector" είναι μία κλάση που μπορεί να περιέχει κείμενο. Δεν έχει πεδία και δημιουργείται με την εξής δήλωση:

```
$myMsgVector=new MessageVector();
```

Από τη στιγμή που έχει δημιουργηθεί ένα στιγμιότυπο της "MessageVector", συμπεριφέρεται σαν ουρά (queue) και μπορούμε να προσθέσουμε γραμμές κειμένου με τις μεθόδους *pushFrontLine()* και *pushBackLine()*.

Επιστρέφοντας στην υλοποίησή μας, παρατηρείστε τη μέθοδο *attach()* στο τμήμα κώδικα 6.1. Η παράμετρος "mainMessageVector" είναι το αντικείμενο "MessageVector" που αντιστοιχεί στο "ChatHud" (η δήλωση του "mainMessageVector" βρίσκεται στο αρχείο "./control/client/misc/chatHud.cs"). Οτιδήποτε εμφανίζεται στην οθόνη μηνυμάτων είναι περιεχόμενα του "mainMessageVector".

Η επεξεργασία των μηνυμάτων που πρέπει να εμφανιστούν, γίνεται στην πλευρά του πελάτη. Ο εξυπηρετητής ειδοποιεί τον πελάτη ότι πρέπει να εμφανίσει κάποιο μήνυμα μέσω των συναρτήσεων *infoMessage()* (για απλές πληροφορίες που πρέπει να εμφανιστούν) και *voiceMessage()* (για μηνύματα που προέρχονται από τη "Φωνή") που ορίζονται στο αρχείο "./control/server/game.cs".

Στην πλευρά του πελάτη, οι παραπάνω συναρτήσεις αντιστοιχούν στις *clientCmdInfoMessage()* και *clientCmdVoiceMessage()* αντίστοιχα (βλ. "./control/client/interfaces/messageBox.gui"). Κάθε μία προσθέτει το απαραίτητο κείμενο στο "mainMessageVector" με κλήση, αντιστοίχως, των *infoMessages()* και *voiceMessages()* (βλ. "./control/client/misc/chatHud.cs"). Από εκεί αναλαμβάνει η *manageHudMessages()* (ορίζεται στο ίδιο αρχείο με τις δύο προηγούμενες).

Η συνάρτηση αυτή δίνει προτεραιότητα σε μηνύματα από τη "Φωνή". Δηλαδή αν ένα τέτοιο μήνυμα δεν έχει ολοκληρωθεί, τότε οι απλές πληροφορίες μπαίνουν σε αναμονή και εμφανίζονται στην οθόνη μετά το τέλος του. Επίσης η όλη διαδικασία εμφάνισης ενός μηνύματος περιέχει βοηθητικές συναρτήσεις για τη σταδιακή παρουσίαση των επιπλέον γραμμών του μηνύματος (η οθόνη χωράει μέχρι και τέσσερις γραμμές κειμένου).

Listing 6.3: Ο κώδικας για την εμφάνιση των μηνυμάτων στην οθόνη

```
//================================================
// ./contol/client/misc/chatHud.cs
//================================================

$scheduledEventToHideInfoIcon=-1;
$howManyEventsOfScrollUpdate=0;

new MessageVector(mainMessageVector);

//%type->1 for voice messages, 0 for info messages
function manageHudMessages(%this,%text,%type)
{
   //enable/disable icons
   InfoIcon.visible=!%type;
   VoiceIcon.visible=%type;

   //make sure the infoIcon is off when the time has passed
   $scheduledEventToHideInfoIcon=schedule(4000,0,hideInfoIcon);

   //for voice message make sure the messageBox is visible
   if(%type==1)
      MainMessageHudContainer.setVisible(true);

   //a voiceMessage ends with #
```

```
    if(%text$="#")
    {
        $endOfVoiceMessage=true;
        VoiceIcon.visible=false;
    }
    else
        %this.pushBackLine(%text,0);  //add line at the bottom of
            messageBox
}

function voiceMessages(%line,%i)
{
    $endOfVoiceMessage=false;
    %delay=2000*%i;
    if(%i==1)    //this is mean that we have a new voiceMessage
    {
        cancelAllScrollEvents();
        mainMessageVector.clear();
    }
    schedule(%delay,0,manageHudMessages,mainMessageVector,%line
        ,1);
}

function infoMessages(%text,%i)
{
    if($endOfVoiceMessage==false) //wait until a voiceMessage is
         over
    {
        schedule(2000,0,infoMessages,%text,%i);
    }
    else
    {
        if(%i==1)    //this is mean that we have a new infoMessage
        {
            mainMessageVector.clear();
            cancel($scheduledEventToHideInfoIcon);
            cancelAllScrollEvents();
        }

        //messageBox can display 4 lines
        %numOfLines=getFieldCount(%text);
        if (%numOfLines<=4) //4 lines message
            manageHudMessages(mainMessageVector,%text,0);
        else  //more than 4 lines
            scrollMessage(%text,%numOfLines);
    }
}

function hideInfoIcon()
{
    InfoIcon.visible=false;
}
```

```
function scrollMessage(%text,%numOfLines)
{
   //the first 4 lines --the index for line1 is 0, for line2 is
       1 etc
   %mainPartOfText=getFields(%text, 0, (%numOfLines<5 ? %
      numOfLines-1 : 4-1));

   %extraLine="";
   %newText=%mainPartOfText;
   %numOfExtraLines=%numOfLines-4;

   for(%j=0;%j<=%numOfExtraLines;%j++)
   {
      scrollUpdate(%newText,%j);
      %extraLine=getField(%text,4+%j); //add one new line each
         time
      %newText=%newText NL %extraLine;
   }
}

//ana 5sec 8a emfanizetai mia nea grammh sto message box
function scrollUpdate(%text,%j)
{
   $howManyEventsOfScrollUpdate++;
   $scheduledEventToScrollText[%j]=schedule(5000*%j,0,
      manageHudMessages,mainMessageVector,%text,0);
}

//an to scroll den exei teleiwsei kai er8ei neo infoMessage
//me thn synarthsh auth "akyrwnoume" to scroll tou prohgoumenou
//mhnymatos kai etsi den yparxei periptwsh na emfanistei
//sto neo mhnyma plhrofories apo to prohgoumeno
function cancelAllScrollEvents()
{
   for(%numOfEvents=0;%numOfEvents<$howManyEventsOfScrollUpdate
      ;%numOfEvents++)
      cancel($scheduledEventToScrollText[%numOfEvents]);
   $howManyEventsOfScrollUpdate=0;
}
```

6.2.4 Οθόνη διαλόγου

Η οθόνη διαλόγου εμφανίζεται στο HUD όταν ο ήρωας συνομιλεί με τον φυλακισμένο ή όταν διαβάζει ένα βιβλίο. Έχει υλοποιηθεί με τον ίδιο τρόπο που αναπτύχθηκε και η οθόνη μηνυμάτων.

Σε αναλογία λοιπόν με τα προηγούμενα, το "container" είναι το "TheDialogHud" που ορίζεται στο αρχείο "./control/client/interfaces/dialogBox". Περιέχει το αντικείμενο "dialogHud", τύπου "GuiMessageVectorCtrl". Το "MessageVector" για το κείμενο των διαλόγων είναι το "dialogVector". Οι κλήσεις των απαιτούμενων συναρτήσεων βρίσκονται μέσα στο αρχείο "./control/client/misc/dialogHud.cs".

Το μόνο που αξίζει να σημειωθεί εδώ είναι ότι χρειάζεται ο παίκτης να προβεί σε

κάποια ενέργεια για να εμφανιστεί η εν λόγω οθόνη. Συγκεκριμένα, όταν ο ήρωας βρεθεί σε θέση, μέσα στο εικονικό κόσμο, όπου μπορεί να αλληλεπιδράσει με το βιβλίο ή τον φυλακισμένο (χρησιμοποιείται ο μηχανισμός των **triggers** για τον έλεγχο της θέσης του παίκτη), τότε ειδοποιείται ο χρήστης (στην οθόνη μηνυμάτων) να πατήσει το πλήκτρο "F". Με το που πατηθεί το πλήκτρο, ο εξυπηρετητής στέλνει μήνυμα στην πλευρά του πελάτη (*readbook()* και *dialogMessage()* - βλ. αρχείο"./control/server/game.cs") για να ενεργήσει ανάλογα.

Σχήμα 6.8: Οθόνη διαλόγου

6.2.5 Πυξίδα

Με το πλήκτρο "tilde" γίνεται εναλλαγή μεταξύ οθόνης μηνυμάτων και πυξίδας. Το ενδιαφέρον εδώ είναι ο τρόπος με τον οποίο "στήθηκε" η πυξίδα.

Η λογική πίσω από την πυξίδα είναι η χρήση μίας ταινίας/λωρίδας που περιέχει τις διευθύνσεις του ορίζοντα. Η ταινία αυτή βρίσκεται μέσα σε ένα πλαίσιο, στο μέσο του οποίου υπάρχει ένας δείκτης. Ο δείκτης δηλώνει την κατεύθυνση προς την οποία κινείται ο παίκτης. Το μυστικό στην υλοποίηση είναι ότι το πλαίσιο παραμένει σταθερό και αυτό που μετακινείται είναι η ταινία! Αυτό σημαίνει ότι η ταινία πρέπει να είναι μεγαλύτερη από το πλαίσιο έτσι ώστε να μην υπάρχει κενό (στο πλαίσιο) κατά τη μετακίνησή της. Τα παραπάνω φαίνονται στην παρακάτω εικόνα.

Σχήμα 6.9: Τα επιμέρους τμήματα της πυξίδας

Η πυξίδα στην εφαρμογή αρχικοποιείται όπως και στην εικόνα. Το τμήμα της ταινίας που περισσεύει δεν εμφανίζεται στο HUD (κατάλληλες διαστάσεις στο GUI αντικείμενο

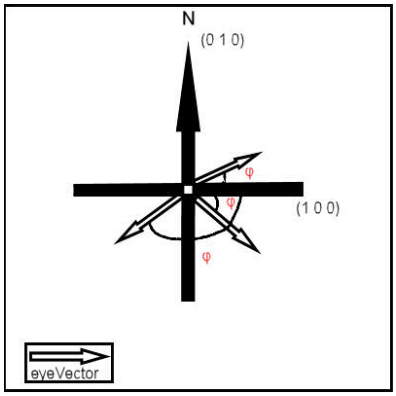
που αποτυπώνει την ταινία) και το μόνο ορατό είναι αυτό που βρίσκεται κάθε φορά μέσα στο πλαίσιο. Για να μην υπάρξει ποτέ κενό στο πλαίσιο, η ταινία πρέπει να κινείται πάντα αριστερά. Παρατηρείστε ότι με τον τρόπο αυτό και σε συνδυασμό με το πώς και πόσες φορές αναγράφονται οι διευθύνσεις πάνω στην ταινία, μπορούμε να έχουμε ένδειξη για όλες (τις διευθύνσεις του ορίζοντα).

Πρέπει τώρα να αναπτυχθεί ο μηχανισμός εκείνος όπου θα επιτρέπει στην πυξίδα να δίνει σωστές ενδείξεις, με βάση την κίνηση του παίκτη. Πρώτο βήμα είναι να βρεθεί προς τα που κατευθύνεται ο παίκτης. Σε αυτό θα βοηθήσει το εσωτερικό γινόμενο δύο διανυσμάτων.

Αρχικά χρησιμοποιείται το διάνυσμα "eyeVector" (κανονικοποιημένο) που δείχνει την κατεύθυνση προς την οποία κοιτάει ο παίκτης (το avatar του παίκτη). Το εσωτερικό γινόμενο του "eyeVector" με το διάνυσμα '1 0 0' (που αντιστοιχεί στην Ανατολή, στον κόσμο του παιχνιδιού μας) μας δίνει τη διεύθυνση του διανύσματος "eyeVector" σε σχέση με το Βορρά. Για το εσωτερικό γινόμενο δύο διανυσμάτων Α και Β γνωρίζουμε:

$$\vec{A} \cdot \vec{B} = 0 \Leftrightarrow \varphi = \pi/2$$
$$\vec{A} \cdot \vec{B} > 0 \Leftrightarrow 0 \leq \varphi < \pi/2$$
$$\vec{A} \cdot \vec{B} < 0 \Leftrightarrow \pi/2 < \varphi \leq \pi$$

Άρα, και σε συνδυασμό με το σχήμα 6.10, αρνητική τιμή εσωτερικού γινομένου στην περίπτωση μας σημαίνει ότι ο παίκτης κοιτάει αριστερά σε σχέση με το Βορρά.



Σχήμα 6.10: EyeVector και σημεία του ορίζοντα

Στη συνέχεια παίρνουμε εσωτερικό γινόμενο του "eyeVector" με το διάνυσμα '0 1 0' (αντιστοιχεί στο Βορρά, στον κόσμο του παιχνιδιού μας). Ουσιαστικά πρόκειται για το συνημίτονο της γωνίας φ μεταξύ αυτών των διανυσμάτων αφού είναι και τα δύο κανονικοποιημένα (μοναδιαίου μέτρου). Μέσω αυτού μπορούμε πλέον να υπολογίσουμε τη γωνία φ σε ακτίνια (arc cosine).

Απομένει λοιπόν να αποτυπωθεί η γωνία αυτή πάνω στην ταινία της πυξίδας ή πιο σωστά να αντιστοιχήσουμε τη μετακίνηση της ταινίας με τη γωνία φ, για να έχουμε σωστή ένδειξη. Ο τύπος που χρησιμοποιείται είναι:

νέαΘέσηΤαινίας= 8/12 * ΠοσοστόΠεριστροφής * ΣχέσηΤαινίαςΜεΠλαίσιο * ΜήκοςΤαινίας

όπου

- νέαΘέσηΤαινίας: η θέση της ταινίας στον X άξονα. Έχει να κάνει με την παράμετρο "position" του GUI αντικειμένου που αντιστοιχεί στην ταινία,
- 8/12: το πλήθος των σημείων του ορίζοντα που θέλουμε να αποτυπώσουμε προς το πλήθος των σημείων που αναγράφονται στην ταινία,
- ΠοσοστόΠεριστροφής: η γωνία που υπολογίστηκε προς 360,
- ΣχέσηΤαινίαςΜεΠλαίσιο: η πραγματική διάσταση "width" της εικόνας της ταινίας ως προς την πραγματική διάσταση "width" της εικόνας του πλαισίου (έτσι όπως έχουν αυτές αποθυκευτεί στο φάκελο "./control/client/interfaces/art"),
- ΜήκοςΤαινίας: η πραγματική διάσταση "width" της εικόνας της ταινίας.

Τόσο τα αντικείμενα GUI όσο και η συνάρτηση που υλοποιούν τα παραπάνω βρίσκονται στο αρχείο "./control/client/interfaces/compass.gui".

Listing 6.4: Το αρχείο "./control/client/interfaces/compass.gui"

```
new GuiControl(compassInterface)
{
   profile = "GuiDefaultProfile";
  horizSizing = "width";
  vertSizing = "height";
  position = "0 0";
  extent = "640 480";
  minExtent = "8 8";
  visible = "1";
  noCursor = "1";
   new GuiControl(container) {
   profile = "GuiDefaultProfile";
    horizSizing = "right";
    vertSizing = "bottom";
    position = "220 0";
    extent = "400 50";
    minExtent = "8 2";
    visible = "1";
      stripName = "compassStrip";
      stripWidth = 1200;
      new GuiBitmapCtrl(compassStrip) {
      profile = "GuiDefaultProfile";
      horizSizing = "right";
      vertSizing = "bottom";
      position = "0 0";
      extent = "1200 50";
      minExtent = "8 2";
      visible = "1";
      bitmap = "./art/compassStrip";
```

```
      wrap = "0";
    };
    new GuiBitmapCtrl(compassFrame) {
      profile = "GuiDefaultProfile";
      horizSizing = "right";
      vertSizing = "bottom";
      position = "0 0";
      extent = "400 50";
      minExtent = "8 2";
      visible = "1";
      bitmap = "./art/compassFrame";
      wrap = "0";
      };
   };
};
//--- OBJECT WRITE END ---

function clientCmdUpdateCompassStrip(%facingVector)
{
   //normalize facing vector (just in case)
   %facingVector=vectorNormalize(%facingVector);
   //dot product to check if we are looking east or west ("1 0
      0"->east)
   %leftFacing=(vectorDot("1 0 0",%facingVector)<0) ? true :
      false;
   //angle in RADIANS ( "0 1 0"->forward(north) )
   %forwardTheta=vectorDot("0 1 0",%facingVector);

   //radians to degrees
   if (%leftFacing)
         %rotationDegrees=360-(mACos(%forwardTheta)
            *180/3.1415927);
   else
         %rotationDegrees=mACos(%forwardTheta)*180/3.1415927;


   // We've created a strip that is three times as wide as the
      frame, giving
   // it 12 compass points vs. the normal 8.
   //
   // If we calculate our rotation as a percentage, account the
       ratio 8/12, and
   // scale based on our current extent vs. the pre-scaled
      width of the image, we can
   // calculate the exact position to place the strip at:

   %curPosY=getWord(container.stripName.getPosition(),1);
   %curExtX=getWord(container.stripName.getExtent(),0);
   %curExtY=getWord(container.stripName.getExtent(),1);

   %percentageRot=%rotationDegrees/360.0;
   %extentRatio=%curExtX/container.stripWidth;
```

```
    %newPosX=-1*(8/12 * %percentageRot * %extentRatio *
        container.stripWidth);
    container.stripName.resize(%newPosX,%curPosY,%curExtX,%
        curExtY);
}

function clientCmdmakeMessageBoxActive()
{
    Canvas.popDialog(compassInterface);
    MainMessageHudContainer.visible=true;
}
```

Όσο η πυξίδα είναι ενεργή ο εξυπηρετητής στέλνει στον πελάτη το "eyeVector", το οποίο συνεχώς μεταβάλλεται καθώς αλλάζει διευθύνσεις ο παίκτης (βλ. συνάρτηση *serverCmdCompass()* μέσα στο "./control/server/game/cs").

Σχήμα 6.11: Εμφάνιση της πυξίδας στο HUD

6.2.6 Blood Splash Screen

Πρόκειται για την οθόνη με την οποία αντιλαμβάνεται ο παίκτης ότι έχει τραυματιστεί. Υλοποιείται ως αντικείμενο τύπου "GuiBitmapCtrl", το οποίο επιτρέπει την εμφάνιση οποιασδήποτε εικόνας στην οθόνη. Περιέχεται στο container "PlayerInterface" το οποίο, όπως είδαμε και νωρίτερα, είναι υπεύθυνο για τα περισσότερα συστατικά του HUD.

Η παράμετρος "visible" του GUI αντικείμενου της "Blood Splash Screen" αρχικοποιείται σε μηδέν. Μόνο όταν ο παίκτης τραυματιστεί παίρνει την τιμή 1 και ενεργοποιείται η εν λόγω οθόνη. Η εικόνα που πρέπει να εμφανιστεί δηλώνεται στην παράμετρο "bitmap". Καλύπτει όλο το HUD κάνοντας δύσκολη την πλοήγηση του παίκτη. Σταδιακά αρχίζει και απενεργοποιείται.

Για να επιτευχθεί αυτό χρησιμοποιήθηκαν 4 ίδιες εικόνες με διαφορετική τιμή διαφάνειας (transparency) η κάθε μία. Η συνάρτηση *fadeOut()* (βλ. "./control/client/interfaces/playerInterface.gui") αναλαμβάνει να φορτώνει (να αλλάζει δηλαδή την παράμετρο "bitmap") την "περισσότερο διαφανή" από την προηγούμενη εικόνα, ανά μισό δευτερόλεπτο. Όταν εμφανιστεί και η τελευταία επαναφέρουμε την τιμή 0 στην παράμετρο "visible".

Επίσης σε διαδοχικά χτυπήματα έχει γίνει προγραμματισμός ώστε να σταματήσει το εφέ και να ξεκινήσει η όλη διαδικασία από την πρώτη εικόνα και πάλι (βλ. *ClearScheduledEvents()* στο ίδιο αρχείο με πριν).

Σχήμα 6.12: Blood Splash Screen

6.2.7 Οθόνες αλληλεπίδρασης με υπολογιστή

Ο παίκτης μπορεί να αλληλεπιδράσει με ηλεκτρονικό υπολογιστή για να συγκεντρώσει πληροφορίες. Το νέο στοιχείο που μπαίνει εδώ είναι η δημιουργία γραφικών διασυνδέσεων όπου δέχονται εισόδους από το πληκτρολόγιο όπως συμβαίνει και στον πραγματικό κόσμο.

Τα αντικείμενα GUI λοιπόν που υλοποιήθηκαν είναι τύπου "GuiTextEditCtrl" όπου δέχονται ως είσοδο μία απλή γραμμή κειμένου. Από εκεί και πέρα συνδυάζονται διάφορα αντικείμενα GUI, όπως αυτά που είδαμε παραπάνω, για να εμφανιστεί η εικόνα που απεικονίζει τον υπολογιστή, η επιφάνεια εργασίας του, παράθυρα που ανοίγουν και παρουσιάζουν μηνύματα κτλ. Παράλληλα αναπτύχθηκαν οι κατάλληλες συναρτήσεις που διαχειρίζονται την είσοδο του χρήστη και ελέγχουν για παράδειγμα αν ο κωδικός πρόσβασης στον υπολογιστή είναι έγκυρος, ή αν δόθηκε εντολή για άνοιγμα κάποιου αρχείου.

Σχήμα 6.13: Αλληλεπίδραση με υπολογιστή

6.3 Σύστημα καταγραφής (Inventory System)

Ο παίκτης κατά την περιήγηση του στον κόσμο του παιχνιδιού βρίσκει αντικείμενα με τα οποία μπορεί να αλληλεπιδράσει. Κάποια από αυτά μπορεί να τα μαζέψει και να τα χρησιμοποιήσει. Εδώ έρχεται η έννοια του συστήματος καταγραφής (**inventory system**). Αποτελεί έναν ενιαίο τρόπο παρουσίασης των αντικειμένων του χρήστη, δίνοντάς του τη δυνατότητα να παρατηρήσει τις ιδιότητές των αντικειμένων που κουβαλάει και ενδεχομένως να επιλέξει κάποιο από αυτά προς χρήση. Συνήθως αυτό γίνεται μέσω ενός νέου παραθύρου.

Το σύστημα στο παρόν παιχνίδι είναι πιο απλό. Ο παίκτης μπορεί να συγκεντρώσει πακέτα ενέργειας, όπλα, πολεμοφόδια και μάσκα αερίων. Νωρίτερα είδαμε τις θέσεις στο HUD όπου αυτά καταγράφονται. Η χρήση τους είναι άμεση με κάποιο κουμπί ("R" για το πακέτο ενέργειας, δεξί κλικ του ποντικιού για το όπλο, "M" για τη μάσκα) και παράλληλα ενημερώνονται οι αντίστοιχες οθόνες.

Συνοπτικά, στο σημείο αυτό, θα δούμε τον τρόπο με τον οποίο υλοποιήθηκε το σύστημα καταγραφής. Πρώτα από όλα να σημειωθεί ότι τα αντικείμενα τα οποία μπορεί να συλλέξει ο χρήστης ανήκουν στην κλάση "Item". Το αντίστοιχο datablock είναι το "ItemData".

Για κάθε αντικείμενο που θέλουμε να κρατάμε στο σύστημα μας έχει οριστεί ένα σχετικός πίνακας στις ιδιότητες του avatar. Ο πίνακας αυτός κρατάει το μέγιστο αριθμό από το συγκεκριμένο αντικείμενο που μπορεί να έχει στη διάθεσή του ο παίκτης. Για παράδειγμα, το παρακάτω τμήμα κώδικα είναι ο ορισμός του datablock που χρησιμοποιούμε για το αντικείμενο τύπου "Player" που ελέγχει ο παίκτης. Οι τέσσερις τελευταίες γραμμές της δήλωσης ορίζουν ότι ο παίκτης μπορεί να διαθέτει μέχρι ένα όπλο, εννιά πακέτα ενέργειας, τριακόσιες σφαίρες και μία μάσκα αερίου.

```
datablock PlayerData(JackShepard)
{
   shapeFile="~/data/avatars/jackShepard.dts";
   emap=true;
   renderFirstPerson=true;

   cameraMaxDist=4;
```

```
firstPersonOnly=false;
observeThroughObject=true;
cameraDefaultFOV=90;
cameraMinFOV=45;
cameraMaxFOV=120;
minLookAngle=-1.05;  //look up
maxLookAngle=1.00;
maxFreeLookAngle=2.1;

mass=90;
density=10;
drag=0.3;
maxDrag=0.4;
maxDamage=100;
maxEnergy=100;
repairRate=0.35;  //health
rechargeRate=0; //energy

footStepsOn=false;

maxForwardSpeed=15;
maxBackwardSpeed=12;
maxSideSpeed=12;
minJumpSpeed=20;
maxJumpSpeed=30;
runSurfaceAngle=44;
jumpSurfaceAngle=63;

runForce=48*90;
jumpForce=8*90;

runEnergyDrain=0;
jumpEnergyDrain=0;
minRunEnergy=0;
minJumpEnergy=0;

recoverDelay=62.5; //delay after a fall
recoverRunForceScale=1.2;
minImpactSpeed=15;
speedDamageScale=1.0;

groundImpactMinSpeed=10.5;
groundImpactShakeFreq="4.0 4.0 4.0";
groundImpactShakeAmp="1.25 1.25 1.25";
groundImpactShakeDuration=1.25;
groundImpactShakeFalloff=8.0;

//multiplier for hit that causes damage (from enemy)
headHit=1.45;
torsoHit=0.55;
legsHit=0.10;

maxInventory[HealthKit]=9;
```

```
    maxInventory[MP5]=1;
    maxInventory[MP5Ammo]=300;
    maxInventory[GasMask]=1;
};
```

Ας δούμε τις ενέργειες που συμβαίνουν με το που έρθει ο παίκτης σε επαφή με το όπλο. Καλείται η *JackShepard::onCollision()* (όπου "JackShepard" είναι τύπου "PlayerData" και είναι το datablock του ήρωα) η οποία με τη σειρά της καλεί τη μέθοδο *collision()* του όπλου (όπως είπαμε τα αντικείμενα με τα οποία αλληλεπιδράμε είναι της κλάσης "Item" άρα καλείται η *Item::collision()*). Από εκεί ο έλεγχος μεταφέρεται στην *ShapeBase::pickup()* (όλα τα αντικείμενα που σχεδιάζονται στη σκηνή έχουν την "ShapeBase" ως γονέα-ανατρέξτε στο παράρτημα με το δένδρο κληρονομικότητας). Η μέθοδος αυτή χρησιμοποιεί το datablock του αντικειμένου με το οποίο ήρθαμε σε επαφή για να καλέσει την κατάλληλη μέθοδο *onPickup()*.

Εδώ μπαίνει ο πολυμορφισμός της γλώσσας. Το datablock του όπλου είναι το αντικείμενο "MP5" που είναι τύπου ItemData. Ο ορισμός του φαίνεται στο παρακάτω τμήμα κώδικα:

```
datablock ItemData(MP5)
{
    category = "Weapon";

  //Hook into Item Weapon class hierarchy. The weapon namespace
  //provides common weapon handling functions in addition to
  //hooks into the inventory system.
    className = "Weapon";

  //Basic Item properties
    shapeFile = "~/data/items/gunAK47.dts";
    mass = 1;
    elasticity = 0.2;
    friction = 0.6;
    emap = true;

    canPickup = 1;

    // Dynamic properties defined by the scripts
    pickUpName = "a machinegun";
    image = MP5Image;
    ammount=1;
};
```

Το πεδίο **"className"** προσθέτει το **namespace** "Weapon" μεταξύ "MP5" και "ItemData". Μπορούμε να το φανταστούμε σαν δημιουργία μιας νέας κατηγορίας όπου εντάσσουμε όλα τα όπλα του παιχνιδιού. Πλέον για κάθε μέθοδο που καλείται γίνεται έλεγχος αν ανήκει στο χώρο ονομάτων "MP5". Αν ναι τότε έχει καλώς. Αν όχι μεταφερόμαστε στο χώρο ονομάτων "Weapon". Αν ούτε εκεί αντιστοιχεί η μέθοδος, μεταφερόμαστε στο "ItemData" και πάει λέγοντας μέχρι να εντοπίσουμε τη συνάρτηση στην ιεραρχία των κλάσεων.

Επιστρέφοντας στο πρόγραμμα, δεν υπάρχει *MP5::onPickup()*, οπότε μεταφερόμαστε στην *Weapon::onPickup()* η οποία με τη σειρά της καλεί το γονέα, δηλαδή την *Item-*

Data::onPickup(). Εκεί γίνονται οι έλεγχοι για το αν ο παίκτης μπορεί να μαζέψει το αντικείμενο (και δεν υπάρχει ήδη κάποιο όπλο). Αν ναι, οι *incInventory()* και *setInventory()* μέθοδοι της "ShapeBase" κλάσης ενημερώνουν κατάλληλα το σύστημα καταγραφής και τις αντίστοιχες οθόνες του HUD.

Η ίδια ακολουθία από κλήσεις συμβαίνει και με κάθε αντικείμενο που μαζεύουμε. Φυσικά το πρόγραμμα φροντίζει κάθε φορά να καλέσει τις μεθόδους που ανήκουν στο ίδιο χώρο ονομάτων με το αντικείμενο (για παράδειγμα αν μαζέψουμε σφαίρες για το όπλο, τότε η *Weapon::onPickup()* δεν καλείται μιας και οι σφαίρες δεν ανήκουν σε αυτό το χώρο ονομάτων-απευθείας γίνεται κλήση της *ItemData::onPickup()*).

Σίγουρα η παραπάνω περιγραφή δεν είναι αρκετή για αυτό και ακολουθεί ένας πίνακας με τις συναρτήσεις που αναφέρθηκαν και τα αρχεία μέσα στα οποία δηλώνονται. Ο κώδικας των αρχείων αυτών συνοδεύεται με αρκετά σχόλια και γίνεται εύκολα κατανοητός:

Μέθοδος	Αρχείο
JackShepard::onCollision()	./control/server/players/player.cs
Item::collision()	./control/server/misc/item.cs
ShapeBase::pickup()	./control/server/misc/inventorySystem.cs
Weapon::onPickup()	./control/server/weapons/weapon.cs
ItemData::onPickup()	./control/server/misc/item.cs
ShapeBase::incInventory()	./control/server/misc/inventorySystem.cs
ShapeBase::setInventory()	./control/server/misc/inventorySystem.cs

Πίνακας 6.1: Κατάλογος συναρτήσεων για την υλοποίηση του συστήματος καταγραφής

6.4 Τεχνητή Νοημοσύνη (Artificial Intelligence - AI)

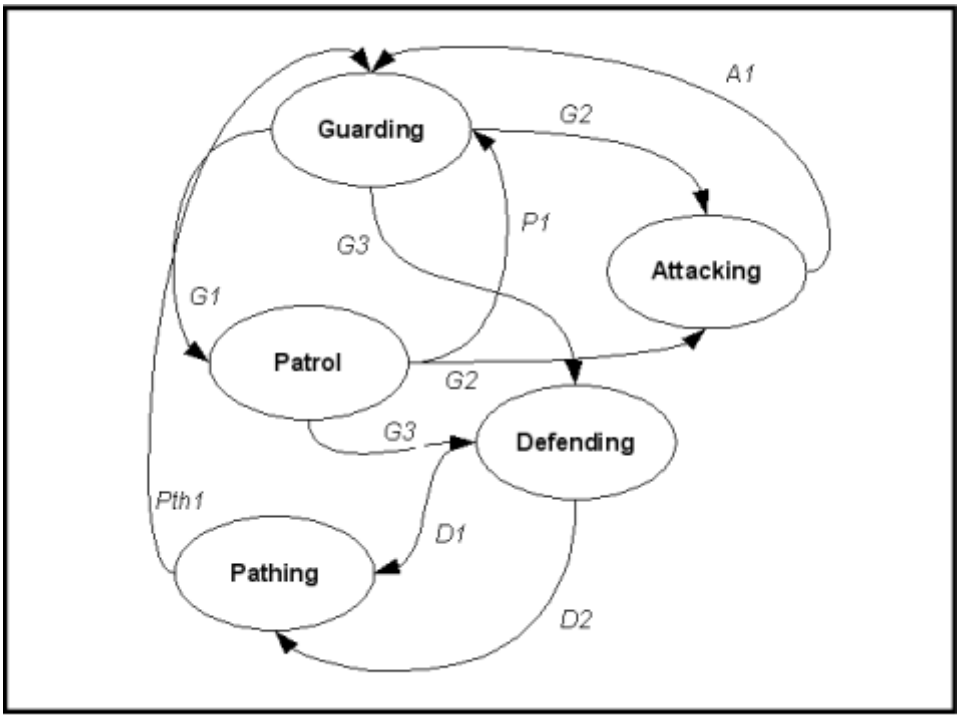
Ένα από τα σημαντικότερα "συστατικά" των ηλεκτρονικών παιχνιδιών είναι αυτό της τεχνητής νοημοσύνης. **Στα παιχνίδια, η τεχνική νοημοσύνη έχει να κάνει κυρίως με τον τρόπο που συμπεριφέρονται οι χαρακτήρες που δεν ελέγχονται από τον παίκτη (bots ή NPC)**.

Η Torque δεν έχει κάποιο σύστημα τεχνητής νοημοσύνης. Χρειάστηκε λοιπόν να προγραμματιστούν οι εχθροί έτσι ώστε να μπορούν να κινηθούν στο χώρο, να εντοπίζουν τον παίκτη, να επιτίθενται και γενικά να έχουν μια προσαρμοζόμενη συμπεριφορά για τις καταστάσεις που προκύπτουν κατά τη διάρκεια του παιχνιδιού. Ο κώδικας αυτός βρίσκεται στο αρχείο "./control/server/players/ai_new.cs".

6.4.1 Μηχανή Πεπερασμένων Καταστάσεων

Μία από τις πιο συνηθισμένες τεχνικές για τον προγραμματισμό της τεχνητής νοημοσύνης, στα ηλεκτρονικά παιχνίδια, είναι αυτή της **μηχανής πεπερασμένων καταστάσεων (FSM). Κάθε αντικείμενο που χειρίζεται ο υπολογιστής ξεκινάει με κάποια αρχική κατάσταση και ανάλογα με τα 'ερεθίσματα'' που δέχεται μεταβαίνει σε κάποια άλλη.**

Στην παρούσα εφαρμογή έχουμε πέντε δυνατές καταστάσεις στις οποίες μπορεί να βρεθεί το bot. Οι καταστάσεις αυτές φαίνονται στο παρακάτω σχήμα, μαζί με τις συνθήκες μετάβασης:



Σχήμα 6.14: Μηχανή πεπερασμένων καταστάσεων για τα bots

όπου,

- G1: έχει μεσολαβήσει τυχαίο χρονικό διάστημα,
- G2: έχει εντοπιστεί ο παίκτης,
- G3: έχει εντοπιστεί ο παίκτης και το bot έχει την ιδιότητα του defender,
- A1: ο στόχος έχει χαθεί και το bot έχει μεταβεί ήδη στην τελευταία θέση που τον είχε εντοπίσει,
- P1: έχει διανύσει τυχαία απόσταση,
- D1: έχει βρεθεί στην τελευταία θέση κάλυψης,

- D2: ο παίκτης δεν είναι ορατός στο bot για 3 δευτερόλεπτα,
- Pth1: το bot έχει χτυπήσει το συναγερμό.

Η μηχανή καταστάσεων ορίζεται μέσα στη μέθοδο *AIPlayerDB::Think()*. Κάθε φορά που ολοκληρώνεται ένας κύκλος, το bot "ξανασκέφτεται" μετά από κάποιο χρόνο. Ο χρόνος αυτός είναι μεταβλητός και εξαρτάται από το πόσο κοντά βρίσκεται ο παίκτης. Όσο πιο κοντά τόσο πιο γρήγορη αντίδραση πρέπει να έχουμε (ελάχιστος χρόνος μεταξύ διαδοχικών κλήσεων της "Think()" είναι τα 500ms). Σε αντίθετη περίπτωση ο χρόνος απόκρισης αυξάνεται. Με τον τρόπο αυτό εξοικονομούμε κύκλους του επεξεργαστή σε περίπτωση που ο παίκτης βρίσκεται σε μεγάλη ακτίνα από τον εχθρό.

Guarding

Είναι η κατάσταση στην οποία ξεκινάει το bot. Κάθε bot τοποθετείται σε συγκεκριμένη θέση μέσα στη σκηνή. Η θέση αυτή είναι η "σκοπιά" για το εκάστοτε bot. Έχουν οριστεί παράμετροι που καθορίζουν τη μέγιστη απόσταση στην οποία μπορούν να δουν και τη μέγιστη γωνία θέασης. Ανά τυχαία διαστήματα αλλάζουν διεύθυνση και παρατηρούν προς άλλο σημείο του ορίζοντα.

Attacking

Τα bots έχουν εντοπίσει τον παίκτη και κατευθύνονται προς το μέρος του. Η κίνηση γίνεται σε ευθεία γραμμή. Παράλληλα ειδοποιούν όλους τους γείτονές τους για τον εχθρό. Όλες οι "αισθήσεις" αυξάνονται (η γωνία θέασης αλλάζει στις 360 μοίρες ενώ ο "χρόνος σκέψης" είναι ο ελάχιστος των 500ms). Έχει καθοριστεί μία ελάχιστη απόσταση στην οποία πρέπει να βρεθούν από τον στόχο τους πριν αρχίσουν να πυροβολούν. Κάθε φορά που ο στόχος χάνεται προσπαθούν να μεταβούν στο τελευταίο σημείο που αυτός εθεάθη.

Patrol

Το bot αφήνει τη θέση που βρισκότανε στην κατάσταση "Guarding". Αν αυτή ήτανε η αρχική θέση (σκοπιά), κατευθύνεται κάποια μέτρα προς τη διεύθυνση που κοιτούσε στο τέλος της κατάστασης "Guarding". Σε αντίθετη περίπτωση προσπαθεί να επιστρέψει στην αρχική θέση φύλαξης.

Defending

Σε αυτή την κατάσταση μπορούν να μπουν μόνο οι "defenders". Έτσι ονομάζονται τα bots που στόχος τους είναι να σημάνουν συναγερμό με το που εντοπίσουν τον παίκτη. Στην κατάσταση αυτή τα bots πυροβολούν, αν ο εχθρός είναι στο οπτικό τους πεδίο και σε κοντινή απόσταση, αλλά δεν τον κυνηγούν! Αντιθέτως μεταβαίνουν σε τυχαίες θέσεις με στόχο να καλυφθούν.

Pathing

Όπως και πριν μόνο οι "defenders" μπορούν να βρεθούν σε αυτήν την κατάσταση. Αν ο παίκτης είναι στο οπτικό τους πεδίο πυροβολούν, ενώ παράλληλα κατευθύνονται προς

τη θέση που βρίσκεται ο συναγερμός. Για την κίνηση εδώ χρησιμοποιήθηκε η μέθοδος **"path-following", δηλαδή η κίνηση πάνω σε καθορισμένα μονοπάτια. Τα μονοπάτια αυτά δεν είναι τίποτα παραπάνω από ένα σύνολο σημείων/κόμβων πάνω στο έδαφος της αποστολής και δηλώνονται στο αρχείο της αποστολής. Προγραμματιστικά καθορίζουμε με ποια σειρά θα προσπελάσουν οι "defenders" τους κόμβους.** Φτάνοντας στο τέλος του μονοπατιού, ενεργοποιούν το συναγερμό.

6.4.2 Ενέργειες των bots

Αυτό που πρέπει να λαμβάνουμε υπόψιν κάθε φορά που προγραμματίζουμε το AI είναι να μην παρέχουμε "κρυφή" πληροφορία στα bots. Για παράδειγμα, αν ο παίκτης βρεθεί σε θέση από την οποία δεν είναι ορατός στο bot τότε, στο τελευταίο, δεν είναι σωστή τακτική να του "δώσουμε" τις συντεταγμένες του παίκτη, έτσι ώστε αυτό να μπορέσει να τον εντοπίσει.

Εντοπισμός

Ο εντοπισμός του παίκτη από το bot γίνεται σε τρεις φάσεις. Αρχικά ελέγχεται η μεταξύ τους απόσταση. Σε κάθε bot υπάρχει μεταβλητή που καθορίζει τη μέγιστη απόσταση στην οποία μπορεί να βλέπει.

Αν ο παίκτης είναι μέσα στην ακτίνα του bot τότε γίνεται έλεγχος για τη γωνία που βρίσκεται σε σχέση με αυτό. **Η γωνία θέασης (FOV)** για τον χαρακτήρα που ελέγχεται από τον υπολογιστή αρχικοποιείται στις 140 μοίρες, εκτός και αν ο παίκτης βρεθεί υπερβολικά κοντά. Στη τελευταία αυτή περίπτωση η εν λόγω γωνία παίρνει την τιμή των 360 μοιρών (σε όποια γωνία δηλαδή και να βρίσκεται ο παίκτης θα εντοπιστεί άμεσα). Δίνουμε έτσι την αίσθηση ότι το bot "άκουσε" τον ήρωα, με ότι αυτό συνεπάγεται.

Ο υπολογισμός της γωνίας στην οποία βρίσκεται ο παίκτης σε σχέση με το bot, επιτυγχάνεται με τις μεθόδους *getBearingAngle()* και *getRelativeAngle()* της κλάσης "AIPlayer":

```
//*****************************************
//* function AIPlayer::getBearingAngle(%bot)
//*Args:
//* (1)%bot: the AI player
//*Returns:
//* the bearing angle (angle between
//*north direction and facing vector)
//*Purpose:
//* Where the bot is facing
//*****************************************
function AIPlayer::getBearingAngle(%bot)
{
   %bearingAngle=getWord(%bot.rotation,3);
   if(getWord(%bot.rotation,2)$="-1")
      %bearingAngle=360-%bearingAngle;
   return %bearingAngle;
}
```

```
//*********************************************
//* function AIPlayer::getReativeBearingAngle(%bot,%enemy)
//*Args:
//* (1)%bot: the AI player
//* (2)%enemy: a player's avatar
//*Returns:
//* bearing angle of the enemy from the bot (we use
//*the bot's position as the zero point (0,0))
//*Purpose:
//* Use the bot's position as the zero point (0,0)
//*and check where is the player (angle). Calculate
//*bearing angle that is depended on the quadrant
//*the enemy is.
//*********************************************
function AIPlayer::getRelativeBearingAngle(%bot,%enemy)
{
   //assume that bot and enemy are at the same level z
   %xBot=getWord(%bot.getPosition(),0);
   %yBot=getWord(%bot.getPosition(),1);
   %xEnemy=getWord(%enemy.getPosition(),0);
   %yEnemy=getWord(%enemy.getPosition(),1);

   %x=%xEnemy-%xBot;
   %y=%yEnemy-%yBot;

   if(%x!=0)
   {
      %slope=%y/%x;
      // We use bot's x,y position as the start point
      //so we get 4 quadrants. %angle is the angle between
      //x+ vector or x- vector (start) and the enemy vector
      //(finish) and is -90degrees<%angle<90degrees.
      %angle=mRadToDeg(mATan(%slope,1));

      // We dont need the sign just the absolute value
      %angle=mAbs(%angle);
   }
   else
      %angle=90;  //enemy on y-axis

   // Calculate the bearing angle--it is depended on
   //the quadrant where the enemy is
   if ( (%x>=0) && (%y>=0) )  //enemy in quadrant 1, 0-90degs
      %bearingAngle=90-%angle;
   else if ( (%x>=0) && (%y<0) ) //enemy in quadrant 2,
      90.001-180degs
      %bearingAngle=90+%angle;
   else if ( (%x<0) && (%y<0) )  //enemy in quadrant 3,
      180.001-269.999degs
      %bearingAngle=270-%angle;
   else  //enemy in quadrant 4, 270-359.999degs
      %bearingAngle=270+%angle;
```

```
    return %bearingAngle;
}
```

Η πρώτη υπολογίζει τη γωνία που σχηματίζει το "eyeVector" του bot με το γεωγραφικό Βορρά. Η δεύτερη παίρνει ως σημείο αναφοράς τη θέση του bot και βρίσκει τη γωνία που βρίσκεται ο παίκτης. Η γωνία αυτή εξαρτάται από το τεταρτημόριο μέσα στο οποίο είναι ο τελευταίος.

Τα αποτελέσματα αυτών των μεθόδων επεξεργάζονται από τη μέθοδο *checkArcOfSight()* της "AIPlayer" έτσι ώστε να δούμε αν ο παίκτης βρίσκεται μέσα στο οπτικό πεδίο του bot:

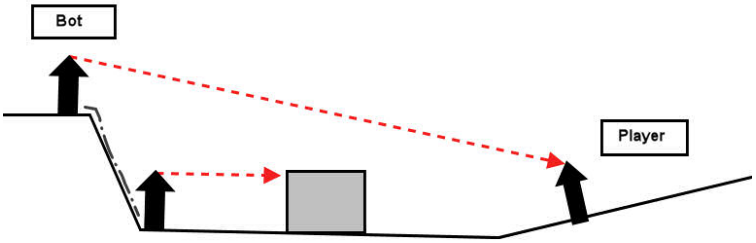
```
function AIPlayer::checkArcOfSight(%bot,%enemy)
{
   %botBearingAngle=%bot.getBearingAngle();
   %arcOfSightRight=%botBearingAngle+%bot.fov/2;
   %arcOfSightLeft=%botBearingAngle-%bot.fov/2;
   %relativeAngle=%bot.getRelativeBearingAngle(%enemy);

   if ( (%arcOfSightLeft<=%relativeAngle) && (%relativeAngle<=%
      arcOfSightRight) )
      %spot=true;
   else
   {
      if (%arcOfSightRight>=360)
      {
         %arcOfSightRight=%arcOfSightRight-360;
         if( (0<=%relativeAngle) && (%relativeAngle<=%
            arcOfSightRight) )
            %spot=true;
         else
            %spot=false;
      }
      else if (%arcOfSightLeft<0)
      {
         %arcOfSightLeft=%arcOfSightLeft+360;
         if( (%arcOfSightLeft<=%relativeAngle) && (%
            relativeAngle<=360) )
            %spot=true;
         else
            %spot=false;
      }
      else
         %spot=false;
   }
   return %spot;
}
```

Ο τρίτος και τελευταίος έλεγχος που γίνεται, αν και ο προηγούμενος ήτανε αληθής, είναι για το λεγόμενο **"line of sight" (LOS)**. Μια ακτίνα ξεκινάει από το ύψος που βρίσκονται τα μάτια του μοντέλου του bot και κατευθύνεται προς τη θέση του παίκτη **(raycasting)**. Αν δεν υπάρχει εμπόδιο (για παράδειγμα ο παίκτης δεν βρίσκεται πίσω

από κάποιο κτήριο), τότε και αυτός ο έλεγχος είναι αληθής και ο παίκτης έχει εντοπιστεί!

Μετάβαση σε καλύτερη θέση

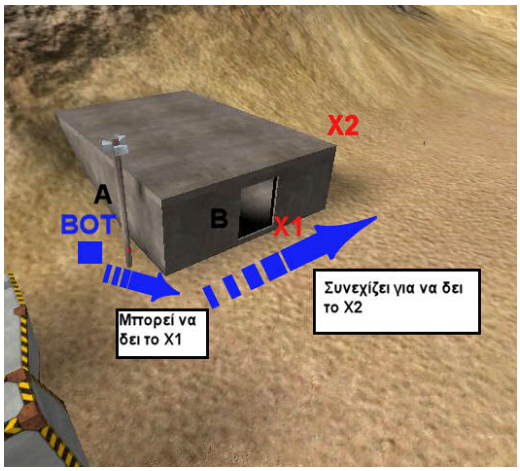
Όπως είδαμε νωρίτερα, στην κατάσταση "Attacking" το bot προσπαθεί να πλησιάσει τον παίκτη, κινούμενο σε ευθεία πορεία. Ας δούμε το παρακάτω σχήμα:



Σχήμα 6.15: Κίνηση του bot προς τον παίκτη

Αρχικά η θέση του παίκτη είναι στο LOS του bot. Καθώς όμως το bot κινείται, το κτήριο μπαίνει στο LOS με αποτέλεσμα ο παίκτης να μην είναι πλέον ορατός. Στην περίπτωση αυτή η κίνηση διακόπτεται και το bot προσπαθεί να μετακινηθεί σε καλύτερο σημείο, όπου θα μπορεί να δει και πάλι τη θέση του παίκτη. Ίσως ο ήρωας να έχει φύγει από εκεί. Ο εχθρός "θυμάται" το τελευταίο σημείο που τον είχε εντοπίσει!

Ποιά είναι όμως η καλύτερη θέση που μπορεί να μεταβεί; Δείτε το σχήμα που ακολουθεί:



Σχήμα 6.16: Μετάβαση του bot σε καλύτερη θέση

Αν το bot σταμάτησε την κίνηση του, για παράδειγμα, στην πλευρά A και η θέση του παίκτη (X1) είναι στην B (πλαϊνή της A), τότε μεταβαίνει κοντά στη γωνία που σχηματίζουν αυτές οι δύο. Αυτή είναι η απλή περίπτωση. Αν συμβεί το bot και η τελευταία γνωστή θέση του παίκτη να είναι σε αντίθετες πλευρές (X2), τότε ελέγχεται πρώτα

ποια πλαϊνή πλευρά του εμποδίου είναι πιο κοντά στον παίκτη και εκεί κατευθύνεται το bot. Πλέον είμαστε στην πρώτη περίπτωση και το bot ενεργεί όπως περιγράψαμε.

Εμπόδια κατά την κίνηση

Κατά την κίνηση του το bot μπορεί να συναντήσει κάποιο εμπόδιο και να κολλήσει. Πρώτα από όλα πρέπει να αναπτύξουμε ένα μηχανισμό ώστε να ξέρουμε ότι το bot κόλλησε. Το τελευταίο πράγμα που θέλουμε να δούμε στην εφαρμογή μας είναι ένας χαρακτήρας που προσπαθεί, με τις ώρες, να περάσει μέσα από τον τοίχο ενός κτηρίου!

Η λογική είναι ιδιαίτερα απλή και στηρίζεται στην απόσταση που διανύει το bot. Μία μεταβλητή χρησιμοποιείται ως σημαία για να δηλώσει ότι το bot κινείται (περίπτωση περιπολίας για παράδειγμα). Κάθε φορά που μπαίνουμε στην *Think()* γίνεται έλεγχος της θέσης που εκείνη τη στιγμή βρίσκεται το bot και της θέσης που βρισκότανε στο προηγούμενο πέρασμα. Με βάση αυτές τις δύο, υπολογίζεται η απόσταση που έχει διανύσει. Αν είναι πολύ μικρή, δηλαδή μικρότερη από 1wu, θεωρούμε ότι έχει κολλήσει.

Σε αυτήν την περίπτωση δίνεται μία μικρή ώθηση στον Z άξονα του bot ώστε να πραγματοποιήσει άλμα. Αν στο επόμενο κύκλο της *Think()* η απόσταση μεταβληθεί, συνεχίζει κανονικά την πορεία του. Αν όχι, σταματάμε το bot σε εκείνο το σημείο και το βάζουμε σε κατάσταση "Guarding". Όταν ξαναμπεί σε περιπολία, υπάρχει πιθανότητα να κινηθεί σε αντίθετη κατεύθυνση από το εμπόδιο.

6.5 Συνοψίζοντας

Στο κεφάλαιο που μόλις ολοκληρώθηκε περιγράφηκε από πλευράς κώδικα πλέον, η υλοποίηση του παιχνιδιού "WaterWar". Σταθήκαμε κυρίως σε θέματα που έχουν να κάνουν με τα γραφικά περιβάλλοντα διασύνδεσης και με την τεχνητή νοημοσύνη. Είναι αδύνατο να περιγράψουμε αναλυτικά όλα τα τμήματα κώδικα.

Εκτός λοιπόν των παραπάνω, έχουν εφαρμοστεί όλες οι τεχνικές που χρησιμοποιούνται στην ανάπτυξη βιντεοπαιχνιδιών με τη χρήση κάποιας μηχανής. Έτσι έχουν προγραμματιστεί τα απαραίτητα **'triggers", που κατά κύριο λόγο είναι περιοχές μέσα στη σκηνή, στις οποίες όταν βρεθεί ο παίκτης, συμβαίνει "αυτόματα" κάποιο γεγονός** (για παράδειγμα, ακούγεται ένα ηχητικό μήνυμα). Υπάρχει επίσης ο κώδικας που φροντίζει για τα εφέ όπως ο καπνός και οι σπίθες. Σε όλες σχεδόν τις μηχανές παιχνιδιών αυτό επιτυγχάνεται με τις οντότητες **particles** και **emitters**. Οι πρώτες είναι στην ουσία τα εικονίδια που χρησιμοποιούνται για τις σπίθες, για παράδειγμα, ενώ οι δεύτερες είναι εκείνες που καθορίζουν το σημείο από το οποίο εκπέμπονται τα particles, τη συχνότητα εκπομπής, τη διάρκεια ζωής κτλ. Φυσικά σε ένα παιχνίδι δεν θα μπορούσε να λείπουν και οι ήχοι. Η υλοποίησή τους δεν είναι κάτι το ιδιαίτερο μιας και η μηχανή μας παρέχει έτοιμες μεθόδους για αυτό.

Κεφάλαιο 7

Game Over

Μετά από τρεις μήνες συγκέντρωσης του υλικού, μελέτης και προετοιμασίας των μοντέλων και εννιά μήνες σχεδίασης και προγραμματισμού του παιχνιδιού, η διπλωματική εργασία έφτασε στο τέλος της. Αυτό που υλοποιήθηκε είναι ένα τρισδιάστατο παιχνίδι, κατηγορίας FPS, ενός και μόνο παίκτη, χρησιμοποιώντας τη μηχανή "Torque Game Engine".

7.1 Συνοπτικά το WaterWar

Το "WaterWar" είναι ένα παιχνίδι πρώτου προσώπου βολής για ηλεκτρονικό υπολογιστή (περιβάλλον Windows). Ο παίκτης αναλαμβάνει το ρόλο ενός στρατιώτη όπου περιπλανιέται κάπου στην αφρικάνικη ήπειρο με στόχο να βρει το πολυτιμότερο αγαθό που έχει απομείνει: το νερό! Ένοπλη ομάδα ανδρών έχει κρύψει μέσα σε στρατιωτική περιοχή ένα μπουκάλι με πόσιμο νερό. Για να το αποκτήσει ο παίκτης θα πρέπει να πολεμήσει εναντίον τους αλλά και να συγκεντρώσει πληροφορίες που θα τον οδηγήσουν στο στόχο του. Οι πληροφορίες αυτές συλλέγονται είναι με επαφή με κάποιο αντικείμενο (π.χ. δένδρο), είτε με συνομιλία με άλλους χαρακτήρες αλλά ακόμα και με "χάκινγκ" υπολογιστών!

Για την πλοήγηση στον κόσμο του παιχνιδιού απαιτούνται πληκτρολόγιο και ποντίκι:

- Κίνηση:
  - κουμπί "W": κίνηση προς τα εμπρός,
  - κουμπί "S": κίνηση προς τα πίσω,
  - κουμπί "A": πλάγια βήματα αριστερά,
  - κουμπί "D": πλάγια βήματα δεξιά,
  - κουμπί "space": άλμα,
  - ποντίκι: αλλαγή διεύθυνσης παίκτη,
- Ενέργειες:
  - αριστερό κουμπί ποντικιού: πυροδότηση όπλου,
  - κουμπί "F": αλληλεπίδραση με αντικείμενα (π.χ. χρήση κανονιού) ή άλλους χαρακτήρες,

  - κουμπί "R": χρήση πακέτου ενέργειας,
  - κουμπί "M": χρήση μάσκας αερίων,

- Άλλα:

  - κουμπί "tab": εναλλαγή από κάμερα τρίτου προσώπου σε κάμερα πρώτου,
  - κουμπί "tilde": εμφάνιση/απόκρυψη πυξίδας,
  - κουμπί "Alt + Enter": πλήρης οθόνη.

Για να ξεκινήσει το παιχνίδι το μόνο που χρειάζεται είναι η αντιγραφή του φακέλου "WaterWar" στο σκληρό δίσκο του υπολογιστή και η εκτέλεση του "torqueDemo.exe". Η εφαρμογή τερματίζει όταν ο παίκτης φτάσει στο στόχο ή με το κουμπί "Esc".

7.2 Δοκιμή του παιχνιδιού

Με την ολοκλήρωσή του το παιχνίδι διανεμήθηκε σε τρεις φοιτητές του Πολυτεχνείου Κρήτης για τις απαραίτητες δοκιμές. Αφού διορθώθηκαν σφάλματα στον κώδικα που προκαλούσαν βίαιο τερματισμό της εφαρμογής, έγιναν αλλαγές στη διεπαφή του παιχνιδιού με το χρήστη (μέσω πληκτρολογίου).

Και οι τρεις δοκιμαστές παραπονέθηκαν για το επίπεδο δυσκολίας. Αρχικά οι εχθροί του παιχνιδιού ήτανε ιδιαίτερα δύσκολο να καμφθούν. Αυτό είχε ως αποτέλεσμα ο παίκτης να κουράζεται, να ξεκινάει συνεχώς το παιχνίδι από το μοναδικό σημείο επανένταξής του και να χάνει το ενδιαφέρον του και συνεπώς την προσπάθεια τερματισμού του παιχνιδιού.

Αυτό οδήγησε πρώτα από όλα στον ορισμό τριών σημείων επανατοποθέτησης του ήρωα μετά από τον θάνατό του. Ανάλογα με τη θέση που η ενέργεια του μηδενίστηκε, καθορίστηκε και ένα σχετικά κοντινό σε αυτή σημείο, με αποτέλεσμα ο παίκτης να μην ξεκινάει κάθε φορά από το πρωταρχικό σημείο εισόδου. Επίσης έγινε αλλαγή στο κουμπί με το οποίο ο παίκτης ανακτά μέρος της ενέργειάς του. Πριν τις δοκιμές χρησιμοποιούταν το κουμπί "H". Παρατηρήθηκε ότι σε ένα συνηθισμένο πληκτρολόγιο, το κουμπί "R" είναι πιο λειτουργικό καθώς δεν χρειάζεται ο χρήστης να αφήνει τα δάκτυλα του από τα κουμπιά κινήσεως και μπορεί να κάνει χρήση πακέτων ενέργειας ακόμα και όταν δέχεται επίθεση. Τέλος, προγραμματιστικά, μειώθηκε η ζημιά που προκαλούν οι σφαίρες των εχθρών στον ήρωα και αντιθέτως αυξήθηκε η "βλάβη" στους εχθρούς από τα πυρομαχικά του παίκτη.

Κάτι άλλο το οποίο απουσίαζε και θεωρήθηκε χρήσιμο ήταν η πυξίδα. Κατά τη συνομιλία με τον φυλακισμένο δίνονται πληροφορίες σχετικές με τον γεωγραφικό βορρά. Χρειάστηκε λοιπόν οι πληροφορίες αυτές να μπορούν να αξιοποιηθούν με κάποιο τρόπο οπότε και υλοποιήθηκε η πυξίδα της ενότητας 6.2.5.

Τέλος κατά την πλοήγηση στον τρισδιάστατο χώρο και ενώ δεν εξελισσόταν κάποια μάχη, δεν υπήρχε το παραμικρό δείγμα ήχου. Έτσι προστέθηκε ο ήχος από τα βήματα του ήρωα κάνοντας πιο ενδιαφέρον τη μετάβαση προς το επόμενο σημείο μάχης.

7.3 Μελλοντικές προεκτάσεις και βελτιώσεις

Αρχικά υπήρχε ο στόχος το παιχνίδι να είναι δικτυακό, να μπορούν δηλαδή να συνδέονται παραπάνω του ενός παίκτες και να πολεμούν είτε μεταξύ τους, είτε από κοινού με τον υπολογιστή. Στην πορεία κάτι τέτοιο δεν κατέστη δυνατό χρονικά.

Η σημαντικότερη λοιπόν προσθήκη που μπορεί να γίνει είναι αυτή της δικτύωσης. Το παιχνίδι στηρίζεται στην αρχιτεκτονική πελάτη-εξυπηρετητή οπότε υπάρχει η εν λόγω δυνατότητα χωρίς να απαιτούνται μεγάλες αλλαγές. Ενδεικτικά να αναφερθεί ότι αυτό που πρέπει να προγραμματιστεί είναι ο λεγόμενος "master server". Η δουλειά αυτού είναι να κρατάει λίστα με όλους τους ενεργούς εξυπηρέτες και να προσφέρει στους πελάτες την απαραίτητη πληροφορία για να συνδεθούν με κάποιον από αυτούς.

Απαιτείται επίσης δημιουργία νέων διεπαφών για τη σύνδεση με κάποιον εξυπηρετητή, ο διαχωρισμός των τμημάτων κώδικα του πελάτη και του εξυπηρέτη (κάτι που υπάρχει ήδη απλά να μεταφερθούν στα κατάλληλα τερματικά) αλλά και η απόφαση για το αν πρέπει να σχεδιαστεί μία νέα αποστολή που αποκλειστικά θα παίζεται από τους διαδικτυακούς παίκτες.

Από εκεί και πέρα ο καθένας μπορεί να σκεφτεί ένα σωρό διαφορετικές προσθήκες. Κάποιος μπορεί μελλοντικά να προσθέσει δυνατότητα επιλογής όπλων στο ήδη υπάρχον παιχνίδι ή τη δυνατότητα για χρήση κάποιου οχήματος. Αυτό το τελευταίο απαιτεί φυσική κίνησης κάτι το οποίο μας παρέχει η μηχανή Torque. Μπορούν να προστεθούν νέες αποστολές, εχθροί με επιπλέον κινήσεις και όπλα και με πιο ανεπτυγμένη τεχνητή νοημοσύνη.

Στο θέμα της τεχνητής νοημοσύνης, για παράδειγμα, θα μπορούσε να υλοποιηθεί ο A* αλγόριθμος. Κάτι τέτοιο, στο συγκεκριμένο παιχνίδι και με την συγκεκριμένη αποστολή, θα είχε μόνο ερευνητικό σκοπό και δεν θα πρόσφερε κάτι παραπάνω. Ο εν λόγω αλγόριθμος εφαρμόζεται κυρίως σε περιοχές με πολλά εμπόδια και όχι σε ανοικτά εδάφη όπως αυτό που υπάρχει στην αποστολή. Για αυτό και προτιμήθηκε η κίνηση σε ευθεία!

Αυτό δεν σημαίνει ότι δεν θα μπορούσαμε να έχουμε ένα παιχνίδι με πιο έξυπνους εχθρούς όπου θα προσπαθούν να καλυφθούν από τα πυρά του παίκτη, θα ψάχνουν να τον εντοπίσουν όταν χάνεται από το οπτικό τους πεδίο και θα χρησιμοποιούν στρατιωτικές τακτικές επίθεσης. Γενικά η τεχνητή νοημοσύνη είναι ένας από τους λιγότερο εξελιγμένους τομείς στα βιντεοπαιχνίδια. Η αληθοφάνεια και η ποικιλία στη συμπεριφορά και στη δράση του κόσμου του παιχνιδιού είναι χαρακτηριστικά που αυξάνουν την ψυχαγωγική του αξία. Καθώς η υπολογιστική ισχύ των παιχνιδιών αυξάνει ραγδαία, η τεχνητή νοημοσύνη είναι το συστατικό των παιχνιδιών που θα γνωρίσει τη μεγαλύτερη εξέλιξη τα επόμενα χρόνια.

7.4 Επίλογος

Η εργασία ξεκίνησε με πληροφορίες για τη βιομηχανία των βιντεοπαιχνιδιών. Ως επίλογος θα παρουσιαστούν κάποιες πληροφορίες για το τί συμβαίνει στον ελληνικό χώρο.

Το πρόβλημα που υπάρχει στην Ελλάδα, όπως και σε άλλες χώρες, όσον αφορά τα βιντεοπαιχνίδια δεν είναι αυτό της διάδοσης και του ποσοστού χρήσης, αλλά της αποδοχής τους από το ευρύ κοινό ως μέσο ψυχαγωγίας. Αν και ένα πολύ μεγάλο ποσοστό ενηλίκων παίζουν ηλεκτρονικά παιχνίδια (σε όλες τις πλατφόρμες), επικρατεί

για αυτά η κοινή στερεότυπη άποψη περί εθιστικής ασχολίας των παιδιών. Επιπλέον αν και υπάρχουν αρκετές ομάδες ερασιτεχνών που αναπτύσσουν παιχνίδια στην Ελλάδα, καθώς και λίγες επαγγελματικές, δεν ήταν δυνατό προς το παρόν να δημιουργηθεί μια βιώσιμη βιομηχανία ανάπτυξης.

Αργά αλλά σταθερά τα πράγματα αρχίζουν να αλλάζουν. Τα βιντεοπαιχνίδια εντάσσονται ως μάθημα στο πρόγραμμα ελληνικών πανεπιστημίων. Επίσης, στα πανεπιστήμια διεξάγεται έρευνα σχετικά με τα βιντεοπαιχνίδια και τις χρήσεις τους. Έχει συσταθεί Ελληνικός Σύλλογος Δημιουργών Λογισμικού Ψυχαγωγίας. Δημοσιογράφοι με γνώση και όχι προκατάληψη ενάντια στα παιχνίδια, αρχίζουν και αρθρογραφούν σε εφημερίδες.

Υπάρχει πολύ δρόμος μπροστά, αλλά η κατάσταση μπορεί να γίνει μόνο καλύτερη στο μέλλον.

Παράρτημα Α΄

Προγραμματισμός σε Matlab του "TerrainGenerator"

Το "Matlab" είναι ένα μαθηματικό πακέτο που παρέχει ένα εύχρηστο περιβάλλον για υλοποίηση επιστημονικών εφαρμογών σε ένα μεγάλο φάσμα πεδίων, όπως στη γραμμική άλγεβρα, στατιστική, εφαρμοσμένα μαθηματικά, επεξεργασία σημάτων, θεωρία ελέγχου. Υποστηρίζει ένα μεγάλο αριθμό λειτουργιών και συναρτήσεων καθώς και εξωτερικές βιβλιοθήκες για εξειδικευμένες περιοχές εφαρμογών. Εκτελεί από απλούς μαθηματικούς υπολογισμούς μέχρι και προγράμματα με εντολές παρόμοιες με αυτές που υποστηρίζει μια γλώσσα υψηλού επιπέδου. Συγκεκριμένα εκτελεί απλές μαθηματικές πράξεις, αλλά εξίσου εύκολα χειρίζεται μιγαδικούς αριθμούς, δυνάμεις, ειδικές μαθηματικές συναρτήσεις, πίνακες, διανύσματα και πολυώνυμα. Μπορεί επίσης να αποθηκεύει και να ανακαλεί δεδομένα, να δημιουργεί και να εκτελεί ακολουθίες εντολών που αυτοματοποιούν διάφορους υπολογισμούς και να σχεδιάζει γραφικά. Κατά κύριο λόγο η χρήση του "Matlab" γίνεται με την παραγωγή αρχείων με κατάληξη ".m" όπου περιέχουν κώδικα γραμμένο σε μια ευέλικτη, απλή και δομημένη γλώσσα προγραμματισμού (script) που μοιάζει με τη γλώσσα Pascal.

Ποιος ο λόγος όμως που χρησιμοποιήσαμε το εν λόγω πακέτο στην υλοποίηση της εφαρμογής μας; Στο βασικό κείμενο αναφέραμε ότι η εφαρμογή "TerrainGenerator" δημιουργεί και "σώζει" ως εικόνα με κατάληξη ".png", έναν υψομετρικό χάρτη. Με το "Matlab" μία και μόνο εντολή είναι αρκετή για να δημιουργήσουμε ισοϋψείς καμπύλες από έναν NxN πίνακα, ενώ η δημιουργία εικόνων οποιουδήποτε τύπου και η αποθήκευση τους στο δίσκο, γίνεται εξίσου εύκολα και γρήγορα.

Επιγραμματικά για τον κώδικα σε "Matlab" που παρουσιάζεται παρακάτω αναφέρουμε:

- Συναρτήσεις **terrainGeneratorMain** και **createInputBoxes**: υλοποιούν το απλό, γραφικό περιβάλλον του εργαλείου. Πιο συγκεκριμένα είναι υπεύθυνες για το βασικό παράθυρο στο οποίο εισάγει ο χρήστης τα δεδομένα.

- Συνάρτηση **terrainGenerator**: υπολογίζει τον πίνακα επιπέδων (elevation table) και γεμίζει με τυχαίες τιμές (καλεί πολλές φορές την randomGenerator) έναν 32x32 πίνακα. Με βάση τα δεδομένα του τελευταίου θα σχεδιαστεί, παρακάτω, ο υψομετρικός χάρτης.

- Συνάρτηση **randomGenerator**: επιστρέφει έναν ακέραιο αριθμό επιλέγοντας τυχαία από ένα πλήθος ακεραίων που της δίνονται ως όρισμα.
- Συνάρτηση επιστροφής (callback) **okButton**: "συνδέεται" με το πλήκτρο "OK" του γραφικού περιβάλλοντος και αποθηκεύει τα δεδομένα εισόδου των πεδίων "#Hills" και "#Elevations" όταν το πλήκτρο πατηθεί.
- Συνάρτηση επιστροφής (callback) **generateButton**: "συνδέεται" με το πλήκτρο "Generate" του γραφικού περιβάλλοντος και όταν το πλήκτρο "ενεργοποιηθεί":
  1. καλείται η συνάρτηση terrainGenerator και
  2. εμφανίζεται σε νέο παράθυρο ο υψομετρικός χάρτης που αντιστοιχεί στα δεδομένα εισόδου.
- Συνάρτηση επιστροφής (callback) **printButton**: "συνδέεται" με το πλήκτρο "Print", το οποίο βρίσκεται στο παράθυρο που εμφανίζεται ο υψομετρικός χάρτης. Όταν το πλήκτρο πατηθεί η συνάρτηση επιστροφής:
  1. αποθηκεύει προσωρινά στο δίσκο τον υψομετρικό χάρτη, ως μία 256x256 εικόνα τύπου ".png",
  2. δημιουργεί ένα γκαουσιανό φίλτρο (gaussian filter),
  3. διαβάζει την εικόνα από το δίσκο και εφαρμόζει το φίλτρο σε αυτήν,
  4. αποθηκεύει τη νέα εκδοχή της εικόνας.

```
1  function terrainGeneratorMain
2      close all;
3      clear all;
4      mainWindowPos=[512 650 650 650];
5      mainWindow = figure('position',mainWindowPos,'Toolbar','
           auto','NumberTitle','off','Name','TerrainGenerator Main
            Window-Beta');
6      h(1).elevHillsText=uicontrol('style','text','position', [5
           620 60 25],'string',{'Hills' '#'});
7      h(1).elevHillsInput = uicontrol('style','edit','position',
           [70 620 40 25]);
8      h(2).elevHillsText=uicontrol('style','text','position', [5
           590 60 25],'string',{'Elevations' '#'});
9      h(2).elevHillsInput = uicontrol('style','edit','position',
           [70 590 40 25]);
10     buttonOne = uicontrol('style', 'pushbutton','position', [5
           540 105 40] , 'string'  , 'OK');
11     set(buttonOne, 'callback', {@okButton, h});
12     frameOne = uicontrol('style', 'frame','position', [5 5 300
           500]);
13     frameText=uicontrol('style','text','position', [10 10 280
           480],'string',{'NOTES' ' ' '(+) Lets say we want to
           create one hill (so Hills#=1) with maximun height at 8
           meters. If we set 5 as Elevations# then we will have
           level0=0m, level1=2m, level2=4m, level3=6m, level4=8m.
           The program will assign a greyscale color to each level
           . '...
```

```
14          '' '(+) All the hills are going to have the same #
                elevations.'...
15          '' '(+) Press OK to get input boxes for the hills data.
                '...
16          '' '(+) Consider that we are sketching on a canvas with
                 a 32x32 grid pattern (32x32 pixels). We start from
                 a (x,y) pixel and we can move only right (on x-
                axes) and up (on y-axes).'...
17          '' '(+) Example of input for hills data: x=5, y=12,
                max_x=4, max_y=5, height(max)10 -> On x-axes of the
                 canvas we draw on pixels 5,6,7 and 8 (max_x=4
                means use 4 pixels starting from 5). On Y-axes of
                the canvas we draw on pixels 12,13,14,15 and 16 (
                max_y=5 means use 5 pixels starting from 12).'...
18          '' '(+) Try to stay inside the 32x32 limits of canvas.'
                ...
19          '' '(+) Press Generate to view your art.'...
20          '' '(+) Press Print to save what you have drawn. You
                are going to get a 256x256 blurred png image (check
                 work folder)'...
21          '' '(+) Anytime you can go back to the main window and
                change the hills data.'});
22  %end terrainGeneratorMain
23
24  %*********************************
25  %*function createInputBoxes(val)
26  %* Create a Nx5 input boxes where
27  %*N is the number of hills and 5 is #columns:
28  %*x, y , maxX, maxY, maxHeight
29  %*args: (1)val->number of hills (integer uint8)
30  %*********************************
31  function createInputBoxes(val)
32      pos=[350 590 40 25];
33      columnLabels={'x' 'y' 'max_x' 'max_y' 'height (max)'};
34      for j=1:5    %columns->x, y , maxX, maxY, maxHeight
35          uicontrol('style','text','position', [pos(1) pos(2)+30
                pos(3) pos(4)],'string',columnLabels(j));
36          for i=1:val(1)  %create a row for each hill
37              if j==1
38                  hillNumber=strcat('#',num2str(i));  %row labels
39                  uicontrol('style','text','position', [pos(1)-30
                        pos(2) pos(3)-15 pos(4)],'string',
                        hillNumber);
40              end
41              switch j    %struct hill has 5 fields->one field
                    for each column
42                  case 1
43                      hill(i).inputDataX=uicontrol('style','edit'
                            ,'position', pos);
44                  case 2
45                      hill(i).inputDataY=uicontrol('style','edit'
                            ,'position', pos);
46                  case 3
```

```
47                      hill(i).inputDataMaxX=uicontrol('style','
                            edit','position', pos);
48                  case 4
49                      hill(i).inputDataMaxY=uicontrol('style','
                            edit','position', pos);
50                  otherwise
51                      hill(i).inputDataMaxHeight=uicontrol('style
                            ','edit','position', pos);
52              end
53              pos(2)=pos(2)-30;
54          end
55          pos(1)=pos(1)+45;
56          if (j==5)&&(i==val(1))
57              ;%nothing-I need last y position
58          else
59              pos(2)=590;
60          end
61      end
62      buttonTwo = uicontrol('style', 'pushbutton','position',
            [320 pos(2) 250 20] , 'string'  , 'Generate');
63      set(buttonTwo, 'callback', {@generateButton, hill, val});
64  %end createInputBoxes
65
66  %*********************************************************
67  %* function terrainGenerator(heightMap,x,y,distanceX, distanceY
        ,
68  %                                                  maxHeight,
        elevations)
69  %* Creates a hill starting from a startpoint(x,y),expanded to
        left on x axis
70  %and to north on y axis
71  %
72  %* args: (1)heightMap-> a 2dim table(NxN) with values. Each
        cell's value is
73  %the height of the point x(row of the table),y(column of the
        table).
74  %       (2)x-> startpoint of the hill on x axis
75  %       (3)y-> startpoint of the hill on y axis
76  %       (4)distanceX->endof hill on x axis
77  %       (5)distanceY->endof hill on y axis
78  %       (6)maxHeight->the maximum height of hill
79  %       (7)elevations->number of height levels
80  %        i.e. maxHeight=10 and elevations=6 ->available heights
        =0,2,4,6,8,10
81  %
82  %* return value: z->the 2dim table with heights
83  %*********************************************************
84  function z=terrainGenerator(heightMap,x,y,distanceX,distanceY,
        maxHeight,elevations)
85      z=heightMap;
86      try
87          elevationTable(1)=maxHeight/(elevations-1); %dont
                include zero height
```

```
88      catch
89          elevations=3;    %minimum elevations number
90      end
91      for i=2:elevations-1
92          elevationTable(i)=elevationTable(i-1)+elevationTable(1)
                ;
93      end
94      for j=y:y+distanceY-1
95          for i=x:x+distanceX-1
96              randomHeight=randomGenerator(elevationTable);
97                  z(i,j)=randomHeight;
98          end
99      end
100 %end terrainGenerator
101
102 %************************************************
103 %* function randomGenerator(data)
104 %* Generates and returns a random value
105 %* args: (1)data-> a 1xN (1 row and N columns)table.
106 %*               Assume each value of table 'data' as an
        available height
107 %* returns: randomValue-> one random value/data of the table '
        data'
108 %************************************************
109 function randomValue=randomGenerator(data) %i.e data=[10 20 25
        100]
110     choose=randperm(length(data));  %i.e. if data[] is as above
            then length(data)=4
111                                     %so choose=[4 1 3 2] or [3
                                           4 1 2] or ...
112     randomValue=data(choose(1));
113 %end randomGenerator
114
115
116 %***********Callbacks***********
117
118 function handles = okButton(hObject, eventdata, handles)
119     for i=1:2
120         input_string = get(handles(i).elevHillsInput,'string');
121         try
122             val(i)=uint8(str2num(input_string));    %integers
                    0-255
123         catch   %incase that thereis not input for hills or/and
                elevations
124             if i==1
125                 defaultValue=1; %set 1 if there is not input
                        for #Hills
126                 set(handles(i).elevHillsInput,'string',num2str(
                        defaultValue));
127                 val(1)=uint8(defaultValue);
128             else
129                 defaultValue=6; %%set 6 if there is not input
                        for #Elevations
```

```
130                 set(handles(i).elevHillsInput,'string',num2str(
                        defaultValue));
131                 val(2)=uint8(defaultValue);
132             end
133         end
134     end
135     if val(1)==0
136         defaultValue=1;
137         set(handles(1).elevHillsInput,'string',num2str(
                defaultValue));
138         val(1)=defaultValue;
139     end
140     createInputBoxes(val);
141     set(hObject, 'enable', 'off');  %disable 'ok' button
142 %end okButton
143
144 function handles = generateButton(hObject, eventdata, handles,
        arg)
145     n=32;    %Z dimensions
146     Z=zeros(n,n);    %initialize Z with zeros
147     %*********************************************
148     %*Get input from input boxes and generate Z table
149     for i=1:arg(1) %arg(1) contains the number of hills
150         dataX = get(handles(i).inputDataX,'string');
151         dataY = get(handles(i).inputDataY,'string');
152         dataMaxX = get(handles(i).inputDataMaxX,'string');
153         dataMaxY = get(handles(i).inputDataMaxY,'string');
154         dataMaxH = get(handles(i).inputDataMaxHeight,'string');
155         try
156         x(i)=uint8(str2num(dataX));
157         y(i)=uint8(str2num(dataY));
158         maxX(i)=uint8(str2num(dataMaxX));
159         maxY(i)=uint8(str2num(dataMaxY));
160         maxHeight(i)=uint8(str2num(dataMaxH));
161         catch
162             quit();
163         end
164         Z=terrainGenerator(Z,x(i),y(i),maxX(i),maxY(i),
                maxHeight(i),arg(2)); %Z contains integers 0-255
165     end
166     %*************************************************
167
168     %**********************
169     %*Clear previous figures
170     figs = get(0, 'Children');
171     close(figs(figs ~= gcf));
172     %**********************
173
174     %***************************************
175     %*plot the heightmap on a nxn (pixels) grid
176     h=figure('NumberTitle','off','Name','Heightmap');
177     [C,h]=contourf(Z',arg(2)-1);
178     colormap('gray');
```

```
179     xlabel('pixel');
180     ylabel('pixel')
181     set(h,'LineStyle','none');
182     %*****************************************
183
184     buttonThree = uicontrol('style', 'pushbutton','Units','
            normalized','position', [.90 .95 .1 .05] , 'string'  ,
            'Print');
185     set(buttonThree, 'callback', {@printButton, h, Z, arg});
186 %end generateButton
187
188 function handles = printButton(hObject, eventdata, handles, Z,
        arg)
189     delete(hObject);    %delete printButton from the new figure
190
191     %************************************************
192     %Trick:When I scaled image at 256x256 I lost pixels at
            right, left,
193     %top and bottom of image
194     %so I did a larger table with zeros there. Now I am going
            to lost black
195     %color there
196     n=44;   %use only evens number greater than 32
197     temp=n-length(Z);
198     newZ=zeros(n,n);
199     m=temp/2+2; %shift right
200     n=temp/2+1; %shift up
201     for j=1:length(Z)
202         for i=1:length(Z);
203             newZ(m,n)=Z(i,j);
204             m=m+1;
205         end
206         n=n+1;
207         m=temp/2+2;
208     end
209     [C,h]=contourf(newZ',arg(2)-1);
210     set(h,'LineStyle','none');
211     %************************************************
212
213     axis off;
214
215     %*************************************
216     %*Eliminate not using space-for example the space where
            axis were
217     set(gca, 'Position', get(gca, 'OuterPosition') - get(gca, '
            TightInset') * [-1 0 1 0; 0 -1 0 1; 0 0 1 0; 0 0 0 1]);
218     set(gca, 'Position',get(gca,'OuterPosition'));
219     %*****************************************
220
221     %*************************************
222     %*Save last figure as a 256x256 png-We are going to blur
            this image
223     set(gcf,'Position',[256 512 256 256]);
```

```
224     set(gcf, 'PaperPositionMode', 'auto');
225     print(gcf, '-r0', 'heightmap.png', '-dpng');
226     %*****************************************
227
228     close(gcf);
229
230     %*************************************
231     %*Blur Heightmap, using gaussian filter
232     gaussianFilter=fspecial('gaussian',[5 5], 1);
233     heightmap = imread('heightmap.png');
234     blurredHeightMap = imfilter(heightmap, gaussianFilter);
235     figure('NumberTitle','off','Name','Blurred 256x256
           Heightmap');
236     axis off;
237     set(gca, 'Position', get(gca, 'OuterPosition') - get(gca, '
           TightInset') * [-1 0 1 0; 0 -1 0 1; 0 0 1 0; 0 0 0 1]);
238     set(gca, 'Position',get(gca,'OuterPosition'));
239     imshow(blurredHeightMap);
240     set(gcf,'Position',[256 512 256 256]);
241     set(gcf, 'PaperPositionMode', 'auto');
242     print(gcf, '-r0', 'heightmap.png', '-dpng');
243     %*************************************
244 %end printButton
```

Βιβλιογραφία

[1] Edward F. Maurina III-*The Game Programmer's Guide to Torque*, A K Peters Ltd., 2006

[2] Kenneth C. Finney-*3D Game Programming All in One*, Stacy L. Hiquet, 2004

[3] Kenneth C. Finney-*Advanced 3D Game Programming All in One*, Stacy L. Hiquet, 2005

[4] *GarageGames' community*, `http://www.garagegames.com/community`

[5] Siri Sjoqvist, Erik Balgard-*3D Game Engines and Design Patterns*, Department of Computer Science and Electronics Malardalen University, 2006

[6] Jake Simpson-*Game Engine Anatomy*, `http://www.extremetech.com/article2/0,3973,594,00.asp`, 2002

[7] Bendik Stang-*Game Engines - Features and possibilities*, IMM DTU, 2003

[8] Wikipedia-*Game Engine*, `http://en.wikipedia.org/wiki/Game_engine`

[9] Wikipedia-*History of video games*, `http://en.wikipedia.org/wiki/History_of_video_game_consoles`

[10] Wikipedia-*OpenGL*, `http://en.wikipedia.org/wiki/OpenGL`

[11] Wikipedia-*Direct3D*, `http://en.wikipedia.org/wiki/Direct3D`

[12] Game Development Wiki-*Game Engines*, `http://wiki.gamedev.net/index.php/Game_Engines`

[13] DevMaster.net-*Engines Listing*, `http://www.devmaster.net/engines/index.php`

[14] Θ. Θεοχάρης, Α. Μπεμ-*Γραφικά Αρχές & Αλγόριθμοι*, Εκδόδεις ΣΥΜΜΕΤΡΙΑ, 1999

[15] Κ. Αναγνώστου-*ΒΙΝΤΕΟΠΑΙΧΝΙΔΙΑ Βιομηχανία και Ανάπτυξη*, Εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ, 2009

[16] Α. Αντωνοπούλου, *Μοντελοποίηση και πλοήγηση σε αλληλεπιδραστικούς εικονικούς κόσμους*, Εθνικό Μετσόβιο Πολυτεχνείο, 2006

[17] Σ. Αποστόλου-Καραμπέλης, *Διαδραστικά εικονικά περιβάλλοντα πολλαπλών χρηστών*, Εθνικό Μετσόβιο Πολυτεχνείο, 2007

[18] Μ. Στάθης, *Ανάπτυξη διαδραστικού τρισδιάστατου παιχνιδιού βασισμένο στη φυσική*, Πολυτεχνείο Κρήτης, 2009

[19] *MilkShape 3D*, `http://chumbalum.swissquake.ch/`

[20] *LithUnwrap*, `http://www.sharecg.com/v/5169/Software-and-Tools/LithUnwrap---Free-UV-Mapper-for-Windows`

[21] Chad Cox-*CS Girl Tutorial*, `http://www.dosfx.com/tutorials/csgirl/default.asp`

[22] Richard Williams-*The Animator's Survival Kit*, Richard Williams Animation Masterclass, 2008