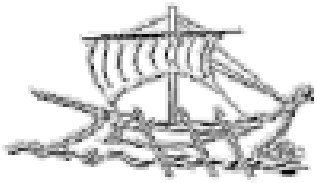


ΑΕΙ ΠΕΙΡΑΙΑ Τ.Τ.
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ Τ.Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη βιντεοπαιχνιδιού στρατηγικής με χρήση της μηχανής Unity

Χρήστος Κρεζίας

Εισηγητής: Γεώργιος Πρεζεράκος, Καθηγητής

ΑΘΗΝΑ
ΔΕΚΕΜΒΡΙΟΣ 2016

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη βιντεοπαιχνιδιού στρατηγικής με χρήση της μηχανής Unity

Χρήστος Κρεζίας
Α.Μ. 40570

Εισηγητής:

Γεώργιος Πρεζεράκος, Καθηγητής

Εξεταστική Επιτροπή:

Ημερομηνία εξέτασης

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/Η κάτωθι υπογεγραμμένος/η ……, του ……, με αριθμό μητρώου …….. φοιτητής/τρια του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο, όπως αυτό της ανάπτυξης βιντεοπαιχνιδιών. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, τον οποίο θα ήθελα να ευχαριστήσω για την ευκαιρία που μου έδωσε.

Ακόμα θα ήθελα να ευχαριστήσω τους συμφοιτητές και φίλους που δοκίμασαν το παιχνίδι και με συμβούλευσαν ως προς την βελτίωση του. Τέλος θέλω να εκφράσω την απεριόριστη αγάπη και ευγνωμοσύνη μου στην οικογένειά μου και τη φίλη μου που με στηρίζουν σε κάθε απόφαση και βήμα.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία ασχολείται με τον σχεδιασμό και την ανάπτυξη βιντεοπαιχνιδιού στρατηγικής με χρήση της μηχανής Unity. Χρησιμοποιείται η γλώσσα προγραμματισμού C#. Γίνεται μια σύντομη ιστορική αναδρομή στα βιντεοπαιχνίδια και πιο συγκεκριμένα στο είδος του παιχνιδιού που θα αναπτυχθεί. Θα αναλυθεί η μηχανή Unity και θα επιχειρηθεί να γίνει χρήση όσων περισσότερων εκ των εργαλείων που παρέχει γίνεται ώστε να καλυφθεί όσο πιο ολοκληρωμένα είναι δυνατό η χρησιμότητα του στην ανάπτυξη και οργάνωση παιχνιδιών. Το παιχνίδι είναι βασισμένο σε γύρους, με χωρισμένο σε κελιά χάρτη, του οποίου παρουσιάζονται οι μέθοδοι δημιουργίας, με τους παίχτες να έχουν προκαθορισμένο αριθμών κινήσεων ανά γύρο. Γίνεται χρήση αλγορίθμων για τη δημιουργία του συστήματος κίνησης και αναπτύσσεται τεχνητή νοημοσύνη για παιχνίδι εναντίων του υπολογιστή. Παρουσιάζεται ο τρόπος χρήσης περίπλοκων μοντέλων με animations και η προσθήκη ήχου και ρυθμίσεων. Μελετάται η ανάπτυξη μενού επιλογών και λειτουργίας παιχνιδιού πολλαπλών παιχτών σε τοπικό δίκτυο για ένα πιο ολοκληρωμένο έργο αλλά και για επίδειξη των συγκεκριμένων δυνατοτήτων. Γίνεται αναφορά στις νομικές διαφυλάξεις του δημιουργού που επιθυμεί την πώληση ή εκμετάλλευση του παιχνιδιού. Παρουσιάζονται προβλήματα που δημιουργήθηκαν στην ανάπτυξη και ο τρόπος που διορθώθηκαν ύστερα από δοκιμή. Σκοπός της πτυχιακής είναι η κατάδειξη των γνώσεων και εργαλείων που χρειάζονται για τη δημιουργία ενός βιντεοπαιχνιδιού.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Ανάπτυξη Βιντεοπαιχνιδιού
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Βιντεοπαιχνίδια, Στρατηγική, Τεχνητή Νοημοσύνη, Unity, C#

ABSTRACT

This thesis deals with the design and development of a strategy video game using the Unity game engine. The C# programming language is being used. There is a brief historical retrospective in video games and more specifically the type of the game to be developed. The Unity engine will be analyzed and the use of as many of the tools provided will be attempted in order to cover the most complete possible usefulness in the development and organization of games. The game is based in turns, with divided cells in the map, with the methods of creation showed, with the players able of a predetermined number of actions per turn. Algorithms for the generation of a path system are being used and artificial intelligence is developed for play against the computer. The use of complex models with animations is shown and the addition of sound and settings. The development of menus is studied as well as a multiplayer in local network game mode for a more integrated work and demonstration of these capabilities. The legal safeguards of the creator who wishes to sell or exploit the game are being referenced. Problems encountered in the development are presented and the way they were corrected after testing. The aim of the thesis is to demonstrate the knowledge and tools needed to create a video game.

SCIENTIFIC SCOPE: Game development
KEY WORDS: Video Games, Strategy, Artificial Intelligence, Unity, C#

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

XML EXtensible Markup Language

UNET Unity Networking System

GUI Graphical User Interface

SDK Software Development Kit

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

1.1 Ανάπτυξη βιντεοπαιχνιδιού με χρήση του Unity

Το θέμα της πτυχιακής είναι ο σχεδιασμός και ανάπτυξη βιντεοπαιχνιδιού. Για την ανάπτυξη του χρησιμοποιείται η μηχανή Unity και το παιχνίδι θα είναι είδους στρατηγικής και μορφής βασισμένου σε γύρους. Θα αναλυθεί ο τρόπος λειτουργίας της μηχανής και η συμβολή της στους μηχανισμούς και τα γραφικά στοιχεία του παιχνιδιού. Παρουσιάζονται και εξετάζονται αλγόριθμοι και κώδικες που χρησιμοποιήθηκαν για την ανάπτυξη του παιχνιδιού. Η γλώσσα προγραμματισμού που χρησιμοποιείται είναι η C Sharp. Θα αναφερθεί επίσης η προσθήκη παιχνιδιού πολλών παιχτών μέσω κοινού δικτύου.

1.2 Ιστορική αναδρομή στα βιντεοπαιχνίδια

Η ιστορία των βιντεοπαιχνιδιών ξεκινάει από το 1947, όταν οι T. Goldsmith Jr. και Estle Ray Mann έφτιαξαν την «Συσκευή Διασκέδασης Καθοδικού Σωλήνα», με σκοπό του παίχτη να πετύχει ένα στόχο. Το 1952 ο A.S. Douglas δημιουργεί το OXO, μια έκδοση της γνωστής σε εμάς «Τρίλιζας», το πρώτο παιχνίδι με γραφικά. Το 1958 ο William Higinbotham δημιουργεί σε παλμοσκόπιο το «Tennis for Two» και το 1967 ο Ralph Baer δημιουργεί το «Chase», το πρώτο παιχνίδι που απεικονίζεται σε τηλεόραση. Το 1971 οι Nolan Bushnell και Ted Dabney δημιουργούν το «Computer Space», βασισμένο στο «SpaceWar!» του Steve Russel, και είναι το πρώτο Arcade παιχνίδι και το 1972 κυκλοφορεί η πρώτη παιχνιδο-κονσόλα οικιακής χρήσης «Odyssey» με 12 παιχνίδια. Την ίδια χρονιά ο Al Alcorn δημιουργεί το διάσημο «Pong» που επανακυκλοφορεί από την ATARI Computers το 1075 για οικιακά συστήματα. Το 1976 η Fairchild εμφανίζει την πρώτη προγραμματιζόμενη παιχνιδο-κονσόλα, το Channel F. Το 1978 αρχίζει η χρυσή εποχή των arcade βιντεοπαιχνιδιών με το Space Invaders της Taito και ακολουθούν τα «Asteroids» και «Luna Lander» της Atari, «Pac-Man» και «Ms. Pac-Man» της Namco και «Donkey Kong» της Nintendo. Το 1983 κυκλοφορεί το πρώτο multiplayer παιχνίδι με τίτλο M.U.L.E. και το 1984 ο μαθηματικός Alexey Pajitnov σχεδιάζει και προγραμματίζει το Tetris. Ακολουθούν τίτλοι όπως τα

«Legend of Zelda», «Final Fantasy» και «Metal Gear», που απόγονοι τους συνεχίζουν μέχρι και σήμερα, και το John Madden Football βάζει το ρεαλισμό στο είδος. Πλέον φορητές και οικιακής χρήσης κονσόλες είναι ευρέως διαδεδομένες, υπολογιστές οικιακής χρήσης είναι ικανοί να τρέξουν παιχνίδια, και έτσι τα βιντεοπαιχνίδια γίνονται κομμάτι της διασκέδασης του κόσμου και βρίσκονται σε κάθε σπίτι.

1.3 Βιντεοπαιχνίδια στρατηγικής

Τα βιντεοπαιχνίδια στρατηγικής είναι ένα είδος που απαιτεί σκέψη και οργάνωση από τον παίχτη για να νικήσει. Έτσι τα περισσότερα παιχνίδια στρατηγικής περιλαμβάνουν πολεμικά στοιχεία σε διάφορες βαθμίδες που εμφανίζουν ένα συνδυασμό τακτικών και στρατηγικών σκέψεων. Εκτός από μάχη πολλές φορές προκαλούν τις ικανότητες του παίχτη στην εξερεύνηση και οικονομική διαχείριση. Οι ρίζες τους πηγάζουν σε παραδοσιακά παιχνίδια όπως το σκάκι και το κινέζικο γκο. Το πρώτο βιντεοπαιχνίδι ήταν το «Invasion» που κυκλοφόρησε το 1972 στην κονσόλα Odyssey. Το 1980 κυκλοφόρησε το «Computer Bismarck», το πρώτο ιστορικό παιχνίδι στρατηγικής. Το 1983 το «Reach for the Stars» προσθέτει τα πρώτα στοιχεία οικονομικής και τεχνολογικής ανάπτυξης στο είδος. Ακολουθούν το μεγάλου μεγέθους ιστορικό «Nobunaga’s Ambition» και το βραβευμένο «The Lords of Midnight» που συνδυάζει περιπέτεια και στρατηγική. Το «Herzog Zwei» του 1989 θεωρείται ως το πρώτο παιχνίδι στρατηγικής πραγματικού χρόνου, προσθέτοντας την απαίτηση διαχείρισης χρόνου. Το είδος έγινε δημοφιλές το 1992 με την κυκλοφορία του «Dune II». Το 2000 κυκλοφόρησε ο πρώτος τίτλος της ιστορικής σειράς «Total War» από την Creative Assembly, σειρά που συνδύασε στρατηγική σε γύρους με αυτήν πραγματικού χρόνου και περιείχε στοιχεία οικονομικής και τεχνολογικής ανάπτυξης καθώς και διπλωματίας, κάνοντας εκατομμύρια πωλήσεις μέχρι και σήμερα.

Το παιχνίδι που θα αναπτυχθεί για τους σκοπούς της πτυχιακής ανήκει στο είδος «Τακτικές με βάση τους γύρους» (Turn-based tactics ή Tactical turn-based). Το είδος μέσω της παύσης της δράσης προσομοιώνει τις σκέψεις και συνθήκες στρατιωτικών τακτικών σε μικρού σκέλους μάχες. Χαρακτηρίζονται από την προσδοκία ότι ο παίχτης θα ολοκληρώσει τις δοθείσες αποστολές με περιορισμένο δυναμικό και πόρους. Παραδείγματα του είδους αποτελούν τα

«Wars», «Jagged Alliance» και «X-COM» καθώς και παιχνίδια ρόλων τακτικής όπως τα «Fire Emblem» και «Final Fantasy Tactics».

1.4 Μηχανή Unity

Η μηχανή παιχνιδιών Unity συμβάλει στην ανάπτυξη δισδιάστατων και τρισδιάστατων βιντεοπαιχνιδιών. Ο χρήστης μπορεί με ευκολία να δημιουργήσει υψηλής ποιότητας παιχνίδια για πολλαπλές κονσόλες και λειτουργικά. Με τη χρήση των διαφόρων εσωτερικών υπηρεσιών της μηχανής μπορεί να επιταχύνει την ανάπτυξη και βελτιστοποίηση του παιχνιδιού, καθώς και να συνδεθεί με το κοινό για δοκιμή, βοήθεια και ενημέρωση. Επιτρέπει την άμεση διάθεση σε ένα μεγάλο πλήθος κινητών και επιτραπέζιων συσκευών, το διαδίκτυο, οικιακές κονσόλες και τηλεοπτικές πλατφόρμες, καθώς και τα νέα συστήματα εικονικής πραγματικότητας. Προσφέρει υπηρεσίες όπως τη δημιουργία και διανομή διαφημίσεων, αναλύσεις, εργαλεία διευκόλυνσης διαδικτυακής συνεργασίας, διαδικτυακό χώρο αποθήκευσης, ελεγκτή απόδοσης, λειτουργίες διαδικτυακού παιχνιδιού πολλών παιχτών, καθώς και πιστοποίησης στις γνώσεις χρήσης της μηχανής. Παρέχει διαδικτυακό κατάστημα με χιλιάδες ελεύθερα ή επί-πληρωμή εργαλεία, προσβάσιμο από την ιστοσελίδα ή και την ίδια την μηχανή. Χρησιμοποιείται από πολλούς ελεύθερους επαγγελματίες ή μικρές εταιρείες ως μια φθηνή, ικανή και αξιόπιστη λύση στο χώρο της ανάπτυξης βιντεοπαιχνιδιών ικανών να αντιμετωπίσουν κολοσσούς της αγοράς σε ποιότητα και κέρδη. Επίσης έχει χρησιμοποιηθεί από μεγάλες εταιρείες όπως τις EA και Disney, και σε αυτό έχουν δημιουργηθεί μεγάλοι τίτλοι όπως τα «Battlestar Galactica Online» της Bigpoint, «Hearthstone» της Blizzard και «Kerbal Space Program» της Squad, με το τελευταίο να έχει λάβει την προσοχή της ίδιας της NASA.

1.5 Γλώσσα προγραμματισμού C Sharp

Η μηχανή παιχνιδιών Unity προσφέρει δύο γλώσσες προγραμματισμού για την ανάπτυξη βιντεοπαιχνιδιών, τη Javascript και τη C Sharp (C#). Για τη δημιουργία του βιντεοπαιχνιδιού της πτυχιακής χρησιμοποιήθηκε η C#, μια απλή, μοντέρνα, γενικού σκοπού και αντικειμενοστραφής γλώσσα προγραμματισμού. Η σύνταξη της γίνεται άμεσα αναγνωρίσιμη από όσους έχουν δουλέψει με τις C, C++ και Java γλώσσες προγραμματισμού. Προγραμματιστές που γνωρίζουν οποιαδήποτε από τις προηγούμενες γλώσσες μπορούν να ξεκινήσουν να

δουλεύουν αμέσως με την C#, η οποία απλοποιεί πολλές από τις πολυπλοκότητες της C++ και της Java και προσφέρει δυνατά στοιχεία όπως τύπους που επιδέχονται null τιμές, απαριθμητές και λάμδα παραστάσεις. Αναπτύχθηκε το 1999 από τον Anders Hajlsberg και την ομάδα του με την αρχική ονομασία «Cool» ως μια αντικειμενοστραφής γλώσσα βασισμένη στη C, και μετονομάστηκε σε C# το 2000 όπου και πρωτοεμφανίστηκε. Η τελευταία έκδοση της είναι η C# 7.0 και κυκλοφόρησε στις αρχές του 2016.

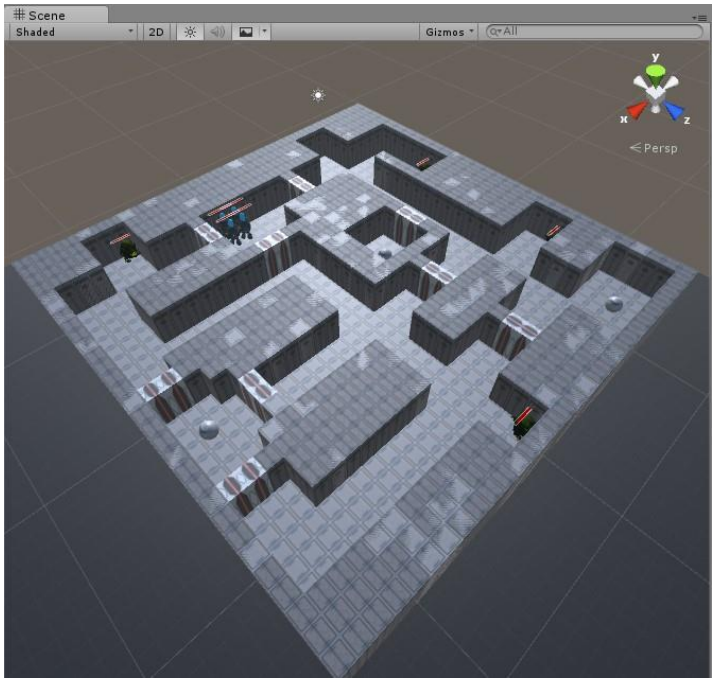
ΚΕΦΑΛΑΙΟ 2

Η ΜΗΧΑΝΗ ΠΑΙΧΝΙΔΙΩΝ UNITY

Η μηχανή παιχνιδιών Unity διαθέτει ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) με έναν ολοκληρωμένο επιμελητή, κατασκευαστή σκηνών, συντάκτη κώδικα, διαδικτυακές λειτουργίες και άλλα. Έχει επίσης μια τεράστια κοινότητα που προσφέρει συμβουλές και απαντήσεις για όποιον αναζητεί οδηγίες χρήσης. Υπάρχουν έξι βασικές όψεις για την ολοκλήρωση των εργασιών, η όψη σκηνής, η όψη έργου, η όψη παιχνιδιού, η όψη ιεραρχίας, η όψη κονσόλας και η όψη ελέγχου, οι οποίες αναλύονται παρακάτω. Επίσης θα αναλυθούν το κατάστημα προσόντων και ο ελεγκτής κίνησης, παροχές όχι απαραίτητες, αλλά που θα χρησιμοποιηθούν για την οπτική βελτίωση του παιχνιδιού.

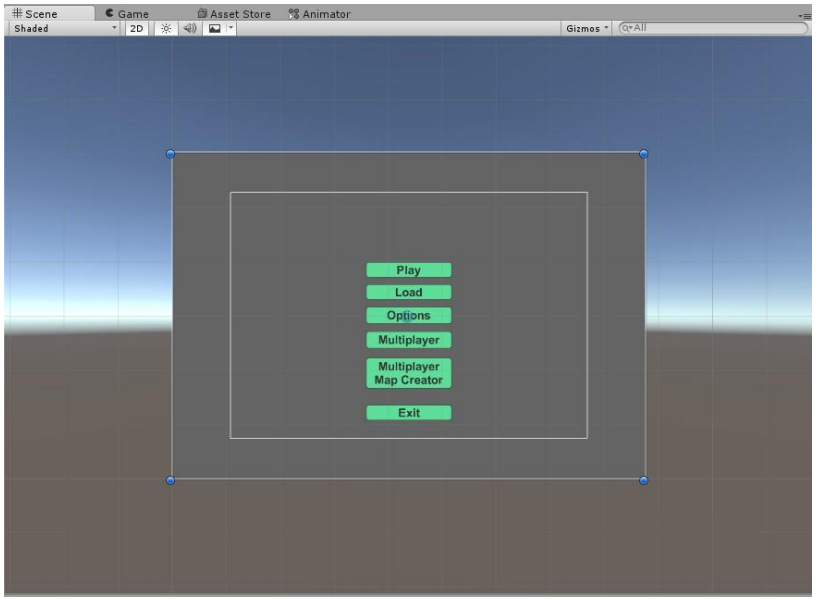
2.1 Όψη Σκηνής (Scene View)

Η όψη σκηνής είναι μία από τις πιο χρησιμοποιημένες όψεις καθώς είναι αυτή στην οποία τοποθετούνται όλα τα αντικείμενα του παιχνιδιού και δημιουργούνται οι σκηνές του.



Εικόνα 2.1.1: Τρισδιάστατη όψη σκηνής

Στην εικόνα 2.1.1 φαίνεται στην όψη σκηνής ένα επίπεδο του παιχνιδιού στρατηγικής που φτιάχτηκε για την πτυχιακή. Το επίπεδο έχει σχεδιαστεί τρισδιάστατα, αλλά επιτρέπεται και η δισδιάστατη όψη του, που χρησιμοποιήθηκε κυρίως για τη δημιουργία των διαφόρων μενού του παιχνιδιού.

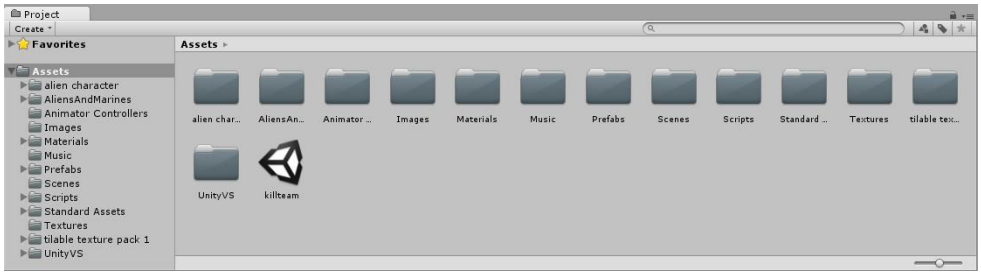


Εικόνα 2.1.1: Δισδιάστατη όψη σκηνής

Η όψη αυτή επιτρέπει στον προγραμματιστή να κινηθεί σε ολόκληρο το 3D περιβάλλον του παιχνιδιού. Για να τοποθετηθούν αντικείμενα σε συγκεκριμένες θέσεις ο χρήστης μπορεί είτε να τα σύρει εκεί, είτε να χρησιμοποιήσει την Snap λειτουργία πατώντας Ctrl, που θα στείλει το αντικείμενο στη θέση οριζόμενη στις ρυθμίσεις της λειτουργίας(Snap Settings) που βρίσκονται στον κατάλογο «Επεξεργασία». Ο χρήστης μπορεί να δει όλα τα σημεία της σκηνής που επεξεργάζεται, ακόμη και αυτά που δε θα είναι εμφανή στον παίχτη στο τελικό προϊόν. Η όψη σκηνής είναι θεωρητικά άπειρη οπότε όποιοι περιορισμοί εξαρτώνται από τον χρήστη.

2.2 Όψη Έργου (Project View)

Η όψη έργου είναι αυτή στην οποία τοποθετούνται όλα τα στοιχεία που αποτελούν το παιχνίδι. Κώδικες, υλικά, μοντέλα, σκηνές, μουσική, εικόνες και άλλα καθώς και το αρχείο του έργου. Μπορεί να οργανωθεί σε φακέλους με υποφακέλους και θυμίζει πολύ την αναζήτηση αρχείων των Windows.

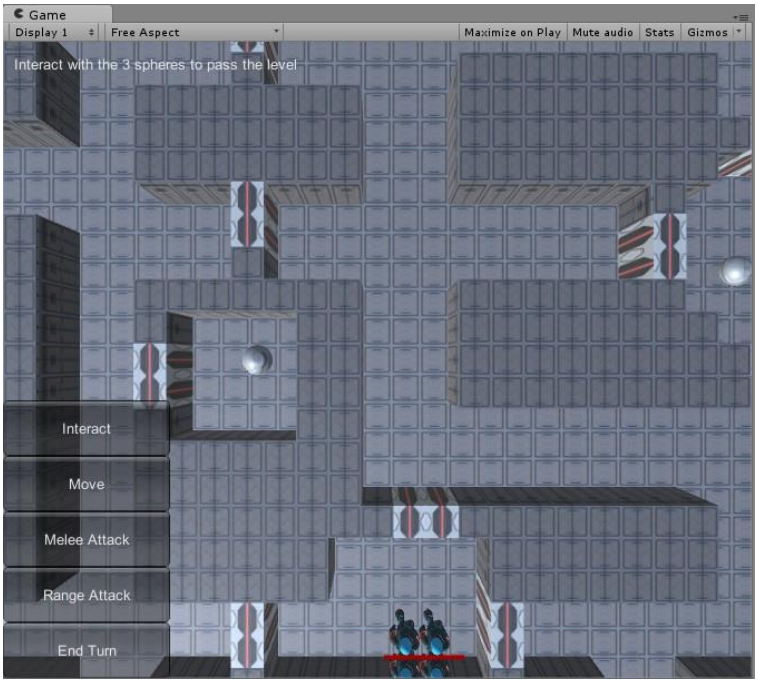


Εικόνα 2.2.1: Όψη Έργου

Στην εικόνα 2.2.1 φαίνεται η όψη έργου του παιχνιδιού. Αποφασίστηκε τα διάφορα στοιχεία να αποθηκευτούν σε διαφορετικούς φακέλους για την ευκολότερη αναζήτηση και περιήγηση σε αυτούς μειώνοντας την περίπτωση λαθών.

2.3 Όψη Παιχνιδιού (Game View)

Η όψη παιχνιδιού είναι αυτή που θα βλέπει ο παίχτης. Στο πάνω μέρος του παραθύρου δίνονται επιλογές όπως η αλλαγή ανάλυσης και μεγιστοποίησης του, σίγασης, παρουσίαση υποπαραθύρου στατιστικών με λεπτομέρειες όπως τα FPS και η λειτουργία του επεξεργαστή, καθώς και λίστα που επιτρέπει στο χρήστη να διαχειριστεί τα αντικείμενα που υπάρχουν στη σκηνή με την απόκρυψη ή εμφάνιση τους.



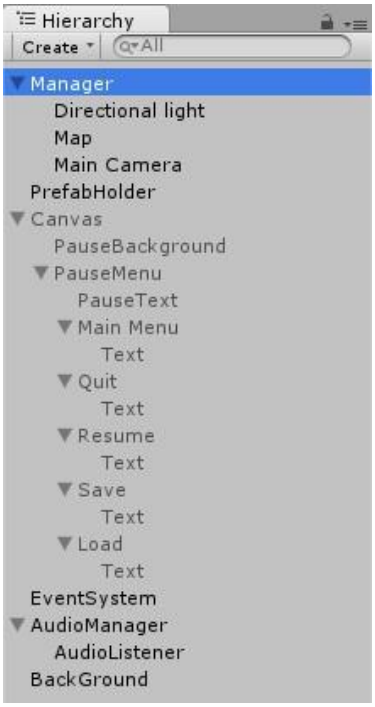
Εικόνα 2.3.1: Όψη παιχνιδιού

Όταν ο παίχτης ξεκινήσει το παιχνίδι της πτυχιακής θα δει μια σειρά από αντικείμενα που θα αποτελούν το χώρο και τους χαρακτήρες του παιχνιδιού, καθώς και μια σειρά από GUI επιλογές με τις οποίες μπορεί να χειριστεί τους χαρακτήρες. Ο παίχτης έρχεται σε επαφή με τον άπειρο 3D κόσμο του παιχνιδιού, μόνο για τόσο όσο του επιτρέπει ο προγραμματιστής. Σε αυτή την όψη ο χρήστης δοκιμάζει ότι το παιχνίδι συμπεριφέρεται όπως επιθυμεί.

Η όψη αυτή ξεκινάει στην σκηνή που έχει επιλέξει εκείνη την στιγμή να εξετάσει ο χρήστης, αλλά επιτρέπει την μετάβαση και σε άλλες σκηνές του συνολικού έργου αν δοθούν οι αντίστοιχες εντολές ή συνθήκες από τον χρήστη κατά την δοκιμή. Εκτός δηλαδή από την εκκίνηση, κατά τα άλλα λειτουργεί σαν το τελικό προϊόν.

2.4 Όψη Ιεραρχίας (Hierarchy View)

Στην όψη ιεραρχίας φαίνονται όλα τα αντικείμενα του παιχνιδιού που βρίσκονται εκείνη τη στιγμή στο 3D κόσμο του. Μπορούν να επιλεχθούν, συγκεντρωθούν και διαχειριστούν από τον χρήστη για τη δημιουργία του παιχνιδιού. Με το να τοποθετήσουμε ένα αντικείμενο σε ένα άλλο στην όψη ιεραρχίας τα συνδέουμε.



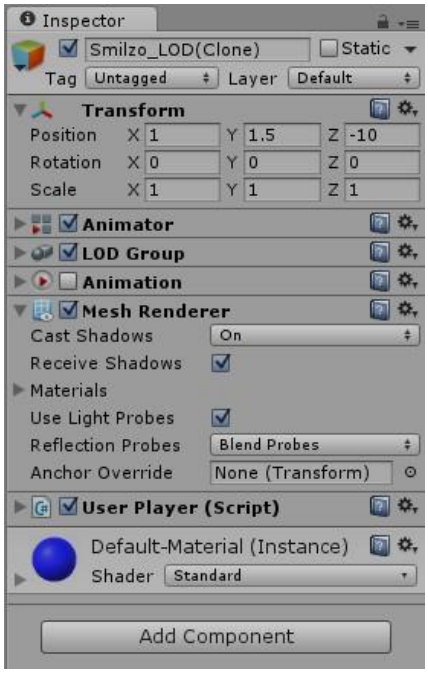
Εικόνα 2.4.1: Όψη Ιεραρχίας

Στην εικόνα 2.4.1 μπορούμε να διακρίνουμε ότι το αντικείμενο «Canvas» και όσα συνδέονται με αυτό είναι γκριζαρισμένα. Αυτό συμβαίνει γιατί επιλέχθηκε να μείνει κρυφό παρότι βρίσκεται στον 3D κόσμο του παιχνιδιού και εμφανίζεται μόνο όταν εφαρμοσθεί η κατάλληλη συνθήκη. Στην συγκεκριμένη περίπτωση όταν ο παίχτης πατάει «Esc» εμφανίζεται το μενού παύσης.

Στο παιχνίδι της πτυχιακής, ενώ το παιχνίδι δεν τρέχει, ελάχιστα από τα αντικείμενα που το απαρτίζουν βρίσκονται στον 3D κόσμο του γιατί τα περισσότερα από αυτά δημιουργούνται αυτόματα κατά την λειτουργία του. Η όψη ιεραρχίας τότε είναι εντελώς διαφορετική, δημιουργώντας στοιχεία για κάθε ένα από τα αντικείμενα που θα εμφανιστούν.

2.5 Όψη Ελέγχου (Inspector View)

Στην όψη ελέγχου φαίνονται, επεξεργάζονται, αλλάζουν και αποθηκεύονται όλα τα στοιχεία του επιλεχθέντος αντικειμένου. Αυτά μπορεί να είναι η τοποθεσία του στον 3D κόσμο, η γωνία στην οποία βρίσκεται, το μέγεθος που έχει, όλα αυτά σε τρεις άξονες, μαζί με ότι κώδικες το επηρεάζουν, μοντέλα που είναι συνδεδεμένα σε αυτό και καθορίζουν την εμφάνιση του και πολλά άλλα.



Εικόνα 2.5.1: Όψη Ελέγχου

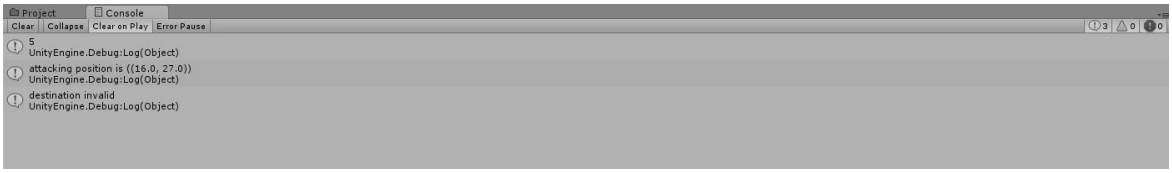
Στην εικόνα 2.5.1 φαίνονται τα στοιχεία που αποτελούν έναν από τους χαρακτήρες που μπορεί να χειριστεί ο παίχτης. Η θέση του στο χώρο, ο ελεγκτής των κινήσεων του και ο κώδικας που χρειάζεται για τη σωστή λειτουργία του είναι μερικά από αυτά. Η Όψη ελέγχου επιτρέπει την πρόσθεση τόσων στοιχείων όσων κρίνει απαραίτητα ο προγραμματιστής. Αλλαγές μπορούν να γίνουν πάντα εκεί, αλλά διατηρούνται μόνο αυτές που γίνονται ενώ δεν τρέχει το παιχνίδι.

Επεξεργασία των στοιχείων μπορεί να γίνει κανονικά από την οθόνη αλλά και μέσω του κώδικα συνδεδεμένου με το αντικείμενο. Για να εμφανιστεί η όψη ελέγχου ο χρήστης μπορεί να πατήσει στο αντικείμενο μέσα στον 3D κόσμο ή επιλέγοντας το από την όψη ιεραρχίας, κάτι που προτιμάται, καθώς έτσι έχει ακριβή γνώση ποιο από τα στοιχεία ενός ομαδοποιημένου αντικειμένου επεξεργάζεται.

Πρόσθεση στοιχείων μπορεί να πραγματοποιηθεί είτε από το αντίστοιχο κουμπί στο τέλος της όψης, με πολλές προκαθορισμένες επιλογές, είτε με σύρσιμο στοιχείων από την όψη έργου, αλλά και με σύρσιμο από την όψη ιεραρχίας. Πολλά στοιχεία απαιτούν υποστοιχεία για να λειτουργήσουν, τα οποία επιλέγονται με τους δύο τελευταίους προαναφερθέντες τρόπους.

2.6 Όψη Κονσόλας (Console View)

Στην όψη κονσόλας φαίνονται όλα τα μηνύματα που έχει ορίσει ο προγραμματιστής να εμφανίζονται κατά την συγγραφή του κώδικα και εμφανίζει που υπάρχουν σφάλματα σε αυτόν.



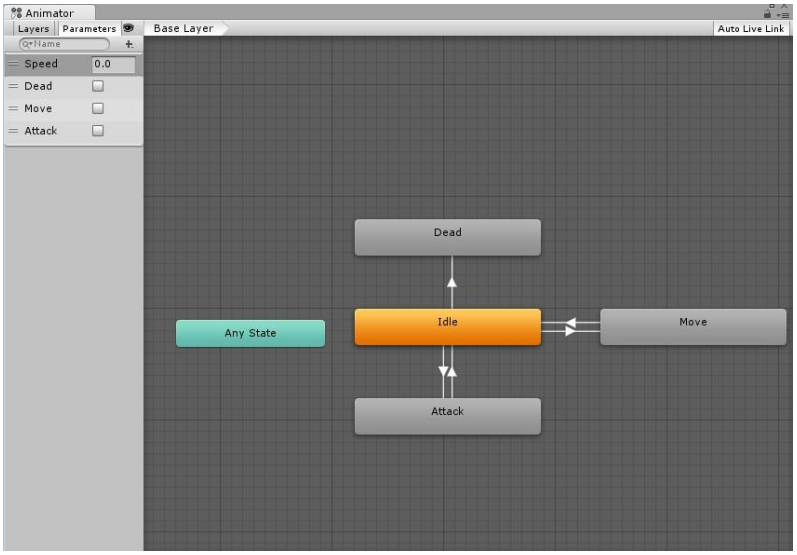
Εικόνα 2.6.1: Όψη Κονσόλας

Έχει ενδείξεις για τον αριθμό των σφαλμάτων που εμφανίζονται, καθώς και σημείων που μπορεί να δημιουργηθούν προβλήματα και χρειάζονται προσοχή. Αναβαθμίζεται σε πραγματικό χρόνο και επιτρέπει την άμεση αποστολή του χρήστη στο σημείο που ανιχνεύθηκε το σφάλμα στον κώδικα με διπλό πάτημα στο μήνυμα. Το ίδιο μπορεί να γίνει και με τα μηνύματα που έχει ορίσει ο

προγραμματιστής να εμφανίζονται για συγκεκριμένες εντολές, ανακατευθύνοντας τον εκεί.

2.7 Ελεγκτής Κίνησης (Animator Controller)

Ο ελεγκτής κίνησης επιτρέπει την οργάνωση των διαφόρων κινήσεων ενός αντικειμένου. Βοηθάει στη ρύθμιση της ομαλής εναλλαγής από τη μια κίνηση στην άλλη και δίνει την δυνατότητα ενεργοποίησης των κινήσεων από λαμβανόμενες από το χρήστη εντολές.



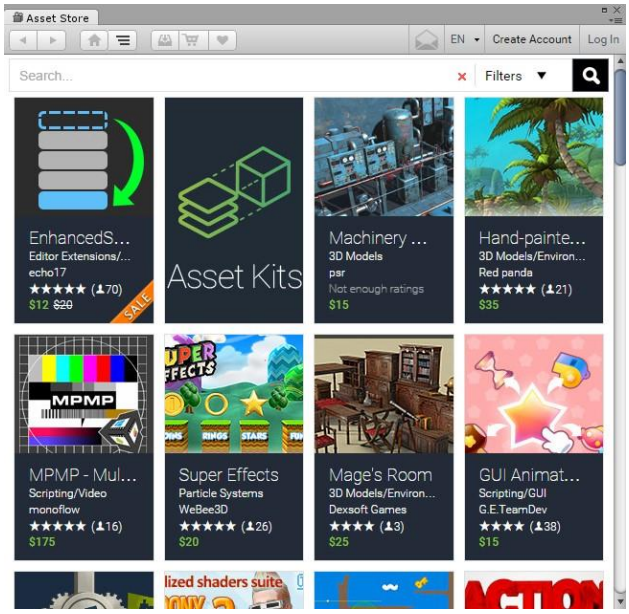
Εικόνα 2.7.1: Ελεγκτής Κίνησης

Στην εικόνα 2.7.1 φαίνεται η σύνδεση και επιτρεπτή μετάβαση των κινήσεων. Η αρχική κίνηση ξεχωρίζει με πορτοκαλί χρώμα και είναι αυτή με την οποία θα ξεκινάει πάντα το αντικείμενο. Για το συγκεκριμένο παράδειγμα οι κινήσεις ενεργοποιούνται από λογικές τιμές αληθούς ψευδούς. Η κατάσταση «Any State» που φαίνεται αλλά δεν είναι συνδεδεμένη κάπου, είναι προ υπάρχουσα κατάσταση που επιτρέπει τη μετάβαση σε μια ή περισσότερες καταστάσεις από οποιαδήποτε άλλη κατάσταση.

2.8 Κατάστημα Προσόντων (Asset Store)

Το κατάστημα προσόντων αποτελεί μια υπηρεσία του Unity που επιτρέπει την απόκτηση ελεύθερων ή επί πληρωμή, καθώς και την πώληση η διανομή, πακέτων σχετικών με την ανάπτυξη βιντεοπαιχνιδιών, όπως 3D ή 2D μοντέλα και

κινήσεις τους, έτοιμους κώδικες, προσθήκες για τη μηχανή, και άλλα.



Εικόνα 2.8.1: Κατάστημα προσόντων

Πρόσβαση μπορεί να αποκτηθεί από την επίσημη ιστοσελίδα του Unity με οποιονδήποτε φυλλομετρητή ή από την ίδια την μηχανή όπως φαίνεται στην εικόνα 2.8.1. Το Unity δε παρέχει εργαλεία δημιουργίας πολύπλοκων 3D αντικειμένων και κινήσεων, οπότε για τη δημιουργία τους χρειάζεται η χρήση άλλων προγραμμάτων που απαιτούν καλές γραφιστικές γνώσεις από τον χρήστη.

Τα μοντέλα και animations των παιχτών, τα μοντέλα των δέντρων και τα skins του εδάφους, των τοίχων και των πορτών που χρησιμοποιούνται στο παιχνίδι έχουν παρθεί από το κατάστημα.

ΚΕΦΑΛΑΙΟ 3

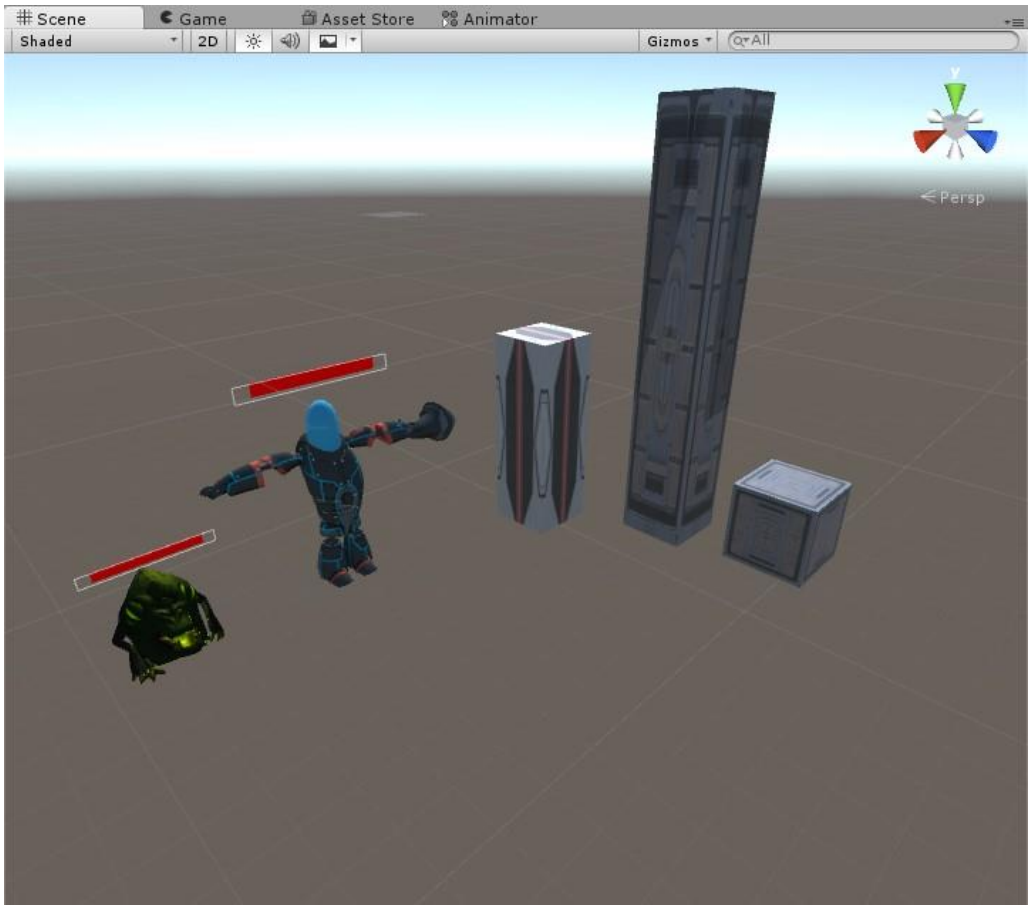
ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ ΣΤΡΑΤΗΓΙΚΗΣ

Πριν ξεκινήσει η ανάπτυξη ενός παιχνιδιού, γίνεται ο σχεδιασμός του. Είναι ένα απαραίτητο κομμάτι που καθορίζει τι θα είναι το παιχνίδι και κατευθύνει τον προγραμματιστή ως προς το τι πρέπει να κάνει. Έτσι ο σχεδιασμός για αρχή πρέπει να καθορίσει το παιχνίδι και να δώσει τις απαραίτητες οδηγίες στον προγραμματιστή να το δημιουργήσει. Πρώτα λοιπόν σχεδιάζονται και αναπτύσσονται τα ελάχιστα απαραίτητα στοιχεία που χρειάζονται ώστε να υπάρχει παιχνίδι. Στο συγκεκριμένο αυτά είναι ένας χώρος, χάρτης, χωρισμένος σε τετράγωνα, πάνω στον οποίο υπάρχει ένας χαρακτήρας που ελέγχει ο παίχτης και ένας χαρακτήρας που ελέγχει ο υπολογιστής. Κινούνται σε συγκεκριμένες αποστάσεις και επιτίθενται ο ένας στον άλλο. Αυτό το κάνουν ανά γύρο και πραγματοποιούν μέχρι δύο τέτοιες ενέργειες ο καθένας στο γύρο του. Παρακάτω θα αναλυθεί πως αναπτύχθηκαν κάθε ένα από αυτά τα στοιχεία.

3.1 Αρχικές Ρυθμίσεις

Στο Unity δημιουργούμε αντικείμενα στον άπειρο χώρο του παιχνιδιού και τους ορίζουμε λειτουργίες για το πως θα συμπεριφέρονται σε αυτόν. Οι λειτουργίες αυτές καθορίζονται από κώδικα που ενεργοποιείται με την έναρξη της λειτουργίας παιχνιδιού. Αν κάποιο αντικείμενο δεν επηρεάζεται από κάποιον κώδικα, απλά θα βρίσκεται άπραγο στο χώρο στη θέση που το είχαμε ορίσει. Έτσι για να μπορεί να ελέγχεται η συμπεριφορά των αντικειμένων και με αυτά η σωστή λειτουργία του κώδικα είναι απαραίτητη η δημιουργία τους στην αρχή των εργασιών. Μέσω του Asset store μπορεί να χρησιμοποιηθεί ένα μεγάλο πλήθος μοντέλων που θα αποτελέσουν τα αντικείμενα για τα οποία θα γραφτεί ο κώδικας. Επίσης μπορούν να χρησιμοποιηθούν απλά τρισδιάστατα σχήματα που προσφέρονται από το ίδιο το Unity. Για το παιχνίδι επιλέχθηκαν ένα ανθρώπινο μοντέλο που θα αποτελέσει το μοντέλο του παίχτη και ένα που θα αποτελέσει το μοντέλο του υπολογιστή. Επίσης επιλέχθηκαν τρία διαφορετικά «skins» τα οποία ορίστηκαν πάνω από ένα απλό κυβικό μοντέλο του Unity ώστε να αναδείξουν

ξεχωριστά σημεία του χάρτη. Με χρήση των ρυθμίσεων που έχει το καθένα άλλαξε η μορφή του στο επιθυμητό για χρήση αποτέλεσμα.



Εικόνα 3.1.1: Τα βασικά μοντέλα που θα χρησιμοποιηθούν.

Στην εικόνα 3.1.1 φαίνονται από αριστερά προς τα δεξιά το μοντέλο του υπολογιστή, το μοντέλο του παίχτη, το μοντέλο αντικειμένου, το μοντέλο χώρου που δεν αλληλοεπιδρά με τον παίχτη και το βασικό μοντέλο του εδάφους. Αφού για λόγους οργάνωσης δημιουργηθεί ένα φάκελος αποθήκευσης των μοντέλων, τα τραβάμε μέσα και μπορούμε να τα διαγράψουμε πλέον από το χώρο του παιχνιδιού.

Απαραίτητη επίσης είναι η προσθήκη αντικειμένου βασικής κάμερας η οποία θα δείχνει αυτά τα σημεία του χώρου που θα βλέπει ο παίχτης. Επιθυμητές είναι η προσθήκες μιας πηγής φωτός και μια πηγής ήχου για την καλύτερη παρουσία του παιχνιδιού. Όλα αυτά μπορούν να είναι προσαρτημένα στο αντικείμενο της κάμερας και να χρησιμοποιηθούν ως σημεία κύριας πηγής.

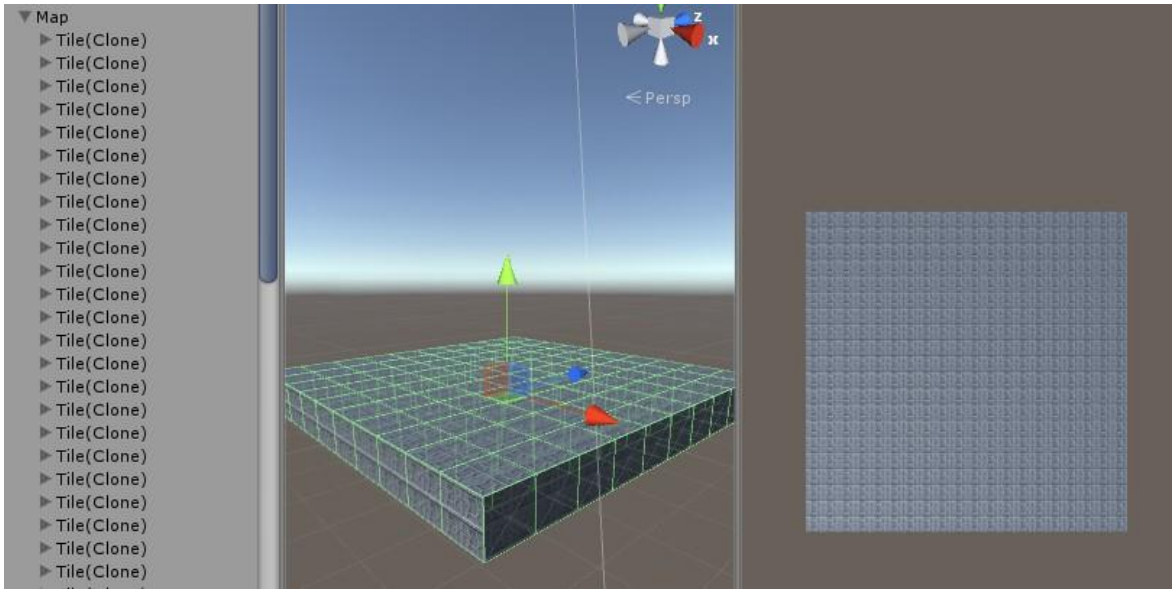
3.2 Δημιουργία Χάρτη

Το πρώτο που πρέπει να γίνει είναι η δημιουργία του εδάφους κίνησης των παιχτών. Εφόσον θα χρησιμοποιηθούν περισσότεροι από έναν χάρτες, θα δημιουργηθεί μια νέα σκηνή που σκοπός της θα είναι εξολοκλήρου ο σχεδιασμός και επεξεργασία των χαρτών που θα χρησιμοποιούνται στις άλλες σκηνές του παιχνιδιού.

Υπάρχουν δύο τρόποι για τη δημιουργία χάρτη στη μορφή παιχνιδιού που θα φτιαχτεί. Στον πρώτο, τον εύκολο αλλά και λάθος τρόπο, θα δημιουργούταν ένα τετράγωνο κίνησης για κάθε τετράγωνο του χάρτη. Αυτό δυσκολεύει τη δημιουργία και επεξεργασία των χαρτών, ενώ είναι και πιο απαιτητικό γιατί κάθε τετράγωνο είναι ένα ξεχωριστό ανεξάρτητο από τα άλλα αντικείμενο στο χώρο. Κάθε λάθος είναι πιο δύσκολο να διορθωθεί και είναι πολύ χρονοβόρο. Οπότε προτιμήθηκε ο δεύτερος τρόπος που απαιτεί χρήση κώδικα αλλά είναι πολύ πιο γρήγορος και αποτελεσματικός γιατί συνδέει όλα τα τετράγωνα του εδάφους δημιουργώντας κλώνους του αρχικού. Θέτοντας λοιπόν το επιθυμητό μέγεθος του χάρτη, ο κώδικας αντιγράφει το βασικό αντικείμενο εδάφους σε στύλους και σειρές έτσι ώστε να δημιουργήσει ένα χάρτη του μεγέθους επί αυτού.

```
void generateBlankMap(int mSize){
    mapSize = mSize;
    for (int i = 0; i < mapTransform.childCount; i++) {
        Destroy (mapTransform.GetChild (i).gameObject);
    }
    map = new List<List<Tile>> ();
    for (int i = 0; i < mapSize; i++) {
        List <Tile> row = new List<Tile> ();
        for (int j = 0; j < mapSize; j++) {
            Tile tile = ((GameObject)Instantiate (PrefabHolder.instance.BASE_TILE_PREFAB,
                new Vector3 (i - Mathf.Floor (mapSize / 2), 0, -j + Mathf.Floor (mapSize / 2)),
                Quaternion.Euler (new Vector3 ()))).GetComponent<Tile> ();
            tile.transform.parent = mapTransform;
            tile.gridPosition = new Vector2 (i, j);
            tile.setType (TileType.normal);
            row.Add (tile);
        }
        map.Add (row);
    }
}
```

Εικόνα 3.2.1: Κώδικας δημιουργίας άδειου χάρτη



Εικόνα 3.2.2: Αποτέλεσμα του κώδικα στις διάφορες οθόνες

3.2.1 XML

Η XML είναι μια γλώσσα σήμανσης που χρησιμοποιείται για την αποθήκευση και μεταφορά δεδομένων. Απλοποιεί την μετάδοση και μεταφορά δεδομένων αποθηκεύοντας τα σε απλή μορφή κειμένου και κάνοντας την ανάγνωση τους ευκολότερη. Αρχεία μορφής XML θα χρησιμοποιηθούν για την αποθήκευση του τύπου και της τοποθεσίας των τετραγώνων του χάρτη.

3.2.2 Αποθήκευση και φόρτωση

Το παιχνίδι όμως θα χρειαστεί παραπάνω από έναν χάρτες, οπότε θα πρέπει να συμπεριληφθεί ένα σύστημα αποθήκευσης και φόρτωσης τους για μελλοντική επεξεργασία και χρήση. Συντάσσεται λοιπόν κώδικας που αποθηκεύει σε ένα αρχείο τύπου .xml όλα τα τετράγωνα του χάρτη με τη θέση στην οποία βρίσκεται το καθένα και τον τύπο του. Συντάσσεται επίσης κώδικας που θα διαβάζει αυτό το αρχείο και θα φορτώνει στον χώρο του παιχνιδιού τον χάρτη όταν θα τρέχει το παιχνίδι. Κάθε χάρτης θα έχει το δικό του .xml αρχείο το οποίο θα φορτώνεται ξεχωριστά όποτε χρειάζεται. Επίσης έχει γραφτεί κώδικας για την επίκληση αυτών των λειτουργιών που θα αντιστοιχεί σε κάθε σκηνή που χρησιμοποιεί χάρτες.

```
public static class mapSaveLoad {
    public static MapXmlContainer CreateMapContainer(List <List<Tile>> map){
        List<TileXml> tiles = new List<TileXml> ();
        for (int i = 0; i < map.Count; i++) {
            for (int j = 0; j < map [i].Count; j++) {
                tiles.Add (mapSaveLoad.CreateTileXml (map [i] [j]));
                }
            }
        return new MapXmlContainer(){
            size=map.Count,
            tiles=tiles
            };
        }

    public static TileXml CreateTileXml(Tile tile){
        return new TileXml(){
            id=(int)tile.type,
            locX=(int)tile.gridPosition.x,
            locY=(int)tile.gridPosition.y
        };
    }

    public static void Save(MapXmlContainer mapContainer, string filename){
        var serializer = new XmlSerializer (typeof(MapXmlContainer));
        var encoding = Encoding.GetEncoding("UTF-8");
        using (StreamWriter stream = new StreamWriter (filename, false, encoding)) {
            serializer.Serialize (stream, mapContainer);
        }
    }

    public static MapXmlContainer Load(string filename){
        var serializer = new XmlSerializer (typeof(MapXmlContainer));
        var encoding = Encoding.GetEncoding("UTF-8");
        using (StreamReader stream = new StreamReader (filename, encoding)) {
            return serializer.Deserialize (stream) as MapXmlContainer;
        }

    }
}

void loadMapFromXml(){
    MapXmlContainer container = mapSaveLoad.Load (mapXml);
    mapSize = container.size;
    for (int i = 0; i < mapTransform.childCount; i++) {
        Destroy (mapTransform.GetChild (i).gameObject);
    }
    map = new List<List<Tile>> ();
    for (int i = 0; i < mapSize; i++) {
        List <Tile> row = new List<Tile> ();
        for (int j = 0; j < mapSize; j++) {
            Tile tile = ((GameObject)Instantiate (PrefabHolder.instance.BASE_TILE_PREFAB,
                new Vector3 (i - Mathf.Floor (mapSize / 2), 0, -j + Mathf.Floor (mapSize / 2)),
                Quaternion.Euler (new Vector3 ()))).GetComponent<Tile> ();
            tile.gridPosition = new Vector2 (i, j);
            tile.setType ((TileType)container.tiles.Where(x=>x.locX==i&&x.locY==j).First().id);
            row.Add (tile);
        }
        map.Add (row);
    }
}

void saveMapToXml(){
    mapSaveLoad.Save (mapSaveLoad.CreateMapContainer (map), mapXml);
}
```

Εικόνα 3.2.3: Κώδικας φόρτωσης και αποθήκευσης χάρτη και επίκλησης τους

Για την επεξεργασία του κενού χάρτη δημιουργούνται οι επιλογές του απλού κελιού κίνησης και του αδιάβατου που θα λειτουργεί σαν εμπόδιο ή τοίχος. Επικαλούνται για το καθένα το αντίστοιχο μοντέλο που δημιουργήθηκε και η χρήση τους ξεχωρίζει κυρίως με μια λογική τιμή που αργότερα καθορίζει αν οι παίχτες μπορούν να κινηθούν στην επιθυμητή θέση. Τοποθετούνται στο χάρτη με το πάτημα του ποντικιού στο τετράγωνο που πρέπει να αλλάξει.

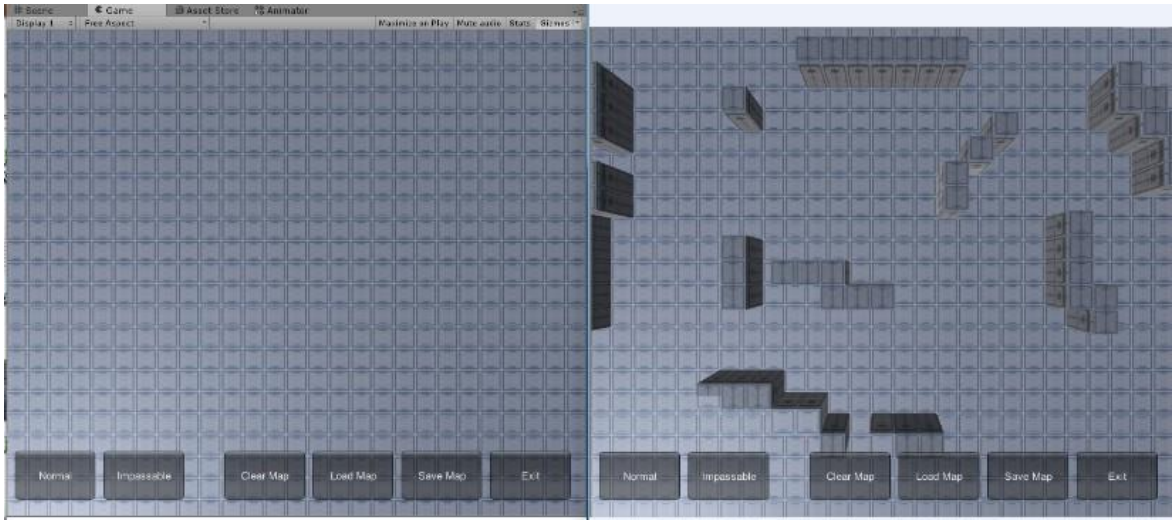
```
void OnMouseEnter(){
    if (SceneManager.GetActiveScene().name == "mapCreatorScene" && Input.GetMouseButton (0)) {
        setType (mapCreatorManager.instance.palletSelection);
    }
}

public void setType(TileType t){
    switch(t){
    case TileType.normal:
        movementCost = 1;
        impassable = false;
        Prefab = PrefabHolder.instance.NORMAL_TILE_PREFAB;
        break;
    case TileType.impassable:
        movementCost = 1;
        impassable = true;
        Prefab = PrefabHolder.instance.IMPASSABLE_TILE_PREFAB;
        break;
    }
    generateVisuals ();
}

public void generateVisuals(){
    GameObject container = transform.FindChild ("Visuals").gameObject;
    for (int i = 0; i < container.transform.childCount; i++) {
        Destroy (container.transform.GetChild (i).gameObject);
    }
    GameObject newVisual = (GameObject)Instantiate (Prefab, transform.position,
        Quaternion.Euler (new Vector3 (0,0,0)));
    newVisual.transform.parent = container.transform;
    visual = newVisual;
}
```

Εικόνα 3.2.4: Κώδικας δημιουργίας εδάφους

Τέλος για τη σκηνή δημιουργίας χάρτη σχεδιάζονται και κουμπιά με τις διάφορες επιλογές που χρειάζονται, δηλαδή η επιλογή του τύπου του κελιού, η επιλογή αποθήκευσης και φόρτωσης του χάρτη, η επιλογή καθαρισμού των αλλαγών και τέλος αυτής της εξόδου από την σκηνή για μελλοντική χρήση εφόσον υπάρχει σκοπός η σκηνή να συμπεριληφθεί στο παιχνίδι για να φτιάχνουν οι παίχτες τους δικούς τους χάρτες.

```
void OnGUI(){
    Rect rect = new Rect (10, Screen.height - 80, 100, 60);
    if (GUI.Button (rect, "Normal")) {
        palletSelection = TileType.normal;
    }
    rect = new Rect (10+(100+10)*1, Screen.height - 80, 100, 60);
    if (GUI.Button (rect, "Impassable")) {
        palletSelection = TileType.impassable;
    }
    rect = new Rect (Screen.width-(10+(100+10)*4), Screen.height - 80, 100, 60);
    if (GUI.Button (rect, "Clear Map")) {
        generateBlankMap (mapSize);
    }
    rect = new Rect (Screen.width-(10+(100+10)*3), Screen.height - 80, 100, 60);
    if (GUI.Button (rect, "Load Map")) {
        loadMapFromXml ();
    }
    rect = new Rect (Screen.width-(10+(100+10)*2), Screen.height - 80, 100, 60);
    if (GUI.Button (rect, "Save Map")) {
        saveMapToXml ();
    }
    rect = new Rect (Screen.width-(10+(100+10)*1), Screen.height - 80, 100, 60);
    if (GUI.Button (rect, "Exit")) {
        SceneManager.LoadScene ("introScene");
    }
}
```

Εικόνα 3.2.5: Κώδικας επιλογών



Εικόνα 3.2.6: Σύγκριση χαρτών πριν και μετά την επεξεργασία

3.3 Δημιουργία Χαρακτήρων

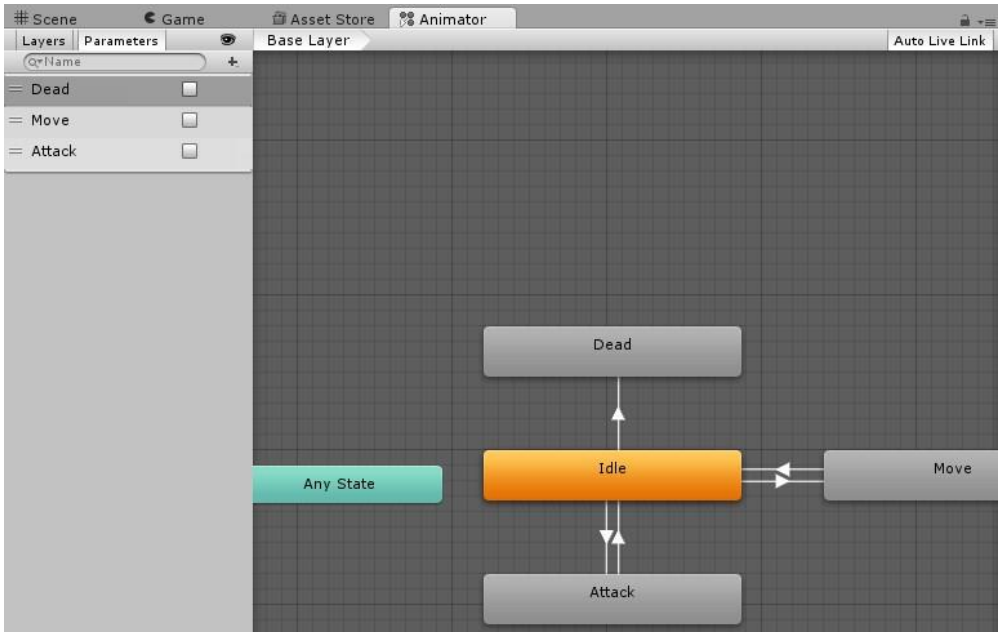
Εφόσον ολοκληρώθηκε ο χάρτης σειρά έχει το επόμενο απαραίτητο στοιχείο του παιχνιδιού, ο παίχτης. Οι παίχτες, αυτοί που θα χειρίζεται ο χρήστης και αυτοί που θα ελέγχει ο υπολογιστής, μοιράζονται ορισμένα χαρακτηριστικά που θα τους επιτρέπουν να αλληλοεπιδρούν, να κινούνται και να διαχειρίζονται αντικείμενα. Ένα σύστημα ποσοστών επηρεάζει την επιτυχία τους στις επιθέσεις και το σύνολο της ζημίας που προκαλούν, λειτουργούν με σύστημα πόντων ζωής και συγκεκριμένο αριθμό κινήσεων ανά γύρο, στοιχεία εμπνευσμένα από τα τακτικά παιχνίδια ρόλων και στρατηγικής. Οι κινήσεις ελέγχονται με λογικές τιμές, όπως και τα αντίστοιχα animations τα οποία ορίζονται να ενεργοποιούνται όταν χρειάζονται.

```
public Vector2 gridPosition=Vector2.zero;
public float moveSpeed = 10.0f;
public int movementPerActionPoint = 5;
public int meleeAttackRange = 1;
public int rangedAttackRange = 5;
public Vector3 moveDestination;
public bool move=false;
public bool attack=false;
public bool interact=false;
public bool heal=false;
public bool animationDeadBool=false;
public bool animationMoveBool=false;
public bool animationAttackBool=false;
public float HP = 25;
public float maxHP = 25;
public float attackchance = 0.75f;
public float AC = 0.15f;
public int dmg = 5;
public float dmgdie = 6;//d6
public string playerName="chris";
public int actionPoints = 2;
public int playerindex=0;
public bool healer=false;
public int attackRange=2;
public Image Health;
public List<Vector3>positionQueue=new List<Vector3>();

void Awake(){
    moveDestination = transform.position;
}

public virtual void TurnUpdate(){
    if (actionPoints <= 0) {
        actionPoints = 2;
        move = false;
        attack = false;
        interact = false;
        heal = false;
        Manager.instance.nextTurn ();
    }
}
```

Εικόνα 3.3.1: Κώδικας παιχτών

3.3.1 Ορισμός κινήσεων και μπάρα ζωής

Το Unity επιτρέπει τον ορισμό animation στα μοντέλα με την λειτουργία animator. Τα animations σχεδιάζονται όπως και τα μοντέλα σε άλλα προγράμματα των οποίων το τελικό προϊόν μπορεί να περαστεί στη μηχανή. Μεγάλη συμβολή σε αυτό έχει και το Asset store όπου κάποιος μπορεί να βρει πληθώρα μοντέλων και animations μεταξύ άλλων, όπως αυτά που χρησιμοποιούνται στο παιχνίδι. Θέτοντας λοιπόν ένα απλό animator στοιχείο στο αντικείμενο του παίχτη, τραβάμε τα animation αρχεία στον αντίστοιχο ελεγκτή και τα συνδέουμε με τους πιθανούς συνδυασμούς οπού μπορεί να μεταβεί το ένα στο άλλο, ρυθμίζοντας παράλληλα το χρόνο της μετάβασης. Τέλος θέτουμε την ενεργοποίηση τους από λογικές τιμές οι οποίες θα αλλάζουν κατά τη διάρκεια του παιχνιδιού για να παιχτεί το αντίστοιχο animation.



Εικόνα 3.3.1.1: Ελεγκτής κινήσεων

Τα διαφορετικά μοντέλα παιχτών που χρησιμοποιούνται δε θα διαφέρουν στη λειτουργία των animation τους, οπότε επαναλαμβάνουμε τη διαδικασία για κάθε μοντέλο με τα κατάλληλα για αυτό animations.

Για ευκολία αναγνώρισης του συνόλου των πόντων ζωής δημιουργείται ένα αντικείμενο καμβά. Σε αυτόν τοποθετούνται δύο εικόνες διαφορετικού χρώματος. Η μία είναι πάνω από την άλλη καλύπτοντας την και αλλάζει σε τύπο γεμίσματος. Συνδέουμε το αντικείμενο του καμβά με αυτό του παίχτη και του προσδίδουμε τον

έτοιμο από το Unity κώδικα να είναι γυρισμένο συνεχώς προς την κεντρική κάμερα. Αργότερα όποτε ο παίχτης θα χάνει πόντους ζωής, ένα ποσοστό θα μειώνεται από την εμπρόσθια εικόνα τύπου γεμίσματος.

3.3.2 Χαρακτήρας Παίχτη

Ο χαρακτήρας παίχτη χρησιμοποιεί κυρίως τον κοινό κώδικα κίνησης, επίθεσης και επίδρασης αντικειμένων, που αναλύεται παρακάτω. Γίνονται οι αρχικοποιήσεις και ορισμοί των animations, ο έλεγχος του σημείου που βρίσκεται ο παίχτης στο χάρτη και η ενημέρωση του αν αυτό έχει αλλάξει, αφαιρώντας έτσι από το χαρακτήρα πόντους κινήσεων.

```
void Start () {
    anim = GetComponent<Animator> ();
    hash = GameObject.FindGameObjectWithTag ("Manager").GetComponent<HashIDs> ();
    Health = transform.FindChild ("Canvas").FindChild ("HealthBar").FindChild ("Health").GetComponent<Image> ();
}

void Update () {
    deadAnimation (animationDeadBool);
    moveAnimation (animationMoveBool);
    attackAnimation (animationAttackBool);
    if (Manager.instance.players [Manager.instance.currentPlayerIndex] == this) {
        transform.GetComponent<Renderer>().material.color = Color.blue;
    } else {
        transform.GetComponent<Renderer>().material.color = Color.white;
    }
    if (HP <= 0) {
        animationAttackBool = false;
        animationDeadBool = true;
    }
}

public void deadAnimation(bool dead){
    anim.SetBool (hash.deadBool, dead);
}

public void moveAnimation(bool move){
    anim.SetBool (hash.moveBool, move);
}

public void attackAnimation(bool attack){
    anim.SetBool (hash.attackBool, attack);
}

public override void TurnUpdate(){
    if (positionQueue.Count > 0) {
            transform.position += (positionQueue[0] - transform.position).normalized * moveSpeed * Time.deltaTime;
            if (Vector3.Distance (positionQueue[0], transform.position) <= 0.1f) {
                transform.position = positionQueue[0];
                positionQueue.RemoveAt (0);
                if (positionQueue.Count == 0) {
                    actionPoints--;
                    Manager.instance.removeTileHighlights();
                }
            }
    }
    base.TurnUpdate ();
}
```

Εικόνα 3.3.2.1: Κώδικας συμπεριφοράς παίχτη

Επίσης συντάσσεται κώδικας που εμφανίζει τα κουμπιά των επιλογών κινήσεων του παίχτη και συνδέονται άμεσα με τους κώδικες λειτουργίας του χαρακτήρα, ενεργοποιώντας την αντίστοιχη συμπεριφορά στο πάτημα τους.

```
public override void TurnOnGUI(){
    float buttonHeight = 50;
    float buttonWidth = 150;

    Rect buttonRect = new Rect (0, Screen.height - buttonHeight * 5, buttonWidth, buttonHeight);
    if (GUI.Button (buttonRect, "Interact")) {
        if (!attack) {
            Manager.instance.removeTileHighlights();
            move = false;
            attack = false;
            interact = true;
            Manager.instance.highlightTilesAt (gridPosition, Color.yellow, meleeAttackRange, false,true);
        } else {
            move = false;
            attack = false;
            interact = false;
            Manager.instance.removeTileHighlights();
        }
    }
```

Εικόνα 3.3.2.2: Κώδικας επιλογών παίχτη

Στην εικόνα 3.3.2.2 φαίνεται ένα δείγμα της μορφής του κώδικα, συγκεκριμένα το κουμπί για την χρήση αντικειμένων του παιχνιδιού από τον παίχτη. Καλεί την συνάρτηση που δείχνει την περιοχή δυνατότητας επιλογής και αλλάζει τις απαραίτητες λογικές τιμές. Προστίθενται οι επιλογές για επίθεση και κίνηση που ενεργοποιούν τις αντίστοιχες συναρτήσεις και αλλάζουν τις αντίστοιχες τιμές.

3.3.3 Χαρακτήρας Υπολογιστή

Ο χαρακτήρας που ελέγχει ο υπολογιστής απαιτεί την ανάπτυξη τεχνητής νοημοσύνης που θα καθορίζει τις κινήσεις του στο χώρο του παιχνιδιού. Οι βασικές λειτουργίες που χρειάζεται είναι να κινείται προς τους χαρακτήρες που χειρίζεται ο παίχτης και να τους επιτίθεται όταν βρίσκεται σε εμβέλεια. Οπότε ο χαρακτήρας ελέγχει όλο το χάρτη για να εντοπίσει τις τοποθεσίες των χαρακτήρων του παίχτη. Τους κατηγοριοποιεί ανάλογα με το ποιος βρίσκεται πιο κοντά στο σημείο έναρξης του και τους κοντινότερους τους κατηγοριοποιεί ανάλογα με το σύνολο των πόντων ζωής τους. Ξοδεύει τους απαραίτητους από το σύνολο των πόντων δράσεως για να κινηθεί στην κοντινότερη απόσταση από την οποία μπορεί να επιτεθεί σε αυτόν με τους λιγότερους πόντους ζωής. Ύστερα ξοδεύει πόντους δράσεως και συνεχίζει να επιτίθεται μέχρι ο παίχτης του χρήστη να χάσει

όλους τους πόντους ζωής του, όπου και η διαδικασία επαναλαμβάνεται. Όπως και ο παίχτης του χρήστη ενημερώνει συνεχώς την παρούσα θέση του. Κατά τη διάρκεια επιλογής των κινήσεων του ενεργοποιούνται τα αντίστοιχα animations. Στο τέλος των μέγιστων πόντων δράσεως του, τελειώνει το γύρο του παραχωρώντας τον στον επόμενο σε σειρά προτεραιότητας παίχτη, ανανεώνοντας τους.

```
public override void TurnUpdate(){

    if (positionQueue.Count > 0) {
        transform.position += (positionQueue[0] - transform.position).normalized * moveSpeed * Time.deltaTime;
        if (Vector3.Distance (positionQueue [0], transform.position) <= 0.1f) {
            transform.position = positionQueue [0];
            positionQueue.RemoveAt (0);
            if (positionQueue.Count == 0) {
                actionPoints--;
            }
        }
        } else {
        List<Tile> attacktilesInRange=TileHighlight.FindHighlight
            (Manager.instance.map[(int)gridPosition.x][(int)gridPosition.y],attackRange);
        List<Tile> movementTilesInRange=TileHighlight.FindHighlight
            (Manager.instance.map[(int)gridPosition.x][(int)gridPosition.y],movementPerActionPoint+1000);
        if (attacktilesInRange.Where(x=>Manager.instance.players.Where(y=>y.GetType()!=typeof(AIPlayer)&&
            y.HP>0&&y!=this&&y.gridPosition==x.gridPosition).Count()>0).Count()>0){
            if (animationMoveBool) animationMoveBool = false;
            animationAttackBool = true;
            var opponentsInRange = attacktilesInRange.Select
                (x => Manager.instance.players.Where (y=>y.GetType()!=typeof(AIPlayer)&&y.HP>0 &&
                    y != this && y.gridPosition == x.gridPosition).Count() > 0 ? Manager.instance.players.Where
                    (y => y.gridPosition == x.gridPosition).First () : null).ToList();
                Player opponent = opponentsInRange.OrderBy (x =>x!=null? -x.HP:1000).First ();
                Manager.instance.removeTileHighlights ();
                move = false;
                attack = true;
                Manager.instance.highlightTilesAt (gridPosition, Color.red, attackRange, true, false);
                Manager.instance.attackWithCurrentPlayer (Manager.instance.map[(int)opponent.gridPosition.x][(int)opponent.gridPosition.y]);
                Manager.instance.removeTileHighlights ();
        }else if (attacktilesInRange.Where(x=>Manager.instance.doors.Where(y=>y.GetType()!=typeof(AIPlayer)&&y.HP>0&&
            y!=this&&y.gridPosition==x.gridPosition).Count()>0).Count()>0){
                if (animationMoveBool) animationMoveBool = false;
                animationAttackBool = true;
                var opponentsInRange = attacktilesInRange.Select
                (x => Manager.instance.doors.Where (y=>y.GetType()!=typeof(AIPlayer)&&y.HP>0
                    && y != this && y.gridPosition == x.gridPosition).Count() > 0 ? Manager.instance.doors.Where
                    (y => y.gridPosition == x.gridPosition).First () : null).ToList();
                Door door = opponentsInRange.OrderBy (x =>x!=null? -x.HP:1000).First ();
                Manager.instance.removeTileHighlights ();
                move = false;
                attack = true;
                Manager.instance.highlightTilesAt (gridPosition, Color.red, attackRange, true, true);
                Manager.instance.interactWithObject (Manager.instance.map[(int)door.gridPosition.x][(int)door.gridPosition.y]);
                Manager.instance.removeTileHighlights ();
            }
```

Εικόνα 3.3.3.1: Κώδικας συμπεριφοράς της τεχνητής νοημοσύνης για επίθεση και χρήση αντικειμένων

```
        else if(movementTilesInRange.Where(x=>Manager.instance.players.Where(y=>y.GetType()!=typeof(AIPlayer)&&y.HP>0&&
            y!=this&&y.gridPosition==x.gridPosition).Count()>0).Count()>0){
            var opponentsInRange = movementTilesInRange.Select (x => Manager.instance.players.Where (y=>y.GetType()!=typeof(AIPlayer)&&y.HP>0 &&
                y != this && y.gridPosition == x.gridPosition).Count() > 0 ? Manager.instance.players.Where
                (y => y.gridPosition == x.gridPosition).First () : null).ToList();
            Player opponent = opponentsInRange.OrderBy (x =>x!=null ? -x.HP:1000).ThenBy
                (x =>x!=null ? TilePathFinder.FindPath(Manager.instance.map[(int)gridPosition.x][(int)gridPosition.y],
                    Manager.instance.map[(int)x.gridPosition.x][(int)x.gridPosition.y]).Count():1000).First ();
            Manager.instance.removeTileHighlights ();
            if (animationAttackBool) animationAttackBool = false;
            animationMoveBool = true;
            move = true;
            attack = false;
            Manager.instance.highlightTilesAt (gridPosition, Color.blue, movementPerActionPoint,false,false);
            List<Tile> path = TilePathFinder.FindPath
                (Manager.instance.map [(int)gridPosition.x] [(int)gridPosition.y],
                    Manager.instance.map [(int)opponent.gridPosition.x] [(int)opponent.gridPosition.y],
                    Manager.instance.players.Where
                    (x=>x.gridPosition!=gridPosition && x.gridPosition!=opponent.gridPosition).Select(x=>x.gridPosition).ToArray(),false);
            if (path.Count () > 1) {
                List<Tile> actualMovement = TileHighlight.FindHighlight
                    (Manager.instance.map [(int)gridPosition.x] [(int)gridPosition.y],
                        movementPerActionPoint, Manager.instance.players.Where
                        (x => x.gridPosition != gridPosition).Select (x => x.gridPosition).ToArray (),
                        Manager.instance.doors.Where (x => x.gridPosition != gridPosition).Select (x => x.gridPosition).ToArray ());
                path.Reverse ();
                if (path.Where (x => actualMovement.Contains (x)).Count () > 0)
                    Manager.instance.moveCurrentPlayer (path.Where (x => actualMovement.Contains (x)).First ());
                    Manager.instance.removeTileHighlights ();
            }
        }
    }
    if (actionPoints <= 1 && (attack || move)) {
        move = false;
        attack = false;
    }
    Debug.Log ("my position is (" + gridPosition.x + "," + gridPosition.y + ")");
    base.TurnUpdate ();
}
```

Εικόνα 3.3.3.2: Κώδικας συμπεριφοράς της τεχνητής νοημοσύνης για κίνηση και τέλος του γύρου

Η τεχνητή νοημοσύνη επιλέγει μόνη της σύμφωνα με τις οδηγίες που έχουν δοθεί τα σημεία στα οποία κινείται ή επιτίθεται ο παίχτης του υπολογιστή. Για τον παίχτη του χρήστη επιλέγει ο χρήστης μέσω πατήματος του ποντικιού. Αν η τεχνητή νοημοσύνη δεν βρει κάποιο σημείο να επιλέξει για οποιαδήποτε ενέργεια, όπως να κινηθεί κοντά σε κάποιον εχθρό του οποίου τα γύρω κελιά είναι κατειλημμένα, τελειώνει το γύρο της και παραδίδει την κίνηση στον επόμενο σε σειρά παίχτη.

3.4 Κίνηση

3.4.1 Αλγόριθμος του Dijkstra

Το 1956 ο Ολλανδός επιστήμονας πληροφορικής Edsger Wyde Dijkstra επινόησε έναν αλγόριθμο εύρεση συντομότερων διαδρομών από κοινή αφετηρία. Σε κάθε βήμα επιλέγεται η τοπικά βέλτιστη λύση με το τελευταίο να συνθέτει την συνολικά βέλτιστη διαδρομή.

3.4.2 Ανάπτυξη κίνησης

Για την κίνηση των παιχτών αναπτύχθηκε κώδικας που τους δείχνει τα τετράγωνα στα οποία θα μπορούν να κινηθούν στο χάρτη για τον αριθμό κίνησης τους. Με πρότυπο τον αλγόριθμο του Dijkstra ο παίχτης ελέγχει από την αρχική του θέση τα τετράγωνα στα οποία μπορεί να κινηθεί, αγνοώντας τοίχους ή/και παίχτες και αντικείμενα μέχρι το μέγιστο που του επιτρέπεται από το όριο που του έχει τεθεί. Έτσι θα δημιουργήσει διαδρομές που θα πηγαίνουν γύρω από όλα τα εμπόδια που μπορεί να συναντήσει.

```
public TilePath(TilePath tp){
    listOfTiles = tp.listOfTiles.ToList();
    costOfPath = tp.costOfPath;
    lastTile = tp.lastTile;
}
public void addTile(Tile t){
    costOfPath += t.movementCost;
    listOfTiles.Add (t);
    lastTile = t;
}
public void addStaticTile(Tile t){
    costOfPath += 1;
    listOfTiles.Add (t);
    lastTile = t;
}
```

Εικόνα 3.4.2.1: Κώδικας υπολογισμού και ανανέωσης απόστασης

```
public static List<Tile> FindHighlight(Tile originTile, int movementPoints,
    Vector2[] playeroccupied, Vector2[] dooroccupied, bool staticRange){
    Debug.Log ("5");
    List<Tile> closed = new List<Tile> ();
    List<TilePath> open = new List<TilePath> ();
    TilePath originPath = new TilePath ();
    if (staticRange)
        originPath.addStaticTile (originTile);
    else originPath.addTile (originTile);
    open.Add (originPath);
    while (open.Count > 0) {
        TilePath current = open[0];
        open.Remove (open [0]);
        if (closed.Contains (current.lastTile)) {
            continue;
        }
        if (current.costOfPath > movementPoints+1) {
            continue;
        }
        closed.Add (current.lastTile);
        foreach (Tile t in current.lastTile.neighbors) {
            if (t.impassable ||
                playeroccupied.Contains (t.gridPosition)||dooroccupied.Contains (t.gridPosition))
                continue;
            TilePath newTilePath = new TilePath(current);
            if (staticRange)
                newTilePath.addStaticTile (t);
            else newTilePath.addTile (t);
            open.Add (newTilePath);
        }
    }
    closed.Remove (originTile);
    closed.Distinct();
    return closed;
}
```

Εικόνα 3.4.2.2: Κώδικας εισαγωγής συνθηκών για την εύρεση διαδρομής

Για να μπορεί ο παίχτης να επιλέξει το τετράγωνο στο οποίο θέλει να κινηθεί στον επιτρεπτό χώρο συντάχθηκε κώδικας που φωτίζει τα αντίστοιχα κελιά. Η συνάρτηση δέχεται την τοποθεσία πηγή, το επιθυμητό χρωματισμό, την μέγιστη απόσταση και λογικές τιμές για την αγνόηση παιχτών ή/και αντικειμένων ανάλογα την ενέργεια που πρόκειται να πραγματοποιήσει ο παίχτης. Καλώντας την συνάρτηση της εικόνας 3.4.3, η οποία με τη χρήση των δεδομένων των συναρτήσεων της εικόνας 3.4.1 βρίσκει τις πιθανές διαδρομές στις οποίες μπορεί να επιδράσει ο παίχτης και αυτές φωτίζονται σαν μια ενιαία περιοχή στο αντίστοιχο χρώμα.

```
public void highlightTilesAt(Vector2 originLocation,Color highlightColor, int distance, bool ignorePlayers, bool ignoreDoors){
    List<Tile> highlightedTiles = new List<Tile> ();
    if (ignorePlayers==true||ignoreDoors==true)
        highlightedTiles = TileHighlight.FindHighlight (map [(int)originLocation.x] [(int)originLocation.y], distance);
    else
        highlightedTiles = TileHighlight.FindHighlight (map [(int)originLocation.x] [(int)originLocation.y], distance,
            players.Where (x => x.gridPosition != originLocation).Select (x => x.gridPosition).ToArray (),
            doors.Where (x => x.gridPosition != originLocation).Select (x => x.gridPosition).ToArray (),highlightColor==Color.red);
    foreach (Tile t in highlightedTiles) {
        t.visual.transform.GetComponent<Renderer>().materials[0].color = highlightColor;
    }
}

public void removeTileHighlights(){
    for (int i = 0; i < mapSize; i++) {
        for (int j = 0; j < mapSize; j++) {
            if (!map[i][j].impassable)map [i] [j].visual.transform.GetComponent<Renderer>().materials[0].color = Color.white;
        }
    }
    if (players [currentPlayerIndex].playerindex == 1) {
        if (players [currentPlayerIndex].animationMoveBool) players [currentPlayerIndex].animationMoveBool = false;
        if (players [currentPlayerIndex].animationAttackBool) players [currentPlayerIndex].animationAttackBool = false;
    }
}
```

Εικόνα 3.4.2.3: Κώδικας διαχείρισης φωτισμού περιοχής

Η πρώτη συνάρτηση της εικόνας 3.4.3 θα χρησιμοποιηθεί για κάθε περίπτωση όπου ο παίχτης θα κληθεί να αλληλοεπιδράσει με τον χάρτη. Ο παίχτης για οτιδήποτε κάνει στην ουσία αλληλοεπιδρά με τον χάρτη, ακόμη και όταν επιτίθεται ή χρησιμοποιεί κάποιο αντικείμενο. Η επίδραση στα μοντέλα γίνεται μέσω των τετραγώνων που βρίσκονται, οπότε όταν ο παίχτης για παράδειγμα επιτίθεται σε έναν αντίπαλο, επιλέγει το τετράγωνο του και όχι το μοντέλο του. Οι παίχτες του υπολογιστή επίσης κάνουν χρήση της λειτουργίας φώτισης της διαδρομής, διαδικασία όμως που γίνεται πολύ γρήγορα και δε φαίνεται στο χρήστη.

Η δεύτερη συνάρτηση σβήνει τα φωτισμένα μοντέλα. Όταν επικαλείται διαβάζει όλο το χάρτη και μετατρέπει τον τύπο χρώματος της φωτεινότητας σε λευκό.

Για τη μεταφορά του παίχτη από το ένα σημείο στο άλλο στην περίπτωση της κίνησης γράφεται συνάρτηση που δέχεται το τετράγωνο πηγής, αυτό του προορισμού και ένας πίνακας με τα στοιχεία των τετραγώνων που βρίσκονται στο χώρο όπου θα γίνει η κίνηση. Θα σχηματιστούν διαδρομές που θα αποφεύγουν αδιαπέραστα και κατειλημμένα κελιά και θα επιλεχθεί η συντομότερη για να κινηθεί ο παίχτης.

```
public static List<Tile> FindPath(Tile originTile, Tile destinationTile,
    Vector2[] occupied, bool door){
    List<Tile> closed = new List<Tile> ();
    List<TilePath> open = new List<TilePath> ();
    TilePath originPath = new TilePath ();
    originPath.addTile (originTile);
    open.Add (originPath);
    while (open.Count > 0) {
        TilePath current = open[0];
        open.Remove (open [0]);
        if (closed.Contains (current.lastTile)) {
            continue;
        }
        if (current.lastTile == destinationTile) {
            current.listOfTiles.Distinct();
            current.listOfTiles.Remove (originTile);
            return current.listOfTiles;
        }
        closed.Add (current.lastTile);
        foreach (Tile t in current.lastTile.neighbors) {
            TilePath newTilePath = new TilePath (current);
            if (door) {
                newTilePath.addTile (t);
                open.Add (newTilePath);
            } else if (t.impassable || occupied.Contains (t.gridPosition))
                continue;
            else {
                newTilePath.addTile (t);
                open.Add (newTilePath);
            }
        }

    }
    return null;
}
```

Εικόνα 3.4.2.4: Κώδικας εύρεσης διαδρομής

Τέλος για την δυνατότητα κίνησης συντάσσεται συνάρτηση που δέχεται το κελί προορισμού. Ελέγχει ότι βρίσκεται εντός του επιτρεπτού χώρου κίνησης και κάνει της απαραίτητες ρυθμίσεις στις λογικές τιμές λειτουργίας και animation και με τη χρήση της συνάρτησης στην εικόνα 3.4.4 μετακινήσει τον παίχτη στον προορισμό του ανανεώνοντας τη θέση του. Η συνάρτηση καλείται όταν γίνεται πάτημα σε επιτρεπτό τετράγωνο για τον παίχτη του χρήστη ενώ για τον παίχτη του υπολογιστή καλείται από την τεχνητή νοημοσύνη που ορίζει το κελί προορισμού.

```
public void moveCurrentPlayer(Tile destTile){
    if (destTile.visual.transform.GetComponent<Renderer>().materials[0].color != Color.white && !destTile.impassable) {
        removeTileHighlights ();
        players [currentPlayerIndex].move = false;
        turnAround (destTile);
        players [currentPlayerIndex].animationMoveBool = true;
        foreach(Tile t in TilePathFinder.FindPath
            (map [(int)players[currentPlayerIndex].gridPosition.x] [(int)players[currentPlayerIndex].gridPosition.y],destTile,
                players.Where (x => x.gridPosition != players[currentPlayerIndex].gridPosition).Select (x => x.gridPosition).ToArray (),false)){
            players[currentPlayerIndex].positionQueue.Add(map[(int)t.gridPosition.x][(int)t.gridPosition.y].transform.position+ 1.5f * Vector3.up);
        }
        players [currentPlayerIndex].gridPosition = destTile.gridPosition;
    } else {
        Debug.Log ("destination invalid");
    }

}
```

Εικόνα 3.4.2.5: Κώδικας κίνησης επιλεγμένου παίχτη

Εικόνα 3.4.2.6: Κίνηση χαρακτήρα

Στην εικόνα 3.4.2.6 φαίνεται η διαδικασία κίνησης του χαρακτήρα του χρήστη. Πρώτα επιλέγεται το κουμπί κίνησης, μετά στον επιτρεπτό φωτεινό χώρο το τετράγωνο προορισμού και ακολουθεί η κίνηση μεταφοράς του χαρακτήρα εκεί.

3.5 Ανάπτυξη Μάχης

Με τις συναρτήσεις εύρεσης και σήμανσης της εντός των επιτρεπτών ορίων περιοχής έτοιμες, η ανάπτυξη του συστήματος μάχης είναι σχετικά ευκολότερη από αυτού της κίνησης. Αφού γίνει η επιλογή επίθεσης θα φωτιστεί η αντίστοιχη περιοχή για να επιλεχθεί το κελί με τον εχθρικό στόχο. Η συνάρτηση διαβάζει τη θέση που επιλέχθηκε και τη μεταφράζει σα θέση στόχου. Μετά από τις αντίστοιχες animation επικλήσεις και τις αλλαγές στα στοιχεία του παίχτη ενεργοποιείται το σύστημα μάχης με μια τυχαία επί καθορισμένου ποσοστού επιτυχίας επίθεση. Στην περίπτωση χτυπήματος υπολογίζεται μια τυχαία, από ένα εύρος, τιμή που είναι οι πόντοι ζημιάς και αφαιρούνται από τους πόντους ζωής του στόχου. Το ακριβές σύνολο των πόντων ζωής μένει κρυφό στους παίχτες, αλλά μπορεί να εκτιμηθεί από την ενδεικτική μπάρα που σχεδιάστηκε προηγουμένως.

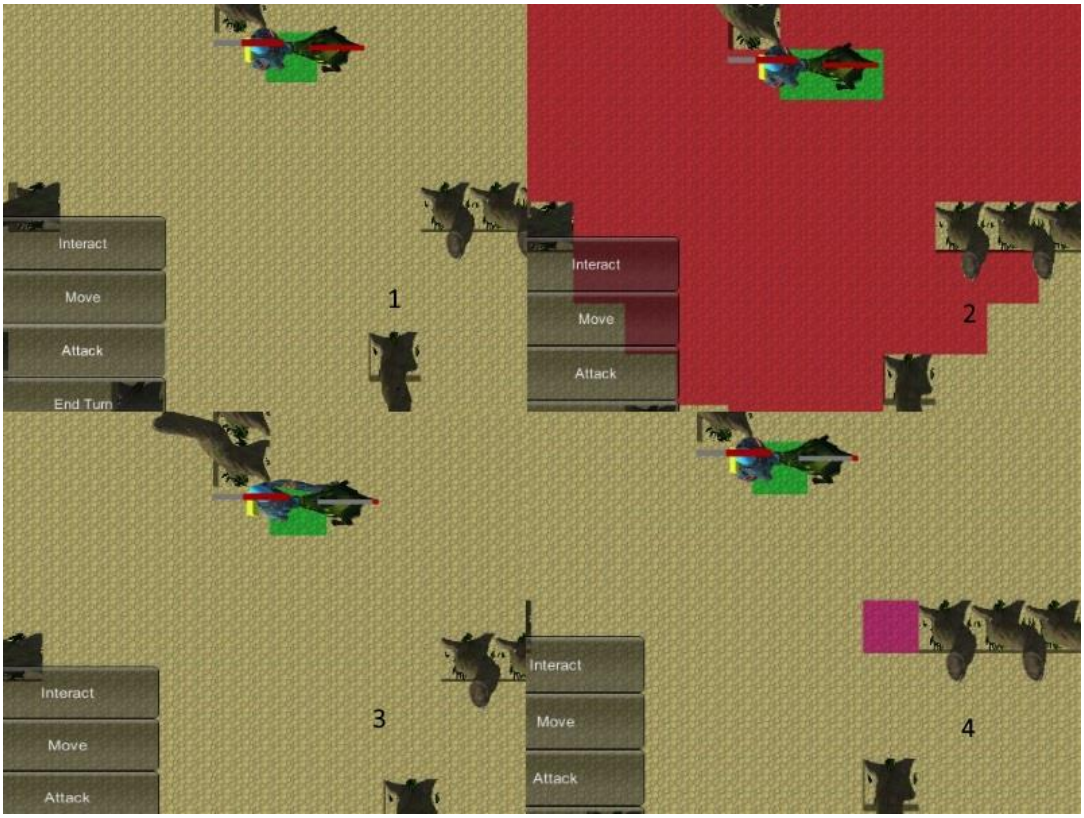
```
public void attackWithCurrentPlayer(Tile destTile){
    Debug.Log ("attacking position is (" + destTile.gridPosition + ")");
    turnAround (destTile);
    if (destTile.visual.transform.GetComponent<Renderer>().materials[0].color != Color.white && !destTile.impassable) {
        Player target = null;
        foreach (Player p in players) {
            if (p.gridPosition == destTile.gridPosition) {
                target = p;
            }
        }
        if (target != null) {
            players [currentPlayerIndex].animationAttackBool = true;
            players [currentPlayerIndex].actionPoints--;
            removeTileHighlights();
            players [currentPlayerIndex].attack = false;
            players [currentPlayerIndex].animationAttackBool = true;
            bool hit = Random.Range (0.0f, 1.0f) <= players [currentPlayerIndex].attackchance;
            if (hit) {
                int amountOfDamage = (int)Mathf.Floor (players [currentPlayerIndex].dmg
                    + Random.Range (1, players [currentPlayerIndex].dmgdie));
                target.HP -= amountOfDamage;
                target.Health.fillAmount = target.HP/target.maxHP;
                if (target.HP <= 0) {
                    players [currentPlayerIndex].animationAttackBool = false;
                }
                if (target.playerindex == 1 && target.HP <= 0) {
                    target.gridPosition.y = target.gridPosition.y - 50;
                    numofdeaduserplayers++;
                    if (numofdeaduserplayers == 4) {
                        target.animationDeadBool = true;
                        gameOver ();
                    }
                }
                if (target.playerindex == 2 && target.HP <= 0) {
                    target.gridPosition.y = target.gridPosition.y - 50;
                    numofdeadaiplayers++;
                    if (numofdeadaiplayers == 3) {
                        numofdeadaiplayers = 0;
                        generateAIPlayers ();
                    }
                }
            }
        }
    }
}
```

Εικόνα 3.5.1: Κώδικας επίθεσης επιλεγμένου παίχτη

Μέσω αυτής της διαδικασίας δίνεται η δυνατότητα πολλών και διαφόρων γεγονότων να συμβούν με το πώς επηρεάζονται οι παίχτες. Επειδή την ίδια συνάρτηση χρησιμοποιούν και οι παίχτες του υπολογιστή, χρησιμοποιείται συνθήκη που ελέγχει τον αριθμό των νεκρών παιχτών του χρήστη και τερματίζει το παιχνίδι όταν πεθαίνουν όλοι. Ώστε να παραμένουν τα σώματα των νεκρών παιχτών πάνω στο χάρτη αλλά δίχως να εμποδίζουν τις κινήσεις, με το που μηδενιστούν οι πόντοι ζωής του στόχου αυτός μετατίθεται εκτός του πραγματικού χάρτη αλλά με το μοντέλο να παραμένει στη θέση του. Μπορούν επίσης να καλεστούν άλλες συναρτήσεις όπως ο ερχομός νέων εχθρών όταν κάποιοι από αυτούς σκοτωθούν.

Παρόμοιες συναρτήσεις υπάρχουν και για την χρήση αντικειμένων. Με μόνη διαφορά ότι το «χτύπημα» είναι συνέχεια επιτυχές ώστε οι παίχτες μπορούν να αλληλοεπιδρούν με αντικείμενα, ανοίγοντας για παράδειγμα πόρτες.

Τέλος το ίδιο γίνεται και με την δυνατότητα θεραπείας από τον παίχτη θεραπευτή. Σε αυτή την περίπτωση το «χτύπημα» είναι πάντα επιτυχία και ο παίχτης, αντί να αφαιρεί, προσθέτει στους πόντους ζωής των στόχων του.



Εικόνα 3.5.2: Αναπαράσταση μάχης παιχτών

3.6 Τελικές Ρυθμίσεις

3.6.1 Δημιουργία παιχτών και αντικειμένων

Εφόσον έχουν αναπτυχθεί όλοι οι μηχανισμοί απαραίτητοι για να υπάρχει παιχνίδι έτσι όπως αρχικά σχεδιάστηκε, γίνονται κάποιες τελικές ρυθμίσεις που καθιστούν το παιχνίδι σε κατάσταση ικανή να τρέξει. Για αρχή φτιάχνεται συνάρτηση που εμφανίζει τον παίχτη στο χάρτη. Η συνάρτηση δημιουργεί ένα προκαθορισμένο από τον προγραμματιστή μοντέλο στο θεμιτό τετράγωνο του οπτικού χάρτη και του προσδίδει το αντίστοιχο κελί του χάρτη. Η συνάρτηση καλείται κατά την έναρξη του παιχνιδιού.

```
void generateUserPlayers(){
    UserPlayer player;
    player = ((GameObject)Instantiate (UserPlayerPrefab,
        new Vector3 (16 - Mathf.Floor (mapSize / 2), 1.5f, -25 + Mathf.Floor (mapSize / 2)),
        Quaternion.Euler (new Vector3 ()))).GetComponent<UserPlayer> ();
    player.gridPosition = new Vector2 (16, 25);
    player.playerName = "Player1";
    player.playerindex = 1;
    players.Add (player);
```

Εικόνα 3.6.1.1: Κώδικας δημιουργίας παίχτη στην πίστα

Η συνάρτηση της εικόνας 3.6.1 δημιουργεί έναν παίχτη του χρήστη και του αλλάζει κάποιες προ υπάρχουσες τιμές. Αυτές οι αλλαγές βοηθάνε να δοθούν διαφορετικά χαρακτηριστικά σε κάθε παίχτη. Στη συνάρτηση μπορούν να προστεθούν όσοι παίχτες θέλει ο προγραμματιστής απλά επαναλαμβάνοντας για τον καθένα τον κώδικα με τις απαραίτητες αλλαγές στα στοιχεία και τη θέση.

Παρόμοιες συναρτήσεις συντάσσονται και για την δημιουργία παιχτών του υπολογιστή και των αντικειμένων που θα υπάρχουν στην σκηνή και επικαλούνται στο ξεκίνημα του παιχνιδιού.

3.6.2 Έλεγχος κάμερας

Για διευκόλυνση στην εξερεύνηση του χάρτη από το χρήστη γράφεται κώδικας που επιτρέπει την κίνηση της κάμερας. Με το πάτημα προκαθορισμένων από τον προγραμματιστή κουμπιών του πληκτρολογίου ο χρήστης μπορεί να κινεί την κάμερα προς την επιθυμητή κατεύθυνση και του δίνεται η δυνατότητα να την φέρει κοντινότερα ή να την απομακρύνει από τον χάρτη. Τίθενται όρια που

καθορίζουν μέχρι που μπορεί να φτάσει η κάμερα αλλιώς θα ήταν ικανή να ταξιδεύει στον άπειρο κενό χώρο του παιχνιδιού.

```
void Update () {
    Vector3 dir = new Vector3 ();
    if (Input.GetKey (KeyCode.W))
        dir.y += 1.0f;
    if (Input.GetKey (KeyCode.S))
        dir.y -= 1.0f;
    if (Input.GetKey (KeyCode.A))
        dir.x -= 1.0f;
    if (Input.GetKey (KeyCode.D))
        dir.x += 1.0f;
    if (Input.GetKey (KeyCode.Q))
        dir.z -= 1.0f;
    if (Input.GetKey (KeyCode.E))
        dir.z += 1.0f;
    dir.Normalize ();
    transform.Translate (dir * speed * Time.deltaTime);
    transform.position = new Vector3 (Mathf.Clamp (transform.position.x, -15, 15),
        Mathf.Clamp (transform.position.y, 10, 20),
        Mathf.Clamp (transform.position.z, -15, 15));
}
```

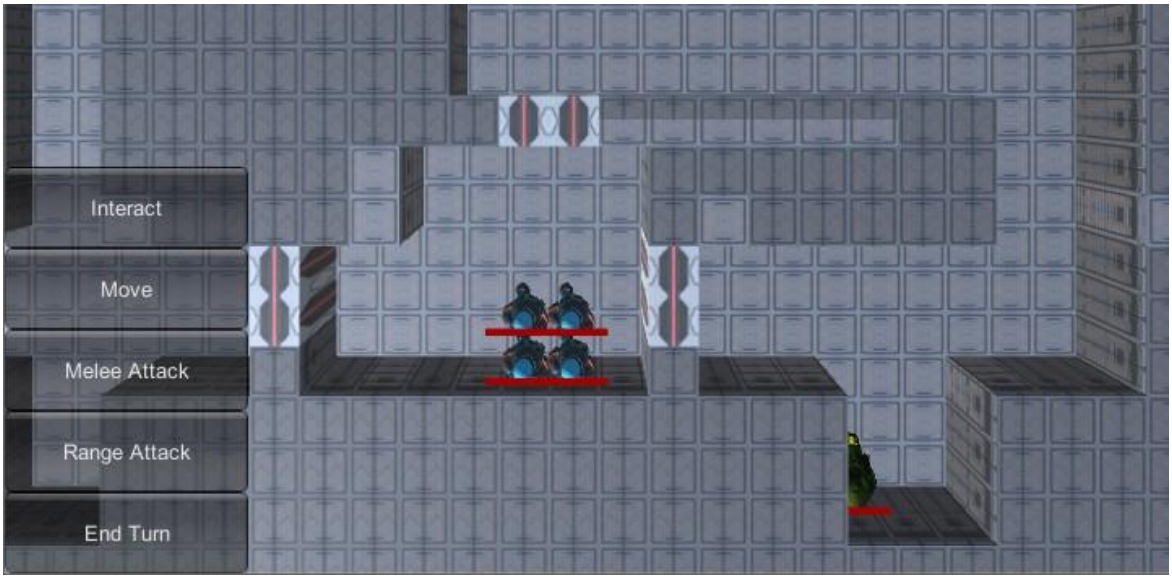
Εικόνα 3.6.2.1: Κώδικας κίνησης κάμερας

Από την οθόνη σκηνής μπορεί να καθοριστεί η αρχική θέση της κάμερας όταν ξεκινάει μια σκηνή.

3.6.3 Ολοκλήρωση παιχνιδιού

Έχοντας τελειώσει με όλα τα απαραίτητα στοιχεία του αρχικού σχεδιασμού του παιχνιδιού, μπορεί να ειπωθεί ότι το παιχνίδι ως μηχανισμοί είναι έτοιμο. Πλέον ότι άλλο προστεθεί είναι ως επί το πλείστον θέμα παρουσίασης. Μπορούν να προστεθούν παραπάνω σκηνές ως πίστες, άλλα μοντέλα για διαφορετική αίσθηση, παραπάνω είδη παιχτών ή εχθρών. Εφόσον έχουν ολοκληρωθεί οι βασικοί μηχανισμοί μπορούν γύρω από αυτούς να δημιουργηθούν πολλά διαφορετικά, σύμφωνα με την αγορά, παιχνίδια του ίδιου προτύπου.

Μπορούν επίσης να προστεθούν περαιτέρω λειτουργίες στα πρότυπα αυτών που έχουν ήδη σχεδιαστεί ή εντελώς νέες που χρησιμοποιούν τις ήδη υπάρχουσες σαν βάση. Οι δυνατότητες είναι πολλές και περιορίζονται μόνο στον χρόνο, την ικανότητα και το σύνολο του ανθρωπίνου δυναμικού που θα αναλάβει τη δημιουργία ενός παιχνιδιού.



Εικόνα 3.6.3.1: Τα βασικά στοιχεία του παιχνιδιού είναι έτοιμα

Για να μπορεί όμως ένα παιχνίδι να βγει στην αγορά πρέπει να έχει μερικά παραπάνω στοιχεία. Τα σχεδόν απαραίτητα για τη διευκόλυνση του χρήστη είναι τα μενού επιλογών. Άλλα στοιχεία που προτιμώνται και εκτιμώνται είναι ο ήχος και η δυνατότητα παιχνιδιού πολλών παιχτών. Το Unity προσφέρει τέτοιες δυνατότητες, αλλού περισσότερο, όπως είναι η δημιουργία μενού, και αλλού λιγότερο, όπως το παιχνίδι πολλαπλών παιχτών. Στο επόμενο λοιπόν κεφάλαιο θα αναλυθούν αυτές οι προσθήκες και πως συμπληρώθηκαν στο παιχνίδι.

ΚΕΦΑΛΑΙΟ 4

ΑΝΑΠΤΥΞΗ ΕΠΙΠΛΕΟΝ ΣΤΟΙΧΕΙΩΝ ΠΑΙΧΝΙΔΙΟΥ

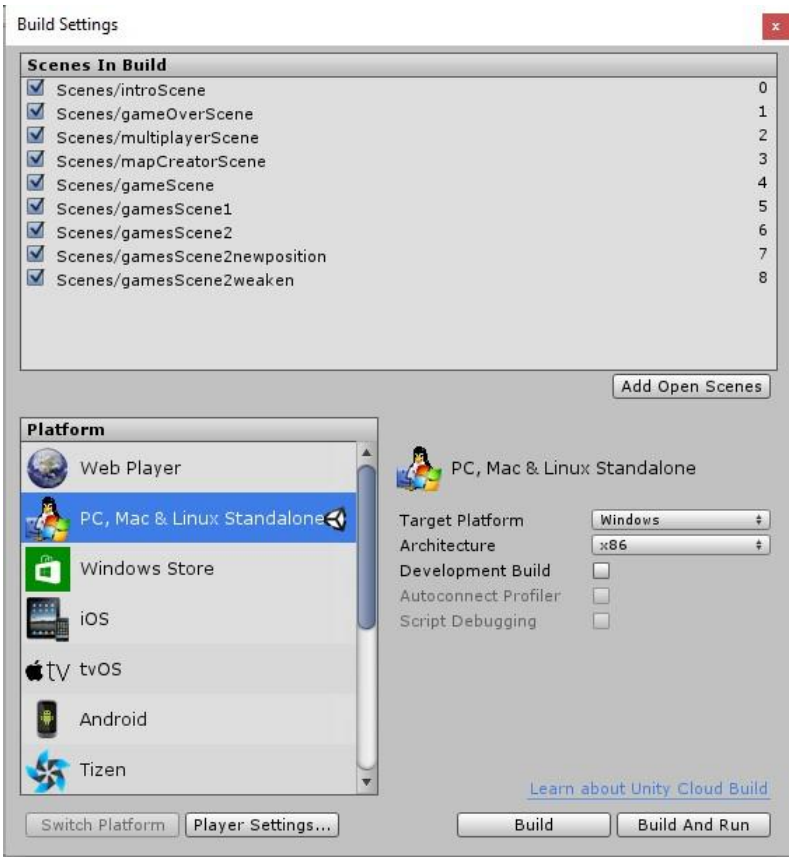
Τα βιντεοπαιχνίδια εκτός από τους βασικούς μηχανισμούς τους απαρτίζονται και από άλλα στοιχεία όπως μενού, μουσική, δυνατότητα πολλαπλών παιχτών και διάφορα επίπεδα. Κάποια βιντεοπαιχνίδια φτάνουν στο σημείο να έχουν μέχρι και άλλα μικρότερα βιντεοπαιχνίδια εντός τους. Πολλά από αυτά τα στοιχεία, είτε βρίσκονται στο συνολικό πακέτο για τη διευκόλυνση του χρήστη, είτε για λόγους αισθητικής, είτε για περαιτέρω ικανοποίηση του, μπορούν να θεωρηθούν πλέον δεδομένα και απαραίτητα στη σύνολο του παιχνιδιού. Στο κεφάλαιο αυτό θα αναλυθεί η ανάπτυξη τέτοιων στοιχείων για το παιχνίδι που ολοκληρώθηκε στο προηγούμενο στο επίπεδο που επιτρέπει αυτό αλλά και η μηχανή Unity.

4.1 Ανάπτυξη εισαγωγικού μενού

Τα περισσότερα παιχνίδια έχουν ένα αρχικό μενού που αποτελεί το πρώτο πράγμα με το οποίο έρχεται σε επαφή ο χρήστης όταν εκκινεί την εφαρμογή. Σε αυτό το μενού ο χρήστης έχει συνήθως τη δυνατότητα να αλλάξει κάποιες ρυθμίσεις του παιχνιδιού, όπως η ανάλυση στην οποία θα τρέχει, να δει το ή και να αλλάξει τον χειρισμό καθώς και να περιηγηθεί σε άλλες λειτουργίες του ή να κλείσει την εφαρμογή.

Για τη δημιουργία του μενού φτιάχνεται μια νέα σκηνή, και αν το μενού προορίζεται σαν αρχικό, πρέπει και η σκηνή να ορίζεται ως πρώτη στη σειρά σκηνών στις ρυθμίσεις χτισίματος που είναι προσβάσιμες από την επιλογή «Αρχείο». Σε αυτές τις ρυθμίσεις πρέπει να προστεθούν όλες οι σκηνές τις οποίες ο προγραμματιστής σκοπεύει να συμπεριλάβει στο παιχνίδι στην τελική του μορφή. Η σειρά, εκτός από την πρώτη επιλογή, είναι προαιρετική επειδή ο προγραμματιστής μπορεί να επικαλεστεί στον κώδικα άλλες σκηνές είτε με τον αριθμό σήμανσης που δίνεται στις ρυθμίσεις, είτε με το όνομα της ίδιας της σκηνής, μέθοδος που προτιμάται για ευκολότερη αλλαγή εκτός σειράς. Από τις ίδιες ρυθμίσεις μπορεί να καθοριστεί η πλατφόρμα ή το λειτουργικό για το οποίο προορίζεται να τρέξει το παιχνίδι, τι επιλογές θα έχει ο χρήστης για να επηρεάσει

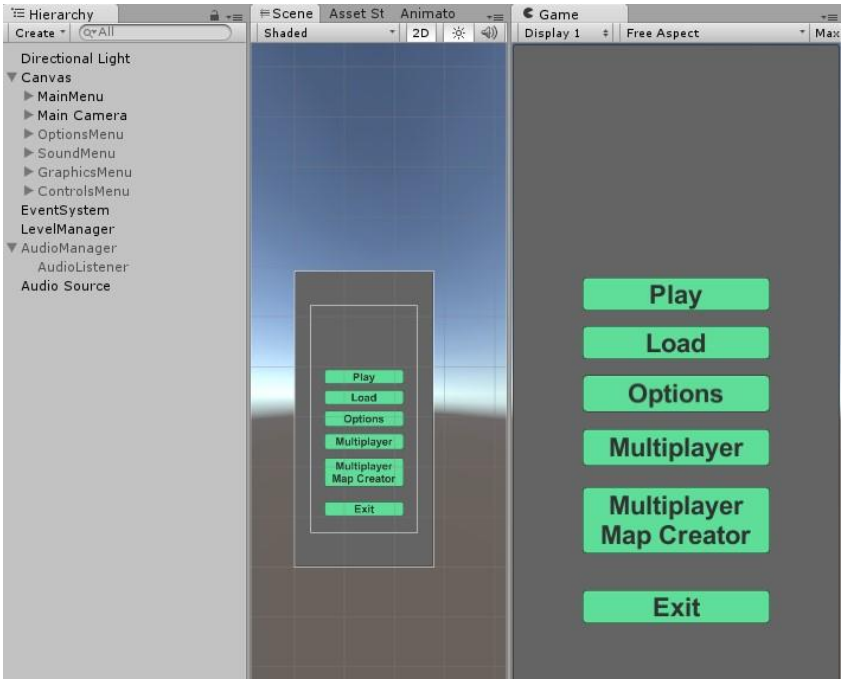
το παιχνίδι ως προς τη λειτουργία του, καθώς και αν το παιχνίδι θα είναι σε τελική μορφή ή μορφή ελέγχου.



Εικόνα 4.1.1: Ρυθμίσεις χτισίματος

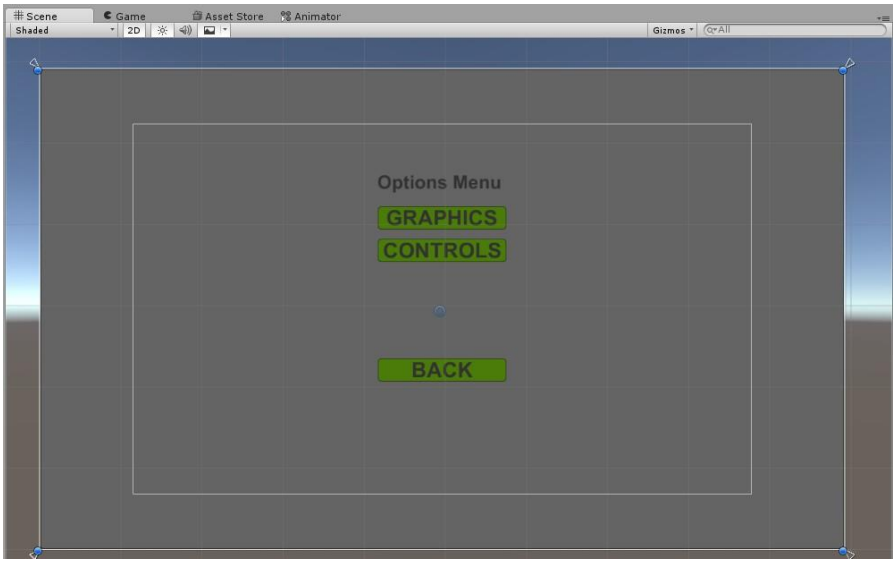
Στη σκηνή που έχει δημιουργηθεί επιλέγεται η δισδιάστατη απεικόνιση στην οθόνη σκηνής. Δημιουργείται ένα αντικείμενο καμβά στο οποίο προστίθενται η κεντρική κάμερα, μια εικόνα ως πίσω οθόνη και κουμπιά επιλογών όπως έναρξη παιχνιδιού, ρυθμίσεις και ότι θέλει ο προγραμματιστής να επιτρέψει στο χρήστη να κάνει. Όλων αυτών τη μορφή μπορεί να επεξεργαστεί τόσο όσο επιθυμεί ο προγραμματιστής.

Πολλαπλά μενού μπορούν να δημιουργηθούν που θα ενεργοποιούνται με τα πατήματα των επιλογών του αρχικού. Μια πληθώρα επιλογών, όπως πρόσθεση κειμένου, κουτιά σήμανσης, μπάρες κύλισης διατίθενται από το Unity. Τα παραπάνω μενού θα πρέπει όμως να απενεργοποιηθούν από εμφάνιση η οποία θα ρυθμίζεται από κώδικα που θα ανταποκρίνεται στις επιλογές του αρχικού μενού.



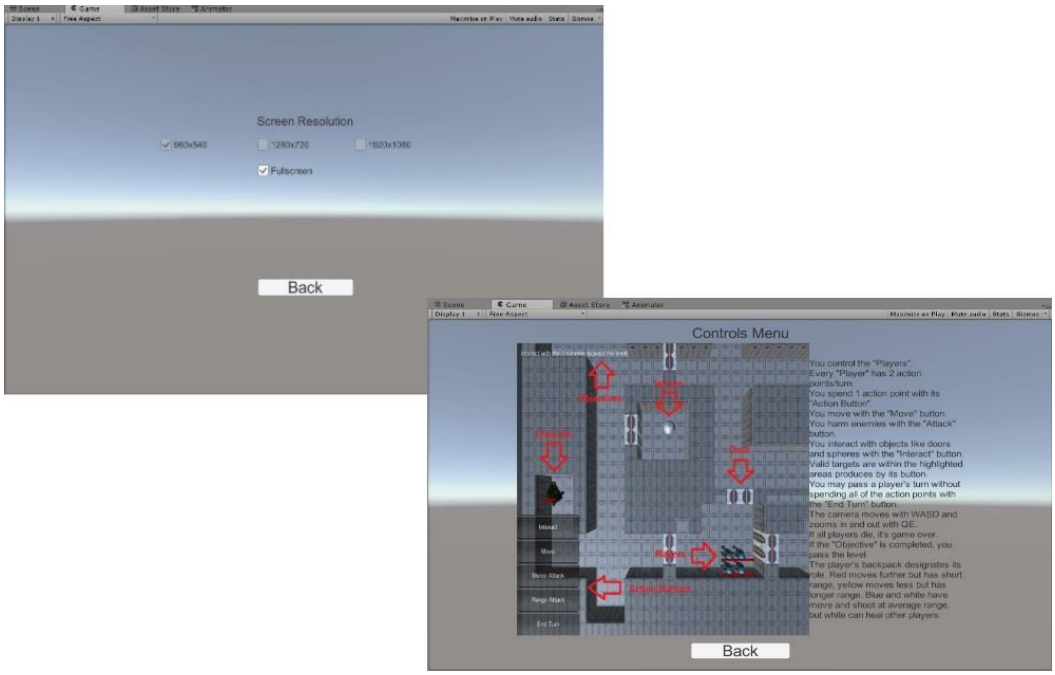
Εικόνα 4.1.2: Εισαγωγικό μενού σε όλες τις οθόνες

Στην εικόνα 4.1.2 φαίνονται οι έξι επιλογές που θα έχει ο χρήστης από το αρχικό μενού. Θα μπορεί να εκκινήσει το παιχνίδι, να φορτώσει το τελευταίο επίπεδο που έσωσε, ανοίξει το μενού ρυθμίσεων, να ξεκινήσει ένα παιχνίδι πολλαπλών παιχτών, να φτιάξει ένα χάρτη για αυτό το παιχνίδι και να κλείσει την εφαρμογή. Οι περισσότερες από αυτές τις επιλογές μεταφέρουν τον παίχτη σε κάποια άλλη σκηνή, εκτός από αυτή των ρυθμίσεων που φορτώνει ένα κρυμμένο από την αρχική οθόνη μενού.



Εικόνα 4.1.3: Μενού επιλογών στην οθόνη σκηνής

Το μενού ρυθμίσεων όπως φαίνεται στην εικόνα 4.1.3 έχει δύο επιλογές για τον χρήστη. Στην πρώτη του δίνεται η δυνατότητα να αλλάξει την ανάλυση που θα τρέχει το παιχνίδι και αν θα πιάνει όλη την οθόνη ή θα λειτουργεί σε μορφή παραθύρου. Στην δεύτερη ο χρήστης μπορεί να βρει οδηγίες για τους χειρισμούς του παιχνιδιού πριν ξεκινήσει να παίζει. Τέλος μπορεί να επιστρέψει στο αρχικό μενού.



Εικόνα 4.1.4: Μενού αλλαγής διαστάσεων και οδηγού ελέγχου χαρακτήρων

Για τη σωστή λειτουργία των μενού θα πρέπει να συνταχθούν συναρτήσεις που καλούνται στο πάτημα των κουμπιών. Για τις περισσότερες απλά θα προστεθεί η αλλαγή σκηνής και η λειτουργία εξόδου για το κλείσιμο της εφαρμογής.

```
public void LoadScene(string name){
    SceneManager.LoadScene (name);
}

public void MultiplayerScene(string name){
    SceneManager.LoadScene (name);
}

public void QuitGame(){
    Application.Quit ();
}

public void Load(){
    SceneManager.LoadScene (PlayerPrefs.GetString ("currentscenesave"));
}
```

Εικόνα 4.1.5: Κώδικας αλλαγής σκηνής για το αρχικό μενού

Για το μενού ρυθμίσεων θα δίνεται και αλλάζει ανάλογα στη συνάρτηση μια λογική τιμή που θα ρυθμίζει την εμφάνιση του μενού. Η ίδια συνάρτηση θα χρησιμοποιηθεί και για την εμφάνιση των ρυθμίσεων γραφικών, τις οδηγίες και την επιστροφή σε προηγούμενο μενού.

Από τα τετράγωνα επιλογής του μενού γραφικών ρυθμίσεων θα δίνεται ένας ακέραιος αριθμός που θα διαιρείται το λόγο διάστασης της οθόνης που επιλέγει ο προγραμματιστής με το αποτέλεσμα να χρησιμοποιείται για το μήκος ενώ ο ακέραιος για το πλάτος της ανάλυσης. Η πλήρης οθόνη θα δέχεται μια λογική τιμή που θα ενεργοποιεί την μέγιστη ανάλυση, κρατώντας παγωμένη την προηγούμενη ανάλυση για επιστροφή σε αυτή σε περίπτωση απενεργοποίησης.

```
public void GraphicsMenu(bool clicked){
    if (clicked == true) {
        graphicsMenu.gameObject.SetActive (clicked);
        optionsMenu.gameObject.SetActive (false);
    } else {
        graphicsMenu.gameObject.SetActive (clicked);
        optionsMenu.gameObject.SetActive (true);
    }
}

public void SetScreenResolution(int i){
    if (resolutionToggles [i].isOn) {
        activeScreenResIndex = i;
        float aspectRatio = 16 / 9f;
        Screen.SetResolution (screenWidths [i], (int)(screenWidths [i] / aspectRatio), false);
        PlayerPrefs.SetInt ("screen res index", activeScreenResIndex);
        PlayerPrefs.Save ();
    }
}

public void SetFullscreen(bool isFullscreen){
    for (int i = 0; i < resolutionToggles.Length; i++) {
        resolutionToggles [i].interactable = !isFullscreen;
    }
    if (isFullscreen) {
        Resolution[] allResolutions = Screen.resolutions;
        Resolution maxResolution = allResolutions [allResolutions.Length - 1];
        Screen.SetResolution (maxResolution.width, maxResolution.height, true);
    } else {
        SetScreenResolution (activeScreenResIndex);
    }
    PlayerPrefs.SetInt ("fullscreen", ((isFullscreen) ? 1 : 0));
    PlayerPrefs.Save ();
}
```

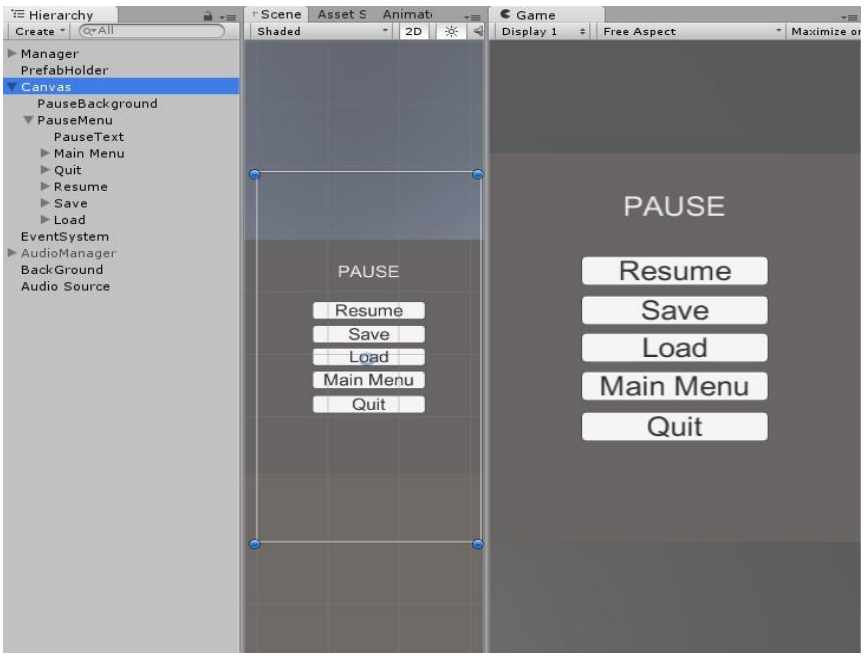
Εικόνα 4.1.6: Κώδικας ρυθμίσεων ανάλυσης

4.2 Ανάπτυξη μενού παύσης

Όπως με το αρχικό μενού, έτσι φτιάχνεται και το μενού παύσης, με μόνη διαφορά ότι αντί να είναι στη δικιά του σκήνη, θα βρίσκεται στις σκηνές του παιχνιδιού. Κώδικας ελέγχει για το πάτημα κάποιου προκαθορισμένου από τον προγραμματιστή κουμπί του πληκτρολογίου για να καλέσει την συνάρτηση που ενεργοποιεί το μενού και παγώνει το χρόνο στο παιχνίδι.

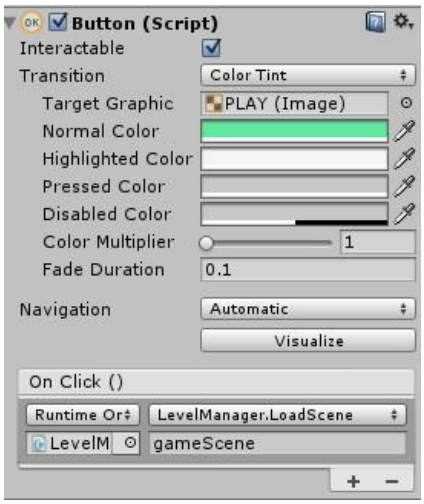
```
void Update () {
    if (Input.GetKeyDown (KeyCode.Escape)) {
        Pause ();
    }
}

public void Pause(){
    if (canvas.gameObject.activeInHierarchy == false) {
        canvas.gameObject.SetActive (true);
        Time.timeScale = 0;
    } else {
        canvas.gameObject.SetActive (false);
        Time.timeScale = 1;
    }
}
```

Εικόνα 4.2.1: Κώδικας μενού παύσης



Εικόνα 4.2.2: Μενού παύσης σε όλες τις οθόνες

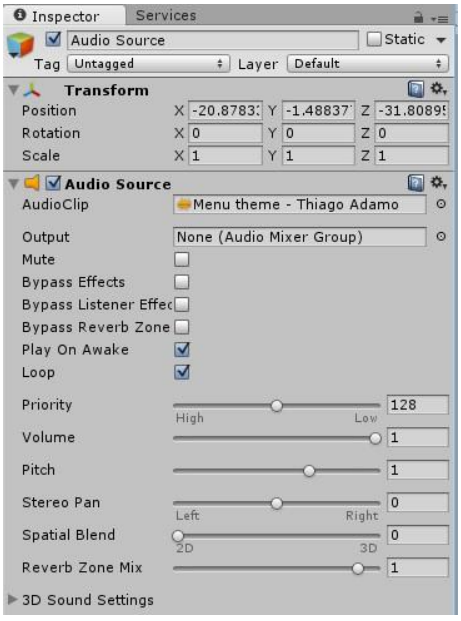
Για το κάλεσμα των συναρτήσεων με το πάτημα των κουμπιών των μενού προστίθεται απλά ο κώδικας και η συνάρτηση που χρησιμοποιείται στην επιλογή πατήματος σε κάθε αντικείμενο κουμπιού.



Εικόνα 4.2.3: Κάλεσμα συναρτήσεων από τα κουμπιά των μενού

4.3 Προσθήκη ήχου

Τα περισσότερα παιχνίδια χρησιμοποιούν ήχους στη μορφή μουσικής ή εφέ. Σε πολλά παιχνίδια ο ήχος παίζει και σημαντικό ρόλο για τον προσανατολισμό του παίχτη στο χώρο. Στο παιχνίδι προστίθεται μουσική σε κάθε σκηνή, οπότε δημιουργείται ένα αντικείμενο πηγής ήχου που δέχεται το αρχείο μουσικής και ένα στοιχείο ακουστικού προστίθεται στο αντικείμενο της κύριας κάμερας. Η μουσική ρυθμίζεται να παίζει σε επανάληψη και με την έναρξη της σκηνής.



Εικόνα 4.3.1: Αντικείμενο με στοιχείο ήχου

4.4 Παιχνίδι πολλών παιχτών σε τοπικό δίκτυο

4.4.1 UNET

Το UNET(Unity Networking System) είναι η προσπάθεια των δημιουργών του Unity να δώσουν την ικανότητα στους χρήστες της μηχανής να αναπτύξουν διαδικτυακά παιχνίδια. Η μηχανή λοιπόν προσφέρει αντικείμενα που περιέχουν ένα μεγάλο σύνολο από έτοιμους κώδικες που αναλαμβάνουν μόνοι τους την διαδικτυακή σύνδεση δίχως να χρειάζεται την ενασχόληση του χρήστη, καθώς και βιβλιοθήκες με έτοιμες εντολές και συναρτήσεις που μπορεί ο χρήστης να βρει και χρησιμοποιήσει με ευκολία.

Το UNET είναι ένα σχετικά νέο σύστημα που ακόμα αναπτύσσεται και δεν προσφέρεται πλήρης κάλυψη των δυνατοτήτων του. Επειδή όμως κάποιες από τις απλούστερες λειτουργίες του μπορούν να συμπληρωθούν στο παιχνίδι, προστέθηκαν για να αναλυθεί και αυτό το κομμάτι της μηχανής.

4.4.2 Ανάπτυξη προσθήκης πολλαπλών παιχτών σε τοπικό δίκτυο

Το πρώτο πράγμα που χρειάζεται να γίνει για να αναπτυχθεί η σύνδεση πολλαπλών παιχτών σε ένα τοπικό δίκτυο είναι η δημιουργία ενός έτοιμου από το Unity αντικείμενο σε μια νέα σκηνή, το «Network Manager». Αυτό το αντικείμενο περιέχει τους απαραίτητους κώδικες για τη σύνδεση των παιχτών στο δίκτυο. Μπορεί να δεχθεί μια σκηνή που θα τρέχει ενώ δεν υπάρχει σύνδεση και μια στην οποία θα τρέχει το διαδικτυακό παιχνίδι. Δέχεται και το μοντέλο του βασικού παίχτη που θα δημιουργήσει στο παιχνίδι.

Εικόνα 4.4.2.1: GUI σύνδεσης τοπικού δικτύου

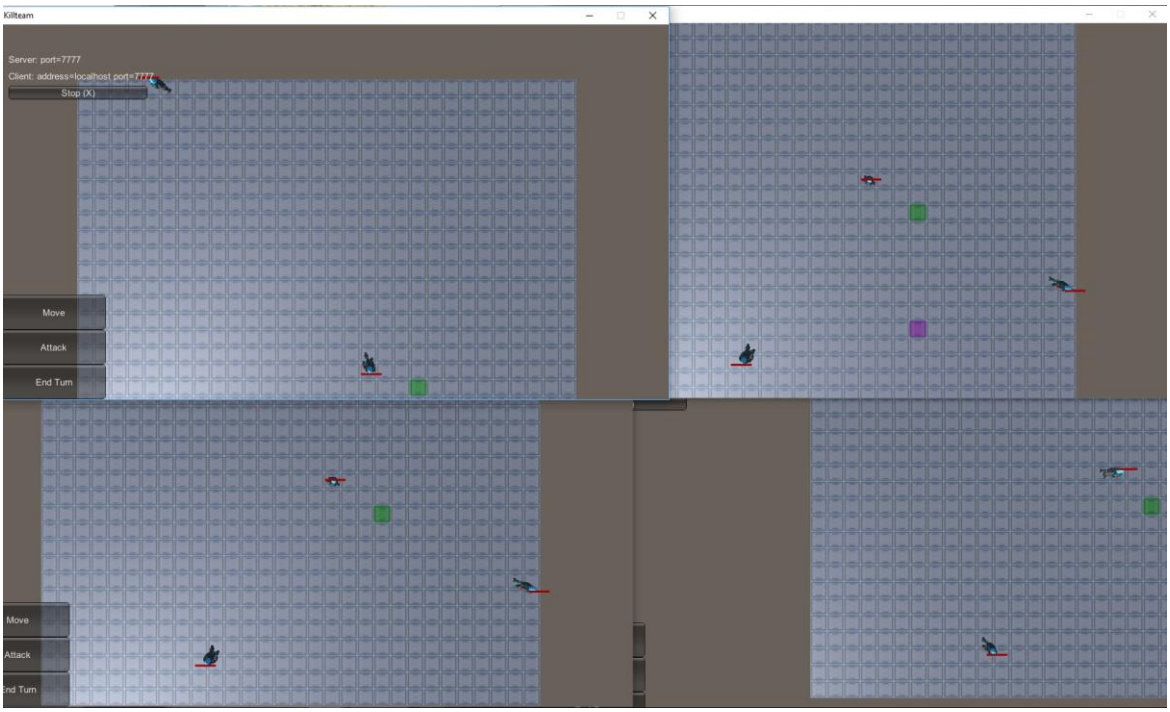
Το αντικείμενο προσφέρει και ένα GUI, αυτό που φαίνεται στην εικόνα 4.4.2.1, το οποίο επιτρέπει σε έναν χρήστη να λειτουργεί ως εξυπηρετητής και οι υπόλοιποι παίχτες να συνδέονται ως εξυπηρετούμενοι.

Εφόσον το μοντέλο του παίχτη είναι αυτό που θα πρέπει να ενημερώνεται συνεχώς στο χώρο του παιχνιδιού, γίνονται οι απαραίτητες αλλαγές στον κώδικα του παίχτη. Ξεκινώντας με την προσθήκη της βιβλιοθήκης που προσφέρει το Unity για την διαχείριση της σύνδεσης προστίθεται η εντολή συγχρονισμού της θέσης του παίχτη, ο server προσδίδει ένα αναγνωριστικό νούμερο σε κάθε παίχτη που μπαίνει στο παιχνίδι, το βάζει σε σειρά με τους υπόλοιπους και τον εμφανίζει σε μια τυχαία θέση πάνω στον χάρτη και ο παίχτης ενημερώνει τον server για τις κινήσεις του στο παιχνίδι και τη θέση του.

```
[SyncVar]
public Vector2 gridPosition=Vector2.zero;

[SyncVar]
public int id;

[Command]
public void CmdSetId(int newId){
    id = newId;
}

[ClientRpc]
public void RpcSyncNetManagerPlayerIds(string serializedList){
    var lst = JsonUtility.FromJson<int[]> (serializedList);
    var newOrderedList = new List<NetworkPlayer>();
    foreach (var id in lst) {
        var added = GameObject.FindObjectsOfType<NetworkPlayer> ().Where (x => x.id == id).First ();
        newOrderedList.Add (added);
    }
    NetManager.instance.players = newOrderedList;
}

[Command]
public void CmdSyncNetManagerPlayerIds(string serializedList){
    var lst = JsonUtility.FromJson<int[]> (serializedList);
    var newOrderedList = new List<NetworkPlayer> ();
    foreach (var id in lst) {
        var added = GameObject.FindObjectsOfType<NetworkPlayer> ().Where (x => x.id == id).First ();
        newOrderedList.Add (added);
    }
    NetManager.instance.players = newOrderedList;
}
[Command]
public void CmdMoveCurrentPlayer(string serializedGridPosition){
    RpcMoveCurrentPlayer (serializedGridPosition);
    NetManager.instance.moveCurrentPlayer (serializedGridPosition);
}
[ClientRpc]
public void RpcMoveCurrentPlayer(string serializedGridPosition){
    NetManager.instance.moveCurrentPlayer (serializedGridPosition);
}
[Command]
public void CmdNextTurn(){
    if (NetManager.instance.currentPlayerIndex + 1 < NetManager.instance.players.Count) {
        NetManager.instance.currentPlayerIndex++;
    } else {
        NetManager.instance.currentPlayerIndex = 0;
    }
}
[Command]
public void CmdSetGridPosition(Vector2 newPos){
    gridPosition = newPos;
}
[Command]
public void CmdUpdateTransformPosition(){
    transform.position = new Vector3 (gridPosition.x - Mathf.Floor
        (NetManager.instance.mapSize / 2), 1.5f, -gridPosition.y + Mathf.Floor (NetManager.instance.mapSize / 2));
}
```

Εικόνα 4.4.2.2: Βασικές αλλαγές στον κώδικα του παίχτη

Τη χρονική περίοδο που αναπτύχθηκε η προσθήκη παιχνιδιού σε τοπικό δίκτυο το Unity επέτρεπε την υποστήριξη μέχρι τεσσάρων παιχτών και δεν δινόταν δυνατότητα δημιουργίας εχθρικών προς τους παίχτες χαρακτήρων ελεγχόμενους από τον υπολογιστή και ο συγχρονισμός δεν ήταν πάντα σταθερός ή άμεσος. Έτσι το παιχνίδι είναι της μορφής αρένας, με κάθε παίχτη εναντίων των υπολοίπων.



Εικόνα 4.4.2.3: Παιχνίδι ανάμεσα σε πολλούς παίχτες

4.5 Δημιουργία περισσοτέρων επιπέδων

Τα περισσότερα παιχνίδια έχουν πολλαπλά επίπεδα με νέες προκλήσεις για τους παίχτες. Στον αρχικό σχεδιασμό του παιχνιδιού, και σύμφωνα με το είδος του, ήταν η ολοκλήρωση αποστολών για την προώθηση σε νέα επίπεδα. Η ανάπτυξη τους είναι μια απλή διαδικασία όπου απαιτεί την δημιουργία νέων σκηνών και αντικειμένων που θα κάνουν χρήση του ήδη υπάρχοντα κώδικα με όποιες προσθήκες μπορεί να είναι απαραίτητες κυρίως για την προσθήκη των αποστολών.

Στο παιχνίδι συγκεκριμένα στο πρώτο επίπεδο ο χρήστης πρέπει να κινήσει τους παίχτες του έτσι ώστε να επιδράσει με όλα τα σφαιρικά αντικείμενα του επιπέδου για να προχωρήσει στο επόμενο. Οι εχθροί ανά κάποιες απώλειες

αυξάνονται, οπότε ο χρήστης αυξάνει τις πιθανότητες να ηττηθεί όσο δεν προσπαθεί να τελειώσει την αποστολή αλλά ασχολείται με τους εχθρούς.

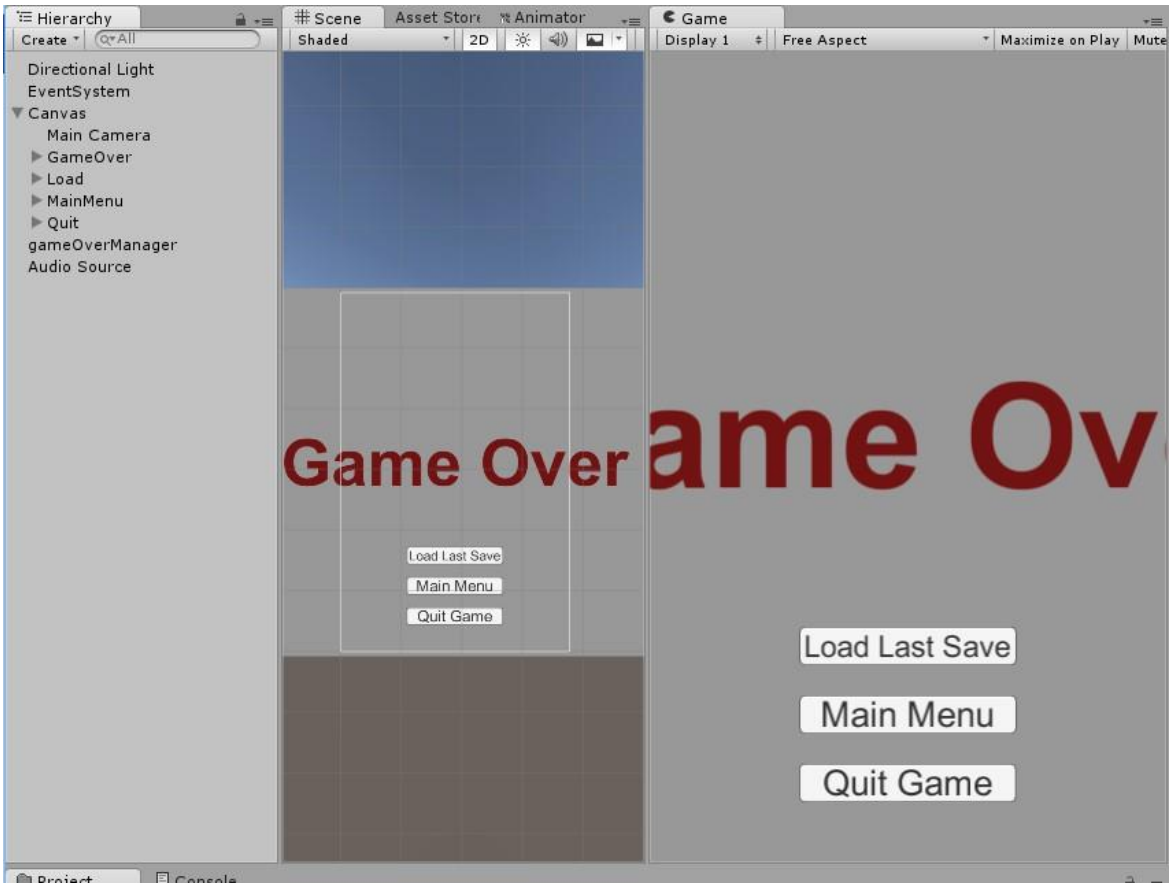
Στη δεύτερη ο παίχτης έχει να επιλέξει ανάμεσα σε δυο αποστολές. Είτε θα κινηθεί προς την διπλή πόρτα και θα συνεχίσει αμέσως στο επόμενο επίπεδο, ή θα επιδράσει με μια σφαίρα και την μονή πόρτα, όπου θα προχωρήσει σε άλλο σημείο του επόμενου επιπέδου με πιο αδύναμους εχθρούς κάνοντας το ευκολότερο. Σκοπός είναι να δυσκολευτεί ο παίχτης στην μια πίστα για να έχει όφελος στην επόμενη. Αν χρησιμοποιήσει την σφαίρα και κινηθεί στην διπλή πόρτα, το παιχνίδι τερματίζει, ενώ αν πάει απευθείας στην μονή, απλά θα ξεκινήσει την επόμενη πίστα στη νέα θέση. Σε αυτό το επίπεδο προστέθηκε ένα είδος εχθρών που μπορούν να χτυπάνε από απόσταση. Αν ένας αριθμός των εχθρών σκοτωθεί έρχεται ένα δεύτερο κύμα να επιτεθεί στον παίχτη.

Στο τρίτο επίπεδο η αποστολή είναι απλή. Ο παίχτης πρέπει να νικήσει όλους τους εχθρούς και η δυσκολία της βασίζεται καθαρά στην απόφαση που πήρε στην προηγούμενη πίστα.

Εικόνα 4.5.1: Διάφορα επίπεδα του παιχνιδιού

Τέλος δημιουργείται μια σκηνή με ένα μενού που λειτουργεί για τον τερματισμό του παιχνιδιού. Αυτή θα επικαλείται άμα ο παίχτης χάσει όλους τους

χαρακτήρες, αν δεν ολοκληρώσει την δεύτερη αποστολή όπως του υποδεικνύεται και όταν ολοκληρώσει το παιχνίδι.



Εικόνα 4.5.2: Μενού λήξης του παιχνιδιού

Από το μενού στην εικόνα 4.5.2 ο παίχτης έχει τη δυνατότητα να φορτώσει την τελευταία πίστα που αποθήκευσε και να ξαναπαίξει, να μεταφερθεί στο μενού έναρξης ή να κλείσει την εφαρμογή.

ΚΕΦΑΛΑΙΟ 5

ΟΛΟΚΛΗΡΩΣΗ ΠΑΙΧΝΙΔΙΟΥ

Με το τέλος της ολοκλήρωσης όλων των σκηνών το παιχνίδι μπορεί να χτιστεί και να παιχτεί κανονικά. Αλλά για να είναι σε κατάσταση να κυκλοφορήσει στην αγορά θα πρέπει να δοκιμαστεί και να εξεταστεί ότι δεν προσβάλει άδειες και πνευματικά δικαιώματα ώστε να μην κινδυνέψει η κυκλοφορία του νομικά. Επίσης για να είναι εμπορική επιτυχία χρειάζεται την συμβολή και άλλων εργαλείων. Όλα αυτά θα αναλυθούν σε αυτό το κεφάλαιο ώστε να ολοκληρωθεί πλήρως η ανάλυση του αναπτυξιακού κύκλου του παιχνιδιού.

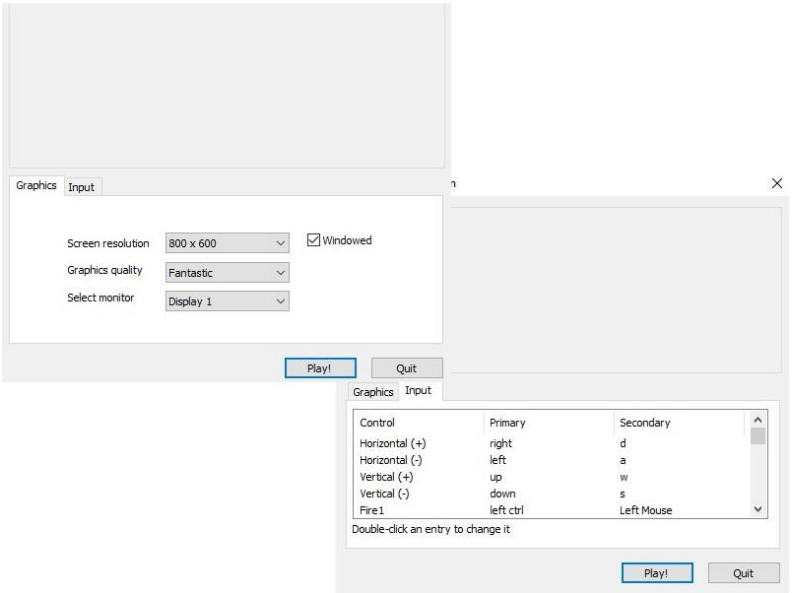
5.1 Χτίσιμο παιχνιδιού

Στο υποκεφάλαιο «4.1 Ανάπτυξη εισαγωγικού μενού» παρουσιάστηκε το μενού ρυθμίσεων χτισίματος. Από εκεί ο προγραμματιστής ορίζει τη μορφή της εφαρμογής που θα δημιουργηθεί. Μπορεί να επιλέξει την πλατφόρμα, όπως PS4 και XBOXOne, ή το λειτουργικό σύστημα, όπως Windows και Android, και να ρυθμίσει στοιχεία που το αφορούν, όπως την αρχιτεκτονική στα Windows. Δίνεται η δυνατότητα προκαθορισμού της ανάλυσης και άλλων γραφικών ρυθμίσεων. Επίσης ο προγραμματιστής μπορεί να αλλάξει το εικονίδιο με το οποίο θα εμφανίζεται η εφαρμογή.

Ένα παιχνίδι πρέπει να είναι προγραμματισμένο να λειτουργεί σε κάποια κονσόλα ή λειτουργικό σύστημα. Το Unity διαθέτει βιβλιοθήκες για τη διευκόλυνση των προγραμματιστών αλλά αυτό δε σημαίνει πως ένα παιχνίδι που σχεδιάστηκε για Windows θα λειτουργήσει με τον ίδιο τρόπο σε Android. Πρέπει να ληφθούν από τον σχεδιασμό υπόψιν οι περιορισμοί σε τεχνολογία, δυνατότητες και χειρισμό αλλιώς θα υπάρχουν προβλήματα σταθερότητας που θα καθιστούν το παιχνίδι αδύνατον να λειτουργήσει σωστά, αν λειτουργήσει. Οπότε οι επιλογές που δίνονται στο μενού ρυθμίσεων χτισίματος είναι το τελευταίο βήμα για να φτιαχτεί μια εφαρμογή συμβατή με το λειτουργικό που θα τρέξει, δεν μετατρέπουν σε σταθερές εκδόσεις παιχνίδια που δεν είναι σχεδιασμένα με αυτό το λειτουργικό κατά νου..

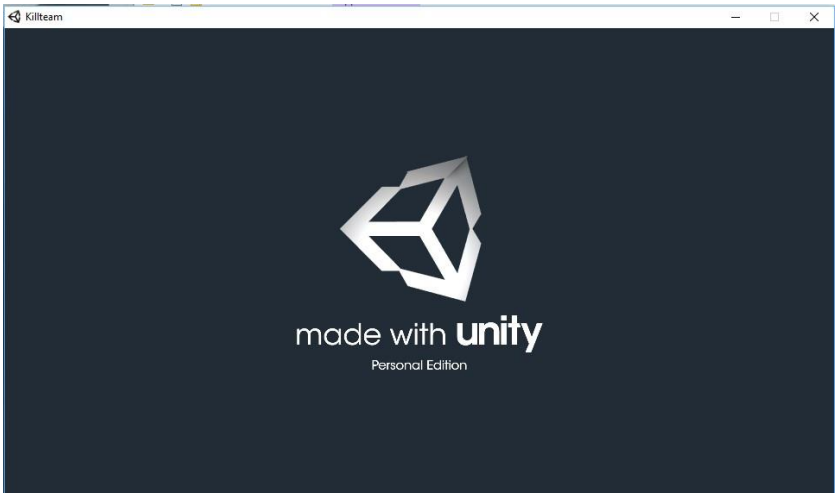
Εικόνα 5.1.1: Τελικές ρυθμίσεις για τον παίχτη

Ο προγραμματιστή μπορεί από τις ρυθμίσεις της εικόνας 5.1.1 να επιτρέψει στον χρήστη να έχει την δυνατότητα αλλαγής ανάλυσης και ρύθμισης γραφικών ή χειρισμού πριν την έναρξη της εφαρμογής. Στο παιχνίδι δεν επιτράπηκαν αυτές οι δυνατότητες εφόσον έχουν προστεθεί εντός του παιχνιδιού.



Εικόνα 5.1.2: Ρυθμίσεις πριν την έναρξη

Όταν ο προγραμματιστής πατάει το πλήκτρο χτισίματος του ζητείται ο φάκελος τοποθεσίας αποθήκευσης της εφαρμογής και αφού επιλεχθεί ξεκινάει η διαδικασία δημιουργίας. Όταν ολοκληρωθεί, στο φάκελο θα υπάρχουν μια εκτελέσιμη εφαρμογή και ένας φάκελος που θα περιέχει όλα τα δεδομένα που χρειάζεται το παιχνίδι για να τρέξει και διαβάζει από εκεί. Στο φάκελο δεν περιέχονται τα .xml αρχεία με τους χάρτες που χρησιμοποιούνται στις διάφορες σκηνές, οπότε ο προγραμματιστής πρέπει να φροντίσει να τους συμπεριλάβει ο ίδιος στο φάκελο που βρίσκεται το εκτελέσιμο. Το παιχνίδι πλέον είναι έτοιμο να τρέξει.



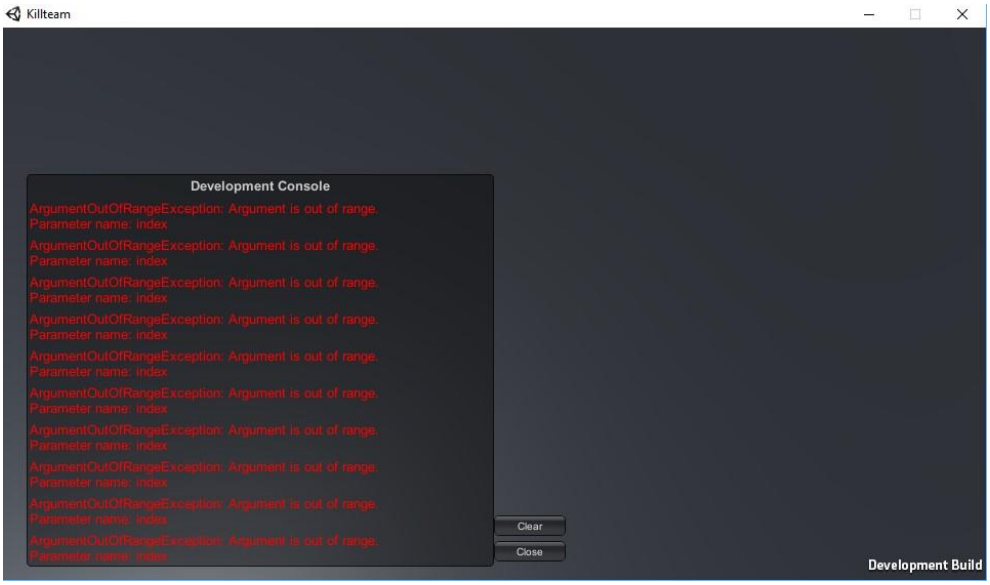
Εικόνα 5.1.3: Logo κατά την έναρξη του παιχνιδιού

5.2 Δοκιμή και αλλαγές

Οι ρυθμίσεις χτισίματος επιτρέπουν την δημιουργία ενός δοκιμαστικού ελεγκτικού εκτελέσιμου που θα εμφανίζει ένα παράθυρο το οποίο θα δείχνει μηνύματα σφαλμάτων ή προκαθορισμένα μηνύματα που έχει ορίσει ο προγραμματιστής. Λειτουργεί σαν την οθόνη κονσόλας του Unity. Έχει ο δοκιμαστής μπορεί να ελέγχει και να γνωρίζει άμεσα αν κάτι δε λειτουργεί σωστά στο παιχνίδι.

Εκτός από την οθόνη ελέγχου ο δοκιμαστής μπορεί να δει τα ίδια μηνύματα στο αρχείο output_log.txt που βρίσκεται στο φάκελο του παιχνιδιού με τα δεδομένα του. Ο έλεγχος του παιχνιδιού σε αυτή τη μορφή είναι απαραίτητος για να εξακριβωθεί και η σταθερότητα του να τρέχει στο λειτουργικό για το οποίο

προορίζεται, εφόσον μπορεί να παρουσιάσει σφάλματα που δε θα ήταν εμφανή από τις δοκιμές στην μηχανή Unity.



Εικόνα 5.2.1: Δοκιμαστική έκδοση παιχνιδιού

Στην εικόνα 5.2.1 για παράδειγμα εμφανίζει ένα συνεχές σφάλμα γιατί το παιχνίδι δεν μπορεί να δημιουργήσει την σκηνή της πρώτης πίστας. Αυτό συμβαίνει γιατί στη συγκεκριμένη περίπτωση δεν έχει προστεθεί ο αντίστοιχος χάρτης στον ίδιο φάκελο με το εκτελέσιμο.

Στο παιχνίδι της πτυχιακής είχαν παρατηρηθεί προβλήματα σταθερότητας με κάποιες συνθήκες της τεχνητής νοημοσύνης. Στην αρχή ο παίχτης του υπολογιστή θα κινούταν στην μέγιστη απόσταση που χρειαζόταν για να επιτεθεί και χτυπούσε τους παίχτες του χρήστη από εκεί. Μετά όμως από μερικούς γύρους το παιχνίδι κόλλαγε αναγκάζοντας σε επανεκκίνηση του. Έγιναν αλλαγές στις συνθήκες ώστε ο παίχτης να κινηθεί προς τους εχθρούς του και να τους επιτεθεί όταν αυτοί βρίσκονται στο πεδίο επίθεσης του. Αυτή η αλλαγή σταθεροποίησε το παιχνίδι.

Επίσης πάλι στην τεχνητή νοημοσύνη παρατηρήθηκε ότι στη δεύτερη πίστα οι παίχτες του υπολογιστή θα κινούνταν προς τις πόρτες και θα ενεργοποιούσαν έτσι τον τερματισμό της. Αυτό το πρόβλημα παρατηρήθηκε μετά από πολλές δοκιμές του παιχνιδιού. Για να αντιμετωπισθεί αυτό συντάχθηκε συνθήκη ώστε οι παίχτες του υπολογιστή να αγνοούν τα αντικείμενα της πίστας.

Τρίτο πρόβλημα που παρουσιάστηκε ήταν στην περίπτωση που όλα τα πιθανά τετράγωνα γύρω από τον παίχτη του χρήστη ήταν κατειλημμένα και ο

παίχτης του υπολογιστή δεν είχε άλλο στόχο οπότε δεν ήξερε τι να κάνει. Εφόσον δε μπορούσε να πλησιάσει για να επιτεθεί αλλά και ούτε είχε κάτι άλλο να κάνει, του δόθηκε η δυνατότητα να παραχωρήσει την σειρά του στον επόμενο παίχτη.

Από τις δοκιμές όμως έγιναν και βελτιώσεις. Χρήστες που δεν ήταν εξοικειωμένοι με το παιχνίδι δυσκολευόντουσαν στην επιλογή των τετραγώνων επειδή πολλές φορές μπερδευόντουσαν με τη γωνία της κάμερας. Οπότε έγινε η προσθήκη της φωτεινότητας του τετραγώνου από το οποίο περνάει ο κέρσορας. Έτσι η επιλογή έγινε ευκολότερη.

Για να μπορούν οι χρήστες να ξεχωρίζουν τους παίχτες που χειρίζονται προστέθηκε η φωτεινότητα του τετραγώνου που βρισκόταν ο παίχτης που έπαιζε σε εκείνο τον γύρω. Επίσης έγιναν αλλαγές στα μοντέλα των παιχτών του χρήστη ώστε να μπορεί να τα ξεχωρίζει ανάλογα με τα χαρακτηριστικά τους.

5.3 Άδειες και πνευματικά δικαιώματα

Το παιχνίδι είναι πλέον ολοκληρωμένο και δοκιμασμένο. Πριν όμως κυκλοφορήσει ο προγραμματιστής πρέπει να ελέγξει ότι είναι το παιχνίδι του δεν καταπατάει άδειες και πνευματικά δικαιώματα και δεν απειλείται από νομικές διαδικασίες.

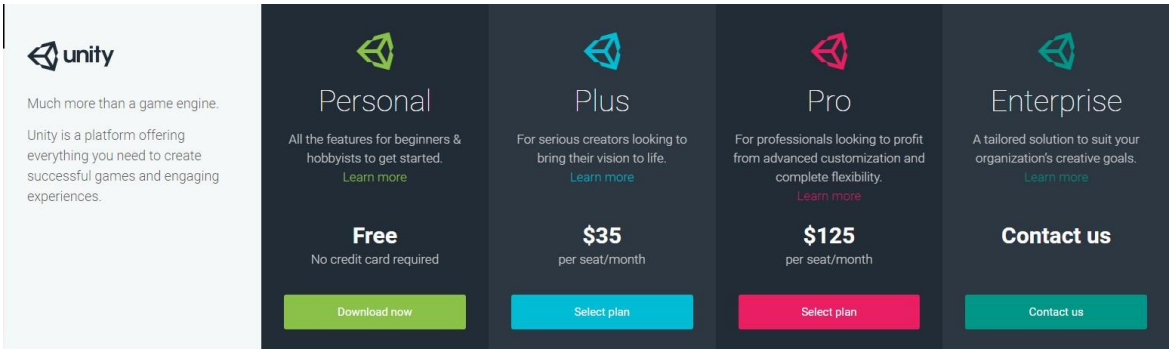
Όσων αφορά τα προτερήματα που παρέχονται από το Asset Store ισχύει ότι «ο χορηγών άδεια χορηγεί στον τελικό χρήστη μια μη αποκλειστική, παγκόσμια και διαρκή άδεια στο προτέρημα(Asset) για να ενσωματώσει τα προτερήματα μόνο όπως συνδυάζονται και ενσωματώνονται συστατικά των ηλεκτρονικών παιχνιδιών και των διαλογικών μέσων και να διανείμει τέτοιο ηλεκτρονικό παιχνίδι και διαλογικό μέσο. Εκτός από τα πακέτα ανάπτυξης λογισμικού υπηρεσιών παιχνιδιού («υπηρεσίες SDKs»), οι τελικοί χρήστες μπορούν να τροποποιήσουν τα προτερήματα. Ο τελικός χρήστης δεν μπορεί ειδάλλως να αναπαραγάγει, να διανείμει, να εκχωρήσει, να ενοικιάσει, να μισθώσει ή να δανείσει τα προτερήματα. Υπογραμμίζεται ότι οι τελικοί χρήστες δεν έχουν το δικαίωμα για να διανείμουν ή να μεταφέρουν με κάθε τρόπο (συμπεριλαμβανομένου, χωρίς, του περιορισμού μέσω της εκχώρησης) τα προτερήματα με οποιοδήποτε άλλο τρόπο απ’ ό, τι ως ενσωματωμένα συστατικά των ηλεκτρονικών παιχνιδιών και των διαλογικών μέσων. Χωρίς περιορισμό στα προεκτεθέντα υπογραμμίζεται ότι ο τελικός χρήστης δεν έχει το δικαίωμα για να μοιραστεί τις δαπάνες σχετικές με την αγορά

ενός προτερήματος και να αφήσει έπειτα οποιοδήποτε τρίτο που έχει συμβάλει σε τέτοια χρήση αγορών τέτοιο προτέρημα (φόρουμ που συγκεντρώνει).»**[1]**

Άρα όσον αφορά τη χρήση των αντικειμένων που βρίσκονται στο Asset store το παιχνίδι από μόνο του είναι νομικά καλυμμένο και κανείς δεν μπορεί να διεκδικήσει δικαιώματα σε αυτό.

Περί της έκδοσης του Unity στην οποία φτιάχτηκε το παιχνίδι οι περιορισμοί είναι «μια ετήσια εισοδηματική ικανότητα ή αντληθέντος ποσού ύψους $100k ανά οικονομικό έτος για τους πελάτες του Personal πακέτου. Μια ετήσια εισοδηματική ικανότητα ή αντληθέντος ποσού ύψους $200k ανά οικονομικό έτος για τους πελάτες του Plus πακέτου. Οι Pro και Enterprise πελάτες έχουν απεριόριστες εισοδηματικές ικανότητες.»**[2]**

Οπότε όσο ο δημιουργός του παιχνιδιού που χρησιμοποίησε την ελεύθερη Personal έκδοση του Unity και θέλει να πουλήσει το παιχνίδι δεν έχει ετήσιο εισόδημα άνω των 100.000$, μπορεί να το κάνει.



Εικόνα 5.3.1: Εκδόσεις του Unity

Η έκδοση που χρησιμοποιήθηκε για την πτυχιακή είναι η Personal και είναι ελεύθερη προς κατέβασμα και χρήση, όπως φαίνεται από την εικόνα 5.3.1.

5.4 Κυκλοφορία παιχνιδιού

Ο δημιουργός μπορεί να εκδώσει μόνος του το παιχνίδι σε φυσική ή ψηφιακή μορφή, αλλά εκτός και αν μπορεί να διαφημίσει το παιχνίδι, πιθανόν να δυσκολευτεί να βρει πολλούς αγοραστές. Πολλές εταιρείες όμως, όπως το Facebook και η Valve μέσω του Steam Store, είναι πρόθυμες να αναρτήσουν το παιχνίδι στις ιστοσελίδες τους, σε ελεύθερη ή επί πληρωμής έκδοση, και από τα όποια κέρδη από πωλήσεις ή διαφημίσεις να δίνεται ένα συμφωνημένο ποσοστό στον δημιουργό.

ΚΕΦΑΛΑΙΟ 6

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΟΠΤΙΚΕΣ

Η δημιουργία ενός παιχνιδιού είναι μια αρκετά περίπλοκη διαδικασία που απαιτεί γνώσεις σε πολλούς τομείς άμα πρόκειται να ολοκληρωθεί πλήρως από ένα άτομο. Υπάρχουν όμως πολλά εργαλεία που καθιστούν αυτό το εγχείρημα πραγματοποιήσιμο. Στην πτυχιακή έγινε εφαρμογή των απαραίτητων γνώσεων με τη χρήση των κατάλληλων εργαλείων και σε αυτό το κεφάλαιο θα αναλυθούν ποιες ήταν αυτές.

6.1 Συμπεράσματα για την ανάπτυξη βιντεοπαιχνιδιού

Από την αρχή της ανάπτυξης φάνηκε η ανάγκη για εξοικείωση με τις θεμελιώδεις αρχές του αντικειμενοστραφούς προγραμματισμού. Οι γλώσσες που υποστηρίζει και ο τρόπος με τον οποίο λειτουργεί το Unity το κατέστησαν αυτό σαφές. Σαφής επίσης γίνεται και η ανάγκη γνώσεων πάνω σε αλγόριθμους και πως μπορούν να εφαρμοστούν. Με αυτές τις γνώσεις κάποιος μπορεί να επιχειρήσει την δημιουργία ενός απλού παιχνιδιού, αλλά δεν είναι αρκετό για κάτι πιο περίπλοκο.

Γνώσεις πάνω στην τεχνητή νοημοσύνη είναι απαραίτητες εφόσον υπάρχει σκοπός προσθήκης εχθρών κινούμενων από τον υπολογιστή, με την δυσκολία να αυξάνεται ανάλογα με την πολυπλοκότητα των αποφάσεων που πρέπει να γίνουν. Επίσης λόγω του μεγάλου αριθμού των αρχείων που πρέπει να δημιουργηθούν, ανάλογα και με το μέγεθος της δουλειάς, ο χρήστης πρέπει να γνωρίζει πως να οργανώνει και κρατάει σε τάξη το παιχνίδι.

Ανάλογα με το πόσο επιθυμεί ο προγραμματιστής να κάνει χρήση του στοιχείου παιχνιδιού πολλαπλών παιχτών, θα πρέπει να έχει καλές γνώσεις πάνω στα δίκτυα και την ασφάλεια τους, κυρίως άμα έχει υψηλούς στόχους για το παιχνίδι του.

Το Unity είναι μια πολύ καλή μηχανή παιχνιδιού με πολλά εργαλεία και διευκολύνσεις για την ανάπτυξη παιχνιδιών, αλλά δεν προορίζεται για τον οποιοδήποτε και χρειάζεται πάρα πολύ δουλειά από τον προγραμματιστή.

6.2 Τρόποι αναβάθμισης της ανάπτυξης

Το Unity πλέον προσφέρει και το Unity Cloud. Είναι μια υπηρεσία που αυτοματοποιεί το χτίσιμο του παιχνιδιού και το φορτώνει σε έναν διαδικτυακό χώρο άμεσα επιτρέποντας την εύκολη και γρήγορη διανομή του παιχνιδιού για δοκιμή.

Για την δημιουργία μοντέλων και animation χρειάζονται εφαρμογές όπως τα Blender και Sketch up. Τα μοντέλα των περισσοτέρων προγραμμάτων είναι συμβατά με το Unity και αν ο δημιουργός του παιχνιδιού δεν επιθυμεί να χρησιμοποιεί μοντέλα άλλων από το Asset store ή τα πολύ απλά που προσφέρει το Unity, θα πρέπει να χρησιμοποιήσει ένα από αυτά.

Για την δημιουργία των εικόνων που περιβάλουν τα μοντέλα θα πρέπει να χρησιμοποιηθεί κάποιο πρόγραμμα επεξεργασίας εικόνας. Από το απλό Paint των Windows μέχρι το περίπλοκο αλλά πιο κατάλληλο για αυτή τη δουλειά Photoshop, ο δημιουργός θα χρειαστεί εικόνες που να ταιριάζουν στα μοντέλα του αλλιώς θα είναι μονόχρωμα.

Για ήχους αρχικά χρειάζεται η καταγραφή τους και η επεξεργασία και μετατροπή σε μορφή αρχείου που υποστηρίζει το Unity, όπως .mp3 και .wav.

Η ανάπτυξη λοιπόν ενός παιχνιδιού δε βασίζεται μόνο στο Unity. Η μηχανή αποτελεί το σημαντικότερο εργαλείο εφόσον διαχειρίζεται τα μοντέλα, προσφέρει πληθώρα εργαλείων για τον έλεγχο όλων των πόρων, διαθέτει υπηρεσίες για την διευκόλυνση του χρήστη, αλλά για να φτιαχτεί ένα πλήρες παιχνίδι χρειάζονται προσθήκες που μόνο από άλλα προγράμματα μπορούν να παραχθούν.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] ‘Asset store terms of service and EULA’ Προσβάσιμο: https://unity3d.com/legal/as_terms (Ημερομηνία επίσκεψης: 2016, Οκτώβριος)

[2] ‘Revenue Capacity’ Προσβάσιμο: https://store.unity.com/ (Ημερομηνία επίσκεψης: 2016, Οκτώβριος)