ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΑΛΙΝΔΡΕΤΟΣ ΕΛΕΥΘΕΡΙΟΣ

Ανάπτυξη Τρισδιάστατου Παιχνιδιού,
ενός και Πολλαπλών Παιχτών

Εξεταστική επιτροπή
Αν. Καθηγήτρια Αικατερίνη Μανιά (επιβλέπουσα)
Αν. Καθηγητής Μιχαήλ Λαγουδάκης
Αν. Καθηγητής Αντώνιος Δεληγιαννάκης

Χανιά 2015

Περίληψη

Η εργασία παρουσιάζει την υλοποίηση ενός τρισδιάστατου παιχνιδιού με λειτουργία ενός παίχτη και με δυνατότητα συμμετοχής πάνω από έναν παίχτες. Το παιχνίδι δημιουργήθηκε με τη χρήση της μηχανής παιχνιδιών Unity και υλοποιεί διαφορετικές λειτουργίες-τύπους παιχνιδιών. Στο πρώτο σκέλος έχει λειτουργικότητα παιχνιδιού πλατφόρμας. Στο δεύτερο σκέλος έχει στοιχεία δράσης – turn-base παιχνιδιού ρόλων και στο τρίτο υπάρχει η δυνατότητα ενός παιχνιδιού πολλών παιχτών με στοιχεία πρώτου προσώπου shooter.

Η υλοποίηση ενός σύγχρονου παιχνιδιού είναι μια ιδιαίτερα απαιτητική εργασία που απαιτεί ανθρώπους με διαφορετικές ειδικότητες (προγραμματιστές, γραφίστες κλπ). Έτσι για την εκπόνηση του παιχνιδιού πήραμε αρκετά δωρεάν κομμάτια από το Asset Store που μας παρέχει η Unity (μοντέλα, ήχους και κινήσεις μοντέλων) που βοήθησαν στην υλοποίηση και στη διαμόρφωση του παιχνιδιού.

Η υλοποίηση έγινε σε C# (μέσα από τη λειτουργικότητα που προσφέρει η βασική κλάση Monobehaviour), μέσω της οποίας προγραμματίζονται οι αλληλεπιδράσεις των τρισδιάστατων αντικειμένων. Μέσω του manual της Unity καθώς και την υποστήριξη της κοινότητας (forum, tutorial κλπ) προγραμματίσαμε ένα παιχνίδι που έχει την δυνατότητα να συμμετέχουν παραπάνω από ένας παίχτες. Αφού ολοκληρώθηκε η υλοποίηση του παιχνιδιού το παιχνίδι δοκιμάστηκε από χρήστες διαφόρων ηλικιών και φύλων και αξιολογήθηκε με την χρήση του ερωτηματολογίου αξιολόγησης διαλογικής χρήσης Συστήματος (QUIS - Πανεπιστήμιο του Maryland, 1986-1998).

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτρια κ. Αικατερίνη Μανιά και τον διδακτορικό φοιτητή της Κουλιέρη Γεώργιο Αλέξανδρο και τον μεταπτυχιακό φοιτητή της Κονταδάκη Γρηγόρη για την πολύτιμη βοήθεια και καθοδήγηση που μου παρείχαν. Επίσης, θα ήθελα να ευχαριστήσω τους καθηγητές κ. Μιχαήλ Λαγουδάκη και κ. Αντώνη Δεληγιαννάκη για τον χρόνο που θα αφιερώσουν στην ανάγνωση της διπλωματικής μου εργασίας.

Περιεχόμενα

Λίστα εικόνων

Λίστα πινάκων

Κεφάλαιο 1
Εισαγωγή

1.1 Εισαγωγή

Στόχος της διπλωματικής εργασίας είναι η υλοποίηση ενός τρισδιάστατου παιχνιδιού με λειτουργία ενός παίχτη και με δυνατότητα συμμετοχής περισσοτέρων παιχτών με την χρήση της μηχανής παιχνιδιών Unity. Το παιχνίδι έχει στοιχεία από διάφορες κατηγορίες παιχνιδιών. Στην αρχή υλοποιείται σαν ένα παιχνίδι πλατφόρμας. Ο χρήστης χειρίζεται μια σφαίρα και προσπαθεί να αποφύγει ή να σκοτώσει τους εχθρούς που συναντάει μέχρι να φτάσει στο τέλος της πίστας. Στο επόμενο σκέλος το παιχνίδι μετατρέπεται σε ένα δράσης-τρίτου προσώπου shooter ή ένα turn-base παιχνίδι ρόλων. Ο χρήστης επιλέγει τι λειτουργία του αρέσει να παίξει και χειρίζεται έναν μάγο που αντιμετωπίζει εχθρούς προσπαθώντας να φτάσει στο τέλος της πίστας. Στο τελευταίο σκέλος, το παιχνίδι με δυνατότητα συμμετοχής πολλών παιχτών, έχει την λειτουργία πρώτου προσώπου Shooter, όπου ο χρήστης ή επιτίθεται σε άλλους χρήστες ή συνεργάζεται με αυτούς ώστε να αντιμετωπίσει δεινοσαύρους και να μαζέψει χρυσό.

1.2 Περιγραφή παιχνιδιού

Η διπλωματική μας εργασία αποτελείτε από τρία σκέλη. Το πρώτο σκέλος αποτελείτε από μια υλοποίηση τύπου παιχνιδιού πλατφόρμας το δεύτερο σκέλος από μια υλοποίηση τύπου παιχνιδιού δράσης ή τύπου παιχνιδιού ρόλων και το τρίτο σκέλος που αποτελεί και το κομμάτι συμμετοχής πολλών παιχτών τύπου παιχνιδιού shooter. Στα πλαίσια της υλοποίησης του παραπάνω παιχνιδιού δημιουργήσαμε το παρακάτω σενάριο για το παιχνίδι μας. Σε έναν μακρινό κόσμο ένα ζευγάρι μάγων έκανε κάποια πειράματα. Ένα αποτυχημένο πείραμα είχε ως αποτέλεσμα να δημιουργηθούν δύο νέοι κόσμοι. Ένας κόσμος με δεινοσαύρους και ένας με γεωμετρικά στερεά. Κάποια στιγμή ο κόσμος τον δεινοσαύρων εισβάλει στο κόσμο των στερεών και αιχμαλωτίζει όλα τα στερεά εκτός από τη σφαίρα (που θα είναι ο βασικός χαρακτήρας μας στο πρώτο κομμάτι του παιχνιδιού). Η σφαίρα προσπαθεί να μαζέψει αρκετή ποσότητα ενός φίλτρου, που δίνει τρομακτική δύναμη σε όποιον το πιει, και με την βοήθεια αυτού να καταφέρει να ελευθερώσει

τα άλλα στερεά. Όταν η σφαίρα καταφέρνει να ελευθερώσει τα φυλακισμένα στερεά βρίσκει ένα αιχμάλωτο μάγο μαζί τους (που θα είναι ο βασικός χαρακτήρας μας στο δεύτερο σκέλος της εργασίας μας). Ο μάγος είχε βρεθεί σε αυτόν τον κόσμο τυχαία και τον είχαν αιχμαλωτίσει οι δεινόσαυροι. Ο σκοπός του μάγου τώρα πηγαίνοντας μέσα από διαφορές πύλες (οι πύλες σε μεταφέρουν σε τυχαίους κόσμους) να προσπαθήσει να επιστρέψει στο κόσμο από όπου ήρθε. Η δράση του παιχνιδιού εξελίσσεται σε ένα τρισδιάστατο χώρο (πίστα). Στο πρώτο σκέλος του παιχνιδιού ο χρήστης χειρίζεται μια σφαίρα και ο σκοπός του είναι να μαζέψει ένα μαγικό φίλτρο. Για να το καταφέρει αυτό πρέπει ο χρήστης να χρησιμοποιήσει τα αντανακλαστικά του ώστε να πατήσει την σωστή στιγμή τα κουμπιά για να περάσει τα εμπόδια που έχει ώστε να μαζέψει το μαγικό φίλτρο. Στο δεύτερο σκέλος ο χρήστης χειρίζεται ένα μάγο και έχει σκοπό να εξολοθρεύσει πέντε εχθρούς σε κάθε πίστα ώστε να εμφανιστεί η πύλη για την επόμενη πίστα. Στο τρίτο σκέλος ο χρήστης χειρίζεται έναν μάγο με σκοπό είτε να εξολοθρεύσει άλλους μάγους που χειρίζονται άλλοι χρήστες, ή με την βοήθεια τους να εξολοθρεύσει δεινοσαύρους ώστε να μαζέψει χρυσό.

1.3 Δομή και περιεχόμενα εργασίας

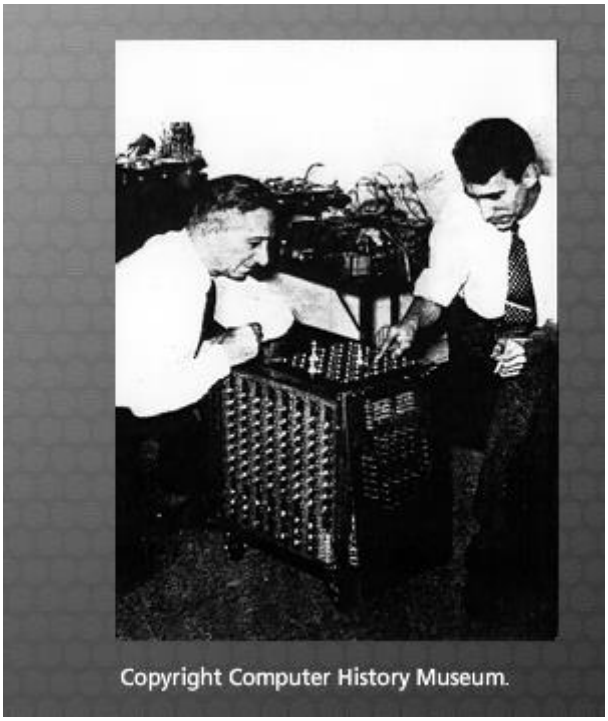
Στο δεύτερο κεφάλαιο παρουσιάζουμε την ιστορία των βιντεοπαιχνιδιών. Παρουσιάζουμε ψυχολογικά οφέλη που μπορούν να αποφέρουν τα βιντεοπαιχνίδια, αναλύουμε τα είδη τα ων βιντεοπαιχνιδιών και αναλύουμε την τεχνολογική βάση της εργασίας. Πιο συγκεκριμένα αναλύουμε τα χαρακτηριστικά της μηχανής παιχνιδιών και τα βασικά χαρακτηριστικά που χρησιμοποιήσαμε. Στο τρίτο κεφάλαιο αναλύουμε την υλοποίηση και το user interface που δημιουργήσαμε. Στο τέταρτο κεφάλαιο αναλύουμε πώς υλοποιήσαμε διάφορα μέρη του παιχνιδιού και παρατίθενται ενδεικτικά κομμάτια κώδικα αυτής της υλοποίησης. Στο πέμπτο κεφάλαιο γράφουμε τα αποτελέσματα από μερικές κριτικές παιχτών του παιχνιδιού και βελτιώσεις που κάναμε με αυτές τις κριτικές πάνω στην εφαρμογή μας.

Κεφάλαιο 2
Σχετική Έρευνα και Τεχνολογική Βάση

2.1 Ιστορία βιντεοπαιχνιδιού

Παρακάτω παρουσιάζεται το χρονολόγιο με τις πιο σημαντικές εξελίξεις στον τομέα τον βιντεοπαιχνιδιών [5], [10],[11],[18]:

1940: Ο Edward U. Condon δημιουργεί ένα υπολογιστή που παίζει το παιχνίδι Nim (ο παίχτης προσπαθεί να μην σηκώσει το τελευταίο σπίρτο).
1947: Οι Thomas T. Goldsmith Jr. και Estle Ray Mann δημιουργούν το πρώτο arcade παιχνίδι όπου ο χρήστης πυροβολεί σε μια οθόνη και ανάβουν-σβήνουν κάποια φωτάκια.
1950: Ο Claude Shannon δίνει την βασική λογική για την υλοποίηση ενός προγράμματος που οι παίχτες παίζουν σκάκι στο άρθο του "Programming a Computer for Playing Chess". Λίγα χρόνια ο Alan Turing δημιούργησε αρκετά προγράμματα παιχνιδιού με σκάκι.



Εικόνα 2.1 Η μηχανή που δημιουργήθηκε για να παίζει σκάκι

1952: Ο A. S. Douglass δημιουργεί το OXO για να μελετήσει τις αντιδράσεις μεταξύ ανθρώπων και υπολογιστών.

1954: Δημιουργείται το πρώτο πρόγραμμα για blackjack.

1955: Οι Αμερικάνοι δημιουργούν το Hutspiel. Το πρώτο πολεμικό παιχνίδι.

1956: Ο Arthur Samuel παρουσιάζει στην τηλεόραση ένα πρόγραμμα σκακιού σε υπολογιστή.

1957: Ο Alex Bernstein φτιάχνει το πρώτο ολοκληρωμένο πρόγραμμα σκακιού που μπορεί και υπολογίζει τεσσερισήμισι κινήσεις μπροστά.

1958: Ο Willy Higinbotham δημιουργεί το πρώτο παιχνίδι τέννις που αργότερα έδωσε ιδέες για άλλα παιχνίδια όπως το pong.

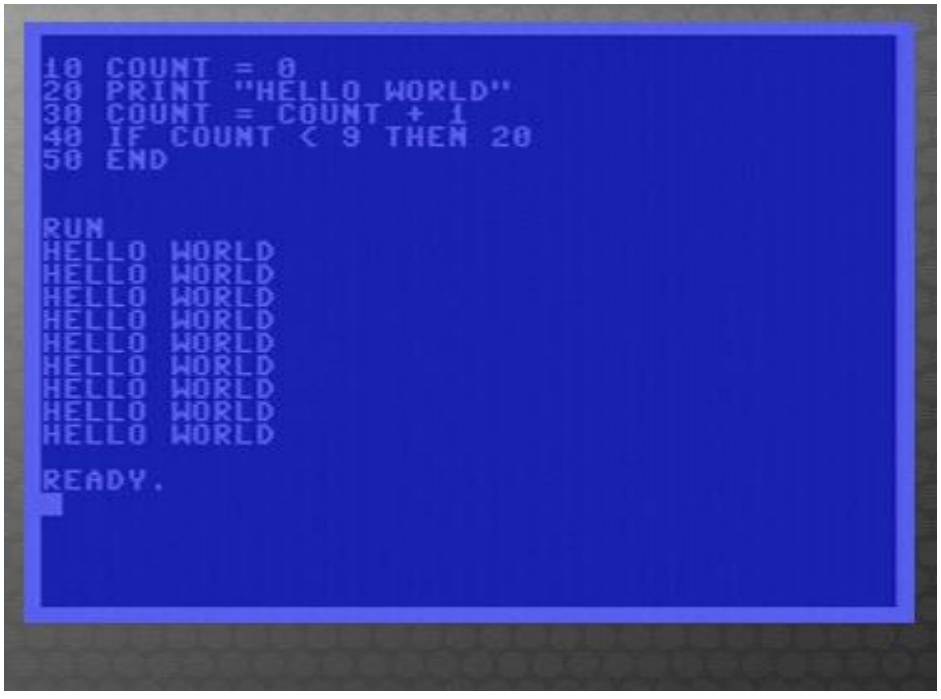
1959: Φοιτητές στο MIT δημιουργούν το παιχνίδι με το ποντίκι στον λαβύρινθο.

1960: Ο προγραμματιστής John Burgeson δημιουργεί τον πρώτο προσομοιωτή baseball.

1961: Η εταιρία Raytheon Company δημιουργεί έναν εξομοιωτή του ψυχρού πολέμου.

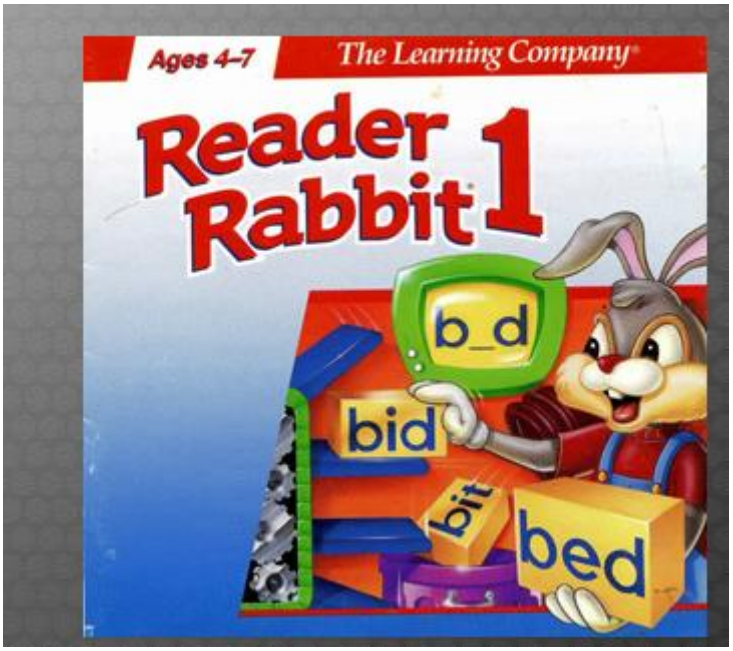
1962: Ο Steve Russell δημιουργεί το πρώτο βιντεοπαιχνίδι με βάση τον υπολογιστή. Το παιχνίδι το ονόμασε Spacewar.

1964: Ο John Kemeny δημιουργεί την πρώτη γλώσσα προγραμματισμού και δίνει την δυνατότητα σε χιλιάδες ανθρώπους να δημιουργήσουν τα δικά τους παιχνίδια. Μέχρι εκείνη την εποχή το προγραμματιστικό σκέλος γινόταν μόνο από μαθηματικούς ή επιστήμονες που χρησιμοποιούσαν τον προγραμματισμό για να υπολογίσουν δεδομένα. Ο Kemeny ήθελε να ανοίξει το προγραμματιστικό κομμάτι και σε άλλα είδη σπουδαστών. Γι αυτό το λόγο υλοποίησε την Basic language.



Εικόνα 2.2 Παράδειγμα του προγράμματος που δημιούργησε ο kemeny

1966: Ο Ralph Baer εμπνέεται για την δημιουργία ενός παιχνιδιού στην τηλεόραση. Αυτή η ιδέα είναι προπομπός των κονσόλων παιχνιδιών που χρησιμοποιούμε σήμερα.
1971: Οι Don Rawitsch, Bill Heinemann και Paul Dillenberger δημιουργούν το Oregon Trail.
1972: Οι Nolan Bushnell και Al Alcorn δημιουργούν το παιχνίδι pong, ένα ηλεκτρονικό παιχνίδι πινκ πονκ.
1974: Δημιουργείται το Maze Wars το πρώτο first person shooter.
1977: Δημιουργείται το Atari 2600.
1978: Δημιουργείται το arcade παιχνίδι Space Invaders.
1980: Δημιουργείται το Pac-Man.
1984: Δημιουργείται το Tetris από τον μαθηματικό Alexey Pajitnov.
1985: Δημιουργείται η κονσόλα Nes από την Nintedo.
1986: Δημιουργείται το εκπαιδευτικό πρόγραμμα Reader Rabbit.



Εικόνα 2.3 Το εκπαιδευτικό παιχνίδι Reader Rabbit

1987: Δημιουργείται το legend of Zelda του Shigeru Miyamoto.
1989: Δημιουργείται το Nintendo Game Boy.
1990: Η Microsoft εισάγει την πασιέντσα στο λειτουργικό της σύστημα των Windows 3.0.
1995: Δημιουργείται το PlayStation.
1999: Η Sony δημιουργεί το online παιχνίδι Everquest.
2003: Δημιουργείται το Steam από την Valve.
2004: Δημιουργείται το Nintendo ds.

2005: Δημιουργείται το Xbox360.
2006: Δημιουργείται το Nintendo Wii.
2010: Δημιουργείται το Minecraft.
2014: Το είδος παιχνιδιού free to play γίνεται ευρέως διαδεδομένο με διάφορα παραδείγματα τέτοιου τύπου το league of legends, crossfire κλπ.

2.2 Ψυχολογικά οφέλη βιντεοπαιχνιδιού

Αρκετές έρευνες απέδειξαν ότι τα βιντεοπαιχνίδια αυξάνουν την αίσθηση του γύρω χώρου, μνήμη, λογική και αντίληψη. Πιο συγκεκριμένα παιδιά που έπαιζαν παιχνίδια shooter αύξαναν την αίσθηση τους για τον χώρο των τριών διαστάσεων. Παιδιά που έπαιζαν παιχνίδια Puzzle ή RPG είχαν πιο πολύπλοκο τρόπο σκέψης. Εύκολα παιχνίδια δίνουν στον χρήστη μια αίσθηση ικανοποίησης και μπορεί να του αυξήσουν την αυτοπεποίθηση. Διαδικτυακά παιχνίδια πολλών παιχτών παρατηρήθηκε να αυξάνουν την κοινωνικότητα των παιδιών που τα παίζουν. Σε σχετικό πείραμα που έγινε σε διάφορα νοσοκομεία σε ασθενείς με καρκίνο τα παιδιά που έπαιζαν το παιχνίδι “Re-Mission” είχαν αξιοσημείωτη προσήλωση στην θεραπεία της ασθένειας. Το“Re-Mission” είναι ειδικό παιχνίδι που σχεδιάστηκε για να παρέχει πληροφορίες στους ασθενείς της εν λόγω ασθένειας [6].

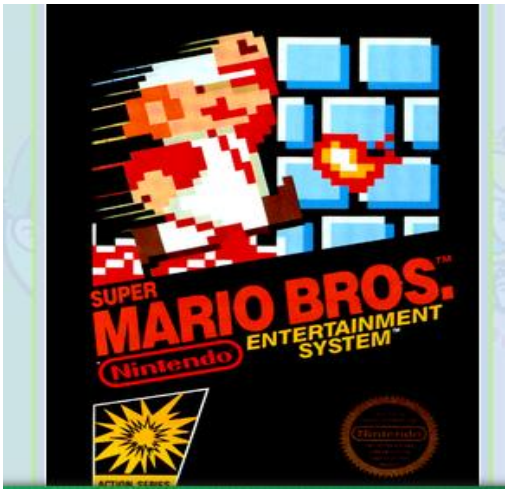
2.3 Είδη βιντεοπαιχνιδιού

Τα είδη των βιντεοπαιχνιδιών και τα χαρακτηριστικά τους παρατίθενται στον πίνακα παρακάτω [7],[19]:

Είδη βιντεοπαιχνιδιών	**Βασικά χαρακτηριστικά**
Shooter	Ο χρήστης πυροβολεί σε διάφορες κατευθύνσεις εχθρούς ώστε να περάσει στο επόμενο επίπεδο.
Περιπέτειας (Adventure)	Ο χρήστης συνήθως κινείται σε έναν ελεύθερο χάρτη εξερευνώντας και λύνοντας κάποιους γρίφους.
Πλατφόρμας (Platform)	Ο χρήστης συνήθως χρειάζεται να πατάει τα κουμπιά σε σωστό χρόνο προκειμένου να αποφεύγει παγίδες ή εχθρούς.
Ρόλων (Role playing games / RPG)	Χαρακτηρίζεται από δυο βασικά στοιχεία: Συγκεκριμένες αποστολές και εξέλιξη του χαρακτήρα.
Γρίφων (Puzzle)	Ο χρήστης έχει να λύσει διάφορους γρίφους.
Εξομοίωσης (Simulation)	Συνήθως γίνεται εξομοίωση τη πραγματικής ζωής ή μιας δραστηριότητας.
Στρατηγικής (Strategy)	Ο χρήστης βρίσκει τρόπους επίτευξης των σκοπών του παιχνιδιού χρησιμοποιώντας τα μέσα που του

	παρέχονται.
Αθλημάτων (Sports)	Αποτελούν εξομοίωση αθλημάτων (ποδόσφαιρο, μπάσκετ κλπ).
Μάχης (Fighting)	Απαιτούν συντονισμό κινήσεων και γρήγορα αντανακλαστικά προκειμένου να εξουδετερωθεί ένας αντίπαλος χρήστης.
Πολλών παιχτών μέσω διαδικτύου (Massive Multiplayer Online / MMO)	Υπάρχουν πολλοί άνθρωποι χρήστες που αλληλεπιδρούν μεταξύ τους και συνδέονται στο παιχνίδι μέσω διαδικτύου.
Επιβίωσης τρόμου (Survival Horror)	Ο χρήστης δίνει έμφαση στο να μείνει ζωντανός.
Οδήγησης αγώνων (Driving / racing)	Ο χρήστης οδηγεί ένα όχημα με σκοπό συνήθως να τερματίσει πριν από τους άλλους χρήστες σε μια πίστα.
Arcade games	Συνήθως υπάρχουν μικρά επίπεδα με απλό χειρισμό και εκθετική δυσκολία παιχνιδιού καθώς ανεβαίνεις επίπεδο.
Υβριδικά (Hybrids)	Εδώ μπορεί να υπάρξει δημιουργία ενός παιχνιδιού με χαρακτηριστικά περισσότερα του ενός γένους.

Πίνακας 2.1 Είδη βιντεοπαιχνιδιού



Εικόνα 2.4 Το παιχνίδι πλατφόρμας super Mario bros

2.4 Μηχανή παιχνιδιού Unity

Η Unity είναι μια από τις πιο διαδεδομένες μηχανές προγραμματισμού. Παρέχει την δυνατότητα δημιουργίας εκτελέσιμου σε διάφορες πλατφόρμες (iOS, Android, Mac, Windows, Web Browsers). Παρέχει δωρεάν την personal έκδοση της γλώσσας που δίνει την δυνατότητα δημιουργίας παιχνιδιού ακόμα και για εμπορική χρήση μέχρι το όριο κερδών των εκατό χιλιάδων δολαρίων. Εάν τα κέρδη είναι περισσότερα τότε απαιτείται η χρήση της επαγγελματικής έκδοσης [1],[3].

2.4.1 Μηχανή γραφικών

Η Unity χρησιμοποιεί βιβλιοθήκες γραφικών ανάλογα με το περιβάλλον εκτέλεσης του παιχνιδιού:

- Direct 3D σε περιβάλλον Windows (εάν έχει υποστήριξη κάτω από dx 9.0) και σε Xbox360.
- OpenGL σε περιβάλλον Windows, Mac, Linux.
- OpenGL ES σε περιβάλλον Andoid, IOS.
- Ιδιόκτητες API σε συγκεκριμένες κονσόλες.

2.4.2 Κώδικας

Ο κώδικας πάνω στον οποίο είναι χτισμένη η μηχανή Unity είναι βασισμένος στον Mono. Ο Mono είναι μια ανοιχτή πλατφόρμα βασισμένο στην πλατφόρμα .NET framework που παρέχει την δυνατότητα σε χρήστες του να υλοποιήσουν προγράμματα σε πολλές πλατφόρμες χρησιμοποιώντας πολλές γλώσσες προγραμματισμού. Συγκεκριμένα στην Unity χρησιμοποιείται είτε C# ή Javascript [8],[9].

2.4.3 Μηχανή Φυσικής

Η Unity χρησιμοποιεί μια ενσωματωμένη μηχανή Φυσικής: την PshysX3 της Nvidia. Με τη μηχανή φυσικής παρέχεται η δυνατότητα ανίχνευσης συγκρούσεων, προσθήκη βαρύτητας και άλλων δυνάμεων μέσα σε μια τρισδιάστατη σκηνή.

2.5 Asset store

Το Asset store που έχει υλοποιήσει η Unity είναι ένα μέρος στο οποίο μπορεί ο σχεδιαστής να προμηθευτεί περιεχόμενο για τα παιχνίδια του, καθώς και να ανεβάσει περιεχόμενο. Αναλυτικότερα δίνεται η δυνατότητα για αγορά έτοιμων τρισδιάστατων μοντέλων (χαρακτήρες ή ακόμα και τρισδιάστατα αντικείμενα), animation δηλαδή κινήσεις μοντέλων, έτοιμων project, έτοιμου κώδικα ή έτοιμων εφαρμογών. Μέσω του asset store κατεβάσαμε το μοντέλο του χαρακτήρα που χρησιμοποιούμε, τα μοντέλα από τους εχθρούς (σκελετούς και δεινοσαύρους), αρκετά τρισδιάστατα μοντέλα του περιβάλλοντος χώρου που χρησιμοποιήσαμε (τραπέζια, βιβλία κλπ), ήχους που βάλαμε στο παιχνίδι μας καθώς και αρκετά textures που χρησιμοποιήθηκαν για τα μοντέλα μας [13].

2.6 Δομή της Unity

Τα προγράμματα τα οποία υλοποιούνται στην Unity αποτελούνται πρώτα από τρισδιάστατες σκηνές. Η κάθε σκηνή αποτελείται από ένα σύνολο αντικειμένων. Τα αντικείμενα που υπάρχουν μέσα σε μια σκηνή μπορεί να είναι η πίστα του παιχνιδιού (διάφορα μοντέλα, ήχοι μέσα στην σκηνή κλπ) και οι γραφικές διεπαφές του χρήστη. Αναλυτικότερα τα κομμάτια που αποτελούν τη Unity περιγράφονται στις παρακάτω ενότητες.

2.6.1 Gameobject

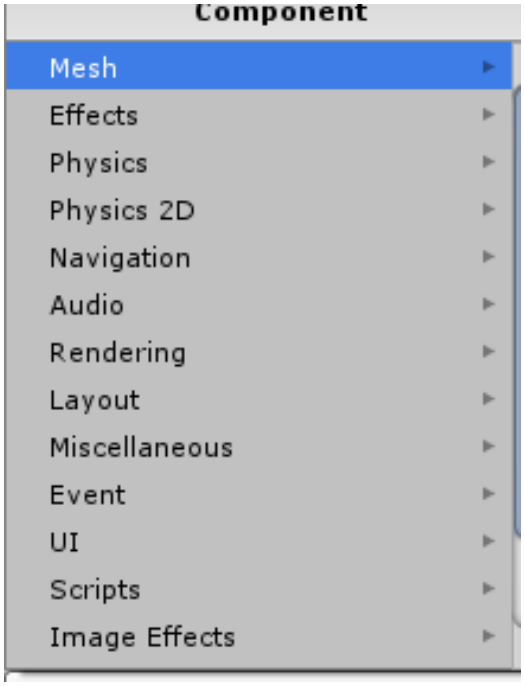
Το κάθε αντικείμενο που υπάρχει μέσα στην σκηνή ονομάζεται gameobject. Στο κάθε gameobject για να μπορέσει να είναι λειτουργικό μέσα στην πίστα μας, του εισάγουμε components. Σε κάθε gameobject μπορούμε να εισάγουμε όσα components επιθυμούμε. Ουσιαστικά τα gameobjects μπορούν να παρομοιαστούν με άδεια κουτιά που δεν είναι χρήσιμα μέχρι να βάλεις κάτι μέσα τους.

2.6.2 Prefab

Prefab είναι ένα έτοιμο υλοποιημένο gameobject μαζί με τα component του, που είναι αποθηκευμένο μέσα στον τρέχοντα φάκελο του project. Δημιουργώντας το prefab δίνεται η δυνατότητα να εισαχθεί ένα αντικείμενο με συγκεκριμένα χαρακτηριστικά πολλαπλές φορές στη σκηνή εξοικονομώντας πόρους του συστήματος.

2.6.3 Components

Τα Components είναι ουσιαστικά τα κομμάτια που εισάγουμε στα gameobjects για να μπορέσουν να είναι λειτουργικά μέσα στο πρόγραμμα μας. Οι τύποι component είναι οι παρακάτω.



Εικόνα 2.5 Τα είδη των component

Mesh: Δίνει την δυνατότητα δημιουργίας σχημάτων.
Effects: Εφέ όπως το particle system που δίνουν την δυνατότητα δημιουργίας βροχής κλπ.
Physics: Μας δίνει την δυνατότητα εισάγοντας Rigidbody να μπορούμε να κινήσουμε το αντικείμενο μας και εισάγοντας κάποιον collider να μπορούμε να ανιχνεύσουμε οποιαδήποτε σύγκρουση έχει με άλλα αντικείμενα.
Physics 2D: Ότι και το physics αλλά για το δισδιάστατο επίπεδο.
Navigation: Δίνει την δυνατότητα εισάγοντας το σε έναν χαρακτήρα, να αποκτήσει μια ευρεία αίσθηση του χώρου π.χ. να καταλαβαίνει πότε θα πρέπει να ανέβει τις σκάλες για να πάει στον δεύτερο όροφο.
Audio: Δίνει την δυνατότητα εισαγωγής είτε μιας Audiosource όπου θα μπορεί το gameobject να παίζει κάποιον ήχο, ή Audiolistener όπου θα μπορεί ο χρήστης να ακούει τους ήχους που υπάρχουν στην σκηνή (κάθε σκηνή μπορεί να έχει μόνο έναν Audiolistener).
Rendering: Τα components που αποτελούν το rendering κομμάτι είναι components τα οποία χρησιμοποιούνται για την εμφάνιση των μοντέλων στην σκηνή και για την εισαγωγή διαφόρων ειδικών εφέ. Μερικά παραδείγματα rendering components

είναι η κάμερα που δίνει τη δυνατότητα προβολής των gameobject σε μια σκηνή, το Skybox που δίνει την δυνατότητα εισαγωγής μιας εικόνας στο background μιας σκηνής κλπ. [20]
Layout: Χρησιμοποιείται για την εισαγωγή γραφικών διεπαφών.
Miscallaneous: Εδώ υπάρχουν διάφορα χρήσιμα components όπως το networkview που θα αναλυθεί εκτεταμένα πιο κάτω.
Event: Εδω εισάγουμε κάποια events στο gameobject, το gameobject ενημερώνεται όταν υπάρχουν αντικείμενα που αλληλεπιδράνε με αυτό.
UI: Εδώ δίνεται η δυνατότητα εισαγωγής γραφικών κομματιών στη σκηνή (όπως εικόνες, κείμενα κλπ).
Scripts: Με την εισαγωγή των scripts δίνεται η δυνατότητα επεξεργασίας των αντικειμένων. Θα αναλυθεί εκτεταμένα πιο κάτω.
Image Effects: Εδώ εισάγονται εφέ όπως θόλωση εικόνας κλπ.

2.7 Υλοποίηση

Για να υλοποιηθούν οι αλληλεπιδράσεις και η λειτουργικότητα των τρισδιάστατων αντικειμένων σε μια σκηνή είναι απαραίτητη η χρήση μιας scripting γλώσσας. Η Unity μας δίνει την δυνατότητα να προγραμματίσουμε σε δυο γλώσσες προγραμματισμού C# και javascript. Εμείς υλοποιήσαμε το πρόγραμμα μας σε C#.

2.7.1 Βασικές συναρτήσεις της Mono behavior

```
using UnityEngine
using System.Collections
```

Με τον παραπάνω τρόπο εισάγουμε τις βιβλιοθήκες από τις οποίες κληρονομούμε τη βασική λειτουργικότητα που χρειαζόμαστε. Οι βασικές συναρτήσεις που χρησιμοποιούμε από την Monobehavior είναι οι παρακάτω:

```
void Awake() {}
```

Καλείται με το που δημιουργείται το αντικείμενο ακόμα και αν δεν είναι ενεργοποιημένο.

```
void Start() {}
```

Καλείται μετά την `Awake`() εάν το αντικείμενο είναι ενεργοποιημένο.

```
void FixedUpdate() {}
void Update() {}
```

Το `Update()` καλείτε σε κάθε frame ενώ η `FixedUpdate()` καλείται μετά από συγκεκριμένα frames.

```
void OnGUI() {}
```

Η `OnGUI()` καλείται για να υλοποιηθούν τα γεγονότα που έχουν σχέση με το γραφικό κομμάτι της εφαρμογής.

2.7.2 Συναρτήσεις αλληλεπιδράσεων αντικειμένων

```
gameObject.tag = "tag"
```

Καλώντας την μαρκάρουμε το αντικείμενο για να μπορέσουμε να το βρούμε εύκολα αργότερα.

```
GameObject.FindGameObjectWithTag("tag")
GameObject.Find("name")
```

Οι παραπάνω συναρτήσεις επιστρέφουν ένα αντικείμενο. Η πρώτη συνάρτηση επιστρέφει αντικείμενο που θα κάνει αναζήτηση με βάση το tag που επιθυμούμε και η δεύτερη με βάση το όνομα του αντικειμένου.

```
void OnTriggerEnter(Collider other) {}

void OnTriggerExit(Collider other) {}
```

```
void OnTriggerStay(Collider other) {}
```

Οι παραπάνω συναρτήσεις καλούνται όταν το αντικείμενο μας συγκρουστεί με ένα άλλο αντικείμενο και το άλλο αντικείμενο το διαπεράσει. Η πρώτη συνάρτηση καλείται την στιγμή που αρχίζει η σύγκρουση. Η δεύτερη συνάρτηση καλείται τελειώνει η σύγκρουση και η τρίτη συνάρτηση καλείται επανειλημμένα κατά τη διάρκεια της σύγκρουσης.

```
void OnCollisionEnter(Collision collision) {}
```

Η παραπάνω συνάρτηση καλείται όταν ξεκινάει κάποια σύγκρουση αλλά το ένα αντικείμενο δεν μπορεί να διαπεράσει το άλλο.

```
gameObject.GetComponent<>()
```

Με την παραπάνω συνάρτηση έχουμε πρόσβαση σε ένα συγκεκριμένο component ενός αντικειμένου.

```
Instantiate(prefab, position,rotation)
```

Με την παραπάνω συνάρτηση αφαιρούμε ένα αντικείμενο από την σκηνή μας. Με την παραπάνω συνάρτηση δημιουργούμε ένα prefab σε μια συγκεκριμένη θέση και με συγκεκριμένη ροπή.

```
Destroy(gameObject)
```

2.8 Masterserver

Ο Masterserver είναι ένας σέρβερ που παρέχει η Unity. Μέσω αυτού οι χρήστες της μπορούν να τρέχουν και να δοκιμάζουν τα διαδικτυακά προγράμματα τους.

2.8.1 Βασικές συναρτήσεις Masterserver

Στην αρχή πρέπει να δηλώσουμε το παιχνίδι μας στο Masterserver της Unity. Αυτό το καταφέρνουμε με τις τρείς παρακάτω γραμμές κώδικα.

```
Network.InitializeSecurity()
Network.InitializeServer(MaxPlayers, 25565, true)
MasterServer.RegisterHost("gameserverName",ServerName,"")
```

Την πρώτη συνάρτηση πρέπει να την καλέσουμε πριν δημιουργήσουμε τον σέρβερ. Την δεύτερη συνάρτηση την καλούμε για να δημιουργήσουμε τον σέρβερ δηλώνοντας παράλληλα τον μέγιστο αριθμό παιχτών και εάν θα χρησιμοποιήσουμε το NAT punchthrough (ώστε να μπορεί να περνάει το firewall μας). Με την τρίτη συνάρτηση δηλώνουμε το δικό μας παιχνίδι πάνω στον Masterserver της Unity.

```
MasterServer.RequestHostList("servername")
```

Με την παραπάνω εντολή ζητάμε από τον Masterserver να μας ενημερώσει για τα παιχνίδια που υπάρχουν δηλωμένα στον σέρβερ που βάζουμε ως όρισμα.

```
Network.Connect("ip","port")
```

Με την παραπάνω εντολή συνδεόμαστε στον εκάστοτε σερβερ παιχνιδιού.

2.8.2 Διαδικτυακές συναρτήσεις

Οι βασικές συναρτήσεις που χρησιμοποιούμε και κληρονομούνται από τον Mono είναι οι παρακάτω:

```
void OnConnectedToServer() {}
```

Καλείται όταν ο χρήστης έχει καταφέρει να συνδεθεί στον σέρβερ.

```
void OnPlayerConnected(NetworkPlayer id) {}
```

Καλείται στον σέρβερ όταν συνδεθεί νέος χρήστης.

```
void OnPlayerDisconnected(NetworkPlayer id) {}
```

Καλείται όταν αποσυνδεθεί ο οποιοδήποτε χρήστης.

```
Network.Destroy(GetComponent<NetworkView>().viewID)
```

Καλείται για να καταστρέψουμε το αντικείμενο σε όλα τα instances του παιχνιδιού.

```
Network.Instantiate(playerPrefab,spawn.position,Quaternio
n.identity,0)
```

Καλείται για να δημιουργήσουμε ένα prefab σε όλα τα instances του παιχνιδιού.

2.8.3 NetworkView

Τα NetworkView είναι από τα πιο βασικά κομμάτια που πρέπει να χρησιμοποιούμε για να δημιουργήσουμε ένα διαδικτυακό παιχνίδι. Με την χρήση των NetworkView μας δίνετε η δυνατότητα να επικοινωνούν τα δεδομένα μεταξύ των συσκευών που είναι συνδεδεμένες στο ίδιο δίκτυο. Πιο αναλυτικά τα NetworkView σου δίνουν τις δύο παρακάτω λειτουργίες:

- Συγχρονίζουν τα δεδομένα των αντικειμένων που τους τα έχουμε εισάγει σε όλες τις συσκευές που είναι συνδεδεμένες σε αυτό το δίκτυο
- Δίνουν την δυνατότητα να χρησιμοποιήσουμε τις Remote Procedure Calls (RPC),μέσο των οποίων μπορούμε και επικοινωνούμε μεταξύ διαφορετικών υπολογιστών με την προϋπόθεση ότι είναι συνδεδεμένες στο ίδιο δίκτυο.

Πιο συγκεκριμένα η επιλογή του συγχρονισμού των δεδομένων στο αντικείμενο στο οποίο εισάγαμε το NetworkView γίνετε με δύο τρόπους , αξιόπιστο ή μη αξιόπιστο τρόπο.
Αν ο χρήστης επιλέξει να αξιόπιστο τρόπο γλιτώνει από άποψη bandwidth επειδή η Unity ελέγχει την κατάσταση που βρίσκετε το αντικείμενο με την κατάσταση που περιέχει το προηγούμενο πακέτο που είχε παραλάβει και στέλνει μόνο τις αλλαγές

(πχ αν αλλάξει η διεύθυνση ενός αντικειμένου αλλά η θέση του είναι ίδια τότε στέλνει μόνο την αλλαγή διεύθυνση ς), αλλά το μειονέκτημα είναι αν δεν λάβει ένα πακέτο τα υπόλοιπα πακέτα περιμένουν το πακέτο αυτό να ξανασταλθεί και περιμένουν μέσα στον buffer.
Αν ο χρήστης επιλέξει μη αξιόπιστο τρόπο τότε η Unity στέλνει όλη την πληροφορία χωρίς να ελέγξει εάν έχει αλλάξει κάτι καταναλώνοντας περισσότερο bandwidth.
Τα κομμάτια τα οποία συγχρονίζει το NetworkView είναι τα παρακάτω:

- Το transform component της Unity πιο συγκεκριμένα την θέση διεύθυνση και κλίμακα του αντικείμενου.
- Το Animation component της Unity πιο συγκεκριμένα για όλες τις animation καταστάσεις που έχει το Animation component το χρόνο , το βάρος , την ταχύτητα και εάν ή όχι είναι ενεργοποιημένο.
- Το Rigidbody component της Unity πιο συγκεκριμένα την θέση διεύθυνση , ταχύτητα , γωνιακή ταχύτητα και θέση του σώματος του αντικειμένου.

Σε περίπτωση που κάποιος ενδιαφέρετε μόνο να χρησιμοποιήσει RPC τότε μπορεί να επιλέξει off στην μέθοδο συγχρονισμού.
Τα RPC είναι συναρτήσεις που σου δίνουν την δυνατότητα να επικοινωνήσεις με ένα απομακρυσμένο υπολογιστή. Τα RPC έχουν δύο διαφορές από τις κανονικές συναρτήσεις. Οι διαφορές είναι οι παρακάτω:

- Τα RPC χρησιμοποιούν το bandwidth του δικτύου, που ενώ δεν έχεις όριο στον αριθμό των μεταβλητών που εισάγεις όσο τις αυξάνεις τόσο αυξάνετε το bandwidth που καταναλώνεις
- Με τα RPC καθορίζεις σε ποιόν θα στείλεις τα δεδομένα (στους άλλους υπολογιστές εκτός του εαυτού σου, στον server ή σε όλους τους υπολογιστές), επίσης έχεις την δυνατότητα να αποθηκεύσεις τα δεδομένα σε έναν buffer έτσι ώστε εάν συνδεθεί ένας καινούριος παίχτης τότε παίρνει τα νέα δεδομένα από τον buffer .

Ο τύπος δεδομένων που μπορούν να σταλθούν μέσο των RPC είναι ο παρακάτω:

- int
- float
- string
- NetworkPlayer
- NetworkViewID
- Vector3
- Quaternion
  Μέσω των RPC ουσιαστικά γίνετε η επικοινωνία μεταξύ των συσκευών που είναι μέσα στο δίκτυο και η αλλαγή μεταβλητών μέσα στα script που τους έχουμε εισάγει [4].

Κεφάλαιο 3
Σχεδιασμός παιχνιδιού και γραφικής διεπαφής

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναλύσουμε την λειτουργία του παιχνιδιού μας καθώς και την γραφική διεπαφή του παιχνιδιού με τον χρήστη.

3.2 Υπόθεση παιχνιδιού

Σε ένα κόσμο μακρινό ένα ζευγάρι μάγων προσπαθούσαν να δημιουργήσουν ένα φίλτρο που θα αύξανε τις μαγικές τους ικανότητες. Μετά από ένα ατύχημα το φίλτρο χύθηκε πάνω σε δυο βιβλία. Επειδή το πείραμα με το φίλτρο ήταν αποτυχημένο έφερε διαφορετικά αποτελέσματα. Η πτώση του φίλτρου πάνω στα βιβλία είχε σαν αποτέλεσμα να ζωντανέψει το περιεχόμενο τους και να δημιουργήσει δύο καινούριους κόσμους. Ο κάθε κόσμος που δημιούργησε ήταν επηρεασμένος από το περιεχόμενο του κάθε βιβλίου.

Το ένα βιβλίο περιείχε γεωμετρικά στερεά και το άλλο περιείχε δεινόσαυρους. Οι κόσμοι αυτοί ζούσαν ειρηνικά μεταξύ τους ώσπου κάποια στιγμή οι δεινόσαυροι κατάφεραν να μετακινηθούν από τον κόσμο τους στον κόσμο των στερεών. Ο στόχος των δεινοσαύρων έγινε η επικράτηση τους στον κόσμο των στερεών. Οι δεινόσαυροι με την βοήθεια ενός προδότη (του κύβου), καταφέρνουν να αιχμαλωτίσουν όλα τα στερεά εκτός από τη σφαίρα (που θα είναι ο κεντρικός χαρακτήρας του παιχνιδιού που θα χειριζόμαστε). Η σφαίρα προσπαθεί να μαζέψει αρκετή ποσότητα ενός φίλτρου, που δίνει τρομακτική δύναμη σε όποιον το πιει, και με την βοήθεια αυτού να καταφέρει να ελευθερώσει τους φυλακισμένους φίλους του. Η σφαίρα όταν μάζεψε αρκετή ποσότητα του φίλτρου καταφέρνει να ελευθερώσει τα φυλακισμένα στερεά και βρίσκει και ένα αιχμάλωτο μαζί τους. Ο αιχμάλωτος ήταν ο μάγος που πειραματίζονταν με το φίλτρο (είχε δοκιμάσει ένα άλλο φίλτρο αλλά και αυτό ήταν αποτυχημένο και τον μετέφερε στον κόσμο των στερεών). Ο μάγος αιχμαλωτίστηκε από τους δεινοσαύρους και φυλακίστηκε στο ίδιο μέρος που είχαν φυλακίσει και τα στερεά. Μετά από ανάκριση στον κύβο τα στερεά μάθανε ότι υπάρχουν κάποιες πύλες που δίνουν την δυνατότητα μετακίνησης μεταξύ κόσμων. Ο μάγος αποφασίζει να μπει σε αυτές τις πύλες για να μπορέσει να βρεθεί στον κόσμο του. Στο ενδιάμεσο τα στερεά αποφασίζουν να

φυλάνε τις πύλες που υπάρχουν στον κόσμο τους προκειμένου να μην εμφανιστούν άλλοι εχθροί από εκεί.

3.3 Αρχικό Μενού

Με την εμφάνιση του αρχικού μενού δίνεται η δυνατότητα στον χρήστη να επιλέξει μεταξύ Ελληνικών και Αγγλικών για την γλώσσα που θα λειτουργήσει το παιχνίδι.

Εικόνα 3.1 Αρχική οθόνη του παιχνιδιού

Η δυνατότητα επιλογής της γλώσσας γίνεται με το κουμπί πάνω δεξιά. Μετά την επιλογή της γλώσσας λειτουργίας του παιχνιδιού εμφανίζεται ένα καινούριο μενού με τις έξι παρακάτω επιλογές:

1) Ιστορία
2) Περιπέτεια
3) Επιπλέον επίπεδο
4) Σβήσιμο χαρακτήρα
5) Πληροφορίες χαρακτήρα
6) Παιχνίδι με φίλους

Εικόνα 3.2 Το κεντρικό μενού του παιχνιδιού

Οι επιλογές ένα και δύο αφορούν το παιχνίδι με ένα παίχτη που αναλύεται παρακάτω.

Πατώντας την επιλογή τρία (επιπλέον επίπεδο) εμφανίζεται η παρακάτω σκηνή και δίνοντας την δυνατότητα στον χαρακτήρα μας να αυξήσει ένα στατιστικό πληρώνοντας χίλια χρυσά νομίσματα (βλ. κεφ. 3.6.3).

Εικόνα 3.3 Στιγμιότυπο της σκηνής του επιπλέον επιπέδου

Εάν πατηθεί η επιλογή τέσσερα (σβήσιμο χαρακτήρα) σβήνονται όλα τα δεδομένα για το χαρακτήρα μας που είναι τοπικά αποθηκευμένα στον υπολογιστή μας.

Στην επιλογή πέντε (πληροφορίες χαρακτήρα) εμφανίζονται τα στατιστικά του χαρακτήρα μας.

Στην τελευταία επιλογή (παιχνίδι με φίλους) δημιουργούμε ή συμμετέχουμε σε διαδικτυακό παιχνίδι όπως αναλύεται παρακάτω.

3.4 Παιχνίδι ενός παίχτη

Εδώ θα αναλύσουμε την δομή του παιχνιδιού ενός παίχτη. Το παιχνίδι ενός παίχτη αποτελείται από δύο σκέλη, την ιστορία και την περιπέτεια.

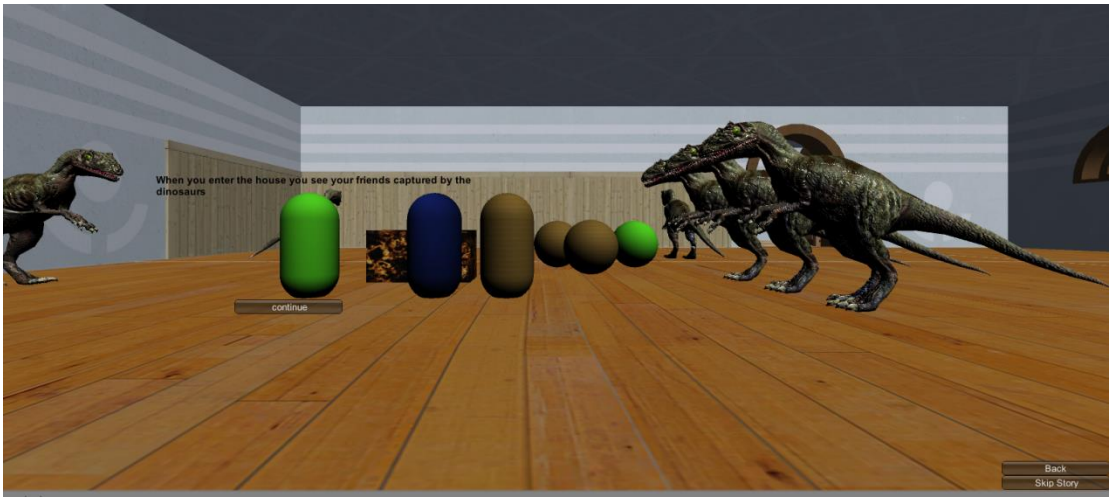
Το καθένα σκέλος έχει διαφορετικούς κύριους χαρακτήρες που χειρίζεται ο χρήστης και διαφορετικό τρόπο παιξίματος. Στο σκέλος της ιστορίας στην αρχή εξιστορούμε την υπόθεση του παιχνιδιού πλατφόρμας που υλοποιήσαμε. Στο σκέλος της περιπέτειας χρησιμοποιούμε έναν καινούριο χαρακτήρα (ο οποίος εμφανίζεται στο τέλος της ιστορίας) και τον χειριζόμαστε σε σκηνές με τρόπο λειτουργίας παιχνιδιού δράσης και με πολλά στοιχεία παιχνιδιού ρόλων (αύξηση στατιστικών, εμπειρία κλπ). Παρακάτω αναλύουμε τα δυο σκέλη παιχνιδιού ενός παίχτη.

3.5 Ιστορία

Η ιστορία χωρίζεται σε δυο σκέλη:

1) Την εκπαίδευση όπου ενημερώνουμε τον χρήστη για τον τρόπο χειρισμού του χαρακτήρα και εξιστορούμε την ιστορία του παιχνιδιού.
2) Διάφορα επίπεδα όπου ο χρήστης χειρίζεται τον χαρακτήρα και πρέπει να διασχίσει την σκηνή αποφεύγοντας εμπόδια για να φτάσει στο τέλος της πίστας.

Στο σκέλος της εκπαίδευσης εξιστορούμε την ιστορία του παιχνιδιού μέσα από οχτώ σκηνές.



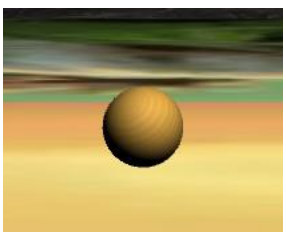
Εικόνα 3.4 Στιγμιότυπο της ιστορίας του παιχνιδιού

Στην συνέχεια αφού έχουμε αφηγηθεί την ιστορία αρχίζει το κομμάτι του παιχνιδιού πλατφόρμας (η αφήγηση του κομματιού της ιστορίας μπορεί να παραληφθεί πατώντας το κουμπί παράληψη) όπου ο κεντρικός χαρακτήρας προσπαθεί να μαζέψει κομμάτια του φίλτρου (που συνήθως βρίσκονται στο τέλος της σκηνής) για να μπορέσει να αποκτήσει αρκετή δύναμη να ελευθερώσει τους φίλους του.

3.6 Κομμάτι παιχνιδιού πλατφόρμας

Το πρώτο σκέλος του παιχνιδιού ουσιαστικά αποτελείτε από το κομμάτι παιχνιδιού πλατφόρμας. Στο κομμάτι παιχνιδιού πλατφόρμας το μοντέλο του χαρακτήρα που χειρίζεται ο χρήστης είναι μια σφαίρα. Ο σκοπός του παιχνιδιού είναι ο χαρακτήρας να μαζέψει το φίλτρο που υπάρχει κάπου στην πίστα. Στην σκηνή ο χρήστης βρίσκεται αντιμέτωπος με διάφορους εχθρούς. Ο χαρακτήρας μας έχει την δυνατότητα να πετάει πέτρες ώστε να τραυματίζει ή να εξουδετερώνει τους εχθρούς. Ο χρήστης όταν συναντάει το φίλτρο θα μεταφέρεται στο κύριο μενού και θα ξεκλειδώνεται το επόμενο επίπεδο παιχνιδιού για να μπορέσει να το επιλέξει.

3.6.1 Χειρισμός χαρακτήρα


Εικόνα 3.5 Ο χαρακτήρας μας στο πρώτο σκέλος του παιχνιδιού

Κουμπιά	Λειτουργίες χαρακτήρα
W,πάνω βελάκι	Πηγαίνει μπροστά

A, αριστερό βελάκι	Πηγαίνει αριστερά
S,κάτω βελάκι	Πηγαίνει πίσω
D,δεξί βελάκι	Πηγαίνει δεξιά
Δεξί κλικ ποντικιού	Ρίχνει πέτρες μπροστά και πίσω
Αριστερό κλικ ποντικιού	Ρίχνει πέτρες δεξιά και αριστερά
Space	Πηδάει

Πίνακας 3.1 Λειτουργίες χαρακτήρα στο πρώτο σκέλος του παιχνιδιού

Το ποντίκι χρησιμοποιείται μόνο για να εκτοξεύει ο χαρακτήρας μας πέτρες.

Παρακάτω εμφανίζουμε τα χαρακτηριστικά του παίχτη:

- ***Υγεία***. Η υγεία που έχει ο χαρακτήρας μας, μειώνεται όταν τραυματίζεται από εχθρούς και αυξάνεται όταν μαζεύει φίλτρα θεραπείας.

Εικόνα 3.6 Η μπάρα υγείας του χαρακτήρα

- ***Ζωές***. Είναι πέντε οι ζωές που έχει ο χαρακτήρας, αυτές μειώνονται εάν η ζωή του παίχτη πάει στο μηδέν. Όταν μηδενιστούν και ο παίχτης χρειάζεται και άλλη ζωή το επίπεδο ξαναρχίζει από την αρχή.

Εικόνα 3.7 Ο αριθμός ζωών του χαρακτήρα

3.6.2 Εχθροί και η λειτουργία τους

Σε αυτό το κομμάτι παρουσιάζουμε τους εχθρούς που συναντάει ο χαρακτήρας κατά την διάρκεια της πίστας και αναλύουμε την λειτουργία τους:

Εικόνα 3.8 Ο προδότης κύβος

Εικόνα 3.9 Ο εχθρός δεινόσαυρος

Ο κύβος όπως και ο δεινόσαυρος ακολουθούν τους παρακάτω κανόνες:

1) Ελέγχουν αν είναι κοντά στον παίχτη. Αν είναι τότε πηγαίνουν προς τον παίχτη. Αν δεν είναι μένουν στην θέση τους.

2) Εάν είναι ο παίχτης κοντά τους ο κύβος του πετάει μπλε σφαίρες και τον ακολουθεί, ενώ ο δεινόσαυρος τον πλησιάζει και μόνο αν είναι δίπλα του, του επιτίθεται.
3) Εάν κυνηγώντας τον παίχτη φτάσουν στο τέλος της πλατφόρμας ή ο παίχτης απομακρυνθεί αρκετά τότε επιστρέφουν στο μέρος όπου ήταν αρχικά.

3.6.3 Διάφορα αντικείμενα

Παρακάτω παρουσιάζουμε κάποια αντικείμενα που συναντάει ο χρήστης κατά την διάρκεια του παιχνιδιού πλατφόρμας.

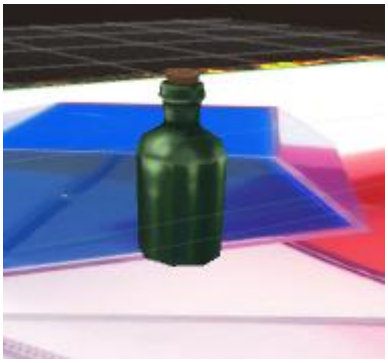
- ***Ποτό θεραπείας***. Μαζεύοντας το ποτό ο χρήστης θεραπεύει ένα πόντο υγείας που είχε χάσει.

Εικόνα 3.10 Το ποτό θεραπείας

- ***Χρυσά νομίσματα***. Τα χρησιμοποιεί ο χρήστης για να ανεβάσει τα στατιστικά του. Τα χρυσά νομίσματα μπορεί να τα βρει κατά την διάρκεια της πίστας σκοτώνοντας εχθρούς ή σκορπισμένα στο έδαφος.

Εικόνα 3.11 Τα χρυσά νομίσματα

- ***Τέλος πίστας***. Συναντώντας το τέλος πίστας ο χρήστης αποθηκεύει ότι χρυσά νομίσματα είχε μαζέψει και μετακινείται στο κεντρικό μενού ξεκλειδώνοντας το επόμενο επίπεδο.



Εικόνα 3.12 Το αντικείμενο που καθορίζει το τέλος της πίστας

- ***Σημεία επαναφοράς***. Υπάρχουν μερικά σημεία από τα οποία όταν περνάει ο χρήστης του δίνουν την δυνατότητα σε περίπτωση που πεθάνει να μπορεί να επιστρέφει σε αυτά τα σημεία και όχι στην αρχή της σκηνής.

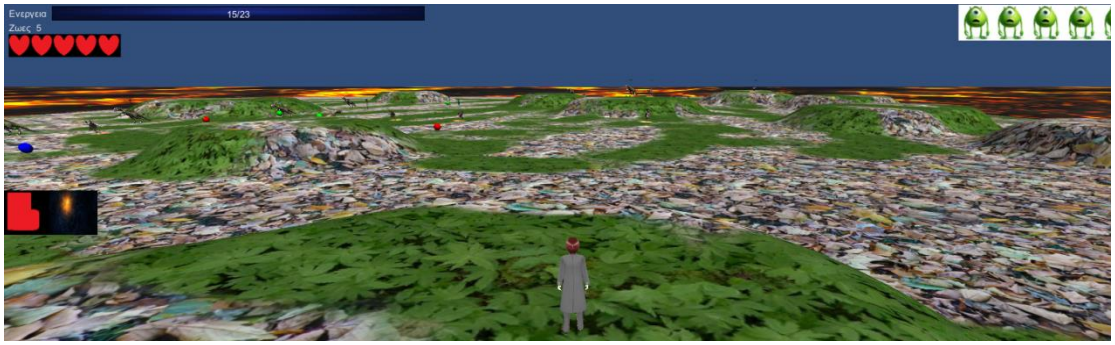
3.7 Δομή της περιπέτειας

Το κομμάτι της περιπέτειας είναι το δεύτερο σκέλος τις εργασίας, αποτελείται από δύο διαφορετικά κομμάτια: το παιχνίδι δράσης και το turn-base. Στο μενού της περιπέτειας δίνεται η επιλογή στον χρήστη να επιλέξει ποιο από τα δύο κομμάτια προτιμάει να παίξει. Παρακάτω θα αναλύσουμε αυτά τα δύο κομμάτια.

3.8 Κομμάτι δράσης της περιπέτειας

Στο κομμάτι δράσης ο χαρακτήρας της περιπέτειας μπαίνει στην τρισδιάστατη σκηνή και συναντά εχθρούς σε διάφορα σημεία της. Εάν ο χαρακτήρας μας πλησιάσει ή επιτεθεί σε κάποιον εχθρό τότε ο εχθρός έρχεται προς το μέρος του και επιτίθεται.

Στόχος του χαρακτήρα είναι να εξουδετερώσει πέντε συνολικά εχθρούς για να μεταβεί στο επόμενο επίπεδο. Όταν σκοτωθούν πέντε εχθροί εμφανίζεται μια πύλη σε συγκεκριμένο σημείο στη σκηνή. Έτσι όταν ο χαρακτήρας μας πάει πάνω της μεταβαίνει στο επόμενο επίπεδο. Όταν φτάσει στο τέλος της πίστας μετακινείται σε μια νέα σκηνή στην οποία υπάρχει ένα όπλο. Το όπλο αυξάνει τα στατιστικά του παίχτη. Ένας παίχτης μπορεί να έχει μόνο ένα όπλο κάθε φορά. Εάν επιλέξει να αποκτήσει το καινούριο όπλο το παλιό χάνεται.



Εικόνα 3.13 Στιγμιότυπο δράσης της ιστορίας

Παρακάτω εξηγούνται τα βασικά στοιχεία της γραφικής διεπαφής χρήστη σε αυτό το επίπεδο.

Πάνω δεξιά είναι τα εικονίδια με τους εναπομείναντες εχθρούς που χρειάζεται να αντιμετωπίσουμε για να εμφανιστεί η πύλη για το επόμενο επίπεδο.

Πάνω αριστερά είναι η ενέργεια του χαρακτήρα. Εάν η ενέργεια πάει στο μηδέν αφαιρείται μια καρδιά (ζωή) που υπάρχει ακριβώς από κάτω. Αν δεν υπάρχουν άλλες καρδιές και η ενέργεια έχει πάει ξανά στο μηδέν ο χαρακτήρας χάνει και η πίστα ξαναρχίζει.

Μέση αριστερά είναι μερικές ειδικές κινήσεις που μπορεί να κάνει ο χαρακτήρας μας (πατώντας συγκεκριμένα κουμπιά θα βλέπε παρακάτω).

3.8.1 Χειρισμός χαρακτήρα

Παρακάτω παρουσιάζονται τα κουμπιά που πατάει ο χρήστης ώστε να μπορεί να χειριστεί τον χαρακτήρα. Η είσοδος από το χρήστη γίνονται μέσω πληκτρολογίου και ποντικιού συνδυαστικά δεν μπορείς να χειριστεί ο χρήστης τον χαρακτήρα μεμονωμένα μόνο με το ποντίκι ή μόνο με το πληκτρολόγιο.

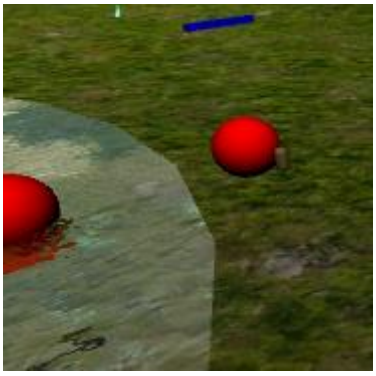
Κουμπιά	**Λειτουργίες χαρακτήρα**
W, πάνω βελάκι	Πηγαίνει μπροστά
A, αριστερό βελάκι	Πηγαίνει αριστερά
S, κάτω βελάκι	Πηγαίνει πίσω
D, δεξί βελάκι	Πηγαίνει δεξιά
Δεξί κλικ ποντικιού	Κάνει επίθεση από κοντά χτυπώντας όλους τους εχθρούς που έχει δίπλα του
Αριστερό κλικ ποντικιού	Κάνει επίθεση από μακριά ρίχνοντας μια μπάλα φωτιάς στην κατεύθυνση που κοιτάζει
Space	Αποκτάει μεγαλύτερη ταχύτητα για λίγο χρόνο
C	Εκτοξεύει μια τεράστια μπάλα μαγείας που όποιον εχθρό τον ακουμπήσει του μειώνει στο μισό την υπόλοιπη υγεία του
V	Μετακινεί την κάμερα του χαρακτήρα ανάποδα ώστε να μπορεί να βλέπει και πίσω του

Πίνακας 3.2 Λειτουργίες χαρακτήρα κομμάτι δράσης

3.8.2 Είδη εχθρών και λειτουργία τους

Οι εχθροί που υπάρχουν στο κομμάτι δράσης εμφανίζονται σε διάφορα σημεία. Οι εχθροί κινούνται μεταξύ δύο σημείων και εάν συναντήσουν τον χαρακτήρα ή αν ο χαρακτήρας τους επιτεθεί κινούνται προς το μέρος του και του κάνουν επίθεση. Εάν σκοτωθούν ενημερώνουν ότι πέθαναν (ώστε να μειωθεί ο αριθμός τον εχθρών που υπολείπονται), δίνουν ένα αριθμό εμπειρίας στον χαρακτήρα μας και αφήνουν στο έδαφος χρυσά νομίσματα. Ο αριθμός των διαφορετικών εχθρών είναι πέντε, αναλυτικότερα οι εχθροί είναι:

- Μια απλή σφαίρα που κυνηγάει τον χαρακτήρα μας και του επιτίθεται από μακριά.



Εικόνα 3.14 Ο εχθρός σφαίρα

- Ένας δεινόσαυρος που κυνηγάει τον χαρακτήρα μας και του πετάει μπάλες φωτιάς.

Εικόνα 3.15 Επίθεση δεινοσαύρου από μακριά

- Ένας δεινόσαυρος κυνηγάει τον χαρακτήρα μας και του επιτίθεται.

Εικόνα 3.16 Επίθεση δεινοσαύρου από κοντά

- Ένας σκελετός κυνηγάει τον χαρακτήρα μας και όταν τον πλησιάσει του επιτίθεται.

Εικόνα 3.17 Επίθεση σκελετού από κοντά

- Ένας σκελετός που κυνηγάει τον χαρακτήρα μας και του πετάει μπάλες φωτιάς.

Εικόνα 3.18 Επίθεση σκελετού από μακριά

3.8.3 Διάφορα αντικείμενα

Παρακάτω αναλύουμε τα υπόλοιπα αντικείμενα που μπορεί να συναντήσει ο χρήστης μας στο κομμάτι δράσης της περιπέτειας.

- ***Κομμάτι εμφάνισης εχθρών (Spawner)***. Ο Spawner έχει σκοπό να εμφανίσει τον εχθρό και να του δώσει οδηγίες για το ποιο μονοπάτι θα ακολουθήσει.

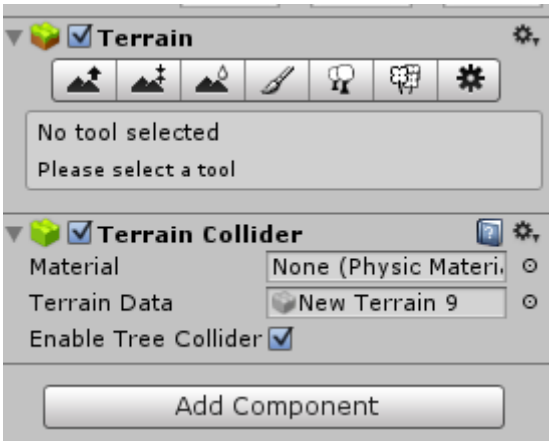
Εικόνα 3.19 Ο spawner

- ***Τέλος πίστας***. Όταν εμφανιστεί η παραπάνω πύλη και ο χαρακτήρας μας πάει πάνω της τότε τελειώνει η πίστα, αποθηκεύονται τα στατιστικά που έχει ο παίχτης (εμπειρία που έχει πάρει, αύξηση στατιστικών και χρυσά νομίσματα) και μετακινείται στην σκηνή όπου επιλέγει όπλο.

Εικόνα 3.20 Το τέλος πίστας

3.8.4 Σχεδιασμός επιπέδων

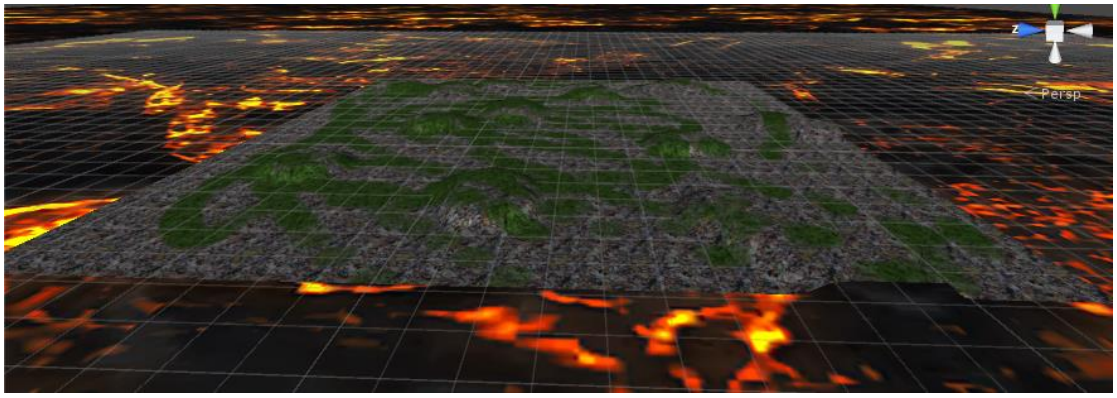
Τα επίπεδα που δημιουργήσαμε στο κομμάτι δράσης της περιπέτειας δημιουργήθηκαν με την χρήση του terrain editor της Unity.



Εικόνα 3.21 Ο Terrain editor του unity

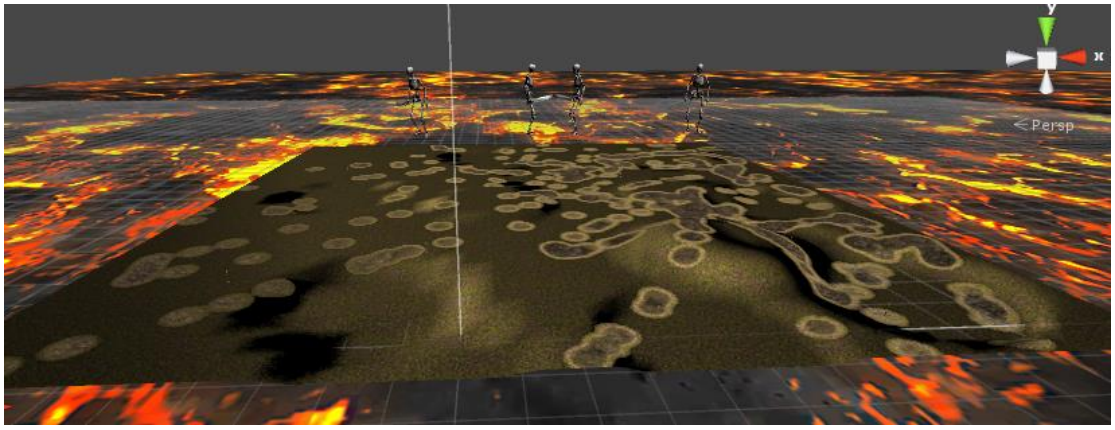
Συνολικά δημιουργήσαμε τέσσερα επίπεδα παιχνιδιού. Τα παρακάτω:

- Ένα επίπεδο στο οποίο οι εχθροί που εμφανίζονται είναι σκελετοί, δεινόσαυροι και σφαίρες.



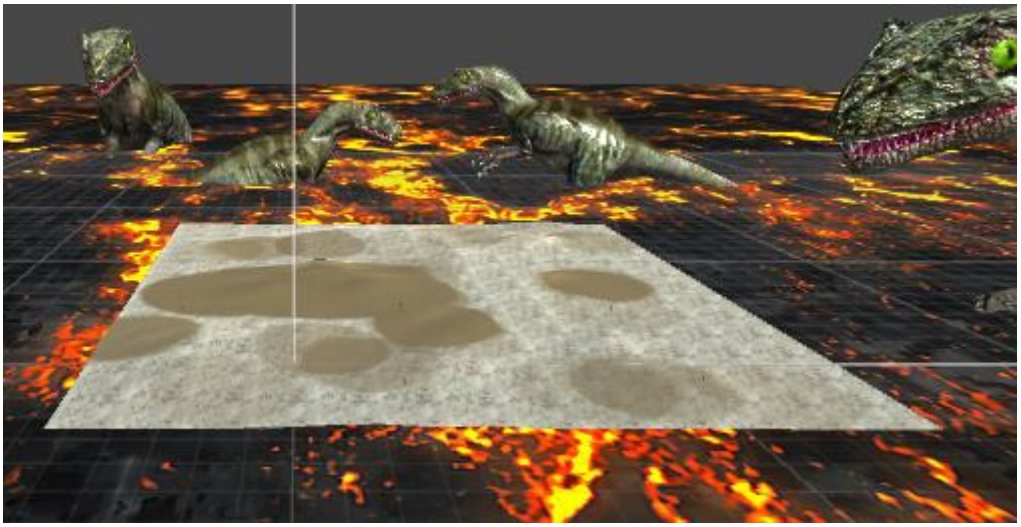
Εικόνα 3.22 Πίστα στο κομμάτι δράσης περιπέτειας

- Ένα επίπεδο στο οποίο οι εχθροί που εμφανίζονται είναι μόνο σκελετοί.



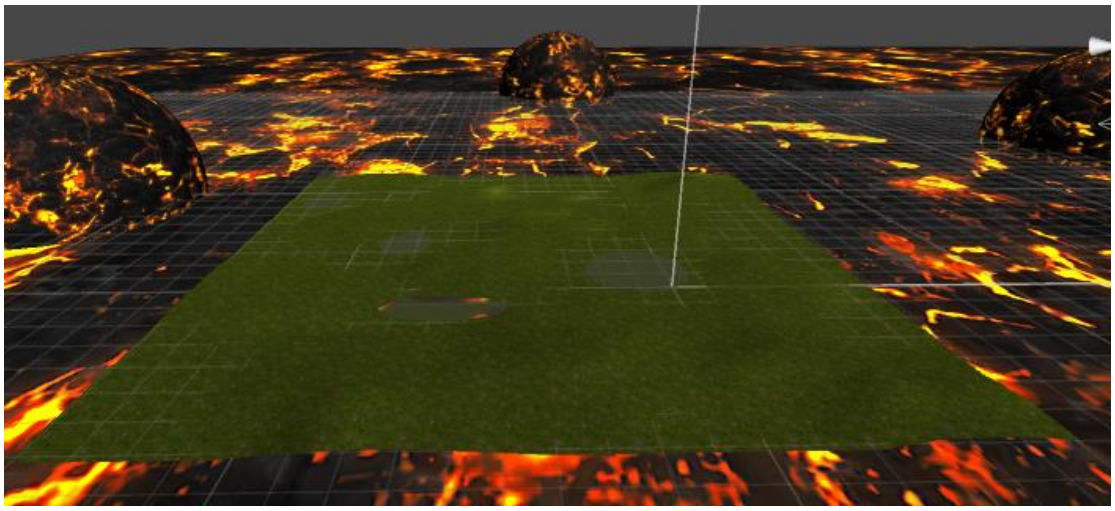
Εικόνα 3.23 Πίστα στο κομμάτι δράσης περιπέτειας

- Ένα επίπεδο στο οποίο οι εχθροί που εμφανίζονται είναι μόνο δεινόσαυροι.



Εικόνα 3.24 Πίστα στο κομμάτι δράσης περιπέτειας

- Ένα επίπεδο στο οποίο οι εχθροί που εμφανίζονται είναι μόνο σφαίρες.



Εικόνα 3.25 Πίστα στο κομμάτι δράσης περιπέτειας

Η επιλογή των παραπάνω επιπέδων γίνεται τυχαία. Έτσι όταν ο χρήστης φτάνει στο τέλος κάποιου επιπέδου τότε ξεκινάει ένα άλλο τυχαία.

3.9 Ανάλυση δομής turn-base επιπέδου

Το turn-base επίπεδο χωρίζεται σε δύο επίπεδα. Τα επίπεδα που το αποτελούν είναι: Το επίπεδο συνάντησης με τον εχθρό και το επίπεδο μάχης.

3.9.1 Επίπεδο συνάντησης με τον εχθρό

Στο επίπεδο συνάντησης με τον εχθρό ο χαρακτήρας μας προσπαθεί να συναντήσει εχθρούς, ώστε να μεταφερθεί στο επίπεδο μάχης και να τους πολεμήσει. Εάν έχει

σκοτώσει συνολικά πέντε εχθρούς εμφανίζεται η πύλη για το τέλος της πίστας όπως στο κομμάτι δράσης και δίνεται η δυνατότητα αλλαγής όπλου.

3.9.2 Επίπεδο μάχης

Στην σκηνή του επιπέδου μάχης στο δεξί μέρος βρίσκεται ο χαρακτήρας μας και αριστερά ένας τυχαίος αριθμός από εχθρούς. Ο αριθμός εχθρών που μπορεί να εμφανιστεί σε αυτό το επίπεδο είναι μεταξύ ένα και τρία και επιλέγεται τυχαία. Ο τύπος των εχθρών είναι ίδιος με τον τύπο του εχθρού που συγκρούστηκε με τον χαρακτήρα μας στο επίπεδο συνάντησης με τον εχθρό. Πάνω αριστερά στην οθόνη μας βλέπουμε τι κινήσεις μπορούμε να κάνουμε και επιλέγουμε την επόμενη ενέργεια. Οι επιλογές είναι οι παρακάτω:

- *Σειρά χρήστη*
    - Επίθεση από κοντά
    - Επίθεση από μακριά

- *Σειρά εχθρού*
    - Επίθεση εχθρού

Πάνω δεξιά είναι η υγεία των εχθρών και η υγεία του παίχτη μας. Όταν τελειώσει η μάχη εάν έχουμε σκοτώσει όλους τους εχθρούς γυρίζουμε στο επίπεδο της περιπέτειας και αφαιρούμε τον αριθμό των εχθρών που σκοτώσαμε (κάθε εχθρός που σκοτώνουμε δίνει εμπειρία).

Εικόνα 3.26 Στιγμιότυπο της μάχης στο turn-base με δεινοσαύρους

Εικόνα 3.27 Στιγμιότυπο της μάχης στο turn-base με σκελετούς

3.10 Ανάλυση Παιχνιδιού με φίλους

Το παιχνίδι με φίλους είναι το τρίτο σκέλος της εργασίαςκαι ουσιαστικά είναι το παιχνίδι με δυνατότητα συμμετοχής πολλών παιχτών κομμάτι της εργασίας. Το κομμάτι με δυνατότητα συμμετοχής πολλών παιχτών είναι ένα παιχνίδι τύπου πρώτου προσώπου shooter. Ο κάθε χρήστης έχει τα στατιστικά που έχει ο χαρακτήρας του στο παιχνίδι ενός παίχτη και παίζει σε τρεις διαφορετικές πίστες μαζί με τους φίλους του.

3.10.1 Βήματα δήλωσης νέου παιχνιδιού

Στην αρχή της δημιουργίας του παιχνιδιού με δυνατότητα συμμετοχής πολλών παιχτών, εμφανίζεται ένα μενού όπου πρέπει να δηλώσουμε και να αποθηκεύσουμε το όνομα του χαρακτήρα μας.

Εικόνα 3.28 Το κεντρικό μενού συμμετοχής πολλών παιχτών

Στη συνέχεια εμφανίζεται ένα μενού το οποίο μας δίνει την δυνατότητα να αποθηκεύσουμε το όνομα του παιχνιδιού και να ορίσουμε τον μέγιστο αριθμό

παιχτών που μπορούν να συνδεθούν στο παιχνίδι (στο δικό μας παιχνίδι λόγω υλοποίησης μπορούμε να έχουμε μόνο οχτώ παίχτες να συμμετέχουν σε ένα παιχνίδι).

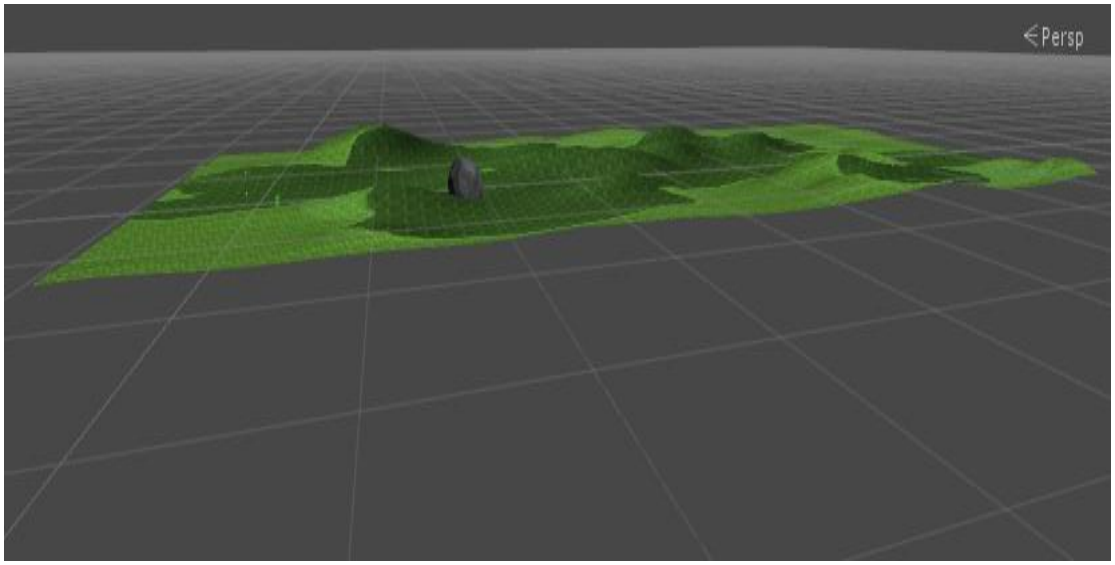
Εικόνα 3.29 Το μενού δημιουργίας παιχνιδιού

Στο τέλος επιλέγουμε τον τύπο της πίστας και το επίπεδο δυσκολίας.

3.10.2 Επίπεδα του παιχνιδιού

Τα επίπεδα παιχνιδιού στο κομμάτι παιχνιδιού με δυνατότητα συμμετοχής πολλών παιχτών είναι συνολικά τρία.

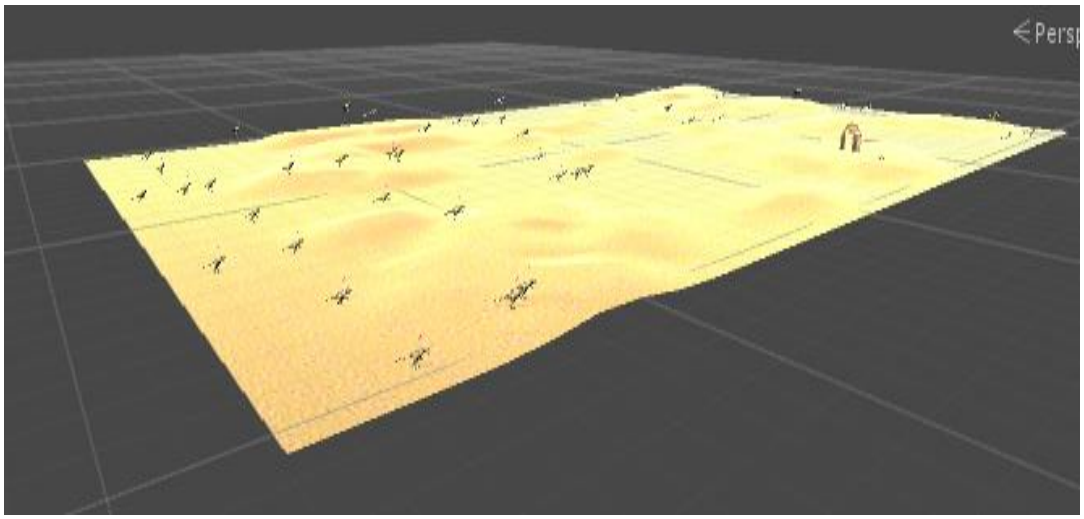
Στο πρώτο επίπεδο ο σκοπός του παιχνιδιού είναι οι χρήστες να αντιμετωπίσουν ο ένας τον άλλον μέχρι να μείνει ο τελευταίος επιζών όπου νικάει το παιχνίδι. Όταν ξεκινάει το επίπεδο ο κάθε χρήστης εμφανίζεται σε διάφορα σημεία της πίστας και προσπαθεί να βρει τους άλλους χρήστες για να τους εξουδετερώσει πυροβολώντας τους (πατώντας το δεξί κλικ). Όταν μείνει ο τελευταίος παίχτης ζωντανός τότε το παιχνίδι τελειώνει.



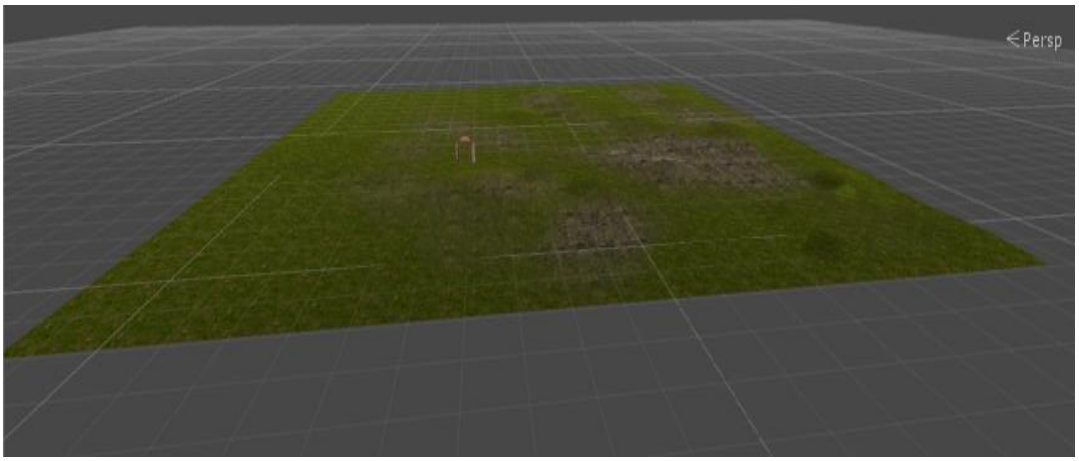
Εικόνα 3.30 Το πρώτο επίπεδο συμμετοχής πολλών παιχτών

Το δεύτερο και τρίτο επίπεδο είναι στην ουσία πίστες συνεργασίας μεταξύ τον παικτών. Ο χρήστης σκοτώνει με τη βοήθεια των άλλων χρηστών δεινοσαύρους. Οι

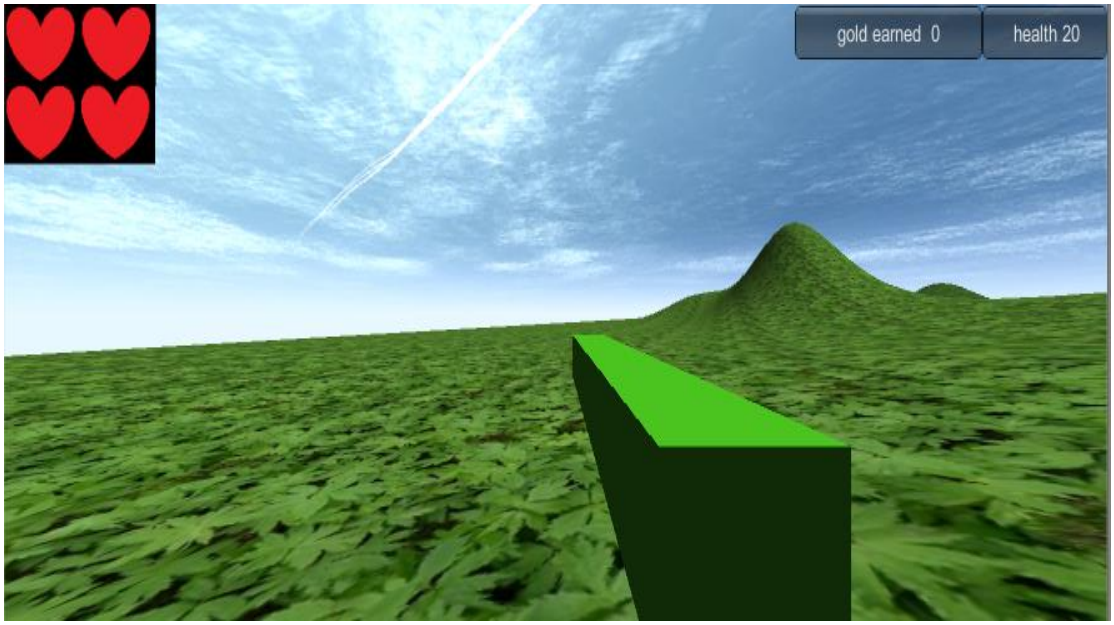
δεινόσαυροι αφήνουν χρυσά νομίσματα και οι χρήστες τα μαζεύουν. Το παιχνίδι τελειώνει όταν εξοντωθούν όλοι οι χρήστες (ήττα) ή όταν φτάσει ένας χρήστης στον τερματισμό (νίκη).



Εικόνα 3.31 Το δεύτερο επίπεδο συμμετοχής πολλών παιχτών



Εικόνα 3.32 Το τρίτο επίπεδο συμμετοχής πολλών παιχτών



Εικόνα 3.33 Στιγμιότυπο του χρήστη

Πάνω αριστερά είναι οι ζωές που έχει ο χρήστης. Εάν τελειώσουν και φτάσει η ενέργεια στο μηδέν χάνει. Πάνω δεξιά είναι η ενέργεια του παίχτη. Εάν η ενέργεια του χρήστη φτάσει στο μηδέν τότε αφαιρεί μια ζωή από τα αριστερά και του δίνεται η δυνατότητα να ξαναεμφανιστεί στην πίστα. Δίπλα βρίσκεται ένας μετρητής που ενημερώνει τον χρήστη πόσο χρυσό έχει μαζέψει.

Κεφάλαιο 4
Υλοποίηση παιχνιδιού

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα μιλήσουμε αναλυτικά για το προγραμματιστικό και τεχνικό κομμάτι του παιχνιδιού [2],[14],[15],[16],[17].
Στο πρόγραμμα μας εισάγουμε ή δημιουργούμε αντικείμενα και πάνω σε αυτά βάζουμε ένα ή περισσότερα συστατικά (components) της Unity ώστε να κάνουν τις λειτουργίες που θέλουμε.

4.2 Οργάνωση κεντρικών μενού και παιχνιδιών

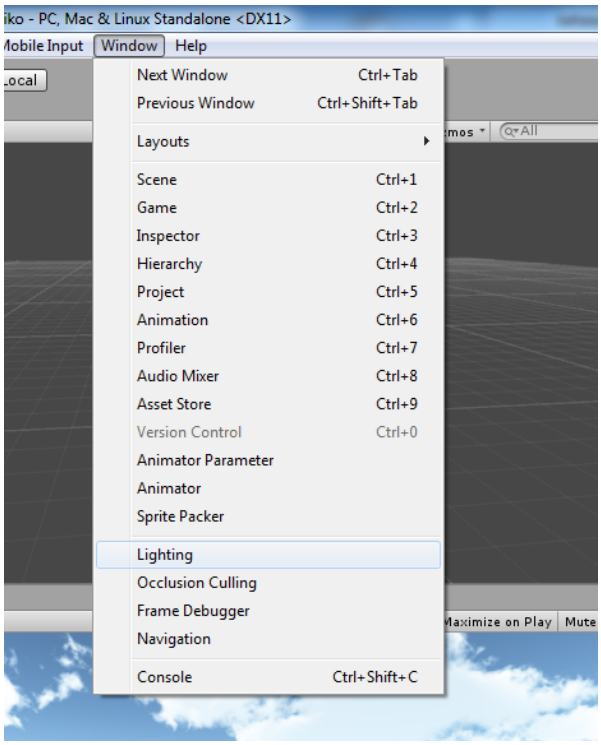
4.2.1 Κεντρικό μενού

Εικόνα 4.1 Το αρχικό μενού του παιχνιδιού

Στην αρχή όταν βρισκόμαστε στο κεντρικό μενού δημιουργούμε ένα άδειο αντικείμενο (τον GameControler). Αυτό έχει ένα C# script στο οποίο αποθηκεύουμε όλα τα βασικά χαρακτηριστικά που θέλουμε (π.χ. σε τι γλώσσα είναι το παιχνίδι) και τον μαρκάρουμε βάζοντας του ένα tag (dm). Επειδή τα αντικείμενα καταστρέφονται κάθε φορά που αλλάζουμε σκηνή σε μερικά αντικείμενα που δεν θέλουμε να καταστραφούν (όπως ο GameControler) έχουμε βάλει την παρακάτω γραμμή κώδικα:

```
DontDestroyOnLoad(this)
```

Επίσης εισάγαμε ένα script στην κύρια κάμερα με το οποίο δημιουργούμε το κουμπί (με την βοήθεια της συνάρτησης `OnGUI()` της Unity) που επιλέγουμε την γλώσσα που θα υπάρχει στο παιχνίδι. Πατώντας το κεντρικό κουμπί μεταφερόμαστε στο μενού του παιχνιδιού. Για την εικόνα που έχουμε βάλει στο background απλά χρησιμοποιήσαμε την λειτουργία Skybox και εισάγαμε την εικόνα που θέλαμε.



Εικόνα 4.2 Το μενού για επιλογή του lightning της Unity

4.2.2 Υπόλοιπα μενού παιχνιδιών

Όλα τα υπόλοιπα μενού έχουν σαν κύριο κορμό ένα C# script το οποίο ψάχνει τον GameControler με την βοήθεια της συνάρτησης της Unity.

```
GameObject.FindGameObjectWithTag("tag.name")
```

Έτσι βάζοντας το tag του GameControler (dm) βρίσκουμε το αντικείμενο και βλέπουμε εάν είναι η γλώσσα ελληνικά ή όχι (μέσω μιας bool του Gamecontroler). Στην συνέχεια εμφανίζουμε τα κείμενα που έχουμε στις επιλογές σε Αγγλικά ή Ελληνικά μέσω πάλι της συνάρτησης της Unity `OnGUI()`.

Τα μενού στο παιχνίδι εμφανίζονται πατώντας το πλήκτρο ESC. Όταν πατηθεί το μενού θέλουμε να σταματήσουμε ότι γίνεται στο παιχνίδι. Αυτό το καταφέρνουμε χρησιμοποιώντας την συνάρτηση της Unity `Time.Scale()` και με όρισμα μηδέν. Για να δούμε εάν πατήθηκε το ESC βάζουμε την ακόλουθη συνάρτηση μέσα στο `OnGUI()`.

```
if (Input.GetKeyDown (KeyCode.Escape)) {}
```

4.3 Χαρακτηριστικά παιχνιδιού και μέθοδοι αποθήκευσης

Για την μέθοδο αποθήκευσης επιλέξαμε την μέθοδο της Playerprefs, της Mono development. Τα δεδομένα που αποθηκεύουμε και τα χαρακτηριστικά τους εμφανίζονται στον παρακάτω πίνακα:

Χαρακτηριστικό	Περιγραφή
Όνομα	Το όνομα του χαρακτήρα μας.
Δύναμη	Αυξάνοντας το αυξάνεται η ζημιά που προκαλούμε σε εχθρούς από κοντά.
Ταχύτητα	Αυξάνοντας το αυξάνεται η ζημιά που προκαλούμε σε εχθρούς από μακριά.
Ευελιξία	Αυξάνοντας το αυξάνεται η αντοχή στην ζημιά που προκαλείται από εχθρούς που επιτίθενται από μακριά.
Συγκέντρωση	Αυξάνοντας το αυξάνεται η αντοχή στην ζημιά που προκαλείται από εχθρούς που επιτίθενται από κοντά.
Αντοχή	Αυξάνοντας το αυξάνεται η υγεία του χαρακτήρα μας.
Επίθεση από κοντά όπλου	Αυξάνοντας το αυξάνεται η ζημιά που προκαλούμε σε εχθρούς από κοντά.
Επίθεση από μακριά όπλου	Αυξάνοντας το αυξάνεται η ζημιά που προκαλούμε σε εχθρούς από μακριά.
Άμυνα από μακριά όπλου	Αυξάνοντας το αυξάνεται η αντοχή στην ζημιά που προκαλείται από εχθρούς που επιτίθενται από μακριά.
Άμυνα από κοντά όπλου	Αυξάνοντας το αυξάνεται η αντοχή στην ζημιά που προκαλείται από εχθρούς που επιτίθενται από κοντά.
Χρυσός	Ο χρυσός που έχει ο παίχτης.
Εμπειρία του όπλου	Η εμπειρία που έχει το όπλο.
Εμπειρία του παίχτη	Η εμπειρία που έχει ο χαρακτήρας μας.

Επίπεδο χαρακτήρα	Το επίπεδο που έχει ο χαρακτήρας μας.
Επίπεδο όπλου	Το επίπεδο που έχει το όπλο.
Πόσες πιστές έχει τερματίσει στην ιστορία	Ο συνολικός αριθμός διαφορετικών πιστών που έχει τερματίσει ο χαρακτήρας μας.

Πίνακας 4.1 Χαρακτηριστικά παιχνιδιού

Η αποθήκευση του κάθε χαρακτηριστικού γίνεται ανάλογα με το τι τύπου είναι (integer, float ή string). Η αποθήκευση γίνεται με μια από τις παρακάτω τρεις συναρτήσεις `SetFloat`, `SetInt`, `SetString` και αντίστοιχα παίρνουμε τα δεδομένα πίσω με τις παρακάτω συναρτήσεις `GetFloat`, `GetInt`, `GetString`.

4.4 Αντικείμενο μουσικής

Για να υπάρξει ήχος στο παιχνίδι δημιουργούμε ένα αντικείμενο με AudioSource (component της Unity). Στην συνέχεια εισάγουμε ένα script στο αντικείμενο με τον παρακάτω κώδικα για να δώσουμε την δυνατότητα στον χρήστη να κάνει της παρακάτω ενέργειες:

Κουμπιά	Λειτουργίες
K,L	Αύξηση-Μείωση έντασης ήχου
P	Σταμάτημα της μουσικής
M	Κλείσιμο τελείως του ήχου

Πίνακας 4.2 Κουμπιά λειτουργίας μουσικής

Ο κώδικας είναι ο παρακάτω:

```
using UnityEngine;
using System.Collections;

public class music : MonoBehaviour {


void Awake(){

            AudioListener.volume = 0.5f;

      }
      // Update is called once per frame
      void Update () {
            if (Input.GetKeyDown (KeyCode.P)) {

                  AudioListener.pause = !AudioListener.pause;

            }
            if (Input.GetKeyDown (KeyCode.M)) {

                  AudioListener.volume = 0f;
```

```
            }
            if (Input.GetKeyDown (KeyCode.L)) {
                  if(AudioListener.volume >0.1f)
                  {AudioListener.volume = AudioListener.volume -0.1f;
                  }
            }
            if (Input.GetKeyDown (KeyCode.K) ){
                  if(AudioListener.volume <0.9f)
                  {AudioListener.volume = AudioListener.volume +0.1f;
                  }
            }
      }
}
```

4.5 Αφήγηση ιστορίας

Εδώ θα αναλύσουμε την υλοποίηση του πρώτου κομμάτι του παιχνιδιού, το κομμάτι της εκπαίδευσης. Μέσα στη σκηνή υπάρχει ένα κομμάτι μουσικής για να μπορεί να ακούγεται ο ήχος στο παιχνίδι μας (βλ 4.4). Υπάρχει ένα C# script στην κάμερα που ή κρατάει τη θέση της κάμερας σταθερή ή την μετακινεί μεταξύ των αντικειμένων που βρίσκονται μέσα στην τρισδιάστατη σκηνή. Δημιουργήσαμε μερικά C# script και τα εισάγαμε σε διάφορα αντικείμενα ώστε να μπορούν να κινούνται στο χάρτη (π.χ. η μετακίνηση των εχθρών από το ένα σημείο στο άλλο). Τέλος χρησιμοποιούμε τον Canvas της Unity ώστε να έχουμε την δυνατότητα να εμφανίζουμε τα μηνύματα στον χρήστη.

Αυτό γίνεται ενεργοποιώντας ή απενεργοποιώντας κάποια κομμάτια κειμένου που βρίσκονται μέσα στο Canvas. Παρακάτω είναι ένα ενδεικτικό κομμάτι κώδικα που χρησιμοποιήσαμε για να αλλάξει το κείμενο που βλέπει ο χρήστης. Στην αρχή ορίζουμε τα αντικείμενα που υπάρχουν στην οθόνη.

```
public GameObject text1;//ta text einai ta keimena pou emfanizontai
ston xristi
public GameObject text2;
public GameObject text3;
public GameObject text4;
public GameObject text5;
public GameObject text6;
public GameObject dino;//edw einai ta gameobject pou energopoioume
analogo me to poio seimeio //tisistorias eimaste
public GameObject dino3;
public GameObject dino2;
public GameObject enisxiseis1;
public GameObject enisxiseis2;
public GameObject enisxiseis3;
public GameObject enisxiseis4;
public dinoai22 dino11;
public dinoai22 dino12;
public dinoai22 dino13;
public GameObject p;
public GameSettings2 scr;
private bool elinika;

private int j=0;
```

```
void Start () {
            dino11 = dino.GetComponent<dinoai22> ();
            dino12 = dino2.GetComponent<dinoai22> ();
            dino13 = dino3.GetComponent<dinoai22> ();
            p=GameObject.FindWithTag("dm");
            if(p!=null){
                  scr=p.GetComponent<GameSettings2>();
                  elinika=scr.Elinika;
            }
            if(elinika)
                  {text1.SetActive(true);}
            else
                  {text4.SetActive(true);}
      }
```

Παρακάτω με τη βοήθεια της `OnGUI()` καθορίζουμε ποιο κείμενο θα εμφανιστεί όπως και ποια αντικείμενα της σκηνής μας θα είναι ενεργοποιημένα (καθώς και τι κάνουν).

```
void  OnGUI()
{
      if (j < 3) {
            if (elinika) {
                  if (GUI.Button (new Rect ((Screen.width) - 150,
                  Screen.height / 2 + 20, 150, 20), "συνεχεια")) {
                              if (j == 0) {
                                    text1.SetActive (false);
                                    text2.SetActive (true);
                                    j++;

                                    dino.SetActive (true);
                                    dino3.SetActive (true);
                                    dino2.SetActive (true);
                              } else
                              if (j == 1) {
                                    text2.SetActive (false);
                                    text3.SetActive (true);
                                    j++;
                                    dino11.elenxos = true;
                                    dino12.elenxos = true;
                                    dino13.elenxos = true;
                                    enisxiseis1.SetActive (true);
                                    enisxiseis2.SetActive (true);
                                    enisxiseis3.SetActive (true);
                                    enisxiseis4.SetActive (true);
                              } else
                              if (j == 2) {

                                    text3.SetActive (false);
                                    j++;
                                    Application.LoadLevel ("tut05");
                                                }
                              }
                        }
                  else{

            if (GUI.Button (new Rect ((Screen.width ) - 150,
            Screen.height / 2 + 20, 150, 20), "continue")) {
                        if(j==0){
```

```
                                    text4.SetActive(false);
                                    text5.SetActive(true);
                                    j++;

                                    dino.SetActive(true);
                                    dino3.SetActive(true);
                                    dino2.SetActive(true);
                            }
                    Else{
                            if(j==1) {
                                    text5.SetActive(false);
                                    text6.SetActive(true);
                                    j++;
                                    dino11.elenxos=true;
                                    dino12.elenxos=true;
                                    dino13.elenxos=true;
                                    enisxiseis1.SetActive(true);
                                    enisxiseis2.SetActive(true);
                                    enisxiseis3.SetActive(true);
                                    enisxiseis4.SetActive(true);
                                }
                            else if(j==2){

                                    text6.SetActive(false);
                                    j++;
                                    Application.LoadLevel("tut05");
                                }
                            }
                        }
                    }
            }
```

Τέλος παρουσιάζουμε ένα παράδειγμα με ένα κομμάτι από τον κώδικα που χρησιμοποιήσαμε στον εχθρό για να μετακινηθεί αφού πήρε την εντολή του από τον παραπάνω κώδικα.

```
void Update () {
      if (!elenxos) {
            myTransform.position = Vector3.MoveTowards
            (transform.position, start.position, 4 * Time.deltaTime);
            if(myTransform.position==start.position)
            {

                  dinome.GetComponent<Animation> ().Stop
                  ("Allosaurus_Walk");
                  dinome.GetComponent<Animation> ().Play
                  ("Allosaurus_Attack01");
            }
      }
      if (elenxos)
      {
            dinome.GetComponent<Animation> ().Stop
            ("Allosaurus_Attack01");
            dinome.GetComponent<Animation> ().Play
            ("Allosaurus_Run");
            myTransform.position =
            Vector3.MoveTowards(transform.position, end.position, 4 *
            Time.deltaTime);
            myTransform.rotation = Quaternion.Slerp
```

```
            (myTransform.rotation, Quaternion.LookRotation
            (end.position - myTransform.position), 11*
            Time.deltaTime);// kanei rotation ston paixti
        }
}
```

4.6 Ανάλυση υλοποίησης ιστορίας στο παιχνίδι

Παρακάτω θα αναλύσουμε την λειτουργία του παιχνιδιού στο κομμάτι της ιστορίας. Η βασική αλληλεπίδραση μεταξύ των αντικειμένων σε αυτό το κομμάτι του παιχνιδιού γίνεται με τη χρήση των collider της Unity και τη χρησιμοποίηση tags. Τα tags χρησιμοποιούνται για να μπορέσουμε να αναγνωρίσουμε από ποιον collider προέκυψε η σύγκρουση. Όταν ξεκινάει ένα επίπεδο δημιουργούμε τον χαρακτήρα μας (τον έχουμε μετατρέψει σε prefab) και του βάζουμε τα στατιστικά του. Ο χρήστης ελέγχει τα input με τις συναρτήσεις της `Unity Input.GetAxis()` και `Input.GetButton()` για να δει μήπως πρέπει να μετακινήσει τον χαρακτήρα. Η δημιουργία ενός αντικειμένου στη σκηνή γίνεται μέσω της συνάρτησης `Instantiate()` (π.χ. εάν ο χαρακτήρας να πετάξει πέτρες) και για να μετακινήσουμε τον χαρακτήρα μας χρησιμοποιούμε την εντολή `AddForce()`.

Μπορούμε να αναγνωρίσουμε τις συγκρούσεις αντικειμένων με τον χαρακτήρα μας μέσω της `OnTriggerEnter()` και για να αναγνωρίσουμε τον τύπο των αντικειμένων που συγκρούστηκαν χρησιμοποιούμε τον έλεγχο `if (other.gameObject.tag == "")`. Έτσι αναγνωρίζουμε αν ο χαρακτήρας ήρθε σε επαφή με τα όπλα των εχθρών ώστε να μειώσουμε την υγεία του ή αν ήρθε σε επαφή με θεραπευτικά φίλτρα για να την αυξήσουμε.

Οι εχθροί σε αυτό το κομμάτι εκτελούν τις παρακάτω ενέργειες:

1) Ελέγχουν αν βρίσκονται κοντά στον χαρακτήρα με την συνάρτηση `Vector3.Distance`. Με την `Vector3.Distance` βλέπουν την απόσταση της θέσης που βρίσκονται σε σχέση με τη θέση που βρίσκεται ο χαρακτήρας.

2) Εάν η απόσταση είναι μικρότερη από ένα όριο που ορίζουμε ο εχθρός μετακινείται προς τον χαρακτήρα μας με την χρήση της παρακάτω συνάρτησης.

```
Transform.position=Vector3.MoveTowards
(transform.position,target.position,movespeed)
```

3α) Ένα δεν συναντήσει το τέρμα της πλατφόρμας και έχει πλησιάσει αρκετά τον χαρακτήρα τότε του επιτίθεται.

3β) Εάν συναντήσει το τέρμα της πλατφόρμας τότε επιστρέφει στην θέση που ήταν αρχικά.

Επίσης οι εχθροί ελέγχουν αν συναντάνε τις πέτρες που πετάει ο χαρακτήρας. Εάν τη συναντάνε τότε μειώνουν την υγεία τους και εάν η υγεία τους φτάσει στο μηδέν καταστρέφονται.

Ένα άλλο σημαντικό κομμάτι είναι τα checkpoint. Τα checkpoint είναι κάποια σημεία από τα οποία εμφανίζουμε τον χαρακτήρα μας σε περίπτωση που πεθάνει. Το κάθε checkpoint το ενεργοποιούμε εάν ο χρήστης περάσει ένα συγκεκριμένο σημείο της πίστας. Δίνουμε στον χαρακτήρα μας τις συντεταγμένες στις οποίες θα εμφανιστεί εάν πεθάνει.

Το τελευταίο κομμάτι είναι το αντικείμενο που όταν το πάρει ο χαρακτήρας ξεκλειδώνει την επόμενη πίστα και πηγαίνει τον χρήστη στο αρχικό μενού. Η μετακίνηση στο αρχικό μενού γίνεται με την `Application.LoadLevel()` και ο έλεγχος εάν συναντάει τον χρήστη γίνεται με την `OnTriggerEnter()` και την `if(other.gameObject.tag == "")`.

4.7 Υλοποίηση δράσης της περιπέτειας

Παρακάτω θα αναλύσουμε το πώς υλοποιήσαμε το κομμάτι δράσης της περιπέτειας. Στο κομμάτι δράσης της περιπέτειας η βασική αλληλεπίδραση μεταξύ των αντικειμένων γίνεται με την χρήση των collider της Unity σε συνδυασμό και την χρησιμοποίηση tags για να αναγνωρίσουμε από ποιον collider προέκυψε η σύγκρουση.

Σε αυτό το κομμάτι χρησιμοποιήσαμε ένα έτοιμο μοντέλο χαρακτήρα που το κατεβάσαμε δωρεάν από το Unity asset store. Σε αυτό το μοντέλο εισάγαμε διάφορα scripts και components που δημιουργήσαμε ώστε να μπορέσουμε να το κάνουμε λειτουργικό μέσα στο παιχνίδι μας.

Στο μοντέλο εισάγουμε αρχικά ένα rigid body ώστε με την `GetComponent<Rigidbody>().AddForce()` να μπορούμε να το μετακινούμε μέσα στην σκηνή. Επίσης διαβάζουμε τα inputs του χρήστη με την `Input.GetButton()` και ελέγχουμε εάν συναντάει αντικείμενα με την χρήση της `OnTriggerEnter (Collider other)`. Στην αρχή της σκηνής έχουμε εισάγει ένα empty gameobject που είναι υπεύθυνο για την δημιουργία των εχθρών. Για λόγους ευκολίας ονομάσαμε το αντικείμενο αυτό spawncontroler.

Ο spawncontroler δημιουργεί τους εχθρούς και καθορίζει (δίνοντας τους δύο συντεταγμένες) που θα κινούνται στην σκηνή. Ο spawncontroler με το που δημιουργείται στην σκηνή ψάχνει να βρει και τους άλλους spawncontroler που υπάρχουν στο παιχνίδι ώστε να αποθηκεύσει την θέση που έχει ο πιο κοντινός spawncontroler. Αυτό το κάνει ώστε να μπορεί να δώσει τις συντεταγμένες του κοντινότερου spawncontroler και τις συντεταγμένες του εαυτού του στον εχθρό που θα δημιουργήσει. Αυτό γίνεται για να δημιουργηθεί μια διαδρομή μεταξύ αυτών

των δύο αντικειμένων πάνω στην οποία θα κινείται ο εχθρός. Η εύρεση των άλλων spawncontroler γίνεται με την χρήση της παρακάτω συνάρτησης:

```
go2=GameObject.FindGameObjectsWithTag("spawncontrolerai")
```

(`go2` είναι ενας πίνακας από gameobject που έχουμε δηλώσει πιο πάνω `public GameObject[] go2`).

Τους spawncontroler τους ταξινομούμε με τον παρακάτω κώδικα:

```
foreach (GameObject loc in go2)
      {
      if (loc!=null)
            {
            if(Vector3.Distance (loc.transform.position,
            gameObject.transform.position)>megalosarithmos)
                  {
                        megalosarithmos=Vector3.Distance
                        (loc.transform.position, gameObject.
                        transform.position);
                        loc2=loc.transform;
                  }
            }
      }
```

Έτσι βάζουμε στο `loc2` την θέση που θα δώσουμε στους εχθρούς. Ο spawncontroler αφού βρίσκει τον πιο κοντινό spawncontroler δημιουργεί τους εχθρούς. Ο spawncontroler έχει πάνω του τέσσερα διαφορετικά prefabs από εχθρούς, έτσι επιλέγει τυχαία τι τύπο εχθρού θα δημιουργήσει με τη χρήση της συνάρτησης `Random.Range()`.
Για να δημιουργήσει ένα εχθρό χρησιμοποιεί τον παρακάτω κώδικα:

```
z=Instantiate(enemy3,gameObject.transform.position+position,
spawnRotation) as GameObject
```

Μετά παίρνει το script του εχθρού και του δίνει τις δύο συντεταγμένες. Στην σκηνή έχουμε δημιουργήσει ένα επιπλέον άδειο gameobject που δίνει ουσιαστικά την ζημιά που κάνουν οι εχθροί στον παίχτη (η ζημιά δεν είναι σταθερή προσαρμόζεται ανάλογα με το επίπεδο που βρισκόμαστε), ενημερώνει τα στατιστικά του παίχτη όταν δημιουργηθεί μέσω των playerprefs και αποθηκεύει τα στατιστικά του παίχτη όταν τελειώνει η πίστα. Ο κώδικας της ενημέρωσης των στατιστικών είναι ο ακόλουθος.

```
public void SetStats(){
      player=GameObject.FindGameObjectWithTag("player");
      pc2part pcClass =player.GetComponent<pc2part>();
      pcClass.Elinika = elinika;
```

```
        pcClass.Onoma=PlayerPrefs.GetString("Onoma");
        pcClass.Dinami=PlayerPrefs.GetInt("Dinami");
        pcClass.Antoxi=PlayerPrefs.GetInt("Antoxi");
        pcClass.Taxitita=PlayerPrefs.GetInt("Taxitita");
        pcClass.Ygeia=PlayerPrefs.GetInt("Ygeia");
        pcClass.Magia=PlayerPrefs.GetInt("Magia");
        pcClass.BasikiTimi=PlayerPrefs.GetInt("Epipedo");
        pcClass.Xrisos=PlayerPrefs.GetInt("xrisos");
        pcClass.PollaplasiastisEmpeirias=PlayerPrefs.GetInt("pollaplasi
        astisEmpeirias");
        pcClass.Bonuslevels=PlayerPrefs.GetInt("bonuslevel");
        pcClass.Bonusweaponlevels=PlayerPrefs.GetInt("bonusweaponlevels
        ");
        pcClass.Arithmosauxisisoplou=PlayerPrefs.GetInt("arithmosauxisi
        soplou");
        pcClass.EmpeiriaOplou=PlayerPrefs.GetInt("empeiriaOplou");
        pcClass.EpipedoOplou=PlayerPrefs.GetInt("epipedoOplou");
        pcClass.EpomenoEpipedoOplou=PlayerPrefs.GetInt("epomenoEpipedoO
        plou");
        pcClass.MeeleDinOplou=PlayerPrefs.GetInt("meeleDinOplou");
        pcClass.RangedDinOplou=PlayerPrefs.GetInt("rangedDinOplou");
        pcClass.MeeleAminaOplou=PlayerPrefs.GetInt("meeleAminaOplou");
        pcClass.RangedAminaOplou=PlayerPrefs.GetInt("rangedAminaOplou";

        pcClass.RangedDamageExthrou1 = rangeddmg1;
        pcClass.RangedDamageExthrou2 = rangeddmg2;
        pcClass.RangedDamageExthrou3 = rangeddmg3;
        pcClass.MelleDamageExthrou1= meeledmg1;
        pcClass.MelleDamageExthrou2= meeledmg2;
        pcClass.MelleDamageExthrou3= meeledmg3;
        pcClass.LevelModifier=PlayerPrefs.GetFloat("apomenousa exp gia
        epomeno epipedo");
        pcClass.UpdateStatistika ();
}
```

Καθώς επίσης με παρόμοιο κώδικα αποθηκεύει τα στατιστικά του παίχτη.
Οι εχθροί που δημιουργούνται στην πίστα έχουν απλές λειτουργίες. Στην αρχή όταν δημιουργούνται πηγαίνουν μεταξύ δυο σημείων. Η μετακίνηση τους γίνεται με την `Vector3.MoveTowards(transform.position,target.position,movespeed)`. Εάν ο εχθρός συναντήσει τον χρήστη (είναι σε μια κοντινή απόσταση με τον χαρακτήρα του χρήστη) τότε κινείται προς το μέρος του και αν πλησιάσει αρκετά του επιτίθεται. Ο εχθρός ελέγχει τις αποστάσεις μεταξύ αυτού και του χρήστη με τη βοήθεια της συνάρτησης `Vector3.Distance(target.position, myTransform.position)`. Όταν η υγεία του εχθρού γίνεται μηδέν τότε εμφανίζει χρυσά νομίσματα στο έδαφος και δίνει εμπειρία στον χαρακτήρα μας. Όταν ο βασικός χαρακτήρας εξουδετερώσει πάνω από πέντε εχθρούς εμφανίζεται το τέλος της πίστας.

4.8 Περιπέτεια turn-base κομμάτι

Το επίπεδο turn-base έχει διαφορετική λειτουργικότητα από το κομμάτι δράσης. Αρχικά θα το διαχωρίσουμε σε δύο επίπεδα. Το πρώτο επίπεδο που είναι το

επίπεδο συνάντησης με τον εχθρό και το δεύτερο επίπεδο είναι το επίπεδο της μάχης με τον εχθρό.

4.8.1 Επίπεδο συνάντησης με τον εχθρό

Σε αυτό το επίπεδο ο χαρακτήρας μας προσπαθεί να συναντήσει εχθρούς, ώστε να μεταφερθεί στο επίπεδο μάχης. Εδώ όπως και στο κομμάτι δράσης έχουμε δημιουργήσει κάποια άδεια gameobject. Έπειτα τους εισάγουμε ένα script ώστε να μπορούν να δημιουργούν τους εχθρούς και του δίνουν τις δύο θέσεις που θα καθορίζουν την διαδρομή που θα κινούνται. Οι εχθροί, όπως και στο κομμάτι δράσης, ελέγχουν την απόσταση τους από τον χαρακτήρα μας με την `Vector3.Distance()` και πηγαίνουν προς το μέρος του με την `Vector3.MoveTowards()`. Εάν συγκρουστούν με τον χαρακτήρα μας μεταφέρονται στο επίπεδο μάχης. Πριν μεταφερθούν στο επίπεδο μάχης αποθηκεύουν σε τρία Playerprefs τις συντεταγμένες που έχει ο χρήστης πριν την σύγκρουση ώστε εάν νικήσει να επιστρέψει στο σημείο που έγινε η σύγκρουση. Ο χαρακτήρας μας σε αυτό το κομμάτι απλά κινείται μέσα στην πίστα. Εδώ επίσης εμφανίζεται μια πύλη (όπως στο κομμάτι δράσης) όταν έχει σκοτώσει τουλάχιστον πέντε εχθρούς για να πάει στο επόμενο επίπεδο αφού επιλέξει όπλο.

4.8.2 Επίπεδο μάχης με τον εχθρό

Στο επίπεδο μάχης αρχικά δημιουργούμε ένα άδειο gameobject όπου του εισάγουμε ένα script για να κάνει τις παρακάτω λειτουργείες:

- Να μπορεί να δημιουργήσει τους εχθρούς,
- Να ενεργοποιεί ανάλογα με το ποιανού είναι η σειρά το κατάλληλο αντικείμενο (τον χρήστη ή κάποιον εχθρό)
- Να επιστρέφει τον χαρακτήρα στο επίπεδο συνάντησης με τον εχθρό αν νικήσει την μάχη.

Το αντικείμενο αυτό για ευκολία το ονομάσαμε controlermaxis. Ο controlermaxis αρχικά επιλέγει τον αριθμό των εχθρών που θα εμφανιστούν. Ο αριθμός των εχθρών μπορεί να είναι μεταξύ του αριθμού ένα και τρία. Αυτό γίνεται με την χρήση της συνάρτησης `Random.Range(1F, 4F)`. Ο controlermaxis ενεργοποιεί τον χρήστη και όταν τελειώσει η κίνηση του χρήστη, ενεργοποιεί τους εχθρούς. Αυτό γίνεται με μια απλή λογική μηχανής καταστάσεων. Πρώτα ενεργοποιείται ο χρήστης. Όταν τελειώσει εάν οι εχθροί δεν είναι νεκροί τότε ενεργοποιούνται οι εχθροί. Διαφορετικά γυρίζει τον χρήστη στο προηγούμενο επίπεδο. Όταν ενεργοποιούνται οι εχθροί αν ο χρήστης μετά την επίθεση είναι νεκρός τότε πάμε στο κεντρικό μενού ενώ εάν δεν είναι ενεργοποιεί τον χρήστη. Η παραπάνω υλοποίηση γίνεται με απλό έλεγχο σε boolean μεταβλητές. Όταν ο χαρακτήρας ενεργοποιείται, εμφανίζουμε με την χρήση της `OnGUI()` ένα μενού.

Το μενού εμφανίζει στον χρήστη τις επιλογές εάν θα κάνει επίθεση από κοντά ή μακριά και σε ποιον εχθρό. Έχουμε βάλει επιπλέον μία λειτουργία εάν ο χαρακτήρας μας έχει μικρότερη από το ένα τρίτο της υγείας του να κάνει διπλάσια

ζημιά και να ενεργοποιούμε με ένα απλό enable ένα particle system που θα τον εμφανίζει σαν να βγάζει φωτιά.

Εικόνα 4.3 Ο παίχτης ενώ βγάζει φωτιά

Όταν τελειώσει ο χαρακτήρας μας πάλι με την χρήση της `OnGUI()` εμφανίζουμε ένα μενού και επιλέγουμε την επίθεση του εχθρού. Ο εχθρός επιτίθεται στον χαρακτήρα μας και μετά ενεργοποιεί τον επόμενο εχθρό, εάν δεν είναι νεκρός, ή τον χαρακτήρα μας εάν ήταν ο τελευταίος ζωντανός εχθρός. Το ενδιαφέρον κομμάτι που υλοποιήσαμε είναι ότι όλοι οι εχθροί έχουν ουσιαστικά το ίδιο script νοημοσύνης παρόλο που έχουν διαφορετικά animations.

4.9 Παιχνίδι με δυνατότητα συμμετοχής πολλών παιχτών

4.9.1 Λειτουργία

Το κομμάτι του παιχνιδιού με δυνατότητα συμμετοχής πολλών παιχτών το υλοποιήσαμε με την χρήση των εντολών που μας δίνει ο Masterserver της Unity. Στον Masterserver ο χρήστης ο οποίος δηλώνει το παιχνίδι μας (που γίνεται και ο σέρβερ) έχει τη δυνατότητα να επικοινωνήσει με όλους τους χρήστες που θα συνδεθούν σε αυτό το παιχνίδι. Έτσι ο κάθε χρήστης για να μπορέσει να επικοινωνήσει με τους άλλους χρήστες επικοινωνεί με τον σέρβερ και αυτός επικοινωνεί με τους χρήστες. Αρχικά με τη χρήση της `OnGUI()` δίνουμε το μενού που ο χρήστης μπορεί να δημιουργήσει ή να συνδεθεί σε ένα παιχνίδι. Εάν ο χρήστης επιλέξει να δημιουργήσει παιχνίδι του δίνεται η δυνατότητα να επιλέξει να δημιουργήσει ένα παιχνίδι που μπορούν να συμμετάσχουν δύο μέχρι οχτώ παίχτες (επειδή έχουμε υλοποιήσει την λειτουργικότητα του παιχνιδιού να έχει μέγιστο αριθμό παιχτών τους οχτώ). Για να βάλουμε αυτό το όριο χρησιμοποιούμε την εντολή `Players=Mathf.Clamp (Players,2,8)` και για να το ενεργοποιήσουμε στον σέρβερ χρησιμοποιούμε τον παρακάτω κώδικα.

```
public void StartServer(string ServerName, int MaxPlayers)
{
	Network.InitializeSecurity();
	Network.InitializeServer(MaxPlayers, 25565, true);
	MasterServer.RegisterHost("lefterismal1",ServerName,"");
}
```

Εάν ο χρήστης επιλέξει να βρει κάποιον σέρβερ για να παίξει τότε στην αρχή ζητάει από τον masterserver να δώσει όλα τα παιχνίδια που υπάρχουν από αυτόν του τύπο παιχνιδιού. Αυτό γίνεται με την χρήση της εντολής `MasterServer.RequestHostList("lefterismal1")`. Έπειτα μπορούμε να συνδεθούμε με την χρήση της εντολής `Network.Connect()` στο παιχνίδι που επιθυμούμε. Ένα ο παίχτης συνδεθεί στο παιχνίδι τότε καλείται αυτόματα η συνάρτηση `OnConnectedToServer()`. Η επικοινωνία μέσα στο παιχνίδι γίνεται μέσω RPC(remote procedure calls).

4.9.2 Τρόπος χρησιμοποίησης αποθηκευμένων στατιστικών

Στο κομμάτι του παιχνιδιού με δυνατότητα συμμετοχής πολλών παιχτών οι αλληλεπιδράσεις μεταξύ των χρηστών γίνονται μέσω colliders. Για να γίνει η σύγκρουση σωστά πρέπει πρώτα να ενημερώσουμε τον κάθε χρήστη για το χαρακτηριστικό "επίθεση" όλων των χρηστών. Αυτό προϋποθέτει ότι πάνω στο instance του παιχνιδιού που έχουμε δημιουργήσει ο κάθε χρήστης πρέπει να έχει τα δικά του χαρακτηριστικά αποθηκευμένα μέσα σε ένα κοινό script. Αυτό βρίσκεται μέσα σε ένα αντικείμενο που μπορούν να έχουν πρόσβαση όλοι οι χρήστες. Αναλυτικότερα για αυτό το λόγο δημιουργούμε ένα empty gameobject τον networkmanager που του εισάγουμε ένα script για να μπορούμε πάνω του να αποθηκεύουμε τους χρήστες όταν μπαίνουν στο παιχνίδι καθώς και τα χαρακτηριστικά τους. Αυτό γίνεται με την χρήση της `void OnPlayerConnected(NetworkPlayer id){}`. Η παραπάνω συνάρτηση καλείται κάθε φορά που ένας χρήστης συνδέεται στο παιχνίδι μας. Πάνω σε αυτή δημιουργήσαμε μια RPC την `Client_PlayerJoined` και την καλούμε με την παρακάτω εντολή `nView.RPC ("Client_PlayerJoined", id, pl.PlayerName, pl.OnlinePlayer, pl.kodikos)`. Ο κώδικας της RPC είναι ο παρακάτω:

```
[RPC]
public void Client_PlayerJoined(string Username, NetworkPlayer
id,int kodiko)
{
	Player temp = new Player();
	temp.PlayerName = Username;
	temp.OnlinePlayer = id;
	if (kodiko == 0)
	{
		eidikoskodikos = Random.Range (0, 900000000);
		kodiko=eidikoskodikos;
```

```
            }
            temp.kodikos = kodiko;
            temp3 = temp;
            PlayerList.Add(temp);

            if(Network.player == id)
            {
                  nView.RPC("dimpaixti2",RPCMode.AllBuffered);
                  GameObject LastPlayer=
                  Network.Instantiate(SpawnPlayer,Vector3.zero,Quater
                  nion.identity,0) as GameObject;

                  r=MyPlayer;
            }
      }
```

Έτσι ενημερώνονται απευθείας τα χαρακτηριστικά που έχει στο struct του ο παίχτης ο οποίος συνδέθηκε. Παρακάτω παρουσιάζουμε το struct player.

```
[System.Serializable]
public class Player{
      public string PlayerName;
      public NetworkPlayer OnlinePlayer;
      public float Health =200;
      public UserPlayer Manager;
      public bool IsAlive;
      public int idikoskodikos;
      public int kodikos;
      public int zwes=4;
      public int xrysos=0;
      public bool xtipithike =false;
      public string tagg;
      public int def;
      public int attack=0;
}
```

Μετά θα πρέπει να ενημερώσουμε τα χαρακτηριστικά του κάθε παίχτη στο networkmanager του χρήστη που έχει γίνει server μιας και είναι ο μόνος που μπορεί να είναι σε επαφή με όλους τους χρήστες. Αυτό γίνεται με τις παρακάτω συναρτήσεις.

```
public void entag(){

      if (Network.isServer)
      {
            NetworkManager
            kooo=gameplan.GetComponent<NetworkManager>();
            float o=0;
```

Παραπάνω κοιτάξουμε εάν ο χρήστης μας είναι ο server (ουσιαστικά σε όλες τις αλληλεπιδράσεις επιλέγουμε να γίνουν μόνο στον server αλλιώς ο αριθμός των φορών που θα επαναλαμβανόταν ο παραπάνω κώδικας θα ήταν όσο ο αριθμός των συσκευών που είναι συνδεδεμένες στο δίκτυο, δηλαδή θα καλούσε τον κώδικα κάθε συσκευή) εάν είναι παίρνουμε το component NetworkManager (είναι το όνομα του script) για να μπορέσουμε να το επεξεργαστούμε.

```
            foreach (Player tt in kooo.PlayerList)
```

```
        {
            if(tt.OnlinePlayer==kooo.MyPlayer.OnlinePlayer)
            {
                int k=(PlayerPrefs.GetInt ("Dinami") +
                PlayerPrefs.GetInt ("rangedDinOplou") +
                PlayerPrefs.GetInt ("meeleDinOplou") +
                PlayerPrefs.GetInt ("Taxitita")) / 4 ;
                int uj= (PlayerPrefs.GetInt ("Antoxi") +
                PlayerPrefs.GetInt ("Magia") +
                PlayerPrefs.GetInt ("meeleAminaOplou") +
                PlayerPrefs.GetInt ("rangedAminaOplou")) / 4;
                tt.def=uj;

                tt.attack=k;
                GetComponent<NetworkView>().RPC
                ("enimepithall", RPCMode.All,
                tt.OnlinePlayer, k,uj);
            }
```

Παραπάνω στον κώδικα παίρνουμε κάθε ένα χρήστη που έχουμε μέσα στο networkmanager και του ενημερώνουμε την επίθεση που έχει. Με τον κώδικα παρακάτω βάζουμε σε καθένα χρήστη ένα tag ώστε με το να περνάει το tag του στα πυρά που ρίχνει να μπορούμε να βρίσκουμε από ποιον προήλθε το βλήμα για να υπολογίσουμε πόσο ζημιά θα πάθει ο χαρακτήρα που πέτυχε.

```
            if(o==0)
            {
                tt.Manager.MyPlayer.tagg="mpl1";

                GetComponent<NetworkView> ().RPC
                ("enimemrositat", RPCMode.All,
                tt.Manager.MyPlayer.OnlinePlayer,tt.Manager.M
                yPlayer.tagg);
            }
```

Με παρόμοιο τρόπο το κάνουμε και για τους οχτώ χαρακτήρες.

```
            if(o==7)
            {
                tt.Manager.MyPlayer.tagg="mpl8";
                GetComponent<NetworkView> ().RPC
                ("enimemrositat", RPCMode.All,
                tt.Manager.MyPlayer.OnlinePlayer,tt.Manager.M
                yPlayer.tagg);
            }
            o++;

            }
        }
    }
```

Καθώς και την RPC:

```
public void enimemrositat(NetworkPlayer id,string tag){

    NetworkManager kooo = gameplan.GetComponent<NetworkManager> ();
    foreach (Player pl in kooo.PlayerList)
    {
```

```
            if (pl.OnlinePlayer == id)
            {
                  pl.tagg=tag;
            }
            if(pl.OnlinePlayer==kooo.MyPlayer.OnlinePlayer)
            {
                  int k=(PlayerPrefs.GetInt ("Dinami") +
                  PlayerPrefs.GetInt ("rangedDinOplou") +
                  PlayerPrefs.GetInt ("meeleDinOplou") +
                  PlayerPrefs.GetInt ("Taxitita")) / 4 ;
                  int uj= (PlayerPrefs.GetInt ("Antoxi") +
                  PlayerPrefs.GetInt ("Magia") + PlayerPrefs.GetInt
                  ("meeleAminaOplou") + PlayerPrefs.GetInt
                  ("rangedAminaOplou")) / 4;
                  GetComponent<NetworkView> ().RPC ("enimepiths",
                  RPCMode.Server, pl.OnlinePlayer, k,uj);
            }
      }
}
```

Με τις παραπάνω μεθόδους έχουμε ένα αντικείμενο που περιέχει μία λίστα με όλους τους χρήστες. Επίσης περιέχει πληροφορίες για την επίθεση και άμυνα που έχει ο κάθε χρήστης.

4.9.3 Ανάλυση της δομής του χαρακτήρα

Παρακάτω θα αναλύσουμε την δομή και τις λειτουργίες του κάθε κομματιού που αποτελεί τον χαρακτήρα μας για το κομμάτι του παιχνιδιού με δυνατότητα συμμετοχής πολλών παιχτών. Η βασική δομή του χαρακτήρα αποτελείται από τρία βασικά κομμάτια: το κυρίως σώμα, το κομμάτι χρήσης πρώτου προσώπου(first person controller) και το κομμάτι χρήσης τρίτου προσώπου (third person controller). Το κυρίως σώμα ελέγχει τα άλλα δύο κομμάτια. Ανάλογα εάν ο χρήστης χειρίζεται τον συγκεκριμένο χαρακτήρα ή όχι απενεργοποιεί το κατάλληλο κομμάτι. Στο κύριο σώμα γίνονται και οι βασικές λειτουργίες (μείωση υγείας παίχτη κλπ).

4.9.4 Ανάλυση first person controller

Το first person controller κομμάτι είναι ενεργοποιημένο όταν ο χρήστης είναι εκείνος που χειρίζεται αυτόν τον χαρακτήρα. Με το FPC (first person controller) χειριζόμαστε τον χαρακτήρα με βάση τα κουμπιά που πατάει ο χρήστης. Αναλυτικότερα για να μετακινούμε τον χαρακτήρα εισάγαμε το first person controller που μας δίνεται από την Unity. Το first person controller της Unity

- Περιέχει ένα ενσωματωμένο capsule collider που ελέγχει τις συγκρούσεις
- Διάφορα scripts τα οποία καθορίζουν πως θα κινείται ο χαρακτήρας μας
- Μια κάμερα που τον ακολουθεί.

Στο FPC προσθέσαμε ένα script το οποίο ουσιαστικά ενημερώνει τι κάνει ο χαρακτήρας μας, ώστε να χρησιμοποιεί το κατάλληλο animation στον third person controller για να βλέπουν οι άλλοι χρήστες τη σωστή κίνηση.

Ο κώδικας αυτού του component είναι ο παρακάτω:

```
using UnityEngine;
using System.Collections;

	public class netwanimst2 : MonoBehaviour {
	private bool z=true;
	public float V;
	public float H;
	public netanstates2 States;
```

Παραπάνω δηλώσαμε τις μεταβλητές που θα χρησιμοποιήσουμε ποιο κάτω καθώς και δηλώσαμε ως public την μεταβλητή states για να μπορέσουμε να την πάρουμε με ευκολία από το workspace του Unity.

```
	void Update () {
		V=Input.GetAxis("Vertical");
		H=Input.GetAxis("Horizontal");
		if (z != true)
		{
		}
		else
		{
			if (Input.GetButtonDown ("Fire1"))
			{
			z=false;
			States.SyncAnimation ("punch_20", 1);
			StartCoroutine (delay ());
			}
```

Παραπάνω ελέγχουμε σε κάθε ένα update εάν ο χρήστης έχει πατήσει το αριστερό click.Αν το έχει πατήσει τότε παίζουμε το animation της επίθεσης (το SyncAnimation είναι απλά μια συνάρτηση που έχουμε στο component states. Η συνάρτηση αυτή παίζει το clip που έχουμε βάλει ως string και του δίνει την ταχύτητα που έχουμε βάλει ως int). Επίσης χρησιμοποιούμε την coroutine delay επειδή δεν μπορούμε να βάλουμε μέσα στην update wait και θέλουμε να περάσει κάποιος χρόνος ώστε να σταματήσει αυτό το clip.

```
		else if (V < 0)
		{

			States.SyncAnimation ("walk_00", 1);

		}
		else if (V > 0)
		{

			States.SyncAnimation ("walk_00", -1);


		}


		else if (H < 0)
		{

			States.SyncAnimation ("sidestep_11_p", 1);
```

```
            }
            else if (H > 0)
            {

                  States.SyncAnimation ("sidestep_11_p", -1);


            }
```

Παραπάνω καθορίζουμε τη φορά θα έχει το σώμα βάζοντας ένα ,μείων ένα και επίσης εάν πάει ο χρήστης μπροστά παίζουμε το clip που ο χρήστης περπατάει ενώ εάν πάει ο χρήστης πλάγια το clip που κάνει πλάγια βήματα.

```
            else
            {
                  States.SyncAnimation ("idle_01", 1);
            }
      }
      IEnumerator delay() {
      yield return new WaitForSeconds (1f);
            z = true;
      }

}
```

Στο FPC προσθέσαμε ένα ακόμα script το οποίο ενημερώνει εάν ο χρήστης έχει χτυπηθεί. Αυτό το script είναι το ίδιο και στο third person controller. Το παραπάνω script ενημερώνει εάν ο χρήστης έχει χτυπηθεί από τα πυρά κάποιου παίχτη. Εάν έχει χτυπηθεί τότε βλέπει την επίθεση του εχθρού και την άμυνα του παίχτη και στέλνει την διαφορά ώστε να αφαιρεθεί από την υγεία του. Στην κάμερα του FPC εισάγαμε ένα canvas για να δημιουργήσουμε τη γραφική διεπαφή που βλέπει ο χρήστης (την ζωή κλπ). Εισαγάγαμε επίσης ένα script που δημιουργεί τα πυρά όταν πατάει ο χρήστης το ποντίκι. Αυτό έγινε με τη χρήση της `Network.Instantiate()`. Μετά δώσαμε στο αντικείμενο που δημιουργήσαμε το tag που έχει ο παίχτης (για να μπορούμε να αναγνωρίζουμε ποιος παίχτης μας χτύπησε), αυτό έγινε με την παρακάτω συνάρτηση:

```
GetComponent<NetworkView>().RPC("tagd", RPCMode.All,
tt.MyPlayer.tagg)
```

Για να μπορέσει να λειτουργήσει η παραπάνω εντολή πρέπει να βάλουμε την παρακάτω RPC εντολή σε ένα script και να το εισάγουμε στο αντικείμενο που δημιουργούμε.

```
[RPC]
      public void tagd(string y)
      {
            this.gameObject.tag = y;
```

```
            UserPlayer jj;
            GameObject jjj;
            jjj = this.gameObject;

      }
```

Το παραπάνω το κάναμε ώστε να βάλει το tag (για να διαχωρίζει από ποιον παίχτη δημιουργήθηκε κάθε βλήμα) σε όλα τα instances του παιχνιδιού.

4.9.5 Ανάλυση third person controller

Το third person controller είναι ουσιαστικά το μοντέλο του παίχτη και απλά του εισάγαμε ένα script για να ελέγχουμε τις συγκρούσεις (το ίδιο με του FPC). Το third person controller είναι ενεργοποιημένο στου χαρακτήρες που δεν χειριζόμαστε ώστε να βλέπουμε τα μοντέλα και τα animation τους στο instance του παιχνιδιού μας. Ο κώδικας με τις συγκρούσεις είναι ο παρακάτω:

```
using UnityEngine;
using System.Collections;

public class mulcol : MonoBehaviour {

      public GameObject tager;
      public GameObject networklist;
      public tagger tg;
      public UserPlayer onllineid;

      int y=0;
      bool firstcol=false;

      int def33=0;
      float jj;
      private bool hit=true;
      float j;
      public string tagggg;

      void Start ()
      {
            hit=true;
            tager=GameObject.FindGameObjectWithTag("tager");
            tg = tager.GetComponent<tagger> ();
      }
```

Παραπάνω δηλώνουμε τις μεταβλητές που θα χρησιμοποιήσουμε. Με την start() θέτουμε αρχικές τις τιμές σε μερικές μεταβλητές.

```
      void OnTriggerEnter(Collider other)
      {


            if(!firstcol)
            {
            firstcol=true;
      networklist=GameObject.FindGameObjectWithTag("netmanager");
            NetworkManager kooo =
```

```
networklist.GetComponent<NetworkManager> ();


            tagggg=kooo.MyPlayer.tagg;
            def33= tager.GetComponent<tagger> ().amina (tagggg);
            }
      }
```

Παραπάνω όταν γίνει η σύγκρουση τότε παίρνουμε την τιμή της άμυνας που υπάρχει στον tagger του χαρακτήρα μας την οποία θα την χρησιμοποιήσουμε παρακάτω ώστε βρούμε την διαφορά από την επίθεση του χαρακτήρα που έριξε τα πυρά.

```
      if (other.gameObject.tag == "telospistas")
      {


            Application.LoadLevel("desscene");
      }
```

Παραπάνω ελέγχουμε ένα συνάντησε το τέλος της πίστας οπότε τελειώνει το παιχνίδι και γυρίζουμε στο αρχικό μενού.

```
      if (Network.isServer)
            {
            if (other.gameObject.tag == "multgold")
            {
                  onllineid.goldmm (100,
                  onllineid.MyPlayer.OnlinePlayer);
            }
            if (other.gameObject.tag == "exthrika opla")
            {
                  onllineid.dmm (2,
                  onllineid.MyPlayer.OnlinePlayer,1);
            }
```

Παραπάνω ελέγχουμε ένα συνάντησε χρυσό ή επίθεση από δεινοσαύρους αν συνάντησε τότε ενημερώνουμε το κύριος σώμα. Παρακάτω ελέγχουμε σε περίπτωση που του επιτεθεί κάποιος χρήστης τότε παίρνουμε το damage του από τον tagger αφαιρούμε από αυτό την άμυνα του χαρακτήρα που χτυπήθηκε και στέλνουμε στο κύριος σώμα την ζημιά που έπαθε.

```
      if (other.gameObject.tag == "mpl1" && hit)
            {
                  y = tager.GetComponent<tagger> ().dramage ("mpl1");

                  hit = true;

                  StartCoroutine (Handlet ());
                  y = y - def33;

                  if (y <= 0)
                  {
                        y = 1;
                  }
                  onllineid.dmm (y,
                  onllineid.MyPlayer.OnlinePlayer,0);
```

```
            }
```

Με παρόμοιο τρόπο το κάνουμε και για την πιθανότητα να χτυπηθούμε από ένα βλήμα εκτόξευσε ένας από τους οχτώ χαρακτήρες που μπορεί να είναι συνδεδεμένοι στο δίκτυο.

```
            if (other.gameObject.tag == "mpl8" && hit)
            {
                  hit = true;

                  StartCoroutine (Handlet ());
                  y = tager.GetComponent<tagger> ().dramage ("mpl8");

                  j = (float)y;

                  y = y - def33;

                  if (y <= 0)
                  {
                        y = 1;
                  }
                  onllineid.dmm (y,
                  onllineid.MyPlayer.OnlinePlayer,0);
            }
      }
      else
      {
            if (other.gameObject.tag == "exthrika opla")
            {
            }
      }

      void Update ()
      {
      }
      private IEnumerator Handlet()
      {

            hit = true;
            yield return new WaitForSeconds( 0f );
      }
}
```

4.9.6 Ανάλυση του κυρίως σώματος

Το κυρίως σώμα ουσιαστικά είναι το συνδετικό κομμάτι μεταξύ των εντολών που δίνει ο χρήστης και των κινήσεων και των animation που βλέπουν οι άλλοι χρήστες. Στο κυρίως σώμα αρχικά βάζουμε δύο network view. Ένα πάνω στον παίχτη και ένα πάνω στο script του first player controller. Το κύριως σώμα αρχικά ελέγχει εάν έχουμε τον χρήστη που ελέγχει τον χαρακτήρα με την εντολή `GetComponent<NetworkView>().isMine` και ενεργοποιεί ή απενεργοποιεί το καθένα κομμάτι ανάλογα αν η εντολή πάνω είναι θετική ή αρνητική. Επίσης το κυρίως σώμα είναι υπεύθυνο να μειώνει την υγεία του εάν του έρθει η εντολή ότι χτυπήθηκε από ένα από τα κομμάτια του (first person controller ή third person controller). Αυτό γίνετε με την παρακάτω συνάρτηση.

```
public void dmm(int damage,NetworkPlayer id,int k){

      if (Network.isServer)
      {
            NetworkManager
kooo=gameplan.GetComponent<NetworkManager>();
            float z=0;
            foreach (Player pl in kooo.PlayerList)
            {

                  if (pl.OnlinePlayer == id)
                  {

                        pl.Health -= damage;

                        z=pl.Health;
                        if(z<=0)
                        {

                        pl.zwes--;
                        xxl=pl.zwes;

                        }
```

Παραπάνω ελέγχουμε εάν ο χρήστης ποιος χρήστης τραυματίστηκε και του αφαιρούμε την ζημιά από την υγεία του .Αν η υγεία του χρήστη γίνει μηδέν του αφαιρούμε μια ζωή.

```
                  if(pl.zwes<0&&pl.Health<=0)
                        {


            level=GameObject.FindGameObjectWithTag
                              ("levelmod");
                              if(level!=null)
                              {
                                    levelmod kks=
                                    level.GetComponent<levelmod>();
                                    kks.anan();
                              }



                              if(kooo.MyPlayer.OnlinePlayer==id)
                              {

                                    kooo.enapominantespaixtes--;

                              gameplan.GetComponent<NetworkView>()
                              .RPC ("meiwsi2",
                              RPCMode.All,kooo.enapominantespaixtes);
                              }
                              else
                              {


                              gameplan.GetComponent<NetworkView>()
                              .RPC ("meiwsi2",
```

```
RPCMode.All,kooo.enapominantespaixtes-
1);

}
```

Παραπάνω κάνουμε την μείωση στης ζωές του χρήστη και στέλνουμε τα νέα δεδομένα σε όλους τους χρήστες.

```
if(kooo.enapominantespaixtes==1)
{
level=
GameObject.FindGameObjectWithTag("level
mod");

if(level!=null)
      {

            StartCoroutine (Handlet ()
      );


                        `
Application.LoadLevel("desscene");

}

if(kooo.enapominantespaixtes<=0)
{

      StartCoroutine (Handlet ()
      );


     Application.LoadLevel("desscene");

}

}
```

Παραπάνω ελέγχουμε ένα εάν υπάρχει levelmod. Αν δεν υπάρχει είμαστε στο coop κομμάτι οπότε το παιχνίδι δεν σταματάει παρά μόνο πεθάνουν όλοι οι χρήστες αν υπάρχει και είναι μόνο ένας ζωντανός το παιχνίδι σταματάει και αυτός ο χρήστης νικάει.

```
GetComponent<NetworkView> ().RPC ("enimstat",
RPCMode.All, id,z);
if (z <= 0)
{
      FirstPerson.gameObject.transform.position=new
      Vector3(0,2000,0);
      ThirdPerson.gameObject.transform.position=new
      Vector3(0,2000,0);

      MyPlayer.xtipithike=true;
      FirstPerson.gameObject.SetActive(false);
      MyPlayer.IsAlive=false;
      GetComponent<NetworkView> ().RPC ("pethane",
      RPCMode.All, id,xxl);
```

```
        }
```

Παραπάνω εάν πέθανε ο χρήστης στέλνουμε την θέση του πολύ μακριά ώστε να μην επηρεάσει την αναζήτηση που κάνουν οι δεινόσαυροι για του χρήστες και απενεργοποιούμε και τα δύο κομμάτια του(first person controller και third person controler) μέχρι να πατήσει να αναστηθεί.

Με την RPC εντολή `GetComponent<NetworkView>().RPC ("enimstat", RPCMode.All, id,z)` το κυρίως σώμα ενημερώνει όλους τους παίχτες που συμμετέχουν στο παιχνίδι την νέα υγεία του μετά την αφαίρεση της ζημιάς. Ο κώδικας είναι ο παρακάτω:

```
[RPC]
      public void enimstat(NetworkPlayer id,float health)
      {
            NetworkManager
kooo=gameplan.GetComponent<NetworkManager>();

            foreach (Player pl in kooo.PlayerList)
            {

                  if (pl.OnlinePlayer == id)
                  {
                        pl.Health =health;

                  }
            }

      }
```

Επίσης με παρόμοιο τρόπο ενημερώνει για τα χρυσά νομίσματα που μαζεύει ο κάθε παίχτης καθώς για την αφαίρεση ζωών μετά το θάνατο του παίκτη.
Το κυρίως σώμα είναι επίσης υπεύθυνο για να «γράφει» τις θέσεις του χαρακτήρα πάνω στη σκηνή του παιχνιδιού. Αυτό γίνεται με την παρακάτω εντολή:

```
void OnSerializeNetworkView(BitStream stream,NetworkMessageInfo info)
{

      if (stream.isWriting) {

            CurPos = FirstPerson.position;
            CurRot = FirstPerson.rotation;

            stream.Serialize (ref CurPos);
            stream.Serialize (ref CurRot);

            char Ani =(char)GetComponent<netanstates2>().CurrentAnim;
            stream.Serialize(ref Ani);
      }
      else
      {
            stream.Serialize (ref CurPos);
            stream.Serialize (ref CurRot);

            char Ani =(char)0;
            stream.Serialize(ref Ani);
```

```
        GetComponent<netanstates2>().CurrentAnim=(Animations2)Ani;

        }
}
```

Έτσι μετακινεί το μοντέλο που βλέπουν όλοι στο παιχνίδι (το third person controller) στο σημείο που είναι ο χαρακτήρας του χρήστη (το first person controller) και δείχνει τα σωστά animation του μοντέλου.

4.9.7 Ανάλυση κώδικα εχθρού

Ο μόνος εχθρός που έχουμε στο κομμάτι του παιχνιδιού με δυνατότητα συμμετοχής πολλών παιχτών είναι ο δεινόσαυρος. Το μοντέλο και τα animation είναι ακριβώς τα ίδια με αυτά του εχθρού στο single player. Για να μπορεί ο εχθρός να εμφανίζεται σε όλα τα instances του παιχνιδιού του βάζουμε ένα Networkview. Στην αρχή ο εχθρός βρίσκει τι επίπεδο δυσκολίας έχει επιλέξει ο χρήστης για να προσαρμόσει την υγεία του. Στη συνέχεια έχουμε βάλει διάφορα άδεια gameobjects στην πίστα. Οι εχθροί βρίσκουν τα δύο πιο κοντινά τους και κινούνται μεταξύ αυτών των δυο σημείων. Ο εχθρός μετά ψάχνει να βρει πόσοι παίχτες υπάρχουν στο παιχνίδι και τους βάζει σε ένα πίνακα οχτώ αντικειμένων. Γεμίζει τις υπόλοιπες θέσεις με ένα άδειο αντικείμενο με πολύ μακρινή απόσταση (ώστε να μην επηρεάζει το παιχνίδι). Ο εχθρός κάνει συνεχείς ελέγχους εάν υπάρχει κάποιος παίχτης σε κοντινή απόσταση. Εάν υπάρχει, κινείται προς το μέρος του και του επιτίθεται. Εάν ο εχθρός χτυπηθεί από πυρά ενός χρήστη τότε χάνει υγεία και εάν η υγεία του φτάσει στο μηδέν τότε εμφανίζει χρυσά νομίσματα στην σκηνή. Ο κώδικας του εχθρού είναι ο παρακάτω:

```
using UnityEngine;
using System.Collections;

public class dinoscriptmulttry2 : MonoBehaviour
{
      public GameObject[] mploko;
      public GameObject tager;
      public tagger tg;
      int y=0;
      private int epipedomultpaixnidiou8293;
      public GameObject dinodeath;
      public GameObject dUMMY;
      private GameObject wcont;
      public  Transform dok;
      public bool z1 = true;
      public int maxdistance3;
      private GameObject d;//to instance pou bazeis gia na tou dwseis
to tag
      private string tagname ;
      private int damages;

      private GameObject scenemaster;// gia na dilwsoume tous extrous
na mporesei na energopoihthei i exodos
      private GameObject r33;// gia na dilwsoume tous extrous na
mporesei na energopoihthei i exodos
      private scencontr2 de;
```

```
	public GameObject xrysos;//inst gia to gold pou petan oi extroi
otan tous skwtwseis

	public Transform target;//o paixtis
	public Transform[] playerth;//o paixtis
	public Transform loc1;
	public Transform loc2;
	public bool locconr=true;//true loc 2 false loc 1;
	public Transform destination;
	public Vector3 ff= new Vector3(0f,0.5f,0f);
	private bool kinigaw=false;
	public float trrr=1000000f;
	private int playerdmg;//to damage pou kanoun ta ranged opla tou
paixti
	private pc2part enim;//ta komatia tou  paixti
	private dino_ai_anime kat;//oi katastaseis animation pou exoume
gia ton dinosauro
	private bool z=true;//edw einai gia na stamatisei o dinosauto
tin epithesi kai na sinexisei ta upoloipa anime states tou
	private bool block =false;//enable gia to an einai arketa konta
ston paixti na tou epitethei
	public int movespeed;//i taxitita  tou dino
	public int rotationspeed;//i taxitita pou girizei
	public int maxdistance;//megisti apostasi gia na tou kanei
epithesi
	public int maxdistance2;//megisti apostasi gia na ton kinigisei
	public GameObject shot;//i epithesi tou paixti mas emeis mias
kai einai meele tha to kanoume inst panw ston paixti me to sosto tag
	private GameObject[] go;//o paixtis mas
	private GameObject[] go2;//o paixtis mas
	private float nextfire;//blepe apo katw
	public float firerate;//poso prosthetei sto next  fire na
xanaepitethei
	private Transform myTransform;// to transform tou dino mas
	public GameObject hbar;//to tetragono pragma panw apo ton dino
pou deixnei tin ygeia tou
	public GameObject dino;//to anime controler
	private bool die =false;// to bool gia to teliko anime tou dino
mas

	public int empeiriaTeratos=80;
	private int maxhealth=36;//heallth
	private int currhealth=6;//zwi twra
	public int cueyyy;
	public float hbpercen;// gia tin mpara ygeias
	public float pososto;//
	public GameObject jj;

	public string Tagname{
		get {return tagname;}
		set {tagname=value;}

	}
```

Παραπάνω δηλώνουμε της μεταβλητές που θα χρησιμοποιήσουμε σε αυτό το script.

```
	void Awake ()
		{
		epipedomultpaixnidiou8293=
		PlayerPrefs.GetInt("epipedomultpaixnidiou");
```

```
tager=GameObject.FindGameObjectWithTag("tager");
tg = tager.GetComponent<tagger> ();
go2 = new GameObject[8];
playerth = new Transform[8];
destination = loc2;
hbar.GetComponent<Renderer>().material.color =
Color.green;
myTransform = transform;
firerate = 2;
go = GameObject.FindGameObjectsWithTag ("Player");
mploko = GameObject.FindGameObjectsWithTag ("mploko");
r33 =GameObject.FindGameObjectWithTag ("majorhouse");
int meget = go.Length;
}
```

Παραπάνω παίρνουμε το επίπεδο δυσκολίας παιχνιδιού, δηλώνουμε την μεταβλητή go2 στην οποία θα αποθηκεύσουμε τους χρήστες που υπάρχουν στο δίκτυο και θέτουμε τον προορισμό του δεινοσαύρου το loc2, που είναι μια σταθερή θέση μέσα στην σκηνή .Το r33 είναι ένα αντικείμενο πάρα πολύ μακριά που το χρησιμοποιούμε εάν δεν υπάρχει κάποιος παίχτης. Το mploko είναι ο πίνακας που περιέχει τα σημεία μέσα στα οποία θα κινούνται οι δεινόσαυροι. Επίσης κάνουμε την μπάρα υγείας που είναι πάνω από το κεφάλι του δεινοσαύρου πράσινη.

```
for (int y=0; y<meget; y++)
{
      go2[y]=go[y];
      UserPlayer jj=go[y].GetComponent<UserPlayer>();
      playerth[y]=jj.thesi;
}
```

Παραπάνω παίρνουμε τις θέσεις των παιχτών.

```
float mjk = 80000000;
int first = 0999;
int sec = 9484;
for(int lko=0;lko<mploko.Length;lko++)
{
      if(   Vector3.Distance (mploko
            [lko].transform.position,
             myTransform.position) < mjk )
      {

            mjk=Vector3.Distance (mploko
            [lko].transform.position,
             myTransform.position);
            first=lko;
      }
}
mjk = 80000000;
for(int lko=0;lko<mploko.Length;lko++)
{
      if(   Vector3.Distance (mploko
      [lko].transform.position, myTransform.position)
       < mjk )
      {
            if(lko!=first)
            {
```

```
                        mjk=Vector3.Distance (mploko
                        [lko].transform.position,
                         myTransform.position);
                        sec=lko;
                  }
            }
      }
```

Παραπάνω με τις δύο συναρτήσεις παίρνουμε τα δύο πιο κοντινά σημεία. Μέσα σε αυτά τα σημεία θα κινείτε ο δεινόσαυρος.

```
      loc1 = mploko [first].transform;
      loc2 = mploko [sec].transform;
      if (meget < 8)
      {
            while(meget<8)
            {
                  go2[meget]=r33;
                  playerth[meget]=r33.transform;
                  meget++;
            }
      }
```

Παραπάνω γεμίζουμε τις υπόλοιπες θέσεις με ένα πολύ μακρινό gameobject.

```
void Start ()
{
      kat = dino.GetComponent<dino_ai_anime> ();
      maxdistance = 4;
      maxdistance2 = 15;
      maxhealth = 5+epipedomultpaixnidiou8293*8;
      currhealth = maxhealth;
      playerdmg = 3;
}
```

Παραπάνω δηλώνουμε από ποιά απόσταση και κάτω ο δεινόσαυρος θα κυνηγάει τους χαρακτήρες και από ποια απόσταση και πάνω θα σταματάει να τους κυνηγάει. Επίσης αυξάνουμε την υγεία του δεινοσαύρου με βάση την δυσκολία που παίζουμε.

```
// Update is called once per frame
void Update ()
{
      cueyyy = currhealth;
      for (int y=0; y<go.Length; y++)
      {
            UserPlayer jj = go [y].GetComponent<UserPlayer> ();
            playerth [y] = jj.thesi;
            if (trrr > Vector3.Distance (playerth [y].position,
             myTransform.position))
            {
                  dok = playerth [y];
                  trrr = Vector3.Distance (playerth
                  [y].position, myTransform.position);
            }
      }
}
```

Παραπάνω βρίσκουμε κάθε φορά τον κοντινότερο παίχτη στον δεινόσαυρο.

```
if (locconr)
{
      destination = loc2;
}
else
{
      destination = loc1;
}
```

Παραπάνω βάζουμε τις θέσεις στις οποίες πηγαίνει εάν δεν έχει συναντήσει κάποιον παίχτη.

```
if (die)
{

      if(Network.isServer)
      {
            Network.Destroy (gameObject);
      }

}
```

Παραπάνω καταστρέφουμε το gameobject σε όλα τα instances του παιχνιδιού.

```
if (!z1)
{
      kinigaw = true;
}
else
{
    if (Vector3.Distance (playerth [0].position,
      myTransform.position) > maxdistance2 &&
          Vector3.Distance (playerth [1].position,
          myTransform.position) > maxdistance2 &&
          Vector3.Distance (playerth [2].position,
          myTransform.position) > maxdistance2 &&
          Vector3.Distance (playerth [3].position,
          myTransform.position) > maxdistance2 &&
          Vector3.Distance (playerth [4].position,
          myTransform.position) > maxdistance2 &&
          Vector3.Distance (playerth [5].position,
          myTransform.position) > maxdistance2 &&
          Vector3.Distance (playerth [6].position,
          myTransform.position) > maxdistance2 &&
          Vector3.Distance (playerth [7].position,
          myTransform.position) > maxdistance2)
      {
            kinigaw = false;
      }
      else
      {
            trrr = 1000;
            kinigaw = true;
      }
}
```

Παραπάνω ελέγχουμε εάν είναι κάποιος παίχτης κοντά μας και εάν είναι βάζουμε το χαρακτήρα μας σε mode κυνηγιού.

```
if (!kinigaw)
{
      target = destination;

}
if (kinigaw)
{
      target = dok;

}
```

Παραπάνω εάν «δεν κυνηγάω» ο στόχος (προορισμός του δεινοσαύρου) είναι ένα από τα δυο σημεία ενώ εάν «κυνηγάω» είναι ο κοντινότερος χαρακτήρας.

```
if (!block)
{
      if ((Vector3.Distance (playerth [0].position,
      myTransform.position) < maxdistance - 1 ||
     Vector3.Distance (playerth [1].position,
     myTransform.position) < maxdistance - 1 ||
     Vector3.Distance (playerth [2].position,
      myTransform.position) < maxdistance - 1 ||
      Vector3.Distance (playerth [3].position,
      myTransform.position) < maxdistance - 1 ||
      Vector3.Distance (playerth [4].position,
      myTransform.position) < maxdistance - 1 ||
      Vector3.Distance (playerth [5].position,
      myTransform.position) < maxdistance - 1 ||
      Vector3.Distance (playerth [6].position,
      myTransform.position) < maxdistance - 1 ||
      Vector3.Distance (playerth [7].position,
      myTransform.position) < maxdistance - 1))
      {
      }
      else
      {

            myTransform.rotation = Quaternion.Slerp
            (myTransform.rotation,
            Quaternion.LookRotation (target.position -
            myTransform.position), rotationspeed *
            Time.deltaTime);// kanei rotation ston paixti

            myTransform.position = Vector3.MoveTowards
            (transform.position, target.position + ff,
            movespeed * Time.deltaTime);
            if (z)
            {
                  kat.Katastasi = 2;
            }
      }


      if (Vector3.Distance (target.position,
      myTransform.position) < maxdistance)
      {
            if (kinigaw)
            {
                  if (Time.time > nextfire)
                  {
```

```
                                    z = false;

                                    kat.Katastasi = 1;
                                    StartCoroutine (delay ());
                                    nextfire = Time.time + firerate;
                                    d = Instantiate (shot,
                                    target.position,target.rotation)
                                    as GameObject;

                                    GetComponent<AudioSource> ().Play
                                    ();
                              }
                        }
                  }
            }
```

Παραπάνω εάν ο δεινόσαυρος συναντήσει τον χρήστη και είναι σε κοντινή απόσταση του επιτίθεται.

```
                        else
                        {
                              locconr = !locconr;
                        }
```

Παραπάνω εάν ο δεινόσαυρος φτάσει στο τέλος της διαδρομή αλλάζει προορισμό και πάει προς την αρχή (από εκεί που ξεκίνησε).

```
      public void Adjusthealth(int adj){

            currhealth += adj;


            if (currhealth <= 0 )
            {

                  for(int jks=0;jks<epipedomultpaixnidiou8293;jks++)
                  {
                        Instantiate(xrysos, myTransform.position,
                        myTransform.rotation);
                  }
                  kat.Katastasi=3;
                  StartCoroutine (delay2());

            }
            if (currhealth > maxhealth)
            {
                  currhealth =maxhealth;

            }
      }
```

Παραπάνω είναι η συνάρτηση με την οποία μειώνουμε την υγεία του.

```
            if((currhealth/(float)maxhealth)<0.3f)
                  hbar.GetComponent<Renderer>().material.color =
                  Color.red;
            if((currhealth/(float)maxhealth)>=0.3f)
                  hbar.GetComponent<Renderer>().material.color =
                  Color.blue;
```

```
        hbar.transform.localScale-= new Vector3((-
        adj/(float)maxhealth),0,0);
        hbar.transform.localPosition-= new
        Vector3(((adj/(float)maxhealth)/2),0,0);
```

Παραπάνω αλλάζουμε το χρώμα της μπάρας πάνω από τον δεινόσαυρο ανάλογα με πόση υγεία του απομένει.

```
    void OnTriggerEnter(Collider other) {

        if (currhealth > 0)
        {
            if (other.gameObject.tag == "mpl1")
            {
                y = tager.GetComponent<tagger> ().dramage
                 ("mpl1");

                Adjusthealth (-1 * y);
            }
        }
```

Με παρόμοιο τρόπο το κάνουμε και για την πιθανότητα να χτυπηθεί ο εχθρός από ένα βλήμα που εκτόξευσε ένας από τους οχτώ χαρακτήρες που μπορεί να είναι συνδεδεμένοι στο δίκτυο.

```
                if (other.gameObject.tag == "mpl8")
                {
                    y = tager.GetComponent<tagger>
                    ().dramage ("mpl8");

                    Adjusthealth (-1 * y);

                }
```

Παραπάνω αφαιρούμε υγεία ανάλογα την επίθεση του παίχτη από την οποία προήλθε το βλήμα.

```
                if (other.gameObject.tag == "opla")
                {
                    damages = -1 * playerdmg;
                    Adjusthealth (damages);
                    z1 = false;
                    StartCoroutine (delaytarg ());

                }
                if (other.gameObject.tag == "opla2")
                {
                    Adjusthealth (-1 * (currhealth / 3));
                }
    IEnumerator delay2()
    {
        block = true;
        Instantiate (dinodeath,myTransform.position,
        myTransform.rotation);

        yield return new WaitForSeconds(2.26f);

        die = true;
```

```
}
```

Παραπάνω μπλοκάρουμε τον δεινόσαυρο από το να κινείτε. Αυτό γίνετε κάνοντας το block true, παίζουμε το anime clip που πεθαίνει και το κατάλληλο ήχο.

```
IEnumerator delay()
{
      yield return new WaitForSeconds(1f);
      z = true;
      kat.Katastasi = 4;
}
IEnumerator delaytarg()
{
      yield return new WaitForSeconds(3f);
      z1 = true;
}
```

Κεφάλαιο 5
Αξιολόγηση συστήματος

5.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζουμε μια μέθοδο αξιολόγησης του παιχνιδιού που δημιουργήσαμε στα πλαίσια της διπλωματικής εργασίας.

5.2 Μέθοδος αξιολόγησης και αποτελέσματα ερωτηματολόγιου

Για να αξιολογήσουμε το πρόγραμμα χρησιμοποιούμε το ερωτηματολόγιο για την ικανοποίηση της αλληλεπίδρασης του χρήστη. Το ερωτηματολόγιο αποτελείται από είκοσι έξι ερωτήσεις και χωρίζεται σε πέντε διαφορετικές ενότητες. Παρακάτω παρατίθεται το ερωτηματολόγιο [12] και τα αποτελέσματα που πήραμε:

Ερωτηματολόγιο Αξιολόγησης Διαλογικής Χρήσης Συστήματος
(QUIS - Πανεπιστήμιο του Maryland, 1986-1998)

ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη

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

1.1. Γενική αντίδραση του Συστήματος:

Απαράδεκτη *υπέροχη*
1 2 3 4 5 6 7 8 9 *NA*

1.2. Γενική αντίδραση του Συστήματος :

δύσκολο *εύκολο*
1 2 3 4 5 6 7 8 9 NA

1.3. Γενική αντίδραση του Συστήματος:

Μπερδεύει *ικανοποιεί*
1 2 3 4 5 6 7 8 9 *ΝΑ*

1.4. Γενική αντίδραση του Συστήματος :

Αδύναμο *δυνατό*
1 2 3 4 5 6 7 8 9 ΝΑ

1.5. Γενική αντίδραση του Συστήματος :

Χαζό *ευχάριστο*
1 2 3 4 5 6 7 8 9 ΝΑ

1.6. Γενική αντίδραση του Συστήματος :

άκαμπτο *ευέλικτο*
1 2 3 4 5 6 7 8 9 ΝΑ

Ερώτηση	Μ.Ο. Βαθμολογίας
1.1	7.6
1.2	8
1.3	7.6
1.4	7.3
1.5	6.8
1.6	7

Πίνακας 5.1 Αποτελέσματα αξιολόγησης πρώτου μέρους

ΜΕΡΟΣ 2: Οθόνη

2.1 Ο σχεδιασμός της οθόνης βοήθησε:

ποτέ *πάντα*
1 2 3 4 5 6 7 8 9 ΝΑ

2.2 Ποσότητα πληροφορίας που εμφανίζονταν στην οθόνη :

Ανεπαρκής *επαρκής*
1 2 3 4 5 6 7 8 9 ΝΑ

2.3 Δόμηση της πληροφορίας που εμφανίζονταν στην οθόνη :

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

2.4 Αλληλουχία οθονών :

Μπερδεμένη *σαφής*
1 2 3 4 5 6 7 8 9 NA

Ερώτηση	Μ.Ο. Βαθμολογίας
2.1	7.8
2.2	8.2
2.3	7.8
2.4	8.7

Πίνακας 5.2 Αποτελέσματα αξιολόγησης δευτέρου μέρους

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

3.1 Τα μηνύματα εμφανίζονται στην οθόνη με :

Ασυνέπεια *συνέπεια*
1 2 3 4 5 6 7 8 9 NA

3.2 Εκτελώντας μια κίνηση οδηγούμαστε σε προβλέψιμο αποτέλεσμα :

ποτέ *πάντα*
1 2 3 4 5 6 7 8 9 NA

3.3 Θέση μηνυμάτων στην οθόνη

αποδοτικά *μη αποδοτικά*
1 2 3 4 5 6 7 8 9 NA

3.4 Τα μηνύματα που εμφανίζονται στην οθόνη είναι:

μπερδεμένα *σαφή*
1 2 3 4 5 6 7 8 9 NA

3.5 Ο υπολογιστής σας ενημερώνει για το τι κάνει:

Ποτέ *πάντα*
1 2 3 4 5 6 7 8 9 NA

3.6 Μηνύματα λαθών :

δεν βοηθούν *πολύ βοηθητικά*
1 2 3 4 5 6 7 8 9 NA

Ερώτηση	Μ.Ο Βαθμολογίας
3.1	7
3.2	8
3.3	7.6
3.4	9.3
3.5	-
3.6	-

Πίνακας 5.3 Αποτελέσματα αξιολόγησης τρίτου μέρους

ΜΕΡΟΣ 4: Εκμάθηση χρήσης

4.1 Η εκμάθηση χρήσης του συστήματος είναι:
δύσκολη *εύκολη*
1 2 3 4 5 6 7 8 9 NA

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

Απογοητεύει *αποδίδει*
1 2 3 4 5 6 7 8 9 NA

4.3 Η εκμάθηση των εντολών είναι :

δύσκολη *εύκολη*
1 2 3 4 5 6 7 8 9 NA

4.4 Τα βήματα για την ολοκλήρωση της εργασίας ακολουθούν μια λογική σειρά

ποτέ *πάντα*
1 2 3 4 5 6 7 8 9 NA

4.5 Βοηθητικά μηνύματα στην οθόνη :

δεν βοηθούν *πολύ βοηθητικά*
1 2 3 4 5 6 7 8 9 NA

4.6 Ανάδραση όταν ολοκληρωθεί η εργασία:

Ασαφής σαφής
1 2 3 4 5 6 7 8 9 NA

Ερώτηση	Μ.Ο Βαθμολογίας
4.1	5.8
4.2	8.3
4.3	6.8
4.4	8.2
4.5	7.4
4.6	8.3

Πίνακας 5.4 Αποτελέσματα αξιολόγησης τέταρτου μέρους

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

5.1. Η ταχύτητα του Συστήματος είναι:

πολύ αργή ικανοποιητική
1 2 3 4 5 6 7 8 9 NA

5.2. Το Σύστημα είναι σταθερό :

Ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

5.3 Το Σύστημα τείνει να γίνει :

Θορυβώδες ήσυχο
1 2 3 4 5 6 7 8 9 NA

5.4 Η ευκολία χειρισμού εξαρτάται από την εμπειρία του χρήστη:

Ποτέ πάντα
1 2 3 4 5 6 7 8 9 NA

Ερώτηση	Μ.Ο. Βαθμολογίας
5.1	8.3
5.2	7.8
5.3	6.8
5.4	8.3

Πίνακας 5.5 Αποτελέσματα αξιολόγησης πέμπτου μέρους

5.3 Συμπεράσματα

Παρακάτω θα αναλύσουμε τα συμπεράσματα από το ερωτηματολόγιο με βάση τις πέντε κατηγορίες που το αποτελούν.

ΜΕΡΟΣ 1: Γενική εντύπωση του χρήστη

Η γενική εντύπωση ήταν αρκετά ικανοποιητική. Γενικότερα επειδή το παιχνίδι αποτελούταν από διαφορετικά κομμάτια ικανοποιούσε μεγάλο εύρος τον χρηστών.

ΜΕΡΟΣ 2: Οθόνη

Εδώ ήταν τα πιο θετικά αποτελέσματα, κανείς δεν είχε πρόβλημα με τις πληροφορίες που εμφανίζονταν στην οθόνη. Γενικότερα οι χρήστες ήταν αρκετά ικανοποιημένοι με την εμφάνιση της γραφικής διεπαφής του παιχνιδιού.

ΜΕΡΟΣ 3: Ορολογία και επικοινωνία με το Σύστημα

Δεν παρατηρήθηκαν προβλήματα από τους χρήστες και οι μπάρες με την υγεία πάνω από τους αντιπάλους ήταν αρκετά δημοφιλείς.

ΜΕΡΟΣ 4: Εκμάθηση χρήσης

Εδώ παρατηρήθηκε το μεγαλύτερο πρόβλημα. Στο action κομμάτι της εφαρμογής άτομα που δεν είχαν ιδιαίτερη σχέση με βιντεοπαιχνίδια και με την λογική τους δυσκολευτήκαν αρκετά να κατανοήσουν πως λειτουργεί. Έτσι υπήρχε μεγάλη διαφορά στο εύρος των απαντήσεων. Το μόνο που συμφώνησαν όλοι οι χρήστες είναι ότι παίζοντας το παιχνίδι το μάθαιναν πιο εύκολα (trial and error).

ΜΕΡΟΣ 5: Δυνατότητες του Συστήματος

Το σύστημα αποδείχτηκε πάρα πολύ σταθερό σχεδόν κανένας χρήστης δεν είχε πρόβλημα. Το παιχνίδι κόλλησε στο multiplayer κομμάτι μερικές φορές σε διάφορους χρήστες αλλά υπάρχουν πολλοί διαφορετικοί παράγοντες εκτός του παιχνιδιού που μπορεί να το προκάλεσαν αυτό (μειωμένη ταχύτητα στο ίντερνετ, προβλήματα με τον Masterserver της Unity, προβλήματα με το antivirus κλπ).

5.4 Αλλαγές λόγω αξιολόγησης

Με βάση σχόλια και παρατηρήσεις χρηστών βελτιώσαμε σε διάφορους τομείς την εφαρμογή μας. Αναλυτικότερα για το κομμάτι πλατφόρμας μας έγιναν διάφορες

παρατηρήσεις για την κοντινή απόσταση της κάμερας στον χρήστη, οπότε την πήγαμε πιο μακριά καθώς επίσης μας έγιναν παρατηρήσεις για το πλήθος των εχθρών οπότε και το μειώσαμε. Στο κομμάτι δράσης μας έγιναν παρατηρήσεις πάλι για την κάμερα οπότε βάλαμε την δυνατότητα αλλαγής θέσης της κάμερα (δηλαδή αν ο χρήστης πατήσει το κουμπί V μετακινεί την κάμερα του χαρακτήρα ανάποδα ώστε να μπορεί να βλέπει και πίσω του), επίσης έγιναν κάποιες παρατηρήσεις από κάποιους χρήστες ότι δεν γνώριζαν εάν είχαν ανέβει επίπεδο για να ανεβάσουν τα στατιστικά του (θέλανε δηλαδή να ενημερώνονται κατά την διάρκεια του παιχνιδιού ότι ανέβηκαν επίπεδο χωρίς να ανοίγουν το μενού με το esc για να το κοιτάνε). Έτσι όταν ένας χρήστης ανέβει επίπεδο ή μπορεί να ανεβάσει το στατιστικό του χαρακτήρα του ή μπορεί να ανεβάσει το στατιστικό του όπλου εμφανίζουμε ένα κουμπί πάνω δεξιά για να τον ενημερώνουν. Στο turn base κομμάτι μας έγιναν κάποιες παρατηρήσεις για την μη δυνατότητα του χαρακτήρα να χτυπάει όλους τους εχθρούς με μια επίθεση οπότε βάλαμε ο χαρακτήρας όταν κάνει επίθεση από μακριά να έχει την δυνατότητα να επιλέξει να δημιουργεί πυραύλους όσους είναι και οι εχθροί αλλά να κάνει μόνο την μισή ζημιά. Όσον αφορά το τελευταίο κομμάτι με το παιχνίδι με την συμμετοχή πολλών χρηστών στην αρχή μας έγινε η παρατήρηση ότι φαινόταν “άδειο” οπότε προσθέσαμε δύο επιπλέον επίπεδα (στην αρχή είχαμε υλοποιήσει μόνο το death match κομμάτι στην συνέχεια προσθέσαμε τα επίπεδα συνεργασίας μεταξύ των χρηστών), εχθρούς για να μπορούν να σκοτώνουν ώστε να μην σκοτώνονται μόνο μεταξύ τους οι χρήστες και δυνατότητα αύξησης δυσκολίας επιπέδου (αυξάνοντας το οι εχθροί γίνονται ποιο δύσκολοι αλλά επίσης ρίχνουν περισσότερο χρυσό όταν σκοτωθούν).

5.5 Επίλογος

Ο σκοπός της εργασίας ήταν η εισαγωγή στους κανόνες και τα αντικείμενα για την δημιουργία ενός τρισδιάστατου παιχνιδιού και οι βασικοί κανόνες και λογικές για την υλοποίηση του κομματιού συμμετοχής πολλών παιχτών. Στην αρχή σχεδιάσαμε το σενάριο του παιχνιδιού και τις βασικές αλληλεπιδράσεις που έχουν τα αντικείμενα στο παιχνίδι μας. Στη συνέχεια αφού μάθαμε πώς να προγραμματίζουμε και να λειτουργούμε στην γλώσσα παιχνιδιού Unity υλοποιήσαμε το παιχνίδι μας. Η υλοποίηση του παιχνιδιού είναι αρκετά απαιτητική και χρονοβόρα εργασία. Ενώ αρχικά έγινε προσπάθεια σχεδιασμού των αντικειμένων από το μηδέν(σχεδιασμός μοντέλων παιχτών, δημιουργία μουσικής κλπ) τελικά αποφασίσαμε να πάρουμε έτοιμα τα μοντέλα και τους ήχους από το Unity Asset Store. Για την δημιουργία ενός παιχνιδιού χρειάζονται πολλές ομάδες ατόμων διαφόρων ειδικοτήτων για να το υλοποιήσουν. Ο καθένας από αυτούς τους ανθρώπους έχει γνώση σε διάφορες ειδικότητες (π.χ. προγραμματισμό, σχεδιασμό τρισδιάστατων μοντέλων κλπ). Αυτή η πρώτη προσπάθεια παρέχει τις βασικές γνώσεις για ένα μεγάλο μέρος της προγραμματικής λογικής που χρειάζεται ένα παιχνίδι.

Βιβλιογραφία

[1] Unity, Unity Game Engine http://unity3d.com/unity
[2] Unity Manual, Unity Game Engine http://docs.unity3d.com/Manual
[3] Unity Pro and Unity Personal Software License Agreement 5.x, Unity Game Engine https://unity3d.com/legal/eula
[4] NetworkView Unity Manual, Unity Game Engine
http://docs.unity3d.com/Manual/net-NetworkView.html
[5] Video Game History Timeline, The Strong National Museum Of Play
http://www.museumofplay.org/icheg-game-history/timeline/
[6] Article: "The Benefits of Playing Video Games," Isabela Granic, PhD, Adam Lobel, PhD, and Rutger C.M.E. Engels, PhD, Radboud University Nijmegen; Nijmegen, The Netherlands; American Psychologist, Vol. 69, No. 1
http://www.apa.org/news/press/releases/2013/11/video-games.aspx
[7] Video Game Genres ,Ted Stahl
http://www.thocp.net/software/games/reference/genres.htm
[8] Documentation About Mono, The Mono Project http://www.mono-project.com/docs/about-mono/
[9] Default Values Table (C# Reference) , Microsoft Developer Network
https://msdn.microsoft.com/en-us/library/83fhsxwc.aspx
[10] Super Mario Wikipedia, Super Mario wiki
http://mario.wikia.com/wiki/Super_Mario_Bros.
[11] Mortal Kombat, Mortal Kombat Game Store Page
http://www.mortalkombat.com/en/buynow/
[12] Questionnaire for User Interface Satisfaction, Chin, J.P., Diehl, V.A., Norman, K.L. (1988) Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface. http://garyperlman.com/quest/quest.cgi?form=QUIS
[13] Unity Asset Store, Unity Game Engine https://www.assetstore.unity3d.com/en/
[14] Multiplayer Tutorial ,Paladin studios
http://www.paladinstudios.com/2013/07/10/how-to-create-an-online-multiplayer-game-with-unity/
[15] Game Tutorial, Game To Gamer Tutorial Series
http://www.gamertogamedeveloper.com/gtgd-series-1/video-1
[16] Networking in Unity, Marc Bastiaansen
http://www.3dgep.com/networking-in-unity-3-5/
[17] Unity tutorials, Unity Game Engine https://unity3d.com/learn/tutorials
[18] Thomas T.Goldsmith,Jr, Wikipedia the free encyclopedia
https://en.wikipedia.org/wiki/Thomas_T._Goldsmith,_Jr.
[19] Arcade Games, Wikipedia the free encyclopedia
https://en.wikipedia.org/wiki/Arcade_game
[20] Rendering, Wikipedia the free encyclopedia
https://en.wikipedia.org/wiki/Rendering_(computer_graphics)
[21]Basic language, Wikipedia the free encyclopedia
https://en.wikipedia.org/wiki/BASIC

Χρήσιμοι σύνδεσμοι

Game sounds
final fantasy VII game chocobo song
https://www.youtube.com/watch?v=x4UDIfcYMKI
Wild arms theme song
https://www.youtube.com/watch?v=ajMRQnhQ9zU