Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Τίτλος: Ανάπτυξη ηλεκτρονικού παιχνιδιού μέσω Unity

Πατεράκης Αντώνιος (ΑΜ: 3666)

Επιβλέπων Καθηγητής: κ. Παχουλάκης Ιωάννης

Επιτροπή Αξιολόγησης:

Ημερομηνία Παρουσίασης:

Abstract

This thesis concerns the developement of a shared-screen Action Multiplayer videogame, which is implemented using the Unity GameEngine and the programming language c#, which the engine supports.

This thesis includes a short reference on Unity's environment and some related concepts, such as GameObject, Prefab, etc. References related to the videogame are also presented, so the reader can get familiar and understand its nature (i.e. the player, enemies, other entities, etc). The main purpose of the game is for the players who are standing within a game arena to defeat a team of enemy creatures that appear in every level and manage to survive. Each consequent level has a greater degree of difficulty than the previous one.

Next, details about some of the processes that were used to create the game are given, such as how the game arena and the main menu were constructed. Furthermore, there is an explanation on the basic scripts that were used, concerning the logic in which they operate and the results they achieve. Afterwards, there is a detailed explanation on the scripts' code that is used for the game's and the player's functionality. Finally, some pictures of the gameplay are being presented.

Σύνοψη

Η παρούσα πτυχιακή εργασία αφορά την ανάπτυξη ενός shared-screen Action multiplayer ηλεκτρονικού παιχνιδού το οποίο υλοποιήθηκε με την βοήθεια της μηχανής ανάπτυξης ηλεκτρονικών παιχνιδιών Unity3D και της γλώσσας προγραμματισμού C# την οποία υποστηρίζει.

Κατά την έκταση αυτής της πτυχιακής θα γίνει μία σύντομη αναφορά στο περιβάλλον της Unity καθώς και σε έννοιες οι οποίες έχουνε άμεση σχέση με αυτή όπως για παράδειγμα τα GameObject, Prefab και άλλα. Στην συνέχεια γίνεται αναφορά στο παιχνίδι έτσι ώστε να το γνωρίσει ο αναγνώστης και να καταλάβει περί τίνος πρόκειται. Παρουσιάζονται ο παίκτης, οι εχθροί και διάφορες άλλες οντότητες που αποτελούν το παιχνίδι. Ο κεντρικός σκοπός του παιχνιδιού είναι οι παίκτες, στο πλαίσιο μιας πίστας να εξοντώσουν και να επιβιώσουν ενάντια σε μία ομάδα εχθρών που εμφανίζεται σε κάθε επίπεδο. Κάθε επόμενο επίπεδο έχει μεγαλύτερο βαθμό δυσκολίας από το προηγούμενο.

Έπειτα, αναφέρονται λεπτομέρειες για το πως δημιουργήθηκαν κάποια από τα μέρη του παιχνιδιού, όπως η πίστα και το αρχικό μενού. Παράλληλα εξηγούνται τα βασικά script που χρησιμοποιήθηκαν μαζί με την λογική με την οποία λειτουργούν, το σκοπό που επιτελούν και τα αποτελέσματα που καταφέρνουν. Στη συνέχεια, γίνεται η αναλυτική επεξήγηση του κώδικα των script που αφορούν την λειτουργικότητα του παίκτη και του παιχνιδιού. Τέλος, παρουσιάζονται στιγμιότυπα από το gameplay του παιχνιδιού.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Λίστα Πινάκων

1 Εισαγωγή

1.1 Περίληψη

Η πτυχιακή εργασία αναφέρεται στην διαδικασία ανάπτυξης ενός ηλεκτρονικού παιχνιδιού, χρησιμοποιώντας την μηχανή παιχνιδιών της Unity[1]. Δίνεται μεγαλύτερη βάση στην πλατφόρμα της Unity και στις δυνατότητές που προσφέρει καθώς και στον προγραμματισμό, ο οποίος παίζει εξίσου μεγάλο ρόλο στην ανάπτυξη του παιχνιδιού. Η έκδοση της Unity που χρησιμοποιήθηκε είναι η 5.4.1f1 Personal.

Για την δημιουργία του παιχνιδιού χρησιμοποιήθηκε το Terrain της Unity, έτσι ώστε να δημιουργηθεί το έδαφος της πίστας. Επάνω σε αυτό, τοποθετήθηκαν διάφορα αντικείμενα όπως βράχοι και δέντρα τα οποία βρέθηκαν μέσω του Asset Store. Για την διαχείρηση των animation των παικτών και των εχθρών χρησιμοποιήθηκε το Animator Controller της Unity, έτσι ώστε με βάση συγκεκριμένων μεταβλητών να ελέγχεται η αλλαγή της κατάστασης του animation κάθε οντότητας μέσω script. Η λειτουργικότητα του παιχνιδιού έγινε αξιοποιώντας πολλά από τα Component της Unity καθώς και μέσω του προγραμματισμού διαφόρων script σε C#.

Τα μοντέλα που χρησιμοποιήθηκαν ήταν έτοιμα και βρέθηκαν είτε από το Asset Store της Unity είτε από το διαδίκτυο. Η άδεια χρήσης τους, τουλάχιστον για μη εμπορική χρήση διατίθεται δωρεάν. Οι ήχοι που χρησιμοποιήθηκαν ανήκουν επίσης σε αυτήν την κατηγορία και αντλήθηκαν κυρίως από την ιστοσελίδα www.freesound.org[2].

1.2 Κίνητρο για την Διεξαγωγή της Εργασίας

Το κίνητρο για την διεξαγωγή της εργασίας αυτής ήταν η θέληση για εξάσκηση και εξοικείωση με την ανάπτυξη και οργάνωση ενός παιχνιδιού. Αυτό περιέχει την κεντρική ιδέα και σχεδίαση του παιχνιδιού, την δημιουργία της πίστας, την τοποθέτηση διάφορων αντικειμένων και οντοτήτων σε αυτή και επιπλέον την χρήση της γλώσσας προγραμματισμού, έτσι ώστε να υπάρξει λειτουργικότητα και αλληλεπίδραση.

Επίσης, άλλο κίνητρο ήταν η ευκαιρία να υλοποιηθεί ένα σχετικά ολοκληρωμένο και μεγάλο σε έκταση project. Αυτό έχει ως αποτέλεσμα να έρθεις αντιμέτωπος με νέες προκλήσεις και προβλήματα τα οποία δεν είναι εμφανή και δεν παρουσιάζονται σε εργασίες μικρότερης κλίμακας. Η προσπάθεια αυτή, συμβάλει ταυτόχρονα στην απόκτηση περισσότερης εμπειρίας πάνω σε ρεαλιστικά ζητήματα που πιθανόν απαιτούνται αργότερα στον εργασιακό τομέα.

1.3 Σκοπός και Στόχοι Εργασίας

Σκοπός της εργασίας αυτής είναι η ανάπτυξη και ολοκλήρωση ενός ηλεκτρονικού παιχνιδιού μέσω της μηχανής παιχνιδιών Unity. Στόχος είναι το αποτέλεσμα αυτής να αποτελεί ένα λειτουργικό παιχνίδι, το οποίο να κινεί σε κάποιο βαθμό το ενδιαφέρον του παίκτη και να τον ψυχαγωγεί κατά την διάρκεια του.

Όσον αφορά την διαδικασία δημιουργίας του παιχνιδιού, σαν στόχος ήταν η εξάσκηση, η ενασχόληση καθώς και η κατάκτηση των κατάλληλων γνώσεων που απαιτούνται για την διεκπεραίωση της εργασίας αυτής.

1.4 Δομή Εργασίας

Η εργασία αυτή αρχίζει κάνοντας μία εισαγωγή στην μηχανή παιχνιδιών της Unity η οποία χρησιμοποιήθηκε για την κατασκευή του παιχνιδιού. Αυτό γίνεται με σκοπό ο αναγνώστης να αποκτήσει μία κεντρική ιδέα για το περιβάλλον το οποίο προσφέρει στους χρήστες της. Επίσης, δίνεται βάση σε κεντρικές έννοιες οι οποίες έχουν άμεση σχέση με τον τρόπο λειτουργίας και αρχιτεκτονικής της Unity και του API που αυτή προσφέρει.

Στην συνέχεια, στο επόμενο κεφάλαιο γίνεται αναφορά και εισαγωγή στην φύση του παιχνιδιού. Αναφέρεται το κεντρικό θέμα το οποίο έχει το παιχνίδι και το ποιος είναι ο σκοπός του παίκτη. Παρουσιάζονται οι εχθροί που υπάρχουν σε αυτό, η μορφή τους και ταυτόχρονα τα χαρακτηριστικά και οι ιδιαιτερότητες που έχει ο καθένας. Πέρα από τους εχθρούς, περιέχεται επίσης αναφορά στα μοντέλα των δύο παικτών που πρωταγωνιστούν στο παιχνίδι. Επιπλέον, αναφέρονται και εξηγούνται όλα τα spells που υπάρχουν μέσα στο παιχνίδι περιγράφοντας το τι κάνει το κάθε ένα. Εξηγούνται επίσης έννοιες που παίζουν σημαντικό ρόλο στο παιχνίδι, όπως stats και Buffstands.

Στο τέταρτο κεφάλαιο αρχίζει η παρουσίαση των τεχνικών μερών της δημιουργίας του παιχνιδιού. Αυτή περιέχει τα βήματα και την φιλοσοφία για την κατασκευή της πίστας μέσω του Terrain της Unity, την κατασκευή του αρχικού μενού και την παραμετροποίηση του AnimatorController του παίκτη και των εχθρών. Πέρα από αυτά, στο τέλος του κεφαλαίου αυτού καταγράφονται τα σημαντικότερα script του παιχνιδιού και εξηγείτε ο τρόπος λειτουργίας τους και ο σκοπός τον οποίο επιτελούν.

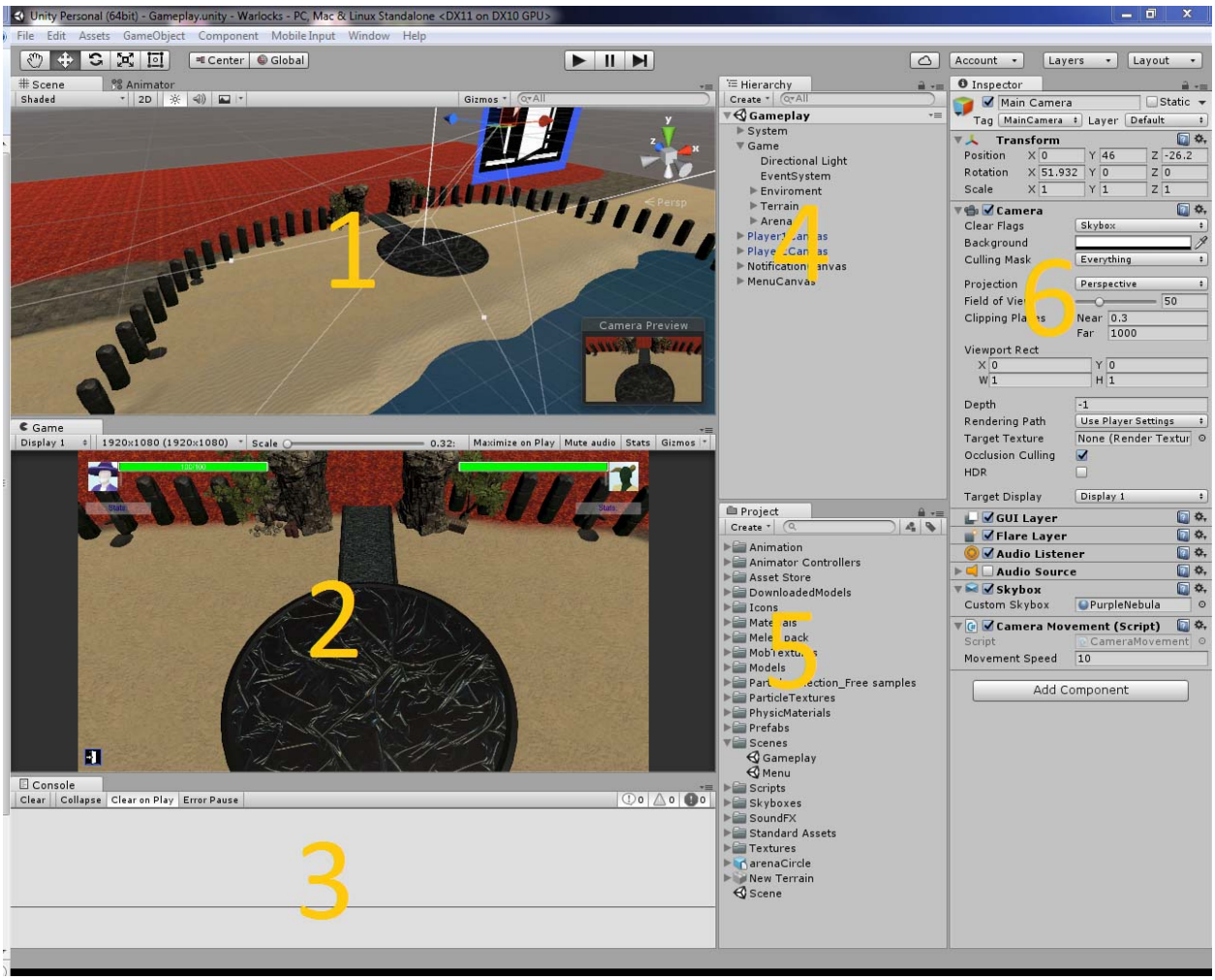
Στο πέμπτο κεφάλαιο γίνεται αναλυτική παρουσίαση και επεξήγηση του κώδικα όλων των script των οποίων ο ρόλος αφορά την λειτουργικότητα του παίκτη και του παιχνιδιού. Στο τελευταίο κεφάλαιο παρουσιάζονται μερικά στιγμιότυπα από το gameplay του τελικού παιχνιδιού, καθώς επίσης και οι δυσκολίες που συναντήθηκαν κατά την διαδικασία ανάπτυξής του. Επίσης, προτείνονται διάφορες ιδέες για πιθανή μελλοντική ανάπτυξη και βελτίωση του παιχνιδιού.

2 Unity

Η Unity είναι μία πλέον διαδεδομένη μηχανή ανάπτυξης ηλεκτρονικών παιχνιδιών από την Unity Technologies. Κύριο χαρακτηριστικό της είναι το μεγάλο πλήθος των πλατφόρμων που υποστηρίζει και το φιλικό της περιβάλλον προς τον χρήστη. Όσο αφορά το scripting, υποστηρίζει τις γλώσσες προγραμματισμού C# και JavaScript.

2.1 Το περιβάλλον της Unity

Το περιβάλλον της Unity είναι αρκετά φιλικό προς τον χρήστη και προσφέρει πολλές δυνατότητες. Στο Σχήμα 1 φαίνονται τα κύρια υποπαράθυρα που υπάρχουν στο περιβάλλον της και εξηγούνται στις παρακάτω υποενότητες, σε αντιστοιχία με τον αριθμό που εμφανίζεται σε κάθε υποπαράθυρο.



Σχήμα 1: Το περιβάλλον της Unity

1. Scene Window
2. Game Window
3. Console Window

4. Hierarchy Window
5. Project Window
6. Inspector

2.1.1 Scene Window

Το παράθυρο αυτό περιέχει την σκηνή και όλα τα αντικείμενα τα οποία υπάρχουν μέσα σε αυτήν. Εκεί, ο χρήστης έχει την δυνατότητα να προσθέτει, να επιλέγει, να μετακινεί και να τροποποιεί αντικείμενα. Με λίγα λόγια είναι ο χώρος εργασίας και στησίματος της σκηνής.

2.1.2 Game Window

Το Game Window παρουσιάζει την εικόνα του παιχνιδιού, από την οπτική του πως θα το βλέπει ο παίκτης. Το πλάνο το οποίο δείχνει προέρχεται από μια προκαθορισμένη κάμερα που έχουμε επιλέξει στο παιχνίδι. Ο χρήστης έχει την δυνατότητα να παίξει το παιχνίδι εφόσον πατήσει το κουμπί ”Play”. Αυτό βοηθάει αυτόν που φτιάχνει το παιχνίδι, διότι παράλληλα με την ανάπτυξή του παιχνιδιού μπορεί να βλέπει και να δοκιμάζει την συμπεριφορά του, έτσι ώστε να αποφασίζει σταδιακά για επιθυμητές αλλαγές και διορθώσεις.

2.1.3 Console Window

Το Console Window εμφανίζει διάφορες παρατηρήσεις και λάθη που μπορεί να υπάρχουν στον κώδικα που έχουν τα scripts. Σκοπός του είναι να δείξει στον προγραμματιστή σε ποια σημεία του κώδικα υπάρχει κάποιο λάθος το οποίο είτε μπορεί να εμποδίζει το compile του προγράμματος, είτε να παρουσιάζεται κατά την διάρκεια του παιχνιδιού. Επίσης χρησιμοποιείται για την εμφάνιση διάφορων μηνυμάτων που ορίζει ο προγραμματιστής με σκοπό το debugging.

2.1.4 Hierarchy Window

Σε αυτό το παράθυρο υπάρχει η ιεραρχία των αντικειμένων που βρίσκονται στην σκηνή. Για κάθε αντικείμενο υπάρχει δυνατότητα εμφάνισης των αντικειμένων που είναι παιδιά του. Έτσι, μας δίνει μια ιδέα για τα αντικείμενα που χρησιμοποιούνται σε μία σκηνή και κάνει εύκολη την επιλογή κάποιοι αντικειμένου για επεξεργασία.

2.1.5 Project Window

Το Project Window περιέχει συνήθως τα αρχεία τα οποία χρησιμοποιούνται για το παιχνίδι, δείχνοντάς μας την ιεραρχία των φακέλων και αρχείων. Τα αρχεία αυτά μπορεί να είναι οτιδήποτε, για παράδειγμα εικόνες, ήχοι, μοντέλα, σκηνές, animatons, scripts και τα λοιπά.

2.1.6 Inspector

Στο παράθυρο του Inspector εμφανίζονται όλα τα components που υπάρχουν σε ένα αντικείμενο που έχει επιλεχθεί. Αυτό μας δίνει μια γενική εικόνα για τα χαρακτηριστικά και τις συμπεριφορές που μπορεί να έχει ένα αντικείμενο. Επίσης κάνει εύκολη την προσθήκη ή αφαίρεση ενός component καθώς και την παραμετροποίηση των πεδίων των component αυτών. Πέρα απ' αυτά, εμφανίζει το όνομα του αντικειμένου, το Tag που μπορεί να έχει και σε ποιο Layer ανήκει.

2.2 Σημαντικές έννοιες στην Unity

Σε αυτήν την υποενότητα θα αναφερθούν και θα εξηγηθούν κάποιες σημαντικές έννοιες που είναι άμεσα συνδεδεμένες με την Unity.

2.2.1 GameObject

Το GameObject είναι βασική έννοια στην Unity. Κάθε αντικείμενο το οποίο χρησιμοποιείται μέσα σε μία σκηνή, χαρακτηρίζεται ως GameObject. Τα GameObjects δεν έχουν από μόνα τους ιδιότητες, αλλά τις κληρονομούν με βάση τα components τα οποία κρατάνε.

2.2.2 Prefab

Ένα Prefab αντιπροσωπεύει ένα αποθηκευμένο GameObject (μαζί με τα components που περιέχει). Χρησιμοποιείται σαν πρότυπο αντικείμενο από το οποίο μπορούν να δημιουργηθούν στην σκηνή πολλά αντίτυπα αυτού. Σε αυτά υπάρχει δυνατότητα τροποποίησης, έτσι ώστε το κάθε ένα ναι μεν να προέρχεται από το ίδιο prefab, αλλά να έχει κάποια διαφορετικά χαρακτηριστικά. Κάθε ολοκληρωμένη αλλαγή σε ένα prefab γίνεται άμεσα και στα αντίτυπα αυτού.

2.2.3 Component

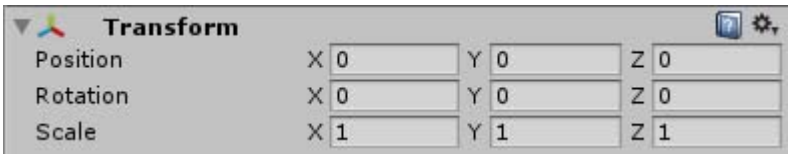
Τα components εμπεριέχονται στα GameObjects. Σκοπός του είναι να δώσουν σε αυτά διάφορες ιδιότητες και συμπεριφορές. Είναι ο συνδυασμός αυτών που κάνει ένα GameObject να μοιάζει και να συμπεριφέρεται σαν έναν χαρακτήρα, ένα περιβάλλον η κάποιο ειδικό εφέ.

2.3 Βασικά Components στην Unity

Η Unity περιέχει μεγάλο αριθμό από components, το κάθε ένα από τα οποία διαχειρίζεται και διευκολύνει κάποιο συγκεκριμένο πρόβλημα. Παρακάτω αναφέρονται και περιγράφονται τα βασικότερα από αυτά. Οι πληροφορίες σε αυτήν την υποενότητα αντλήθηκαν από την ιστοσελίδα με το Documentation της Unity[3].

2.3.1 Transform

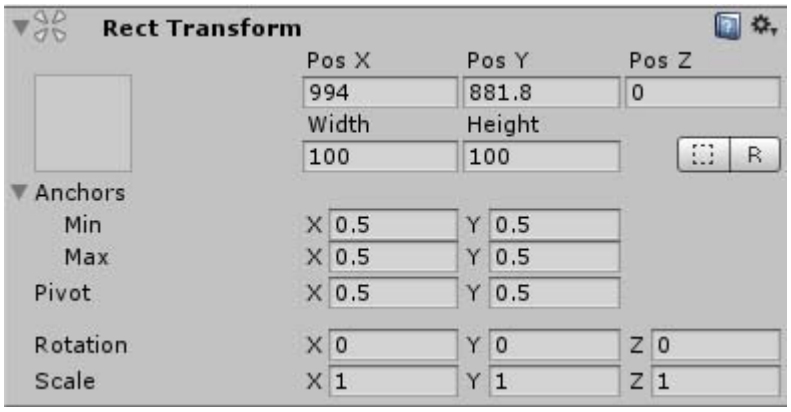
Όλα τα GameObjects έχουν το component Transform. Αυτό καθορίζει την θέση, την περιστροφή και την κλίμακα μεγέθους ενός GameObject.



Σχήμα 2: Transform Component

2.3.2 Rect Transform

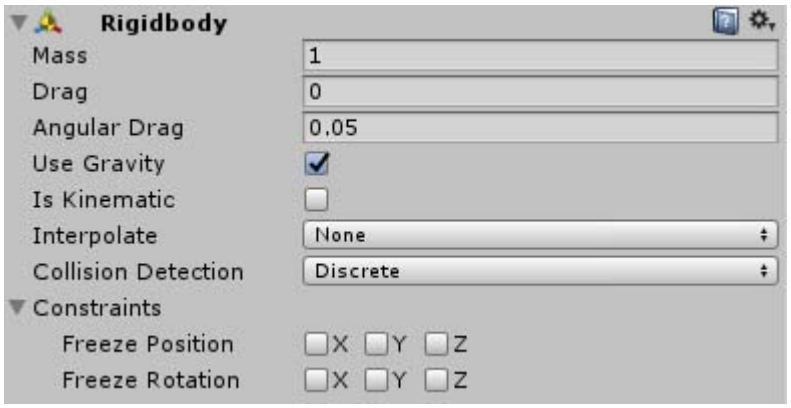
Είναι αντίστοιχο του Transform αλλά σε 2D διάταξη. Ενώ το Transform αντιπροσωπεύει ένα σημείο στη σκηνή, το RectTransform αντιπροσωπεύει ένα ορθογώνιο παραλληλόγραμμο μέσα στο οποίο μπορεί να τοποθετηθεί ένα UI element.



Σχήμα 3: Rect Transform Component

2.3.3 Rigidbody

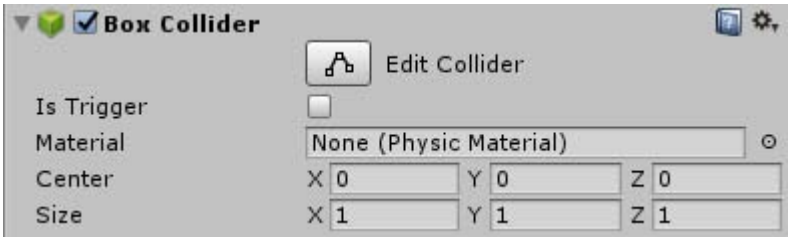
Το RigidBody component δίνει σε ένα GameObject τη δυνατότητα να δρα υπό τον έλεγχο της φυσικής. Δηλαδή να επηρεάζεται από δυνάμεις και ροπές, κάνοντας το να κινείται με ένα ρεαλιστικό τρόπο.



Σχήμα 4: Rigidbody Component

2.3.4 Collider

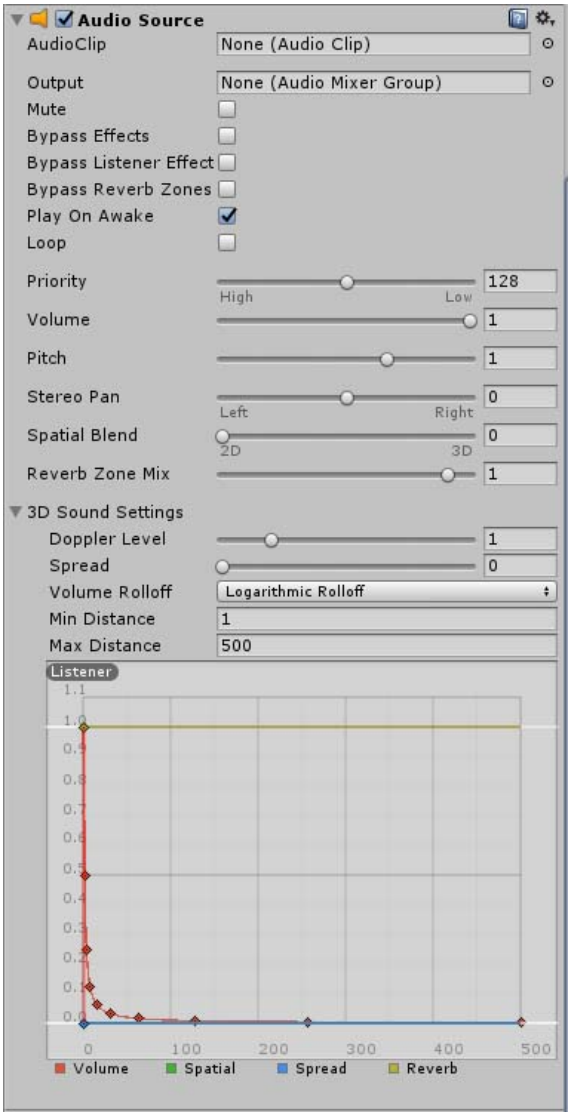
Ένα collider καθορίζει το σχήμα ενός αντικειμένου με σκοπό τις φυσικές συγκρούσεις αυτών. Στην Unity υπάρχουν διάφορα collider components με κάθε ένα να αντιπροσωπεύει ένα διαφορετικό σχήμα και τις ιδιαιτερότητές που έχει. Για αναπαράσταση σε 3D χώρο υπάρχουν τα Box Collider, Capsule Collider, Sphere Collider, Terrain Collider, Wheel Collider, ενώ για 2D υπάρχουν τα Box Collider 2D, Circle Collider 2D, Edge Collider 2D και Polygon Collider 2D.



Σχήμα 5: Box Collider Component

2.3.5 Audio Source

Το Audio Source component χρησιμοποιείται για την αναπαραγωγή AudioClip ήχων. Μπορεί να ρυθμιστεί έτσι ώστε ο ήχος να είναι 2D είτε να αντιπροσωπεύει κάποιον 3D ήχο μέσα στην σκηνή. Ο ήχος όμως μπορεί να ακουστεί μόνο όταν υπάρχει κάποιος Audio Listener ο οποίος θα τον λάβει.



Σχήμα 6: Audio Source Component

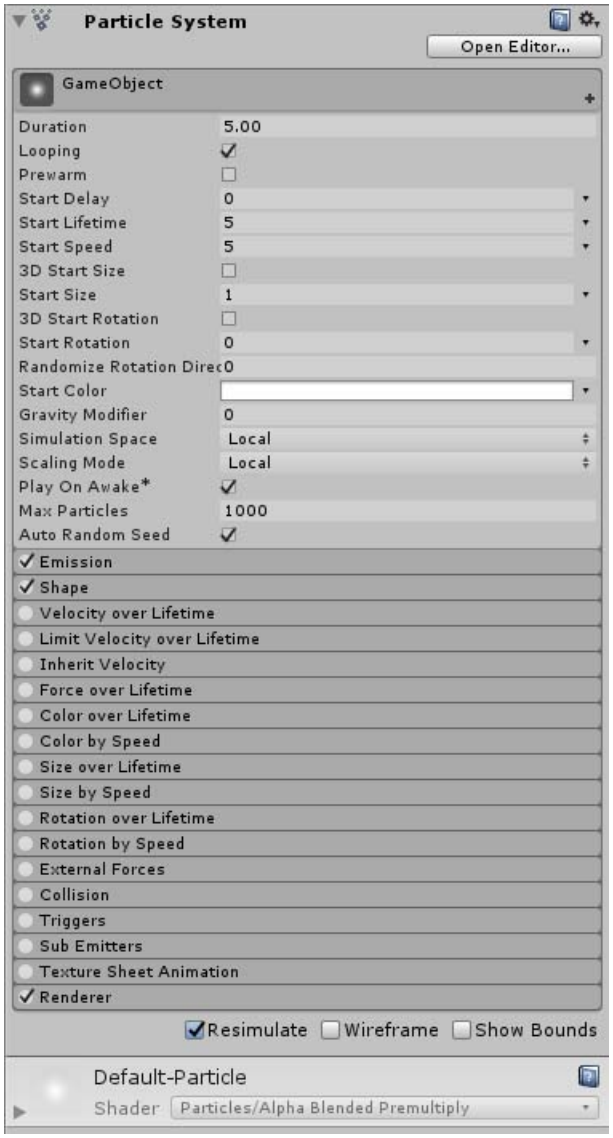
2.3.6 Audio Listener

Σκοπός του Audio Listener component είναι να λαμβάνει ήχους από κάποιο Audio Source και να τον αναπαράγει στα ηχεία του υπολογιστή. Τοποθετείται συνήθως επάνω στην Main Camera.

Σχήμα 7: Audio Listener Component

2.3.7 Particle System

Ένα Particle System χρησιμοποιείται συνήθως για να αναπαραστήσει μη στερεά αντικείμενα, όπως καπνό, σύννεφα, φλόγες, ξόρκια και τα λοιπά. Χρησιμοποιεί διαφορετική προσέγγιση γραφικών από αυτήν των Mesh και Sprite.



Σχήμα 8: Particle System Component

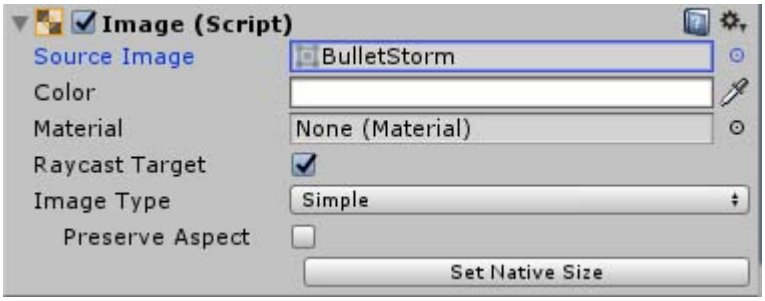
2.3.8 Canvas

Το Canvas component προστίθεται επάνω σε ένα GameObject ώστε να δημιουργηθεί ένας καμβάς. Έχει σχήμα ορθογωνίου παραλληλογράμμου και χρησιμοποιείται σαν μια περιοχή στην οποία μπαίνουν όλα τα UI elements.

Σχήμα 9: Canvas Component

2.3.9 Image

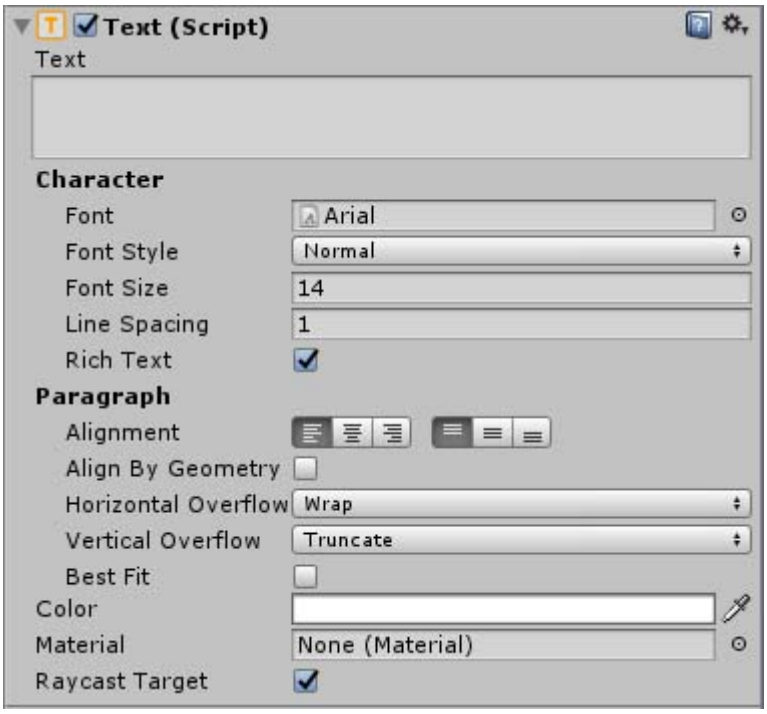
Χρησιμοποιείται για να αναπαραστήσει μία εικόνα της μορφής Sprite στην οθόνη.



Σχήμα 10: Image Component

2.3.10 Text

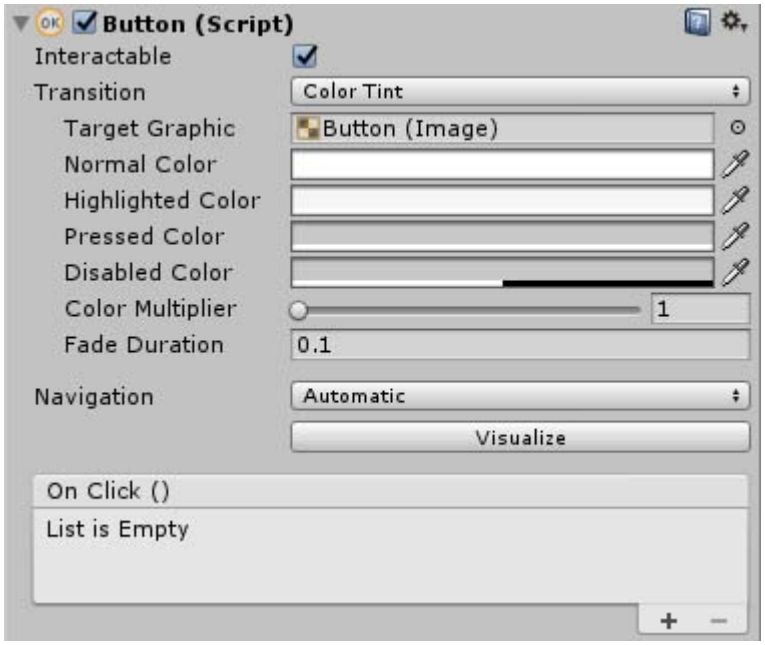
Χρησιμοποιείται για να αναπαραστήσει μία συμβολοσειρά στην οθόνη.



Σχήμα 11: Text Component

2.3.11 Button

Το Button Component δέχεται κάποιο κλικ που κάνει ο χρήστης και αντιδρά κάνοντας μια δράση η οποία έχει οριστεί. Μαζί με αυτό χρησιμοποιείται συνήθως ένα image component για την εμφάνιση του κουμπιού και επίσης μία συμβολοσειρά για την ένδειξη της ενέργειας του.



Σχήμα 12: Button Component

3 Εισαγωγή στο παιχνίδι

Αυτή η ενότητα έχει ως σκοπό να φέρει τον αναγνώστη σε επαφή με τα χαρακτηριστικά και τους διάφορους μηχανισμούς του παιχνιδιού. Θα αναφερθούν στοιχεία του παιχνιδιού, όπως τα spells που έχει κάθε παίκτης, τα είδη των εχθρών, οι δυνατότητές τους καθώς και τα μοντέλα τους.

3.1 Χαρακτήρες παικτών

Υπάρχουν δύο διαφορετικοί χαρακτήρες που μπορεί να έχει ένας παίκτης. Ο βασικός χαρακτήρας, τον οποίο χειρίζεται ο πρώτος παίκτης είναι ο Merlin. Ο δεύτερος παίκτης χειρίζεται τον χαρακτήρα Orc. Στο Σχήμα 13 απεικονίζονται οι δύο χαρακτήρες. Αυτοί έχουν τα ίδια χαρακτηριστικά και τα ίδια spells. Η μόνη διαφορά, η οποία είναι εμφανή βρίσκεται στα μοντέλα. Κάθε παίκτης έχει τέσσερα σταθερά spells: Fireball, Teleport, PlasmaField, BulletStorm και έχει ένα επιπλέον χώρο για άλλα, τα οποία μπορεί να βρει κατά την διάρκεια του παιχνιδιού. Αυτά όμως θα εξηγηθούν περαιτέρω στην υποενότητα Spells που ακολουθεί.

Σχήμα 13: Χαρακτήρες

3.2 Εχθροί

Οι κατηγορίες των εχθρών είναι τρεις. Αυτοί που επιτίθενται από κοντά (meelee), αυτοί που επιτίθενται από μακριά (ranged), και αυτοί που επιτίθενται με spells. Συνολικά υπάρχουν οκτώ διαφορετικοί εχθροί, οι οποίοι παρότι ανήκουν σε διαφορετικές κατηγορίες βασίζονται στο ίδιο script. Αυτό που τους διαφοροποιεί είναι ο τρόπος επίθεσης. Η γενική λογική των εχθρών είναι η εξής: Αρχικά βρίσκονται στο στάδιο της περιπλάνησης στο οποίο βρίσκουν έναν τυχαίο προορισμό στην πίστα και κατευθύνονται προς αυτόν. Όταν φτάσουν στον προορισμό αυτό, τότε βρίσκουν έναν νέο και κατευθύνονται προς αυτόν. Αν όσο περιπλανώνται εντοπίσουν κάποιον παίκτη σε μια προκαθορισμένη απόσταση, τότε μπαίνουν στο στάδιο του κυνηγητού και έχουν κατεύθυνση προς τον παίκτη. Σε περίπτωση που ο παίκτης απομακρυνθεί και είναι σε μεγαλύτερη απόσταση από την προκαθορισμένη του σταδίου του κυνηγητού, τότε ο εχθρός επιστρέφει στο στάδιο της περιπλάνησης. Όταν ο εχθρός φτάσει την προκαθορισμένη απόσταση για επίθεση, τότε επιτίθεται στον παίκτη. Αφού γίνει η επίθεση, ο εχθρός μπαίνει στο στάδιο κυνηγητού μέχρι να περάσει ο χρόνος που χρειάζεται για να μπορεί να επιτεθεί ξανά με βάση την συχνότητα επίθεσης που έχει προκαθοριστεί.

Παρακάτω ακολουθούν ονομαστικά όλοι οι εχθροί μαζί με τις εικόνες τους και την κατηγορία στην οποία ανήκουν.

1. Goblin
   Κατηγορία επίθεσης: meelee.
   Ζωή: 100
   Damage: 11
   Ταχύτητα: 10
   Συχνότητα επίθεσης: κάθε 0.8 δευτερόλεπτα

2. Skeleton
   Κατηγορία επίθεσης: meelee.
   Ζωή: 100
   Damage: 10
   Ταχύτητα: 8
   Συχνότητα επίθεσης: κάθε 1 δευτερόλεπτο

3. Spider
   Κατηγορία επίθεσης: meelee.
   Ζωή: 100
   Damage: 15
   Ταχύτητα: 9
   Συχνότητα επίθεσης: κάθε 1.2 δευτερόλεπτα

4. Golem
   Κατηγορία επίθεσης: meelee.
   Ζωή: 100
   Damage: 15
   Ταχύτητα: 9
   Συχνότητα επίθεσης: κάθε 0.7 δευτερόλεπτα

5. ElfArcher
   Κατηγορία επίθεσης: ranged.
   Ζωή: 70
   Damage: 16
   Ταχύτητα: 8
   Συχνότητα επίθεσης: κάθε 2.3 δευτερόλεπτα

6. LavaCreature
   Κατηγορία επίθεσης: caster.
   Ζωή: 100
   Damage: 11
   Ταχύτητα: 8
   Συχνότητα επίθεσης: κάθε 1.5 δευτερόλεπτα

7. Werewolf
   Κατηγορία επίθεσης: ranged.
   Ζωή: 70
   Damage: 8
   Ταχύτητα: 7
   Συχνότητα επίθεσης: κάθε 0.6 δευτερόλεπτα

8. LavaGolem
   Κατηγορία επίθεσης: caster.
   Ζωή: 100
   Damage: 11
   Ταχύτητα: 6
   Συχνότητα επίθεσης: κάθε 6 δευτερόλεπτα

Σχήμα 14: Goblin, Skeleton, Spider, Golem

Σχήμα 15: ElfArcher, LavaCreature, Werewolf, LavaGolem

3.3 Spells

Υπάρχουν συνολικά οκτώ διαφορετικά spells στο παιχνίδι. Σε αυτά υπάρχουν δύο κατηγορίες με βάση τον τρόπο που γίνονται cast. Οι δύο κατηγορίες είναι: Instant Cast και Normal Cast. Στην πρώτη ο παίκτης χρειάζεται να πατήσει το πλήκτρο που αντιστοιχεί στο κουμπί μία φορά για να γίνει το spell, ενώ στην δεύτερη πατάει μία φορά το πλήκτρο που αντιστοιχεί, έπειτα στοχεύει στην θέση/κατεύθυνση που επιθυμεί και ξαναπατάει το κουμπί ώστε να ολοκληρωθεί το spell.

Ακολουθεί η λίστα με όλα τα spells αρχίζοντας από τα τέσσερα βασικά το οποία κατέχουν οι παίκτες στην αρχή του παιχνιδιού.

1. Fireball
   Περιγραφή: Μπάλα φωτιάς η οποία κινείται σε μία ευθεία και χτυπάει τον πρώτο αντίπαλο με τον οποίο θα συγκρουστεί.
   Τύπος: Normal Cast

2. Teleport
   Περιγραφή: Τηλεμεταφορά του παίκτη στο σημείο που θα επιλέξει.
   Τύπος: Normal Cast

3. Plasma Field
   Περιγραφή: Ασπίδα η οποία περικλείει τον παίκτη και τον προστατεύει από τις από τις περισσότερες μακρινές επιθέσεις. Επίσης απωθεί μακριά όσους εχθρούς βρίσκονται μέσα στην έκτασή της.
   Τύπος: Instant Cast

4. Bulletstorm
   Περιγραφή: Καταιγίδα από σφαίρες οι οποίες πέφτουν κοντά στο σημείο όπου επέλεξε ο παίκτης και τραυματίζουν τους εχθρούς με τους οποίους έρχονται σε επαφή.
   Τύπος: Normal Cast

5. Groundshock
   Περιγραφή: Δυνατό χτύπημα στο έδαφος το οποίο τραυματίζει όποιους εχθρούς έρθουν σε επαφή με αυτό.
   Τύπος: Instant Cast

6. Lavabeam
   Περιγραφή: Ακτίνα λάβας η οποία τραυματίζει περιοδικά όλους τους εχθρούς που είναι σε επαφή με αυτήν.
   Τύπος: Normal Cast

7. Gravity
   Περιγραφή: Πεδίο βαρύτητας το οποίο κινείται σε μία ευθεία, τραυματίζει τους εχθρούς που είναι σε επαφή με αυτό περιοδικά και ταυτόχρονα τους τραβάει προς το κέντρο του.
   Τύπος: Normal Cast

8. Domestication
   Περιγραφή: Κάνει τον κοντινότερο εχθρό σύμμαχο των παικτών και ταυτόχρονα του δίνει επιπλέον ζωή.
   Τύπος: Instant Cast

3.4 Stats

Τα stats είναι στοιχεία που βελτιώνουν μερικές ιδιότητες των παικτών. Η βελτίωση αυτή μπορεί να είναι είτε σταθερά είτε ποσοστιαία με βάση τον υπάρχον βαθμό μιας ιδιότητας του παίκτη. Οι ιδιότητες τις οποίες επηρεάζουν τα stats, είναι οι εξής:

1. Maximum HP - Αντιστοιχεί στο μέγιστο όριο ζωής του παίκτη.

2. Regeneration - Αντιστοιχεί στην ζωή που αναπληρώνεται στον παίκτη με το πέρασμα του χρόνου.

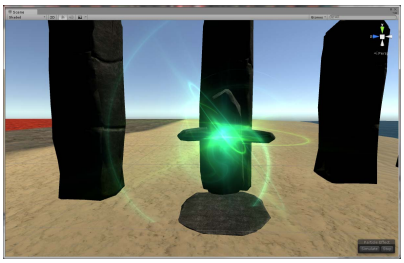
3. Damage - Αντιστοιχεί στο ποσό της ζημιάς που κάνει ο παίκτης στους εχθρούς.

4. Speed - Αντιστοιχεί στην ταχύτητα του παίκτη.

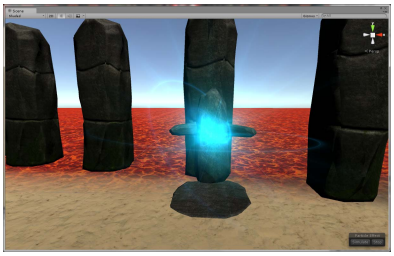
5. Cooldown - Αντιστοιχεί στον χρόνο που χρειάζεται ένα spell για να ξαναχρησιμοποιηθεί από τον παίκτη.

3.5 Buffstands

Τα buffstands είναι πέτρινα πλαίσια που υπάρχουν σε μερικά σημεία της πίστας και σκοπό έχουν να βοηθήσουν τους παίκτες, δίνοντάς τους είτε κάποιο stat, είτε πλήρη αναπλήρωση ζωής, είτε κάποιο ξεχωριστό spell το οποίο δεν ανήκει στα βασικά spells του παίκτη. Σε κάθε αρχή ενός επιπέδου, λίγο πριν εμφανιστούν οι εχθροί, κάθε buffstand έχει πιθανότητα να δώσει μία από τις τρεις αναφερόμενες επιλογές ή να μην δώσει καμία. Για να πάρει ο παίκτης οτιδήποτε του προσφέρει ένα buffstand, πρέπει να πάει επάνω σε μια φωτεινή λάμψη που βρίσκεται στο buffstand, της οποίας το χρώμα αντιπροσωπεύει και το είδος της προσφερόμενης βοήθειας. Η λάμψη για την πλήρη αναπλήρωση ζωής είναι πράσινη, για τα stat είναι μπλε και για τα ξεχωριστά spells είναι χρυσή, όπως φαίνεται στο παρακάτω σχήμα.



(α΄) Buffstand - HP Restore



(β΄) Buffstand - Stat

(γ΄) Buffstand - Special Spell

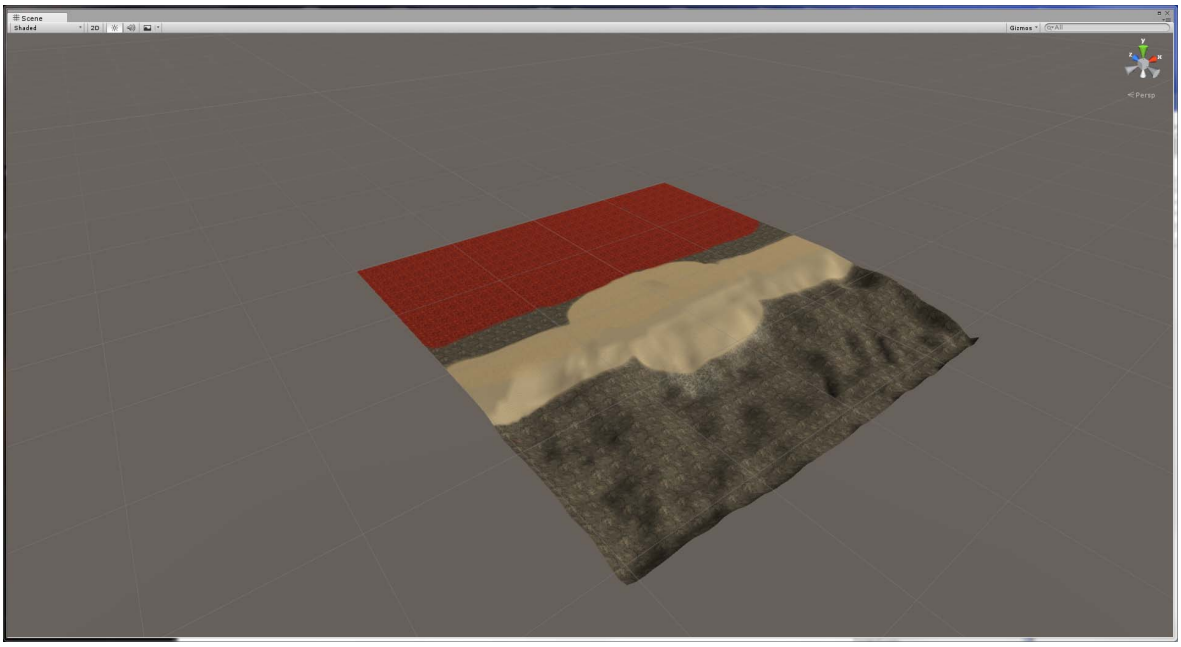
Σχήμα 16: Buffstands

4 Ανάπτυξη Παιχνιδιού

Αυτή η ενότητα έχει ως σκοπό να παρουσιάσει και να εξηγήσει σε γενικό πλαίσιο τα σημαντικότερα στοιχεία της διαδικασίας δημιουργίας του παιχνιδιού.

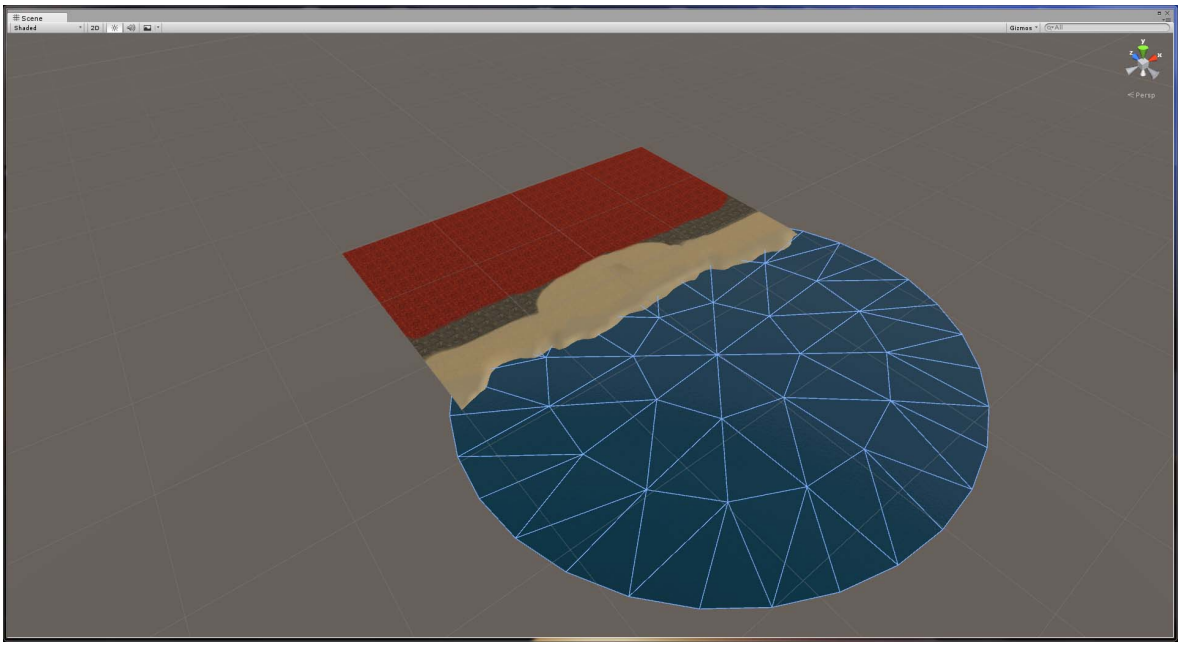
4.1 Δημιουργία Πίστας

Για την δημιουργία της πίστας χρησιμοποιήθηκε αρχικά το αντικείμενο Terrain της Unity το οποίο βρίσκεται στην κατηγορία των 3D Object. Έπειτα, με την δυνατότητα Raise / Lower Terrain τροποποιήθηκε το ανάγλυφο της πίστας σύμφωνα με τις απαιτήσεις του σχεδιασμού για το συγκεκριμένο παιχνίδι. Όλο το επίπεδο της πίστας είναι ίσιο, εκτός από ένα μέρος στο οποίο το terrain είναι χαμηλωμένο, έτσι ώστε να τοποθετηθεί το νερό. Επίσης, με την βοήθεια του Paint Texture ζωγραφίστηκε η επιφάνεια της πίστας, χρησιμοποιώντας τα εισαγόμενα textures (Σχήμα 19).



Σχήμα 17: Terrain

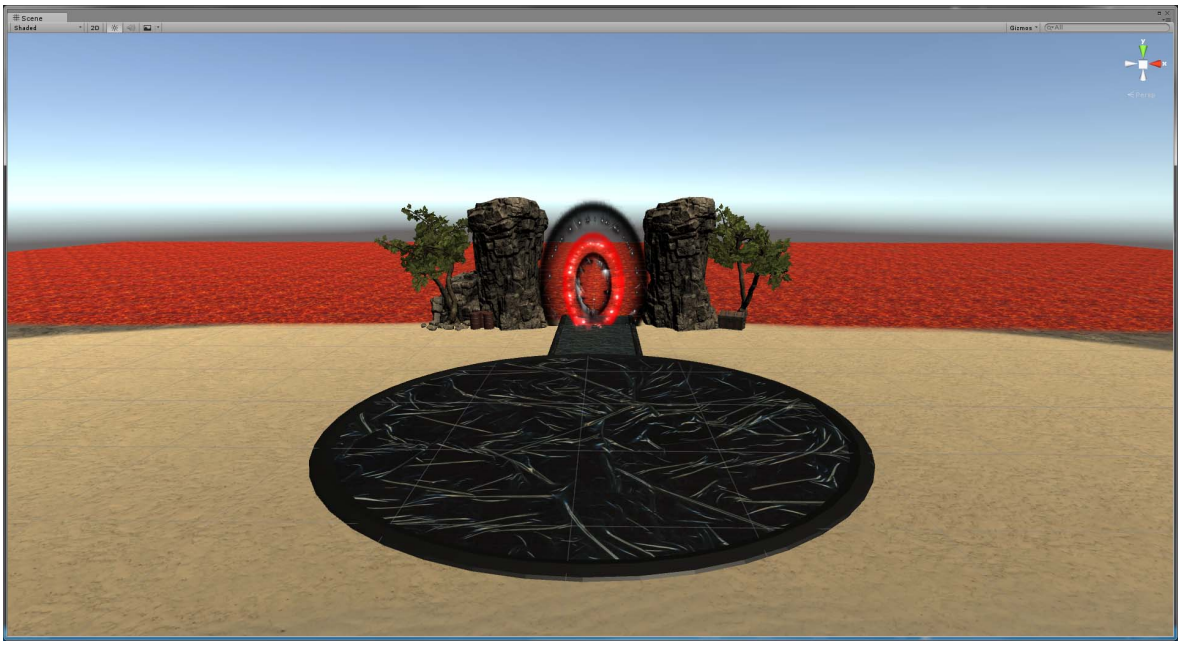
Ως νερό χρησιμοποιήθηκε το αντικείμενο Water που υπάρχει στο πακέτο των Standard Assets της Unity, το οποίο σαν μοντέλο έχει σχήμα κύκλου. Έτσι, τοποθετήθηκε επάνω από την χαμηλωμένη επιφάνεια της πίστας που δημιουργήθηκε προηγουμένως, δημιουργώντας την αίσθηση της ακτής (Σχήμα 18).



Σχήμα 18: Εισαγωγή νερού

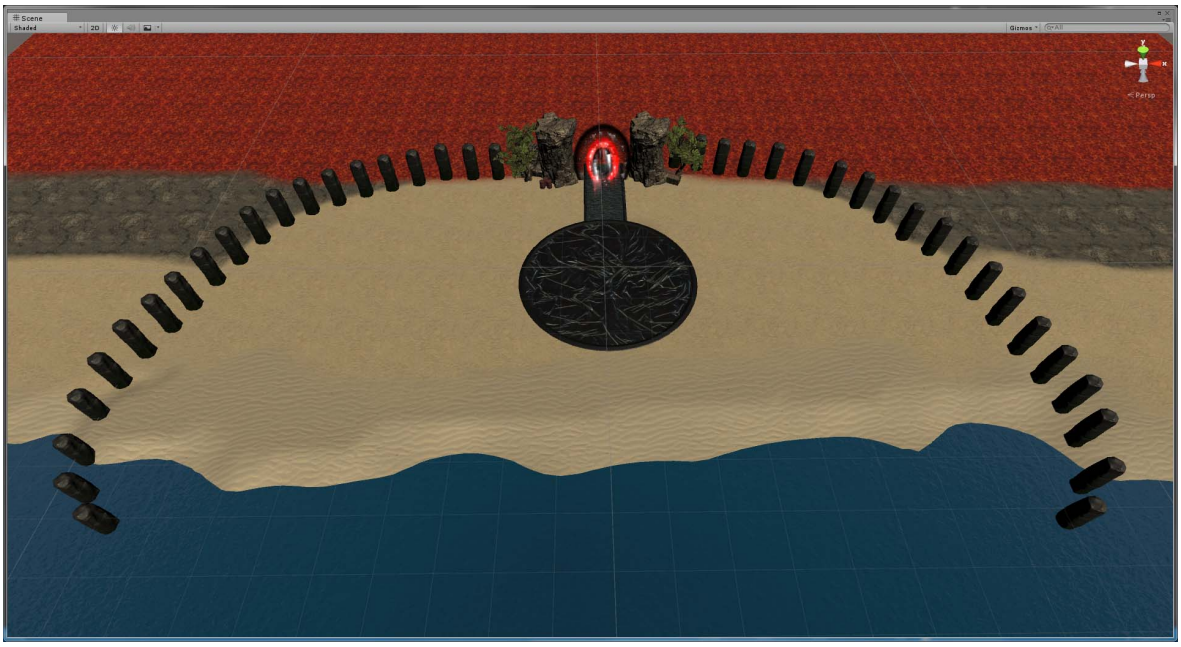
Σχήμα 19: Paint Texture

Μετά από τον σχεδιασμό του terrain ακολουθεί η εισαγωγή και τοποθέτηση αντικειμένων, δίνοντας έτσι περισσότερη λεπτομέρεια και διάκριση στο περιβάλλον της πίστας. Επάνω στην πίστα προστέθηκε μία πύλη, αποτελούμενη από δύο βράχους. Ανάμεσα στους βράχους τοποθετήθηκε επίσης ένας συνδυασμός από particles, ώστε να βελτιωθεί το τελικό αποτέλεσμα και να δείχνει η πύλη πιο λειτουργική. Δίπλα από τους βράχους εισήχθηκαν επιπλέον διακοσμητικά αντικείμενα τα οποία φαίνονται στο Σχήμα 20. Όλα τα παραπάνω αντικείμενα που χρησιμοποιήθηκαν βρίσκονται στο Asset Store της Unity και μπορούν να χρησιμοποιηθούν δωρεάν.

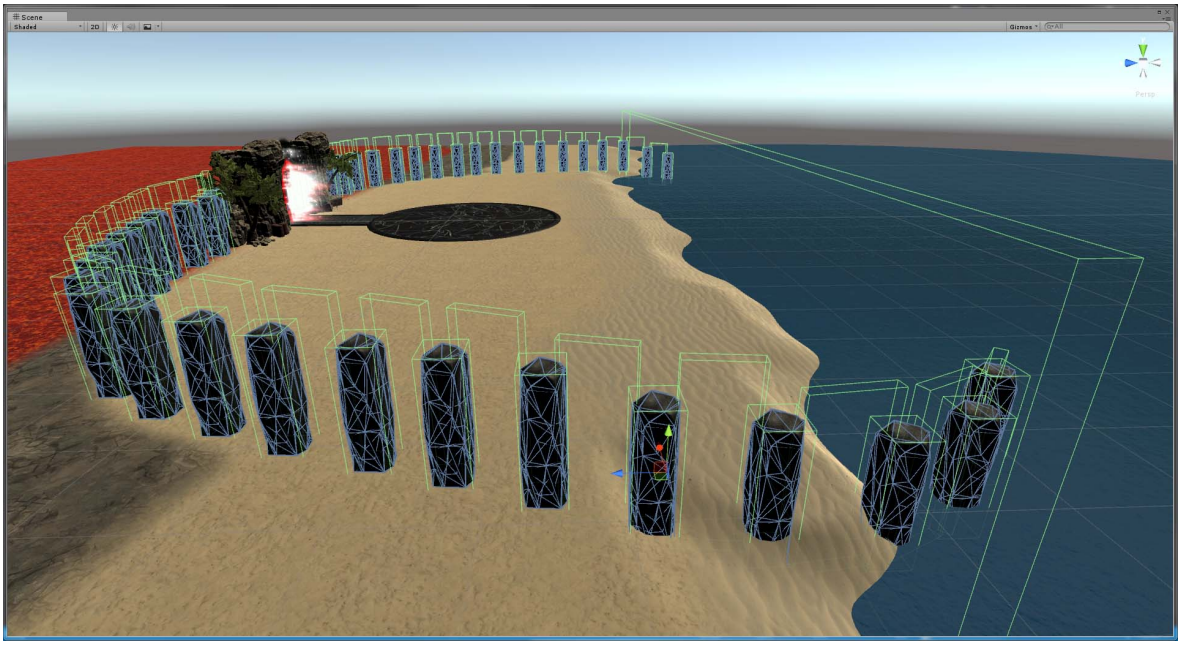


Σχήμα 20: Πύλη

Τέλος, φτιάχτηκε ένα μπλόκο από βράχους το οποίο περικλείει την πίστα και τον χώρο στον οποίο θα βρίσκονται οι παίκτες και οι εχθροί (Σχήμα 21). Κάθε βράχος ξεχωριστά έχει τον δικό του collider ο οποίος αντιστοιχεί στο σχήμα και τον όγκο του. Επειδή υπάρχουν διάκενα μεταξύ των βράχων, τοποθετήθηκαν επιπλέον colliders ανάμεσα από κάθε βράχο, έτσι ώστε το περίγραμμα να είναι αδιαπέραστο. Επιπλέον προστέθηκε άλλο ένα collider εκεί που τελειώνει το περίφραγμα των βράχων και αρχίζει η θάλασσα (Σχήμα 22).



Σχήμα 21: Μπλόκο βράχων

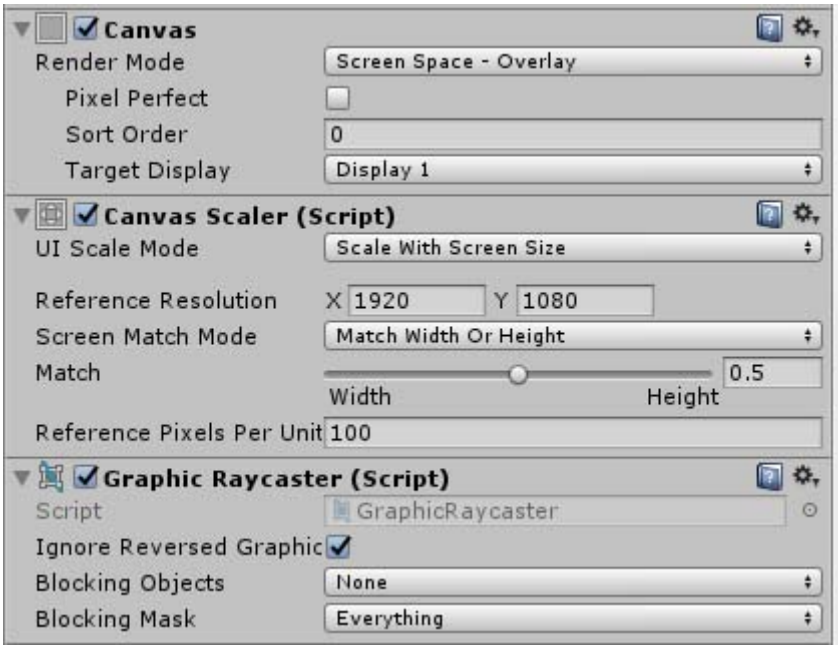


Σχήμα 22: Τοίχος από colliders

4.2 Δημιουργία αρχικού Μενού

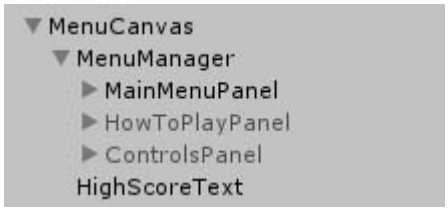
Σχήμα 23: Main Menu

Για το αρχικό μενού,σ έγινε εισαγωγή ενός αντικειμένου Canvas που διαθέτει η Unity επιλέγοντας GameObject>UI>Canvas. Το render mode είναι στην επιλογή Screen Space - Overlay, το UI Scale Mode είναι Scale With Screen Size και επίσης η αναλογία width-height είναι ίση, δηλαδή 0.5, όπως φαίνεται στο Σχήμα 24. Ως παιδί του canvas μπήκε ένα Empty GameObject με όνομα MenuManager μέσα στο οποίο είναι οτιδήποτε αφορά το μενού. Στο MenuManager προστέθηκε το script MenuManager το οποίο διαχειρίζεται την συμπεριφορά του μενού και περιέχει χρήσιμες μεθόδους που αντιπροσωπεύουν τις δράσεις των κουμπιών.



Σχήμα 24: Menu Canvas

Μέσα στο MenuManager βρίσκονται τα MainMenuPanel, HowToPlayPanel, ControlsPanel και το HighScoreText το οποίο περιέχει μέσω κειμένου το μέγιστο σκορ στο οποίο έχει φτάσει το παιχνίδι. Τα περιεχόμενα αυτά φαίνονται μέσα από το παράθυρο Hierarchy στο Σχήμα 25. Στο MainMenuPanel υπάρχει η δυνατότητα εκκίνησης του παιχνιδιού, η επιλογή του αριθμού των παικτών, η έξοδος από το παιχνίδι καθώς και η πρόσβαση στα πάνελ HowToPlay και Controls. Το ControlsPanel έχει για κάθε παίκτη πεδία, τα οποία περιέχουν δύο Texts. Το ένα αναφέρει την ενέργεια του πλήκτρου και το άλλο το σύμβολο του πλήκτρου. Υπάρχει επίσης το κουμπί ”Back”, το οποίο γυρνάει στο κεντρικό μενού, χρησιμοποιώντας μέθοδο του MenuManager. Τέλος, το HowToPlayPanel χρησιμοποιεί το Scroll Rect component, το οποίο προσθέτει μία κινούμενη μπάρα, έτσι ώστε το περιεχόμενο μέσα σε αυτό να μπορεί να μετακινείται ανάλογα με αυτήν. Για να φαίνεται μόνο ένα μέρος του κειμένου και να αποκρύβεται το υπόλοιπο, μπήκε επίσης ένα Mask component. Όπως και στο ControlsPanel, μπήκε και εδώ το κουμπί ”Back” το οποίο επιστρέφει στο κεντρικό μενού. Η ιεραρχία του HowToPlayPanel φαίνεται στο Σχήμα 26.

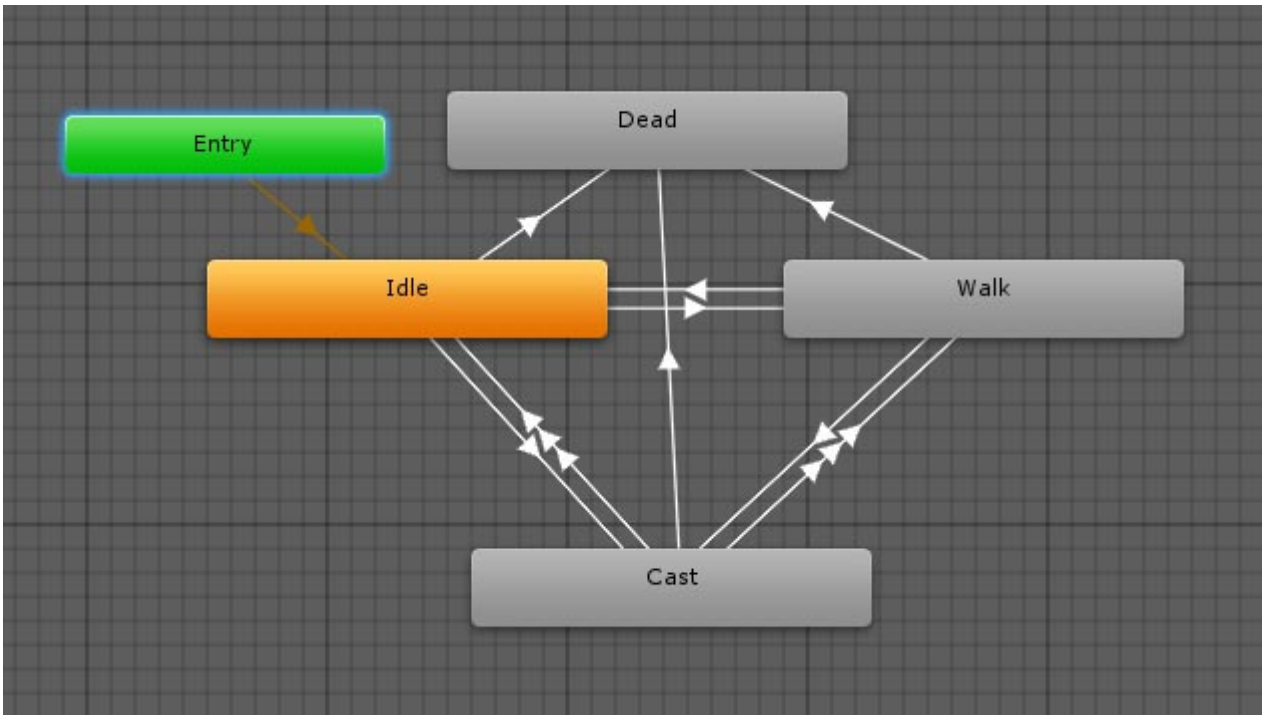


Σχήμα 25: Menu Hierarchy

Σχήμα 26: HowToPlayPanel’s Hierarchy

4.3 Δημιουργία Animator Controller του παίκτη

Για την σωστή διαχείριση και εναλλαγή των animation του παίκτη κατά την διάρκεια του παιχνιδιού, χρησιμοποιήθηκε το Animator Controller της Unity. Αυτό μας δίνει τη δυνατότητα να ορίσουμε διάφορα στάδια τα οποία μπορεί να βρίσκεται ο παίκτης (π.χ. το περπάτημα, τρέξιμο, επίθεση), καθώς και τις μεταβάσεις μεταξύ τους. Οι μεταβάσεις αυτές γίνονται βάση διαφόρων μεταβλητών που μπορούμε να ορίσουμε. Οπότε κατά την αλλαγή τιμής μιας τέτοιας μεταβλητής που γίνεται μέσω κάποιου script, μπορούμε να προκαλέσουμε μετάβαση σε κάποιο άλλο στάδιο. Στο σχήμα 27 φαίνονται τα στάδια και οι πιθανές μεταβάσεις αυτών. Τα βέλη των μεταβάσεων στο παραπάνω σχήμα συμβολίζουν την κατεύθυνση της μετάβασης. Ο Animator Controller μπαίνει στο αντίστοιχο πεδίο που έχει το Animator Component της Unity που βρίσκεται επάνω στον παίκτη.



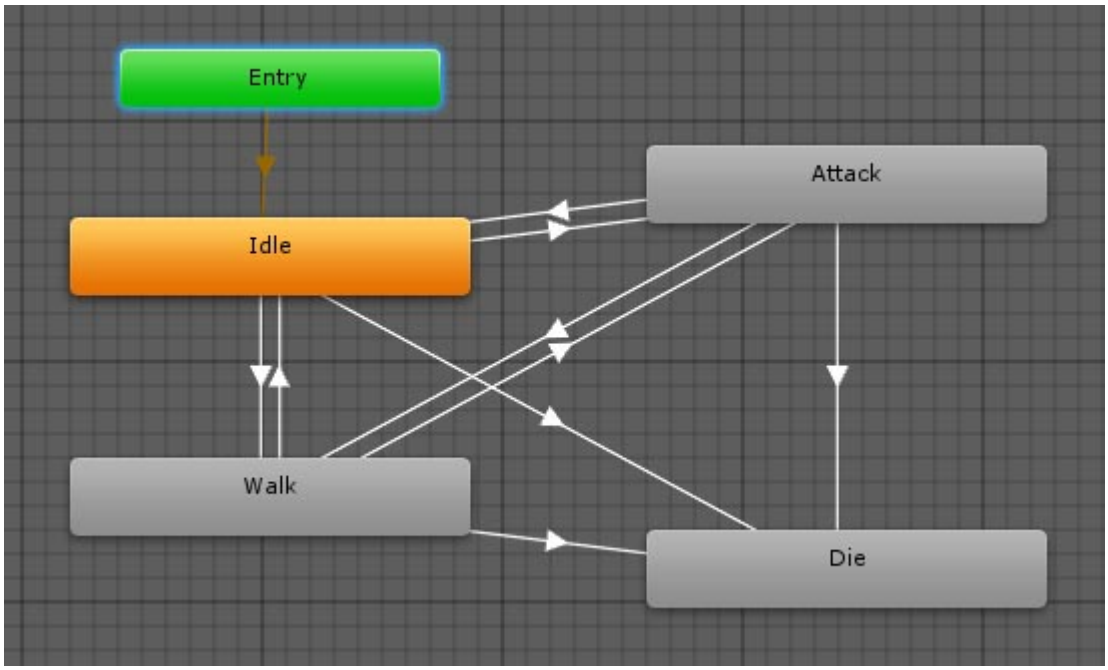
Σχήμα 27: Player Animator Controller

Το στάδιο Idle είναι όταν ο παίκτης είναι ακίνητος και είναι το αρχικό στάδιο. Το στάδιο Walk είναι όταν ο παίκτης περπατάει, το Cast όταν εκτοξεύει κάποιο spell και το στάδιο Dead είναι όταν πεθαίνει. Όπως φαίνεται και στο Σχήμα 28 έχουμε δηλώσει τις εξής μεταβλητές: isMoving, isCasting, OnCastCancel και isDead. Όλες αυτές οι μεταβλητές είναι τύπου boolean, εκτός την OnCastCancel η οποία είναι Trigger. Σε όποιο στάδιο και αν βρίσκεται ο παίκτης, όταν η μεταβλητή isDead γίνει ίση με true, τότε γίνεται άμεσα μετάβαση στο στάδιο Dead, όπου παίζει το αντίστοιχο animation. Αν το στάδιο είναι Idle ή Walk και η μεταβλητή isCasting γίνει ίση με true, τότε γίνεται μετάβαση στο στάδιο Cast. Τότε παίζει το αντίστοιχο animation. Υπάρχουν δύο περιπτώσεις γυρισμού από αυτό το στάδιο προς το Idle ή το Walk. Εξού και η ένδειξη με τα τρία βέλη. Η πρώτη είναι να τελειώσει η εκτόξευση του spell και να γίνει η isCasting ίση με false και η δεύτερη να ακυρωθεί η εκτόξευση του spell, ενεργοποιώντας τον Trigger OnCastCancel. Το σε ποιο στάδιο θα επιστρέψει (Idle ή Walk) εξαρτάται από την τιμή της isMoving.

Σχήμα 28: Player Animator Controller variables

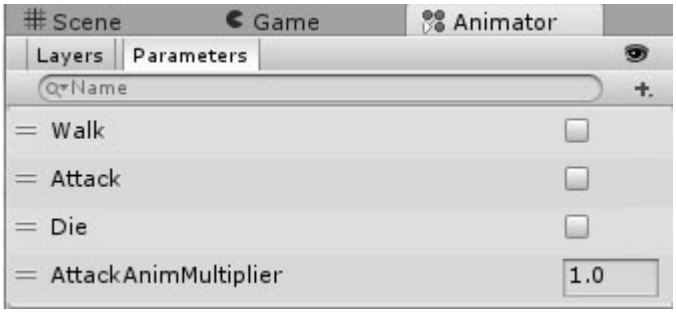
4.4 Δημιουργία Animator Controller των εχθρών

Όπως κάναμε για τον παίκτη, έτσι και για τους εχθρούς κάναμε την αντίστοιχη διαδικασία. Όλοι οι εχθροί έχουν την ίδια λογική στο Animator Controller τους. Τα στάδια και οι μεταβάσεις αυτής φαίνονται στο σχήμα 29.

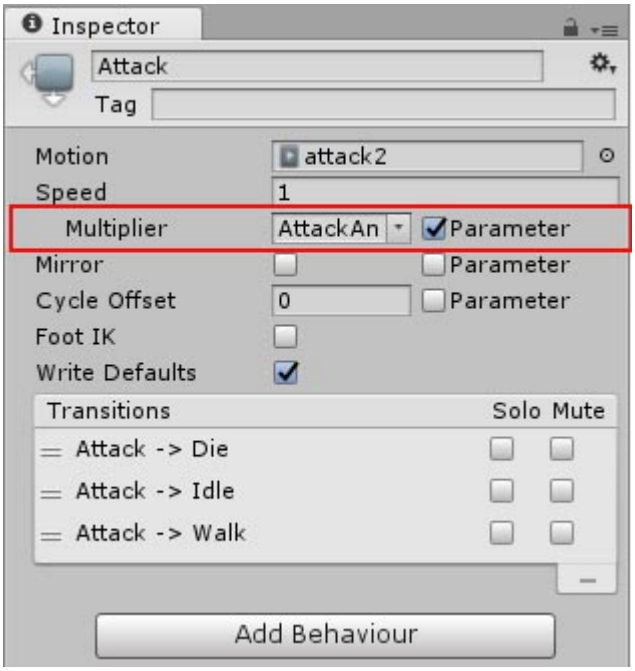


Σχήμα 29: Enemy Animator Controller

Τα στάδια είναι παρόμοια με αυτά του παίκτη με την διαφορά ότι το στάδιο επίθεσης λέγεται Attack. Επίσης, δεν υπάρχει η ιδιότητα της ακύρωσης της επίθεσης ή της εκτόξευσης κάποιου spell, άρα η μετάβαση από το στάδιο Attack στα Idle και Walk δεν είναι διπλή. Εξαιτίας αυτού, δεν υπάρχει και η μεταβλητή OnCastCancel. Εδώ, όλες οι μεταβλητές είναι της μορφής boolean, εκτός την AttackAnimMultiplier η οποία είναι της μορφής float. Αυτή χρησιμοποιείται με σκοπό να μεταβάλει την ταχύτητα με την οποία γίνεται το animation της επίθεσης, ανάλογα με πόσο γρήγορη είναι η επίθεση του παίκτη και η τιμή της ελέγχεται από το MobBehaviour Script. Οι μεταβλητές φαίνονται στο Σχήμα 30, ενώ η ρύθμιση για την σχέση της ταχύτητας του animation της επίθεσης στο στάδιο Attack φαίνεται στο σχήμα 31.



Σχήμα 30: Enemy Animator Controller variables



Σχήμα 31: Enemy Attack State

4.5 Scripts

Σε αυτήν την υποενότητα εξηγούνται τα σημαντικότερα scripts που υπάρχουν στο παιχνίδι, τα οποία συντελούν στην λειτουργικότητα και αλληλεπίδραση του παιχνιδιού.

4.5.1 RigidbodyWrapper

Το RigidBodyWrapper script έχει φτιαχτεί με σκοπό να μεταλλάσσει, να ρυθμίζει και να αυτοματοποιεί διάφορες λειτουργίες που βασίζονται άμεσα στο RigidBody component/script της Unity, σύμφωνα με τις ανάγκες του παιχνιδιού.

Η λογική στην οποία αποσκοπεί το συγκεκριμένο script είναι η κίνηση κάθε αντικειμένου που το έχει ως component, να αντιστοιχεί στο άθροισμα δύο διανυσμάτων. Το ένα είναι ένα σταθερό σε μέτρο διάνυσμα το οποίο αντιπροσωπεύει την ταχύτητα του αντικειμένου (π.χ. ταχύτητα βηματισμού του παίκτη) ενώ το άλλο είναι ένα διάνυσμα που αντιπροσωπεύει το σύνολο των εξωτερικών δυνάμεων που έχουν επηρεάσει το αντικείμενο με το RigidbodyWrapper (π.χ. knockback από κάποιο spell ή χτύπημα). Το μέτρο του δεύτερου

αυτού διανύσματος μικραίνει με τον χρόνο, λόγο της ορισμένης τριβής που έχει προγραμματιστεί να ασκείται ενάντια σε αυτό μέσα σε κάθε Update, φτάνοντας τελικά σε μέτρο μηδέν. Παράλληλα με την κίνηση, το script φροντίζει επίσης την περιστροφή του αντικειμένου, ανάλογα με την κατεύθυνση ή τον προορισμό του.

Σε αυτό το script εμπεριέχονται τρεις διαφορετικές λειτουργίες, με βάση τον τρόπο με τον οποίο κινείται το αντικείμενο. Η μία επιλογή είναι να κινείται με βάση μία κατεύθυνση, η άλλη να κινείται με βάση έναν προορισμό και η τελευταία να κινείται μόνο βάση των εξωτερικών δυνάμεων. Στην πρώτη περίπτωση το αντικείμενο θα κινηθεί εφόσον υπάρχει επιλεγόμενη κατεύθυνση, είτε εφόσον υπάρχουν εξωτερικές δυνάμεις. Αυτή η περίπτωση χρησιμοποιείται για τον χειρισμό των παικτών μέσα στο παιχνίδι. Οι εχθροί λειτουργούν βασισμένοι στην δεύτερη περίπτωση, στην οποία υπάρχει ένας συγκεκριμένος προορισμός στην πίστα τον οποίο ακολουθούν. Έτσι λοιπόν όσο υπάρχει προορισμός, το script φροντίζει να βρει και να κινήσει το αντικείμενο προς την σωστή κατεύθυνση, μέχρι να φτάσει στο σημείο προορισμού. Αν δεν έχει ορισμένη κατεύθυνση, η κίνησή του επηρεάζεται μόνο από τις εξωτερικές δυνάμεις, όπως και στην πρώτη περίπτωση. Η τρίτη περίπτωση χρησιμοποιείται για διάφορα αντικείμενα τα οποία δεν έχουν αυτόνομη κίνηση και η κίνησή τους βασίζεται μόνο στις εξωτερικές δυνάμεις που τους ασκούνται.

4.5.2 PlayerController

Το PlayerController script διαχειρίζεται το input του παίκτη και με βάση αυτό δρα ανάλογα. Οι δράσεις οι οποίες μπορεί να εκτελεστούν είναι η κίνηση του παίκτη, καθώς και η έναρξη ή ακύρωση ενός spell.

Το script αυτό σχετίζεται άμεσα με άλλα scripts τα οποία είναι προστιθέμενα ως components σε κάθε παίκτη και έχουν σκοπό την απλοποίηση και υποδιαίρεση του προβλήματος της συμπεριφοράς του παίκτη. Αυτά είναι τα: RigidbodyWrapper, PlayerSpell, PlayerCooldown, PlayerAnimation, PlayerAudio, Stats, και LivingEntity, τα οποία εκτός του RigidbodyWrapper που ήδη αναφέρθηκε θα εξηγηθούν στις επόμενες υποενότητες.

4.5.3 PlayerSpell

Αυτό το script αναλαμβάνει να υλοποιήσει το σύστημα των spell του παίκτη. Αποθηκεύει μία λίστα με τα spell του παίκτη και κρατάει επίσης πληροφορία για το ποιο spell είναι επιλεγμένο. Διαχειρίζεται τη δημιουργία ενός spell, καθώς και την συμπεριφορά του παίκτη κατά την διάρκεια αυτής. Αυτή η συμπεριφορά περιλαμβάνει την εκτέλεση του cast animation, την ακινητοποίηση του παίκτη όσο γίνεται η διαδικασία εκτέλεσης και περιστροφή του παίκτη προς την κατεύθυνση του spell. Αυτή η κατεύθυνση αποθηκεύεται επίσης μέσα στο παρόν script και η τιμή της ορίζεται από το PlayerController με βάση την είσοδο του παίκτη.

4.5.4 PlayerCooldown

Το PlayerCooldown script διαχειρίζεται τα cooldown των spell σε κάθε παίκτη, όπως συμπεραίνεται και από την ονομασία. Κρατάει σε μία δομή Dictionary κάθε spell του παίκτη ως κλειδί και ως τιμή τον χρόνο που χρειάζεται το spell για να χρησιμοποιηθεί ξανά. Σε κάθε εκτέλεση ενός spell αρχίζει την αντίστροφη μέτρηση από τον χρόνο τον οποίο αντιστοιχεί στο cooldown του spell την συγκεκριμένη στιγμή. Ο χρόνος αυτός μειώνεται σε κάθε Update με βάση τον χρόνο που έχει περάσει. Όταν τελειώσει η αντίστροφη μέτρηση και ο χρόνος γίνει ίσος με μηδέν, τότε το spell είναι έτοιμο για χρήση.

4.5.5 PlayerAnimation

Από αυτό το script γίνονται όλες οι αλλαγές όσον αφορά τις μεταβλητές του Animator Controller του παίκτη. Τα animations και η ροή από κατάσταση σε κατάσταση έχουν οριστεί μέσα σε αυτόν και με βάση την αλλαγή στις τιμές των μεταβλητών που υπάρχουν, ο Animator Controller συμπεριφέρεται ανάλογα. Σκοπός του script είναι να απλοποιήσει την διαδικασία αυτή και να μειώσει την επανάληψη κώδικα, αφού

στις μεθόδους του ρυθμίζει όλες τις απαραίτητες αλλαγές που πρέπει να γίνουν σε κάθε αλλαγή της τιμής σε μια μεταβλητή του Animator Controller.

4.5.6 PlayerAudio

Η δουλειά του PlayerAudio είναι αρκετά απλή. Κρατάει ένα reference του audio component που έχει το αντικείμενο του παίκτη, και με την χρήση αυτού παίζει τον ήχο που θα δοθεί για αναπαραγωγή.

4.5.7 Stats

Το Stats script διαχειρίζεται, υπολογίζει και εφαρμόζει τα stats που τυχαίνει να έχει κάποιος παίκτης. Κρατάει μία δομή Dictionary με κλειδί ένα αντικείμενο stat και ως τιμή τον χρόνο ζωής του επάνω στον παίκτη. Κρατάει επίσης τις αρχικές τιμές των ιδιοτήτων του παίκτη, που με βάση αυτές και σε συνδυασμό με τα υπάρχοντα stat που έχει ο παίκτης, υπολογίζει και αποθηκεύει τις τελικές τιμές κάθε ιδιότητας. Οι μέθοδοί του έχουν να κάνουν με την προσθήκη και αφαίρεση ενός stat, τον υπολογισμό των τελικών τιμών των ιδιοτήτων και την ενημέρωση του εναπομείναντα χρόνου ζωής του κάθε stat.

4.5.8 LivingEntity

Το LivingEntity αντιπροσωπεύει κάθε αντικείμενο που έχει την έννοια του ζωντανού στο παιχνίδι. Κρατάει την ζωή που έχει κάθε αντικείμενο, την μέγιστη τιμή ζωής καθώς και την τιμή του regeneration. Με βάση την τιμή της ζωής, γνωρίζει επίσης αν το αντικείμενο είναι ακόμη ζωντανό ή όχι. Οι μέθοδοι του script έχουν να κάνουν κυρίως με την αυξομείωση της ζωής και εφαρμογή της αναπλήρωσης της ζωής που προσφέρει το regeneration.

4.5.9 Drop

Χρησιμοποιείται σε εχθρούς, όταν θέλουμε κατά τον θάνατό τους να ρίξουν κάποιο βοηθητικό αντικείμενο. Κρατάει μία λίστα με τα αντικείμενα από τα οποία θέλουμε να ρίξει κάποιο από αυτά και επίσης την τιμή της πιθανότητα του να ρίξει κάτι ή όχι.

4.5.10 Spell

Script το οποίο κληρονομούν όλα τα spell του παιχνιδιού. Αποθηκεύει μία reference με την οντότητα στην οποία ανήκει το spell.

4.5.11 MobBehaviour

Το MobBehaviour script αναλαμβάνει τη συμπεριφορά που έχει κάθε εχθρός στο παιχνίδι. Δηλαδή, το πως να κινείται μέσα στην πίστα, το πότε να κυνηγάει τον παίκτη και πότε να επιτίθεται σε αυτόν. Επίσης κρατάει μεταβλητές οι οποίες αφορούν χαρακτηριστικά του εχθρού, όπως η δύναμη της επίθεσης και η συχνότητα επίθεσης. Επιπλέον δυνατότητες που εμπεριέχει το script σχετικά με την συμπεριφορά ενός εχθρού είναι ο εντοπισμός εμποδίων στην πορεία του και έπειτα η αλλαγή πορείας του σε μία νέα. Αυτό το καταφέρνει με την βοήθεια του Raycast που προσφέρει η Unity.

Πιο συγκεκριμένα, το script εκπέμπει συνολικά τρία rays. Ένα μπροστά από κάθε εχθρό και άλλα δύο, με καθένα από αυτά να έχει 30 μοίρες διαφορά από το μπροστινό. Το ένα δηλαδή +30 μοίρες και το άλλο -30 μοίρες σε σχέση με το πρώτο. Τα rays αυτά έχουν συγκεκριμένο μήκος. Το μπροστινό ray έχει σκοπό να εντοπίσει κάποιο τοίχος που βρίσκεται ακριβώς μπροστά από τον εχθρό, ενώ τα άλλα δύο χρησιμοποιούνται

για να ενισχύσουν τον εντοπισμό εμποδίων, σε περίπτωση που ένα τέρας κινείται λοξά προς ένα εμπόδιο κι έτσι το μπροστινό ray δεν είναι ικανό να το εντοπίσει.

Όσον αφορά τον τύπο της επίθεσης κάθε εχθρού, το script υποστηρίζει τις τρεις κατηγορίες: meelee, ranged, caster. Για τον meelee τύπο η δημιουργία της επίθεσης είναι η ακόλουθη. Κάθε meelee εχθρός έχει έναν collider τοποθετημένο στο άκρο του εχθρού που αντιπροσωπεύει το όπλο του, ή το σημείο με το οποίο τραυματίζει τον παίκτη. Αυτός ο collider είναι αρχικά απενεργοποιημένος. Όταν λοιπόν οι συνθήκες επιτρέψουν την επίθεση, ο εχθρός μπαίνει στο στάδιο της επίθεσης και εκεί αλλάζει την boolean μεταβλητή του Animation Controller που αντιστοιχεί στην επίθεση σε αληθή. Τότε αρχίζει το animation της επίθεσης. Ο collider δεν ενεργοποιείται αμέσως, αφού το χτύπημα του εχθρού δεν ταυτίζεται με την αρχή του animation, αλλά σε ένα συγκεκριμένο σημείο αυτού. Το χρονικό αυτό σημείο βρέθηκε παρατηρώντας το animation του κάθε εχθρού και με την βοήθεια του Animation view στον Inspector, κρατάμε σε τι ποσοστό χρόνου του animation αντιστοιχεί το σημείο αυτό. Το script τότε ξέρει σε πόσο χρόνο, αφού αρχίσει η επίθεση να ενεργοποιήσει τον collider, αφού έχει στις μεταβλητές του και το AnimationClip και το χρονικό σημείο που αναφέρθηκε. Στο τέλος της επίθεσης ή όταν ο εχθρός χτυπήσει κάποιον παίκτη, ο collider απενεργοποιείται και πάλι.

Σχετικά με τον ranged τύπο, το script δεν κρατάει κάποιον collider, αλλά το Transform ενός Empty GameObject που έχει τοποθετηθεί στον εχθρό, σε σημείο που αντιπροσωπεύει το πέταγμα ενός αντικειμένου. Σε αυτήν την περίπτωση, κρατιέται πάλι το χρονικό σημείο που αντιστοιχεί στο πέταγμα του αντικειμένου στο animation. Έτσι, όταν φτάσει αυτό το χρονικό σημείο το script αναλαμβάνει να δημιουργήσει το αντικείμενο που είναι να πετάξει ο εχθρός(π.χ. βέλος, μαχαίρι). Αυτό γίνεται με την βοήθεια του Throw script το οποίο θα αναφερθεί σε επόμενη υποενότητα.

Ο τύπος caster είναι όμοιος με τον τύπο ranged, με διαφορά ότι κατά την επίθεση δημιουργείται ένα spell και όχι ένα πετούμενο αντικείμενο. Άρα η χρονική στιγμή που κρατάει το script αντιπροσωπεύει την στιγμή στην οποία θα εκτελεστεί το spell. Η δημιουργία αυτού του spell γίνεται με τη βοήθεια του Cast script που θα εξηγηθεί σε υποενότητα που ακολουθεί.

Άλλη δυνατότητα που προσφέρει το script στους εχθρούς, είναι η δυνατότητα πρόβλεψης του σημείου της επίθεσης με βάση την θέση και ταχύτητα που έχει ένας παίκτης, ο οποίος είναι στόχος του εχθρού. Αυτό επιτυγχάνεται παίρνοντας το σημείο της θέσης του παίκτη/στόχου, προσθέτοντας σε αυτήν την ταχύτητα του παίκτη, πολλαπλασιαζόμενη με Time.deltaTime και με μία άλλη μεταβλητή η οποία κρατάει το πόσο έντονη θέλουμε να είναι η επιρροή της ταχύτητας στο τελικό αποτέλεσμα. Με βάση αυτό, ο εχθρός κατά την επίθεση περιστρέφεται προς το σημείο πρόβλεψης της θέσης του παίκτη-στόχου.

4.5.12 Throw

Το Throw script χρησιμοποιείται από το MobBehaviour και έχει ως σκοπό να δημιουργήσει το πέταγμα ενός αντικειμένου. Κρατάει σε μία μεταβλητή το αντικείμενο που πετάει ο κάθε εχθρός. Κατά την εκκίνησή του το script ψάχνει αναδρομικά από το Parent Transform του εχθρού σε όλα τα παιδιά του, ώστε να εντοπίσει το Empty GameObject που αντιστοιχεί στο σημείο από το οποίο πετάγεται το αντικείμενο. Αυτός ο εντοπισμός γίνεται με βάση το όνομα του gameobject. Στη συνέχεια, όταν έρθει η ώρα να πετάξει ο εχθρός κάποιο αντικείμενο, τότε καλείται μία μέθοδος η οποία παίρνει ως ορίσματα όλα τα χαρακτηριστικά της βολής και του αντικειμένου, τα οποία είναι το damage, η ταχύτητα, η ταχύτητα περιστροφής, ο στόχος της βολής, η διάρκεια ζωής του αντικειμένου και η δύναμη με την οποία χτυπάει κάποιον αντίπαλο.

4.5.13 Cast

Το Cast script είναι όμοιο με το Throw. Χρησιμοποιείται από το MobBehaviour στην περίπτωση που ο τύπος της επίθεσης είναι caster. Στην εκκίνησή του ψάχνει αναδρομικά να βρει στο Parent Object το παιδί

που αντιστοιχεί στο σημείο στο οποίο δημιουργείται το spell. Κρατάει σαν μεταβλητή το GameObject του spell που χρησιμοποιεί ο κάθε εχθρός στην επίθεσή του.

4.5.14 GameController

Το GameController script διαχειρίζεται και αναλαμβάνει την αρχή του παιχνιδιού, την ροή του και το τέλος του. Αρχικά, αρχικοποιεί διάφορες μεταβλητές και έπειτα δημιουργεί την πίστα σύμφωνα με το αν παίζει ένας ή δύο παίκτες.

Σε κάθε update γίνεται έλεγχος για αλλαγή κατάστασης παιχνιδιού ανάλογα με την υπάρχουσα, καθώς και αν οι παίκτες έχουν χάσει. Η πρώτη κατάσταση είναι αυτή στην οποία δεν έχει γίνει spawn η ομάδα των εχθρών ενός wave. Στη συνέχεια, στο επόμενο update ακολουθεί η κατάσταση στην οποία το παιχνίδι είναι έτοιμο να αρχίσει το wave spawn, μετά από τον επιθυμητό χρόνο. Τότε, ακολουθεί η έναρξη δημιουργίας της ομάδας των εχθρών. Πρώτα, το GameController δείχνει στην οθόνη τον αριθμό του wave που είναι να εμφανιστεί. Έπειτα ξεκινάει την εμφάνιση της ομάδας των εχθρών και επίσης ανανεώνει όλα τα buffstands. Κατά την διάρκεια αυτών των ενεργειών η κατάσταση παιχνιδιού είναι ότι βρίσκεται στην διαδικασία δημιουργίας του wave. Μόλις τελειώσει η εμφάνιση των εχθρών, γίνεται περιοδικός έλεγχος για το αν οι εχθροί έχουν εξοντωθεί. Όταν εξοντωθούν όλοι οι εχθροί, τότε η κατάσταση παιχνιδιού ξαναγίνεται η κατάσταση στην οποία δεν έχει γίνει η δημιουργία των εχθρών. Έτσι, συνεχίζεται ο κύκλος παιχνιδιού και παράλληλα προχωράνε τα waves. Αν οι παίκτες νικήσουν το τελευταίο wave, τότε εμφανίζεται ένα αντίστοιχο μήνυμα στην οθόνη και το παιχνίδι τερματίζεται. Αντίστοιχο μήνυμα εμφανίζεται και στην περίπτωση όπου οι παίκτες χάσουν.

4.5.15 GameDictionary

Το GameDictionary είναι ένα βοηθητικό script που περιέχει static μεθόδους και κρατάει διάφορες πληροφορίες κυρίως για τα spells του παιχνιδιού. Για παράδειγμα το τι cooldown έχει κάθε spell, σε ποιο GameObject αντιστοιχεί, ποιο είναι το cast range του, με ποιο κουμπί εκτελείται και ούτω καθεξής. Άλλη μέθοδος που προφέρει είναι η εύρεση για το αν δύο χαρακτήρες είναι εχθροί μεταξύ τους. Αυτές τις μεθόδους τις χρησιμοποιούν τα υπόλοιπα scripts του παιχνιδιού ώστε να πάρουν την πληροφορία που χρειάζεται σε κάθε περίπτωση.

5 Επεξήγηση Κώδικα

Σε αυτό το κεφάλαιο γίνεται αναλυτική επεξήγηση κώδικα. Παρουσιάζονται τα script που αναλαμβάνουν την λειτουργικότητα του παίκτη στο παιχνίδι, αλλά και το GameController script που διαχειρίζεται την ροή του παιχνιδιού. Κάθε υποενότητα αφορά ένα διαφορετικό script. Σε κάθε μία από αυτές παρουσιάζεται ολόκληρος ο κώδικας του script χωρισμένος σε τμήματα και μαζί η επεξήγηση του τμήματος αυτού.

5.1 GameController.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour {

  private enum GameState
  {
    BeforeWaveSetToSpawn,
    ReadyToStartWaveSpawn,
    WaveIsSpawning,
    WaveIsSpawned,
    WaveIsDead,
    GameWon,
    None
  }
```

Στην αρχή γίνεται εισαγωγή των βιβλιοθηκών που χρειάζονται στο script. Εισήχθηκε το System.Collections.Generic για την υποστήριξη της δομής της λίστας. Το UnityEngine.UI χρησιμοποιείται για την υποστήριξη αντικειμένων UI όπως για παράδειγμα Button, Text και Image. Επιπλέον, εισάγουμε το UnityEngine.SceneManagement που διαχειρίζεται την φόρτωση μιας σκηνής στην Unity μέσω κώδικα. Αμέσως μετά την δήλωση της κλάσης GameController δηλώνουμε μια μεταβλητή enum η οποία αντιπροσωπεύει τα στάδια στα οποία μπορεί να βρίσκεται η ροή του παιχνιδιού.

```
  public static GameController instance;

  [SerializeField] private Transform gameWorld;

  [SerializeField] private GameObject _player2Canvas;

  [Header("Players")]
  [SerializeField] private GameObject player1;
  [SerializeField] private GameObject player2;

  [Header("Mobs")]
  [SerializeField] private GameObject[] _mobs;

  [Header("Mob Spawn Points")]
  [SerializeField] private Transform[] _mobSpawnPoints;

  [Header("Player Spawn Points")]
  [SerializeField] private Transform _player1SpawnPoint;
  [SerializeField] private Transform _player2SpawnPoint;

  [Header("Buff Stands")]
  [SerializeField] private BuffStand[] _buffStands;
```

Στη συνέχεια δηλώνουμε διάφορες μεταβλητές οι οποίες είναι ορατές στον Inspector και σκοπό έχουν

να κρατάνε μία αναφορά ενός αντικειμένου του παιχνιδιού. Η gameWord κρατάει το αντικείμενο το οποίο έχει ως παιδί του όλη την πίστα. Η μεταβλητή _player2Canvas κρατάει το αντικείμενο που αντιστοιχεί στο User Interface του δεύτερου παίκτη και αποθηκεύεται για να ενεργοποιηθεί αργότερα στην περίπτωση που παίζουν δύο παίκτες. Στις μεταβλητές player1 και player2 αποθηκεύονται τα πρωτότυπα αντικείμενα των δύο παικτών. Στο Array _mobs εισάγουμε όλα τα αντικείμενα εχθρών που υπάρχουν στο παιχνίδι, ενώ στο Array _mobSpawnPoints εισάγουμε τα αντικείμενα που αντιπροσωπεύουν σημεία στα οποία εμφανίζονται οι εχθροί κατά την φόρτωση του wave. Αντίστοιχα, τα _player1SpawnPoint, _player2SpawnPoint κρατάνε τις αρχικές θέσεις πάνω στις οποίες θα εμφανιστούν ο παίκτης 1 και ο παίκτης 2. Το Array _buffStands κρατάει αναφορά σε όλα τα BuffStands της πίστας.

```
[Header("Entrance Gate Collider")]
[SerializeField] private Transform _EntranceGateCollider;

[Header("Notifications Text")]
[SerializeField] private Text _waveText;
[SerializeField] private Text _gameOverText;
[SerializeField] private Text _winText;
[SerializeField] private Text _waveNumberText;
[SerializeField] private Text _fpsText;
[SerializeField] private AudioClip _winningSound;
[SerializeField] private GameObject _returnAfterWinButton;
```

Εδώ συνεχίζεται η δήλωση μεταβλητών. Η _EntranceGateCollider κρατάει ένα αντικείμενο το οποίο έχει επάνω του ένα Collider Component και χρησιμοποιείται σαν εμπόδιο για να αποτρέπει την έξοδο κάποιου παίκτη ή εχθρού από την πύλη. Αλλά επειδή όταν δημιουργείται ένα wave πρέπει να έχει άλλη θέση για να μπορούν οι εχθροί να μπουν μέσα, την κρατάμε στη μεταβλητή έτσι ώστε να αλλάζουμε την θέση της, όπως θα δούμε αργότερα. Οι Text μεταβλητές _waveText, _gameOverText, _winText, _waveNumberText και _fpsText κρατάνε αντικείμενα της σκηνής τα οποία εμφανίζουν τις κατάλληλες στιγμές διάφορα μηνύματα στην οθόνη. Για παράδειγμα όταν οι παίκτες χάσουν ή νικήσουν, εμφανίζει το αντίστοιχο μήνυμα. Η _winningSound κρατάει ένα AudioClip το οποίο είναι ένας ήχος που παίζει κατά τον τερματισμό του παιχνιδιού και η _returnAfterWinButton αποθηκεύει το κουμπί που εμφανίζεται στο τέλος του παιχνιδιού.

```
private AudioSource _audioSource;

private Vector3 _entranceGateColliderStartScale;
private Vector3 _entranceGateColliderEndScale;

public GameObject player1Instance { get; private set; }
public GameObject player2Instance { get; private set; }

private int _currentWaveNumber;
private List<GameObject> _currentWaveMobs;

private float counter = 0;
private bool _gameWon;

private GameState _gameState = GameState.None;
private GameState _previousGameState = GameState.None;
```

Ακολουθούν μεταβλητές που δεν είναι εμφανείς στον Inspector. Η μεταβλητή _audioSource χρησιμοποιείται για να αποθηκεύσει το AudioSource Component που έχει το αντικείμενο του GameController, έτσι ώστε να μπορεί να παίζει ήχους. Οι _entranceGateColliderStartScale και _entranceGateColliderEndScale αποθηκεύουν το αρχικό και τελικό μέγεθος του Collider που βρίσκεται στην πύλη από την οποία βγαίνουν τα τέρατα. Στις μεταβλητές player1Instance, player2Instanec αποθηκεύονται αντίστοιχα οι παίκτες αφού γίνει η δημιουργία τους από το πρωτότυπο (Prefab). Η _currentWaveNumber αποθηκεύει τον αριθμό του τωρινού wave και η λίστα _currentWaveMobs περιέχει κάθε τέρας το οποίο δημιουργείτε κατά τη διάρκεια

ενός wave. Η counter χρησιμοποιείται για την καταμέτρηση των FPS και η _gameWon αντιπροσωπεύει το αν το παιχνίδι έχει κερδηθεί. Τέλος, οι _gameState και _previousGameState κρατάνε σε κάθε Update το τωρινό και το προηγούμενο στάδιο της ροής του παιχνιδιού.

```
void Awake () {

  instance = this;
  _currentWaveNumber = 1;
  _waveNumberText.text = "Wave " + _currentWaveNumber.ToString();
  _currentWaveMobs = new List<GameObject>();

  _audioSource = GetComponent<AudioSource>();

  _returnAfterWinButton.SetActive(false);
  _player2Canvas.SetActive(false);

  _waveText.color = new Color(_waveText.color.r, _waveText.color.g,
     _waveText.color.b, 0);

  player1Instance = Instantiate(player1, _player1SpawnPoint.position,
     Quaternion.identity) as GameObject;
  if (PlayersNumber.number == 2)
  {
    player2Instance = Instantiate(player2, _player2SpawnPoint.position,
       Quaternion.identity) as GameObject;
    _player2Canvas.SetActive(true);
  }
  else
  {
    player2Instance = null;
  }

  _entranceGateColliderStartScale = _EntranceGateCollider.localScale;
  _entranceGateColliderEndScale = new
     Vector3(_EntranceGateCollider.localScale.x,
     _EntranceGateCollider.localScale.y, 52);

  StartCoroutine(EnableTheWorld());
}
```

Στην μέθοδο Awake κυρίως αρχικοποιούνται οι μεταβλητές. Αποθηκεύουμε στην μεταβλητή instance το στιγμιότυπο αυτής της κλάσης. Επειδή το παιχνίδι αρχίζει με το πρώτο wave, κάνουμε την _currentWaveNumber ίση με το ένα. Έπειτα αρχικοποιείται το κείμενο της μεταβλητής _waveNumberText σύμφωνα με τον τωρινό αριθμό του wave και επίσης αρχικοποιείται η λίστα της μεταβλητής _currentWaveMobs. Στη συνέχεια βάζουμε στην _audioSource το AudioSource Component που βρίσκεται στο αντικείμενο του GameController και απενεργοποιούμε το κουμπί τερματισμού και το UI του δεύτερου παίκτη. Ορίζουμε και την ορατότητα του χρώματος του κειμένου που μας ενημερώνει για τον αριθμό του wave, κάνοντάς το διάφανο.

Επειδή ο πρώτος παίκτης θα είναι σίγουρα στο παιχνίδι, δημιουργούμε τον παίκτη, σώζοντας το αντίστοιχο GameObject που θα παραχθεί στην player1Instance. Για τον δεύτερο παίκτη εξετάζουμε αν η στατική μεταβλητή number της κλάσης PlayersNumber ισούται με ένα ή δύο. Αυτό ορίζετε από το κεντρικό μενού, δηλαδή πριν δοθεί η εντολή για την δημιουργία της σκηνής του παιχνιδιού. Έτσι, αν τελικά ισούται με δύο, δημιουργούμε και σώζουμε τον δεύτερο παίκτη, καθώς επίσης ενεργοποιούμε και το UI που του αντιστοιχεί. Τέλος, αρχικοποιείται το αρχικό και τελικό μέγεθος του Collider που μπλοκάρει την είσοδο στην πύλη και ξεκινάμε την CoRoutine EnableTheWorld η οποία αρχίζει την ενεργοποίηση του αντικειμένου της πίστας.

```
void Update()
{
  counter += 1; //fps
```

```
    if(_gameState == GameState.BeforeWaveSetToSpawn && _gameState !=
        _previousGameState)
    {
      _previousGameState = _gameState;
      StartCoroutine(SetGameStateInTime(GameState.ReadyToStartWaveSpawn, 5f));
    }
    else if(_gameState == GameState.ReadyToStartWaveSpawn && _gameState !=
        _previousGameState)
    {
      _previousGameState = _gameState;
      StartCoroutine(ShowWaveNumber(_currentWaveNumber));
      StartCoroutine(SendNextWave());
      RefreshBuffStands();
      _gameState = GameState.WaveIsSpawning;
    }
    else if(_gameState == GameState.WaveIsSpawning && _gameState !=
        _previousGameState)
    {
      _previousGameState = _gameState;
    }
    else if(_gameState == GameState.WaveIsSpawned && _gameState !=
        _previousGameState)
    {
      _previousGameState = _gameState;
      StartCoroutine(CheckIfWaveIsDeadPerTime(3));
    }
    else if(_gameState == GameState.WaveIsDead && _gameState !=
        _previousGameState)
    {
      CheckAndUpdateHighScore(_currentWaveNumber);
      _previousGameState = _gameState;
      _gameState = GameState.BeforeWaveSetToSpawn;
      ClearPlayersSpecialSpell();
      _currentWaveNumber++;
      _waveNumberText.text = "Wave " + _currentWaveNumber.ToString();
    }
    else if (_gameWon && _gameState != _previousGameState)
    {
      _previousGameState = _gameState;
      StartCoroutine(ShowGameWinAndReturn());
    }
  }
```

Στην μέθοδο Update ελέγχεται και καθορίζεται το στάδιο στο οποίο βρίσκεται η ροή του παιχνιδιού. Η μεταβλητή counter αυξάνεται κατά ένα σε κάθε Update. Όπως αναφέρεται στη συνέχεια, η CoRoutine ManageFpsCount μετράει τα FPS.

Στην αρχή το _gameState ισούται με GameState.None, όπως ορίστηκε προηγουμένως στις μεταβλητές της κλάσης. Έπειτα, εφόσον τρέξει η CoRoutine EnableTheWorld η οποία καλέστηκε από την Awake, το _gameState γίνεται BeforeWaveSetToSpawn. Οπότε όταν γίνει αυτό, το πρόγραμμα περνάει από την πρώτη συνθήκη ελέγχου. Εκεί, ενημερώνουμε την μεταβλητή _previousGameState και καλούμε την CoRoutine SetGameStateInTime, δίνοντας την εντολή να γίνει το _gameState ίσο με ReadyToStartWaveSpawn σε χρόνο πέντε δευτερολέπτων.

Όταν περάσει αυτός ο χρόνος, το πρόγραμμα θα μπει στην δεύτερη συνθήκη ελέγχου. Σε αυτήν, αρχίζει η διαδικασία δημιουργίας του wave. Πρώτα καλείται η CoRoutine ShowWaveNumber, η οποία εμφανίζει στην οθόνη τον αριθμό του wave. Έπειτα καλείται η CoRoutine SendNextWave η οποία αναλαμβάνει την δημιουργία των καθορισμένων εχθρών και στη συνέχεια δίνεται εντολή να ανανεωθούν τα BuffStands με την βοήθεια της μεθόδου RefreshBuffStands. Επιπλέον αλλάζει το _gameState σε WaveIsSpawning.

Αμέσως μετά, θα εκτελεστεί ο κώδικας της τρίτης συνθήκης ελέγχου στον οποίο απλά ενημερώνεται το προηγούμενο στάδιο παιχνιδιού. Αφού ολοκληρωθεί η δημιουργία των εχθρών, η CoRoutine SendNextWave αλλάζει το στάδιο του παιχνιδιού σε WaveIsSpawned. Σε αυτό το σημείο το πρόγραμμα θα περάσει από την τέταρτη συνθήκη ελέγχου. Εκεί θα αρχίσει η CoRoutine CheckIfWaveIsDeadPesTime η οποία ελέγχει κάθε τρία δευτερόλεπτα αν η ομάδα των εχθρών έχει νικηθεί. Αν η ομάδα αυτή έχει νικηθεί, τότε η CoRoutine αυτή αλλάζει το _gameState σε WaveIsDead. Στην περίπτωση όπου η ομάδα των εχθρών αντιστοιχεί στο τελευταίο wave, τότε αλλάζει το _gameState σε GameWon.

Στην πρώτη περίπτωση, το πρόγραμμα περνάει από την πέμπτη συνθήκη ελέγχου. Μέσα σε αυτή θα καλεστεί η μέθοδος CheckAndUpdateHighScore η οποία ελέγχει αν το wave που νικήθηκε αποτελεί High Score και αν ναι, το αποθηκεύει. Σε αυτήν την συνθήκη ελέγχου ανανεώνεται ο αριθμός του τωρινού wave, το κείμενο που ανακοινώνει τον αριθμό του wave και επίσης με την βοήθεια της μεθόδου ClearPlayersSpecialSpell αφαιρούνται τυχόν SpecialSpell που έχουν οι παίκτες. Τέλος, το _gameState αλλάζει σε BeforeWaveSetToSpawn κι έτσι ξαναρχίζει ο κύκλος παιχνιδιού από την αρχή με το επόμενο wave.

Στην δεύτερη περίπτωση, εμφανίζεται στους παίκτες το μήνυμα ότι έφτασαν στον τερματισμό του παιχνιδιού και εμφανίζεται επίσης ένα κουμπί που τους οδηγεί πίσω στο αρχικό μενού. Αυτό γίνεται με την βοήθεια της CoRoutine ShowGameWinAndReturn.

```
private IEnumerator ManageFpsCount()
{
  while (true)
  {
    _fpsText.text = "FPS: " + counter.ToString();
    counter = 0;
    yield return new WaitForSeconds(1);
  }
}
```

Η CoRoutine ManageFpsCount αναλαμβάνει να πάρει την τιμή της μεταβλητής counter, η οποία αυξάνεται κατά ένα σε κάθε Update και να την αποτυπώσει στο αντικείμενο Text το οποίο περιέχει η μεταβλητή _fpsText. Κάθε ένα δευτερόλεπτο αποτυπώνει την τιμή της counter, έπειτα μηδενίζει την μεταβλητή αυτή και τέλος περιμένει ένα δευτερόλεπτο ώστε να ξανά αρχίσει την διαδικασία.

```
public IEnumerator EnableTheWorld()
{
  yield return new WaitForSeconds(0);
  yield return new WaitForSeconds(0);

  gameWorld.gameObject.SetActive(true);
  InitializeBuffStands();

  _gameState = GameState.BeforeWaveSetToSpawn;

  StartCoroutine(ManageFpsCount());
  StartCoroutine(CheckForGameOver());
}
```

Η CoRoutine EnableTheWorld καλείτε στο τέλος της Awake. Αρχικά περιμένει να περάσουν δύο frames και έπειτα ενεργοποιεί το αντικείμενο το οποίο περιέχει την πίστα του παιχνιδιού. Στη συνέχεια αρχικοποιεί τα BuffStands καλώντας την InitializeBuffStands, ενημερώνει το στάδιο του παιχνιδιού και αρχίζει την ManageFpsCount και την CheckForGameOver CoRoutine.

```
private IEnumerator SendNextWave()
{
  yield return new WaitForSeconds(3);

  _EntranceGateCollider.localScale = _entranceGateColliderStartScale;
```

```
    _currentWaveMobs = new List<GameObject>();

    foreach (GameObject go in CreateWaveFromData(_currentWaveNumber))
    {
      GameObject newMob =
      Instantiate(go, _mobSpawnPoints[Random.Range(0,
          _mobSpawnPoints.Length)].position, Quaternion.identity) as
          GameObject;

      if(PlayersNumber.number == 2)
      {
        LivingEntity lv = newMob.GetComponent<LivingEntity>();
        lv.maxHealth = (int) (lv.maxHealth * 1.8);
        lv.IncreaseHealth(lv.maxHealth);
      }

      newMob.GetComponent<MobBehaviour>().SetDestination(new
          Vector3(Random.Range(-5f, 5f), 0, Random.Range(-5f, 5f)));

      _currentWaveMobs.Add(newMob);

      yield return new WaitForSeconds(.8f);
    }
```

Η SendNextWave CoRoutine αναλαμβάνει το να εμφανίσει την ομάδα εχθρών που αντιστοιχεί στο κάθε wave. Αρχικά, υπάρχει αναμονή τριών δευτερολέπτων. Έπειτα ο Collider που μπλοκάρει την είσοδο αποκτά το αρχικό του μέγεθος ώστε να μην μπλοκάρει αργότερα τους εχθρούς και αρχικοποιείται η λίστα που κρατάει τους εχθρούς του τωρινού wave. Η Μέθοδος CreateWaveFromData που χρησιμοποιείται στην επανάληψη, όπως θα δούμε παρακάτω παίρνει σαν είσοδο τον αριθμό που αντιστοιχεί στο wave και επιστρέφει μία λίστα τύπου GameObject με τους εχθρούς. Μέσα στην επανάληψη, δημιουργούμε και αποθηκεύουμε σε μία προσωρινή μεταβλητή το GameObject που δημιουργήθηκε, τοποθετώντας το τυχαία σε κάποια από τις θέσεις που έχουμε ορίσει μέσα στην λίστα με τα _mobSpawnPoints. Στη συνέχεια ελέγχουμε αν στο παιχνίδι παίζουν δύο παίκτες. Στην περίπτωση αυτή, αυξάνουμε την ζωή του τέρατος κατά 80%, χρησιμοποιώντας το LivingEntity Component. Επίσης μέσω του MobBehaviour Script ορίζουμε την κατεύθυνση του εχθρού προς ένα τυχαίο σημείο στο κέντρο της πίστας, ώστε αυτός να κατευθυνθεί προς τα εκεί και να βγει από την πύλη. Τέλος, προσθέτουμε τον νέο εχθρό στην λίστα, και η διαδικασία αυτή επαναλαμβάνεται (με καθυστέρηση 0,8 δευτερολέπτων) για όλους τους επόμενους εχθρούς του wave.

```
    float timePast = 0;

    while(timePast < 3f)
    {
      _EntranceGateCollider.localScale=
          Vector3.Lerp(_entranceGateColliderStartScale,
          _entranceGateColliderEndScale, timePast / 3);

      timePast += Time.deltaTime;
      yield return null;
    }

    _previousGameState = _gameState;
    _gameState = GameState.WaveIsSpawned;
  }
```

Αφού λοιπόν δημιουργηθούν όλοι οι εχθροί, μένει να ξαναμπεί το εμπόδιο κοντά στην πύλη. Έτσι, σε διάρκεια τριών δευτερολέπτων μεταβάλλουμε τον Collider της πύλης από το αρχικό του μέγεθος στο τελικό. Τέλος, ενημερώνουμε το στάδιο του παιχνιδιού σε WaveIsSpawned.

```
private List<GameObject> CreateRandomMobWave(float mobsNumber)
{
  List<GameObject> mobList = new List<GameObject>();

  for(int i = 0; i < mobsNumber; i++)
  {
    mobList.Add(_mobs[Random.Range(0, _mobs.Length)]);
  }

  return mobList;
}
```

Αυτή η CoRoutine δεν χρησιμοποιείται στο υπάρχον παιχνίδι, αλλά μπορεί να φανεί χρήσιμη σε κάποια μελλοντική επέκταση του παιχνιδιού. Αυτό που κάνει είναι να δημιουργεί και να επιστρέφει μία λίστα με τα GameObject τυχαίων εχθρών. Το μέγεθος αυτής ορίζεται από την είσοδο της μεθόδου, δηλαδή την μεταβλητή mobsNumber.

```
private IEnumerator ShowWaveNumber(float waveNumber)
{
  _waveText.text = "Wave " + waveNumber;

  float timeCount = 0;

  while(timeCount < 3)
  {
    timeCount += Time.deltaTime;
    _waveText.color = new Color(_waveText.color.r, _waveText.color.g,
        _waveText.color.b, timeCount / 3);
    yield return null;
  }

  yield return new WaitForSeconds(3);

  timeCount = 0;

  while (timeCount < 3)
  {
    timeCount += Time.deltaTime;
    _waveText.color = new Color(_waveText.color.r, _waveText.color.g,
        _waveText.color.b, 1 - timeCount / 3);
    yield return null;
  }
}
```

Η ShowWaveNumber CoRoutine έχει ως σκοπό να παρουσιάσει στην οθόνη τον αριθμό του wave που πρόκειται να δημιουργηθεί. Πρώτα, αρχικοποιείται το κείμενο του αντικειμένου _WaveText σύμφωνα με το τωρινό wave και μηδενίζεται η μεταβλητή timeCount. Στην πρώτη επανάληψη, σε διάρκεια τριών δευτερολέπτων μεταβάλλουμε σταδιακά την διαφάνεια του κειμένου από μηδέν (διάφανο) σε ένα (μη διάφανο). Έπειτα, υπάρχει αναμονή τριών δευτερολέπτων. Στην δεύτερη επανάληψη γίνεται η αντίστροφη διαδικασία, δηλαδή αλλάζουμε το κείμενο από μη διάφανο σε διάφανο σε διάρκεια τριών δευτερολέπτων.

```
private IEnumerator ShowGameOver()
{
  _gameOverText.text = "Game Over";

  float timeCount = 0;

  while (timeCount < 3)
  {
    timeCount += Time.deltaTime;
    _gameOverText.color = new Color(_gameOverText.color.r,
        _gameOverText.color.g, _gameOverText.color.b, timeCount / 3);
```

```
      yield return null;
    }
  }
```

Η ShowGameOver χρησιμοποιεί ακριβώς την ίδια διαδικασία με την ShowWaveNumber CoRoutine για να κάνει εμφανές το μήνυμα τέλους του παιχνιδιού που περιέχει το κείμενο του αντικειμένου _gameOverText.

```
  private IEnumerator ShowGameWinAndReturn()
  {
    _winText.text = "Congratulations\nYou have won\nthe game!";

    float timeCount = 0;

    _audioSource.Stop();
    _audioSource.clip = _winningSound;
    _audioSource.Play();

    while (timeCount < 3)
    {
      timeCount += Time.deltaTime;
      _winText.color = new Color(_winText.color.r, _winText.color.g,
          _winText.color.b, timeCount / 3);
      yield return null;
    }

    yield return new WaitForSeconds(2);

    _returnAfterWinButton.SetActive(true);
  }
```

Η ShowGameWinAndReturn CoRoutine εμφανίζει και αυτή (όπως φαίνεται μέσα στην επανάληψη) το αντίστοιχο μήνυμα κατά τη νίκη του παιχνιδιού. Πέρα από αυτό, αναπαράγει επίσης ένα AudioClip μέσω του AudioSource Component που κρατάει η μεταβλητή _audioSource. Τέλος, ενεργοποιεί το κουμπί το οποίο προτείνει στον χρήστη την επιστροφή στο αρχικό μενού.

```
  private bool CheckIfWaveIsDead()
  {
    if (_currentWaveMobs.Count > 0)
    {
      foreach (GameObject mob in _currentWaveMobs)
      {
        if (mob != null)
        {
          if (GameDictionary.AreEnemies("Player", mob.tag))
          {
            return false;
          }
        }
      }
      return true;
    }

    return false;
  }
```

Η μέθοδος CheckIfWaveIsDead, όταν η λίστα του τρέχοντος wave δεν είναι άδεια, διατρέχει την λίστα αυτή και ελέγχει αν υπάρχει κάποιος ζωντανός εχθρός. Σημειώνεται ότι όταν κάποιος εχθρός νικηθεί, τότε το αντικείμενό του καταστρέφεται και κάθε μεταβλητή που έδειχνε σε αυτό γίνεται ίση με null. Αν βρει έστω και ένα εχθρό, επιστρέφει false, δηλαδή ότι το wave δεν έχει νικηθεί. Αν όχι, επιστρέφει true, δηλαδή ότι το

wave έχει νικηθεί.

```
private IEnumerator SetGameStateInTime(GameState gameState, float time)
{
  yield return new WaitForSeconds(time);

  _previousGameState = _gameState;
  _gameState = gameState;
}
```

Η SetGameStateInTime, όπως αναφέρει και το όνομά της, ανανεώνει το στάδιο του παιχνιδιού μετά από κάποιον επιθυμητό χρόνο. Αναλαμβάνει επίσης να ενημερώσει και την μεταβλητή που αποθηκεύει το προηγούμενο στάδιο παιχνιδιού.

```
private IEnumerator CheckIfWaveIsDeadPerTime(float time)
{
  while (true)
  {
    if (CheckIfWaveIsDead())
    {
      if (_currentWaveNumber == 15)
      {
        _gameWon = true;
        _previousGameState = _gameState;
        _gameState = GameState.GameWon;
      }
      else
      {
        _previousGameState = _gameState;
        _gameState = GameState.WaveIsDead;
      }
      break;
    }

    yield return new WaitForSeconds(time);
  }
}
```

Η CheckIfWaveIsDeadPerTime ελέγχει διαρκώς με βάση την επιθυμητή περίοδο αν οι εχθροί του wave έχουν νικηθεί. Αν έχουν νικηθεί, τότε αλλάζει το στάδιο του παιχνιδιού σε WaveIsDead και τερματίζεται μέσω της break. Αν ο αριθμός του wave που νικήθηκε είναι το 15, δηλαδή το τελευταίο, τότε αλλάζει το στάδιο του παιχνιδιού σε GameWon.

```
private IEnumerator CheckForGameOver()
{
  while (true)
  {
    yield return new WaitForSeconds(5);

    if (player2Instance == null)
    {
      if (player1Instance.GetComponent<LivingEntity>().isDead())
      {
        break;
      }
    }
    else
    {
      if (player1Instance.GetComponent<LivingEntity>().isDead() &&
          player2Instance.GetComponent<LivingEntity>().isDead() )
      {
```

```
            break;
          }
        }
      }

      StartCoroutine(ShowGameOver());

      yield return new WaitForSeconds(10f);

      SceneManager.LoadScene("Menu");
    }
```

Η CheckForGameOver ελέγχει κάθε πέντε δευτερόλεπτα αν ο παίκτης ή οι παίκτες είναι ζωντανοί, δηλαδή αν έχουν χάσει ή όχι. Αν έχουν χάσει, τότε εμφανίζεται το αντίστοιχο μήνυμα με την βοήθεια της ShowGameOver και μετά από δέκα δευτερόλεπτα γίνεται επιστροφή στο αρχικό μενού.

```
    private void RefreshBuffStands()
    {
      foreach(BuffStand stand in _buffStands)
      {
        if (stand.gameObject.activeSelf)
        {
          stand.GenerateRandomBuff();
        }
      }
    }
```

Η μέθοδος RefreshBuffStands ανανεώνει όλα τα ενεργά BuffStands, έτσι ώστε να ξεκινήσει η διαδικασία εμφάνισης τυχαίας βοήθειας για τον παίκτη.

```
    private void ClearPlayersSpecialSpell()
    {
      if(player1Instance != null)
      {
        player1Instance.GetComponent<PlayerSpell>().RemoveSpecialSpell();
      }
      if (player2Instance != null)
      {
        player2Instance.GetComponent<PlayerSpell>().RemoveSpecialSpell();
      }
    }
```

Η μέθοδος ClearPlayersSpecialSpell φροντίζει να αφαιρεθεί από κάθε παίκτη τυχόν ειδικό spell που μπορεί να κατέχει.

```
    private GameObject GetMobByName(string name)
    {
      foreach(GameObject go in _mobs)
      {
        if(go.name.Equals(name.Trim()))
        {
          return go;
        }
      }

      return null;
    }
```

Η μέθοδος GetMobByName παίρνει ως όρισμα μία συμβολοσειρά που αντιπροσωπεύει το όνομα ενός εχθρού και στην περίπτωση που υπάρχει αυτό το όνομα στην λίστα με τους εχθρούς, τότε επιστρέφει το

αντικείμενο αυτού. Αν δεν βρεθεί ο εχθρός με το ζητούμενο όνομα, τότε επιστρέφει την τιμή null.

```
private List<GameObject> CreateWaveFromData(int waveNum)
{
  List<GameObject> mobList = new List<GameObject>();
  TextAsset waves = Resources.Load("waves") as TextAsset;
  string[] wavesText = waves.text.Split('\n');

  bool startReadingWave = false;

  foreach(string line in wavesText)
  {
    if (line.Contains("[wave" + waveNum.ToString() + ']'))
    {
      startReadingWave = true;
    }

    if (startReadingWave)
    {
      if (line.StartsWith("+"))
      {
        if (GetMobByName(line.Substring(1, line.Length - 1)) != null)
        {
          mobList.Add(GetMobByName(line.Substring(1, line.Length - 1)));
        }
      }
    }

    if (line.Contains("[]") && startReadingWave)
    {
      break;
    }
  }

  return mobList;
}
```

Η μέθοδος CreateWaveFromData παίρνει ως είσοδο έναν ακέραιο ο οποίος αντιπροσωπεύει τον αριθμό του wave και επιστρέφει μία λίστα με όλα τα GameObject των εχθρών που υπάρχουν σε αυτό. Αυτό γίνεται βάση ενός αρχείου κειμένου στο οποίο είναι καταγεγραμμένο το ποιους εχθρούς περιέχει κάθε wave.

Πρώτα, αρχικοποιείται η λίστα που πρόκειται να επιστραφεί και δημιουργείται ένα αντικείμενο τύπου TextAsset το οποίο κρατάει το περιεχόμενου του αρχείου κειμένου. Στον πίνακα συμβολοσειρών wavesText κάνουμε κάθε στοιχείο του πίνακα να είναι μία γραμμή του κειμένου αυτού. Μέσα στην επανάληψη, για κάθε γραμμή αρχικά ψάχνουμε το σημείο στο οποίο αναφέρεται το wave που ψάχνουμε. Όταν αυτό βρεθεί, κάνουμε την μεταβλητή startReadingWave ίση με true. Τότε, το πρόγραμμα περνάει για πρώτη φορά από την δεύτερη συνθήκη ελέγχου. Εκεί, αν η γραμμή αρχίζει με το σύμβολο '+' και η υπόλοιπη γραμμή έχει το όνομα ενός υπαρκτού εχθρού, τότε με την βοήθεια της GetMobByName εισάγεται το GameObject αυτού του εχθρού στην λίστα με τους εχθρούς. Η επανάληψη και η προσθήκη εχθρών στην λίστα συνεχίζεται έως ότου βρεθεί η συμβολοσειρά '[]'. Τέλος, επιστρέφεται η λίστα με τους εχθρούς.

```
private void InitializeBuffStands()
{
  if(PlayersNumber.number == 1)
  {
    _buffStands[4].gameObject.SetActive(false);
    _buffStands[5].gameObject.SetActive(false);
  }
  else
  {
    _buffStands[4].gameObject.SetActive(true);
```

```
      _buffStands[5].gameObject.SetActive(true);
    }
  }
```

Η μέθοδος InitializeBuffStands χρησιμοποιείται για να ενεργοποιήσει ή απενεργοποιήσει τα επιπλέον buffstands ανάλογα με το αν παίζουν ένας ή δύο παίκτες.

```
  private void CheckAndUpdateHighScore(int score)
  {
    if(score > PlayerPrefs.GetInt("highscore"))
    {
      PlayerPrefs.SetInt("highscore", score);
      PlayerPrefs.Save();
    }
  }
}
```

Η μέθοδος CheckAndUpdateHighScore παίρνει ως είσοδο έναν ακέραιο ο οποίος αντιπροσωπεύει τον αριθμό του wave που νικήθηκε και στην περίπτωση που αποτελεί HighScore, τότε καταγράφεται.

5.2 PlayerAnimation.cs

```
using UnityEngine;

public class PlayerAnimation : MonoBehaviour
{
  private Animator _animator;

  void Awake()
  {
    _animator = GetComponent<Animator>();
  }

  public void SetAnimator(Animator animator)
  {
    _animator = animator;
  }

  public void SetIsMoving(bool status)
  {
    _animator.SetBool("isMoving", status);
  }
```

Η κλάση PlayerAnimation χρησιμοποιείται από τα άλλα scripts στην περίπτωση που χρειάζεται να γίνει αλλαγή κάποιας μεταβλητής του Animator Controller του παίκτη. Αρχικά, μέσα στην Awake αποθηκεύουμε τον AnimatorController που βρίσκεται επάνω στο αντικείμενο του παίκτη ως Component. Η μέθοδος SetAnimator δίνει την δυνατότητα αλλαγής του AnimatorController της μεταβλητής _animator. Η SetIsMoving αλλάζει την μεταβλητή isMoving που βρίσκεται στον AnimatorController του παίκτη και αντιπροσωπεύει το εάν ο παίκτης κινείται ή είναι στάσιμος.

```
  public void SetIsCasting(bool status)
  {
    _animator.SetBool("isCasting", status);
    _animator.ResetTrigger("OnCastCancel");
  }

  public void SetIsDead(bool status)
  {
    _animator.SetBool("isDead", status);
```

```
  }

  public void OnCastCancel()
  {
    _animator.SetTrigger("OnCastCancel");
    _animator.SetBool("isCasting", false);
  }

}
```

Η μέθοδος SetIsCasting αλλάζει την μεταβλητή isCasting του AnimatorController του παίκτη. Όταν γίνει αληθής, τότε αρχίζει το animation του spellcasting. Η SetIsDead αλλάζει την μεταβλητή isDead η οποία αντιπροσωπεύει το εάν ο παίκτης έχει ηττηθεί. Τέλος, η OnCastCancel χρησιμοποιείται για την διακοπή κάποιου spell κατά την διαδικασία δημιουργίας του.

5.3 PlayerCooldown.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class PlayerCooldown : MonoBehaviour
{
  private Stats _playerStats;
  private Dictionary<SpellName, float> _spellCooldowns;

  void Awake()
  {
    _playerStats = GetComponent<Stats>();
    _spellCooldowns = new Dictionary<SpellName, float>();
  }
```

Η κλάση PlayerCooldown έχει μία δομή Dictionary, η οποία έχει ως κλειδί το όνομα του spell και ως τιμή έναν δεκαδικό αριθμό που αντιπροσωπεύει τον εναπομένοντα χρόνο σε δευτερόλεπτα για να είναι έτοιμο το spell για επαναχρησιμοποίηση. Η μεταβλητή _playerStats χρησιμοποιείται για να κρατάει το Stats script που έχει ο παίκτης, διότι υπάρχουν Stats τα οποία μειώνουν τον χρόνο επαναχρησιμοποίησης των spell. Στην μέθοδο Awake αρχικοποιούνται αυτές οι δύο μεταβλητές.

```
  public void RegisterSpellCooldown(SpellName spellName)
  {
    if(!_spellCooldowns.ContainsKey(spellName))
    {
      _spellCooldowns.Add(spellName, 0);
    }
  }

  public void UnregisterSpellCooldown(SpellName spellName)
  {
    if (_spellCooldowns.ContainsKey(spellName))
    {
      _spellCooldowns.Remove(spellName);
    }
  }
```

Η RegisterSpellCooldown χρησιμοποιείται κάθε φορά που θέλουμε να προστεθεί μία καινούρια εγγραφή στο Dictionary με τα spells. Παίρνει ως είσοδο λοιπόν το όνομα του spell και αν δεν υπάρχει ήδη το ίδιο κλειδί, τότε το προσθέτει βάζοντας ως τιμή το μηδέν. Η τιμή αυτή δηλώνει ότι το spell είναι έτοιμο να χρησιμοποιηθεί. Αντιθέτως, η UnregisterSpellCooldown παίρνει ως όρισμα το όνομα ενός spell και αν υπάρχει

μέσα στο Dictionary το αφαιρεί.

```
public void StartSpellCooldown(SpellName spellName)
{
  _spellCooldowns[spellName] = GetFinalSpellCooldown(spellName);
  StartCoroutine(ProcessCooldown(spellName));
}

private float GetFinalSpellCooldown(SpellName spellName)
{
  return GameDictionary.GetSpellCooldownTime(spellName) -
      _playerStats.cooldownConstant -
  GameDictionary.GetSpellCooldownTime(spellName) *
      _playerStats.cooldownQuota;
}
```

Η μέθοδος StartSpellCooldown αρχικά βρίσκει την τελική τιμή του χρόνου που απαιτεί η επαναχρησιμοποίηση ενός επιθυμητού spell. Αυτό γίνεται με την βοήθεια της μεθόδου GetFinalSpellCooldown. Αφού την βρει, την αποθηκεύει στην τιμή του Dictionary που αντιστοιχεί στο επιθυμητό spell. Έπειτα ξεκινάει την ProcessCooldown CoRoutine η οποία αναλαμβάνει να μετρήσει το πότε θα είναι έτοιμο το spell για να ξαναχρησιμοποιηθεί. Η GetFinalSpellCooldown βρίσκει τον χρόνο επαναχρησιμοποίησης ενός spell, με βάση το αρχικό cooldown που είναι καταγεγραμμένο στο GameDictionary αλλά και με βάση τα Stats του παίκτη.

```
private IEnumerator ProcessCooldown(SpellName spellName)
{
  float cd = _spellCooldowns[spellName];

  while (cd > 0)
  {
    yield return new WaitForSeconds(.1f);
    cd -= .1f;
    if (_spellCooldowns.ContainsKey(spellName))
    {
      _spellCooldowns[spellName] = cd;
    }
  }

  _spellCooldowns[spellName] = 0;
}
```

Η ProcessCooldown CoRoutine ξεκινάει την αντίστροφη μέτρηση του χρόνου επαναχρησιμοποίησης ενός spell. Αρχικά αποθηκεύει τον χρόνο επαναχρησιμοποίησης σε μια μεταβλητή. Έπειτα, μέσα στην επανάληψη, κάθε εκατοστό του δευτερολέπτου αφαιρεί τον χρόνο αυτό από την μεταβλητή στην οποία αποθηκεύσαμε τον συνολικό χρόνο επαναχρησιμοποίησης και ανανεώνει την αντίστοιχη τιμή με την νέα τιμή στο Dictionary, αφού ελέγξει αν υπάρχει το κλειδί.

```
public bool HasCooldown(SpellName spellName)
{
  float spellCurrentCooldown;
  if (_spellCooldowns.TryGetValue(spellName, out spellCurrentCooldown))
  {
    return spellCurrentCooldown == 0;
  }

  return false;
}
```

```
  public float GetSpellCooldownPercentage(SpellName spellName)
  {
    float spellCurrentCooldown;
    if (_spellCooldowns.TryGetValue(spellName, out spellCurrentCooldown))
    {
      return (spellCurrentCooldown / GetFinalSpellCooldown(spellName)) * 100;
    }

    return -1;
  }
}
```

Η μέθοδος HasCoolDown ελέγχει και επιστρέφει για ένα spell αν η τιμή που αντιστοιχεί στο Dictionary είναι μηδέν ή όχι. Δηλαδή αν το spell είναι έτοιμο για επαναχρησιμοποίηση. Η GetSpellCooldownPercentage επιστρέφει τον αριθμό που αντιπροσωπεύει το ποσοστό κατά το οποίο είναι έτοιμο κάποιο spell.

5.4 PlayerSpell.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class PlayerSpell : MonoBehaviour
{
  private PlayerCooldown _playerCooldown;
  private PlayerAnimation _playerAnimation;
  private RigidbodyWrapper _rgw;

  [HideInInspector]
  public bool spellBarFlag;

  private SpellName _selectedSpell;
  public bool spellOnCast { get; private set; }
  private Vector3 _pointToCastTo;

  [SerializeField] private List<SpellName> _spells;
  private SpellName _specialSpell = SpellName.None;

  private IEnumerator currentCastSelectedSpell;
  private IEnumerator rotateOnCast;

  public SpellName specialSpell
  {
    get { return _specialSpell; }
  }

  public Vector3 pointToCastTo
  {
    get { return _pointToCastTo; }
    set { _pointToCastTo = value; }
  }
```

Η κλάση PlayerSpell διαχειρίζεται το σύστημα δημιουργίας των spell του παίκτη. Όπως φαίνεται και στην δήλωση των τριών πρώτων μεταβλητών, η PlayerSpell χρησιμοποιεί τα Scripts PlayerCooldown, PlayerAnimation και RigidbodyWrapper του παίκτη. Η μεταβλητή spellBarFlag χρησιμοποιείται για να ειδοποιήσει την μπάρα που δείχνει τα spells στον παίκτη, αν υπάρχει κάποια αλλαγή. Η _selectedSpell κρατάει το spell που έχει επιλεχθεί και η spellOnCast δηλώνει εάν κάποιο spell βρίσκεται στην διαδικασία δημιουργίας. Το pointToCastTo είναι το σημείο στόχευσης του spell. Η λίστα _spells περιέχει όλα τα απλά spells του παίκτη τα οποία εισάγονται μέσο του Inspector, ενώ η μεταβλητή _specialSpell περιέχει το ειδικό spell που μπορεί να έχει στην κατοχή του ο παίκτης. Οι δύο μεταβλητές τύπου IEnumerator currentCastSelectedSpell και RotateOnCast αποθηκεύουν τις αντίστοιχες CoRoutines κατά την δημιουργία ενός spell, δίνοντας την

δυνατότητα ακύρωσης της διαδικασίας αυτής, όπως θα δούμε παρακάτω.

```
void Awake()
{
  _playerCooldown = GetComponent<PlayerCooldown>();
  _playerAnimation = GetComponent<PlayerAnimation>();
  _rgw = GetComponent<RigidbodyWrapper>();

  selectedSpell = SpellName.None;
  pointToCastTo = Vector3.zero;
  spellOnCast = false;
  currentCastSelectedSpell = null;
  rotateOnCast = null;

  foreach( SpellName spellName in _spells)
  {
    _playerCooldown.RegisterSpellCooldown(spellName);
  }
  spellBarFlag = true;
}
```

Στην μέθοδο Awake αρχικοποιούνται τα τρία Scripts που χρησιμοποιεί η PlayerSpell με τα αντίστοιχα Scripts/Components που έχει το αντικείμενο του παίκτη. Αρχικοποιούνται επίσης οι μεταβλητές selectedSpell, pointToCastTo, spellOnCast και οι δύο IEnumerator τύπου μεταβλητές που αναφέραμε προηγουμένως. Έπειτα, για κάθε spell της λίστας _spells καλούμε την μέθοδο RegisterSpellCooldown του PlayerCooldown script για να γίνει και εκεί η καταχώρηση. Δηλώνουμε την spellBarFlag ίση με true, ώστε να ενημερωθεί η μπάρα με τα spells του παίκτη.

```
public SpellName selectedSpell
{
  get { return _selectedSpell; }
  private set
  {
    _selectedSpell = value;
  }
}

void SpellIsOnCast(bool status, bool castCancelOccured = false)
{
  spellOnCast = status;

  if (castCancelOccured)
  {
    _playerAnimation.OnCastCancel();
  }
  else
  {
    _playerAnimation.SetIsCasting(status);
  }
}
```

Η SpellIsOnCast χρησιμοποιείται για την αλλαγή τιμής στην μεταβλητή spellOnCast. Επειδή πρέπει να δράσουμε διαφορετικά όταν η τιμή της αλλάζει κατά την διαδικασία ακύρωσης ενός spell ή όχι, χρησιμοποιούμε την boolean μεταβλητή castCancelOccured. Στην πρώτη περίπτωση ενημερώνουμε το script _playerAnimation για την ακύρωση ενός spell, ενώ στην δεύτερη για την δημιουργία του spell.

```
public void CancelCast()
{
```

```
    if (currentCastSelectedSpell != null)
    {
      StopCoroutine(currentCastSelectedSpell);
      if (rotateOnCast != null)
      {
        StopCoroutine(rotateOnCast);
      }
      SpellIsOnCast(false, true);
    }
  }
```

Η μέθοδος CancelCast τρέχει κατά την ακύρωση της διαδικασίας εκτόξευσης ενός spell. Αρχικά ελέγχουμε αν έχει προηγηθεί και είναι σε εξέλιξη κάποια διαδικασία εκτόξευσης, ελέγχοντας αν η currentCastSelectedSpell είναι διάφορη του null. Σε αυτήν την περίπτωση σταματάμε την διαδικασία αυτή, καθώς και την διαδικασία περιστροφής του παίκτης κατά την εκτόξευση του spell χρησιμοποιώντας την StopCoroutine. Τέλος, ενημερώνουμε ανάλογα την spellIsOnCast.

```
  public void CastSpell(SpellName spellName)
  {
    GameObject spellObject;

    if (_spells.Contains(spellName) || spellName == _specialSpell
        && spellName != SpellName.None &&
           _playerCooldown.HasCooldown(spellName))
    {
      spellObject = GameDictionary.GetSpellGameObject(spellName);
      if (spellObject != null)
      {
        currentCastSelectedSpell = StartSpellCast(spellName);
        StartCoroutine(currentCastSelectedSpell);
        if(GameDictionary.GetSpellType(spellName) == SpellType.NormalCast)
        {
          rotateOnCast = StartRotateOnCast(spellName);
          StartCoroutine(rotateOnCast);
        }
      }
    }
  }
```

Η CastSpell εκτελείται όταν δοθεί εντολή εκκίνησης της διαδικασίας εκτόξευσης ενός spell και αποτελεί την πρώτη φάση της διαδικασίας αυτής. Στην πρώτη συνθήκη ελέγχου εξετάζουμε αν ο παίκτης κατέχει το επιλεγόμενο spell αλλά και αν είναι έτοιμο να χρησιμοποιηθεί. Αν ισχύουν αυτά, τότε βρίσκουμε με την βοήθεια της GameDictionary το GameObject του spell και το αποθηκεύουμε στην μεταβλητή spellObject. Έπειτα, αν έχει βρεθεί επιτυχώς το ζητούμενο spell, τότε ξεκινάμε την StartSpellCast CoRoutine και την αποθηκεύουμε στην μεταβλητή currentCastSelectedSpell ώστε να είναι δυνατή η ακύρωση της διαδικασίας. Αν το spell είναι τύπου NormalCast, τότε ξεκινάμε και αποθηκεύουμε και την StartRotateOnCast η οποία αναλαμβάνει να περιστρέψει ομαλά τον παίκτη προς την κατεύθυνση εκτόξευσης.

```
  private IEnumerator StartSpellCast(SpellName spellName)
  {
    SpellIsOnCast(true);

    yield return new
        WaitForSeconds(GameDictionary.GetSpellCastingTime(spellName));

    if (rotateOnCast != null)
    {
```

```
      StopCoroutine(rotateOnCast);
    }
    currentCastSelectedSpell = null;
    rotateOnCast = null;
    SpellIsOnCast(false);

    _playerCooldown.StartSpellCooldown(spellName);

    GameObject spell = GameDictionary.GetSpellGameObject(spellName);
    spell.SetActive(false);
    GameObject castedSpell = Instantiate(spell);
    castedSpell.GetComponent<Spell>().caster = transform;
    castedSpell.SetActive(true);

  }
```

Η StartSpellOnCast αποτελεί την δεύτερη φάση της εκτόξευσης εφόσον έχει δοθεί η εντολή. Πρώτα ενημερώνουμε ότι ο παίκτης βρίσκεται στην διαδικασία εκτόξευσης spell, αλλάζοντας την spellOnCast σε αληθή. Έπειτα περιμένουμε για χρόνο ίσο με την ώρα που χρειάζεται το συγκεκριμένο spell να εκτοξευθεί. Αφού περάσει ο χρόνος αυτός, σταματάμε την περιστροφή του παίκτη σταματώντας την CoRoutine που είναι αποθηκευμένη στην rotateOnCast. Στη συνέχεια, θέτουμε τις δύο IEnumerator μεταβλητές ίσες με null, εφόσον έχουν διακοπεί και με την ίδια λογική αλλάζουμε την spellOnCast σε false. Έπειτα, δίνουμε εντολή στο PlayerCooldown script να ξεκινήσει την μέτρηση χρόνου για την καθυστέρηση επαναχρησιμοποίησης του spell. Τέλος, βρίσκουμε το GameObject του spell, το απενεργοποιούμε ώστε να γίνει η αρχικοποίηση που χρειάζεται (καταχώρηση του Transform του παίκτη στην μεταβλητή caster που κληρονομεί από την κλάση Spell) και τότε το ενεργοποιούμε.

```
  private IEnumerator StartRotateOnCast(SpellName spellName)
  {
    float timeLimit = GameDictionary.GetSpellCastingTime(spellName);

    Vector3 castDestination = pointToCastTo - transform.position;
    castDestination.y = 0;

    Quaternion startRotation = transform.rotation;
    Quaternion rotation = Quaternion.LookRotation(castDestination);

    for (float t = 0; t < 1;)
    {
      t = Mathf.Min(t + Time.deltaTime / timeLimit, 1);
      transform.rotation = Quaternion.Lerp(startRotation, rotation, t);
      yield return null;
    }

  }
```

Η StartRotateOnCast Coroutine διαχειρίζεται την ομαλή περιστροφή του παίκτη, λίγο πριν γίνει η εκτόξευση του spell. Έχει ως σκοπό να περιστρέψει τον παίκτη προς το σημείο εκτόξευσης σε χρόνο ίσο με αυτόν που χρειάζεται το ζητούμενο spell για να εκτοξευθεί. Αρχικά αποθηκεύουμε τον χρόνο αυτόν στην μεταβλητή timeLimit. Βρίσκουμε και αποθηκεύουμε επίσης την κατεύθυνση εκτόξευσης και παρακάτω την κατεύθυνση περιστροφής. Μέσα στην επανάληψη, χρησιμοποιώντας την μέθοδο Lerp, πραγματοποιούμε την ομαλή περιστροφή του παίκτη στον διαθέσιμο χρόνο.

```
  public bool HasSpell(SpellName spellName)
  {
    return _spells.Contains(spellName);
  }
```

```
public void SelectSpell(SpellName spellName)
{
  if(spellName == SpellName.None)
  {
    selectedSpell = spellName;
    return;
  }
  if(_playerCooldown.HasCooldown(spellName))
  {
    selectedSpell = spellName;
  }
}
```

Η μέθοδος HasSpell παίρνει ως είσοδο ένα ζητούμενο spell, εξετάζει αν αυτό υπάρχει στην λίστα με τα spell και επιστρέφει το αποτέλεσμα. Η SelectSpell χρησιμοποιείται για την αλλαγή της τιμής του επιλεγμένου spell που κρατάει η μεταβλητή selectedSpell. Αν το spell που επιλέχθηκε δεν είναι έτοιμο για επαναχρησιμοποίηση, τότε δεν πραγματοποιεί την αλλαγή.

```
public void AddSpell(SpellName spellName)
{
  _spells.Add(spellName);
  _playerCooldown.RegisterSpellCooldown(spellName);
  spellBarFlag = true;
}

public void RemoveSpell(SpellName spellName)
{
  if (_spells.Contains(spellName))
  {
    _spells.Remove(spellName);
    spellBarFlag = true;
  }
}
```

Η AddSpell προσθέτει ένα spell στην λίστα με τα spell του παίκτη, προσθέτει την αντίστοιχη εγγραφή στο PlayerCooldown script και ενημερώνει επίσης ότι έγινε κάποια αλλαγή που ενδιαφέρει το script που διαχειρίζεται την μπάρα με τα spell. Η RemoveSpell κάνει το αντίθετο, δηλαδή αφαιρεί το ζητούμενο spell, αν αυτό υπάρχει.

```
public void RemoveSpecialSpell()
{
  _playerCooldown.UnregisterSpellCooldown(_specialSpell);
  _specialSpell = SpellName.None;
  spellBarFlag = true;
}

public void AddSpecialSpell(SpellName spellName)
{
  _specialSpell = spellName;
  _playerCooldown.RegisterSpellCooldown(spellName);
  spellBarFlag = true;
}

public bool HasSpecialSpell()
{
  return _specialSpell != SpellName.None;
}
```

Η RemoveSpecialSpell και η AddSpecialSpell κάνουν την ίδια διαδικασία με τις AddSpell, RemoveSpell,

αλλά για τα ειδικά spell.

```
  public List<SpellName> GetPlayerSpells(bool includingSpecialSpell = false)
  {
    if (includingSpecialSpell && _specialSpell != SpellName.None)
    {
      List<SpellName> result = new List<SpellName>(_spells);
      result.Add(_specialSpell);
      return result;
    }

    return new List<SpellName>(_spells);
  }
}
```

Η μέθοδος GetPlayerSpells επιστρέφει μια λίστα με όλα τα απλά spell του παίκτη. Στην περίπτωση που ζητηθεί, επιστρέφει μαζί με αυτά και το ειδικό spell.

5.5 Stats.cs

```
using UnityEngine;
using System.Collections.Generic;

public class Stats : MonoBehaviour
{
  private LivingEntity _livingEntity;
  private RigidbodyWrapper _rgw;

  [HideInInspector]
  public bool statsPanelFlag;

  public int baseMaxHealth { get; private set; }
  public int baseRegeneration { get; private set; }
  public float baseSpeed { get; private set; }
  public float baseMass { get; private set; }

  public int maxHealth { get; private set; }
  public int regeneration { get; private set; }
  public float speed { get; private set; }
  public float damageConstant { get; private set; }
  public float damageQuota { get; private set; }
  public float mass { get; private set; }
  public float cooldownConstant { get; private set; }
  public float cooldownQuota { get; private set; }

  private Dictionary<Stat, float> _stats;

  public Dictionary<Stat, float> stats
  {
    get { return _stats; }
  }
```

Η κλάση Stats χρησιμοποιείται για τον υπολογισμό και την αποθήκευση των stat ενός παίκτη. Οι μεταβλητές _livingEntity και _rgw κρατάνε τα δύο αντίστοιχα Components του παίκτη, τα οποία χρειάζεται η κλάση Stats, ώστε να αντλήσει από αυτά τις αρχικές τιμές των στοιχείων του παίκτη. Δηλαδή την αρχική τιμή του μεγίστου της ζωής του παίκτη, την αρχική ταχύτητα, την τιμή αναπλήρωσης της ζωής και την μάζα του. Παρακάτω δηλώνονται αυτές οι τέσσερις αναφερόμενες μεταβλητές, καθώς και η statsPanelFlag, η οποία χρησιμεύει για να ειδοποιήσει το πάνελ με τα stats ότι υπήρξε κάποια αλλαγή.

Στη συνέχεια, οι maxHealth, regeneration, speed, mass κρατάνε την τελική τιμή των στοιχείων που αντιπροσωπεύουν. Η damageConstant και η cooldownConstant κρατάνε έναν σταθερό αριθμό ο οποίος σημαίνει

το κατά πόσο είναι αυξημένη ή μειωμένη η αρχική τιμή της επίθεσης και του χρόνου επαναχρησιμοποίησης ενός spell αντίστοιχα. Οι damageQuota, cooldownQuota κρατάνε το ποσοστό κατά το οποίο είναι αυξημένες ή μειωμένες αυτές οι τιμές. Τέλος, δηλώνουμε ένα Dictionary στο οποίο καταχωρείται κάθε stat που έχει ο παίκτης, καθώς και ο χρόνος διάρκειάς του.

```
void Awake()
{
  _rgw = GetComponent<RigidbodyWrapper>();
  _livingEntity = GetComponent<LivingEntity>();

  _stats = new Dictionary<Stat, float>();

  InitializeData();
}

void Start()
{
  StatInit();
  statsPanelFlag = true;
}
```

Στην μέθοδο Awake αρχικοποιείται η μεταβλητή _rgw που κρατάει το RigidbodyWrapper του παίκτη, η _livingEntity που κρατάει το LivingEntity component του παίκτη και αρχικοποιείται επίσης το Dictionary που αποθηκεύει τα stat με τους αντίστοιχους χρόνους τους. Η μέθοδος InitializeData που καλείται στην συνέχεια, αρχικοποιεί τις αρχικές μεταβλητές.

Στην Start καλείται η μέθοδος StatInit η οποία αρχικοποιεί όλες τις μεταβλητές με τις τελικές τιμές των στοιχείων του παίκτη. Γίνεται στη συνέχεια ενημέρωση του πάνελ με τα stat, κάνοντας την statsPanelFlag ίση με true.

```
void Update()
{
  List<Stat> toDelete = new List<Stat>();
  bool thereAreChanges = false;

  foreach(Stat stat in new List<Stat>(_stats.Keys))
  {
    if (_stats[stat] > 0)
    {
      _stats[stat] -= Time.deltaTime;
    }
    else if(_stats[stat] != -1)
    {
      toDelete.Add(stat);
      if (thereAreChanges == false)
      {
        thereAreChanges = true;
      }
    }
  }

  foreach(Stat stat in toDelete)
  {
    _stats.Remove(stat);
  }

  if (thereAreChanges)
  {
    CalculateStats();
  }
```

```
}
```

Στην μέθοδο Update διαχειριζόμαστε τον χρόνο διάρκειας των stat, δηλαδή αφαιρούμε τον χρόνο που περνάει σταδιακά και όταν αυτός τελειώσει, τότε αφαιρούμε τα stat από την λίστα _stats. Αρχικά, δηλώνουμε μία κενή λίστα τύπου Stat και μία μεταβλητή boolean που δηλώνει αν θα υπάρξουν αλλαγές. Μέσα στην επανάληψη για κάθε stat του παίκτη, όταν ο χρόνος που απομένει είναι μεγαλύτερος του μηδέν, τότε αφαιρούμε τον χρόνο που έχει περάσει από την τιμή του Dictonary στο stat που αντιστοιχεί. Αν είναι μικρότερος του μηδέν και διάφορος του -1 (το οποίο χρησιμοποιείται για stat με άπειρη διάρκεια), τότε προστίθετε στην λίστα με τα προς διαγραφή stat. Στην επόμενη επανάληψη, κάθε stat που περιέχει, το διαγράφουμε από την λίστα με τα stat του παίκτη. Έπειτα, αν έχει γίνει κάποια αλλαγή, γίνεται ξανά ο υπολογισμός των στοιχείων του παίκτη.

```
public void InitializeData()
{
  baseMaxHealth = _livingEntity.maxHealth;
  baseRegeneration = _livingEntity.regeneration;
  baseSpeed = _rgw.speed;
  baseMass = _rgw.mass;
}

public void StatInit()
{
  maxHealth = baseMaxHealth;
  regeneration = baseRegeneration;
  speed = baseSpeed;
  damageConstant = 0;
  damageQuota = 0;
  mass = baseMass;
  cooldownConstant = 0;
  cooldownQuota = 0;
}
```

Η μέθοδος InitializeData αρχικοποιεί τις μεταβλητές που αποτελούν βάση των στοιχείων του παίκτη, με βάση τις αρχικές τους τιμές τις οποίες παίρνουμε από τα άλλα script (π.χ. LivingEntity, RigidbodyWrapper).

Η StatInit αρχικοποιεί όλες τις μεταβλητές οι οποίες αντιπροσωπεύουν τις τελικές τιμές των στοιχείων του παίκτη.

```
public void AddStat(Stat stat, float lifetime = -1f)
{
  _stats.Add(stat, lifetime);

  CalculateStats();
}

public void Remove(Stat stat)
{
  _stats.Remove(stat);

  CalculateStats();
}

public void RemoveAllStats()
{
  _stats = new Dictionary<Stat, float>();

  CalculateStats();
}

public List<Stat> GetAllStats()
```

```
{
  return new List<Stat>(_stats.Keys);
}
```

Η AddStat παίρνει ως είσοδο ένα stat και τον χρόνο διάρκειάς του. Με βάση αυτά, το προσθέτει στο Dictonary _stats που κρατάει όλα τα stat του παίκτη. Η Remove παίρνει ως όρισμα πάλι ένα stat και αν υπάρχει στο Dictionary τότε το αφαιρεί. Και οι δύο αυτές μέθοδοι χρησιμοποιούν στο τέλος την CalculateStats, η οποία ξανά υπολογίζει τις τελικές τιμές των ιδιοτήτων του παίκτη με βάση τις νέες αλλαγές που έγιναν. Η RemoveAllStats αφαιρεί όλα τα stat από τον παίκτη και η GetAllStats επιστρέφει μία λίστα που περιέχει όλα τα stat του παίκτη.

```
public void CalculateStats()
{
  StatInit();
  foreach (Stat stat in _stats.Keys)
  {
    if (stat.type == StatType.Constant)
    {
      switch (stat.category)
      {
        case StatCategory.MaxHealth:
          maxHealth += (int)stat.factorValue;
          break;

        case StatCategory.Regeneration:
          regeneration += (int)stat.factorValue;
          break;

        case StatCategory.Speed:
          speed += stat.factorValue;
          break;

        case StatCategory.Damage:
          damageConstant += stat.factorValue;
          break;

        case StatCategory.Mass:
          mass += stat.factorValue;
          break;

        case StatCategory.Cooldown:
          cooldownConstant += stat.factorValue;
          break;
      }
    }
  }
```

Η μέθοδος CalculateStats υπολογίζει και ενημερώνει τις τιμές των ιδιοτήτων του παίκτη με βάση τα υπάρχοντα stat. Αρχικά αρχικοποιούνται οι μεταβλητές με την βοήθεια της StatInit. Έπειτα εξετάζουμε πρώτα όλα τα stat που είναι τύπου σταθεράς. Άρα μέσα στην επανάληψη ανατρέχουμε το λεξικό με τα stat, για όσα είναι τύπου σταθεράς. Έπειτα, ελέγχουμε την κατηγορία κάθε stat, δηλαδή σε ποια ιδιότητα του παίκτη αποσκοπούν και προσθέτουμε την τιμή τους στην αντίστοιχη τιμή της μεταβλητής που αντιπροσωπεύει την ιδιότητα αυτή.

```
  foreach (Stat stat in _stats.Keys)
  {
    if (stat.type == StatType.Quota)
    {
      switch (stat.category)
```

```
        {
          case StatCategory.MaxHealth:
            maxHealth += (int)(maxHealth * stat.factorValue);
            break;

          case StatCategory.Regeneration:
            regeneration += (int)(regeneration * stat.factorValue);
            break;

          case StatCategory.Speed:
            speed += speed * stat.factorValue;
            break;

          case StatCategory.Damage:
            damageQuota += stat.factorValue;
            break;

          case StatCategory.Mass:
            mass += mass * stat.factorValue;
            break;

          case StatCategory.Cooldown:
            cooldownQuota += stat.factorValue;
            break;
        }
      }
    }

    ReApplyVariables();
    statsPanelFlag = true;
  }
```

Στην δεύτερη επανάληψη κάνουμε το ίδιο αλλά για τα stat τύπου ποσοστού. Για κάθε στοιχείο του παίκτη, η τιμή του θα μεταβληθεί με βάση την τιμή που έχει και την τιμή του ποσοστού που υπάρχει στο stat. Τέλος, με την βοήθεια της ReApplyVariables προσαρμόζουμε και ενημερώνουμε τις νέες τιμές των ιδιοτήτων του παίκτη στα script που τις χρησιμοποιούν και ειδοποιούμε επίσης μέσω της statsPanelFlag ότι η λίστα που δείχνει τα stats στην οθόνη πρέπει να ενημερωθεί.

```
  private void ReApplyVariables()
  {
    _livingEntity.maxHealth = maxHealth;
    _livingEntity.regeneration = regeneration;
    _rgw.mass = mass;
    _rgw.speed = speed;
  }
}
```

Η ReApplyVariables με βάση τις τιμές των ιδιοτήτων του παίκτη που κρατάει η κλάση Stats, ενημερώνει τις τιμές αυτές και στα υπόλοιπα script που τις χρησιμοποιούν.

5.6 PlayerController.cs

```
using UnityEngine;

public class PlayerController : MonoBehaviour
{
  [SerializeField] private PlayerNumber _playerNumber;
  [SerializeField] GameObject _pointToCastToIndex;
  [SerializeField] AudioClip _deathSound;

  private RigidbodyWrapper _rgw;
  private PlayerAnimation _playerAnimation;
  private PlayerSpell _playerSpell;
```

```
private PlayerCooldown _playerCooldown;
private LivingEntity _livingEntity;
private AudioSource _audioSource;

private bool _isDead;

private string _playerNo;
```

Η PlayerController διαχειρίζεται την είσοδο ενός παίκτη και με βάση αυτή πράττει τις κατάλληλες ενέργειες. Περιέχει επίσης την λειτουργία χειρισμού του στόχου, όταν ο παίκτης θέλει να εκτοξεύσει κάποιο spell.

Οι τρεις πρώτες μεταβλητές που φαίνονται και στον Inspector είναι η _playerNumber, η οποία κρατάει αν ο παίκτης είναι ο παίκτης ένα ή δύο, η pointToCastToIndex η οποία είναι ένα gameObject που αντιπροσωπεύει το σημείο εκτόξευσης και η _deathSound η οποία είναι ο ήχος κατά την ήττα του παίκτη. Έπειτα ακολουθούν μεταβλητές που κρατάνε τα script που έχουν να κάνουν με την συμπεριφορά του παίκτη και το καθένα από αυτά αναλαμβάνει και διαχειρίζεται ένα συγκεκριμένο πρόβλημα. Δηλώνουμε επίσης την μεταβλητή _isDead που δηλώνει αν ο παίκτης έχει ηττηθεί ή όχι καθώς και την _playerNo που αντιπροσωπεύει σε μορφή string τον αριθμό του παίκτη.

```
void Start()
{
  _rgw = GetComponent<RigidbodyWrapper>();
  _playerAnimation = GetComponent<PlayerAnimation>();
  _playerSpell = GetComponent<PlayerSpell>();
  _playerCooldown = GetComponent<PlayerCooldown>();
  _livingEntity = GetComponent<LivingEntity>();
  _audioSource = GetComponent<AudioSource>();

  _pointToCastToIndex = Instantiate(_pointToCastToIndex) as GameObject;
  _pointToCastToIndex.SetActive(false);

  _playerNo = _playerNumber.ToString();
}
```

Η μέθοδος Start αρχικοποιεί τις μεταβλητές που αναφέρθηκαν προηγουμένως, θέτοντας σε αυτές το αντίστοιχο Script/Component του παίκτη. Επίσης δημιουργείται το αντικείμενο που κρατάει το σημείο εκτόξευσης και αμέσως μετά απενεργοποιείται για να μην είναι φανερό. Αρχικοποιείται επίσης και η μεταβλητή με τον αριθμό του παίκτη.

```
void Update()
{
  if (_isDead) { return; }

  // Take care of the mouse / move - spellcast
  Vector3 direction = Vector3.zero;
  SpellName spellPressed = SpellName.None;

  // Movement
  if (Input.GetButton("Vertical_" + _playerNo))
  {
    direction.z = Input.GetAxis("Vertical_" + _playerNo);
  }
  if (Input.GetButton("Horizontal_" + _playerNo))
  {
    direction.x = Input.GetAxis("Horizontal_" + _playerNo);
  }
```

Στην μέθοδο Update πραγματοποιείται η λειτουργικότητα που προσφέρει το PlayerController. Στην

αρχή, ελέγχεται αν ο παίκτης έχει ηττηθεί. Σε αυτή την περίπτωση το πρόγραμμα επιστρέφει. Διαφορετικά, το πρόγραμμα συνεχίζει την εκτέλεση του κώδικα της Update.

Είναι σημαντικό να αναφερθεί ότι στον Input Manager της Unity έχουν οριστεί και για τους δύο παίκτες τα ονόματα των εισόδων για κάθε κίνηση και spell. Η ονομασία τους είναι η ίδια, με την διαφορά ότι για τον πρώτο παίκτη όλες οι ονομασίες τελειώνουν σε '_P1', ενώ για τον δεύτερο σε '_P2'. Την επέκταση αυτή αποθηκεύει και η μεταβλητή _playerNo.

Αρχικά ορίζουμε δύο μεταβλητές. Η μεταβλητή direction χρησιμοποιείται για την αποθήκευση την κατεύθυνσης κίνησης του παίκτη, ενώ η spellPressed για το όνομα πιθανού spell. Έπειτα ελέγχουμε την οριζόντια και κατακόρυφη κίνηση του παίκτη, ανάλογα με το αν πατήθηκε η αντίστοιχη είσοδος. Αποθηκεύουμε τότε την κίνηση στον κατάλληλο άξονα.

```
//Spell Choice
if (Input.GetButtonDown("Fireball_" + _playerNo))
{
  spellPressed = SpellName.Fireball;
}
if (Input.GetButtonDown("PlasmaField_" + _playerNo))
{
  spellPressed = SpellName.PlasmaField;
}
if (Input.GetButtonDown("Teleport_" + _playerNo))
{
  spellPressed = SpellName.Teleport;
}
if (Input.GetButtonDown("BulletStorm_" + _playerNo))
{
  spellPressed = SpellName.BulletStorm;
}
if (Input.GetButtonDown("SpecialSpell_" + _playerNo) &&
    _playerSpell.specialSpell != SpellName.None)
{
  spellPressed = _playerSpell.specialSpell;
}
```

Έπειτα, ακολουθεί η είσοδος για τα spell. Ελέγχουμε λοιπόν αν πατήθηκε είσοδος που αντιστοιχεί σε κάποιο spell και αποθηκεύουμε το όνομα αυτού στην μεταβλητή spellPressed.

```
if (_playerSpell.selectedSpell == SpellName.None &&
    !_playerSpell.spellOnCast)
{
  _rgw.direction = direction;

  if (_rgw.HasDirection())
  {
    _playerAnimation.SetIsMoving(true);
  }
  else
  {
    _playerAnimation.SetIsMoving(false);
  }
}
else
{
  _rgw.direction = Vector3.zero;
  _playerAnimation.SetIsMoving(false);
}
```

Σε αυτήν την συνθήκη, αν ο παίκτης δεν έχει επιλέξει κάποιο spell για επιλογή και επίσης δεν βρίσκεται

στην διαδικασία εκτόξευσης ενός spell, τότε ενημερώνει την κίνηση του παίκτη με βάση την μεταβλητή direction, λέγοντας στο RigidbodyWrapper ποια είναι η κατεύθυνση της κίνησης. Ελέγχει επίσης με την βοήθεια της HasDirection του RigidbodyWrapper script αν ο παίκτης έχει κάποια κατεύθυνση διάφορη του μηδενικού διανύσματος. Αν έχει, τότε ενημερώνει ανάλογα το script που διαχειρίζεται το animation του παίκτη, καθώς και όταν δεν έχει. Αν ο παίκτης έχει επιλέξει κάποιο spell ή βρίσκεται στην διαδικασία εκτόξευσής του, τότε θέτουμε την κατεύθυνση ίση με το μηδενικό διάνυσμα και ενημερώνουμε επίσης το PlayerAnimation ότι ο παίκτης δεν κινείται.

```
if(_playerSpell.selectedSpell != SpellName.None &&
   !_playerSpell.spellOnCast)
{
  UpdateCastPoint(direction,
     GameDictionary.GetMinSpellCastDistance(_playerSpell.selectedSpell),
  GameDictionary.GetMaxSpellCastDistance(_playerSpell.selectedSpell));
}
```

Όταν ο παίκτης έχει επιλέξει κάποιο spell και δεν βρίσκεται στην διαδικασία εκτόξευσής του, τότε με την βοήθεια της UpdateCastPoint και με βάση την είσοδο του παίκτη που επηρεάζει την τιμή της μεταβλητής direction, ενημερώνουμε την θέση του στόχου εκτόξευσης. Στην μέθοδο αυτή περνάμε επίσης τα όρια της απόστασης του σημείου εκτόξευσης που έχει το επιλεγμένο spell.

```
foreach (SpellName spellName in _playerSpell.GetPlayerSpells(true))
{
  if (spellName == spellPressed)
  {
    if (_playerSpell.selectedSpell == SpellName.None &&
       !_playerSpell.spellOnCast)
    {
      if (GameDictionary.GetSpellType(spellName) == SpellType.InstantCast
         && _playerCooldown.HasCooldown(spellName))
      {
        _playerSpell.CastSpell(spellName);
        _playerAnimation.SetIsMoving(false);
      }
      else if (GameDictionary.GetSpellType(spellName) ==
         SpellType.NormalCast && _playerCooldown.HasCooldown(spellName))
      {
        _playerSpell.SelectSpell(spellName);
        SetCastPointToDefaultPosition(GameDictionary.GetDefaultSpellCastDistance(spell
      }
    }
    else if (!_playerSpell.spellOnCast && _playerSpell.selectedSpell ==
       spellName) // and selectedSpell != SpellName.none
    {
      _playerSpell.CastSpell(_playerSpell.selectedSpell);
      _playerSpell.SelectSpell(SpellName.None);
      _playerAnimation.SetIsMoving(false);
      HideCastPoint();
    }
  }
}
```

Η επανάληψη αυτή διαχειρίζεται την είσοδο του παίκτη που έχει να κάνει με τα spell. Για κάθε spell του παίκτη λοιπόν ελέγχεται αν έχει επιλεχθεί/πατηθεί. Στην περίπτωση που έχει πατηθεί, υπάρχουν δύο υποπεριπτώσεις. Η μία είναι όταν δεν έχει ήδη επιλεχθεί κάποιο spell και η άλλη όταν υπάρχει κάποιο επιλεγμένο spell.

Στην πρώτη περίπτωση, αν το spell είναι τύπου InstantCast και είναι έτοιμο για χρήση, τότε προχωράμε στην διαδικασία εκτόξευσής του. Αν είναι τύπου NormalCast και είναι έτοιμο για χρήση, τότε επιλέγουμε αυτό το spell και εμφανίζουμε και κάνουμε αρχικοποίηση του σημείου εκτόξευσης (με την βοήθεια της SetCastPointToDefaultPosition) ανάλογα με την προκαθορισμένη θέση που έχει το κάθε spell.

Στην δεύτερη περίπτωση ελέγχουμε αν το spell που επιλέχθηκε/πατήθηκε έχει ήδη επιλεχθεί. Αν έχει επιλεχθεί το ίδιο, τότε προχωράμε στην διαδικασία εκτόξευσής του. Αυτή η περίπτωση χρησιμοποιείται για να γίνει η εκτόξευση των spell που είναι τύπου NormalCast, αφού ο παίκτης πρώτα επιλέγει το spell, έπειτα διαλέγει τον στόχο εκτόξευσης και τέλος το εκτοξεύει. Αφού δοθεί αυτή η εντολή, επιλέγουμε το SpellName.None, δηλαδή κανένα spell, σταματάμε το animation της κίνησης και απενεργοποιούμε το σημείο εκτόξευσης.

```
    if (Input.GetButtonDown("SpellCancel_" + _playerNo))
    {
      _playerSpell.SelectSpell(SpellName.None);
      HideCastPoint();
    }

    if (!_isDead)
    {
      if (_livingEntity.isDead())
      {
        _isDead = true;
        _rgw.StopMoving();
        _rgw.EraseExternalForces();
        _audioSource.PlayOneShot(_deathSound);
        _playerAnimation.SetIsDead(true);
      }
    }
  }
```

Στην πρώτη συνθήκη ελέγχου εξετάζουμε αν ο παίκτης με βάση την είσοδο θέλησε να ακυρώσει την επιλογή κάποιου spell. Σε αυτήν την περίπτωση, επιλέγουμε το SpellName.None και απενεργοποιούμε το σημείο εκτόξευσης. Στην δεύτερη συνθήκη ελέγχου εξετάζουμε αν ο παίκτης έχει ηττηθεί, παίρνοντας την πληροφορία από το LivingEntity του παίκτη. Τότε, ενημερώνουμε την μεταβλητή _isDead, σταματάμε την κίνηση του παίκτη, παίζουμε τον αντίστοιχο ήχο και ενημερώνουμε το PlayerAnimation ότι ο παίκτης ηττήθηκε, ώστε να ξεκινήσει το ανάλογο animation.

```
  private void SetCastPointToDefaultPosition(float defaultDistance = 10f)
  {
    Vector3 newIndexPos = transform.position + transform.forward *
       defaultDistance + Vector3.up * .3f;
    newIndexPos.y = .2f;

    _pointToCastToIndex.SetActive(true);
    _pointToCastToIndex.transform.SetParent(transform);
    _pointToCastToIndex.transform.position = newIndexPos;
    _playerSpell.pointToCastTo = _pointToCastToIndex.transform.position;
  }
```

Η μέθοδος SetCastPointToDefaultPosition ενεργοποιεί το σημείο εκτόξευσης και το τοποθετεί στην επιθυμητή απόσταση από τον παίκτη. Η θέση αυτή ορίζεται στην μεταβλητή newIndexPos. Με βάση αυτό ανανεώνουμε την θέση του σημείου εκτόξευσης που βλέπει ο παίκτης στην οθόνη καθώς και το σημείο εκτόξευσης που κρατάει το PlayerSpell script.

```
  private void UpdateCastPoint(Vector3 direction, float minDistance = 1f,
```

```
    float maxDistance = 40f)
  {
    Vector3 moveDelta = (_pointToCastToIndex.transform.position -
       transform.position).normalized * direction.z * Time.deltaTime * 70;
    moveDelta.y = .2f - _pointToCastToIndex.transform.position.y;

    if (Vector3.Distance(transform.position,
       _pointToCastToIndex.transform.position + moveDelta) < maxDistance &&
    Vector3.Distance(transform.position,
       _pointToCastToIndex.transform.position + moveDelta) > minDistance)
    {
      _pointToCastToIndex.transform.Translate(moveDelta, Space.World);
    }

    _pointToCastToIndex.transform.RotateAround(transform.position,
       Vector3.up, direction.x * Time.deltaTime * 150);

    _playerSpell.pointToCastTo = _pointToCastToIndex.transform.position;
  }
```

Η UpdateCastPoint αναλαμβάνει την ενημέρωση της θέσης του σημείου εκτόξευσης ενός spell. Έχει ως είσοδο την επιθυμητή κατεύθυνση που πρέπει να κινηθεί το σημείο αυτό, καθώς και το μέγιστο και ελάχιστο όριο απόστασης από την παίκτη.

Ο άξονας z αντιπροσωπεύει την κίνηση κοντά και μακριά από τον παίκτη. Αυτή η κίνηση υπολογίζεται και αποθηκεύεται αρχικά στην moveDelta. Η direction.z παίρνει τις τιμές 0, 1, και -1. Έπειτα, στην συνθήκη ελέγχου ελέγχουμε αν η νέα θέση του σημείου εκτόξευσης θα είναι μέσα στα επιθυμητά όρια. Αν είναι μέσα σε αυτά, τότε ενημερώνουμε την θέση του στην καινούρια. Στην συνέχεια, με βάση τον άξονα x του διανύσματος υπολογίζουμε την κίνηση περιστροφής του σημείου εκτόξευσης γύρω από τον παίκτη και ενημερώνουμε την θέση αυτή.

```
  private void HideCastPoint()
  {
    _pointToCastToIndex.SetActive(false);
  }
}
```

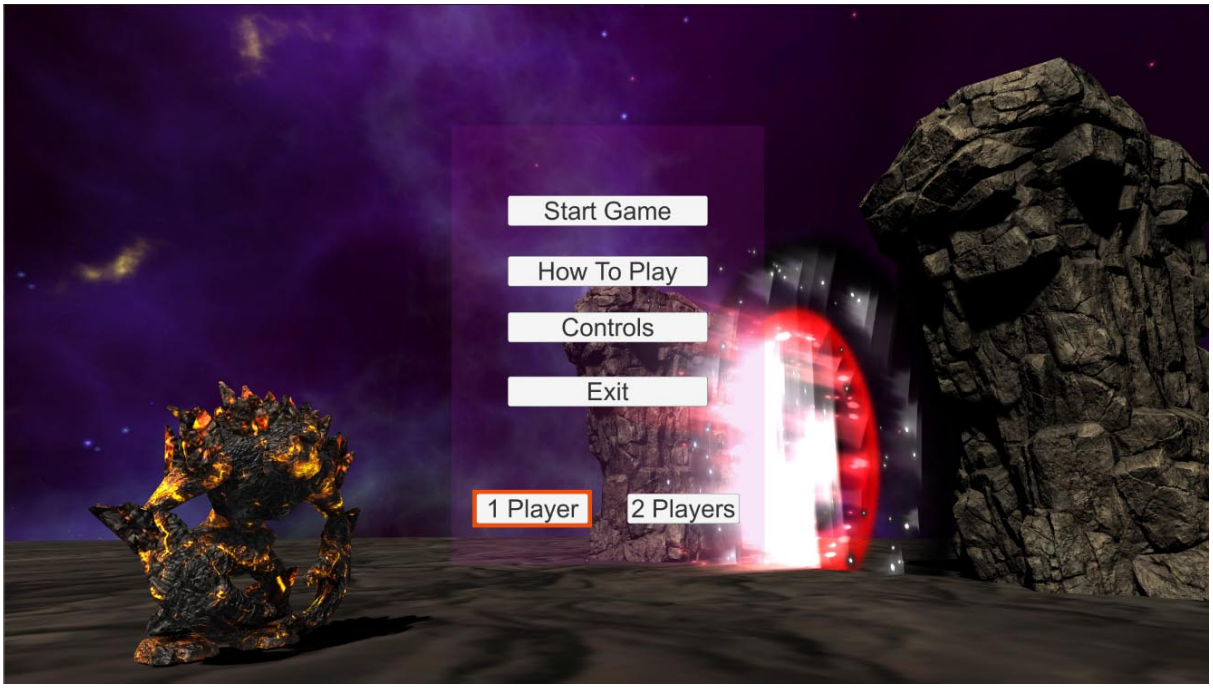
Η μέθοδος HideCastPoint απενεργοποιεί το αντικείμενο που αντιπροσωπεύει το σημείο εκτόξευσης που βλέπει ο παίκτης στην οθόνη.

6 Αποτέλεσμα

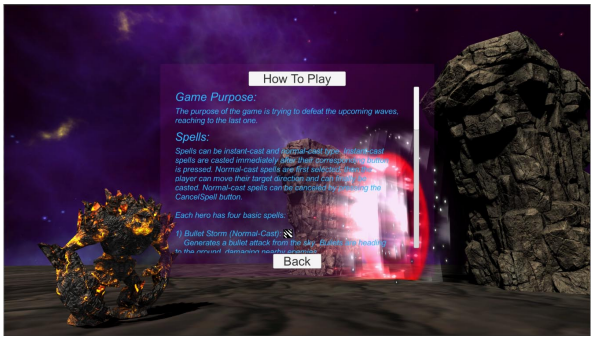
Αυτό το κεφάλαιο είναι το τελευταίο κεφάλαιο της εργασίας. Σκοπό έχει να παρουσιάσει μερικές σκηνές και εικόνες μέσα από το τελικό παιχνίδι και να αναφέρει διάφορες δυσκολίες που υπήρξαν κατά την ανάπτυξή του. Επίσης, στο τέλος αναφέρονται διάφορες προτάσεις για τυχόν μελλοντική βελτίωση του παιχνιδιού.

6.1 Gameplay

Στις πρώτες τρεις εικόνες στο Σχήμα 32 βλέπουμε τα στιγμιότυπα περιήγησης στο αρχικό μενού. Αυτά αποτελούν την κεντρική εικόνα του μενού, την καρτέλα με τις οδηγίες του παιχνιδιού και την καρτέλα με τα κουμπιά τα οποία χρησιμοποιούνται. Στη συνέχεια, στο Σχήμα 33 παρουσιάζονται στιγμιότυπα από το gameplay του παιχνιδιού.



(α΄) Αρχικό

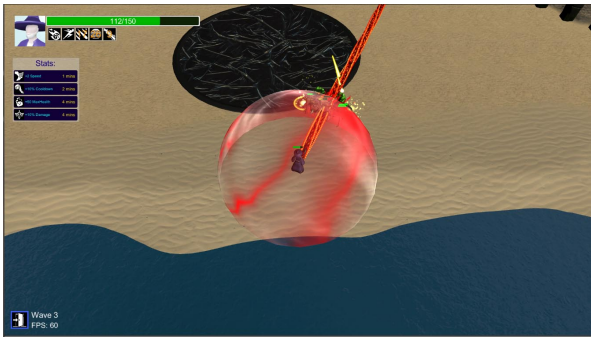


(β΄) How To Play

(γ΄) Controls

Σχήμα 32: Μενού

(α΄)



(β΄)

(γ΄)

Σχήμα 33: Σκηνές Παιχνιδιού

6.2 Δυσκολίες κατά την υλοποίηση

Κατά την υλοποίηση της ανάπτυξης του παιχνιδιού, παρουσιάστηκαν όπως ήταν αναμενόμενο διάφορες δυσκολίες. Η πρώτη από αυτές ήταν η κατάκτηση των απαιτούμενων γνώσεων σχετικά με την Unity, τις δυνατότητες τις οποίες προσφέρει καθώς και το Scripting API. Πριν την εξοικείωση μου με αυτά, ήταν αδύνατο να αρχίσει η οργάνωση και ο προγραμματισμός του τελικού Project. Σε αυτή την φάση, μεγάλη βοήθεια αποτέλεσε η μελέτη του βιβλίου Unity In Action του Joseph Hocking[4], η ιστοσελίδα της Unity με το Documentation και επίσης η ιστοσελίδα Unity Answers[5].

Όσον αφορά τη δυσκολία υλοποίησης του τελικού Project, υπήρξε κάποια αβεβαιότητα στο θέμα οργάνωσης συγκεκριμένων script και στην σχεδίαση του πως να αλληλεπιδρούν μεταξύ τους. Για παράδειγμα το PlayerController μαζί με τα υπόλοιπα βοηθητικά script/component που έχει ο παίκτης. Παρότι εκτελούν επιτυχώς τον σκοπό τους, οι σχέσεις μεταξύ τους έχουν έντονη εξάρτηση που σημαίνει ότι μια ριζική αλλαγή στο ένα script θα απαιτεί αντίστοιχες αλλαγές και στα υπόλοιπα.

Επίσης, η εύρεση των κατάλληλων και ταιριαστών Sound Effect ήταν λίγο δύσκολη και χρονοβόρα,

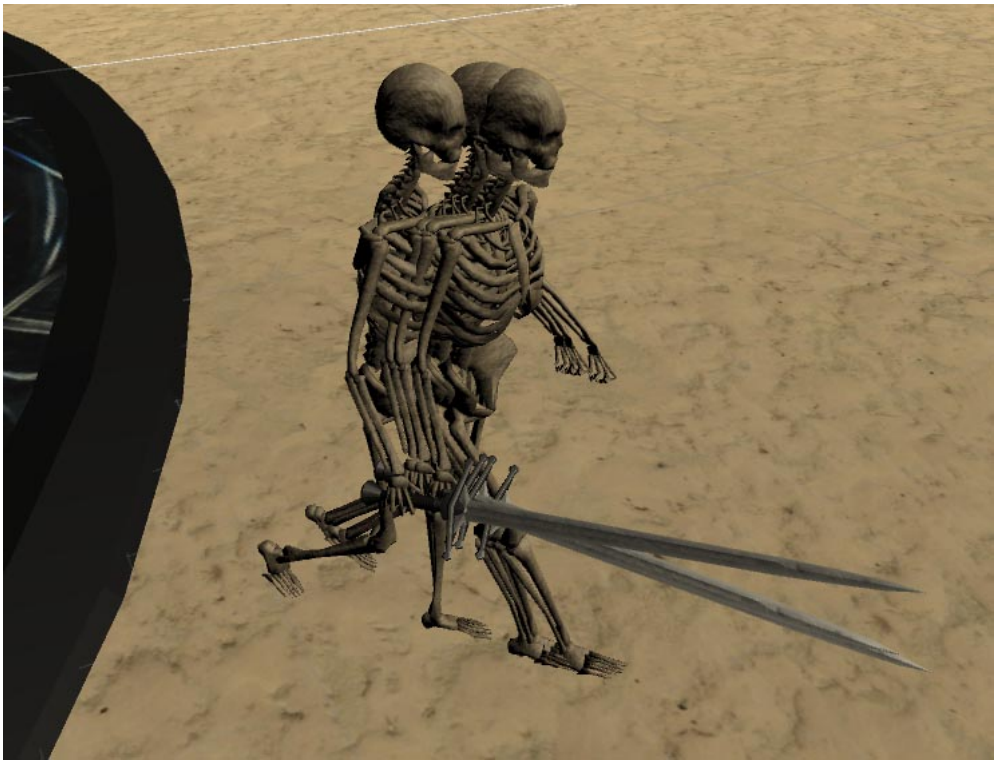
παρά την μεγάλη γκάμα δωρεάν ήχων στο internet. Οι ήχοι του παιχνιδιού βρέθηκαν συγκεκριμένα από την ιστοσελίδα www.freesound.org, όπως προαναφέρθηκε και στην εισαγωγή. Ένας παράγοντας που βοήθησε σε αυτό το θέμα ήταν η επεξεργασία ήχων με την χρήση του δωρεάν και ανοικτού κώδικα προγράμματος Audacity[6].

Τέλος, ο επιθυμητός τρόπος με τον οποίο ήθελα να κινείται ο παίκτης και οι εχθροί δεν μπορούσε να υλοποιηθεί μόνο με την χρήση του Rigidbody Component της Unity και για αυτό τον λόγο δημιουργήθηκε το script RigidbodyWrapper, το οποίο πετυχαίνει το επιθυμητό αποτέλεσμα. Η επιθυμητή κίνηση που εφαρμόστηκε ήταν το διάνυσμα της κίνησης του παίκτη να είναι ίσο με ένα σταθερό διάνυσμα ταχύτητας βασισμένο στην ταχύτητα του παίκτη συν την ταχύτητα που παράγεται εξαιτίας των εξωτερικών δυνάμεων που ασκούνται στον παίκτη.

6.3 Πιθανές Βελτιώσεις

Οι πιθανές βελτιώσεις αφορούν μερικές σκέψεις έτσι ώστε να παιχνίδι να γίνει καλύτερο από το τωρινό στάδιο. Αυτές είτε έχουν σκοπό να προσθέσουν κάτι επιπλέον στο παιχνίδι, είτε να διορθώσουν και βελτιώσουν κάτι που υπάρχει ήδη.

Μία πιθανή βελτίωση είναι η προσθήκη ενός μικρού σχετικά Capsule Collider στους εχθρούς, τα οποία θα ανήκουν αποκλειστικά σε ένα δικό τους layer. Δηλαδή η φυσική θα λειτουργεί μόνο μεταξύ τους. Αυτό θα βελτιώνει το εξής πρόβλημα: όταν πολλοί εχθροί κυνηγάνε έναν παίκτη, τότε μετά από λίγο συγχωνεύονται αφού ακολουθούν την ίδια πορεία. Επιπλέον, για τους εχθρούς που είναι όμοιοι μεταξύ τους, η συγχώνευση αυτή έχει ως αποτέλεσμα κάποιες στιγμές να φαίνεται ότι είναι ένας εχθρός σε ένα σημείο, ενώ είναι πολλοί. Σημειώνεται ότι τα βασικά collider των εχθρών δεν αλληλεπιδρούν μεταξύ τους, διότι αυτό, λόγο του μεγέθους των collider αυτών θα έκανε την κίνηση του ενός να επηρεάζει σε μεγάλο βαθμό την κίνηση του άλλου, μπλοκάροντας τον δρόμο τους. Στο Σχήμα 34 φαίνεται η συγχώνευση αυτή.



Σχήμα 34: Merged Enemies

Άλλη βελτίωση θα μπορούσε να είναι να αυξηθούν τα αντικείμενα τα οποία μπορούν να πέσουν όταν πεθάνει κάποιος εχθρός. Στο τωρινό παιχνίδι υπάρχει μόνο το HealthKit το οποίο αναπληρώνει μερικώς την ζωή του παίκτη. Αυτά θα μπορούσαν είτε να δίνουν stats στον παίκτη, είτε να κάνουν κάποια άλλη ξεχωριστή ενέργεια μέσα στον παιχνίδι.

Άλλες πιο αφηρημένες ιδέες για βελτίωση είναι η προσθήκη περισσοτέρων wave, περισσότερων special spell και η δημιουργία κάποιου Final Boss μετά το τελευταίο wave. Επίσης, θα πρόσθετε μεγαλύτερο ενδιαφέρον η προσθήκη μιας ξεχωριστής συμπεριφοράς για κάθε εχθρό, αφού η βασική λογική με την οποία οι εχθροί κυνηγάνε και επιτίθενται κατά του παίκτη είναι η ίδια. Αυτό που διαφέρει είναι ο τρόπος επίθεσης του κάθε παίκτη και η διαφοροποίηση των τιμών σε κοινές μεταβλητές που χρησιμοποιεί το MobBehaviour script.

6.4 Αρχεία που χρησιμοποιήθηκαν

Πίνακας 1: Μοντέλα που χρησιμοποιήθηκαν

Model	Model Source
ElfArcher	http://opengameart.org/content/elf-game-ready-and-animated
Goblin	https://www.assetstore.unity3d.com/en/#!/content/12131
Golem	https://www.assetstore.unity3d.com/en/#!/content/33260
LavaGolem	http://opengameart.org/content/fireicestone-golem
LavaCreature	http://opengameart.org/content/lava-spawn-animated
Skeleton	https://www.assetstore.unity3d.com/en/#!/content/30659
Spider	https://www.assetstore.unity3d.com/en/#!/content/11869
Werewolf	http://opengameart.org/content/warwolf
Barrels	https://www.assetstore.unity3d.com/en/#!/content/591
Trees	https://www.assetstore.unity3d.com/en/#!/content/54724
Gate rocks	https://www.assetstore.unity3d.com/en/#!/content/6947
Fence rocks	https://www.assetstore.unity3d.com/en/#!/content/71569
Treasure chest	https://www.assetstore.unity3d.com/en/#!/content/72498
Werewolf dagger	https://www.assetstore.unity3d.com/en/#!/content/17203
Skeleton sword	https://www.assetstore.unity3d.com/en/#!/content/17203
Menu Skybox	https://www.assetstore.unity3d.com/en/#!/content/25117
Arrow	http://opengameart.org/content/crossbow-1
Portal Particle	https://www.assetstore.unity3d.com/en/#!/content/72399
BulletStorm Particle	https://www.assetstore.unity3d.com/en/#!/content/72399
PlasmaShield Particle	https://www.assetstore.unity3d.com/en/#!/content/72399
GroundShock particle	https://www.assetstore.unity3d.com/en/#!/content/72399
Domestication particle	https://www.assetstore.unity3d.com/en/#!/content/68246
Buff particle	https://www.assetstore.unity3d.com/en/#!/content/68246

Πίνακας 2: Ήχοι που χρησιμοποιήθηκαν

Sound	Sound Source
Gravity	http://freesound.org/people/shinshi/sounds/192192/
BulletStorm	https://www.freesound.org/people/klangfabrik/sounds/232936/
PlasmaField	https://www.freesound.org/people/Apenguin73/sounds/351812/
Hit Sound	https://www.freesound.org/people/moca/sounds/49040/
GroundShock	https://www.freesound.org/people/bareform/sounds/218721/
Teleport	http://freesound.org/people/FlechaBr/sounds/340159/
Golem Death	https://www.freesound.org/people/Bernuy/sounds/268506/
LavaGolem Death	https://www.freesound.org/people/Kirat/sounds/158687/
LavaCreature Death	https://www.freesound.org/people/malexmedia/sounds/31811/
ElfArcher Death	https://www.freesound.org/people/11linda/sounds/168815/
Werewolf Death	https://www.freesound.org/people/SkinnySoundGuy/sounds/172004/
Goblin Death	https://www.freesound.org/people/spookymodem/sounds/202100/
Skeleton Death	https://www.freesound.org/people/Deganoth/sounds/348698/
Spider Death	https://www.freesound.org/people/DrMinky/sounds/174436/
Merlin Death	https://www.freesound.org/people/ecfike/sounds/154554/
Orc Death	https://www.freesound.org/people/Timbre/sounds/86242/
Golem Attack1	https://www.freesound.org/people/Bernuy/sounds/268506/
Golem Attack2	https://www.freesound.org/people/Bernuy/sounds/268506/
Werewolf Attack1	https://www.freesound.org/people/Smullen93/sounds/340333/
Werewolf Attack2	https://www.freesound.org/people/Smullen93/sounds/340333/
Werewolf Attack3	https://www.freesound.org/people/Smullen93/sounds/340333/
Goblin Attack1	https://www.freesound.org/people/-sihiL/sounds/213846/
Goblin Attack2	https://www.freesound.org/people/-sihiL/sounds/213846/
Goblin Attack3	https://www.freesound.org/people/-sihiL/sounds/213846/
Skeleton Attack	https://www.freesound.org/people/qubodup/sounds/184422/
Spider Attack1	https://www.freesound.org/people/Timbre/sounds/100905/
Spider Attack2	https://www.freesound.org/people/Timbre/sounds/100905/
Spider Attack3	https://www.freesound.org/people/Timbre/sounds/100905/
Menu Click	https://www.freesound.org/people/medetix/sounds/177912/
Lavabeam	https://www.freesound.org/people/Apenguin73/sounds/351812/
Fireball Boom	http://freesound.org/people/Werra/sounds/244394/
Fireball Cast	http://freesound.org/people/LiamG_SFX/sounds/334237/
Win Music	https://www.freesound.org/people/M-RED/sounds/52908/
Menu Music	http://www.freesfx.co.uk/download/?type=mp3&id=17658
Buff Picked	https://www.freesound.org/people/renatalmar/sounds/264981/
Heal	https://www.freesound.org/people/Timbre/sounds/221683/

Πίνακας 3: Εικόνες που χρησιμοποιήθηκαν

Image	**Image Source**
Spell Icons	http://game-icons.net
LavaBeamTexture	https://mysteriousdove.deviantart.com/art/Lava-Texture-2560-x-1440-498672313
Gravity Image	https://naruttebayo67.deviantart.com/art/ki-ball-electric-v2-327138855
Fireball Particle	https://naruttebayo67.deviantart.com/art/Fire-Ball1-423341140

Αναφορές

[1] (2017), [Online]. Available: https://unity3d.com/.

[2] (2017), [Online]. Available: www.freesound.org.

[3] (2017), [Online]. Available: https://docs.unity3d.com/Manual/.

[4] J. Hocking et al., Unity in action. Manning Publications, 2015.

[5] (2017), [Online]. Available: http://answers.unity3d.com/index.html.

[6] (2017), [Online]. Available: http://www.audacityteam.org/.