ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΙΟΥΡΓΙΑ ΠΑΙΧΝΙΔΙΟΥ ΕΠΑΥΞΗΜΕΝΗΣ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑΣ

Του φοιτητή
Αβράμη Κωνσταντίνου
Αρ. Μητρώου:134057

Επιβλέπων
Κεραμόπουλος Ευκλείδης
Αναπληρωτής Καθηγητής

Θεσσαλονίκη 2022

Τίτλος Δ.Ε. Δημιουργία παιχνιδιού επαυξημένης πραγματικότητας
Κωδικός Δ.Ε. 21322
Ονοματεπώνυμο φοιτητή. Αβράμης Κωνσταντίνος
Ονοματεπώνυμο εισηγητή. Κεραμόπουλος Ευκλείδης
Ημερομηνία ανάληψης Δ.Ε. 05-10-2021
Ημερομηνία περάτωσης Δ.Ε. 30-01-2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Η βιομηχανία των βιντεοπαιχνιδιών είναι μια από τις ταχύτερα αναπτυσσόμενες βιομηχανίες στον χώρο της ψυχαγωγίας . Η συνεχής αναβάθμιση των ηλεκτρονικών υπολογιστών αποτέλεσε πρόσφορο έδαφος για να αναπτυχθεί η εν λόγω βιομηχανία σε σημείο να ξεπεράσει σε ετήσια έσοδα τις βιομηχανίες της μουσικής και των ταινιών[1–3] οι οποίες μέχρι και πριν μερικά χρόνια είχαν την πρωτιά σε αυτόν τον τομέα. Πέρα από την εκρηκτική εξάπλωση των βιντεοπαιχνιδιών στο ευρύ κοινό αξίζει να σημειωθεί πως ανάλογα εντυπωσιακή ήταν - και εξακολουθεί να είναι - η αύξηση της πολυπλοκότητας τους καθώς αυτά εξελίχθηκαν από απλά κινούμενα δισδιάστατα σχήματα σε τρισδιάστατους κόσμους. Όπως ήταν αναμενόμενο, παράλληλα με την εξέλιξη των βιντεοπαιχνιδιών και με ολοένα περισσότερους ανθρώπους να τα επιλέγουν ως ασχολία, οι ανάγκες για καινοτόμες ιδέες και νέες τεχνολογίες στον χώρο αυξάνονται συνεχώς. Μία από τις τεχνολογίες που εισήχθησαν τα τελευταία χρόνια στα βιντεοπαιχνίδια με σκοπό να ικανοποιήσει τις προαναφερθείσες ανάγκες και παρουσιάζει ιδιαίτερο ενδιαφέρον είναι αυτή της εκτεταμένης πραγματικότητας. Ο όρος εκτεταμένη πραγματικότητα περιέχει τις έννοιες, εικονική, μικτή και επαυξημένη πραγματικότητα οι οποίες θα αναλυθούν λεπτομερώς στην συνέχεια με την επαυξημένη πραγματικότητα να αποτελεί μεγάλο μέρος της παρούσας διπλωματικής εργασίας.

Περίληψη

Η παρακάτω διπλωματική εργασία παρουσιάζει την πορεία σχεδιασμού και ανάπτυξης ενός βιντεοπαιχνιδιού στρατηγικής σε συνδυασμό με την τεχνολογία της επαυξημένης πραγματικότητας. Οι στόχοι της εργασίας είναι δύο. Ο πρώτος είναι η δημιουργία ενός πρωτότυπου παιχνιδιού το οποίο επιδιώκει να ψυχαγωγήσει τον χρήστη παρουσιάζοντας μια πνευματική πρόκληση. Ο δεύτερος είναι να εξερευνήσει τα σύγχρονα εργαλεία ανάπτυξης βιντεοπαιχνιδιών και εφαρμογών επαυξημένης πραγματικότητας. Το παρόν σύγγραμμα χωρίζεται σε τρείς κύριες θεματικές ενότητες. Η πρώτη ενότητα εστιάζει στις τεχνολογίες εκτεταμένης πραγματικότητας και την ενσωμάτωση τους στα βιντεοπαιχνίδια. Η δεύτερη ενότητα ασχολείται με τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη του παιχνιδιού καθώς και τις ρυθμίσεις που έγιναν προκειμένου να στηθεί το κατάλληλο περιβάλλον εργασίας. Η τρίτη ενότητα της εργασίας περιλαμβάνει την παρουσίαση του παιχνιδιού και την τεχνική ανάλυση των επιμέρους στοιχείων του. Καταληκτικά στο επίλογο γίνεται μια γενική αξιολόγηση της χρήσης τεχνολογιών εκτεταμένης πραγματικότητας στα βιντεοπαιχνίδια.

Λέξεις κλειδιά : Εκτεταμένη πραγματικότητα, επαυξημένη πραγματικότητα, εικονική πραγματικότητα, μικτή πραγματικότητα, Unity, Vuforia Engine, βιντεοπαιχνίδια.

«Development of an augmented reality game»

Avramis Konstantinos

Abstract

The following paper presents the design and development course of a strategy video game combined with augmented reality technology. This paper has two main goals. The first one is the creation of an innovative game aiming to entertain the user by presenting a brain challenge. The second goal is to explore the modern development tools of video games and augmented reality applications. This paper consists of three main segments. The first segment focuses on the extended reality technologies and their integration into video games. The second segment presents the tools that were used during development and how they were tuned in order to create the necessary framework. The third segment contains the presentation of the game and the technical analysis of the individual parts it consists of. Ultimately there will be a general review regarding the use of extended reality technologies in video games.

Keywords: Extended reality, augmented reality, virtual reality, mixed reality, Unity, Vuforia Engine, video games

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες σε όλους εκείνους που με βοήθησαν στο να εκπονήσω την διπλωματική μου εργασία.

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή κύριο Ευκλείδη Κεραμόπουλο για την υπομονή και την καθοδήγηση του καθώς επίσης και για το ιδιαίτερα ενδιαφέρον θέμα της εργασίας το οποίο είχα την τύχη να μελετήσω.

Θα ήθελα επίσης να ευχαριστήσω την οικογένεια και τους φίλους μου για την στήριξη που μου παρείχαν κατά την διάρκεια των σπουδών μου.

Ακόμη ένα εγκάρδιο ευχαριστώ στην Ιωάννα Πατσή για τον ιδιαίτερο ρόλο και την συνεισφορά της τα τελευταία χρόνια.

Ένα μεγάλο ευχαριστώ στην Μαρία Ζαφειροπούλου και την Δήμητρα Αναστασιάδου για την πολύτιμη βοήθεια τους.

Τέλος θα ήθελα να ευχαριστήσω τα μέλη της εξεταστικής επιτροπής για τον χρόνο που αφιέρωσαν στην αξιολόγηση της παρούσας εργασίας.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
VR	Virtual Reality
AR	Augmented Reality
MR	Mixed Reality
AV	Augmented Virtuality
vs	Versus
UI	User Interface
HMD	Head Mounted Display

Κεφάλαιο 1ο: Εισαγωγή

Τα τελευταία χρόνια οι τεχνολογίες εκτεταμένης πραγματικότητας βρίσκουν όλο και συχνότερη χρήση στα σύγχρονα βιντεοπαιχνίδια. Όντας ακόμα σε πολύ πρώιμο στάδιο τα παιχνίδια τα οποία βασίζονται σε αυτές τις τεχνολογίες δεν προβλέπεται να αντικαταστήσουν σύντομα τα συμβατικά. Ωστόσο έχοντας την δυνατότητα να προσφέρουν πληθώρα νέων εμπειριών αποτελούν κύριο ενδιαφέρον τόσο για τους χρήστες όσο και για τους δημιουργούς.

Ο όρος εκτεταμένη πραγματικότητα χρησιμοποιείται ως όρος ομπρέλα για την αναφορά στις τεχνολογίες της εικονικής, επαυξημένης και μικτής πραγματικότητας[4]. Οι τεχνολογίες της επαυξημένης και μικτής πραγματικότητας αποσκοπούν κυρίως στον συνδυασμό του πραγματικού κόσμου με ψηφιακά αντικείμενα σε αντίθεση με αυτή της εικονικής πραγματικότητας της οποίας στόχος είναι η εκ νέου δημιουργία ενός πλήρως ψηφιακού περιβάλλοντος.

Η εφαρμογή Orbs είναι ένα παιχνίδι επαυξημένης πραγματικότητας το οποίο δημιουργήθηκε προκειμένου να εξυπηρετήσει τους σκοπούς της παρούσας διπλωματικής εργασίας. Για την ανάπτυξη του παιχνιδιού Orbs χρησιμοποιήθηκε η πλατφόρμα Unity σε συνδυασμό με την μηχανή Vuforia για το μέρος της επαυξημένης πραγματικότητας.

Το παρόν σύγγραμμα χωρίζεται σε εννέα κεφάλαια και το κάθε κεφάλαιο σε επιμέρους ενότητες.

Το πρώτο κεφάλαιο «Εισαγωγή», περιέχει τις βασικές έννοιες οι οποίες απασχολούν την διπλωματική εργασία και συστήνει στον αναγνώστη την εφαρμογή που αναπτύχθηκε στα πλαίσια αυτής.

Το δεύτερο κεφάλαιο «Εκτεταμένη πραγματικότητα και βιντεοπαιχνίδια» αναλύει τις τεχνολογίες της εκτεταμένης πραγματικότητας καθώς επίσης και την εφαρμογή τους στα σύγχρονα βιντεοπαιχνίδια μέσα από δημοφιλή παραδείγματα.

Στο τρίτο κεφάλαιο «Εργαλεία ανάπτυξης» θα παρουσιαστούν λεπτομερώς η πλατφόρμα ανάπτυξης παιχνιδιών Unity και η μηχανή επαυξημένης πραγματικότητας Vuforia. Επιπρόσθετα θα γίνει εκτενής αναφορά στις απαραίτητες ρυθμίσεις και πρόσθετα τα οποία χρειάστηκαν κατά την ανάπτυξη του παιχνιδιού.

Το τέταρτο κεφάλαιο «Το παιχνίδι Orbs» περιέχει μια λεπτομερή επεξήγηση των κανόνων του παιχνιδιού, των επιμέρους στοιχείων του καθώς επίσης και του τρόπου παιξίματος .

Το πέμπτο κεφάλαιο «Δομή εφαρμογής» περιγράφει την δομή της εφαρμογής και επεξηγεί τον ρόλο που κατέχουν τα επιμέρους στοιχεία του προγράμματος.

Το έκτο κεφάλαιο «Λειτουργικότητα και κώδικας» αναλύει των κώδικα της εφαρμογής και την λειτουργικότητα που παρέχουν οι διάφορες μέθοδοι και μεταβλητές.

Το έβδομο κεφάλαιο «Χρήση εφαρμογής» περιγράφει την ροή χρήσης της εφαρμογής και την αλληλεπίδραση της με τον χρήστη.

Το όγδοο κεφάλαιο «Φάσεις ανάπτυξης» περιγράφει τα διάφορα στάδια ανάπτυξης της εφαρμογής Orbs σε χρονολογική σειρά.

Το ένατο και τελευταίο κεφάλαιο «Επίλογος και συμπεράσματα» δίνει μια αξιολόγηση για την τρέχουσα κατάσταση των τεχνολογιών εκτεταμένης πραγματικότητας στα βιντεοπαιχνίδια και προτάσεις για μελλοντική βελτίωση της εφαρμογής.

Κεφάλαιο 2ο: Εκτεταμένη πραγματικότητα και βιντεοπαιχνίδια

2.1 Εισαγωγή

Αυτό το κεφάλαιο επιδιώκει να θέσει ένα θεωρητικό υπόβαθρο γύρω από τις τεχνολογίες εκτεταμένης πραγματικότητας. Η εφαρμογή των τεχνολογιών εκτεταμένης πραγματικότητας εκτείνεται σε πληθώρα διαφορετικών τομέων και αντικειμένων, ωστόσο η κύρια εστίαση θα γίνει στην ενσωμάτωση τους στα βιντεοπαιχνίδια.

Ο όρος εκτεταμένη πραγματικότητα ή XR (Extended Reality) χρησιμοποιείται ως όρος ομπρέλα για την αναφορά στις τεχνολογίες της εικονικής πραγματικότητας ή VR (Virtual Reality), της Επαυξημένης Πραγματικότητας ή AR (Augmented Reality) και της Μικτής Πραγματικότητας ή MR (Mixed/Merged Reality) . Όλες οι επιμέρους τεχνολογίες της εκτεταμένης πραγματικότητας στοχεύουν στην δημιουργία ενός νέου περιβάλλοντος με το οποίο αλληλεπιδρά ο χρήστης χρησιμοποιώντας ειδικό λογισμικό και εξοπλισμό. Ο διαχωρισμός ανάμεσα στις τεχνολογίες της εκτεταμένης πραγματικότητας βασίζεται στον τύπο του περιβάλλοντος το οποίο παράγεται και στο επίπεδο αλληλεπίδρασης του χρήστη με αυτό.

2.2 Εικονική Πραγματικότητα

Η εικονική πραγματικότητα μπορεί να οριστεί ως η χρήση ψηφιακών υπολογιστών και άλλου εξειδικευμένου υλικού για την δημιουργία μίας προσομοίωσης εναλλακτικού κόσμου ή περιβάλλοντος σε πραγματικό χρόνο το οποίο είναι πιστευτό από τους χρήστες. Με άλλα λόγια η εικονική πραγματικότητα είναι η αμιγώς ψηφιακή προσομοίωση ενός πραγματικού η φανταστικού περιβάλλοντος με το οποίο ο χρήστης μπορεί να αλληλεπιδράσει μέσω ειδικά σχεδιασμένου εξοπλισμού. Η εικονική πραγματικότητα έχει ως στόχο την πλήρη εμβύθιση του χρήστη μέσα στην προσομοίωση αντικαθιστώντας τα ερεθίσματα του πραγματικού κόσμου με τεχνητά.

2.2.1 Εξοπλισμός εικονικής πραγματικότητας

Πέρα από τον υπολογιστή ο οποίος περιέχει την εφαρμογή εικονικής πραγματικότητας και μπορεί να βρίσκεται σε διάφορες μορφές όπως οικιακός υπολογιστής, κονσόλα παιχνιδιών ή φορητή συσκευή το σύνολο και η μορφή των υπολοίπων μερών του εξοπλισμού μπορεί να διαφέρει μεταξύ των διαφόρων εφαρμογών.

Το βασικότερο μέρος του εξοπλισμού για την χρήση εφαρμογών εικονικής πραγματικότητας είναι η οθόνη προσαρμοσμένη στο κεφάλι ή HMD[5] (Head Mounted Display). Όπως υποδεικνύει και η ονομασίας της η συσκευή HMD είναι μία οθόνη η οποία τοποθετείται σε πολύ κοντινή απόσταση από τα μάτια του χρήστη αποκόπτοντας τα οπτικά ερεθίσματα του πραγματικού κόσμου. Η διαφορά μίας HMD από μία κοινή οθόνη είναι η ανίχνευση της κίνησης η οποία επιτρέπει στον χρήστη να κοιτάξει μέσα στο εικονικό περιβάλλον μεταβάλλοντας την θέση ή την περιστροφή του κεφαλιού του όπως θα έκανε και στον πραγματικό κόσμο. Η συσκευή HMD συνήθως βρίσκεται προσκολλημένη σε ένα κράνος εικονικής πραγματικότητας (VR headset) . Σε κάποιες περιπτώσεις επάνω στο VR Headset βρίσκονται επίσης εγκατεστημένα ακουστικά τρισδιάστατου ήχου ή και μικρόφωνο ειδικά για τις περιπτώσεις στις οποίες το κράνος προορίζεται για VR παιχνίδια.

Για την ανίχνευση των ενεργειών του χρήστη σε μία εφαρμογή εικονικής πραγματικότητας χρησιμοποιούνται διάφορες συσκευές εισόδου. Αυτές οι συσκευές μπορεί να είναι χειριστήρια γενικού σχεδιασμού παρόμοια με εκείνα των κόνσολών παιχνιδιών όπως PlayStation, γάντια με αισθητήρες ή άλλα αντικείμενα ειδικά σχεδιασμένα για την εκάστοτε εφαρμογή.

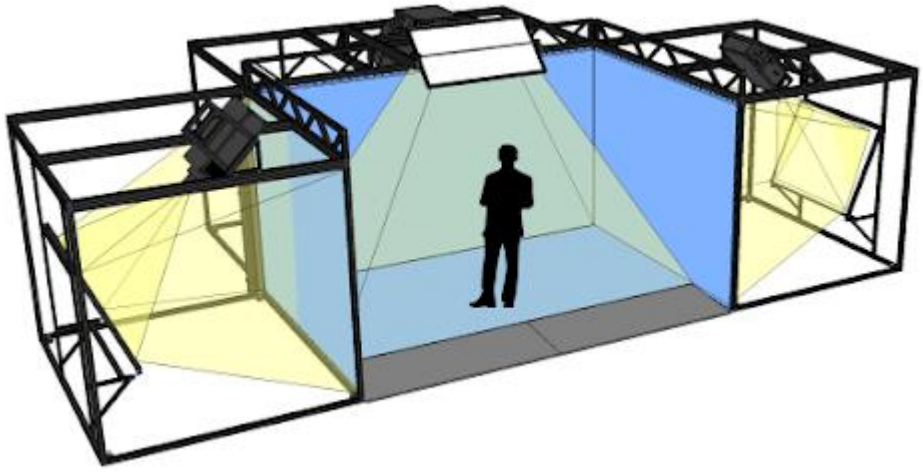
Σχήμα 2.1: VR headset και χειριστήρια

Μέσω ειδικών πανκατευθυντικών διαδρόμων (Σχήμα 2.2), απτικών στολών[6-7] οι οποίες μεταφέρουν την αίσθηση της αφής στον χρήστη (Σχήμα 2.3) και συσκευών προσομοίωσης των αισθήσεων της όσφρησης και γεύσης υπάρχει η δυνατότητα ακόμα μεγαλύτερης εμβύθισης του χρήστη στην εικονική εμπειρία . Ωστόσο τα προαναφερθέντα μέρη εξοπλισμού βρίσκονται πολύ σπανιότερα σε σχέση με τα κράνη εικονικής πραγματικότητας λόγω του μεγάλου μεγέθους, του υψηλού κόστους ή γενικότερα της μη εμπορικής βιωσιμότητας τους κατά την τρέχουσα χρονική περίοδο.

Σχήμα 2.2 : Omnidirectional treadmill

Σχήμα 2.3 : Haptic suit

Το σύστημα Cave (Cave Automatic Virtual Environment) αποτελεί μια ειδική περίπτωση εξοπλισμού εικονικής πραγματικότητας. Η ιδιαιτερότητα του συστήματος CAVE είναι ο τρόπος με τον οποίο αντικαθιστά την HMD . Αντί για την HMD το σύστημα CAVE χρησιμοποιεί ένα ολόκληρο δωμάτιο στο οποίο οι τοίχοι λειτουργούν ως οθόνες (Σχήμα 2.4). Το σύστημα CAVE επιτρέπει σε πολλούς χρήστες να βιώνουν την ίδια εικονική εμπειρία ενώ βρίσκονται στον ίδιο χώρο αντίθετα με το κράνος VR το οποίο είναι σχεδιασμένο για ατομική χρήση.



Σχήμα 2.4 : Cave system

2.3 Επαυξημένη πραγματικότητα

Επαυξημένη πραγματικότητα είναι το περιβάλλον το οποίο συνδυάζει την θέαση του πραγματικού κόσμου με την προσθήκη ψηφιακών/εικονικών στοιχείων. Τα τρία βασικά χαρακτηριστικά της τεχνολογίας AR είναι :

- Συνδυασμός του πραγματικού με το ψηφιακό
- Διάδραση σε πραγματικό χρόνο
- Λειτουργία σε 3 διαστάσεις

[8]

Στο σχήμα 2.5 φαίνεται ή υπέρθεση των ψηφιακών αντικειμένων σε μία εικόνα πραγματικού κόσμου η οποία λαμβάνεται μέσω κάμερας. Στις εφαρμογές AR ο χρήστης έχει την δυνατότητα να αλληλεπιδρά με τα ψηφιακά/εικονικά στοιχεία αλλά ο τρόπος και ο βαθμός της αλληλεπίδρασης εξαρτάται από την εκάστοτε εφαρμογή.

Σχήμα 2.5 : Υπέρθεση αντικειμένων στην AR

2.3.1 Εξοπλισμός επαυξημένης πραγματικότητας

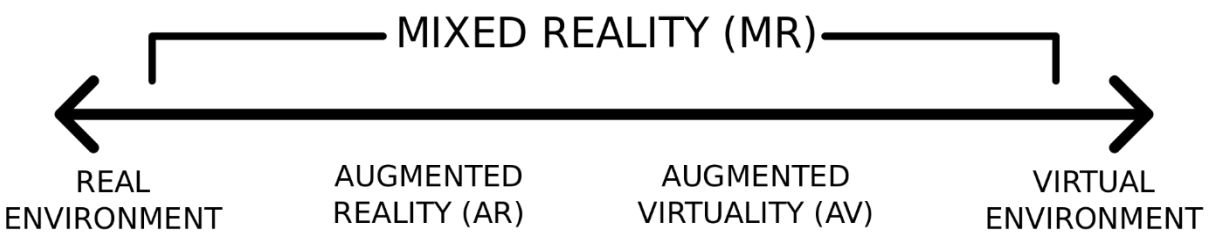
Από πλευράς συσκευών και εξοπλισμού η επαυξημένη πραγματικότητα έχει μικρότερες απαιτήσεις σε σχέση με την εικονική πραγματικότητα. Το μόνο εξάρτημα το οποίο είναι απαραίτητο πέρα από την συσκευή η οποία περιέχει την εφαρμογή AR είναι μια κάμερα επαρκούς ανάλυσης για την θέαση του πραγματικού κόσμου.

Οι πιο δημοφιλείς συσκευές για την χρήση εφαρμογών AR είναι τα smartphones και τα tablets. Αυτό οφείλεται τόσο στην ενσωματωμένη κάμερα αυτών των συσκευών όσο και στην φορητότητα την οποία προσφέρουν. Αντίστοιχη λειτουργικότητα προσφέρουν και τα γυαλιά επαυξημένης πραγματικότητας (Smart Glasses-AR Glasses) ως μια πιο πρόσφατα υλοποιημένη εναλλακτική. Θεωρητικά ένας οποιασδήποτε μορφής υπολογιστής (κονσόλα, laptop κλπ.) με μία οθόνη και μία συνδεδεμένη κάμερα μπορεί να χρησιμοποιηθεί για AR εφαρμογές έχοντας το κατάλληλο λογισμικό, η προτίμηση μεταξύ των συσκευών ορίζεται από την πρακτικότητα χρήσης.

Με την πλειοψηφία των εφαρμογών AR να στοχεύουν φορητές πλατφόρμες όπως τα smartphones οι ανάγκες για συσκευές εισόδου καλύπτονται ως επί το πλείστον από τις οθόνες αφής ωστόσο υπάρχουν περιπτώσεις στις οποίες σχεδιάζονται συσκευές εισόδου για συγκεκριμένες εφαρμογές.

2.4 Μικτή Πραγματικότητα

Η μικτή πραγματικότητα ή MR ως όρος αποτελεί ένα υπερσύνολο της επαυξημένης πραγματικότητας και αναφέρεται σε όλα τα περιβάλλοντα μέσα στο συνεχές πραγματικού-εικονικού εκτός των ακραίων καταστάσεων (πλήρως πραγματικό, πλήρως εικονικό) όπως φαίνεται στον σχήμα 2.6 [9] .



Σχήμα 2.6 : Milgram's Reality-Virtuality Continuum

Ουσιαστικά η μικτή πραγματικότητα αποτελεί συνδυασμό του πραγματικού και εικονικού κόσμου με την δυνατότητα αλληλεπίδρασης τόσο του χρήστη με τα ψηφιακά αντικείμενα όσο και των πραγματικών και ψηφιακών αντικειμένων μεταξύ τους.

2.4.1 Εξοπλισμός μικτής πραγματικότητας

Καθώς η μικτή πραγματικότητα θεωρείται εξέλιξη της επαυξημένης πραγματικότητας ο εξοπλισμός μεταξύ των δύο δεν διαφέρει σημαντικά. Ωστόσο για τις εφαρμογές μικτής πραγματικότητας φαίνεται να προτιμώνται συσκευές Smart Glasses όπως το HoloLens της Microsoft .

Σχήμα 2.7 : Microsoft HoloLens smartglasses

2.5 VR vs AR vs MR

Συνοψίζοντας, η διαφορά μεταξύ της εικονικής πραγματικότητας απέναντι στην επαυξημένη και την μικτή πραγματικότητα είναι εύκολο να εντοπιστεί καθώς στην πρώτη περίπτωση γίνεται αναφορά σε ένα αμιγώς εικονικό περιβάλλον χωρίς καμία προσθήκη από τον πραγματικό κόσμο ενώ στις υπόλοιπες γίνεται λόγος για συνδυασμό του πραγματικού κόσμου με εικονικά στοιχεία.

Η δυσκολία διαχωρισμού έρχεται μεταξύ της επαυξημένης πραγματικότητας και της μικτής πραγματικότητας. Καθώς η MR είναι υπερσύνολο της AR οποιαδήποτε εφαρμογή AR ταυτόχρονα θεωρείται και εφαρμογή MR. Ωστόσο το αντίθετο δεν ισχύει πάντα . Στο σχήμα 2.6 εμφανίζεται ο όρος επαυξημένη εικονικότητα ή AV (Augmented Virtuality) ο οποίος εκφράζει την αλληλεπίδραση των πραγματικών αντικειμένων συμπεριλαμβανομένου και του χρήστη με το παραγόμενο περιβάλλον. Επομένως ο διαχωρισμός ανάμεσα στην MR και την AR ορίζεται από τον βαθμό συμμετοχής των πραγματικών αντικειμένων στο παραγόμενο περιβάλλον.

2.6 Παιχνίδια εκτεταμένης πραγματικότητας

Σε αυτό το κεφάλαιο θα γίνει μια σύντομη αναφορά σε ορισμένους από τους πιο δημοφιλής τίτλους παιχνιδιών εκτεταμένης πραγματικότητας καθώς επίσης και στις πλατφόρμες τις οποίες τους φιλοξενούν.

2.6.1 Πλατφόρμες παιχνιδιών εκτεταμένης πραγματικότητας

Όπως αναφέρθηκε και προηγουμένως παιχνίδια εκτεταμένης πραγματικότητας μπορούν να βρεθούν τόσο σε σταθερές συσκευές όσο και σε φορητές συσκευές.

Στις σταθερές συσκευές ανήκουν οι κονσόλες παιχνιδιών και οι οικιακοί υπολογιστές. Η πιο δημοφιλής κονσόλα για σύγχρονα παιχνίδια XR είναι το PlayStation (PSVR) . Αντίστοιχα στους οικιακούς υπολογιστές το λειτουργικό σύστημα Windows φαίνεται να προτιμάται για παιχνίδια εκτεταμένης πραγματικότητας. Στις φορητές συσκευές ανήκουν Smartphones και Tablets με λειτουργικό σύστημα Android ή iOS.

Όσον αφορά τον διαμοιρασμό των παιχνιδιών στις πλατφόρμες τα σταθερά συστήματα φαίνεται να προτιμώνται για την φιλοξενία των παιχνιδιών VR ενώ τα φορητά για τα παιχνίδια MR. Αυτό δικαιολογείται από το γεγονός ότι τα VR παιχνίδια έχουν γενικότερα υψηλότερες απαιτήσεις συστήματος οι οποίες καλύπτονται καλύτερα από τα κατά κανόνα ισχυρότερα σταθερά συστήματα ενώ ταυτόχρονα δεν δεσμεύονται από την φορητότητα και την χρήση κάμερα όπως τα παιχνίδια MR.

2.6.2 Δημοφιλή παιχνίδια εκτεταμένης πραγματικότητας

Στους πίνακες 2.1 και 2.2 αναφέρονται ορισμένοι από τους δημοφιλέστερους τίτλους XR παιχνιδιών των τελευταίων ετών[10-11] .

Τίτλος	Εταιρεία ανάπτυξης	Πλατφόρμα	Έτος κυκλοφορίας
Half Life: Alyx	Valve	PC	2020
Rez Infinite	Monstars Inc, Resonair	PSVR, Oculus, PC	2016
Astro Bot: Rescue Mission	Team Asobi	PSVR	2018
Beat Saber	Beat Games	PC, Oculus, PSVR	2018

Πίνακας 2.1: Δημοφιλή παιχνίδια VR

Τίτλος	Εταιρεία ανάπτυξης	Πλατφόρμα	Έτος κυκλοφορίας
Ingress	Niantic	Android, iOS	2013
Pokémon GO	Niantic, Nintendo, The Pokémon Company	Android, iOS	2016
Harry Potter: Wizards Unite	Niantic, Portkey Games, WB Games	Android, iOS	2019

Πίνακας 2.2: Δημοφιλή παιχνίδια MR/AR

2.7 Επίλογος

Εκ πρώτης όψεως ο διαχωρισμός μεταξύ των τεχνολογιών XR μπορεί να μην φαίνεται ιδιαίτερα σαφής σε κάποιον χωρίς την απαραίτητη εξοικείωση με τις εν λόγω τεχνολογίες. Ωστόσο μια πιο προσεκτική ματιά στις κύριες διαφορές μεταξύ αυτών των τεχνολογιών, τις περιφερειακές συσκευές και τις προτιμώμενες πλατφόρμες για την κάθε μία καθιστά αυτόν τον διαχωρισμό πολύ πιο ξεκάθαρο. Κάθε επιμέρους τεχνολογία της XR έχει την δική της γκάμα εφαρμογών όπως φαίνεται και από τον διαμοιρασμό των παιχνιδιών MR/AR και VR στις αντίστοιχες φορητές και σταθερές πλατφόρμες (πίνακες 2.1, 2.2) .

Κεφάλαιο 3ο: Εργαλεία Ανάπτυξης

3.1 Εισαγωγή

Αυτό το κεφάλαιο θα αφιερωθεί στην παρουσίαση των εργαλείων τα οποία χρησιμοποιήθηκαν για την κατασκευή της εφαρμογής Orbs. Η παρουσίαση αυτή περιλαμβάνει την αναφορά στα διάφορα προγράμματα και βιβλιοθήκες τα οποία συντέλεσαν το περιβάλλον ανάπτυξης και τις ρυθμίσεις που χρειάστηκε να γίνουν για να επιτευχθεί το επιθυμητό αποτέλεσμα. Τέλος σε αυτό το κεφάλαιο θα γίνει η περιγραφή ορισμένων βασικών διεργασιών της πλατφόρμας Unity[12] με στόχο την καλύτερη κατανόηση των επόμενων κεφαλαίων.

3.2 Μηχανές Παιχνιδιών

Μηχανή παιχνιδιών (Game Engine) χαρακτηρίζεται το λογισμικό το οποίο σχεδιάζεται με σκοπό την κατασκευή βιντεοπαιχνιδιών. Τα βασικότερα επιμέρους στοιχεία μίας μηχανής παιχνιδιών είναι :

1. Φωτοαπόδοση-Rendering
2. Μηχανή φυσικής/εντοπισμού συγκρούσεων – physics engine/collision detection.
3. Μηχανή ήχου – audio engine.
4. Τεχνητή νοημοσύνη.
5. Εργαλεία Animation.
6. Εργαλεία συγγραφής κώδικα.

Από πλευράς προσβασιμότητας οι μηχανές παιχνιδιών χωρίζονται σε δύο κατηγορίες. Η πρώτη κατηγορία είναι οι ιδιόκτητες μηχανές οι οποίες αναπτύσσονται μόνο για ενδοεταιρική χρήση και ως εκ τούτου δεν υπάρχει πρόσβαση σε αυτές από το ευρύ κοινό. Η δεύτερη κατηγορία είναι οι μηχανές εμπορικής χρήσης οι οποίες είναι διαθέσιμές στον οποιοδήποτε είτε δωρεάν είτε έναντι κάποιου χρηματικού αντιτίμου.

Από τις μηχανές εμπορικής χρήσης επιλέχθηκε η πλατφόρμα Unity για την ανάπτυξη της εφαρμογής Orbs με γνώμονα την δημοτικότητα της, την ευκολία εκμάθησης και την ευελιξία την οποία παρέχει η δωρεάν έκδοση της.

3.3 Πλατφόρμα Unity

Η εγκατάσταση της μηχανής Unity έγινε μέσω του προγράμματος Unity Hub . Το πρόγραμμα Unity Hub είναι μία αυτόνομη εφαρμογή η οποία διευκολύνει σημαντικά την εγκατάσταση και την παραμετροποίηση της μηχανής Unity. Πιο συγκεκριμένα μέσω της εφαρμογής Unity Hub έγινε εφικτή η εγκατάσταση πολλαπλών εκδόσεων της μηχανής Unity και προστέθηκε με σχετική ευκολία η υποστήριξη για την android πλατφόρμα στην οποία προβλεπόταν να τρέξει η εφαρμογή Orbs.

Όπως έχει αναφερθεί προηγουμένως η εφαρμογή Orbs κάνει χρήση της μηχανής Vuforia για τις λειτουργίες της επαυξημένης πραγματικότητας γεγονός το οποίο δυσκόλεψε σημαντικά την επιλογή του κατάλληλου συνδυασμού εκδόσεων Unity και Vuforia . Μετά από εκτενείς δοκιμές επιλέχθηκε η έκδοση 2019.4.18f1 για την μηχανή Unity και η έκδοση 9.6.4 για την μηχανή Vuforia για 2 λόγους. Ο πρώτος λόγος είναι η συμβατότητα μεταξύ των 2 εκδόσεων. Ο δεύτερος λόγος είναι η έλλειψη τεκμηρίωσης και εκπαιδευτικού υλικού στο διαδίκτυο για τις πιο σύγχρονες εκδόσεις γεγονός το οποίο έκανε την χρήση τους δύσκολη και χρονοβόρα.

3.4 Visual Studio

Η πλατφόρμα Unity υποστηρίζει διάφορες γλώσσες προγραμματισμού ωστόσο η προτεινόμενη επιλογή είναι η γλώσσα C# η οποία και χρησιμοποιήθηκε στην εφαρμογή Orbs. Για την συγγραφή του κώδικα της εφαρμογής επιλέχθηκε το Visual Studio 2019 λόγω της ενσωματωμένης υποστήριξης που παρέχει για το Unity.

3.5 Vuforia

Η μηχανή Vuforia [13] είναι ένα σύνολο εργαλείων SDK (Software Development Kit) για την ανάπτυξη εφαρμογών επαυξημένης πραγματικότητας. Τα βήματα εγκατάστασης και χρήσης της μηχανής Vuforia θα αναφερθούν σε επόμενη ενότητα.

3.6 Ρυθμίσεις περιβάλλοντος εργασίας

Σε αυτή την ενότητα θα παρουσιαστεί η διαδικασία εγκατάστασης και οι ρυθμίσεις των εργαλείων που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής Orbs.

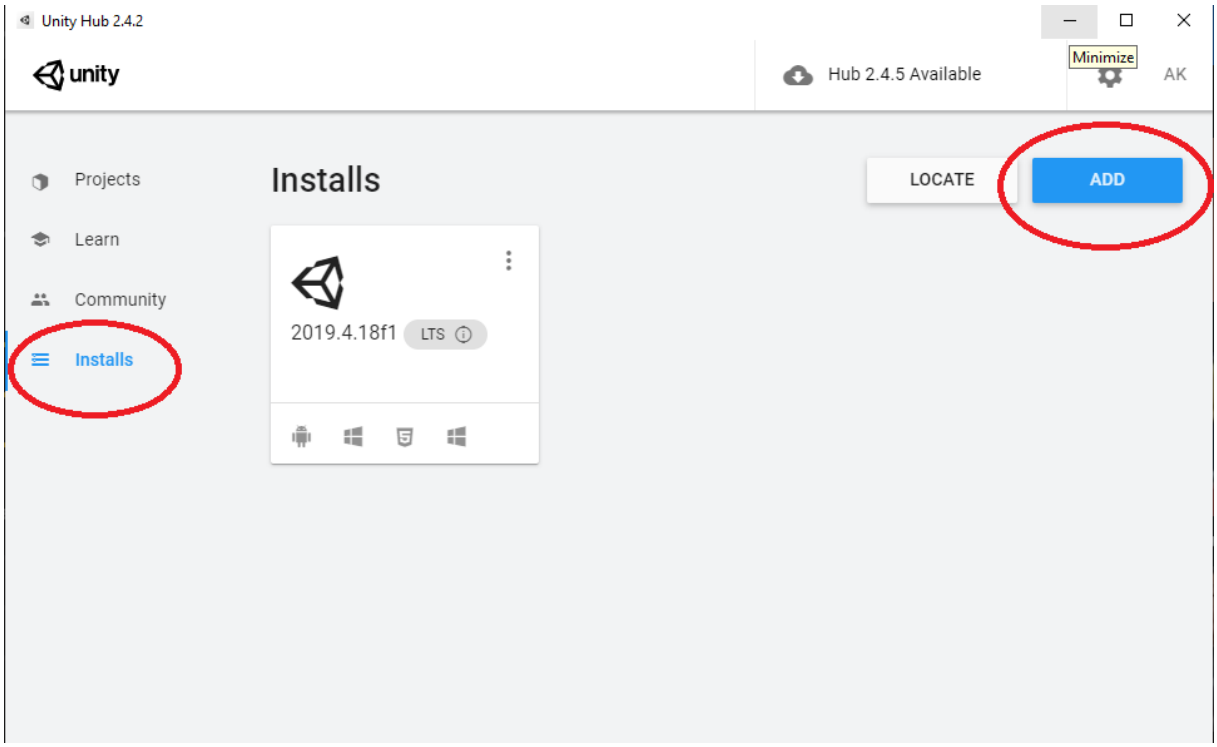
3.6.1 Εγκατάσταση Unity Hub - Unity

Το πρόγραμμα Unity Hub είναι διαθέσιμο στην σελίδα www.unity3d.com. Για την χρήση του Unity και Unity Hub προτείνεται η δημιουργία λογαριασμού unity ή η σύνδεση με λογαριασμό google, Facebook ή Apple.

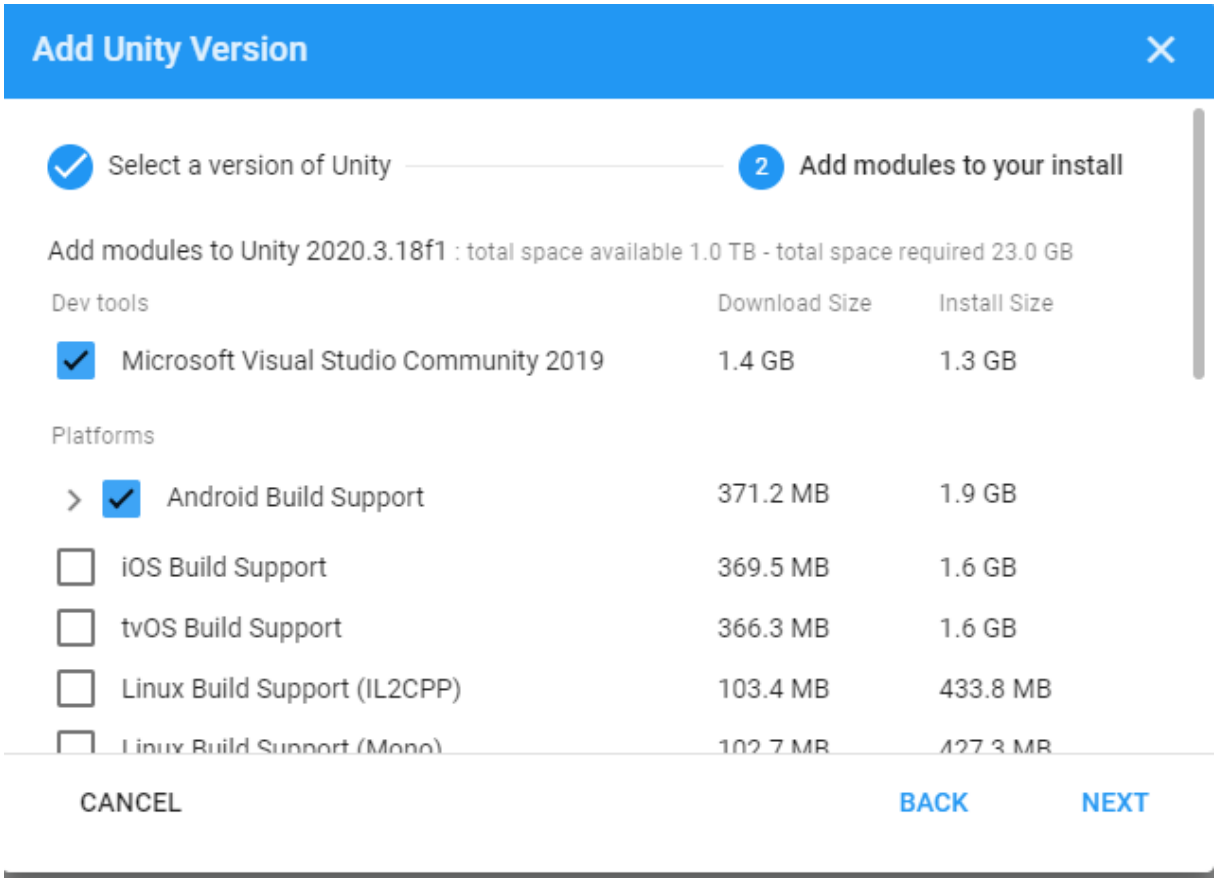
Η εγκατάσταση του Unity έγινε μέσω του Unity Hub με τα παρακάτω βήματα[14] .

1. Κλικ στην επιλογή Installs
2. Κλικ στο κουμπί ADD.
3. Επιλογή έκδοσης Unity
4. Κλικ στο κουμπί NEXT
5. Επιλογή Visual Studio 2019, Android Build Support, WebGl Support, Windows Build Support, Universal Windows Platform Build Support.
6. Αποδοχή των όρων χρήσης.
7. Ολοκλήρωση εγκατάστασης

Οι επιλογές του βήματος 5 διαφέρουν ανάλογα με την πλατφόρμα στην οποία προβλέπεται να τρέξει η εφαρμογή.



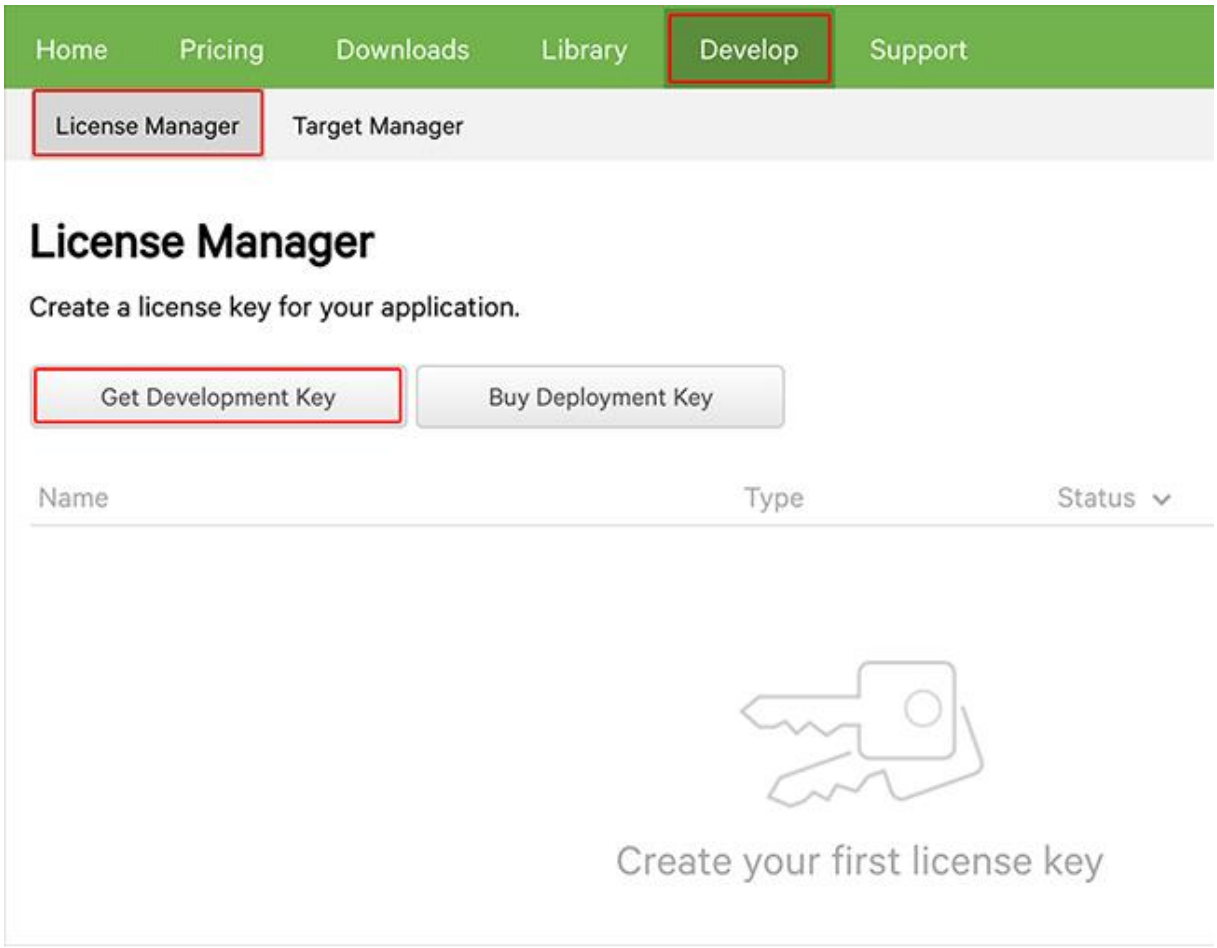
Σχήμα 3.1 : Unity Hub



Σχήμα 3.2 : Unity Hub modules

3.6.2 Εγκατάσταση Vuforia

Η εγκατάσταση και χρήση την μηχανής Vuforia [15] απαιτεί την δημιουργία λογαριασμού στην ιστοσελίδα developer.vuforia.com. Η δημιουργία λογαριασμού αποσκοπεί στην δημιουργία ενός development key το οποίο λειτουργεί ως άδεια χρήσης της μηχανής Vuforia μέσα στο Unity. Η έκδοση του development key γίνεται από την ιστοσελίδα developer.vuforia.com → Develop →License Manager → Get Development Key . Στην συνέχεια εισάγεται το όνομα της εφαρμογής για την οποία θα εκδοθεί το development key.



Σχήμα 3.3 : Vuforia Development Key

Η εγκατάσταση του πακέτου της μηχανής Vuforia μπορεί να γίνει με 2 τρόπους.

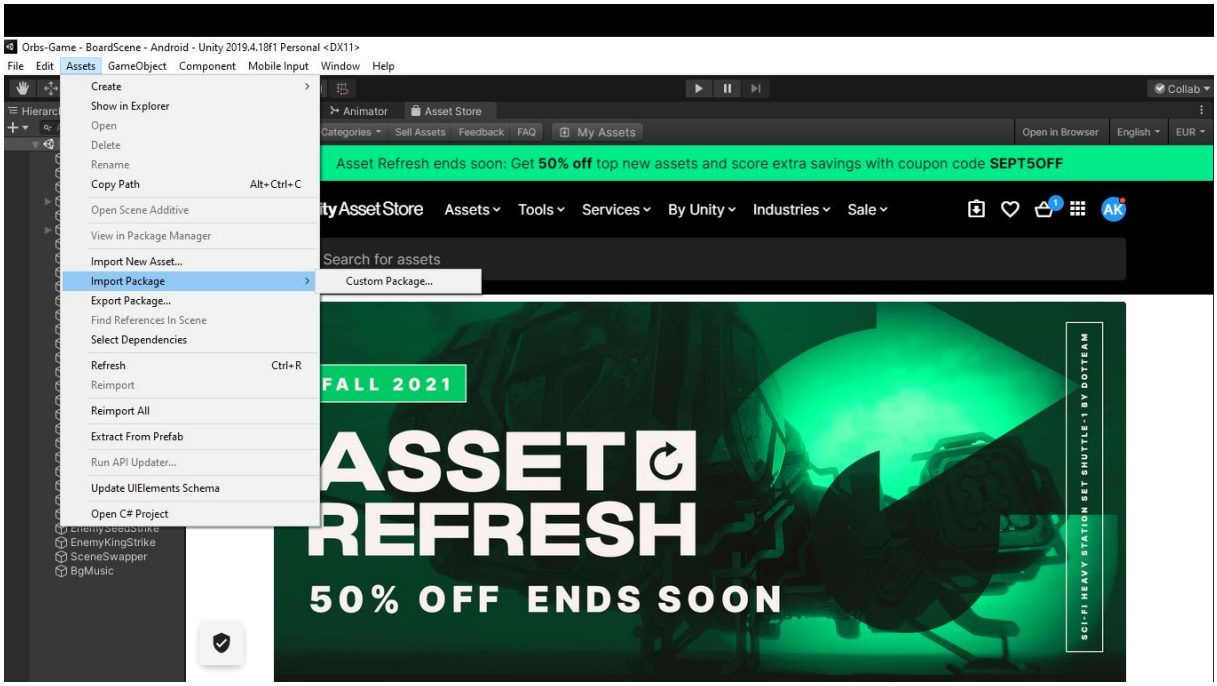
Τρόπος 1

Από το παράθυρο του Asset Store στο Unity εισάγουμε τον όρο Vuforia Engine στην μπάρα αναζήτησης, εντοπίζουμε το κατάλληλο πακέτο στα αποτελέσματα της αναζήτησης και στην συνέχεια Download → Install → και Import στο παράθυρο του Package Manager.

Τρόπος 2

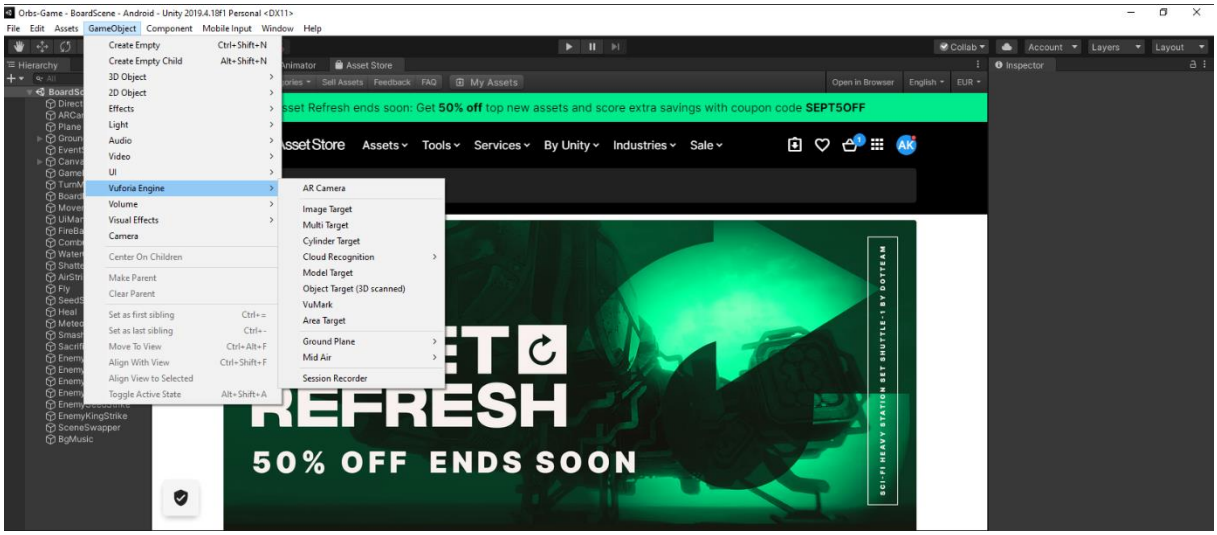
Κατεβάζουμε την επιθυμητή έκδοση της μηχανής Vuforia από την ιστοσελίδα https://developer.vuforia.com/downloads/sdk . Στην συνέχεια από το Unity επιλέγουμε την καρτέλα

Assets → Import Package → Custom Package και επιλέγουμε το αρχείο που κατεβάσαμε μέσω του file explorer.Στην συνέχεια πατάμε import στο παράθυρο του Package Manager για να ολοκληρωθεί η εγκατάσταση της μηχανής Vuforia.



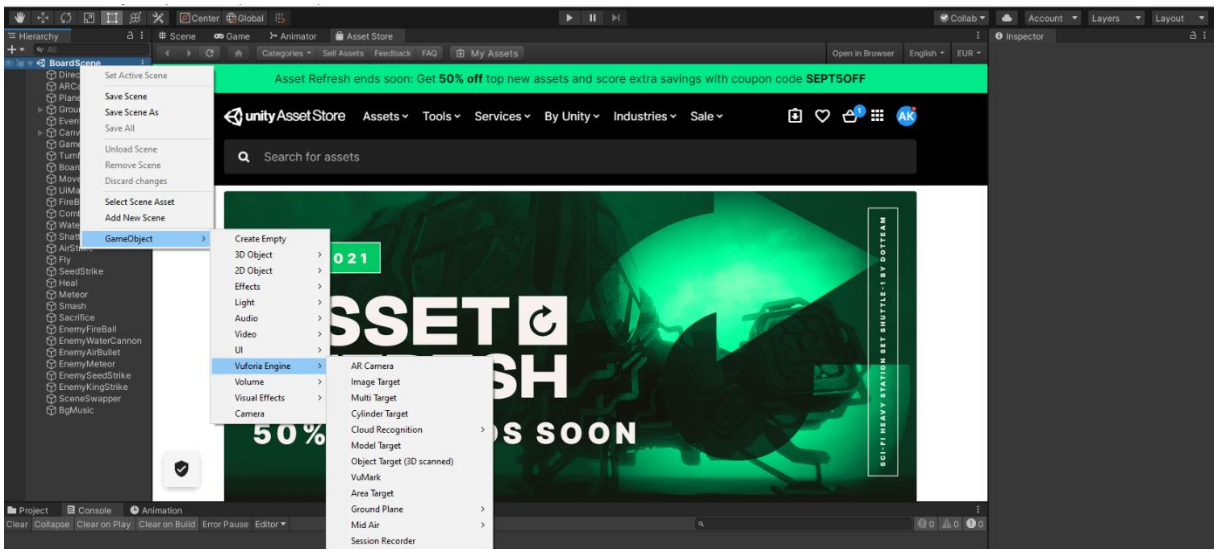
Σχήμα 3.4 : Εισαγωγή πακέτων στο Unity

Μετά την επιτυχή εγκατάσταση της μηχανής Vuforia στην καρτέλα GameObject του Unity εμφανίζεται η επιλογή Vuforia Engine η οποία περιέχει τα αντικείμενα που σχετίζονται με την χρήση επαυξημένης πραγματικότητας στην εφαρμογή.



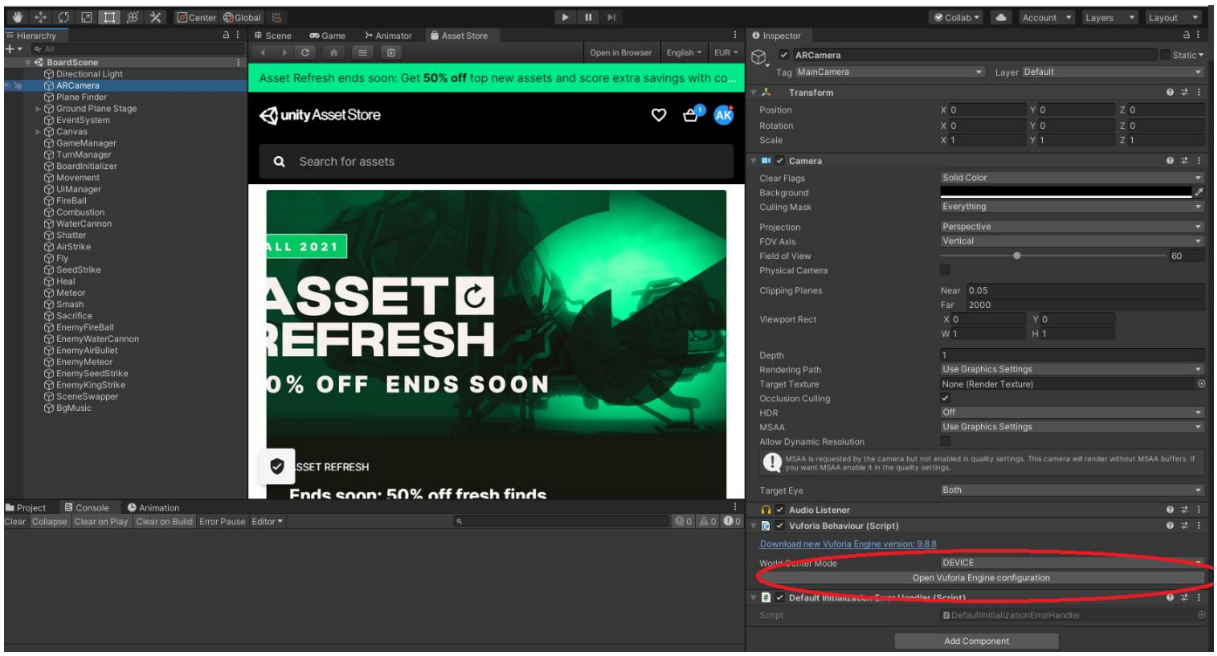
Σχήμα 3.5 : Μενού μηχανής Vuforia στο Unity

Για να ολοκληρωθεί η διαδικασία εγκατάστασης προσθέτουμε στην σκηνή ένα αντικείμενο AR Camera. Για να γίνει αυτό πατάμε δεξί κλικ στην σκηνή και επιλέγουμε GameObject → Vuforia Engine →AR Camera .



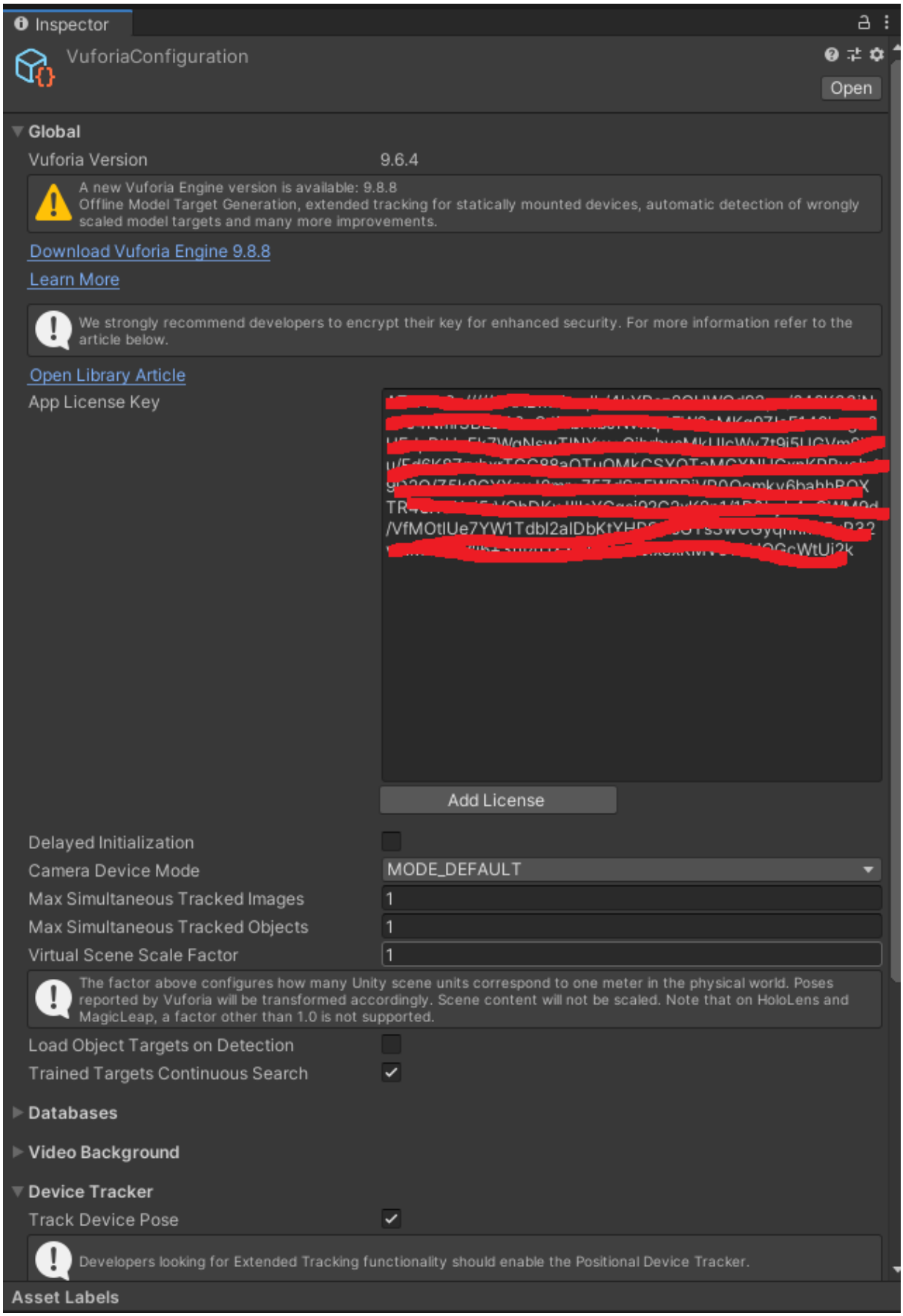
Σχήμα 3.6 : Εισαγωγή αντικειμένων της μηχανής Vuforia

Στην συνέχεια πατάμε επάνω στο αντικείμενο AR Camera και έπειτα στο κουμπί Open Vuforia Configuration για να εμφανιστεί το παράθυρο ρυθμίσεων της μηχανής Vuforia.



Σχήμα 3.7 : Vuforia configuration 1

Στο πεδίο App License Key εισάγουμε το Development Key ενεργοποιώντας την μηχανή Vuforia στην εφαρμογή.



Σχήμα 3.8 : Vuforia configuration 2

3.6.3 Image targets

Μία πολύ δημοφιλής τακτική στην ανάπτυξη εφαρμογών επαυξημένης πραγματικότητας είναι η χρήση των Image Targets. Παρά το γεγονός ότι η εφαρμογή Orbs δεν κάνει χρήση αυτής της τακτικής καλό είναι να γίνει μια σύντομη αναφορά σε αυτήν καθώς είναι ευρέως διαδεδομένη σε αντίστοιχες εφαρμογές και απαιτεί ένα επιπλέον βήμα στην διαδικασία εγκατάστασης της μηχανής Vuforia.

Εν συντομία τα Image Targets είναι εικόνες οι οποίες τοποθετούνται στον πραγματικό χώρο και συσχετίζονται με ορισμένα εικονικά αντικείμενα. Όταν η εφαρμογή ανιχνεύσει μια Image Target εικόνα μέσω της κάμερας στον πραγματικό χώρο εμφανίζει το αντίστοιχο εικονικό αντικείμενο με το οποίο έχει συσχετιστεί η εικόνα στην οθόνη.

Για να οριστεί μία εικόνα ως image target πρέπει να πληροί ορισμένες προϋποθέσεις οι οποίες αφορούν κυρίως την ευκολία ανίχνευσης της από την κάμερα. Για την χρήση αυτής της τακτικής οι εικόνες Image Targets πρέπει να τοποθετηθούν σε μία βάση δεδομένων. Η κατασκευή της βάσης γίνεται από την ιστοσελίδα https://developer.vuforia.com/vui/develop/databases επιλέγοντας Develop →Target Manager → Add Database.

Για την χρήση των εικόνων Image Targets εντός του Unity πρέπει να προστεθεί η βάση δεδομένων μέσω του Vuforia Configuration →Databases → Add Database και να προστεθούν εντός της σκηνής αντικείμενα Image Target από το μενού GameObject → Vuforia Engine →Image Target .

3.7 Βασικά στοιχεία Unity

Ο στόχος αυτής της ενότητας είναι να κάνει σύντομη αναφορά στα διάφορα στοιχεία του Unity τα οποία χρησιμοποιήθηκαν στην ανάπτυξη της εφαρμογής Orbs και το πώς αυτά λειτουργούν προκειμένου να επιτευχθεί το επιθυμητό αποτέλεσμα . Καθώς το Unity είναι μια εξαιρετικά πολύπλοκη πλατφόρμα είναι αδύνατο να καλυφθούν όλα τα στοιχεία και οι δυνατότητες που παρέχονται. Ωστόσο η ανάλυση των βασικότερων από αυτά θα βοηθήσει σημαντικά στην κατανόηση της διαδικασίας ανάπτυξης της εφαρμογής Orbs.

3.7.1 Scenes

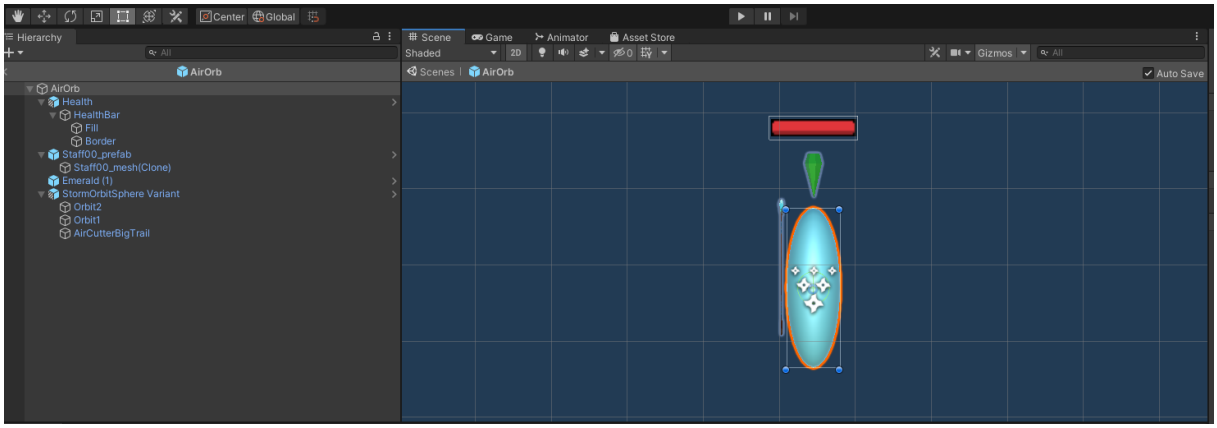
Ο χώρος εργασίας του Unity ονομάζεται Scene (σκηνή) Οι σκηνές είναι αρχεία τύπου Unity Scene File και λειτουργούν ως δοχεία εντός των οποίων τοποθετούνται τα αντικείμενα του παιχνιδιού. Κατά την δημιουργία ενός νέου κενού Unity Project δημιουργείται το αρχείο SampleScene ως η πρώτη σκηνή του παιχνιδιού που πρόκειται να αναπτυχθεί. Ο αριθμός των σκηνών ενός παιχνιδιού καθορίζεται από την κρίση του δημιουργού του . Σε ορισμένες περιπτώσεις ένα παιχνίδι μπορεί να αποτελείται από μία μόνο σκηνή ενώ σε άλλες να χωρίζεται σε μία ή παραπάνω σκηνές ανά επίπεδο (level) .

Πέρα από την δυνατότητα ύπαρξης πολλαπλών σκηνών σε ένα Unity Project είναι εφικτή και η μεταφορά αντικειμένων μεταξύ των σκηνών. Για την μεταφορά ενός αντικειμένου από μία σκηνή σε μία άλλη αρκεί ο χρήστης να πατήσει δεξί κλικ → Copy στο αντικείμενο που επιθυμεί να αντιγράψει από την μία σκηνή και δεξί κλικ → Paste στην σκηνή την οποία θέλει να το μεταφέρει.

3.7.2 Game Objects

Κάθε αντικείμενο στο Unity είναι ένα GameObject.Τα πιόνια, το ταμπλό η κάμερα και το φως είναι μερικά από τα αντικείμενα στο παιχνίδι Orbs .Στην παρούσα διπλωματική εργασία κάθε αναφορά σε «αντικείμενο» υπονοεί ένα GameObject. Κατά την δημιουργία τους τα αντικείμενα δεν έχουν κάποια λειτουργικότητα. Η λειτουργικότητα του κάθε αντικειμένου παρέχεται από τα components (συστατικά)

τα οποία θα προστεθούν σε αυτό μεταγενέστερα. Τα αντικείμενα μέσα σε ένα παιχνίδι μπορεί να βρίσκονται είτε αυτόνομα είτε να είναι μέλη μίας πιο σύνθετης δομής αντικειμένων, δηλαδή να περιέχουν άλλα αντικείμενα ή να βρίσκονται μέσα σε άλλα αντικείμενα. Χαρακτηριστικό παράδειγμα σύνθετης δομής αντικειμένων αποτελούν τα πιόνια του παιχνιδιού Orbs όπως φαίνεται στο σχήμα 3.9 .



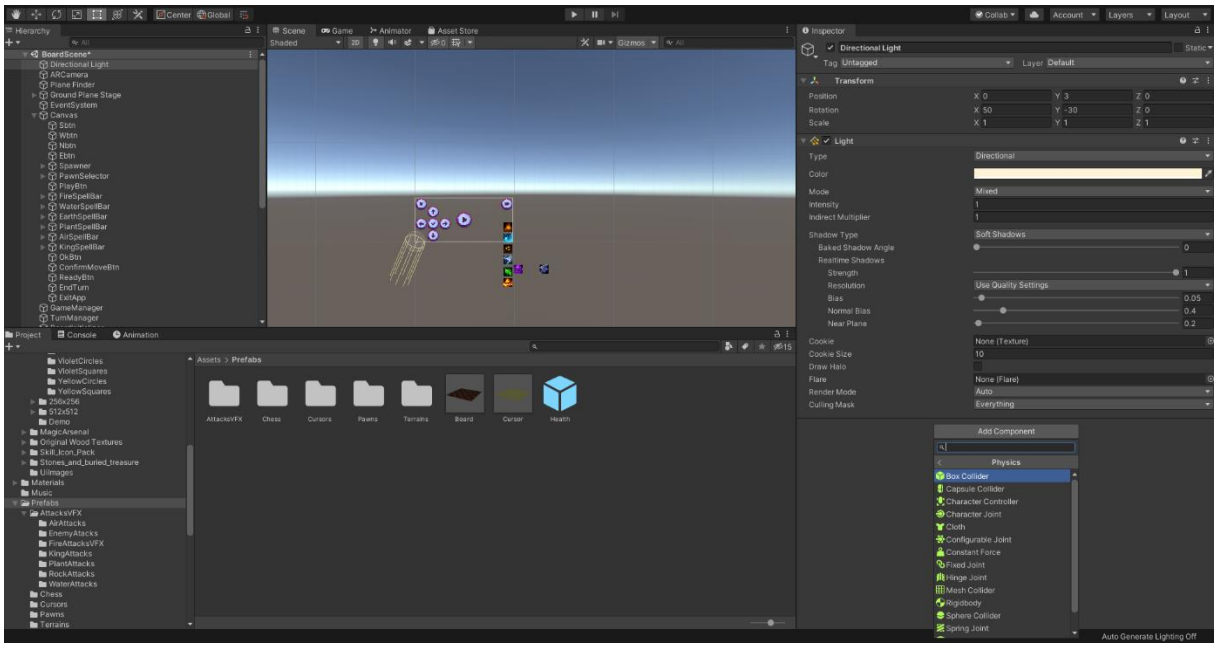
Σχήμα 3.9 : Παράδειγμα σύνθετης δομής GameObject

3.7.3 Components

Τα components ή συστατικά είναι τα χαρακτηριστικά τα οποία ορίζουν την λειτουργικότητα των αντικειμένων. Εξ ‘ορισμού κάθε νέο αντικείμενο περιέχει ένα transform component το οποίο ορίζει το μέγεθος, την τοποθεσία και την περιστροφή του. Για να γίνει πιο κατανοητός ο ρόλος των συστατικών παρακάτω αναφέρονται ορισμένα από τα βασικότερα με τις λειτουργίες τους.

1. Camera – Μετατρέπει ένα αντικείμενο σε κάμερα
2. Light – Μετατρέπει ένα αντικείμενο σε φωτεινή πηγή
3. Rigid Body – Το αντικείμενο λειτουργεί ως στερεό σώμα και υπακούει σε νόμους της φυσικής όπως η βαρύτητα.
4. Colliders – Για την διαχείριση των συγκρούσεων του αντικειμένου με το περιβάλλον του.
5. Mesh Filter/Mesh Renderer – Για την εμφάνιση του σχήματος των αντικειμένων.
6. Audio Source – Για να την αναπαραγωγή ήχου.
7. Button – Για να λειτουργεί το αντικείμενο ως κουμπί.

Προφανώς όταν δημιουργείται μέσω του Unity ένα αντικείμενο με προκαθορισμένο ρόλο όπως μία κάμερα, ένα κουμπί ή μία φωτεινή πηγή τα ανάλογα συστατικά προστίθενται αυτόματα χωρίς όμως να περιορίζεται η περεταίρω παραμετροποίηση της συμπεριφοράς του μέσω επιπρόσθετων συστατικών ή κώδικα.

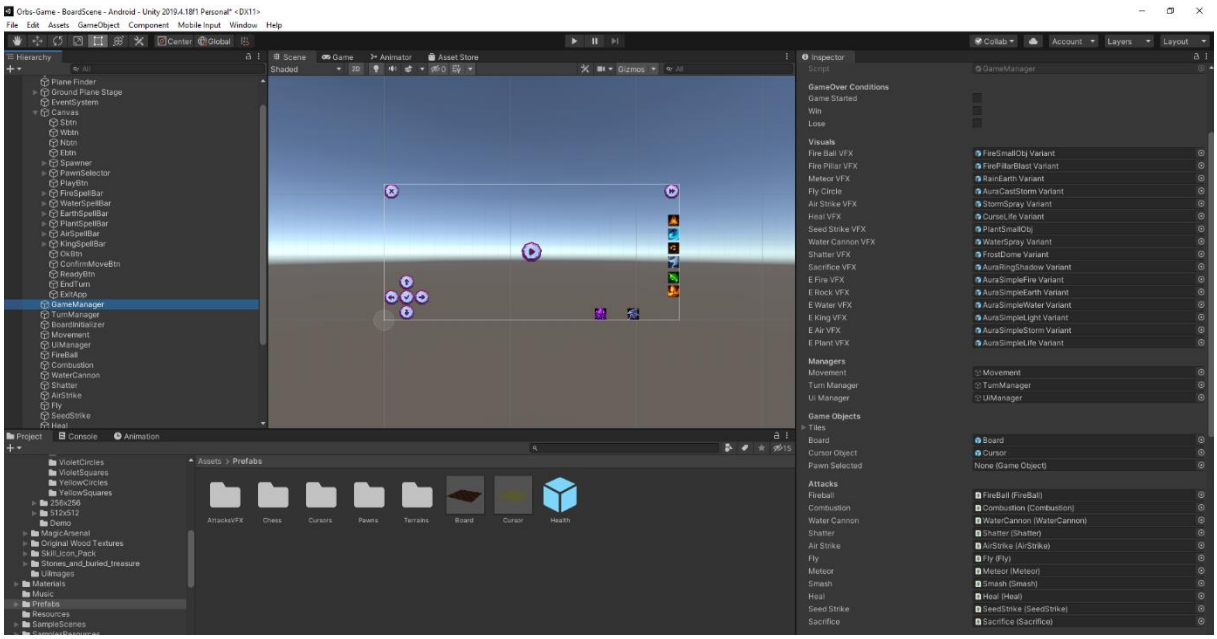


Σχήμα 3.10 : Add Component

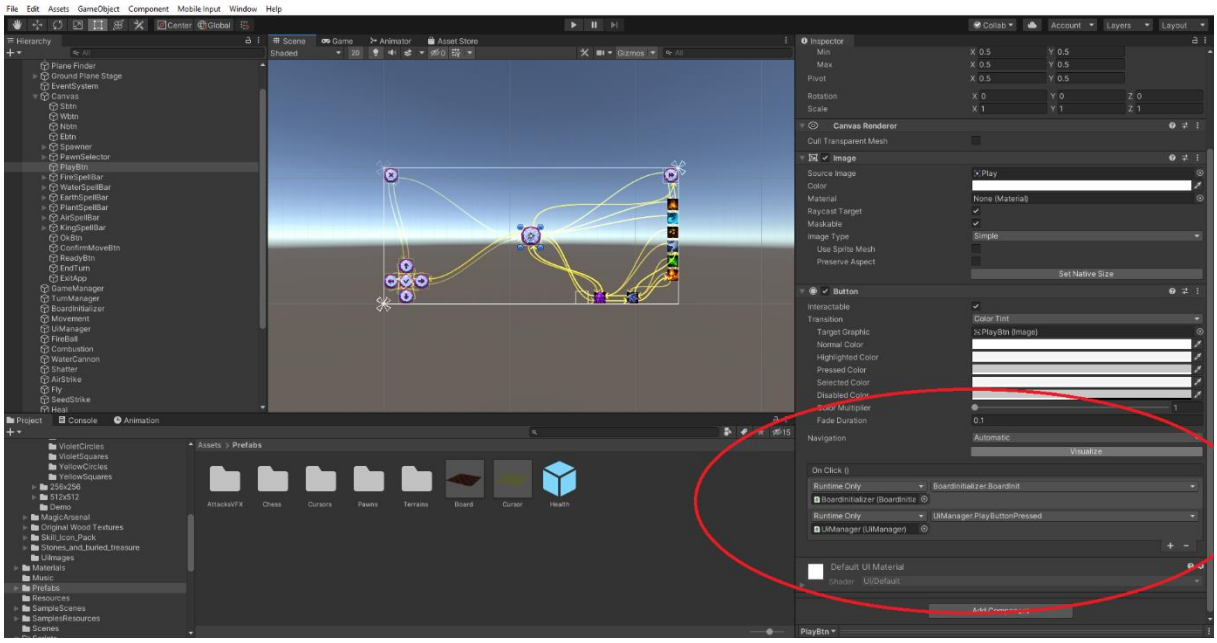
3.7.4 Scripts

Scripts είναι τα αρχεία κώδικα τα οποία ορίζουν τις λειτουργίες του παιχνιδιού ή της εφαρμογής. Όπως αναφέρθηκε στην ενότητα 3.7.3 η συμπεριφορά των αντικειμένων ορίζεται μέσω των συστατικών που προστίθενται σε αυτά. Ουσιαστικά τα συστατικά τα οποία παρέχει η μηχανή Unity αποτελούν προκαθορισμένα scripts για την διαχείριση προκαθορισμένων λειτουργιών. Οι επιπλέον λειτουργίες ενός παιχνιδιού ή μίας εφαρμογής τις οποίες επιθυμούν να προσθέσουν οι δημιουργοί εισάγονται μέσω νέων script τα οποία προστίθενται ως συστατικά στα αντίστοιχα αντικείμενα προκειμένου εκείνα να αποκτήσουν την επιθυμητή λειτουργικότητα.

Όπως αναφέρθηκε σε προηγούμενη ενότητα τα scripts της εφαρμογής Orbs γράφτηκαν σε γλώσσα C# και ως εκ τούτου ακολουθούν τις αρχές του αντικειμενοστραφούς προγραμματισμού. Το Unity παρέχει πρόσβαση στις μεταβλητές και τις μεθόδους των script μέσω του παραθύρου inspector. Για να γίνει μία μέθοδος ή μία μεταβλητή ορατή στο παράθυρο inspector πρέπει να δηλωθεί ως public. Μέσω του παραθύρου inspector δίνεται η δυνατότητα αλλαγής των τιμών των μεταβλητών εκτός και εντός της ροής του προγράμματος. Επίσης μέσω του παραθύρου inspector μπορεί να γίνει ανάθεση άλλων αντικειμένων ή script ως περιεχόμενα μεταβλητών με drag and drop. Αντίστοιχα τα κουμπιά της διεπαφής συνδέονται με την μέθοδο ή τις μεθόδους τις οποίες καλούν μέσω του On Click Event ορίζοντας το script το οποίο περιέχει την μέθοδο και εν συνεχεία την μέθοδο μέσω του παραθύρου Inspector.



Σχήμα 3.11 : Ανάθεση script και αντικειμένων σε μεταβλητές



Σχήμα 3.12 : On Click Event

3.7.5 Prefabs

Στα επόμενα κεφάλαια θα γίνουν πολλαπλές αναφορές σε prefab αντικείμενα. Τα prefab αντικείμενα είναι αντίγραφα αντικειμένων τα οποία αποθηκεύονται εντός του Unity Project σύροντας τα (drag and drop) μέσα σε έναν φάκελο. Μετατρέποντας ένα αντικείμενο σε prefab δίνεται η δυνατότητα δημιουργίας πολλαπλών αντιγράφων και παραλλαγών του. Ένα άλλο πλεονέκτημα των prefab αντικειμένων είναι η δυναμική προσθαφαίρεση τους εντός των σκηνών κατά την ροή του προγράμματος. Η δυνατότητα αυτή είναι εξαιρετικά σημαντική ιδίως σε περιπτώσεις σκηνών με υψηλό αριθμό αντικειμένων καθώς η ύπαρξη ενός αντικειμένου σε μία σκηνή καταναλώνει πόρους

συστήματος. Επομένως από πλευράς οικονομίας πόρων συμφέρει περισσότερο να αφαιρεθεί εντελώς ένα αντικείμενο από το να σταματήσει απλά να είναι ορατό αλλά να εξακολουθεί να βρίσκεται εντός μίας σκηνής.

3.8 Asset Store

Το Unity Asset Store είναι ένας δικτυακός χώρος διαμοιρασμού υλικού για το Unity. Το υλικό του Asset Store περιλαμβάνει πακέτα τα οποία περιέχουν αντικείμενα, scripts, εικόνες, στοιχεία διεπαφής (UI elements), animations, materials και άλλα. Τα πακέτα του Unity Asset Store διατίθενται είτε δωρεάν είτε επί πληρωμή κατά την κρίση του δημιουργού τους.

Για την εφαρμογή Orbs χρησιμοποιήθηκαν τα παρακάτω πακέτα του Unity Asset Store :

- Minimalistic Icons Pack
- AurynSky - Low Poly Toon Battle Arena / Tower Defense Pack
- Free Stylized Fantasy Weapons
- Game GUI Vol1
- Jelly Icons
- Magic Arsenal
- Stones and Buried Treasure
- Thaleah Free Pixel Font

3.9 Deployment

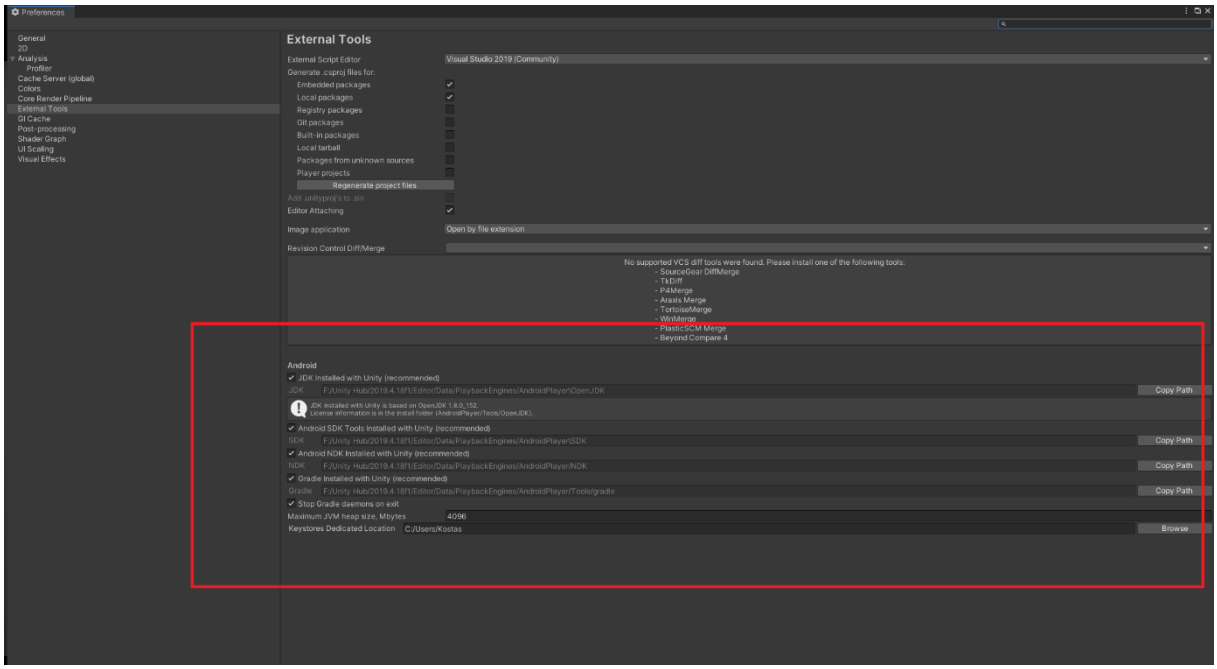
Σε αυτή την ενότητα περιγράφεται η διαδικασία μετατροπής του Unity Project σε εφαρμογή για την Android πλατφόρμα. Σε αυτό το σημείο θα πρέπει να αναφερθεί ότι η μηχανή Unity μπορεί να μεταφέρει ένα Unity Project σε μία πληθώρα από πλατφόρμες (Windows, IOS, Android, PS4, Xbox, Linux κ.α.) με ελάχιστες έως καθόλου αλλαγές.

3.9.1 Προετοιμασία συσκευής

Για να μπορέσει μια android συσκευή να δεχθεί και να τρέξει μία εφαρμογή η οποία δεν προέρχεται από το Google Play Store πρέπει να έχει ενεργοποιημένες τις ρυθμίσεις για προγραμματιστές ή αλλιώς Developer Mode και την ρύθμιση USB Debugging επίσης ενεργοποιημένη. Καθώς ο τρόπος με τον οποίο ενεργοποιούνται οι παραπάνω ρυθμίσεις ενδέχεται να διαφέρει ανάμεσα στις διάφορες συσκευές android δεν θα γίνει αναφορά στα βήματα της διαδικασίας.

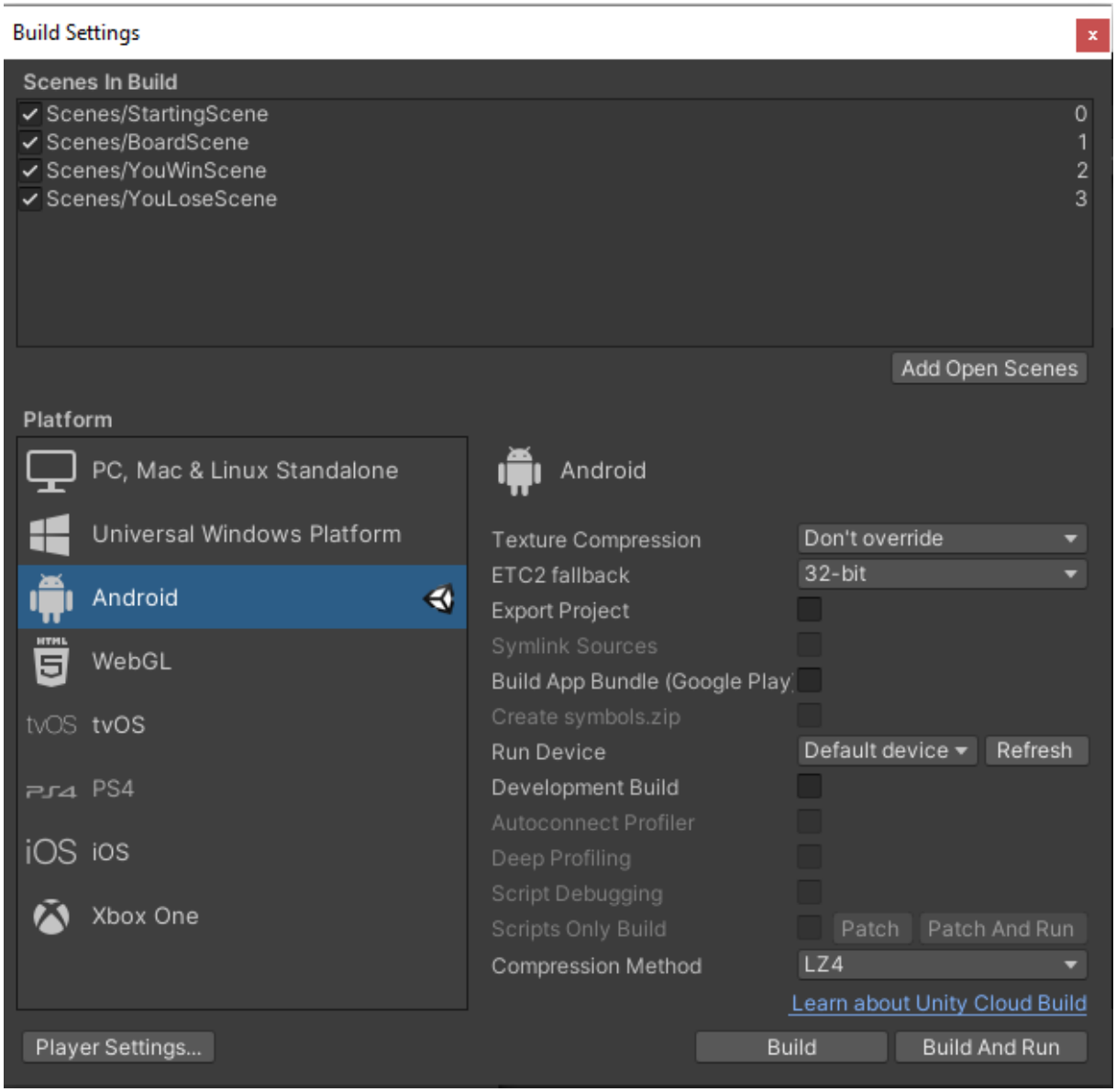
3.9.2 Προετοιμασία Unity

Για την μεταφορά της εφαρμογής στην android πλατφόρμα απαιτείται η εγκατάσταση του Android Module (Android Build Support) το οποίο περιλαμβάνει Android SDK & NDK Tools και OpenJDK. Η εγκατάσταση του Android Module μπορεί να γίνει μέσω του Unity Hub όπως αναφέρθηκε στην ενότητα 3.6.1. Η εγκατάσταση των εν λόγω εργαλείων φαίνεται στο παράθυρο Edit → Preferences → External Tools στην καρτέλα Android (Σχήμα 3.13).



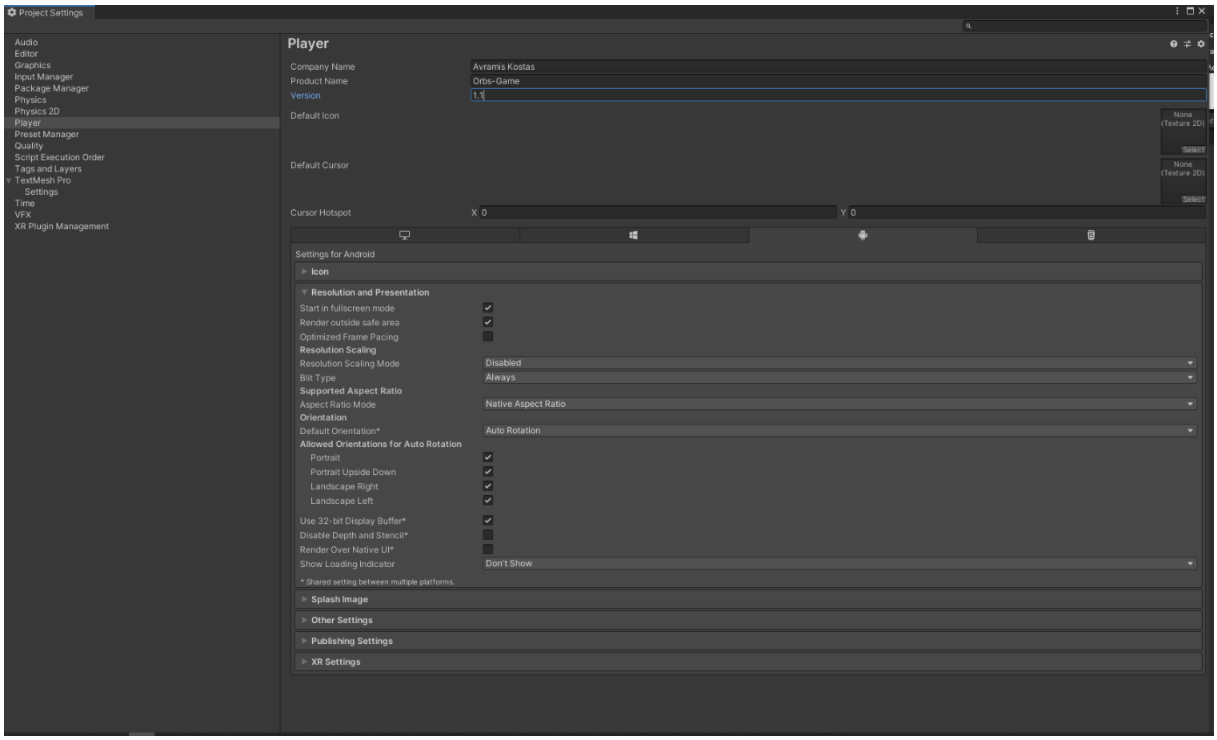
Σχήμα 3.13 : Τοποθεσία εγκατάστασης Android Module

Οι υπόλοιπες ρυθμίσεις που αφορούν την έκδοση της εφαρμογής στην Android πλατφόρμα βρίσκονται στο παράθυρο File → Build Settings. Μέσω του παραθύρου Build Settings επιλέγεται η πλατφόρμα στην οποία θα εκδοθεί το Unity Project καθώς και οι σκηνές οι οποίες θα περιλαμβάνονται στην τελική εφαρμογή (Σχήμα 3.14).



Σχήμα 3.14 : Build Settings

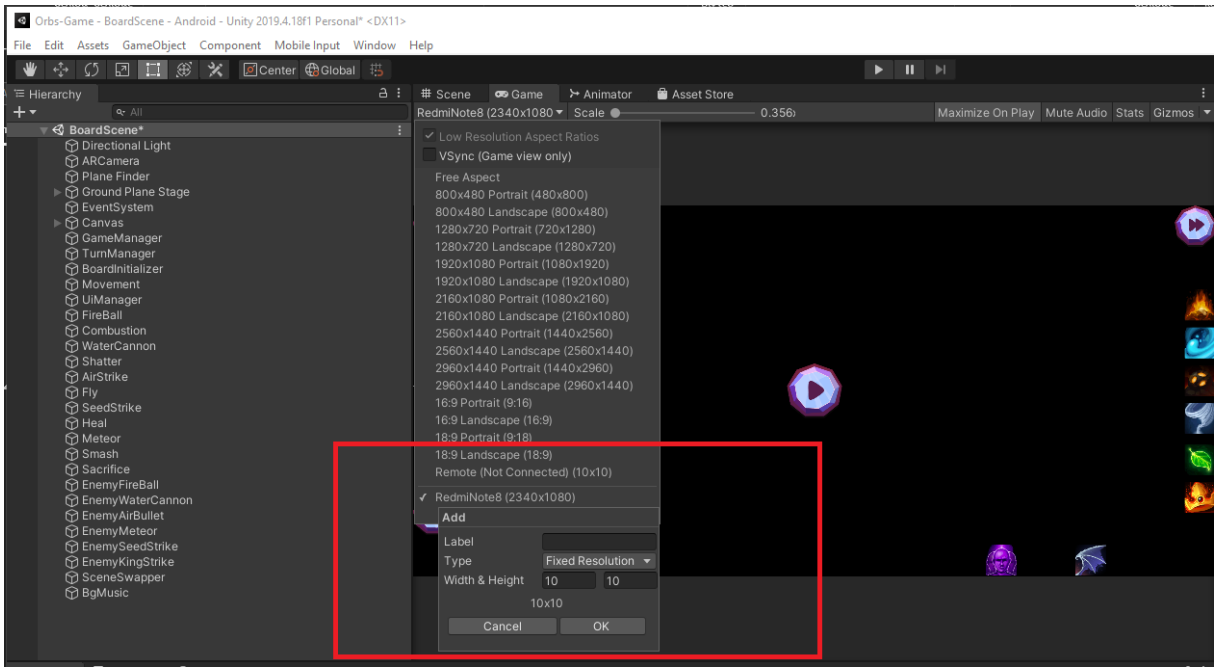
Το κουμπί Player Settings του σχήματος 3.14 οδηγεί στην καρτέλα Player του παραθύρου Project Settings η οποία παρέχει περαιτέρω ρυθμίσεις για την έκδοση της android εφαρμογής. Σημαντικές ρυθμίσεις της καρτέλας Player είναι εκείνες της κατηγορίας Resolution and Presentation οι οποίες εξασφαλίζουν την ομαλή λειτουργία της εφαρμογής ανεξάρτητα από το μέγεθος και την ανάλυση της οθόνης κάθε συσκευής .



Σχήμα 3.15 : Player Settings

3.10 Συσκευή έκδοσης

Η διαθέσιμη android συσκευή στην οποία εκδόθηκε και έτρεξε η εφαρμογή Orbs είναι το Xiaomi Redmi Note 8T με την έκδοση λειτουργικού συστήματος Android 10 . Η ανάλυση του του Xiaomi Redmi Note 8T είναι 1080x2340 Pixels . Για να επιτευχθεί η πλήρης αξιοποίηση του χώρου της οθόνης από την εφαρμογή Orbs η διεπαφή της σχεδιάστηκε αυστηρά με γνώμονα την ανάλυσης της συσκευής. Θεωρώντας ότι η εφαρμογή Orbs θα τρέχει σε Landscape Mode (οριζόντια οθόνη) δημιουργήθηκε μια νέα ανάλυση στο Unity μέσω της καρτέλας Game 2340x1080 pixels για να ταιριάξει η διεπαφή ακριβώς στα όρια της οθόνης της συσκευής. Προφανώς αυτή η τακτική δεν μπορεί να εφαρμοστεί για εφαρμογές οι οποίες προορίζονται για πληθώρα συσκευών με διαφορετικές αναλύσεις και μεγέθη οθονών αλλά στα πλαίσια της παρούσας πτυχιακής εργασία ήταν θεμιτή λόγω της απλότητας που προσέφερε στην διαδικασία έκδοσης της εφαρμογής Orbs.



Σχήμα 3.16 : Επιλογές ανάλυσης εφαρμογής

3.11 Έκδοση εφαρμογής

Η καρτέλα Build Settings περιέχει τα κουμπιά Build και Build and Run. Η επιλογή Build δημιουργεί ένα αρχείο τύπου APK το οποίο εν συνεχεία μπορεί να μεταφερθεί στην android συσκευή και μέσω εκείνης να γίνει η εγκατάσταση της εφαρμογής ενώ η επιλογή Build and Run δημιουργεί το αρχείο APK και τρέχει την εφαρμογή κατευθείαν στην συσκευή android η οποία είναι συνδεμένη στον υπολογιστή μέσω ενός USB καλωδίου.

3.12 Επίλογος

Η μηχανή Unity είναι ένα εξαιρετικά ισχυρό εργαλείο για την ανάπτυξη βιντεοπαιχνιδιών και παρουσιάζει ιδιαίτερη ευκολία στην εκμάθηση της χωρίς όμως να υστερεί σε βάθος πολυπλοκότητας ή δυνατοτήτων συγκριτικά με αντίστοιχα εργαλεία. Η δυνατότητα εξαγωγής παιχνιδιών σε διάφορες πλατφόρμες, σε συνδυασμό με την υποστήριξη τεχνολογιών εκτεταμένης πραγματικότητας καθιστούν την μηχανή Unity ως μια ιδανική επιλογή τόσο για την εκπόνηση της παρούσας διπλωματικής εργασίας, όσο και για την ανάπτυξη βιντεοπαιχνιδιών γενικά.

Κεφάλαιο 4ο: Το παιχνίδι Orbs

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει ανάλυση της εφαρμογής Orbs από την σκοπιά του παιχνιδιού, δηλαδή θα δοθεί έμφαση στους κανόνες, τα κομμάτια και τον τρόπο παιξίματος του παιχνιδιού, όχι στα τεχνικά του στοιχεία. Η κύρια έμπνευση για τον σχεδιασμό του παιχνιδιού είναι τα βιντεοπαιχνίδια στρατηγικής[16] τύπου Auto Chess και κατ' επέκταση το ίδιο το σκάκι από τα οποία η εφαρμογή Orbs δανείστηκε αρκετά στοιχεία. Σε αυτό το σημείο θα πρέπει να σημειώσουμε τις δύο παραδοχές οι οποίες έγιναν κατά τον σχεδιασμό του παιχνιδιού .

Παραδοχή 1:

Το παιχνίδι ως παραλλαγή του σκακιού σχεδιάστηκε για δύο παίχτες. Ωστόσο η υλοποίηση λειτουργικότητας δύο παικτών θα έβγαζε την ανάπτυξη της εφαρμογής από τα χρονικά πλαίσια μίας πτυχιακής εργασίας και επομένως ο δεύτερος παίκτης αντικαταστάθηκε από τον υπολογιστή.

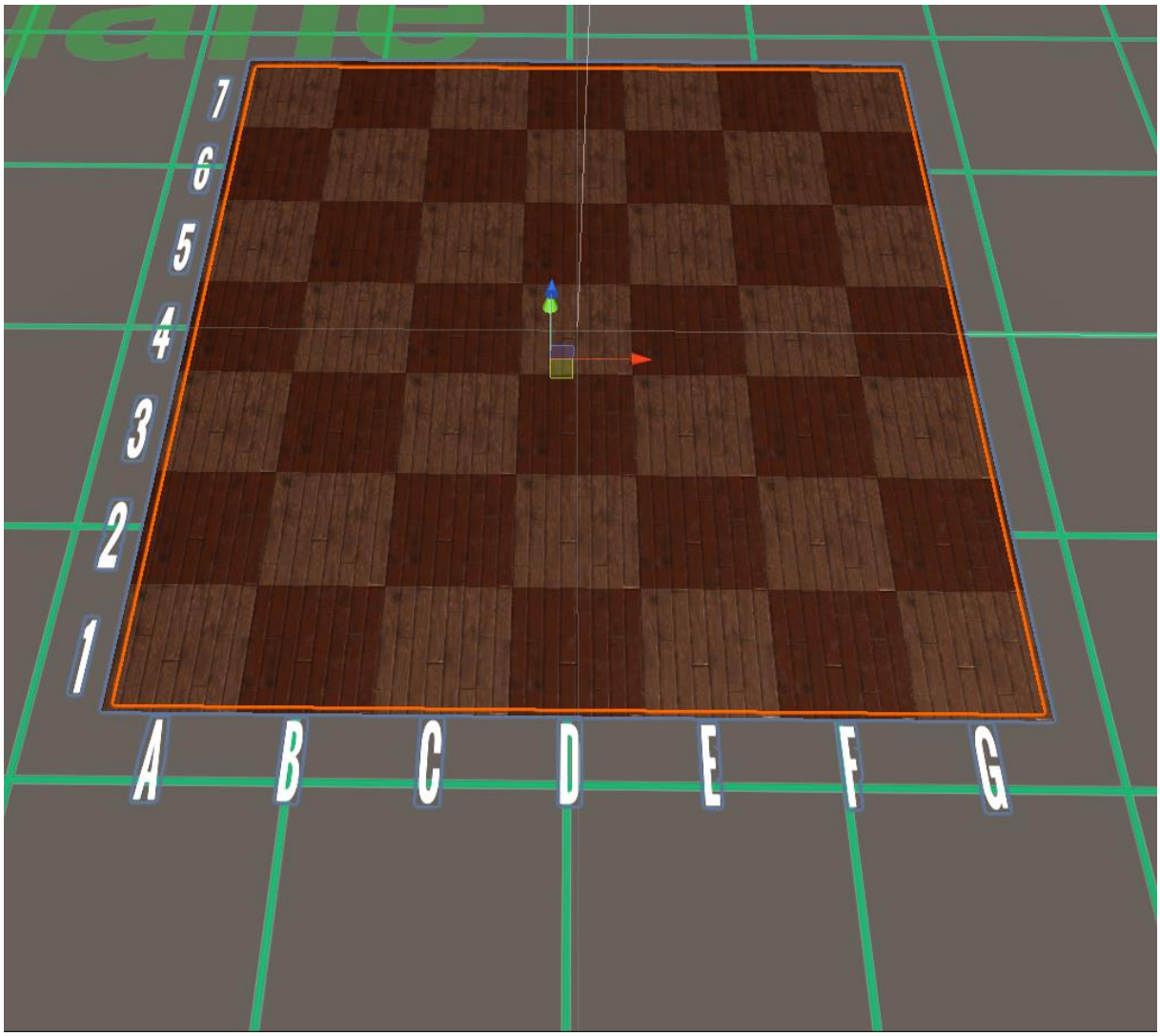
Παραδοχή 2:

Η ανάπτυξη προγράμματος τεχνητής νοημοσύνης για τον ρόλο του αντιπάλου είναι ένα εξαιρετικά περίπλοκο εγχείρημα και ξεφεύγει από το θέμα της παρούσας πτυχιακής εργασίας. Λόγω αυτού οι κινήσεις του αντιπάλου (υπολογιστή) καθορίζονται τυχαία από το σύστημα αλλά με αριθμητική υπεροχή στους πόντους ζημίας των επιθέσεων των πιονιών του προκειμένου να αντισταθμιστεί αυτή η τυχαιότητα.

4.2 Βασικά στοιχεία παιχνιδιού

4.2.1 Ταμπλό

Το ταμπλό του παιχνιδιού είναι μία σκακιέρα 49 τετραγώνων (7 γραμμές και 7 στήλες) με λευκά και μαύρα τετράγωνα εναλλάξ. Η αρίθμηση των γραμμών ξεκινά από το 1 μέχρι το 7 (από κάτω προς τα επάνω) και η αρίθμηση των στηλών από το A έως το G (από αριστερά προς τα δεξιά) .



Σχήμα 4.1 : Ταμπλό

4.2.2 Πιόνια

Ο κάθε παίκτης έχει στην διάθεση του έξι πιόνια. Από αυτά τα πέντε είναι πιόνια στρατιώτες και το ένα ο βασιλιάς. Τα πιόνια του παίκτη στο επάνω μέρος τους έχουν ένα πράσινο πετράδι ενώ αντίθετα τα πιόνια του υπολογιστή έχουν ένα κρανίο. Τα πιόνια βασιλιάδες ξεχωρίζουν από τους στρατιώτες έχοντας μεγαλύτερο μέγεθος και μία κόρωνα στο επάνω μέρος τους. Όλα τα πιόνια ξεκινούν με 10 πόντους ζωής (LP=Life Points). Όταν οι πόντοι ζωής ενός πιονιού φτάσουν στο 0 το πιόνι καταστρέφεται και αφαιρείται από το ταμπλό.

Κάθε πιόνι στρατιώτης είναι ξεχωριστό και σχετίζεται με ένα στοιχείο της φύσης καθορίζοντας τις ικανότητες του αλλά και τις αλληλεπιδράσεις του με τα αντίπαλα πιόνια. Ο παρακάτω πίνακας παρουσιάζει την πλήρη συλλογή πιονιών του κάθε παίκτη.

ΟΝΟΜΑ	ΣΤΟΙΧΕΙΟ-ΙΔΙΟΤΗΤΑ	ΧΡΩΜΑ
Fire Orb	Φωτιά	Πορτοκαλί

Water Orb	Νερό	Μπλε
Rock Orb	Πέτρα	Σκούρο Πράσινο και Καφέ
Plant Orb	Φυτά	Ανοιχτό Πράσινο
Air Orb	Αέρας	Ανοιχτό μπλε
King Orb	Βασιλιάς	Μωβ / Κίτρινο

Πίνακας 4.1 : Τύποι πιονιών

4.2.3 Ειδικό έδαφος

Κατά την έναρξη μίας παρτίδας επιλέγονται τυχαία από το σύστημα 5 τετράγωνα του ταμπλό τα οποία θα αποτελούν ειδικό έδαφος κατά την διάρκεια της. Κάθε τετράγωνο ειδικού εδάφους αντιστοιχεί σε ένα διαφορετικό στοιχείο όπως και με τα πιόνια στρατιώτες (φωτιά, νερό, πέτρα, φυτά, αέρας). Η αντιστοίχιση τετράγωνου-στοιχείου γίνεται τυχαία. Κάθε τετράγωνο ειδικού εδάφους έχει μια σφαίρα επιρροής η οποία αποτελείται από όλα τα άμεσα γειτονικά τετράγωνα γύρω από αυτό. Εάν ένα πιόνι στρατιώτης βρίσκεται σε τετράγωνο το οποίο ανήκει στην σφαίρα επιρροής του αντίστοιχου ειδικού εδάφους λαμβάνει 6 επιπλέον πόντους ζημιάς (Terrain Bonus) στις ζημιογόνες επιθέσεις του. Τα τετράγωνα ειδικού εδάφους διακρίνονται από τα αντικείμενα-σημαίες τα οποία τοποθετούνται επάνω σε αυτά. Στην συγκεκριμένη έκδοση του παιχνιδιού (παίκτης εναντίον υπολογιστή) οι επιπλέον πόντοι επίθεσης από τα τετράγωνα ειδικού εδάφους λαμβάνονται μόνο από τον παίκτη.

ΟΝΟΜΑ	**ΣΤΟΙΧΕΙΟ-ΙΔΙΟΤΗΤΑ**	**ΑΝΤΙΚΕΙΜΕΝΟ ΣΗΜΑΙΑ**
Fire Terrain	Φωτιά/Fire	Brazier/Μαγκάλι
Forest Terrain	Φυτά/Plant	Pinetree/Πεύκο
Water Terrain	Νερό/Water	Lake/Λίμνη
Rock Terrain	Πέτρα/Rock	Boulder/Βράχος
Tornado Terrain	Αέρας/Air	Tornado/Ανεμοστρόβιλος

Πίνακας 4.2 : Τύποι ειδικού εδάφους

4.2.4 Στήσιμο πιονιών

Σε αντίθεση με το σκάκι η εναρκτήρια θέση του κάθε πιονιού δεν είναι προκαθορισμένη. Πριν το ξεκίνημα της κάθε παρτίδας υπάρχει η φάση στησίματος των πιονιών κατά την οποία ο παίκτης επιλέγει τις αρχικές θέσεις των πιονιών του όπως επιθυμεί εκείνος. Τα πιόνια του αντιπάλου (υπολογιστή) στήνονται αυτόματα σε τυχαίες θέσεις κατά την έναρξη της φάσης στησίματος. Για το στήσιμο των πιονιών ισχύουν οι παρακάτω κανόνες :

- Κατά την φάση του στησίματος και μόνο κατά την διάρκεια της το ταμπλό χωρίζεται σε 2 στρατόπεδα. Το πρώτο στρατόπεδο αποτελείται από τις γραμμές 1 έως 3 και το δεύτερο από τις γραμμές 5 έως 7 .

- Σε κάθε παίκτη αποδίδεται ένα στρατόπεδο. Στο σενάριο παίκτης εναντίον υπολογιστή οι γραμμές 1-3 δίνονται για το στήσιμο των πιονιών του παίκτη και οι γραμμές 5-7 για τα πιόνια του υπολογιστή.
- Κανένα πιόνι δεν μπορεί να τοποθετηθεί σε τετράγωνο που καταλαμβάνεται από ειδικό έδαφος.
- Η γραμμή 4 του ταμπλό δεν μπορεί να χρησιμοποιηθεί στο στήσιμο.
- Ο βασιλιάς του κάθε παίκτη πρέπει να τοποθετηθεί απαραίτητα σε κάποιο τετράγωνο της πρώτης γραμμής του ανάλογου στρατοπέδου (γραμμές 1 και 7).

4.2.5 Κίνηση πιονιών

Κάθε πιόνι μπορεί να κινηθεί κατά 1 τετράγωνο προς οποιαδήποτε κατεύθυνση εάν ισχύουν όλα τα παρακάτω:

- Το τετράγωνο προορισμός δεν είναι τετράγωνο ειδικού εδάφους
- Το τετράγωνο προορισμός δεν καταλαμβάνεται από άλλο πιόνι.
- Το πιόνι δεν προσπαθεί να κινηθεί εκτός του ταμπλό.

4.2.6 Ικανότητες πιονιών

Κάθε πιόνι κατέχει μία ή παραπάνω ξεχωριστές ικανότητες ανάλογα με το στοιχείο του. Λόγω των παραδοχών που αναφέρθηκαν στην εισαγωγή αυτού του κεφαλαίου στα πιόνια του αντιπάλου-υπολογιστή έχουν δοθεί διαφορετικές ικανότητες πιονιών ή παραλλαγές τους από ότι στα πιόνια του παίκτη.

4.2.7 Χαρακτηριστικά ικανοτήτων

Τα χαρακτηριστικά ικανοτήτων των πιονιών είναι :

- Εμβέλεια/Range : Η εμβέλεια μίας ικανότητας μπορεί να είναι απεριόριστη (uncapped) ή γειτονική (close). Οι ικανότητες απεριόριστης εμβέλειας μπορούν να στοχεύσουν οποιοδήποτε τετράγωνο ή πιόνι εντός του ταμπλό. Αντίθετα οι ικανότητες γειτονικής εμβέλειας μπορούν να στοχεύσουν μόνο τετράγωνα ή πιόνια τα οποία είναι άμεσα γειτονικά με το πιόνι το οποίο εκτελεί την ικανότητα.
- Στόχος/Target : Ο στόχος μιας ικανότητας μπορεί να είναι είτε αντίπαλο (Enemy) πιόνι, είτε τετράγωνο (Square), είτε συμμαχικό πιόνι (Ally). Όλες οι ικανότητες έχουν μόνο έναν στόχο.
- Ζημιά/Damage : Ορισμένες από τις ικανότητες των πιονιών προκαλούν ζημία στα αντίπαλα πιόνια. Σε αυτό το μηχανισμό θα γίνει αναλυτικότερη αναφορά σε επόμενη υποενότητα.
- Επίδραση/Effect : Ορισμένες από τις ικανότητες έχουν επιπλέον επίδραση στο παιχνίδι πέρα από την ζημία στα πιόνια του αντιπάλου.
- Εκτοπισμός/Displacement : Μετακίνηση του πιονιού που δέχεται επίθεση κατά ένα τετράγωνο προς την αντίθετη κατεύθυνση από την οποία προέρχεται η επίθεση. Η μετατόπιση συμβαίνει μόνο εάν το τετράγωνο προορισμός δεν καταλαμβάνεται από άλλο πιόνι ή ειδικό έδαφος και βρίσκεται εντός ταμπλό.
- Παγωμένη κατάσταση/Frozen Status : Όταν ένα πιόνι βρίσκεται σε παγωμένη κατάσταση δέχεται επιπλέον 5 πόντους ζημίας από τις ικανότητες meteor, seed strike και water cannon. Η παγωμένη κατάσταση επιδρά σε ένα πιόνι μέχρι αυτό να καταστραφεί ή μέχρι κάποιο άλλο πιόνι να εισέλθει σε παγωμένη κατάσταση.
- Επιστροφή ζημιάς/Damage return : Ορισμένες ικανότητες προκαλούν ζημιά και στα πιόνια τα οποία τις εκτελούν πέρα από τον στόχο τους.
- Θεραπεία/Heal : Αναπλήρωση μερικών πόντων ζωής ενός πιονιού χωρίς όμως την δυνατότητα οι πόντοι ζωής του στόχου να ξεπεράσουν το μέγιστο (10 πόντοι ζωής).

- Πέταγμα/Fly : Μετακίνηση του πιονιού Air orb σε ένα τετράγωνο το οποίο δεν καταλαμβάνεται από άλλο πιόνι ή ειδικό έδαφος οπουδήποτε στο ταμπλό. Η ικανότητα Fly θεωρείται κίνηση πιονιού και όχι εκτέλεση ικανότητας (βλ. 4.2.10)

4.2.8 Πίνακες ικανοτήτων

Ο πίνακας 4.3 αναγράφει τις λεπτομέρειες των ικανοτήτων των πιονιών του παίκτη. Αντίστοιχα ο πίνακας 4.4 τις λεπτομέρειες των ικανοτήτων των πιονιών του υπολογιστή.

NAME	ATTRIBUTE	DAMAGE	RANGE	TARGET	EFFECT
Fireball	Fire	2	Close	Enemy	-
Combustion	Fire	1	Uncapped	Enemy	Damage return: 2p
Water Cannon	Water	2	Close	Enemy	-
Shatter	Water	-	Close	Enemy	Frozen Status
Air Strike	Air	1	Close	Enemy	Displacement
Fly	Air	-	Uncapped	Square	Fly
Seed Strike	Plant	2	Close	Enemy	-
Heal	Plant	-	Uncapped	Ally	Heal: 3p
Meteor	Rock	2	Close	Enemy	-
Sacrifice	King	5	Uncapped	Enemy	Damage return:5p

Πίνακας 4.3 : Ικανότητες πιονιών παίκτη

NAME	**ATTRIBUTE**	**DAMAGE**	**RANGE**	**TARGET**
Fireball	Fire	4	Close	Enemy
Air Bullet	Air	4	Close	Enemy
Water Cannon	Water	5	Close	Enemy
Meteor	Rock	5	Close	Enemy
Seed Strike	Plant	4	Close	Enemy
King Attack	King	9	Close	Enemy

Πίνακας 4.4 : Ικανότητες πιονιών υπολογιστή

4.2.9 Αλληλεπιδράσεις

Ως αλληλεπίδραση μεταξύ των πιονιών ορίζεται η ενδυνάμωση ή αποδυνάμωση (damage modifier) ορισμένων ικανοτήτων όταν το πιόνι-στόχος τους ανήκει σε ένα συγκεκριμένο στοιχείο. Οι αλληλεπιδράσεις ανάμεσα στα πιόνια είναι ένας από τους βασικότερους μηχανισμούς του παιχνιδιού επηρεάζοντας σε μεγάλο βαθμό την στρατηγική των παικτών τόσο στο στήσιμο των πιονιών όσο και στις μετέπειτα κινήσεις . Στο σενάριο παίκτης εναντίον υπολογιστή η τροποποίηση ζημιάς των ικανοτήτων λόγω αλληλεπίδρασης προσμετράται μόνο για τις ενέργειες του παίκτη.

NAMES	TARGET	DAMAGE MODIFIER
Fireball, Combustion	Plant	+2 P
Fireball, Combustion	Rock	-2 P
Water Cannon	Plant	-2 P
Water Cannon	Fire	+2 P
Seed Strike	Fire	-2 P
Seed Strike	Water	+2 P

Πίνακας 4.5 : Πίνακας αλληλεπιδράσεων

4.2.10 Γύροι

Μετά την φάση του στησίματος το παιχνίδι Orbs εξελίσσεται σε διαδοχικούς γύρους με τον παίκτη να ξεκινάει πρώτος στο σενάριο παίκτης εναντίον υπολογιστή. Ο κάθε γύρος αποτελείται από την φάση μετατόπισης πιονιού και την φάση εκτέλεσης ικανότητας. Κατά την φάση μετατόπισης ο παίκτης έχει το δικαίωμα να επιλέξει ένα οποιοδήποτε πιόνι του και να το μετακινήσει μία μόνο φορά με βάση τους κανόνες κίνησης της ενότητας 4.2.5. Αντίστοιχα κατά την φάση εκτέλεσης ικανότητας ο παίκτης έχει το δικαίωμα να επιλέξει ένα οποιοδήποτε πιόνι του και να εκτελέσει μία από τις ικανότητες του εάν αυτό είναι εφικτό. Ο παίκτης έχει το δικαίωμα να παραλείψει οποιαδήποτε από τις 2 φάσεις ή και τις 2 εάν επιθυμεί. Η σειρά των φάσεων για κάθε γύρο είναι ελεύθερη και στην κρίση του παίκτη. Η ικανότητα Fly του πιονιού Air Orb αποτελεί μοναδική εξαίρεση στον κανόνα των γύρων καθώς η χρήση της θεωρείται ως μετατόπιση επιτρέποντας την χρήση μίας ακόμα ικανότητας εντός του ίδιου γύρου.

4.2.11 Σύστημα μάχης και σκοπός του παιχνιδιού

Στο παιχνίδι Orbs ως μάχη ορίζεται η στόχευση αντίπαλου πιονιού με ζημιογόνα ικανότητα. Κατά την εκτέλεση της μάχης το επιτιθέμενο πιόνι προκαλεί ζημία στον πιόνι-στόχο της ικανότητας ανάλογα με την ζημία της ικανότητας, τα μπόνους ειδικού εδάφους (εάν υπάρχουν) και την τροποποίηση ζημιάς λόγω αλληλεπίδρασης. Ο τύπος υπολογισμού της ζημιάς είναι ο παρακάτω :

Final damage=Ability Damage + Terrain Bonus + Damage Modifier.

Η τελική ζημιά (Final damage) αφαιρείται από τους πόντους ζωής που έχει εκείνη την στιγμή το πιόνι στόχος. Στην περίπτωση που η τελική ζημιά είναι μεγαλύτερη από τους πόντους ζωής του στόχου την τρέχουσα χρονική στιγμή το πιόνι-στόχος αφαιρείται από το παιχνίδι.

Στο σενάριο παίκτης εναντίον υπολογιστή ο παίκτης κερδίζει όταν αφαιρεθούν όλα τα αντίπαλα πιόνια από το παιχνίδι. Αντίθετα ο παίκτης χάνει το παιχνίδι όταν αφαιρεθούν 3 πιόνια του ή ο βασιλιάς του. Στην περίπτωση κατά την οποία ο βασιλιάς του παίκτη και το τελευταίο πιόνι του υπολογιστή καταστραφούν ταυτόχρονα με την χρήση της ικανότητας sacrifice η νίκη δίνεται στον παίκτη.

4.3 Επίλογος

Το παιχνίδι Orbs επιδιώκει να εμπλουτίσει την παραδοσιακή εμπειρία των επιτραπέζιων παιχνιδιών μέσω των στοιχείων που προσφέρει ένα βιντεοπαιχνίδι .Ένα επίσης μεγάλο πλεονέκτημα αυτού του υβριδικού είδους παιχνιδιών είναι η δυνατότητα των παικτών να βρίσκονται είτε στον ίδιο χώρο είτε σε διαφορετικά μέρη σε αντίθεση με τα παραδοσιακά επιτραπέζια τα οποία θέτουν αυτόν τον περιορισμό. Τέλος είναι ιδιαίτερα σημαντική η δυνατότητα αλλαγών, προσθήκης στοιχείων και αναβαθμίσεων του παιχνιδιού μετά την έκδοση του χωρίς να χρειάζεται εκ νέου αγορά όπως θα γινόταν με ένα επιτραπέζιο παιχνίδι.

Κεφάλαιο 5ο: Δομή εφαρμογής

5.1 Εισαγωγή

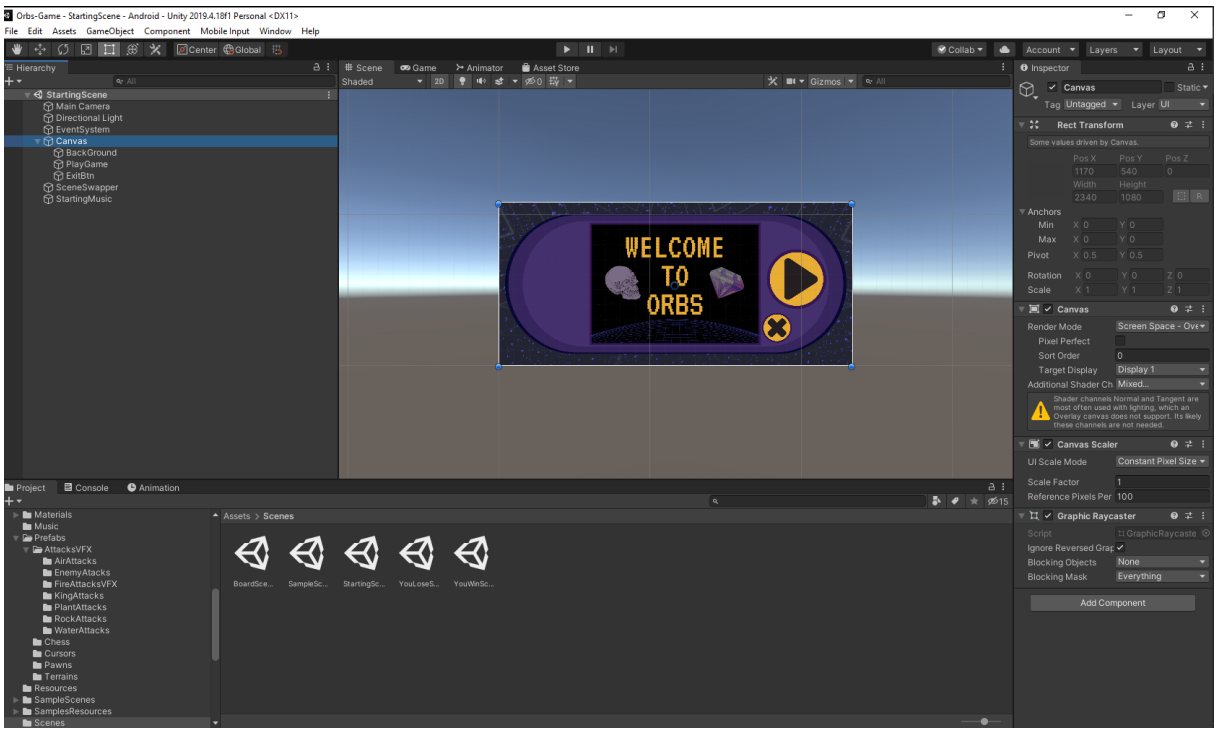
Σε αυτό το κεφάλαιο θα αναλυθεί η δομή της εφαρμογής και τα επιμέρους δομικά στοιχεία τα οποία την απαρτίζουν. Η εφαρμογή Orbs αποτελείται από 4 σκηνές οι οποίες με την σειρά τους περιέχουν όλα τα υπόλοιπα δομικά στοιχεία. Ξεκινώντας από κάθε σκηνή και συνεχίζοντας στα περιεχόμενα της θα αναλυθεί ο ρόλος και η λειτουργικότητα του κάθε στοιχείου χωρίς όμως να γίνει παρουσίαση του κώδικα καθώς αυτή θα γίνει στο επόμενο κεφάλαιο.

5.2 Σκηνές

Όπως αναφέρθηκε παραπάνω η εφαρμογή Orbs αποτελείται από 4 κύριες σκηνές (scenes).

5.2.1 Starting Scene

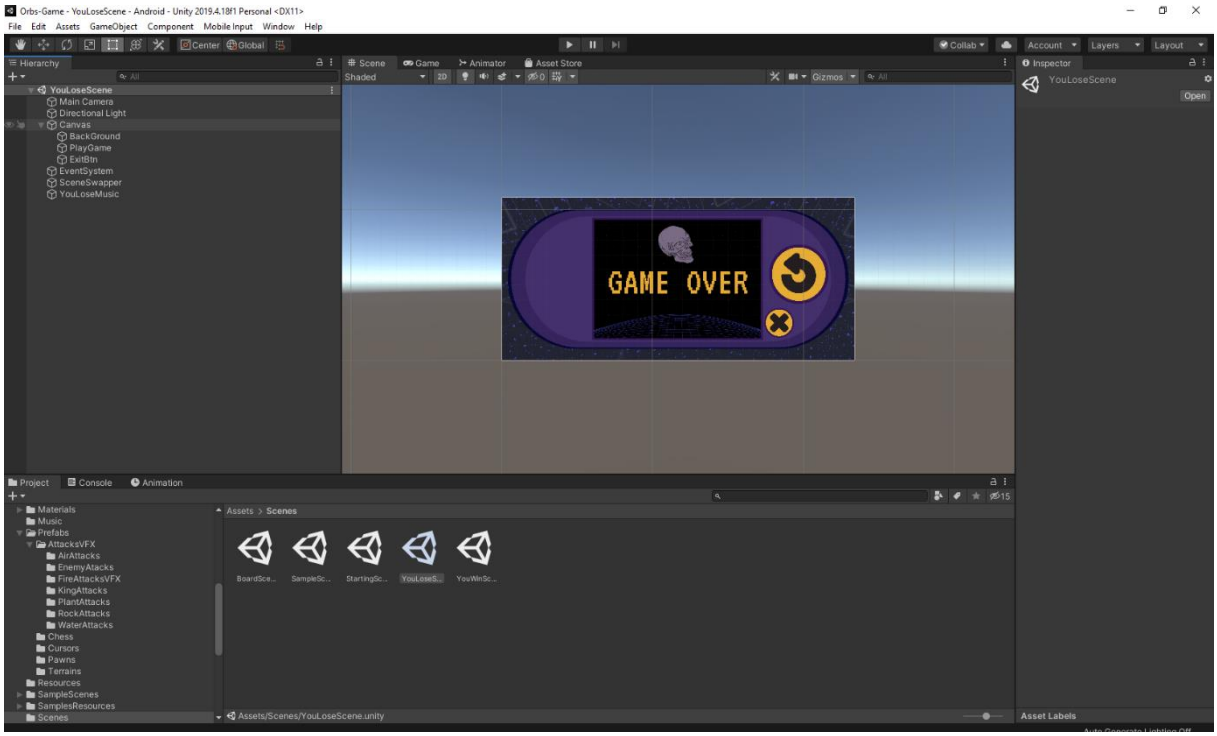
Η αρχική σκηνή (Starting Scene) είναι η πρώτη οθόνη που συναντά ο παίκτης κατά την είσοδο του στην εφαρμογή. Η αρχική σκηνή περιλαμβάνει ένα background image με το μήνυμα καλωσορίσματος του παίκτη ένα κουμπί Play και ένα κουμπί Exit. Οι λειτουργίες των κουμπιών Play και Exit είναι η μετάβαση του παίκτη στην σκηνή του παιχνιδιού (Board Scene) και η έξοδος από την εφαρμογή αντίστοιχα. Στην Starting Scene προστέθηκε το αντικείμενο StartingMusic. Το αντικείμενο StartingMusic περιέχει ένα AudioSource του οποίου ο ρόλος είναι να παίζει σε επανάληψη το μουσικό κομμάτι Lion’s Pride από το δημοφιλές παιχνίδι World of Warcraft της Blizzard Entertainment.



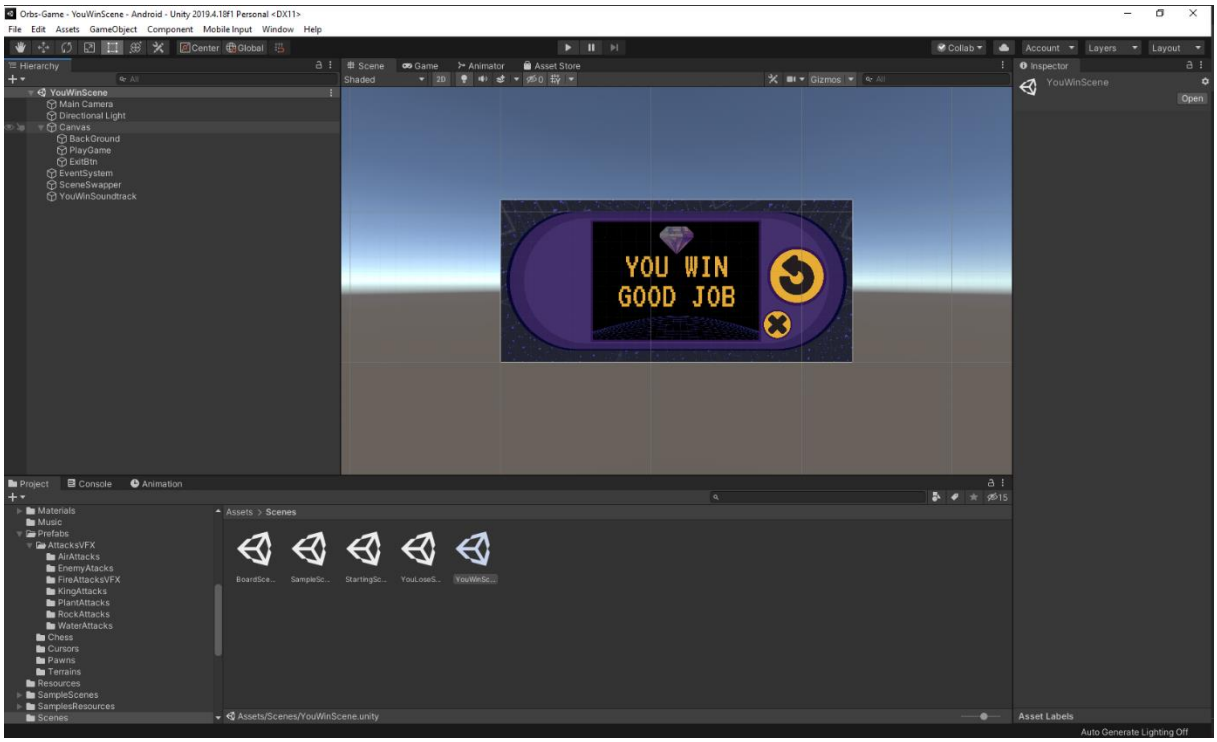
Σχήμα 5.1 : Δομή StartingScene

5.2.2 Ending Scene

Ο παίκτης εισέρχεται στην τελική σκηνή (ending scene) μετά την ολοκλήρωση κάθε παρτίδας. Η τελική σκηνή αποτελείται από 2 πανομοιότυπες παραλλαγές τής ίδιας σκηνής με διαφοροποίηση στο μήνυμα τους ανάλογα με το αποτέλεσμα της πιο πρόσφατης παρτίδας (νίκη/ήττα).Πέρα από το μήνυμα νίκης ή ήττας οι σκηνές YouWinScene και YouLoseScene περιέχουν ένα κουμπί Replay για την έναρξη νέας παρτίδας και ένα κουμπί Exit για την έξοδο από την εφαρμογή. Αντίστοιχα με την αρχική σκήνη οι σκηνές YouWinScene και YouLoseScene έχουν τα αντικείμενα YouWinSoundtrack και YouLoseMusic τα οποία περιέχουν AudioSource αντικείμενα για την αναπαραγωγή των ηχητικών εφέ της νίκης και ήττας.



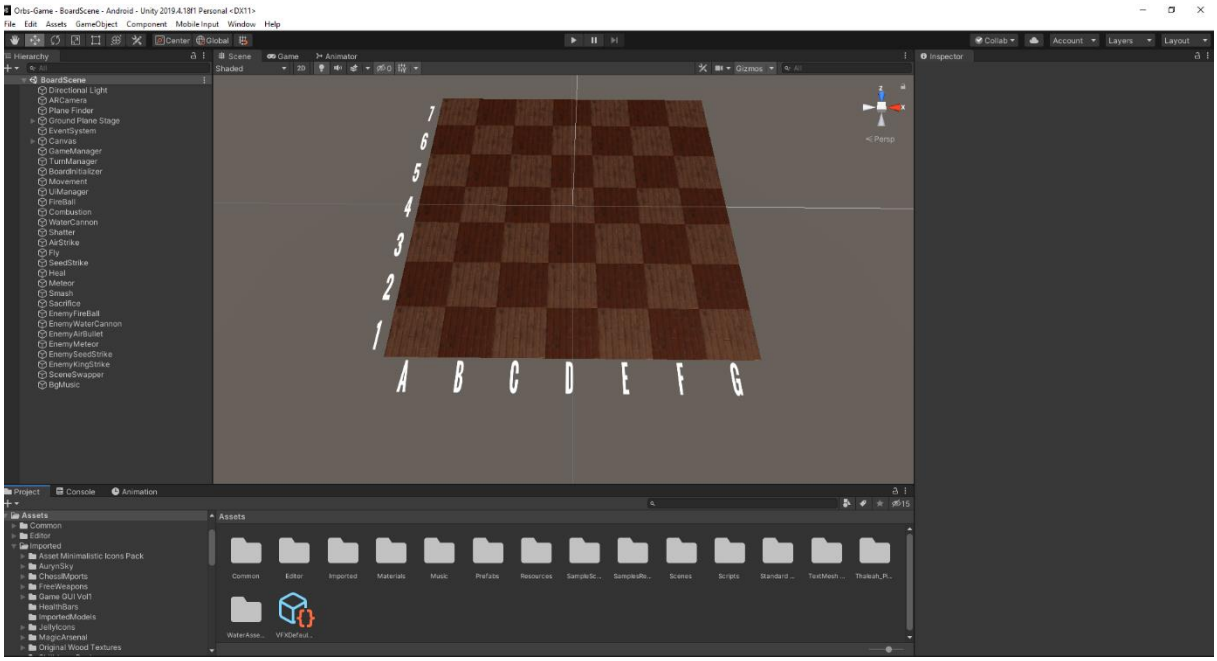
Σχήμα 5.2 : Δομή YouLoseScene



Σχήμα 5.3: Δομή YouWinScene

5.2.3 Board Scene

Η συντριπτική πλειοψηφία των δομικών στοιχείων της εφαρμογής βρίσκεται μέσα στην σκηνή του ταμπλό/παιχνιδιού (Board Scene) εντός της οποίας εξελίσσεται και η παρτίδα. Λόγω του δυσανάλογα μεγαλύτερου μεγέθους της σκηνής του παιχνιδιού σε σχέση με τις υπόλοιπες σκηνές τα δομικά της στοιχεία θα παρουσιαστούν ξεχωριστά στις παρακάτω ενότητες.

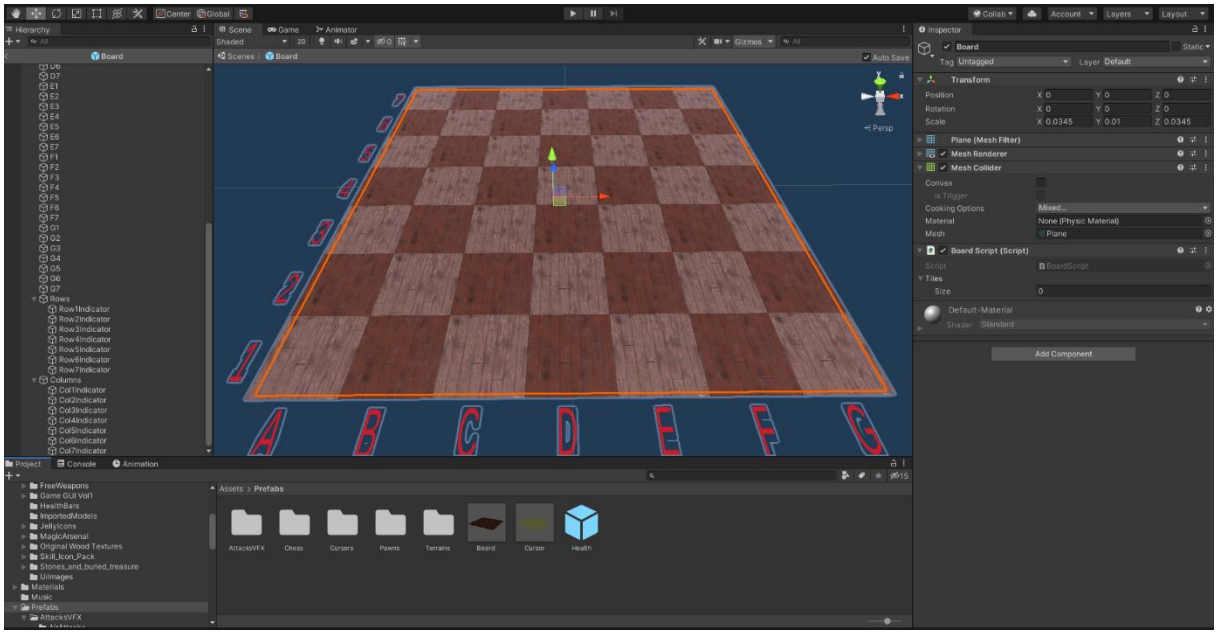


Σχήμα 5.4: Δομή Board Scene

Σχήμα 5.5 : Στιγμιότυπο οθόνης της Board Scene κατά την ροή του παιχνιδιού

5.2.4 Board

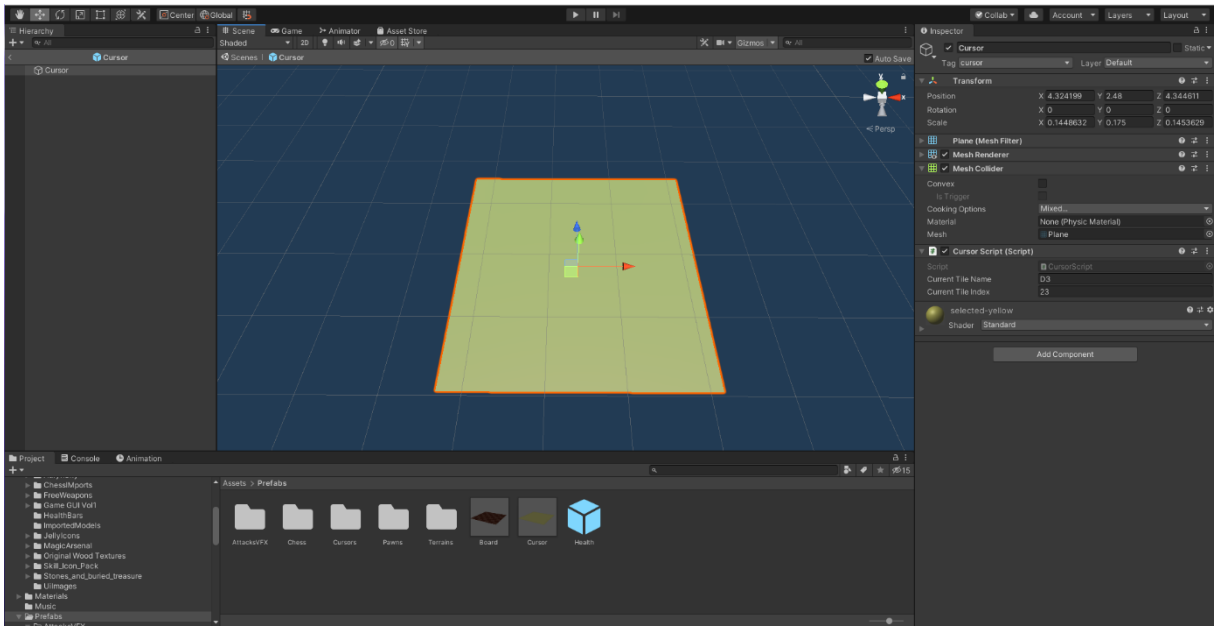
Το αντικείμενο Board το οποίο λειτουργεί ως το ταμπλό του παιχνιδιού αποτελείται από 49 planes διατεταγμένα οριζόντια και κάθετα. Τα αντικείμενα plane είναι τρισδιάστατα επίπεδα και ανήκουν στην συλλογή των προκαθορισμένων αντικειμένων της Unity. Στα planes του ταμπλό προστέθηκαν τα materials wooden floor 2 και wooden floor 4 για να αποδοθεί η υφή του ξύλου και να γίνουν διακριτά τα διαδοχικά τετράγωνα. Οι συλλογές Rows και Columns περιέχουν αντικείμενα text mesh pro για την σήμανση των γραμμών και στηλών αντίστοιχα.



Σχήμα 5.6: Δομή αντικειμένου Board

5.2.5 Cursor

Ο ρόλος του κέρσορα στο παιχνίδι Orbs καλύπτεται από το αντικείμενο Cursor . Το αντικείμενο Cursor είναι ένα ημιδιάφανο plane ίδιων διαστάσεων με τα planes του αντικείμενου Board . Για τον χρωματισμό του κέρσορα δημιουργήθηκε το material selected-yellow σε κίτρινη απόχρωση, για να επιτευχθεί η ημιδιάφανη όψη το rendering mode ορίστηκε σε fade και το alpha component του RGBA σε τιμή 165 (0=πλήρως διαφανές, 255=πλήρως αδιαφανές). Η λειτουργία του κέρσορα είναι να κινείται επάνω από τα τετράγωνα του ταμπλό τονίζοντας το τετράγωνο το οποίο έχει επιλέξει ο παίκτης.



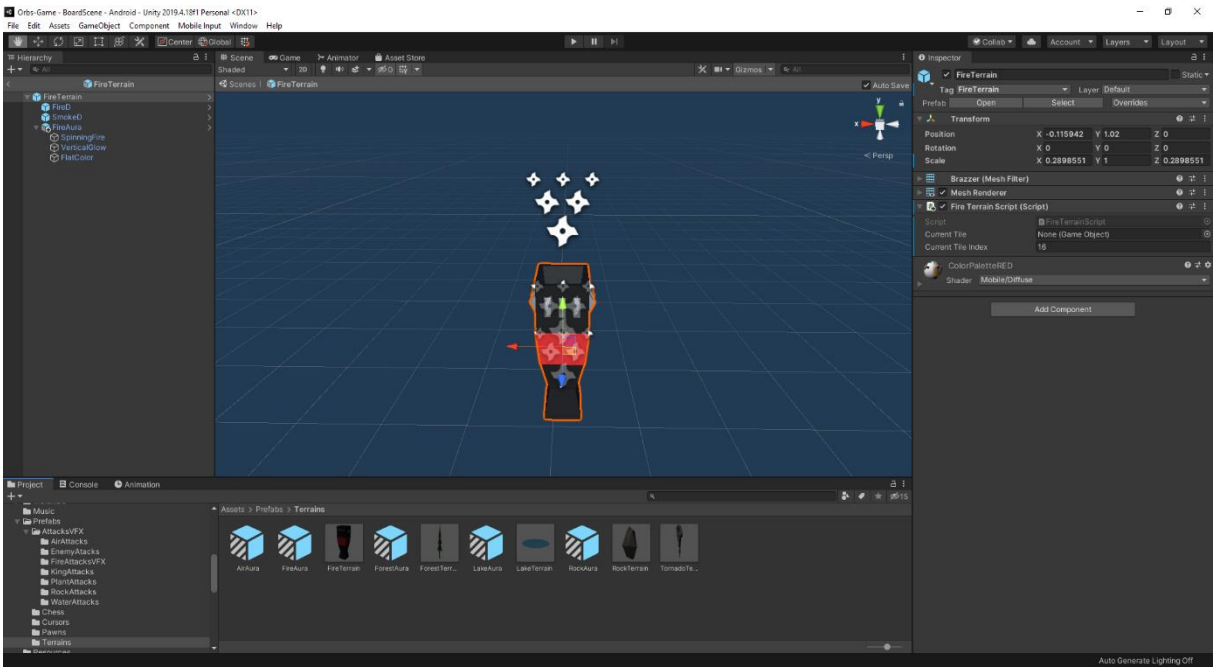
Σχήμα 5.7: Δομή αντικειμένου Cursor

5.2.6 Flag Objects

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο τα τετράγωνα ειδικού εδάφους σηματοδοτούνται από τα αντικείμενα σημαίες. Τα αντικείμενα σημαίες δεν βρίσκονται εξ ’αρχής στην σκηνή του ταμπλό αλλά δημιουργούνται δυναμικά κατά την έναρξη της παρτίδας επάνω στα τετράγωνα του ταμπλό τα οποία δεσμεύουν.

Fire Terrain

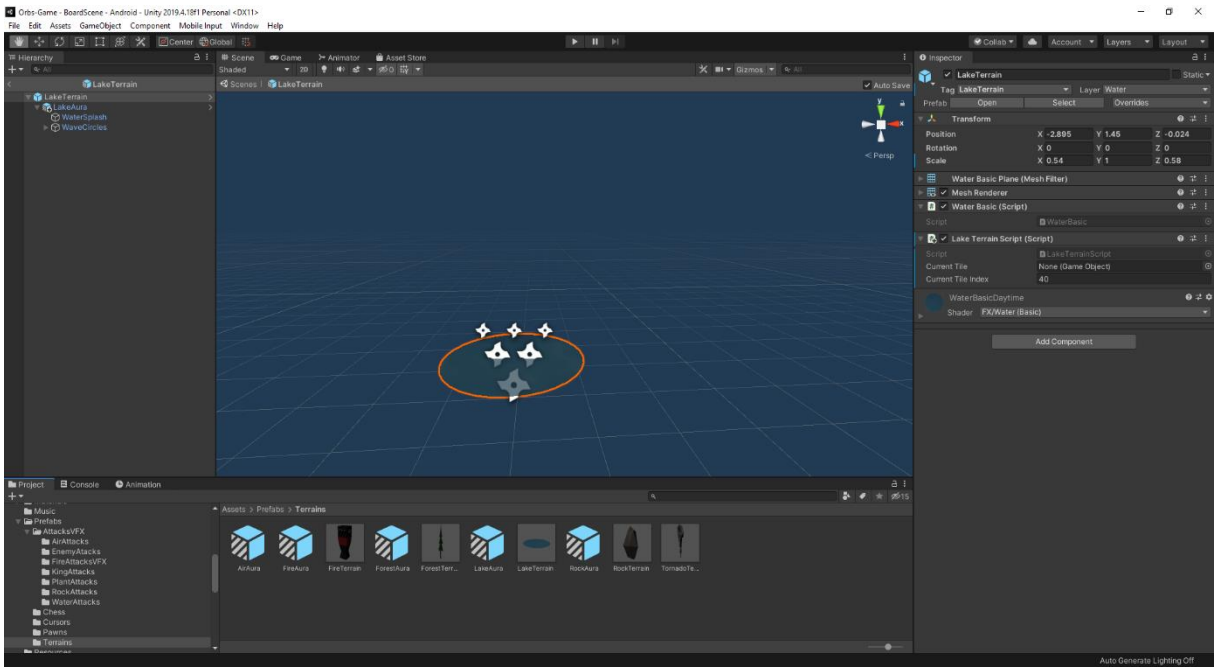
Το αντικείμενο Fire Terrain περιέχει μέσα του τα συστήματα σωματιδίων (particle systems) FireD για την προσομοίωση της φωτιάς, SmokeD για την προσομοίωση του καπνού και FireAura για την περιστρεφόμενη αύρα στο κάτω μέρος του.



Σχήμα 5.8: Δομή αντικειμένου FireTerrain

Lake Terrain

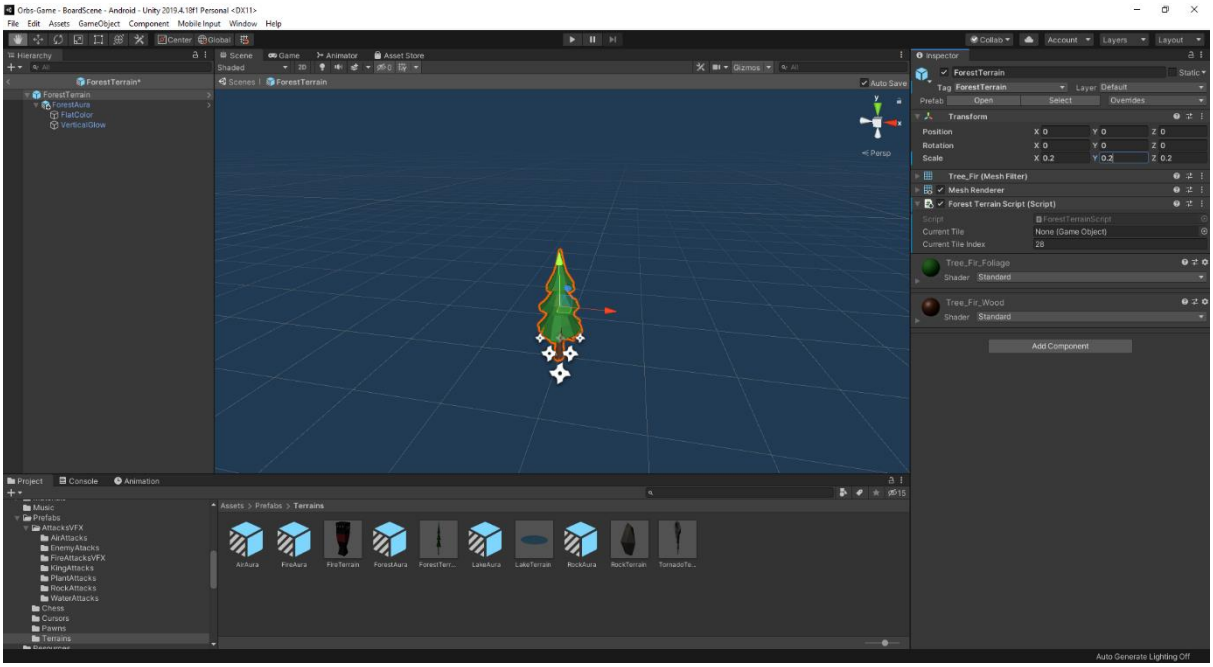
Το αντικείμενο Lake Terrain δημιουργήθηκε με βάση το αντικείμενο WaterBasicDaytime από το πακέτο Standard Assets του Unity για να πετύχει μια βασική προσομοίωση νερού. Επιπρόσθετα χρησιμοποιήθηκε το αντικείμενο LakeAura για την περιστρεφόμενη αύρα το κάτω μέρος του.



Σχήμα 5.9: Δομή αντικειμένου LakeTerrain

Forest Terrain

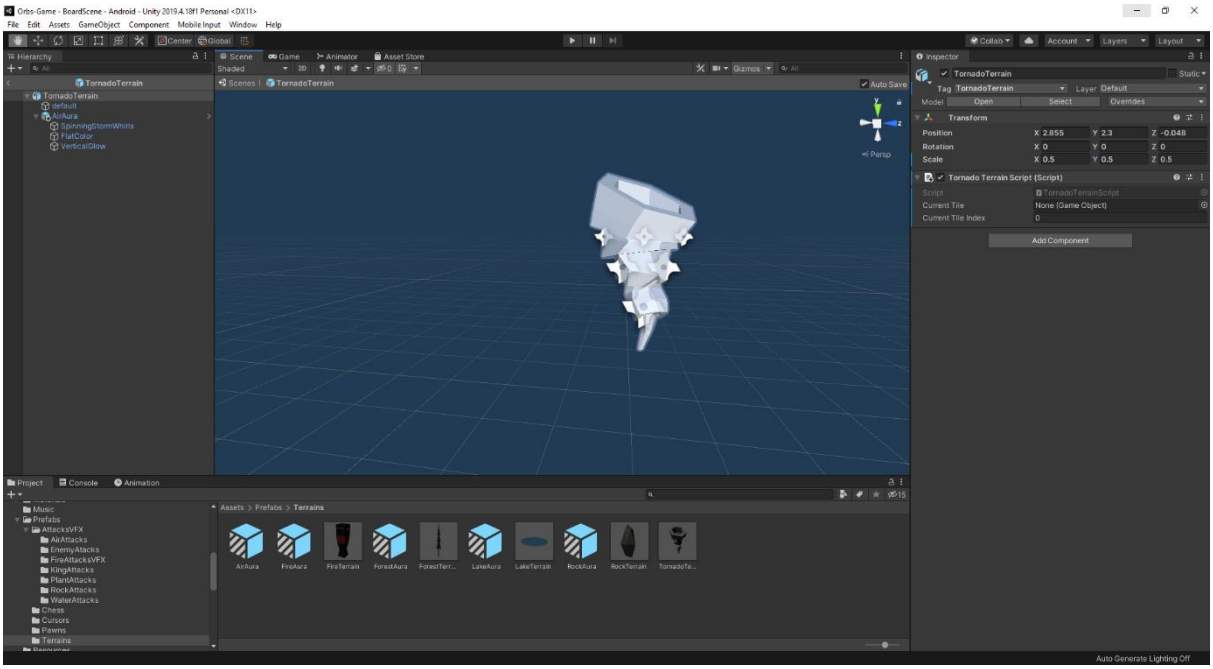
Το αντικείμενο Forest Terrain περιέχει το σύστημα σωματιδίων Forest Aura για την περιστρεφόμενη αύρα στο κάτω μέρος του.



Σχήμα 5.10: Δομή αντικειμένου ForestTerrain

Tornado Terrain

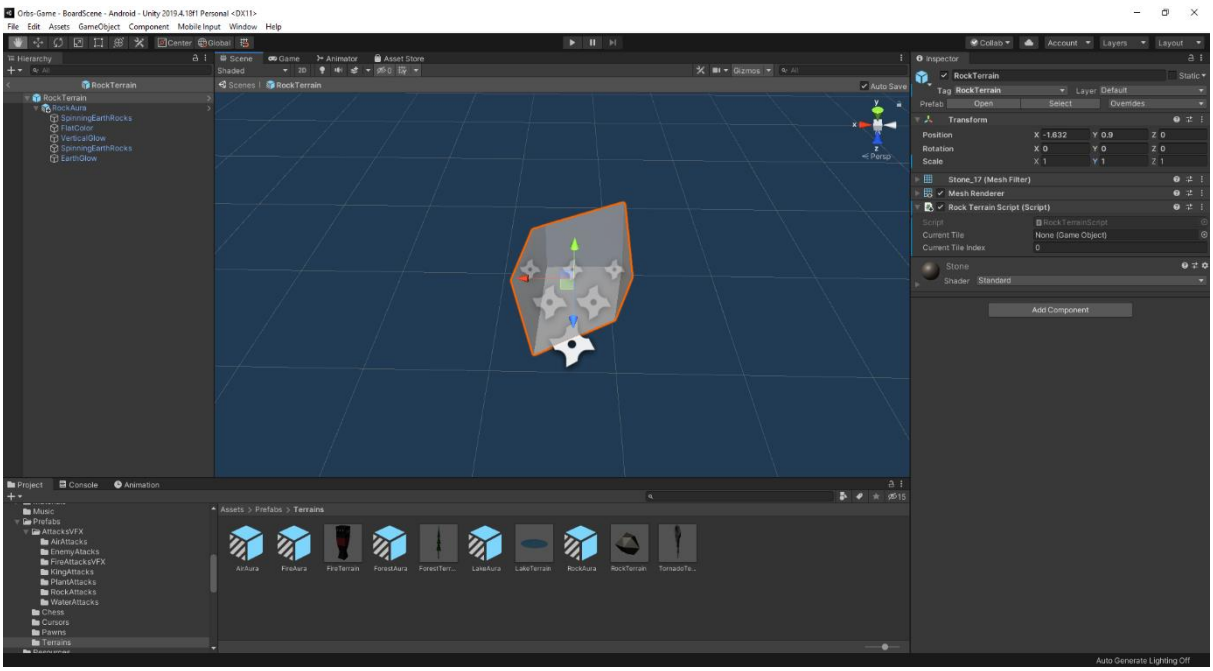
Το αντικείμενο Tornado Terrain περιέχει το σύστημα σωματιδίων Air Aura για την περιστρεφόμενη αύρα στο κάτω μέρος του.



Σχήμα 5.11: Δομή αντικειμένου TornadoTerrain

Rock Terrain

Το αντικείμενο Rock Terrain περιέχει το σύστημα σωματιδίων Rock Aura για την περιστρεφόμενη αύρα στο κάτω μέρος του.

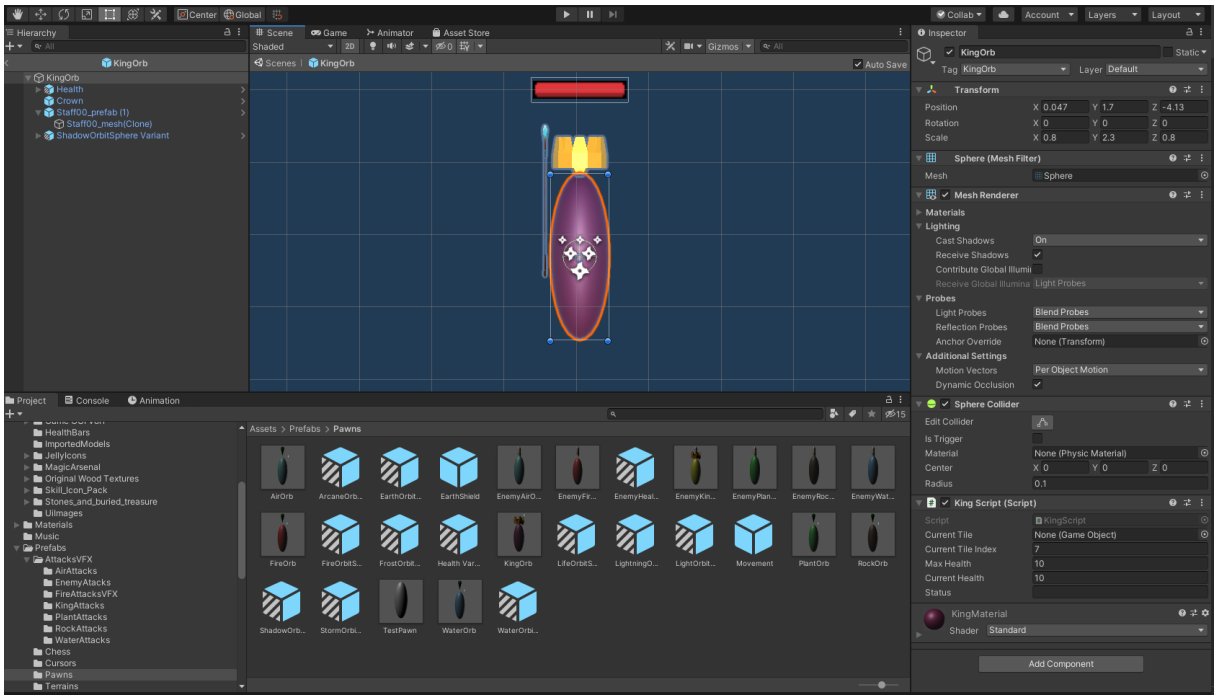


Σχήμα 5.12: Δομή αντικειμένου RockTerrain

Τα μοντέλα των αντικειμένων καθώς και τα συστήματα σωματιδίων με τα αντίστοιχα οπτικά εφέ προέρχονται από πακέτα του Unity Asset Store. Ως εκ τούτου δεν μπορεί να γίνει ανάλυση με περισσότερες λεπτομέρειες για τον σχεδιασμό τους.

5.2.7 Pawns

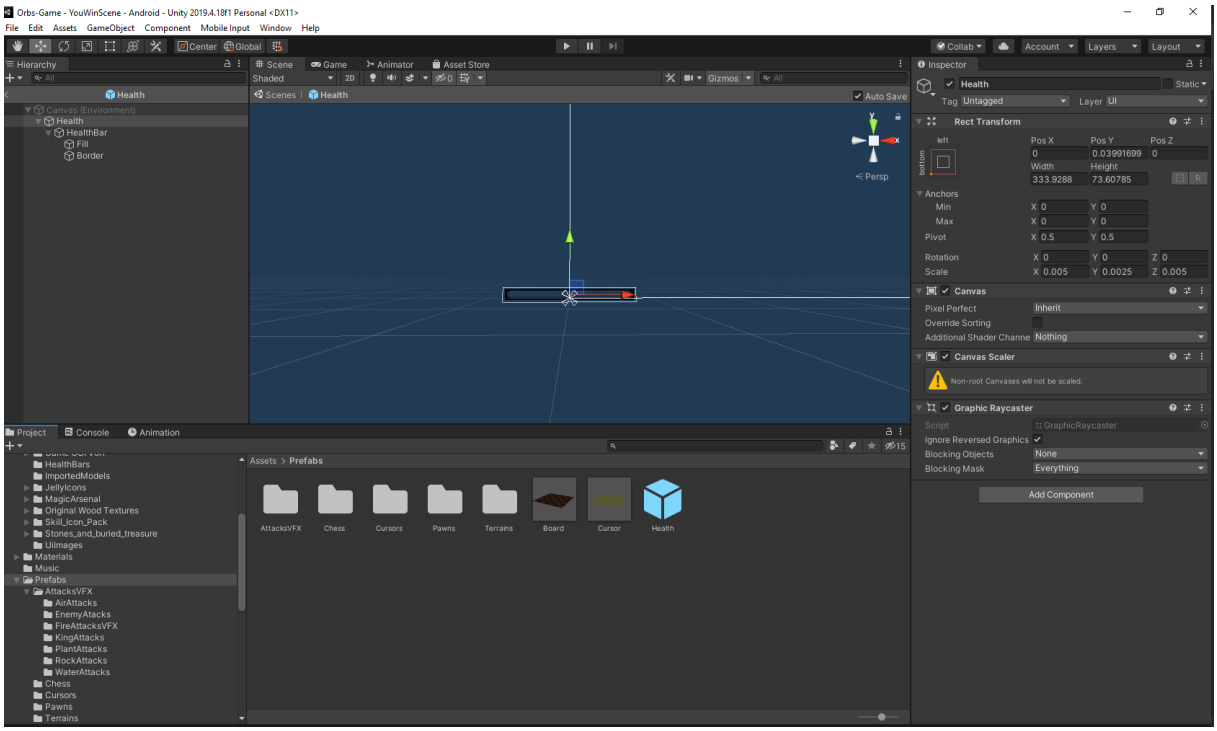
Η δομή των πιονιών αποτελείται από τα εξής επιμέρους κομμάτια:



Σχήμα 5.13: Γενική δομή πιονιών

Health Bar

Το αντικείμενο Health Bar σηματοδοτεί τους πόντους ζωής των πιονιών και εμφανίζεται ως μία κόκκινη μπάρα στο επάνω μέρος τους. Για την κατασκευή του Health Bar χρησιμοποιήθηκαν δύο image objects και ένα slider object.



Σχήμα 5.14: Δομή αντικειμένου HealthBar

Sphere

Ο πυρήνας του μοντέλου των πιονιών είναι η σφαίρα από τα βασικά σχήματα του unity. Στην σφαίρα του κάθε πιονιού δόθηκε και ένα διαφανές material για τον ανάλογο χρωματισμό.

Διακοσμητικά αντικείμενα

Στα πιόνια του παίκτη προστέθηκαν τα αντικείμενα staff00 και Emerald (το σκήπτρο στο πλάι και το πετράδι στο επάνω μέρος αντίστοιχα) ενώ στα πιόνια του αντιπάλου το αντικείμενο Skull0Closed (η νεκροκεφαλή στο επάνω μέρος τους). Τέλος στα πιόνια βασιλιάδες ανεξαρτήτως στρατοπέδου προστέθηκε το αντικείμενο Crown (η κορώνα στο επάνω μέρος τους).

VFX

Στο εσωτερικό των πιονιών τοποθετήθηκαν τα prefab αντικείμενα οπτικών εφέ της κατηγορίας orbital από το πακέτο Magic Arsenal με ανάλογο χρωματισμό για το κάθε πιόνι. Για τα Rock Orbs χρησιμοποιήθηκε επιπλέον το αντικείμενο EarthShield από την κατηγορία Shields του ίδιου πακέτου.

5.2.8 Ικανότητες

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο σε κάθε πιόνι αντιστοιχούν 1 ή περισσότερες ικανότητες. Για το οπτικό μέρος των ικανοτήτων των πιονιών χρησιμοποιήθηκαν διάφορα prefab αντικείμενα από το πακέτο Magic Arsenal. Στον παρακάτω πίνακα εμφανίζεται η αντιστοίχιση ικανοτήτων και οπτικών εφέ.

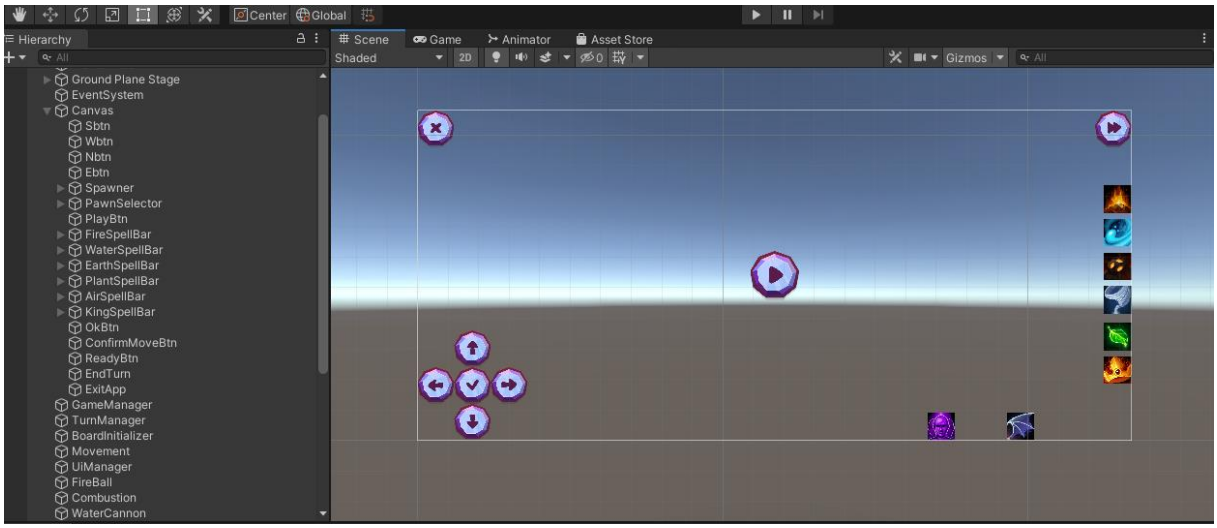
ΟΝΟΜΑ ΙΚΑΝΟΤΗΤΑΣ	ΟΠΤΙΚΟ ΕΦΕ	ΕΙΚΟΝΙΔΙΟ

Fireball	FireSmallObj	
Combustion	FirePillarBlast	
Water Cannon	WaterSpray	
Shatter	FrostDome	
Air Strike	StormSpray	
Fly	AuraCastStorm	
Seed Strike	PlantSmallObj	
Heal	CurseLife	
Meteor	RainEarth	
Sacrifice	AuraRingShadow	
Enemy Fireball	AuraSimpleFire	-
Air Bullet	AuraSimpleStorm	-
Enemy Water Strike	AuraSimpleWater	-
Enemy Meteor	AuraSimpleEarth	-
Enemy Seed Strike	AuraSimpleLife	-
King Attack	AuraSimpleLight	-

Πίνακας 5.1: Ικανότητες-Οπτικά Εφέ-Εικονίδια

5.2.9 Board Scene UI

Για την δημιουργία διεπαφής χρήστη στο Unity είναι απαραίτητο να εισαχθούν τα αντικείμενα Canvas και Event System. Η λειτουργία του αντικειμένου Event System είναι να στέλνει τα γεγονότα όπως τα κλικ του ποντικιού ή το πάτημα ενός κουμπιού στα αντικείμενα της εφαρμογής ενώ το αντικείμενο canvas λειτουργεί ως το δοχείο εντός του οποίου θα τοποθετηθούν τα διάφορα στοιχεία της διεπαφής.



Σχήμα 5.15: Δομή αντικειμένου Canvas στην BoardScene

Όπως φαίνεται στο σχήμα 5.15 το άσπρο περίγραμμα είναι το αντικείμενο Canvas.

Στην επάνω αριστερή γωνία του καμβά βρίσκεται το κουμπί ExitApp το οποίο κλείνει την σκηνή του παιχνιδιού και μεταβαίνει στην αρχική σκηνή κατά το πάτημα του.

Στην κάτω αριστερή γωνία βρίσκονται τα Sbtn (κάτω βελάκι), Wbtn (αριστερό βελάκι), Nbtn (επάνω βελάκι) και Ebtn (δεξί βελάκι) τα οποία αλλάζουν την θέση του κέρσορα επάνω στο ταμπλό κατά ένα τετράγωνο προς την ανάλογη κατεύθυνση.

Στο κέντρο της κάτω αριστερής γωνίας υπάρχουν 2 ίδια κουμπιά με το σύμβολο ✓ το ConfirmMoveBtn και το OkBtn. Το OkBtn τοποθετεί το επιλεγμένο πιόνι στο τετράγωνο επάνω στο οποίο βρίσκεται ο κέρσορας κατά την φάση του στησίματος και είναι εμφανές μόνο κατά την διάρκεια αυτής. Το ConfirmMoveBtn μετακινεί το επιλεγμένο πιόνι επάνω στο τετράγωνο στο οποίο βρίσκεται ο κέρσορας και είναι εμφανές μόνο μετά την λήξη της φάσης του στησίματος.

Στην επάνω αριστερή γωνία του βρίσκονται 2 επικαλυπτόμενα κουμπιά το ReadyBtn (σύμβολο >) και το EndTurn (διπλό δεξί βελάκι). Το ReadyBtn είναι εμφανές μόνο κατά την διάρκεια της φάσης του στησίματος και χρησιμοποιείται για να την λήξει και να μεταβεί ο παίκτης στην παρτίδα εφόσον όλα του τα πιόνια είναι τοποθετημένα στο ταμπλό. Το κουμπί EndTurn είναι εμφανές μόνο μετά την λήξη της φάσης του στησίματος και λειτουργεί για να λήξει τον γύρο του παίκτη και να αρχίσει ο γύρος του αντιπάλου. Στο κέντρο του καμβά βρίσκεται το κουμπί PlayBtn το οποίο είναι εμφανές μόνο πριν την έναρξη της φάσης του στησίματος. Το πάτημα του PlayBtn στήνει τα πιόνια του αντιπάλου και εισάγει τον παίκτη στην φάση του στησίματος.

Στην μέση της δεξιάς πλευράς του καμβά βρίσκονται 2 ίδια αντικείμενα τα οποία λειτουργούν ως ομάδες κουμπιών το αντικείμενο Spawner και το αντικείμενο PawnSelector.Το αντικείμενο Spawner το οποίο περιέχει τα κουμπιά FireSpawnBtn, WaterSpawnBtn, EarthSpawnBtn, AirSpawnBtn, PlantSpawnBtn και KingSpawnBtn είναι εμφανές μόνο κατά την διάρκεια της φάσης του στησίματος. Κάθε ένα από τα κουμπιά του Spawner επιλέγει το αντίστοιχο πιόνι του παίκτη για τοποθετηθεί στο τετράγωνο του ταμπλό στο οποίο βρίσκεται ο κέρσορας μετά το πάτημα του OkBtn. Στην περίπτωση που το πιόνι το οποίο επιλέγει ο παίκτης έχει τοποθετηθεί ήδη τότε εκείνο εξαφανίζεται προκειμένου να τοποθετηθεί ξανά. Το αντικείμενο PawnSelector είναι εμφανές μόνο μετά την λήξη της φάσης του

στησίματος και περιέχει τα κουμπιά FireSelectBtn, WaterSelectBtn, EarthSelectBtn, AirSelectBtn, PlantSelectBtn και KingSelectBtn . Όμοια με το αντικείμενο Spawner τα κουμπιά του PawnSelector επιλέγουν το αντίστοιχο πιόνι του παίκτη εφόσον αυτό υπάρχει για να μετακινηθεί στο τετράγωνο του κέρσορα μετά το πάτημα του ConfirmMoveBtn.

Την κάτω δεξιά γωνία του καμβά καταλαμβάνουν τα αντικείμενα FireSpellBar, WaterSpellBar, EarthSpellBar, PlantSpellBar, AirSpellBar και KingSpellBar τα οποία περιέχουν με την σειρά τους τα κουμπιά που αντιστοιχούν στις ικανότητες των πιονιών ως εξής :

- FireSpellBar - FireBallBtn, CombustionBtn
- WaterSpellBar - WaterCannonBtn, ShatterBtn
- EarthSpellBar – MeteorBtn
- PlantSpellBar - SeedStrikeBtn, HealBtn
- AirSpellBar - AirStrikeBtn, FlyBtn
- KingSpellBar - SacrificeBtn

Το κάθε αντικείμενο spellbar και τα κουμπιά του είναι εμφανή μόνο όταν επιλέγεται το αντίστοιχο πιόνι από κουμπί του αντικειμένου PawnSelector. Όλα τα αντικείμενα spellbar καθώς και το αντικείμενο PawnSelector εξαφανίζονται κατά την διάρκεια του γύρου του αντιπάλου και επανεμφανίζονται όταν είναι η σειρά του παίκτη.

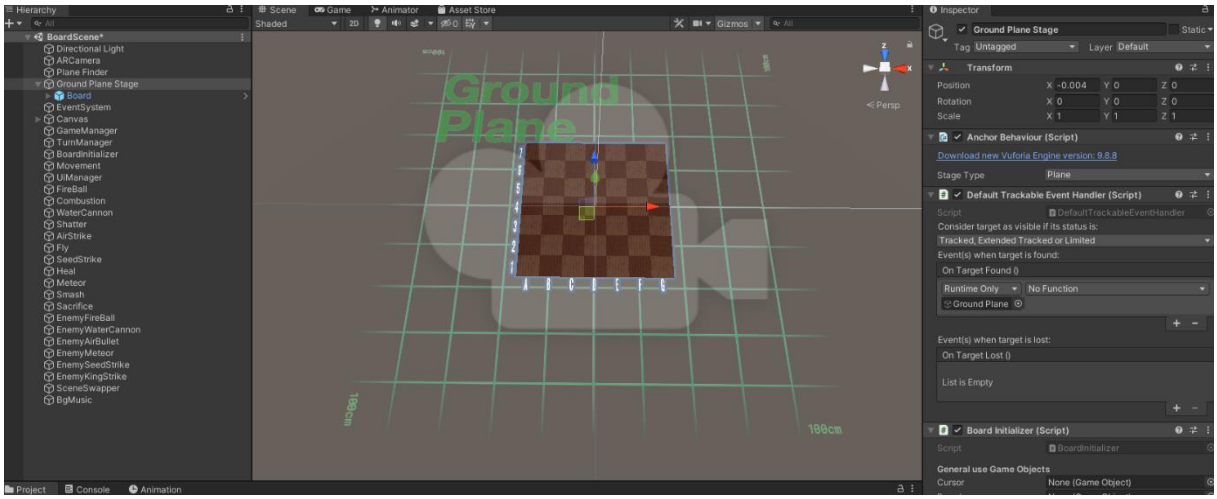
Για το εικαστικό μέρος των κουμπιών χρησιμοποιήθηκαν εικόνες από τα πακέτα Game GUI Vol1 και Skill Icon Pack του Unity Asset Store.

5.2.10 Μουσική υποβάθρου

Στην σκηνή του παιχνιδιού BoardScene προστέθηκε ένα αντικείμενο BgMusic. Το αντικείμενο BgMusic περιέχει ένα AudioSource του οποίου ο ρόλος είναι να παίζει σε επανάληψη την μουσική υποβάθρου κατά την διάρκεια της παρτίδας. Το κομμάτι που επιλέχθηκε για το συγκεκριμένο AudioSource είναι το Tricks of the Trade του Peter McConnel από το δημοφιλές παιχνίδι Hearthstone της Blizzard Entertainment.

5.2.11 Αντικείμενα Vuforia

Η σκηνή του παιχνιδιού περιέχει 3 αντικείμενα της μηχανής Vuforia τα οποία είναι υπεύθυνα για την τις λειτουργίες τις επαυξημένης πραγματικότητας στο παιχνίδι Orbs. Τα αντικείμενα αυτά είναι: ARCamera, Ground Plane Stage και Plane Finder. Η ARCamera αντικαθιστά το αντικείμενο Camera του Unity στον ρόλο της εμφάνισης των αντικειμένων. Με την χρήση των αντικειμένων Ground Plane Stage και Plane Finder η εφαρμογή ανιχνεύει τις επίπεδες επιφάνειες όπως ένα πάτωμα ή ένα τραπέζι και εμφανίζει τα αντικείμενα τις σκηνής μέσα στον χώρο τον οποίο ορίζει το αντικείμενο Ground Plane Stage όπως φαίνεται στο σχήμα 5.16.



Σχήμα 5.16: Αντικείμενα Vuforia στην BoardScene

5.3 Επίλογος

Η εφαρμογή Orbs θα μπορούσε να χωριστεί νοητά σε δυο μέρη, το παιχνίδι και το μενού. Η σκηνή BoardScene στην οποία εξελίσσεται η παρτίδα αποτελεί το μέρος του παιχνιδιού και περιέχει όλα τα σχετικά δομικά συστατικά όπως το ταμπλό και τα πιόνια. Αντίθετα οι σκηνές StartingScene, YouWinScene και YouLoseScene αποτελούν το μενού και ορίζουν την πλοήγηση του χρήστη στην εφαρμογή. Η συγκεκριμένη δομή εξυπηρετεί και τυχόν μελλοντικές επεκτάσεις και αλλαγές στην εφαρμογή Orbs τόσο για το μέρος του παιχνιδιού όσο και για τον εμπλουτισμό επιλογών στο μενού.

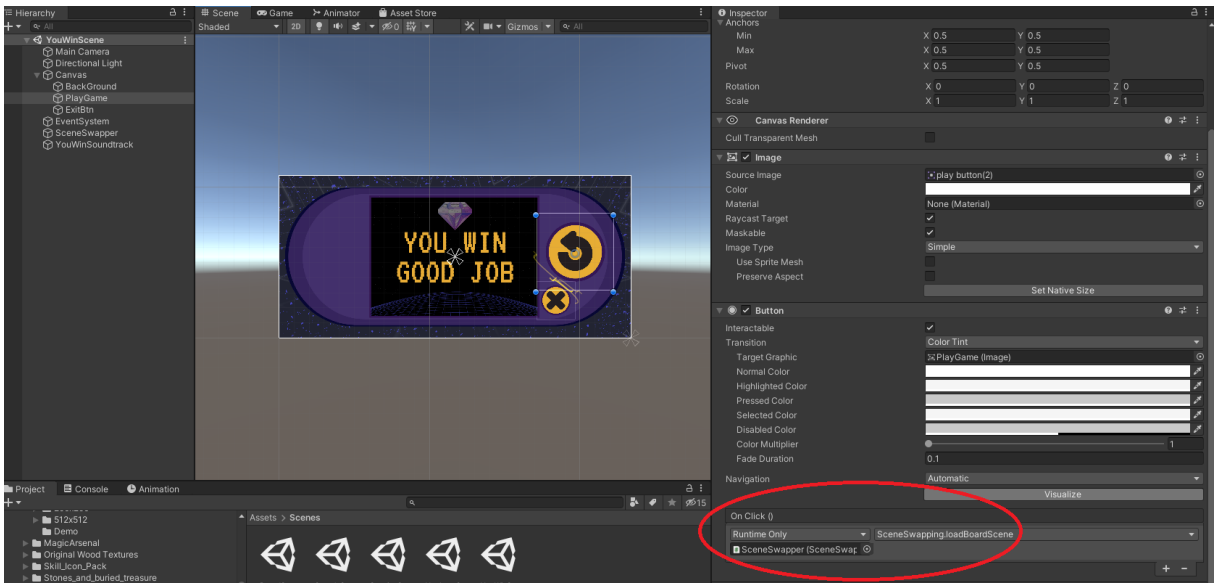
Κεφάλαιο 6ο: Λειτουργικότητα & κώδικας

6.1 Εισαγωγή

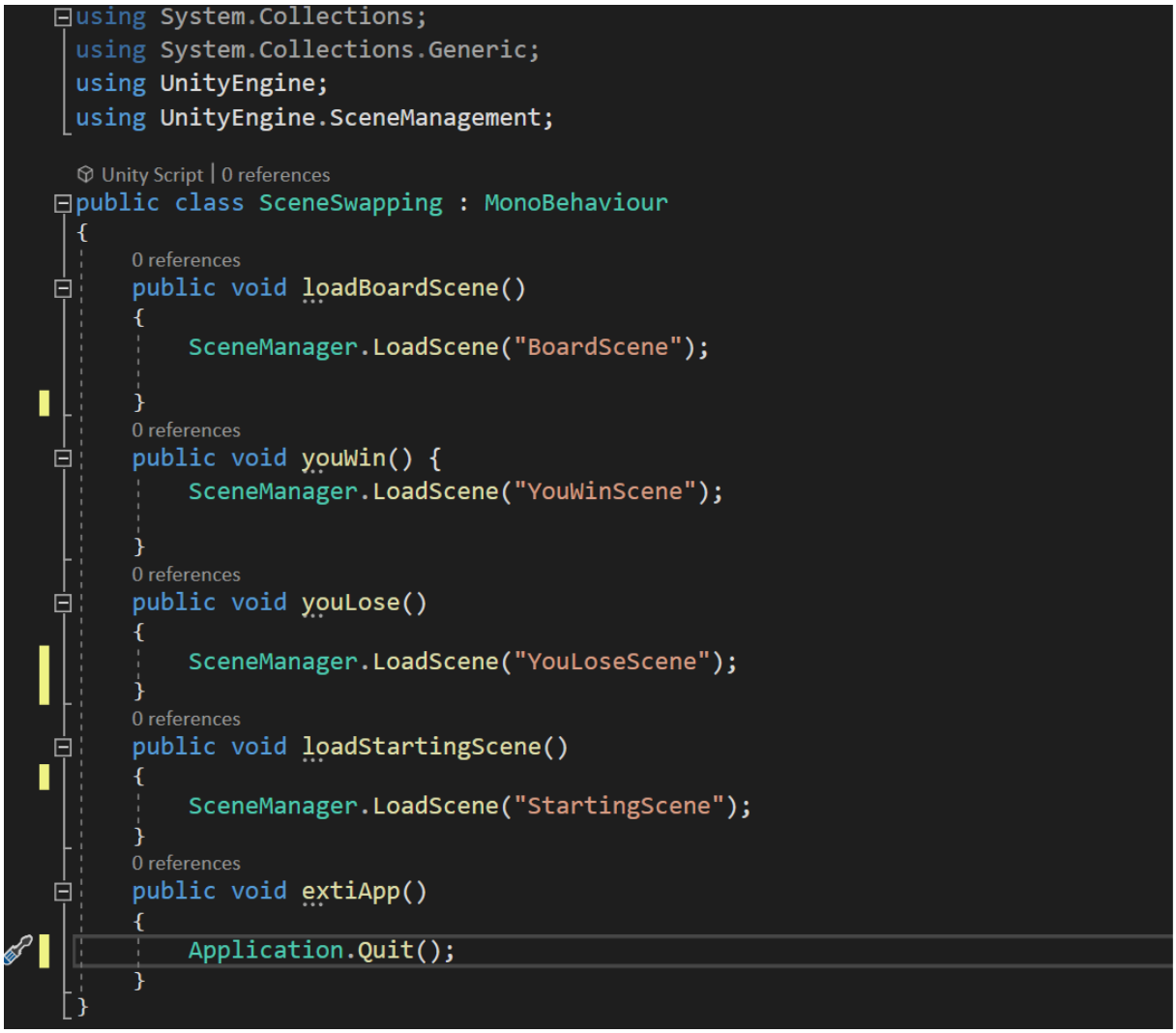
Η λειτουργικότητα και η συμπεριφορά των αντικειμένων του Unity ορίζονται από τον κώδικα τον οποίο περιέχουν τα διάφορα αρχεία script της εφαρμογής. Για να επιτευχθεί η συσχέτιση ανάμεσα σε αντικείμενα και script πρέπει το script να οριστεί σαν συστατικό (component) του αντικειμένου. Ένα αντικείμενο μπορεί να έχει παραπάνω από ένα script σαν component και ένα script μπορεί με την σειρά του να τεθεί σαν component σε παραπάνω από ένα αντικείμενα. Σε αυτό το κεφάλαιο θα γίνει η παρουσίαση του κώδικα αναλύοντας ξεχωριστά το κάθε script, τον ρόλο του και τον τρόπο με τον οποίο αλληλεπιδρά με τα αντικείμενα της εφαρμογής και τα υπόλοιπα αρχεία script.

6.2 Εναλλαγή Σκηνών

Ο ρόλος του script SceneSwapping.cs είναι η πλοήγηση του παίκτη μεταξύ των διαφόρων σκηνών του παιχνιδιού.Το script SceneSwapping.cs κάνει χρήση του namespace UnityEngine.SceneManagement για να αποκτήσει πρόσβαση στην κλάση SceneManager η οποία παρέχει την επιθυμητή λειτουργικότητα για την πλοήγηση μεταξύ των σκηνών. Το script SceneSwapping.cs περιέχει 4 μεθόδους : loadBoardScene(), youWin(), youLose(), loadStartingScene() και exitApp() . Οι 4 πρώτες έχουν την εντολή SceneManager.LoadScene (Όνομα σκηνής) η οποία μεταβαίνει στην σκήνη της οποίας το όνομα δίνεται σαν παράμετρος. Η μέθοδος exitApp() περιέχει την εντολή Application.Quit() η οποία κλείνει την εφαρμογή. Για να γίνει ορατό το script SceneSwapping.cs και να μπορεί να χρησιμοποιηθεί εντός των σκηνών η κάθε σκηνή περιέχει ένα GameObject (αντικείμενο) SceneSwapper το οποίο έχει ως component το εν λόγω script. Για να γίνει η κλήση των μεθόδων του SceneSwapping.cs από τα κουμπιά του UI πρέπει να περαστεί το αντικείμενο SceneSwapper μαζί με την επιθυμητή μέθοδο στα On Click event του αντίστοιχου του κουμπιού όπως φαίνεται στο σχήμα 6.1 προκειμένου να επιτευχθεί η λειτουργικότητα που αναφέρθηκε στο προηγούμενο κεφάλαιο.



Σχήμα 6.1: Παράδειγμα σύνδεσης διεπαφής με το SceneSwapping.cs



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

Unity Script | 0 references
public class SceneSwapping : MonoBehaviour
{
    0 references
    public void loadBoardScene()
    {
        SceneManager.LoadScene("BoardScene");

    }
    0 references
    public void youWin() {
        SceneManager.LoadScene("YouWinScene");

    }
    0 references
    public void youLose()
    {
        SceneManager.LoadScene("YouLoseScene");
    }
    0 references
    public void loadStartingScene()
    {
        SceneManager.LoadScene("StartingScene");
    }
    0 references
    public void extiApp()
    {
        Application.Quit();
    }
}
```

Σχήμα 6.2: Κώδικας SceneSwapper.cs

6.3 Terrain Scripts

Κάθε αντικείμενο ειδικού εδάφους έχει και ένα terrain script ως component για την λειτουργικότητα του . Τα script ειδικού εδάφους (FireTerrain.cs, RockTerrain.cs, TornadoTerrain.cs, ForestTerrain.cs, LakeTerrain.cs και FireTerrain.cs) δεν περιέχουν κώδικα αλλά κληρονομούν τις ιδιότητες του script-γονιού MasterTerrain.cs. Με την σειρά του το MasterTerrain.cs περιέχει τις μεταβλητές currentTile και currentTileIndex . Η μεταβλητή currentTile αποθηκεύει ένα αντικείμενο tile (τετράγωνο του ταμπλό) ενώ η currentTileIndex τον δείκτη του tile στην λίστα tiles των άλλων script. Η χρήση αυτών των δύο μεταβλητών επιτρέπει στο πρόγραμμα να γνωρίζει τις θέσεις του ταμπλό στις οποίες τοποθετούνται τα τετράγωνα ειδικού εδάφους και τα αντικείμενα σημαίες κατά την εκκίνηση της παρτίδας.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
   Unity Script | 12 references
5  public class MasterTerrain : MonoBehaviour
6
7
8  {
9      public GameObject currentTile;
10     public int currentTileIndex;
11     // Start is called before the first frame update
       Unity Message | 0 references
12     void Start()
13     {
14
15     }
16
17     // Update is called once per frame
       Unity Message | 0 references
18     void Update()
19     {
20
21     }
22 }
23
```

Σχήμα 6.3: Κώδικας MasterTerrain.cs

6.4 Pawn Scripts

Τα script τα οποία σχετίζονται με τα αντικείμενα των πιονιών χωρίζονται σε δύο κατηγορίες, Enemy Scripts:

- EnemyAirPawnScript.cs
- EnemyFirePawnScript.cs
- EnemyPlantPawnScript.cs
- EnemyRockPawnScript.cs
- EnemyWaterPawnScript.cs
- EnemyKingPawnScript.cs

για τα πιόνια του υπολογιστή και τα Player Scripts:

- AirPawnScript.cs
- FirePawnScript.cs
- PlantPawnScript.cs
- RockPawnScript.cs
- WaterPawnScript.cs
- EnemyKingPawnScript.cs

για τα πιόνια του παίκτη. Από πλευράς κώδικα τα script των παραπάνω κατηγοριών περιέχουν μόνο τον έλεγχο των πόντων ζωής εντός της μεθόδου update έτσι ώστε στην περίπτωση που οι πόντοι φτάσουν στο 0 να αφαιρεθεί το αντίστοιχο πιόνι από το ταμπλό.

6.4.1 MasterPawn.cs

Τα Player Scripts και Enemy Scripts κληρονομούν το script-γονιό MasterPawn.cs το οποίο περιέχει τον κορμό της λειτουργικότητας των πιονιών.

6.4.1.1 Μεταβλητές MasterPawn.cs

Το script MasterPawn.cs περιέχει 5 μεταβλητές. Οι μεταβλητές currentTile και currentTile index χρησιμοποιούνται για την ανίχνευση του τετραγώνου του ταμπλό στο οποίο βρίσκεται το πιόνι αντίστοιχα με τις ομώνυμες μεταβλητές του MasterTerrain.cs. Η μεταβλητή maxHealth δείχνει τους μέγιστους πόντους ζωής ενός πιονιού ενώ η μεταβλητή currentHealth τους τρέχοντες πόντους ζωής του πιονιού. Τέλος η μεταβλητή status δείχνει εάν ένα πιόνι χτυπήθηκε από την ικανότητα shatter αλλάζοντας από status= “ “ σε status=”frozen”.

```
Unity Script | 90 references
public class MasterPawn : MonoBehaviour
{

    public GameObject currentTile;
    public int currentTileIndex;
    public int maxHealth=10;
    public int currentHealth = 10;
    public string status = "";

```

Σχήμα 6.4: Μεταβλητές MasterPawn.cs

6.4.1.2 Μέθοδοι MasterPawn.cs

Η μέθοδος takeDamage() λαμβάνει σαν παράμετρο τους πόντους ζημιάς οι οποίοι υπολογίζονται στο script GameManager.cs, τους αφαιρεί από την μεταβλητή currentHealth και στην συνέχεια προσαρμόζει την μπάρα ζωής του πιονιού ανάλογα με τους υπολειπόμενους πόντους ζωής.

Η μέθοδος takeHealing() καλείται όταν γίνεται χρήση της ικανότητας Heal και προσθέτει το ανάλογο ποσό πόντων ζωής στο currentHealth του πιονιού το οποίο στόχευσε η ικανότητα. Στον υπολογισμό της πρόσθεσης πόντων ζωής λαμβάνεται υπόψη το γεγονός ότι οι πόντοι ζωής δεν μπορούν να ξεπεράσουν τους 10.

Η τελευταία μέθοδος του MasterPawn.cs ονομάζεται isGonnaDie() και καλείται από το script GameManager.cs κατά την ώρα που ένα πιόνι επιτίθεται σε ένα άλλο και στόχος της είναι να υπολογίσει εάν το πιόνι το οποίο δέχεται την επίθεση θα καταστραφεί επιστρέφοντας true ή false. Στην περίπτωση κατά την οποία η μέθοδος isGonnaDie επιστρέψει true το GameManager.cs καλεί μεθόδους από το script Movement.cs το οποίο θα αναφερθεί στην συνέχεια για να ενημερώσει τις λίστες οι οποίες παρακολουθούν τις θέσεις των πιονιών.

```
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }
    10 references
    public void takeDamage(int damage) {
        currentHealth -= damage;
        this.GetComponentInChildren<HealthBar>().SetHealth(currentHealth);
        Debug.Log(this.gameObject.tag +" Takes " + damage + " Damage ");
    }
    1 reference
    public void takeHealing (int healing)
    {
        if(currentHealth + healing > 10)
        {

            currentHealth = 10;
            this.GetComponentInChildren<HealthBar>().SetHealth(currentHealth);
        }
        else
        {
            currentHealth += healing;
            this.GetComponentInChildren<HealthBar>().SetHealth(currentHealth);
        }
    }
    10 references
    public bool isGonnaDie(int damage) {
        if (damage >= currentHealth)
        {
            return true;
        }
        else { return false; }
    }
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }
}
```

Σχήμα 6.5: Μέθοδοι MasterPawn.cs

6.4.2 Μπάρα Ζωής

Κάθε πιόνι περιέχει στην δομή του ένα αντικείμενο HealthBar [17] για την αναπαράσταση των πόντων ζωής του. Το script HealthBar.cs μέσω των μεθόδων SetMaxHealth() και SetHealth() (για τις μεταβλητές maxHealth και currentHealth των πιονιών αντίστοιχα) χειρίζεται τις διακυμάνσεις της μπάρας ζωής των πιονιών κατά την διάρκεια της παρτίδας.

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

Unity Script | 3 references
public class HealthBar : MonoBehaviour
{

    public Slider slider;
    // Start is called before the first frame update


    0 references
    public void SetMaxHealth(int health) {
        slider.maxValue = health;
    }
    3 references
    public void SetHealth(int health)
    {
        slider.value = health;
    }
}
```

Σχήμα 6.6: Κώδικας HealthBar.cs

6.5 Αρχικοποίηση Ταμπλό

Το script BoardInitializer.cs είναι υπεύθυνο για την αρχικοποίηση του ταμπλό κατά την έναρξη της παρτίδας. Η αρχικοποίηση του ταμπλό περιλαμβάνει μεταξύ άλλων την επιλογή των τετραγώνων ειδικού εδάφους την τοποθέτηση των flag objects αλλά και την τοποθέτηση των πιονιών των δύο παικτών. Το BoardInitializer.cs γίνεται ορατό στην σκηνή του παιχνιδιού μέσω του ομώνυμου αντικειμένου BoardInitializer στο οποίο τοποθετήθηκε ως component. Ο πλήρης κώδικας του script BoardInitializer βρίσκεται στο παράρτημα Β.

6.5.1 Μεταβλητές BoardInitilizer.cs

Οι μεταβλητές του script BoardInitilizer.cs χωρίζονται σε 3 κατηγορίες ανάλογα με την χρήση τους. Η πρώτη κατηγορία είναι οι μεταβλητές γενικής χρήσης και περιλαμβάνει τα παρακάτω GameObjects.

cursor: Περιέχει τον κέρσορα του ταμπλό.

board: Περιέχει το ταμπλό.

cursorTile: Περιέχει το αντικείμενο tile επάνω στο οποίο βρίσκεται ο κέρσορας. Είναι private μεταβλητή.

pawnToPlace: Περιέχει το πιόνι το οποίο επιλέγει ο χρήστης να τοποθετήσει κατά την φάση του στησίματος.

Η δεύτερη κατηγορία μεταβλητών είναι οι λίστες τετραγώνων του ταμπλό (Tile Lists). Οι λίστες τετραγώνων είναι οι παρακάτω.

tiles: Περιέχει όλα τα τετράγωνα του ταμπλό.

potentialEnemyTiles: Περιέχει τα τετράγωνα στα οποία οι κανόνες του παιχνιδιού επιτρέπουν να στηθεί πιόνι του αντιπάλου.

enemyKingTileList: Περιέχει τα τετράγωνα στα οποία οι κανόνες του παιχνιδιού επιτρέπουν να στηθεί το πιόνι-βασιλιάς του αντιπάλου.

enemyTiles: Περιέχει την λίστα των τετραγώνων τα οποία καταλαμβάνονται από τα πιόνια του αντιπάλου μετά την φάση του στησίματος.

allyTiles: Περιέχει την λίστα των τετραγώνων τα οποία καταλαμβάνονται από τα πιόνια του παίκτη μετά την φάση του στησίματος.

specialTerrainTiles: Περιέχει την λίστα των τετραγώνων ειδικού εδάφους μετά την διαδικασία του στησίματος.

Η Τρίτη και τελευταία κατηγορία μεταβλητών είναι τα κομμάτια του παιχνιδιού ή Pieces. Η κατηγορία αυτή περιέχει 3 λίστες αντικειμένων όπως φαίνεται παρακάτω.

enemyPieces: Περιέχει τα πιόνια του αντιπάλου.

playerPieces: Περιέχει τα πιόνια του παίκτη.

terrains: Περιέχει τα αντικείμενα σημαίες.

6.5.2 BoardInit

Η μέθοδος BoardInit καλείται με το πάτημα του PlayBtn της BoardScene η οποία με την σειρά της καλεί τις μεθόδους spawnCursor(), spwnSpecialTerrain() και setupEnemyPawns() τις οποίες θα αναλύσουμε παρακάτω για την αρχικοποίηση του ταμπλό.

6.5.3 spawnSpecialTerrain

Η μέθοδος spawnSpecialTerrain() επιλέγει τυχαία τετράγωνα από την λίστα tiles και τα ορίζει ως τετράγωνα ειδικού εδάφους αντιγράφοντας τα στην λίστα specialTerrainTiles.Στην συνέχεια γίνεται αντιστοίχιση ανάμεσα σε αντικείμενα σημαίες και τετράγωνα ειδικού εδάφους. Τέλος εμφανίζει τα αντικείμενα σημαίες στα κατάλληλα τετράγωνα του ταμπλό.

6.5.4 setupEnemyPawns

Η μέθοδος setupEnemyPawns() είναι υπεύθυνη για το στήσιμο των πιονιών του υπολογιστή. Αρχικά αφαιρούνται από την λίστα potentialEnemyTiles όλα τα τετράγωνα τα οποία ορίστηκαν ως ειδικό έδαφος έτσι ώστε να μην στηθεί πιόνι σε τετράγωνο το οποίο καταλαμβάνεται ήδη από ειδικό έδαφος. Στην συνέχεια κατασκευάζεται η enemyKingTileList, επιλέγεται τυχαία το τετράγωνο στο οποίο θα στηθεί ο βασιλιάς του αντιπάλου και τέλος τοποθετείται στο ταμπλό. Για το στήσιμο των υπολοίπων πιονιών η setupEnemyPawns() ακολουθεί μια παρόμοια διαδικασία επιλέγοντας διαδοχικά το κάθε πιόνι από την λίστα enemy Pieces και ένα τυχαίο αλλά διαφορετικό κάθε φορά τετράγωνο από την potentialEnemyTiles. Κατά την διαδικασία τοποθέτησης των πιονιών στα αρχικά τους τετράγωνα οι μεταβλητές currentTile και currentTileIndex των πιονιών αρχικοποιούνται ενώ ταυτόχρονα κατασκευάζεται και η λίστα enemyTiles.

6.5.5 spawnCursor

Η μέθοδος spawnCursor() εμφανίζει τον κέρσορα στο τετράγωνο D4 .

6.5.6 Τοποθέτηση πιονιών παίκτη.

Οι μέθοδοι kingPawnSelect(), firePawnSelect(), plantPawnSelect(), airPawnSelect(), rockPawnSelect(), και waterPawnSelect() καλούνται από τα κουμπιά του αντικειμένου spawner όπως αναφέρθηκε στην ενότητα 5.2.9 για να επιλέξει ο παίκτης το πιόνι το οποίο επιθυμεί να τοποθετήσει στο ταμπλό κατά την φάση του στησίματος . Εάν το πιόνι το οποίο επιλέγεται έχει τοποθετηθεί ήδη τότε εξαφανίζεται αλλιώς αντιγράφεται στην μεταβλητή pawnToPlace. Με το πάτημα του OkBtn καλείται η μέθοδος placeSelectedPawn() η οποία τοποθετεί το πιόνι το οποίο έχει περαστεί στην μεταβλητή pawnToPlace στο τετράγωνο που βρίσκεται ο κέρσορας. Πριν την τοποθέτηση του πιονιού η μέθοδος placeSelectedPawn ελέγχει την εγκυρότητα της τοποθέτησης του πιονιού βάσει των κανόνων του παιχνιδιού και της διαθεσιμότητας του τετράγωνου στο οποίο βρίσκεται ο κέρσορας. Με την επιτυχή τοποθέτηση του κάθε πιονιού οι μεταβλητές του currentTile και currentTileIndex αρχικοποιούνται με τα στοιχεία του τετραγώνου στο οποίο τοποθετήθηκαν. Τέλος τα τετράγωνα τα οποία επιλέχθηκαν ως αρχικές θέσεις για τα πιόνια του παίκτη αντιγράφονται στην λίστα allyTiles.

6.6 Κίνηση πιονιών και Γεωγραφία ταμπλό

Η παρακολούθηση των θέσεων των στοιχείων του παιχνιδιού αλλά και της κίνησης των πιονιών γίνεται μέσω των μεθόδων του script Movement.cs.Το script Movement.cs γίνεται ορατό στην σκηνή του παιχνιδιού μέσα από το ομώνυμο GameObject Movement στο οποίο έχει περαστεί ως component.Ο πλήρης κώδικας του Movement.cs βρίσκεται στο παράρτημα C.

6.6.1 Μεταβλητές

Το Movement.cs από πλευράς μεταβλητών έχει λάβει την λίστα tiles με τα τετράγωνα του ταμπλό και ένα αντικείμενο BoardInitializer για την πρόσβαση στις μεθόδους και τις μεταβλητές της κλάσης BoardInitializer (το script BoardInitilizer.cs). Επιπρόσθετα στις μεταβλητές του Movement.cs έχει οριστεί η κατηγορία Position Holders η οποία περιλαμβάνει τις παρακάτω λίστες τετραγώνων.

enemyTiles: Τα τετράγωνα τα οποία περιέχουν τα πιόνια του αντιπάλου.

allyTiles: Τα τετράγωνα τα οποία περιέχουν τα πιόνια του παίκτη

specialTerrainTiles: Τα τετράγωνα ειδικού εδάφους.

6.6.2 Αρχικοποίηση Position Holders

Η μέθοδος GetInitialPositions() γεμίζει τις λίστες της κατηγορίας Position Holders αντιγράφοντας τις ομώνυμες λίστες του script BoardInitilizer.cs στο Movement.cs.

6.6.3 Εύρεση γειτονικών τετραγώνων

Η μέθοδος findNeighbors() λαμβάνει σαν παράμετρο τον δείκτη από ένα τετράγωνο της λίστας tiles και επιστρέφει μια λίστα (GameObject) με τα γειτονικά του τετράγωνα. Πέρα από την μέθοδο findNeighbors οι μέθοδοι getNorthTileIndex(), getSouthTileIndex(), getEastTileIndex() και getWestTileIndex() επιστρέφουν τον δείκτη του γειτονικού τετραγώνου το οποίο βρίσκεται σε μια συγκεκριμένη κατεύθυνση (επάνω, κάτω, δεξιά αριστερά αντίστοιχα) και χρησιμοποιούνται για την κίνηση του κέρσορα.

6.6.4 Ενημέρωση λιστών

Στην περίπτωση που ένα πιόνι καταστραφεί καλούνται οι μέθοδοι updateAllyDestroyed() για τα πιόνια του παίκτη και updateEnemyDestroyed() για τα πιόνια του υπολογιστή. Ο ρόλος αυτών των μεθόδων είναι να αφαιρέσουν τα τετράγωνα τα οποία καταλάμβαναν τα πιόνια που καταστράφηκαν από τις λίστες allyTiles και enemyTiles.

6.6.5 Κίνηση Πιονιών Υπολογιστή

Η αυτοματοποιημένη κίνηση το πιονιών του αντιπάλου-υπολογιστή ελέγχεται από 2 μεθόδους η μία μέθοδος βρίσκεται στο script GameManager.cs και θα αναλυθεί αργότερα ενώ δεύτερη μέθοδος ονομάζεται moveEvemyPawn() και βρίσκεται το Movement.cs. Η μέθοδος moveEnemyPawn() λαμβάνει ως παράμετρο το πιόνι του αντιπάλου το οποίο επιλέγει να κινήσει το σύστημα και καλεί την μέθοδο findNeighbors() για να βρει τα γειτονικά τετράγωνα της θέσης του πιονιού. Από τα γειτονικά τετράγωνα που επιστρέφει η findNeighbors() η moveEnemyPawn() ελέγχει ποια από αυτά δεν είναι ελεύθερα και τα αφαιρεί. Από τα υπολειπόμενα τετράγωνα (εάν υπάρχουν) επιλέγεται τυχαία ένα. Τέλος η moveEnemyPawn() βρίσκει τον δείκτη του επιλεγμένου τετραγώνου στην λίστα tiles και την επιστρέφει.

6.6.6 Επίθεση Πιονιών Υπολογιστή

Αντίστοιχα με την κίνηση των πιονιών του υπολογιστή η αυτοματοποιημένη επίθεση πραγματοποιείται από 2 μεθόδους. Η μέθοδος η οποία βρίσκεται στο Movement.cs ονομάζεται enemyAttack() και ο ρόλος της είναι να επιλέγει τα πιόνια στα οποία θα επιτεθεί ο υπολογιστής. Η enemyAttack() λαμβάνει σαν παράμετρο το τετράγωνο του επιτιθέμενου πιονιού βρίσκει τα γειτονικά του τετράγωνα και στην συνέχεια κατασκευάζει μια λίστα πιθανών στόχων από τα γειτονικά τετράγωνα τα οποία καταλαμβάνουν πιόνια του παίκτη (αν υπάρχουν). Εφόσον η λίστα πιθανών στόχων δεν είναι κενή επιλέγεται τυχαία ένα τετράγωνο από αυτή και επιστρέφεται ο δείκτης της λίστας tiles όπως και στην moveEnemyPawn().

6.6.7 Ορθότητα κίνησης

Η μέθοδος findAvailablePositions() είναι μία Boolean μέθοδος η οποία ελέγχει την εγκυρότητα των κινήσεων. Η findAvailablePositions δέχεται σαν παραμέτρους τον δείκτη του τετραγώνου που καταλαμβάνει ένα πιόνι και το τετράγωνο στο οποίο σκοπεύει να μετακινηθεί έπειτα κατασκευάζει την λίστα των τετραγώνων στα οποία μπορεί να μετακινηθεί το πιόνι. Εάν το τετράγωνο στο οποίο σκοπεύει να μετακινηθεί το πιόνι ανήκει στην λίστα των έγκυρων θέσεων η μέθοδος επιστρέφει true ειδάλλως επιστρέφει false και η κίνηση δεν εκτελείται.

6.6.8 Ορθότητα εκτέλεσης ικανότητας

Η μέθοδος isValidAttack() είναι μια Boolean μέθοδος η οποία ελέγχει την εγκυρότητα της εκτέλεσης ικανοτήτων με περιορισμό εμβέλειας σε γειτονικά τετράγωνα.

6.6.9 Εκτοπισμός πιονιών

Η ικανότητα Air Strike έχει ως αποτέλεσμα τον εκτοπισμό του πιονιού το οποίο στόχευσε. Κατά την εκτέλεση της ικανότητας Air Strike καλείται η μέθοδος calculateDisplacement() η οποία λαμβάνει ως παραμέτρους τις θέσεις του Air Orb του παίκτη και του πιονιού το οποίο στοχεύει με την ικανότητα Air Strike και υπολογίζει το τετράγωνο στο οποίο θα γίνει η μετατόπιση του στόχου. Το τετράγωνο της

μετατόπισης είναι το γειτονικό τετράγωνο του στόχου το οποίο βρίσκεται στην προέκταση της ευθείας που ορίζουν τα δύο πιόνια προς την μεριά του πιονιού του υπολογιστή. Προφανώς τα πιόνια τα οποία βρίσκονται στις ακριανές θέσεις του ταμπλό δεν μπορούν να μετακινηθούν εκτός αυτού επομένως για αυτές τις περιπτώσεις η μετατόπιση δεν εκτελείται. Ακολουθώντας την εύρεση του τετραγώνου μετατόπισης καλείται η Boolean μέθοδος isDisplacementPossible() η οποία ελέγχει εάν το τετράγωνο μετατόπισης είναι ελεύθερο η καταλαμβάνεται από άλλο πιόνι η ειδικό έδαφος. Στην πρώτη περίπτωση επιστρέφει true και η μετατόπιση εκτελείται ενώ στην δεύτερη επιστρέφει false και δεν εκτελείται.

6.6.10 Έλεγχος ικανότητας Fly

Για την εκτέλεση της ικανότητας fly είναι απαραίτητο το τετράγωνο στο οποίο σκοπεύει να μετακινηθεί το Air Orb να είναι ελεύθερο. Η μέθοδος Air Orb παίρνει σαν παραμέτρους τον δείκτη του τετράγωνου στο οποίο βρίσκεται το Air Orb και το τετράγωνο στο οποίο σκοπεύει να μετακινηθεί και ελέγχει εάν το δεύτερο είναι ελεύθερο. Εάν το τετράγωνο στόχος είναι ελεύθερο η μέθοδος επιστρέφει true, η ικανότητα Fly εκτελείται και ενημερώνεται η λίστα allyTiles. Σε αντίθετη περίπτωση επιστρέφει false και ικανότητα Fly δεν εκτελείται.

6.6.11 Ενδυνάμωση πιονιών

Οι Boolean μέθοδοι isFireBuffed(), isWaterBuffed(), isPlantBuffed(), isRockBuffed(), και isAirBuffed() ελέγχουν εάν ένα πιόνι βρίσκεται στην σφαίρα επιρροής του αντίστοιχου ειδικού εδάφους του . Όταν ένα πιόνι εκτελεί μία ζημιογόνα ικανότητα καλείται η αντίστοιχη μέθοδος. Εάν η τιμή η οποία επιστρέφει η μέθοδος είναι true η ικανότητα λαμβάνει τους επιπλέον πόντους ζημιάς. Για τον έλεγχο της ενδυνάμωσης των πιονιών οι παραπάνω μέθοδοι λαμβάνουν ως παράμετρο το τετράγωνο του πιονιού για το οποίο γίνεται ο έλεγχος και η μέθοδος κοιτά εάν αυτό το τετράγωνο ανήκει στα γειτονικά τετράγωνα του αντίστοιχου ειδικού εδάφους.

6.7 Διαχειριστής διεπαφής χρήστη

Το script UiManager.cs είναι ορατό στην σκηνή του παιχνιδιού μέσω του GameObject UiManager στο οποίο έχει οριστεί ως component. Το script UiManager.cs είναι υπεύθυνο για την συμπεριφορά των UI στοιχείων της BoardScene. Ο πλήρης κώδικας του UiManager.cs βρίσκεται στο παράρτημα D.

6.7.1 Μεταβλητές

Οι GameObject μεταβλητές boardInititlizer, gameManager, movement εξασφαλίζουν την πρόσβαση στα στοιχεία των αντίστοιχων script.

Η κατηγορία Buttons περιέχει τις GameObject μεταβλητές playButton, readyButton, okButton, confirmMoveButton και endTurnButton στις οποίες έχουν περαστεί τα αντίστοιχα κουμπιά της διεπαφής.

Η κατηγορία selectors περιέχει τις GameObject μεταβλητές pawnSpawner και pawnSelector οι οποίες περιέχουν τις ομάδες κουμπιών για την επιλογή πιονιών.

Η λίστα spellBars περιέχει τις ομάδες ικανοτήτων του κάθε πιονιού.

6.7.2 Εξαφάνιση κουμπιού Play

Η μέθοδος PlayButtonPressed() καλείται με το πάτημα του PlayBtn και το εξαφανίζει από τον καμβά.

6.7.3 Εναλλαγή spellbar

Οι μέθοδοι KingPawnSelected(), FirePawnSelected(), PlantPawnSelected(), AirPawnSelected(), EarthPawnSelected(), και WaterPawnSelected() καλούνται όταν ο χρήστης επιλέξει το αντίστοιχο πιόνι από τα κουμπιά του PawnSelector. Κάθε φορά που καλείται μια από τις παραπάνω μεθόδους για ένα πιόνι εμφανίζεται το αντίστοιχο spellbar με τα κουμπιά των ικανοτήτων του πιονιού ενώ όλα τα υπόλοιπα αντικείμενα spellbar εξαφανίζονται.

6.7.4 Μέθοδος Start

Η μέθοδος Start καλείται απευθείας με την φόρτωση του UiManager.cs . Κατά το κάλεσμα της μεθόδου Start τα αντικείμενα pawnSelector, confirmMoveBtn, endTurnBtn και όλα τα αντικείμενα spellbar εξαφανίζονται.

6.7.5 Μέθοδος setUpReady

Όταν ο παίκτης έχει τελειώσει με το στήσιμο των πιονιών του πατάει το κουμπί ReadyBtn το οποίο με την σειρά του καλεί την μέθοδο setUpReady(). Η μέθοδος setUpReady ουσιαστικά σηματοδοτεί την μετάβαση από την φάση του στησίματος στην υπόλοιπη παρτίδα. Αυτό γίνεται εξαφανίζοντας τα στοιχεία της διεπαφής, pawnSpawner, OkBtn και ReadyBtn τα οποία ελέγχουν το στήσιμο των πιονιών. Έπειτα στην θέση των στοιχείων που εξαφανίστηκαν εμφανίζονται τα στοιχεία PawnSelector, ConfirmMoveBtn και EndTurn τα οποία ελέγχουν την κίνηση των πιονιών, τις ικανότητες τους και την εναλλαγή των γύρων. Τέλος καλούνται οι μέθοδοι GetInitialPositions() του Movement.cs και generatePawnLists() του GameManager.cs.

6.8 Skill Scripts

Στην BoardScene έχει προστεθεί ένα GameObject για την κάθε ικανότητα των πιονιών (και των 2 παικτών) π.χ. FireBall, EnemySeedStrike, Combustion κτλ. Σε αυτά τα αντικείμενα προσκολλήθηκαν τα script των ικανοτήτων ως components για να είναι ορατά από τα υπόλοιπα script και αντικείμενα της BoardScene. Λόγω της ομοιότητας αυτών των script θα γίνει αναφορά μόνο στην δομή τους.Η κατά κανόνα δομή των script των επιθέσεων περιέχει την μεταβλητή int damage η οποία δείχνει τους πόντους ζημιάς τους οποίους προκαλεί η κάθε ικανότητα χωρίς τα μπόνους ειδικού εδάφους και αλληλεπίδρασης. Το script Combustion.cs περιέχει την μεταβλητή selfInflictedDamage η οποία ορίζει την ζημία την οποία δέχεται το Fire Orb κάθε φορά που εκτελεί την ικανότητα Combustion. Το script Heal.cs αντί για την μεταβλητή damage έχει την μεταβλητή healing η οποία αντίστοιχα δείχνει τους πόντους ζωής που αναπληρώνει η ικανότητα Heal.

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

Unity Script | 6 references
public class Combustion : MonoBehaviour
{
    public int damage = 1;
    public int targets = 1;
    public int selfInflictedDamage = 2;
    public GameObject target;
}
```

Σχήμα 6.7: Παράδειγμα κώδικα ικανότητας- Combustion.cs

6.9 Cursor Script

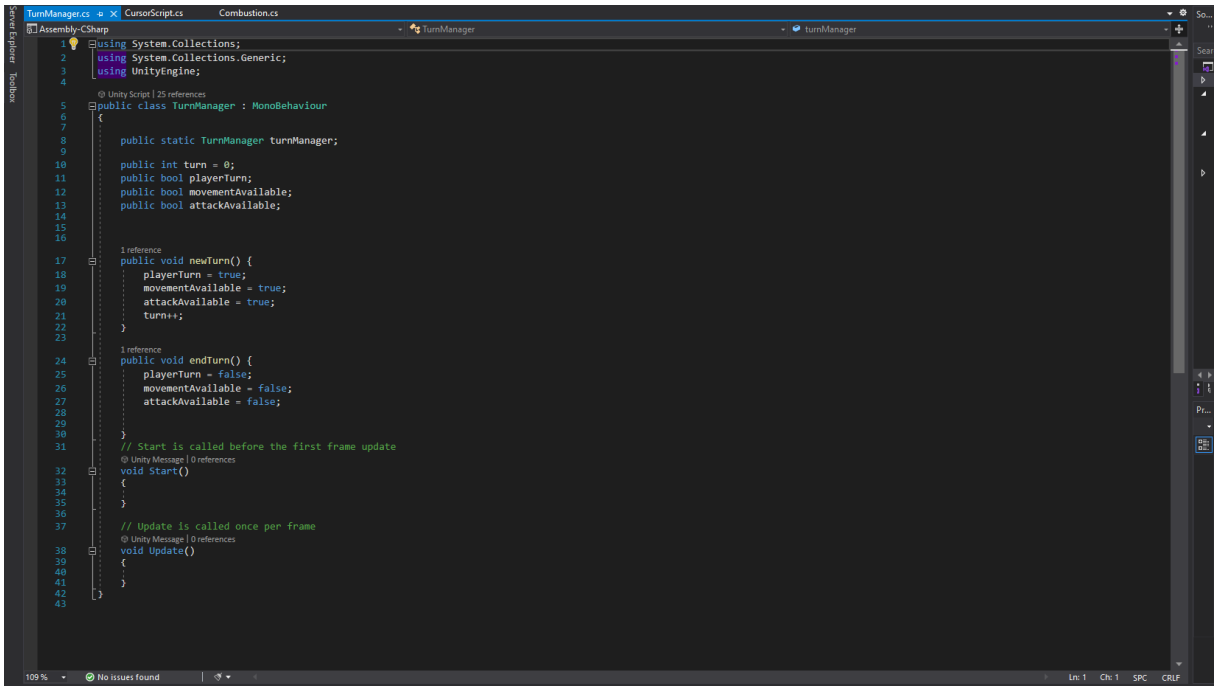
Το CursorScript.cs έχει προσκολληθεί στο αντικείμενο Cursor ως component και περιέχει τις μεταβλητές string currentTileName και int currentTileIndex οι οποίες περιέχουν το όνομα και τον δείκτη του τετραγώνου στο οποίο βρίσκεται ο κέρσορας. Η μέθοδος changeCurrentTile() καλείται κάθε φορά που ο κέρσορας αλλάζει θέση, λαμβάνει σαν παραμέτρους τον δείκτη και το όνομα του νέου τετραγώνου και ενημερώνει τις μεταβλητές currentTileName και currentTileIndex.

```
public class CursorScript : MonoBehaviour
{
    public string currentTileName;
    public int currentTileIndex;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    0 references
    public void DespawnCursor() {
        Destroy(this);

    }
    4 references
    public void changeCurrentTile(int tileValue,string tileName){

        this.currentTileIndex = tileValue;
        this.currentTileName = tileName;

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }
}
```

Σχήμα 6.8:Κώδικας CursorScript.cs

6.10 Διαχείριση γύρων

Το script TurnManager.cs σε συνδυασμό με το GameManager.cs είναι υπεύθυνο για την διαχείριση των γύρων των 2 παικτών.



Σχήμα 6.9: Κώδικας TurnManager.cs

6.10.1 Μεταβλητές TurnManager.cs

int turn: Ο αριθμός των γύρων που έχουν περάσει.

bool playerTurn: Ελέγχει αν εξελίσσεται ο γύρος του παίκτη (true) ή του υπολογιστή (false).

bool movementAvailable: Ελέγχει εάν ο παίκτης έχει διαθέσιμη κίνηση να εκτελέσει.

bool attackAvailable: Ελέγχει εάν ο παίκτης έχει διαθέσιμη ικανότητα πιονιού να εκτελέσει.

6.10.2 Μέθοδοι TurnManager.cs

Η μέθοδος newTurn() καλείται για να έρθει η σειρά του παίκτη, θέτει την τιμή των μεταβλητών playerTurn, movementAvailable, attackAvailable σε true και αυξάνει την τιμή της μεταβλητής turn κατά 1 .

Η μέθοδος endTurn() καλείται για να λήξει ο γύρος του παίκτη και θέτει την τιμή των μεταβλητών playerTurn, movementAvailable, attackAvailable σε false.

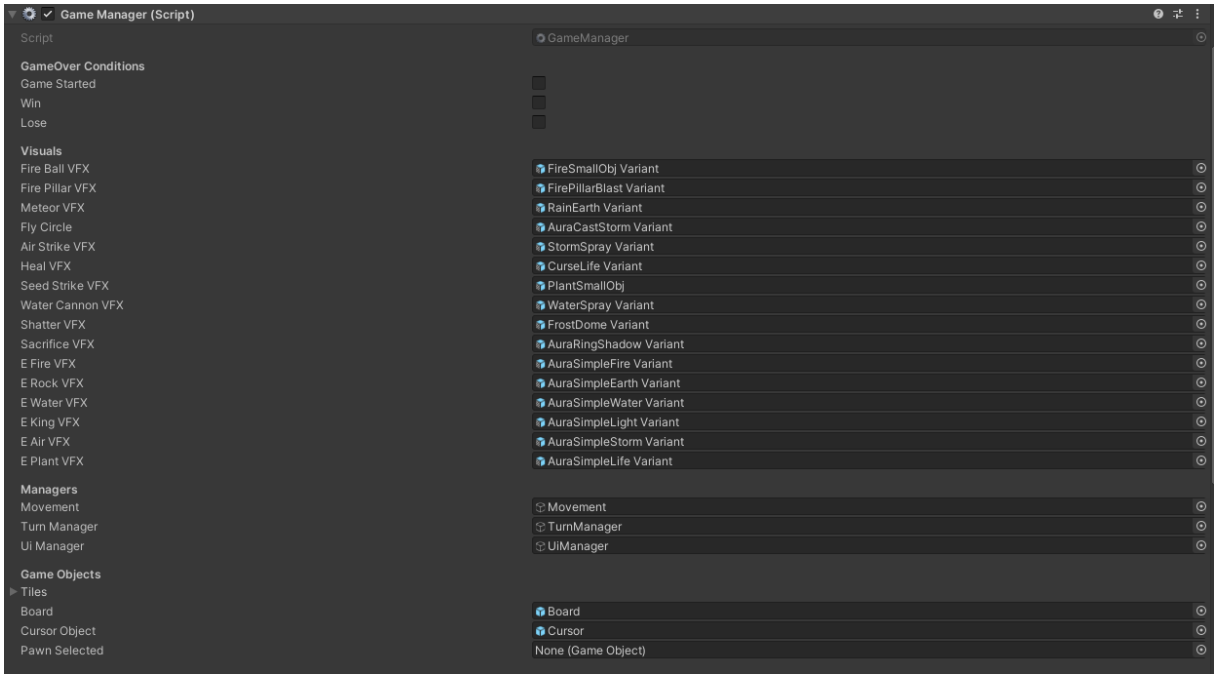
6.11 Game Manager

Το script GameManager.cs είναι ο πυρήνας της εφαρμογής Orbs και το μεγαλύτερο από πλευράς όγκου κώδικα.

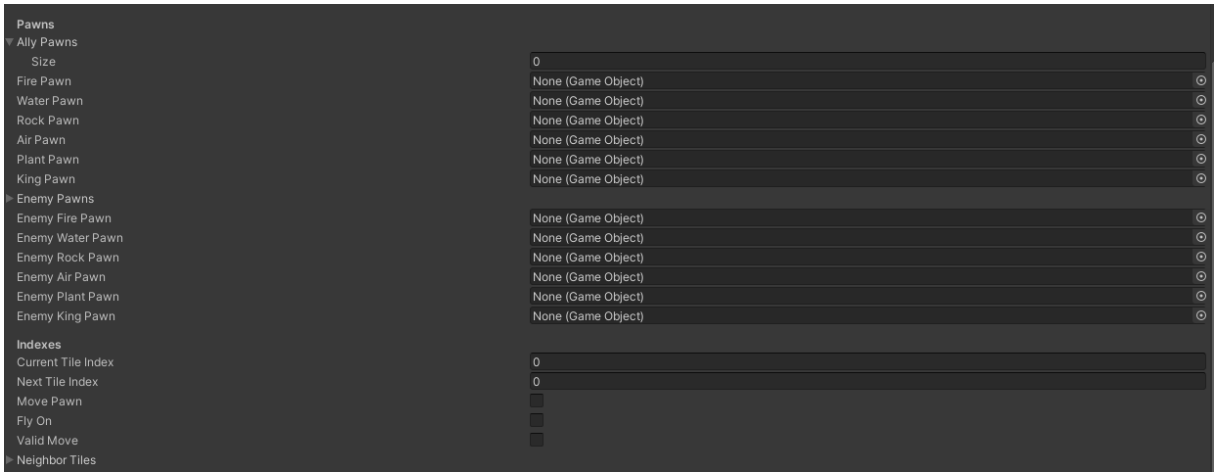
6.11.1 Μεταβλητές GameManager.cs

Οι μεταβλητές του GameManager.cs χωρίζονται στις παρακάτω κατηγορίες.

- GameOver Conditions
- Visuals
- Managers
- Game Objects
- Attacks
- EnemyAttacks
- Pawns
- Indexes
- Movement



Σχήμα 6.10: Μεταβλητές GameManager.cs στο Unity 1



Σχήμα 6.11: Μεταβλητές GameManager.cs στο Unity 2

Η κατηγορία GameOver Conditions περιέχει τις Boolean μεταβλητές gameStarted, win και lose.Η πρώτη μεταβλητή υποδεικνύει εάν το παιχνίδι έχει ξεκινήσει ενώ οι μεταβλητές win και lose παίρνουν τιμή true όταν ο παίκτης κερδίζει ή χάνει την παρτίδα αντίστοιχα.

```
Unity Script | 3 references
public class GameManager : MonoBehaviour
{


    public static GameManager instance;
    [Header("GameOver Conditions")]
    public bool gameStarted = false;
    public bool win;
    public bool lose;
```

Σχήμα 6.12 : Μεταβλητές GameManager.cs σε κώδικα 1

Η κατηγορία Visuals περιέχει τα prefab αντικείμενα του πακέτου Magic Arsenal για τα οπτικά εφέ των ικανοτήτων των πιονιών.

```
18
19      [Header("Visuals")]
20      public GameObject fireBallVFX;
21      public GameObject firePillarVFX;
22      public GameObject meteorVFX;
23      public GameObject flyCircle;
24      public GameObject airStrikeVFX;
25      public GameObject healVFX;
26      public GameObject seedStrikeVFX;
27      public GameObject waterCannonVFX;
28      public GameObject shatterVFX;
29      public GameObject sacrificeVFX;
30      public GameObject eFireVFX;
31      public GameObject eRockVFX;
32      public GameObject eWaterVFX;
33      public GameObject eKingVFX;
34      public GameObject eAirVFX;
35      public GameObject ePlantVFX;
36
```

Σχήμα 6.13: Μεταβλητές GameManager.cs σε κώδικα 2

Η κατηγορία Managers αποτελείται από τις GameObject μεταβλητές Movement, turnManager, uiManager οι οποίες παρέχουν πρόσβαση στις αντίστοιχες κλάσεις-script (Movement.cs, TurnManager.cs, UiManager.cs).

```
40
41      [Header("Managers")]
42      public GameObject Movement;
43      public GameObject turnManager;
44      public GameObject uiManager;
45
46
47
```

Σχήμα 6.14: Μεταβλητές GameManager.cs σε κώδικα 3

Η κατηγορία Game Objects περιλαμβάνει τα αντικείμενα τα οποία σχετίζονται με το ταμπλό. Πιο συγκεκριμένα περιέχει την λίστα tiles με τα τετράγωνα του ταμπλό, την μεταβλητή board για το ταμπλό, την μεταβλητή cursorObject για τον κέρσορα, την μεταβλητή nextTile για την μετακίνηση του κέρσορα, την μεταβλητή pawnSelected για τα πιόνια που επιλέγονται και την μεταβλητή targetPawn για τα πιόνια τα οποία δέχονται επίθεση.

```
47
48      [Header("Game Objects")]
49      public List<GameObject> tiles;
50      public GameObject board;
51      public GameObject cursorObject;
52      GameObject nextTile;
53      public GameObject pawnSelected;
54      GameObject targetPawn;
55
```

Σχήμα 6.15: Μεταβλητές GameManager.cs σε κώδικα 4

Οι κατηγορίες Attacks και Enemy Attacks αποτελούνται από τις μεταβλητές που περιέχουν τα GameObject των ικανοτήτων των πιονιών.

```
57      [Header("Attacks")]
58      public FireBall fireball;
59      public Combustion combustion;
60      public WaterCannon waterCannon;
61      public Shatter shatter;
62      public AirStrike airStrike;
63      public Fly fly;
64      public Meteor meteor;
65      public Smash smash;
66      public Heal heal;
67      public SeedStrike seedStrike;
68      public Sacrifice sacrifice;
69
70
71      [Header("EnemyAttacks")]
72      public EnemyFireBall eFireBall;
73      public EnemyAirBullet eAirBullet;
74      public EnemyMeteor eMeteor;
75      public EnemySeedStrike eSeedStrike;
76      public EnemyWaterCannon eWaterCannon;
77      public EnemyKingAttack eKingAttack;
78
79
```

Σχήμα 6.16: Μεταβλητές GameManager.cs σε κώδικα 5

Η κατηγορία Pawns αποτελείται από τις μεταβλητές οι οποίες περιέχουν τα ξεχωριστά πιόνια των παικτών και τις λίστες allyPawns και enemyPawns οι οποίες περιέχουν την συλλογή πιονιών του κάθε παίκτη.

```
80
81      [Header("Pawns")]
82      public List<GameObject> allyPawns;
83      public GameObject firePawn;
84      public GameObject waterPawn;
85      public GameObject rockPawn;
86      public GameObject airPawn;
87      public GameObject plantPawn;
88      public GameObject kingPawn;
89      //---------------------------
90      public List<GameObject> enemyPawns;
91      public GameObject enemyFirePawn;
92      public GameObject enemyWaterPawn;
93      public GameObject enemyRockPawn;
94      public GameObject enemyAirPawn;
95      public GameObject enemyPlantPawn;
96      public GameObject enemyKingPawn;
97
98
99
```

Σχήμα 6.17: Μεταβλητές GameManager.cs σε κώδικα 6

Η κατηγορία indexes αποτελείται από τις Integer μεταβλητές currentTileIndex και nextTileIndex οι οποίες δέχονται δείκτες τετραγώνων της λίστας tiles.

Η κατηγορία Movement περιέχει τις παρακάτω μεταβλητές:

lertptime-currentLerpTime: Οι οποίες λειτουργούν σαν χρονόμετρα κατά την μετακίνηση των πιονιών.

movePawn: Δείχνει εάν ένα πιόνι βρίσκεται σε κίνηση ή όχι.

flyOn: Δείχνει εάν το Air Orb βρίσκεται στον αέρα (κατά την χρήση της ικανότητας Fly)

validMove: Ελέγχει την εγκυρότητα μίας κίνησης πιονιού.

NextPosition-posA-posB: Σηματοδοτούν θέσεις στον χώρο.

nextPawnTile: Περιέχει το τετράγωνο στο οποίο σκοπεύει να μετακινηθεί ένα πιόνι.

neighborTiles: Λίστα με τα γειτονικά τετράγωνα ενός αντικειμένου.

```
[Header("Movement")]

Vector3 nextPosition = new Vector3(0, 0, 0); //Cursor next Position
private float lerptime = 2;
private float currentLerpTime = 0;
public bool movePawn = false;

public bool flyOn = false;
public bool validMove = false;
Vector3 posA;
Vector3 posB;
private GameObject nextPawnTile;
public List<GameObject> neighborTiles;
```

Σχήμα 6.18 : Μεταβλητές GameManager.cs σε κώδικα 7

6.11.2 Μέθοδοι GameManager.cs

Στις παρακάτω ενότητες αναλύονται οι μέθοδοι του script GameManager.cs.

6.11.2.1 generatePawnLists

Η μέθοδος generatePawnLists είναι μία void μέθοδος η οποία αρχικοποιεί τις μεταβλητές της κατηγορίας Pawns μετά την φάση του στησίματος και γεμίζει τις λίστες allyPawns και enemyPawns.Τέλος θέτει την μεταβλητή gameStarted σε true .

```
1 reference
public void generatePawnLists()
{
    firePawn = GameObject.FindGameObjectWithTag("FireOrb");
    allyPawns.Add(firePawn);
    waterPawn = GameObject.FindGameObjectWithTag("WaterOrb");
    allyPawns.Add(waterPawn);
    rockPawn = GameObject.FindGameObjectWithTag("RockOrb");
    allyPawns.Add(rockPawn);
    airPawn = GameObject.FindGameObjectWithTag("AirOrb");
    allyPawns.Add(airPawn);
    plantPawn = GameObject.FindGameObjectWithTag("PlantOrb");
    allyPawns.Add(plantPawn);
    kingPawn = GameObject.FindGameObjectWithTag("KingOrb");
    allyPawns.Add(kingPawn);
    //-------------------
    enemyFirePawn = GameObject.FindGameObjectWithTag("EnemyFireOrb");
    enemyPawns.Add(enemyFirePawn);
    enemyWaterPawn = GameObject.FindGameObjectWithTag("EnemyWaterOrb");
    enemyPawns.Add(enemyWaterPawn);
    enemyRockPawn = GameObject.FindGameObjectWithTag("EnemyRockOrb");
    enemyPawns.Add(enemyRockPawn);
    enemyAirPawn = GameObject.FindGameObjectWithTag("EnemyAirOrb");
    enemyPawns.Add(enemyAirPawn);
    enemyPlantPawn = GameObject.FindGameObjectWithTag("EnemyPlantOrb");
    enemyPawns.Add(enemyPlantPawn);
    enemyKingPawn = GameObject.FindGameObjectWithTag("EnemyKingOrb");
    enemyPawns.Add(enemyKingPawn);
    gameStarted = true;
}
```

Σχήμα 6.19 Κώδικας μεθόδων GameManager.cs 1

6.11.2.2 Μέθοδοι Ικανοτήτων 1

Ένα υποσύνολο των μεθόδων για την εκτέλεση των ικανοτήτων των πιονιών ακολουθεί την δομή που θα παρουσιαστεί σε αυτή την ενότητα. Η ομαδοποίηση αυτή γίνεται καθώς δεν υπάρχουν ουσιαστικές διαφορές ανάμεσα σε αυτές τις μεθόδους πέρα την χρήση οπτικών εφέ και τις αριθμητικές αξίες ορισμένων μεταβλητών. Η ομάδα αυτής της ενότητας περιλαμβάνει τις μεθόδους castFireBall(), castMeteor(), castSeedStrike(), castAirStrike() και castWaterCannon(). Η διαδικασία εκτέλεσης των μεθόδων αυτής της ομάδας περιλαμβάνει τα παρακάτω βήματα.

1. Εύρεση του τετραγώνου του κέρσορα.
2. Έλεγχος για την εγκυρότητα της εκτέλεσης ικανότητας μέσω της μεθόδου isValidAttack().
3. Εντοπισμός του πιονιού στο οποίο στοχεύει η ικανότητα .
4. Εμφάνιση του αντίστοιχου οπτικού εφέ της ικανότητας.
5. Έλεγχος και εφαρμογή των κανόνων αλληλεπίδρασης εάν υπάρχουν.
6. Αφαίρεση πόντων ζημιάς από τους πόντους ζωής του πιονιού που δέχτηκε την επίθεση
7. Σε περίπτωση καταστροφής πιονιού γίνεται αφαίρεση καλείται η μέθοδος updateTileEnemyDestroyed().

Τα σχήματα (6.20,6.21) δείχνουν τις μεθόδους castWaterCannon () και castFireBall().

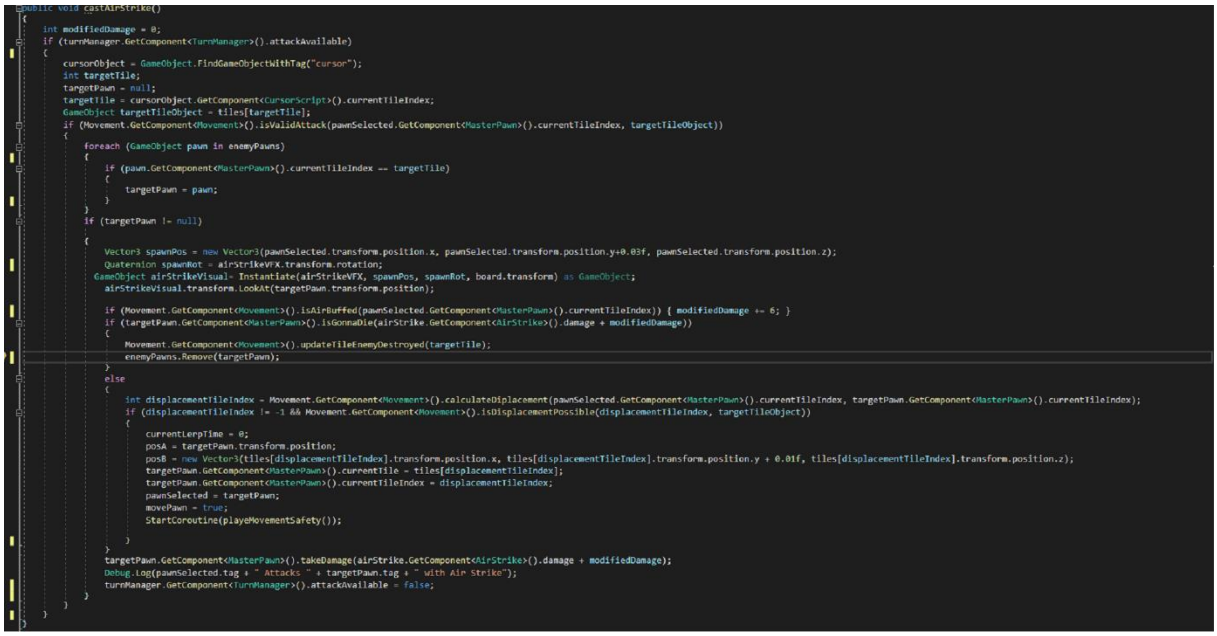
```
public void castWaterCannon()
{
    int modifiedDamage = 0;
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        if (Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetTileObject))
        {
            foreach (GameObject pawn in enemyPawns)
            {

                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                {
                    targetPawn = pawn;
                }
            }
            if (targetPawn != null)
            {
                Vector3 spawnPos = new Vector3(pawnSelected.transform.position.x, pawnSelected.transform.position.y + 0.03f, pawnSelected.transform.position.z);
                Quaternion spawnRot =waterCannonVFX.transform.rotation;

                GameObject waterCannonVisual = Instantiate(waterCannonVFX, spawnPos, spawnRot, board.transform) as GameObject;
                waterCannonVisual.transform.LookAt(targetPawn.transform.position);
                if (targetPawn.tag.Contains("Plant")) { modifiedDamage -= 2; }
                if (targetPawn.tag.Contains("Fire")) { modifiedDamage += 2; }
                if (Movement.GetComponent<Movement>().isWaterBuffed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex)) { modifiedDamage += 6; }
                if (targetPawn.GetComponent<MasterPawn>().status.Equals("Frozen")) { modifiedDamage += 5; }

                if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(waterCannon.GetComponent<WaterCannon>().damage + modifiedDamage))
                {
                    Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
                    enemyPawns.Remove(targetPawn);

                }
                targetPawn.GetComponent<MasterPawn>().takeDamage(waterCannon.GetComponent<WaterCannon>().damage + modifiedDamage);
                Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with water Cannon ");
                turnManager.GetComponent<TurnManager>().attackAvailable = false;
            }
        }
    }
```

Σχήμα 6.20 : Κώδικας μεθόδων GameManager.cs 2

```
0 references
public void castFireBall()
{
    int modifiedDamage = 0;
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];

        if (Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetTileObject))
        {
            foreach (GameObject pawn in enemyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                {
                    targetPawn = pawn;
                }
            }
            if (targetPawn != null)
            {
                Vector3 spawnPos = new Vector3(targetPawn.transform.position.x, targetPawn.transform.position.y + 0.1f, targetPawn.transform.position.z);
                Quaternion spawnRot = fireBallVFX.transform.rotation;
                GameObject projectile = Instantiate(fireBallVFX, spawnPos, spawnRot, board.transform) as GameObject;
                projectile.transform.LookAt(targetTileObject.transform.position);
                projectile.GetComponent<Rigidbody>().AddForce(projectile.transform.forward * 10f);
                if (targetPawn.tag.Contains("Rock")) { modifiedDamage -= 2; }
                if (targetPawn.tag.Contains("Plant")) { modifiedDamage += 2; }
                if (targetPawn.tag.Contains("Water")) { modifiedDamage -= 1; }
                if (Movement.GetComponent<Movement>().isFireBuffed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex)) { modifiedDamage += 6; }
                if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(fireball.GetComponent<FireBall>().damage + modifiedDamage))
                {
                    Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
                    enemyPawns.Remove(targetPawn);
                }
                targetPawn.GetComponent<MasterPawn>().takeDamage(fireball.GetComponent<FireBall>().damage + modifiedDamage);
                Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with fireball ");
                turnManager.GetComponent<TurnManager>().attackAvailable = false;
            }
        }
    }
}
```

Σχήμα 6.21 : Κώδικας μεθόδων GameManager.cs 3

Η μέθοδος castAirStrike() έχει ένα επιπλέον βήμα (γραμμές 421-435 σχήμα 6.22) στην εκτέλεση της για τον υπολογισμό της μετατόπισης του πιονιού επάνω στο οποίο εκτελείται και την ενημέρωση της θέσης του.



```
public void castAirStrike()
{
    int modifiedDamage = 0;
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        if (Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetTileObject))
        {
            foreach (GameObject pawn in enemyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                {
                    targetPawn = pawn;
                }
            }
            if (targetPawn != null)

            {
                Vector3 spawnPos = new Vector3(pawnSelected.transform.position.x, pawnSelected.transform.position.y+0.03f, pawnSelected.transform.position.z);
                Quaternion spawnRot = airStrikeVFX.transform.rotation;
               GameObject airStrikeVisual= Instantiate(airStrikeVFX, spawnPos, spawnRot, board.transform) as GameObject;
                airStrikeVisual.transform.LookAt(targetPawn.transform.position);

                if (Movement.GetComponent<Movement>().isAirBuffed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex)) { modifiedDamage += 6; }
                if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(airStrike.GetComponent<AirStrike>().damage + modifiedDamage))
                {
                    Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
                    enemyPawns.Remove(targetPawn);
                }
                else
                {
                    int displacementTileIndex = Movement.GetComponent<Movement>().calculateDisplacement(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetPawn.GetComponent<MasterPawn>().currentTileIndex);
                    if (displacementTileIndex != -1 && Movement.GetComponent<Movement>().isDisplacementPossible(displacementTileIndex, targetTileObject))
                    {
                        currentLerpTime = 0;
                        posA = targetPawn.transform.position;
                        posB = new Vector3(tiles[displacementTileIndex].transform.position.x, tiles[displacementTileIndex].transform.position.y + 0.01f, tiles[displacementTileIndex].transform.position.z);
                        targetPawn.GetComponent<MasterPawn>().currentTile = tiles[displacementTileIndex];
                        targetPawn.GetComponent<MasterPawn>().currentTileIndex = displacementTileIndex;
                        pawnSelected = targetPawn;
                        movePawn = true;
                        StartCoroutine(playMovementSafety());

                    }
                }
                targetPawn.GetComponent<MasterPawn>().takeDamage(airStrike.GetComponent<AirStrike>().damage + modifiedDamage);
                Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Air Strike");
                turnManager.GetComponent<TurnManager>().attackAvailable = false;
            }
        }
    }
}
```

Σχήμα 6.22: Κώδικας μεθόδων GameManager.cs 4

6.11.2.3 Μέθοδοι Ικανοτήτων 2

Αυτή η ενότητα περιλαμβάνει τις μεθόδους ικανοτήτων οι οποίες αποκλίνουν από την δομή της προηγούμενης ενότητας.

Η μέθοδος castCombustion() και castSacrifice() ακολουθούν τα παρακάτω βήματα.

1. Εύρεση του του τετραγώνου του κέρσορα.
2. Εντοπισμός του πιονιού στο οποίο στοχεύει η ικανότητα.
3. Εμφάνιση του αντίστοιχου οπτικού εφέ της ικανότητας.
4. Έλεγχος και εφαρμογή των κανόνων αλληλεπίδρασης εάν υπάρχουν.
5. Αφαίρεση πόντων ζημιάς από τους πόντους ζωής και των 2 πιονιών.
6. Σε περίπτωση καταστροφής πιονιού γίνεται αφαίρεση του από το ταμπλό και ενημερώνεται η κατάλληλη λίστα θέσεων.

```
public void castCombustion()
{
    int modifiedDamage = 0;
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        foreach (GameObject pawn in enemyPawns)
        {
            if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
            {
                targetPawn = pawn;

            }
        }
        if (targetPawn != null)
        {
            Vector3 spawnPos = new Vector3(targetPawn.transform.position.x, targetPawn.transform.position.y, targetPawn.transform.position.z);
            Quaternion spawnRot = firePillarVFX.transform.rotation;

            Instantiate(firePillarVFX, spawnPos, spawnRot, board.transform);
            if (targetPawn.tag.Contains("Rock")) { modifiedDamage -= 2; }
            if (targetPawn.tag.Contains("Plant")) { modifiedDamage += 2; }
            if (targetPawn.tag.Contains("Water")) { modifiedDamage -= 1; }
            if (Movement.GetComponent<Movement>().isFireBuffed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex))
            {
                modifiedDamage += 6;
            }
            if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(combustion.GetComponent<Combustion>().damage + modifiedDamage))
            {
                Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
                enemyPawns.Remove(targetPawn);
            }
            targetPawn.GetComponent<MasterPawn>().takeDamage(combustion.GetComponent<Combustion>().damage + modifiedDamage);
            if (pawnSelected.GetComponent<MasterPawn>().isGonnaDie(combustion.GetComponent<Combustion>().selfInflictedDamage))
            {
                Movement.GetComponent<Movement>().updateTileAllyDestroyed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex);
                allyPawns.Remove(pawnSelected);
            }
            pawnSelected.GetComponent<MasterPawn>().takeDamage(combustion.GetComponent<Combustion>().selfInflictedDamage);
            Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Combustion");
            Debug.Log(pawnSelected.tag + " inficts to self " + combustion.GetComponent<Combustion>().selfInflictedDamage + " Damage ");
            turnManager.GetComponent<TurnManager>().attackAvailable = false;
        }
    }
}
```

Σχήμα 6.23 : Κώδικας μεθόδων GameManager.cs 5

```
public void castSacrifice() {
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        foreach (GameObject pawn in enemyPawns)
        {
            if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
            {
                targetPawn = pawn;
            }
        }
        if(targetPawn != null)
        {
            Vector3 spawnPos1 = pawnSelected.transform.position;
            Quaternion spawnRot = sacrificeVFX.transform.rotation;
            Vector3 spawnPos2 = new Vector3(targetTileObject.transform.position.x, targetTileObject.transform.position.y + 0.005f, targetTileObject.transform.position.z);
            Instantiate(sacrificeVFX, spawnPos1, spawnRot, board.transform);
            Instantiate(sacrificeVFX, spawnPos2, spawnRot, board.transform);
            if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(sacrifice.GetComponent<Sacrifice>().damage)) {
                Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
                enemyPawns.Remove(targetPawn);
            }
            if (pawnSelected.GetComponent<MasterPawn>().isGonnaDie(sacrifice.GetComponent<Sacrifice>().damage))...
            targetPawn.GetComponent<MasterPawn>().takeDamage(sacrifice.GetComponent<Sacrifice>().damage);
           pawnSelected.GetComponent<MasterPawn>().takeDamage(sacrifice.GetComponent<Sacrifice>().damage);
            Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Sacrifice ");
            turnManager.GetComponent<TurnManager>().attackAvailable = false;
        }
    }
}
```

Σχήμα 6.24 : Κώδικας μεθόδων GameManager.cs 6

Η μέθοδος castFly() ακολουθεί τα εξής βήματα εκτέλεσης.

1. Εύρεση του του τετραγώνου του κέρσορα.
2. Έλεγχος εγκυρότητας μέσω της μεθόδου isFlyPossbile().
3. Εμφάνιση του αντίστοιχου οπτικού εφέ της ικανότητας.
4. Μετακίνηση Air Orb στο τετράγωνο του κέρσορα μέσω της FlyDelay().

```
public void castFly()
{
    if (turnManager.GetComponent<TurnManager>().movementAvailable)
    {
        currentLerpTime = 0;
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        if (Movement.GetComponent<Movement>().isFlyPossible(targetTile, pawnSelected.GetComponent<MasterPawn>().currentTile))
        {

            Vector3 spawnPos1 = pawnSelected.transform.position;
            Quaternion spawnRot =flyCircle.transform.rotation;
            Vector3 spawnPos2 = new Vector3(targetTileObject.transform.position.x, targetTileObject.transform.position.y + 0.005f, targetTileObject.transform.position.z);
            Instantiate(flyCircle, spawnPos1, spawnRot, board.transform);
            Instantiate(flyCircle, spawnPos2, spawnRot, board.transform);

            pawnSelected.GetComponent<MasterPawn>().currentTileIndex = targetTile;
            pawnSelected.GetComponent<MasterPawn>().currentTile = targetTileObject;
            posA = pawnSelected.transform.position;
            posB = new Vector3(pawnSelected.transform.position.x, pawnSelected.transform.position.y + 0.07f, pawnSelected.transform.position.z);
            movePawn = true;
            StartCoroutine(playeMovementSafety());
            StartCoroutine(flyDelay(targetTileObject));
            turnManager.GetComponent<TurnManager>().movementAvailable = false;

        }
    }
}
```

Σχήμα 6.25: Κώδικας μεθόδων GameManager.cs 7

Η μέθοδος castHeal() ακολουθεί τα παρακάτω βήματα.

1. Εύρεση του του τετραγώνου του κέρσορα.
2. Εμφάνιση οπτικού εφέ ικανότητας.
3. Αναπλήρωση πόντων ζωής στο πιόνι-στόχο.

```
public void castHeal()
{
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        foreach (GameObject pawn in allyPawns)
        {
            if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
            {
                targetPawn = pawn;
            }
        }
        Vector3 spawnPos = new Vector3(targetPawn.transform.position.x, targetPawn.transform.position.y + 0.09f, targetPawn.transform.position.z);
            Quaternion spawnRot = healVFX.transform.rotation;
            Instantiate(healVFX, spawnPos, spawnRot, board.transform);
        targetPawn.GetComponent<MasterPawn>().takeHealing(heal.GetComponent<Heal>().healing);
    }
    turnManager.GetComponent<TurnManager>().attackAvailable = false;
    Debug.Log(pawnSelected.tag + " Heals " + targetPawn.tag + " for " + heal.GetComponent<Heal>().healing);
}
```

Σχήμα 6.26: Κώδικας μεθόδων GameManager.cs 8

Η μέθοδος castShatter() ακολουθεί τα παρακάτω βήματα.

1. Εύρεση του τετραγώνου του κέρσορα.
2. Έλεγχος για την εγκυρότητα της εκτέλεσης ικανότητας μέσω της μεθόδου isValidAttack().
3. Εντοπισμός του πιονιού στο οποίο στοχεύει η ικανότητα εάν υπάρχει.
4. Όποιο πιόνι βρίσκεται ήδη σε παγωμένη κατάσταση (status=frozen) βγαίνει από αυτή .
5. Εμφάνιση οπτικού εφέ.
6. Το πιόνι στόχος εισέρχεται σε παγωμένη κατάσταση .

```
0 references
public void castShatter()
{
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        if (Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetTileObject))
        {
            foreach (GameObject pawn in enemyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile && !pawn.tag.Contains("Fire"))
                {
                    targetPawn = pawn;
                    foreach (GameObject otherPawn in enemyPawns)
                    {
                        if (pawn != otherPawn)
                        {
                            otherPawn.GetComponent<MasterPawn>().status = "";
                        }
                    }
                }
            }
            if (targetPawn != null)
            {
                Destroy(GameObject.FindGameObjectWithTag("Frozen"));
                Vector3 spawnPos = targetPawn.transform.position;
                spawnPos.y += 0.005f;
                Instantiate(shatterVFX, spawnPos, shatterVFX.transform.rotation, targetPawn.transform);

                targetPawn.GetComponent<MasterPawn>().status = "Frozen";
                Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Shatter the next attack more damage");
                turnManager.GetComponent<TurnManager>().attackAvailable = false;
            }
        }
    }
```

Σχήμα 6.27 : Κώδικας μεθόδων GameManager.cs 9

6.11.2.4 Κίνηση Κέρσορα

Για την κίνηση του κέρσορα χρησιμοποιούνται οι μέθοδοι πλοήγησης MoveNorth(), MoveSouth(), MoveEast() και MoveWest() οι οποίες καλούνται από τα αντίστοιχα βελάκια πλοήγησης της διεπαφής χρήστη. Οι προαναφερθείσες μέθοδοι με την σειρά τους χρησιμοποιούν τις μεθόδους getNorthTileIndex(), getSouthTileIndex(), getEastTileIndex() και getWestTileIndex() του script Movement.cs για τον εντοπισμό του επιθυμητού τετραγώνου. Η λειτουργία των μεθόδων πλοήγησης ακολουθεί τα εξής βήματα.

1. Εύρεση του κέρσορα.
2. Εύρεση του επόμενου τετραγώνου στο οποίο θα μετακινηθεί ο κέρσορας.
3. Ενημέρωση θέσεων κέρσορα.
4. Μετακίνηση κέρσορα.

```
//Pawn Movements Methods
0 references
public void MoveNorth()
{
    cursorObject = GameObject.FindGameObjectWithTag("cursor");
    currentTileIndex = cursorObject.GetComponent<CursorScript>().currentTileIndex;
    nextTileIndex = Movement.GetComponent<Movement>().getNorthTileIndex(currentTileIndex);
    nextTile = tiles[nextTileIndex];
    nextPosition = new Vector3(nextTile.transform.position.x, nextTile.transform.position.y + 0.001f, nextTile.transform.position.z);
    cursorObject.GetComponent<CursorScript>().changeCurrentTile(nextTileIndex, nextTile.name);
    cursorObject.transform.position = nextPosition;
}
0 references
public void MoveSouth()...
0 references
public void MoveEast()
{
    cursorObject = GameObject.FindGameObjectWithTag("cursor");
    currentTileIndex = cursorObject.GetComponent<CursorScript>().currentTileIndex;
    nextTileIndex = Movement.GetComponent<Movement>().getEastTileIndex(currentTileIndex);
    nextTile = tiles[nextTileIndex];
    nextPosition = new Vector3(nextTile.transform.position.x, nextTile.transform.position.y + 0.001f, nextTile.transform.position.z);
    cursorObject.GetComponent<CursorScript>().changeCurrentTile(nextTileIndex, nextTile.name);
    cursorObject.transform.position = nextPosition;
}
0 references
public void MoveWest()...
```

Σχήμα 6.28 : Κώδικας μεθόδων GameManager.cs 10

6.11.2.5 Επιλογή πιονιών παίκτη

Για την επιλογή των πιονιών του παίκτη χρησιμοποιούνται οι μέθοδοι επιλογής firePawnSelected(), waterPawnSelected(), plantPawnSelected(), airPawnSelected(), rockPawnSelected() και kingPawnSelected() οι οποίες καλούνται από τα αντίστοιχα κουμπιά του αντικειμένου spawner της διεπαφής. Η λειτουργία των μεθόδων επιλογής περιλαμβάνει τα εξής βήματα.

1. Επαναφορά της μεταβλητής movePawn στην τιμή false.
2. Εύρεση του επιλεγμένου πιονιού στην σκηνή.
3. Ανάθεση του επιλεγμένου πιονιού στην μεταβλητή pawnSelected.

```
728
729     //Pawn Selection Methods
        0 references
730     public void firePawnSelected()
731     {
732         movePawn = false;
733         pawnSelected = GameObject.FindGameObjectWithTag("FireOrb");
734     }
        0 references
735     public void waterPawnSelected()
736     {
737         movePawn = false;
738         pawnSelected = GameObject.FindGameObjectWithTag("WaterOrb");
739     }
        0 references
740     public void plantPawnSelected()
741     {
742         movePawn = false;
743         pawnSelected = GameObject.FindGameObjectWithTag("PlantOrb");
744     }
        0 references
745     public void airPawnSelected()...
        0 references
750     public void rockPawnSelected()...
        0 references
755     public void kingPawnSelected()...
760
761
```

Σχήμα 6.29 : Κώδικας μεθόδων GameManager.cs 11

6.11.2.6 Μετακίνηση πιονιών παίκτη

Η μετακίνηση των πιονιών του παίκτη εκτελείται με το κάλεσμα της μεθόδου moveSelectedPawn() μέσω του κουμπιού ConfirmMoveBtn της διεπαφής χρήστη. Η λειτουργία της moveSelectedPawn() περιλαμβάνει τα παρακάτω βήματα.

1. Εύρεση του τετραγώνου του κέρσορα.
2. Έλεγχος εγκυρότητας κίνησης.
3. Μετακίνηση του πιονιού το οποίο περιέχει η μεταβλητή pawnSelected.
4. Ενημέρωση των μεταβλητών currentTile και currentTileIndex του πιονιού.

```
public void moveSelectedPawn()
{
    if (turnManager.GetComponent<TurnManager>().movementAvailable)
    {
        nextPawnTile = tiles[cursorObject.GetComponent<CursorScript>().currentTileIndex];
        if (!pawnSelected.tag.Contains("Enemy"))
        {
            validMove = Movement.GetComponent<Movement>().findAvailablePositions(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, nextPawnTile);
            if (validMove)
            {

                currentLerpTime = 0;
                posA = pawnSelected.transform.position;
                posB = new Vector3(nextPawnTile.transform.position.x, nextPawnTile.transform.position.y + 0.01f, nextPawnTile.transform.position.z);
                pawnSelected.GetComponent<MasterPawn>().currentTile = nextPawnTile;
                pawnSelected.GetComponent<MasterPawn>().currentTileIndex = cursorObject.GetComponent<CursorScript>().currentTileIndex;
                movePawn = true;
                StartCoroutine(playeMovementSafety());
                turnManager.GetComponent<TurnManager>().movementAvailable = false;

            }
        }
    }


}
```

Σχήμα 6.30 : Κώδικας μεθόδων GameManager.cs 12

6.11.2.7 Μετακίνηση πιονιών Υπολογιστή

Την μετακίνηση των πιονιών του υπολογιστή αναλαμβάνει η μέθοδος moveEnemyPawn(). Η μέθοδος moveEnemyPawn του script GameManager.cs επιλέγει τυχαία το πιόνι του υπολογιστή προς μετακίνηση και καλεί την ομώνυμη μέθοδο του script Movement.cs (6.6.5) για την επιλογή του τετραγώνου . Τέλος εκτελείται η μετακίνηση του πιονιού και ενημερώνονται οι μεταβλητές currentTile και currentTileIndex του πιονιού.

```
public void moveEnemyPawn()
{
    System.Random rnd = new System.Random();
    string selectedTag = enemyPawns[rnd.Next(0, enemyPawns.Count)].tag;
    pawnSelected = GameObject.FindGameObjectWithTag(selectedTag);

    int nextPawnTileIndex = Movement.GetComponent<Movement>().moveEnemyPawn(pawnSelected);
    if (nextPawnTileIndex != -1)
    {
        nextPawnTile = tiles[nextPawnTileIndex];
        currentLerpTime = 0;
        posA = pawnSelected.transform.position;
        posB = new Vector3(nextPawnTile.transform.position.x, nextPawnTile.transform.position.y + 0.01f, nextPawnTile.transform.position.z);
        pawnSelected.GetComponent<MasterPawn>().currentTile = nextPawnTile;
        pawnSelected.GetComponent<MasterPawn>().currentTileIndex = nextPawnTileIndex;
        movePawn = true;
        StartCoroutine(executeAttack());
    }
}
```

Σχήμα 6.31 : Κώδικας μεθόδων GameManager.cs 13

6.11.2.8 Επίθεση πιονιών υπολογιστή

Η μέθοδος enemyAttack() αναλαμβάνει την αυτοματοποιημένη επίθεση των πιονιών του υπολογιστή. Πρώτα επιλέγεται τυχαία το πιόνι του υπολογιστή το οποίο θα εκτελέσει την επίθεση και στην συνέχεια

καλείται η μέθοδος enemyAttack() του Movement.cs για την επιλογή του τετραγώνου στο οποίο θα γίνει η επίθεση. Εάν το τετράγωνο το οποίο επιλέχθηκε περιέχει πιόνι του παίκτη η επίθεση προχωρά αλλιώς παραλείπεται και ο γύρος του υπολογιστή τελειώνει. Μετά την επιλογή του πιονιού το οποίο θα δεχτεί την επίθεση εμφανίζεται το οπτικό εφέ που αντιστοιχεί στο επιτιθέμενο πιόνι, αφαιρούνται οι πόντοι ζημίας της επίθεσης και ενημερώνονται οι λίστες θέσεων.

6.11.2.9 Μέθοδος Update

Η μέθοδος Update της κάθε κλάσης καλείται μία φορά σε κάθε frame . Στο GameManager.cs η μέθοδος update έχει τους παρακάτω ρόλους.

1. Εκτελεί την μετακίνηση(animation) ανά frame των πιονιών (γραμμές 1013-1026).
2. Ελέγχει τις συνθήκες τερματισμού της παρτίδας (γραμμές 1029-1041) .
3. Εκτελεί την μετάβαση στις σκηνές YouWin και YouLose ανάλογα με το αποτέλεσμα της παρτίδας.

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    if (movePawn)
    {
        currentLerpTime += Time.deltaTime;
        if (currentLerpTime >= lerptime)
        {
            currentLerpTime = lerptime;
        }
        float percentageTimes = currentLerpTime / lerptime;
        pawnSelected.transform.position = Vector3.Lerp(posA, posB, percentageTimes);

    }

    if (enemyPawns.Count == 0 && gameStarted)
    {
        win = true;
    }
    if (allyPawns.Count <= 3 && gameStarted)
    {
        lose = true;
    }
    if (kingPawn == null && gameStarted)
    {

        lose = true;
    }
    if (win)
    {
        StartCoroutine(loadNextSceneWin());

    }
    if (lose) {
        StartCoroutine(loadNextSceneLose());
    }
}
}
```

Σχήμα 6.32 : Κώδικας μεθόδων GameManager.cs 14

6.11.2.10 Διαχείριση γύρων

Η μέθοδος endPlayerTurn() καλείται με το πάτημα του κουμπιού EndTurn της διεπαφής χρήστη για να λήξει ο γύρος του παίκτη και να ξεκινήσει ο γύρος του υπολογιστή. Η μέθοδος endPlayerTurn εκτελεί τις παρακάτω λειτουργίες .

1. Εξαφανίζει τα αντικείμενα PawnSelector και spellbar.
2. Καλεί την μέθοδο endTurn του TurnManager.cs

3. Καθαρίζει όλα τα οπτικά εφέ που παραμένουν στο ταμπλό από της ικανότητες του αντιπάλου.
4. Καλεί την μέθοδο executeEnemyTurn() για να ξεκινήσει ο γύρος του υπολογιστή.

```
public void endPlayerTurn()
{
    uiManager.GetComponent<UiManager>().pawnSelector.SetActive(false);
    foreach (GameObject spellbar in uiManager.GetComponent<UiManager>().spellBars)
    {
        spellbar.SetActive(false);
    }
    turnManager.GetComponent<TurnManager>().endTurn();
    uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(false);
    GameObject[] garbageVFX = GameObject.FindGameObjectsWithTag("eAttackVFX");
    foreach (GameObject vfx in garbageVFX)
    {
        Destroy(vfx);
    }
    executeEnemyTurn();
}
```

Σχήμα 6.33 :Κώδικας μεθόδων GameManager.cs 15

Η μέθοδος executeEnemyTurn καλεί αρχικά την moveEnemyPawn() η οποία με την σειρά της καλεί ασύγχρονα την μέθοδο executeAttack. Ο ρόλος της μεθόδου executeAttack είναι να καλέσει την μέθοδο enemyAttack() με μία συγκεκριμένη χρονική καθυστέρηση . Η μέθοδος executeEnemyTurn καλεί και την μέθοδο newTurn() για να ξεκινήσει νέος γύρος μετά την εκτέλεση των κινήσεων του υπολογιστή. Η μέθοδος newTurn() καλεί ασύγχρονα την μέθοδο playerDelay() η οποία μετά από καθυστέρηση

1. Εμφανίζει το αντικείμενο PawnSelector και το κουμπί EndTurn.
2. Καλεί την μέθοδο newTurn() του script TurnManager.cs.

Τέλος η μέθοδος newTurn καθαρίζει τα οπτικά εφέ από τις ικανότητες των πιονιών του παίκτη.

```
public void executeEnemyTurn()
{
    moveEnemyPawn();
    newTurn();

}

1 reference
IEnumerator executeAttack() {
    yield return new WaitForSeconds(2.6f);
    movePawn = false;
    enemyAttack();
}
```

Σχήμα 6.34 : Κώδικας μεθόδων GameManager.cs 16

```
    }

    1 reference
    IEnumerator playerDelay()
    {
        yield return new WaitForSeconds(2.5f);
        Debug.Log("Player Turn");
        uiManager.GetComponent<UiManager>().pawnSelector.SetActive(true);
        turnManager.GetComponent<TurnManager>().newTurn();
        uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(true);

    }


    1 reference
    public void newTurn()
    {
        StartCoroutine(playerDelay());

        GameObject[] garbageVFX = GameObject.FindGameObjectsWithTag("AttackVFX");
        foreach (GameObject vfx in garbageVFX) {
            Destroy(vfx);
        }

    }
```

Σχήμα 6.35 : Κώδικας μεθόδων GameManager.cs 17

6.11.2.11 Παράλληλες μέθοδοι

Στην προηγούμενη ενότητα έγινε αναφορά σε ασύγχρονες μεθόδους και σε χρονικές καθυστερήσεις. Σε αυτή την ενότητα θα γίνει η ανάλυση της λογικής πίσω από τις παράλληλες μεθόδους και την σημασία τους στην ομαλή εκτέλεση του προγράμματος. Αρχικά για να δηλωθεί μια μέθοδος η οποία θα τρέξει παράλληλα με την κύρια ροή του προγράμματος πρέπει να οριστεί ως τύπος IEnumerator. Η κλήση των μεθόδων του τύπου IEnumerator γίνεται με την εντολή StartCoroutine (όνομα μεθόδου) και η χρονική καθυστέρηση με την οποία θα εκτελεστούν ορίζεται από την εντολή yield return new WaitForSeconds (χρόνος καθυστέρησης). Ο λόγος ύπαρξης των καθυστερήσεων ανάμεσα στις κλήσεις των μεθόδων είναι διαφορά ταχύτητας ανάμεσα στην εκτέλεση των εντολών του προγράμματος και στην κίνηση των πιονιών επάνω στο ταμπλό. Για την μετακίνηση ενός πιονιού από ένα σημείο Α σε ένα σημείο Β χρειάζονται 2 δευτερόλεπτα (μεταβλητή lerpTime). Η χρήση της χρονικής καθυστέρησης εξασφαλίζει ότι δεν θα εκτελεστεί καμία εντολή κατά την διάρκεια κίνησης του πιονιού. Πέρα από τον χρόνο μετακίνησης των πιονιών έχει σημασία και ο χρόνος ζωής των οπτικών εφέ ο οποίος συμπεριλαμβάνεται στον υπολογισμό της καθυστέρησης των μεθόδων.

Λειτουργικότητα ανά μέθοδο.

1. executeAttack(): Εξασφαλίζει ότι το πιόνι του αντιπάλου έχει ολοκληρώσει την κίνηση του πριν την εκτέλεση ικανότητας.
2. playerMovementSafety(): Απενεργοποιεί το UI του παίκτη μέχρι να ολοκληρωθεί η κίνηση του πιονιού του.
3. PlayerDelay(): Εξασφαλίζει ότι έχουν ολοκληρωθεί όλες οι κινήσεις και τα οπτικά εφέ πριν έρθει η σειρά του παίκτη.
4. loadNextSceneWin(): Δίνει μικρή καθυστέρηση πριν την μετάβαση στην σκηνή YouWin.
5. loadNextSceneLose(): Δίνει μικρή καθυστέρηση πριν την μετάβαση στην σκηνή YouLose.

```
IEnumerator playeMovementSafety()
{
    Debug.Log("Disabling bars");
    uiManager.GetComponent<UiManager>().pawnSelector.SetActive(false);
    uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(false);
     yield return new WaitForSeconds(2.1f);
    uiManager.GetComponent<UiManager>().pawnSelector.SetActive(true);
    uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(true);

}
```

Σχήμα 6.36 : Κώδικας μεθόδων GameManager.cs 18

```
1 reference
private IEnumerator loadNextSceneWin() {

    yield return new WaitForSeconds(3.5f);
    SceneManager.LoadScene("YouWinScene");

}
1 reference
private IEnumerator loadNextSceneLose()
{

    yield return new WaitForSeconds(3.5f);
    SceneManager.LoadScene("YouLoseScene");

}
```

Σχήμα 6.37 : Κώδικας μεθόδων GameManager.cs 19

6.12 Επίλογος

Η Δομή του κώδικα της εφαρμογής Orbs περιστρέφεται γύρω από το script GameManager.cs .Όλη η λειτουργικότητα των διαφόρων script συνδέεται με το GameManager.cs το οποίο διαχειρίζεται τόσο την εξέλιξη της παρτίδας όσο και την πλοήγηση μεταξύ των μενού της εφαρμογής. Ο πλήρης κώδικας του GameManager.cs βρίσκεται στο παράρτημα Ε.

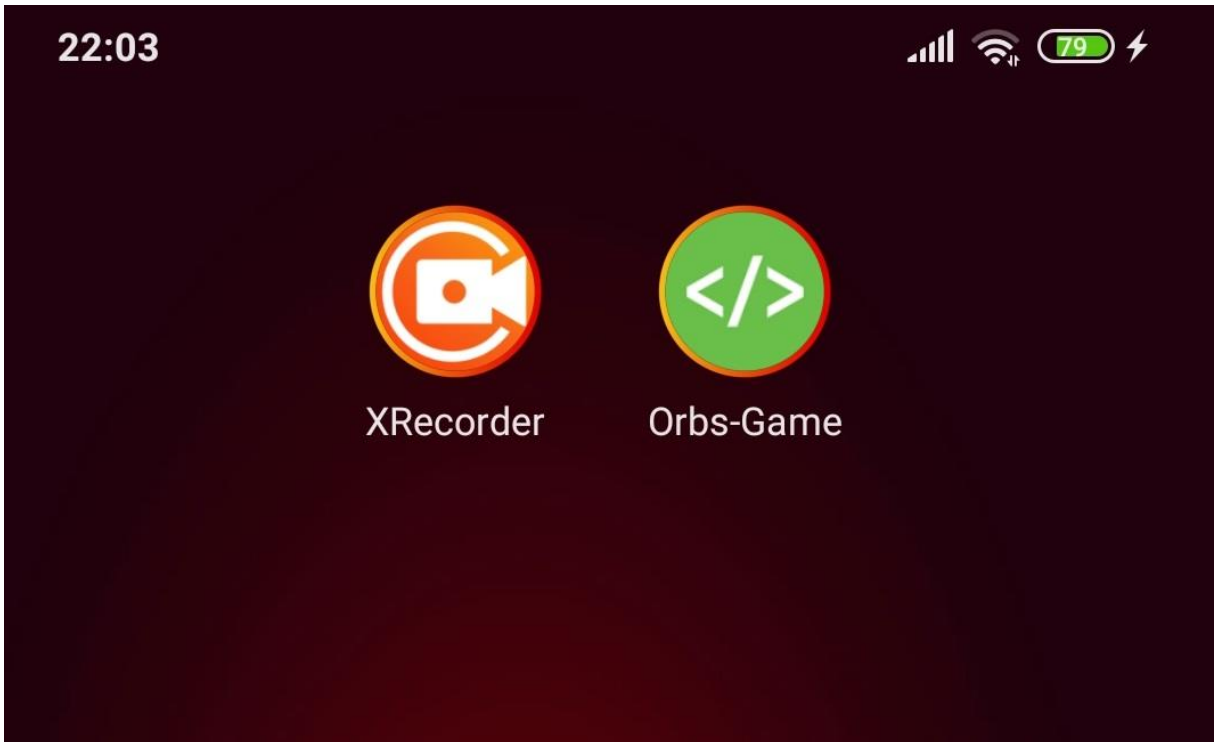
Κεφάλαιο 7ο: Χρήση εφαρμογής

7.1 Εισαγωγή

Ο σκοπός αυτού του κεφαλαίου είναι να περιγράψει τις κινήσεις τις οποίες πρέπει να εκτελέσει ο χρήστης προκειμένου να παίξει το παιχνίδι Orbs . Παρά το γεγονός ότι τόσο οι κανόνες του παιχνιδιού όσο και οι λειτουργίες των ξεχωριστών στοιχείων της διεπαφής καλυφθήκαν σε προηγούμενα κεφάλαια η ύπαρξη ενός κεφαλαίου το οποίο αφορά ειδικά την ροή χρήσης θα βοηθήσει σημαντικά στην κατανόηση της εφαρμογής Orbs από πιθανούς μελλοντικούς χρήστες.

7.2 Άνοιγμα Εφαρμογής

Μετά την εγκατάσταση της η εφαρμογή Orbs θα εμφανιστεί ανάμεσα στις υπόλοιπες εφαρμογές οι οποίες βρίσκονται εγκατεστημένες εντός της συσκευής. Όταν ο χρήστης εντοπίσει το εικονίδιο της εφαρμογής όπως φαίνεται στο σχήμα 7.1 πατάει επάνω του προκειμένου η εφαρμογή να ανοίξει. Κατά το άνοιγμα της εφαρμογής η συσκευή πρέπει να βρίσκεται σε οριζόντια θέση για να εμφανιστεί σωστά η διεπαφή της πρώτης οθόνης του παιχνιδιού.



Σχήμα 7.1: Εμφάνιση εικονιδίου εφαρμογής Orbs στην Android συσκευή

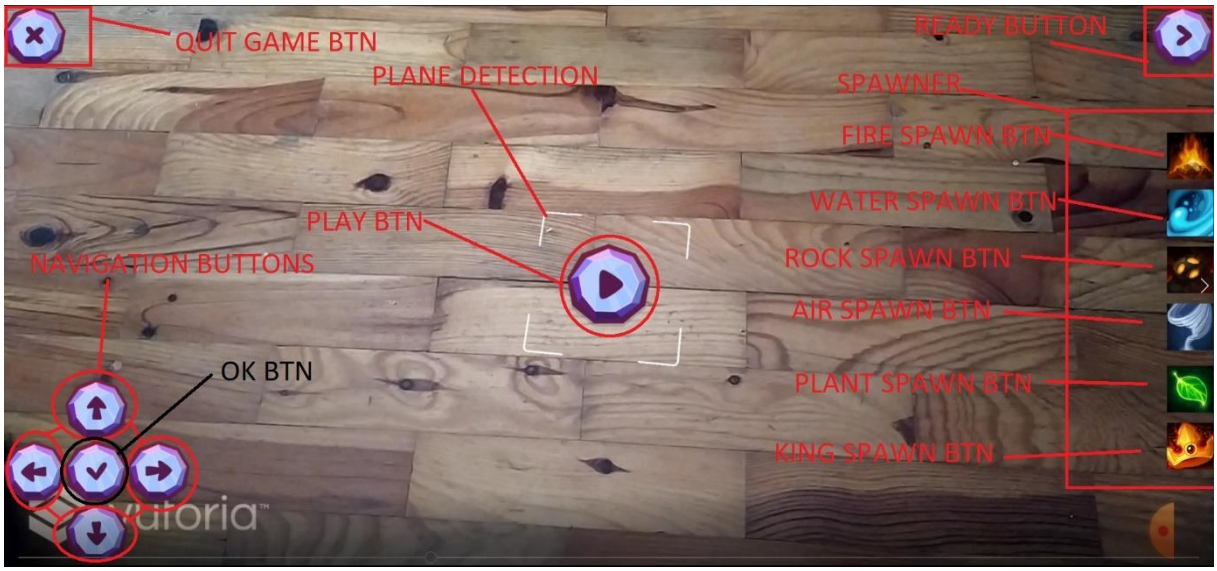
7.3 Ροή Χρήσης

Κατά την είσοδο του χρήση στην εφαρμογή εμφανίζεται η πρώτη οθόνη της εφαρμογής όπως αυτή παρουσιάζεται στο σχήμα 7.2.

Σχήμα 7.2 : Ανάλυση διεπαφής της αρχικής σκηνής

Με το πάτημα του κουμπιού Play η εφαρμογή προχωρά στην οθόνη της παρτίδας ενώ αντίθετα με το κουμπί Exit η εφαρμογή κλείνει. Στην οθόνη της παρτίδας έχουν εμφανιστεί τα κουμπιά εμφάνισης πιονιών (Pawn Spawner) το κουμπί Play, τα κουμπιά πλοήγησης του κέρσορα (navigation buttons), το κουμπί επιβεβαίωσης θέσης (Ok Button), το κουμπί εξόδου από την παρτίδα (Quit Game Button) και το κουμπί ετοιμότητας (Ready Button).

Για την εμφάνιση του ταμπλό και των πιονιών του αντιπάλου ο χρήστης πρέπει πρώτα να πατήσει το κουμπί Play το οποίο θα εμφανίσει τα πιόνια του αντιπάλου και τα αντικείμενα σημαίες στον αέρα. Με το άνοιγμα της οθόνης της παρτίδας η εφαρμογή ξεκινά την ανίχνευση επιφάνειας στον χώρο. Μόλις η εφαρμογή ανιχνεύσει κατάλληλη επιφάνεια εμφανίζει ένα λευκό τετράγωνο στο κέντρο της οθόνης εντός του οποίου πρέπει να πατήσει ο χρήστης για την εμφάνιση του ταμπλό (σχήμα 7.3).



Σχήμα 7.3: Ανάλυση διεπαφής της σκηνής του παιχνιδιού πριν την έναρξη της παρτίδας

Μετά την επιτυχή εμφάνιση του ταμπλό στην οθόνη ο χρήστης ξεκινά την διαδικασία του στησίματος. Για να στηθεί ένα πιόνι επάνω στο ταμπλό ο χρήστης πρέπει να πατήσει στο ανάλογο εικονίδιο εμφάνισης πιονιού, να μετακινήσει τον κέρσορα με τα κουμπιά πλοήγησης στο τετράγωνο στο οποίο επιθυμεί να τοποθετήσει το πιόνι και τέλος να πατήσει το κουμπί επιβεβαίωσης. Εάν το κουμπί επιβεβαίωσης πατηθεί ενώ το επιλεγμένο πιόνι έχει ήδη τοποθετηθεί στο ταμπλό τότε το πιόνι εξαφανίζεται και πρέπει να στηθεί εκ νέου. Εάν ο χρήστης επιχειρήσει να τοποθετήσει ένα πιόνι σε λανθασμένη θέση βάσει των κανόνων δεν εκτελείται καμία ενέργεια μέχρι ο χρήστης να επιλέξει αποδεκτή θέση. Με το πάτημα του κουμπιού εξόδου ο χρήστης μεταφέρεται στην αρχική οθόνη της εφαρμογής και η τρέχουσα παρτίδα τερματίζεται. Το κουμπί ετοιμότητας ολοκληρώνει την φάση του στησίματος και ξεκινά την παρτίδα με την προϋπόθεση όλα να πιόνια του παίκτη να είναι τοποθετημένα επάνω στο ταμπλό, σε διαφορετική περίπτωση δεν εκτελείται καμία ενέργεια.

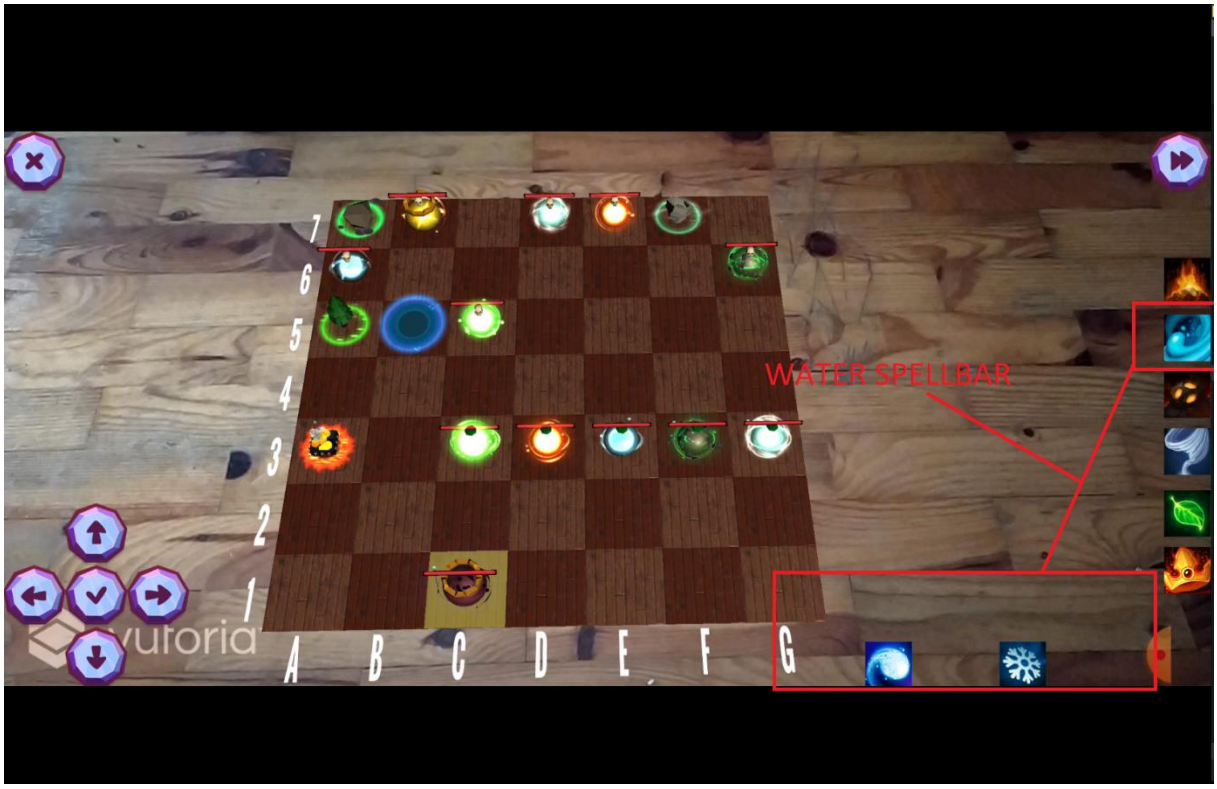
Το πάτημα του κουμπιού ετοιμότητας αντικαθιστά τα κουμπιά εμφάνισης πιονιών(Pawn Selector) με τα κουμπιά επιλογής πιονιών (Pawn Selector) το κουμπί επιβεβαίωσης θέσης(Ok Button) με το κουμπί επιβεβαίωσης κίνησης (Confirm Move Button) και το κουμπί ετοιμότητας(Ready Button) με το κουμπί λήξης γύρου(End Turn Button) όπως φαίνεται στο σχήμα 7.4.

Σχήμα 7.4 : Ανάλυση διεπαφής της σκηνής του παιχνιδιού κατά την διάρκεια της παρτίδας 1

Για να εκτελέσει ο χρήστης μία κίνηση επιλέγει το πιόνι το οποίο επιθυμεί να μετακινήσει πατώντας το ανάλογο κουμπί επιλογής πιονιού, μετακινεί τον κέρσορα στο τετράγωνο στο οποίο επιθυμεί να μετακινήσει το επιλεγμένο πιόνι και πατάει το κουμπί επιβεβαίωσης κίνησης. Σε περίπτωση που η

κίνηση η οποία επιθυμεί να εκτελέσει ο χρήστης απαγορεύεται από τους κανόνες του παιχνιδιού το πάτημα του κουμπιού επιβεβαίωσης κίνησης δεν εκτελεί καμία ενέργεια.

Το πάτημα ενός κουμπιού επιλογής πιονιού εμφανίζει και την ανάλογη μπάρα ικανοτήτων (spellbar) στο κάτω δεξί μέρος της οθόνης όπως φαίνεται στο σχήμα 7.5. Για να εκτελέσει ο χρήστης μία ικανότητα πιονιού πρέπει πρώτα να επιλέξει το πιόνι το οποίο θα εκτελέσει την ικανότητα, να μετακινήσει τον κέρσορα στο τετράγωνο στο οποίο θέλει να εκτελέσει την ικανότητα και τέλος να πατήσει το κουμπί της ικανότητας από την μπάρα ικανοτήτων του επιλεγμένου πιονιού.



Σχήμα 7.5 : Ανάλυση διεπαφής της σκηνής του παιχνιδιού κατά την διάρκεια της παρτίδας 2

Το κουμπί λήξης γύρου μπορεί να πατηθεί ανά πάσα στιγμή κατά την διάρκεια του γύρου του παίκτη. Πατώντας το κουμπί λήξης γύρου έρχεται η σειρά του αντιπάλου και η διεπαφή του παίκτη απενεργοποιείται μέχρι να είναι ξανά η σειρά του παίκτη να εκτελέσει τις κινήσεις του.

Με την λήξη της παρτίδας ο χρήστης μεταφέρεται στην τελευταία οθόνη του παιχνιδιού στην οποία υπάρχουν τα κουμπιά Replay και Exit (σχήμα 7.6).Το πάτημα του κουμπιού Replay ξεκινά μία νέα παρτίδα ενώ αντίθετα το κουμπί Exit κλείνει την εφαρμογή.

Σχήμα 7.6 : Ανάλυση διεπαφής της τελικής σκηνής του παιχνιδιού

7.4 Επίλογος

Ο σχεδιασμός της εφαρμογής Orbs έγινε δίνοντας βαρύτητα στην απλότητα χρήσης και λαμβάνοντας υπόψη την χρήση της επαυξημένης πραγματικότητας. Με βάση αυτά έγινε προσπάθεια έτσι ώστε το UI να αφήνει όσο το δυνατόν περισσότερο χώρο στην οθόνη για να μπορούν να εμφανιστούν τα στοιχεία του παιχνιδιού μέσω της κάμερας αλλά χωρίς να δυσκολεύεται ο χρήστης να αλληλεπιδράσει με την διεπαφή.

Κεφάλαιο 8ο: Φάσεις Ανάπτυξης

Σε αυτό το κεφάλαιο θα γίνει η περιγραφή των φάσεων ανάπτυξης της εφαρμογής Orbs σε χρονολογική σειρά και των συμβιβασμών οι οποίοι χρειάστηκε να γίνουν τόσο λόγω περιορισμένου χρόνου ανάπτυξης όσο και πολυπλοκότητας.

8.1 Δημιουργία του παιχνιδιού

Η πρώτη φάση ανάπτυξης ήταν η συγγραφή των κανόνων του παιχνιδιού και του τρόπου παιξίματος . Στα αρχικά στάδια της συγγραφής δοκιμάστηκαν διάφορα μεγέθη ταμπλό (8x8, 6x6) διαφορετικές αρχικές θέσεις και αριθμοί πιονιών (6,7,8). Τελικά αποφασίστηκε το μέγεθος του ταμπλό να γίνει 7x7 με 6 πιόνια ανά παίκτη ως η ιδανικότερη αναλογία χώρου και ποικιλίας κινήσεων.

Παρόμοιος πειραματισμός έγινε τόσο με το πλήθος των ικανοτήτων ανά πιόνι όσο και με την πολυπλοκότητα των ικανοτήτων αλλά λόγω χρόνου ακολουθήθηκε μια πιο μινιμαλιστική προσέγγιση με 1-2 απλές ικανότητες ανά πιόνι.

8.2 Μεταφορά στην επαυξημένη πραγματικότητα

Σε αυτό το στάδιο προετοιμάστηκε το έδαφος για να λειτουργήσει η εφαρμογή κάνοντας χρήση της επαυξημένης πραγματικότητας εγκαθιστώντας την μηχανή Vuforia όπως περιγράφεται στην ενότητα 3.6 και προσθέτοντας τα αντικείμενα ARCamera, Plane Finder και Ground Plane Stage στην σκηνή του παιχνιδιού (Board Scene) όπως αναφέρεται στην ενότητα 5.2.11. Η χρήση των Plane Finder και Ground Plane Stage επέτρεψε στην εφαρμογή να εμφανίσει το ταμπλό του παιχνιδιού σε οποιαδήποτε επίπεδη επιφάνεια δεδομένων των κατάλληλων συνθηκών φωτισμού. Σε αντίθετη περίπτωση θα έπρεπε να είχε γίνει χρήση Image Target η οποία θα επέβαλε την προσθήκη πραγματικών (μη ψηφιακών) εικόνων για την ομαλή λειτουργία της εφαρμογής μειώνοντας την πρακτικότητα της.

8.3 Σχεδίαση Ταμπλό και πιονιών

Την είσοδο των αντικειμένων επαυξημένης πραγματικότητας της προηγούμενης ενότητας στην σκήνη του παιχνιδιού ακολούθησε η κατασκευή του ταμπλό όπως περιγράφεται στην ενότητα 5.2.4 και η κατασκευή των πιονιών. Σε αυτή την φάση της ανάπτυξης τα πιόνια αποτελούνταν μόνο από το αντικείμενο Sphere και ένα material για τον χρωματισμό τους. Το αντικείμενο board μαζί με τα πλακίδια (tiles) τοποθετήθηκαν ως παιδιά (children) στο αντικείμενο Ground Plane Stage ενώ τα πιόνια μετατράπηκαν σε αντικείμενα prefab.

8.4 Δημιουργία σημαιών

Την κατασκευή των πιονιών ακολούθησε η κατασκευή μίας πρώιμης μορφής των σημαιών (flag Objects) . Για την κατασκευή χρησιμοποιήθηκαν assets από το asset store του Unity τα οποία δέχθηκαν επεξεργασία τόσο στην κλίμακα τους όσο και στα οπτικά εφέ τα οποία είχαν προκειμένου να μπορέσουν να εισαχθούν στην σκηνή του παιχνιδιού. Αντίστοιχα με τα πιόνια τα αντικείμενα σημαίες μετατράπηκαν σε αντικείμενα prefab

8.5 Κατασκευή Κέρσορα

Σε αντίθεση με την κατασκευή και τον προγραμματισμό των προαναφερθέντων αντικειμένων τόσο η κατασκευή όσο και η λειτουργικότητα του κέρσορα ολοκληρώθηκαν σε μία φάση καθώς ήταν

απαραίτητο ο κέρσορας να είναι πλήρως λειτουργικός για να μπορεί να γίνει ο σωστός έλεγχος των υπόλοιπων στοιχείων του προγράμματος σε μεταγενέστερο στάδιο. Το αντικείμενο του κέρσορα αποτελείται από ένα tile με κίτρινο ημιδιαφανές material όπως περιγράφεται στην ενότητα 5.2.5. Σε αυτό το σημείο εισήχθη το αντικείμενο canvas με τα πλήκτρα πλοήγησης και κατασκευάστηκαν τα script GameManager.cs, CursorScript.cs και Movement.cs με τις αντίστοιχες μεθόδους για την μετακίνηση του κέρσορα(βλ. κεφάλαιο 6) και τα αντίστοιχα αντικείμενα-δοχεία εντός της Board Scene.

8.6 Αρχικοποίηση αντικειμένων

Έχοντας και την λειτουργία του κέρσορα έτοιμη το επόμενο στάδιο ανάπτυξης ήταν η αρχικοποίηση των αντικειμένων επάνω στο ταμπλό. Αρχικά δημιουργήθηκε το κουμπί Play και προστέθηκε στην διεπαφή για να μπορεί ο χρήστης να ξεκινήσει την διαδικασία αρχικοποίησης. Στην συνέχεια δημιουργήθηκαν το script BoardInitializer με το αντίστοιχο αντικείμενο-δοχείο και τα script των πιονιών με τις μεθόδους για το στήσιμο των πιονιών του υπολογιστή . Ακολούθησε η προσθήκη των κουμπιών επιλογής πιονιών στην διεπαφή και το κουμπί επιβεβαίωσης τοποθέτησης και τέλος η συγγραφή των σχετικών μεθόδων (βλ. κεφάλαιο 6).

8.7 Αναβάθμιση διεπαφής

Καθώς πλέον η διεπαφή είχε αρκετά κουμπιά για τις μέχρι αυτό το σημείο λειτουργίες έγινε μια αναβάθμιση στις εικόνες των κουμπιών προκειμένου να είναι πιο διακριτά στον χρήστη. Για να γίνει αυτό προστέθηκαν τα αντίστοιχα Icon Packs από το asset Store του Unity όπως περιγράφεται στην ενότητα 3.8 .

8.8 Κίνηση πιονιών παίκτη

Το πρώτο βήμα για την εισαγωγή της λειτουργίας της κίνησης ήταν η προσθήκη των κουμπιών του αντικειμένου PawnSelector, το κουμπί ετοιμότητας (ReadyButton) και το κουμπί επιβεβαίωσης κίνησης (ConfirmMoveButton). Έχοντας πλέον ανάγκη για εμφάνιση και εξαφάνιση στοιχείων της διεπαφής δημιουργήθηκαν το αντικείμενο και το script UiManager με τις μεθόδους διαχείρισης των προαναφερθέντων λειτουργιών. Εν συνεχεία έγινε συγγραφή των μεθόδων για την κίνηση των πιονιών του παίκτη και για τον έλεγχο ορθότητας κίνησης.

8.9 Ικανότητες πιονιών παίκτη -Μπάρα ζωής

Για την υλοποίηση των ικανοτήτων των πιονιών του παίκτη αρχικά προστέθηκαν οι μπάρες ικανοτήτων στην διεπαφή. Στην συνέχεια δημιουργήθηκαν τα script και τα αντικείμενα των ικανοτήτων τα οποία τοποθετήθηκαν στην Board Scene . Για να γίνει ο έλεγχος της σωστής λειτουργίας των ικανοτήτων, σε αυτό το σημείο έπρεπε να κατασκευαστούν και οι μπάρες ζωής των πιονιών προκειμένου να φαίνεται η επίδραση των ικανοτήτων στους πόντους ζωής. Ως εκ τούτου έγινε η δημιουργία του script HealhBar.cs και του αντικειμένου HealhBar το οποίο προστέθηκε στην δομή των prefab αντικειμένων των πιονιών. Στο τέλος αυτής της φάσης ανάπτυξης έγινε η συγγραφή των μεθόδων των ικανοτήτων και της αυξομείωσης πόντων ζωής .

8.10 Οπτική αναβάθμιση

Η οπτική αναβάθμιση του παιχνιδιού έλαβε χώρα σε αυτό το σημείο ανάπτυξης της εφαρμογής Orbs. Μέχρι αυτό το σημείο είχε δοθεί έμφαση κυρίως στην λειτουργικότητα του παιχνιδιού έναντι της αισθητικής και ως εκ τούτου η εφαρμογή υστερούσε σημαντικά στο οπτικό της κομμάτι. Για την

εκτέλεση της οπτικής αναβάθμισης έγινε προσθήκη πακέτων από το asset store τα οποία περιείχαν prefab αντικείμενα (μοντέλα και οπτικά εφέ) προς χρήση (βλ. 3.8). Η οπτική αναβάθμιση περιλάμβανε την εισαγωγή των διακοσμητικών αντικειμένων στα πιόνια (βλ. 5.2.7) και την ενσωμάτωση οπτικών εφέ σε πιόνια, ικανότητες πιονιών και αντικείμενα σημαίες.

8.11 Ενέργειες Υπολογιστή- Διαχείριση γύρων

Έχοντας ολοκληρώσει τις λειτουργίες για τις ενέργειες του χρήστη ξεκίνησε η ανάπτυξη των μεθόδων οι οποίες καθόριζαν τις ενέργειες του υπολογιστή. Για να υπάρξει η εναλλαγή γύρων μεταξύ παίκτη και υπολογιστή δημιουργήθηκε το script TurnManager.cs μαζί με το αντικείμενο-δοχείο του Turn Manager και τις ανάλογες μεθόδους. Επίσης προστέθηκαν οι μέθοδοι διαχείρισης γύρων στο script GameManager. Τέλος έγινε η συγγραφή των μεθόδων για την κίνηση και την εκτέλεση ικανοτήτων του υπολογιστή κατά την διάρκεια του γύρου του.

8.12 Κατασκευή σκηνών πλοήγησης-Λήξη παρτίδας - Ήχος

Η τελευταία φάση ανάπτυξης του παιχνιδιού Orbs περιείχε την κατασκευή των σκηνών StartingScene, YouWinScene και YouLoseScene μαζί με το εικαστικό και τον ήχο υποβάθρου . Επίσης έγινε η συγγραφή του κώδικα για τον έλεγχο λήξης παρτίδας μαζί με τις αντίστοιχες μεθόδους και τέλος δημιουργήθηκαν το αντικείμενο SceneSwapper μαζί με το script SceneSwapper.cs τα οποία προστέθηκαν σε όλες τις σκηνές για την πλοήγηση του παίκτη μεταξύ των οθονών (σκηνών) του παιχνιδιού.

8.13 Προκλήσεις ανάπτυξης

Αυτή η ενότητα αφορά τις δυσκολίες οι οποίες εμφανίστηκαν κατά την ανάπτυξη της εφαρμογής Orbs λόγω της χρήσης επαυξημένης πραγματικότητας και τις λύσεις οι οποίες εφαρμόστηκαν προκειμένου να ξεπεραστούν.

Η εφαρμογή Orbs κάνει χρήση της κάμερας της συσκευής για να ανιχνεύει την επιφάνεια επάνω στην οποία εμφανίζεται το ταμπλό . Εάν κατά διάρκεια χρήσης της εφαρμογής ο χρήστης αλλάξει απότομα ή κατά πολύ την θέση της κάμερας τότε υπάρχει η περίπτωση η εφαρμογή να σταματήσει την ανίχνευση επιφάνειας και το ταμπλό μαζί με τα περιεχόμενα του να εξαφανιστούν. Το γεγονός αυτό από μόνο του δεν αποτελεί πρόβλημα καθώς τα στοιχεία του παιχνιδιού παραμένουν ως έχουν μέχρις ότου η εφαρμογή μπορέσει να ανιχνεύσει ξανά μια αποδεκτή επιφάνεια και να επαναφέρει στην οθόνη το ταμπλό. Η πρόκληση εμφανίζεται λόγω του γεγονότος ότι η θέση στην οποία επανέρχεται το ταμπλό είναι κάθε φορά διαφορετική και ορίζεται ανάλογα με την τρέχουσα θέση της συσκευής. Επομένως, όταν ένας χρήστης έχει την συσκευή στα χέρια του και δεν έχει κάποιο τρόπο να την κρατήσει απολύτως ακίνητη η θέση του ταμπλό στην οθόνη μεταβάλλεται. Αυτό έχει ως αποτέλεσμα τα τετράγωνα του ταμπλό να μην έχουν σταθερές συντεταγμένες στον χώρο δημιουργώντας πρόβλημα τόσο στην μετακίνηση των πιονιών από θέση σε θέση κατά την διάρκεια της παρτίδας όσο και στην εμφάνιση οποιουδήποτε αντικειμένου στην οθόνη καθώς δεν μπορούσαν να δοθούν συντεταγμένες οι οποίες να είναι σωστές σε κάθε περίπτωση. Για την επίλυση αυτού το προβλήματος δημιουργήθηκε μια σχέση γονιού-παιδιού ανάμεσα στα τετράγωνα και τα πιόνια ή τα αντικείμενα σημαίες. Έτσι όταν ένα πιόνι ή αντικείμενο σημαία καταλαμβάνει ένα τετράγωνο τότε θεωρείται παιδί αυτού και η θέση του ορίζεται πάντα στο ίδιο σημείο σε σχέση με το τετράγωνο (επάνω από το τετράγωνο και στο κέντρο του).

Η απόδοση σχέσης γονιού-παιδιού ανάμεσα στα τετράγωνα και τα υπόλοιπα στοιχεία του παιχνιδιού έλυσε αμέσως το ζήτημα της μεταβλητής θέσης αλλά εμφάνισε ένα άλλο πρόβλημα. Κατά την δυναμική

εμφάνιση των αντικειμένων ως παιδιά των τετραγώνων η κλίμακα τους στον άξονα Y εμφανιζόταν ως το 1/3 της ορισμένης τιμής. Για παράδειγμα μία σφαίρα η οποία είχε διαστάσεις X=1, Y=1, Z=1 εμφανιζόταν ως δίσκος καθώς η διάσταση Y (ύψος) χωρίς να γίνει κάποια αλλαγή εμφανιζόταν με τιμή 0,33. Δυστυχώς η αιτία αυτής της συμπεριφοράς δεν βρέθηκε αλλά το πρόβλημα λύθηκε θέτοντας την τιμή x της κλίμακας Y των prefab αντικειμένων σε 3x προκειμένου εκείνα να εμφανιστούν με το επιθυμητό σχήμα στο ταμπλό μετά την παραμόρφωση.

8.14 Επίλογος

Ο λόγος για τον οποίο τα βήματα της εφαρμογής Orbs έγιναν με την σειρά που αναφέρεται στις παραπάνω ενότητες ήταν έτσι ώστε να είναι εφικτός ο έλεγχος των λειτουργιών που είχαν προστεθεί σε προηγούμενα στάδια. Κατά αυτόν τον τρόπο σε κάθε στάδιο ανάπτυξης στο οποίο εισαγόταν μια νέα λειτουργία στην εφαρμογή υπήρχε η δυνατότητα ελέγχου της ομαλής ροής του προγράμματος και διευκολυνόταν σημαντικά ο εντοπισμός και η διόρθωση λαθών

Κεφάλαιο 9ο: Επίλογος-Συμπεράσματα-Τρόποι βελτίωσης

9.1 Επίλογος-Συμπεράσματα

Στην παρούσα διπλωματική εργασία μελετήθηκε σε θεωρητικό επίπεδο η χρήση εκτεταμένης πραγματικότητας με έμφαση στα βιντεοπαιχνίδια και σε πρακτικό επίπεδο η εφαρμογή της επαυξημένης πραγματικότητας μέσω της ανάπτυξης του παιχνιδιού Orbs. Με την ολοκλήρωση του παρόντος συγγράμματος προέκυψαν τα παρακάτω συμπεράσματα:

1.Παρά το γεγονός ότι η ιδέα της εκτεταμένης πραγματικότητας υπάρχει από περίπου τα μέσα του 20ου αιώνα [18] η χρήση της άρχισε να εδραιώνεται στο ευρύ κοινό μόλις την τελευταία δεκαετία. Αυτό οφείλεται κυρίως στο γεγονός ότι η τεχνολογία την οποία είχε στην διάθεση του ο μέσος χρήστης δεν ικανοποιούσε τις απαιτήσεις των εφαρμογών εκτεταμένης πραγματικότητας.

2. Οι τεχνολογίες εκτεταμένης πραγματικότητας έχουν εφαρμογή σε πληθώρα αντικειμένων όπως εκπαίδευση, στρατιωτικές ασκήσεις, ιατρική, βιντεοπαιχνίδια και άλλα δίνοντας ισχυρό κίνητρο για την περαιτέρω έρευνα και ανάπτυξη των εν λόγω τεχνολογιών.

3. Όσον αφορά τα βιντεοπαιχνίδια, μέχρι στιγμής φαίνεται να έχει δοθεί μεγαλύτερη προσοχή στην τεχνολογία της εικονικής πραγματικότητας έναντι της μικτής ωστόσο και οι δύο τεχνολογίες βρίσκονται σε πρώιμο στάδιο συγκριτικά με τα συμβατικά βιντεοπαιχνίδια. Με το ενδιαφέρον γύρω από τις τεχνολογίες εκτεταμένης πραγματικότητας να αυξάνεται και την σταθερή εξέλιξη υλικού και λογισμικού το μέλλον αυτού του είδους παιχνιδιών δείχνει πολλά υποσχόμενο.

4. Η μικτή πραγματικότητα και κατ' επέκταση η επαυξημένη πραγματικότητα έχουν μία ιδιαίτερη εφαρμογή στα βιντεοπαιχνίδια. Αρχικά με την χρήση GPS και την φορητότητα των συσκευών οι εφαρμογές μικτής πραγματικότητας εισήγαγαν τα βιντεοπαιχνίδια σε εξωτερικό χώρο. Εξίσου ενδιαφέρουσα είναι η μεταφορά των ήδη υπάρχοντών επιτραπέζιων παιχνιδιών σε παιχνίδια μικτής πραγματικότητας όπως το σκάκι ή τα παιχνίδια τύπου Dungeons and Dragons δίνοντας νέες επιλογές για παίκτες οι οποίοι βρίσκονται στον ίδιο χώρο . Μια ιδιαίτερα πρωτοπόρα ιδέα από την Nintendo συνδύασε την επαυξημένη πραγματικότητα με τηλεκατευθυνόμενα αυτοκίνητα φέρνοντας το ιδιαίτερα δημοφιλές παιχνίδι Mario Kart στον πραγματικό κόσμο. Τέλος ας μην ξεχνάμε ότι ακόμα και χωρίς κάποιο ιδιαίτερο χαρακτηριστικό τα παιχνίδια εκτεταμένης πραγματικότητας είναι από μόνα τους μια πολύ φρέσκια και ξεχωριστή προσθήκη στον χώρο τον βιντεοπαιχνιδιών και γενικότερα της ψυχαγωγίας. Ο λόγος για τον οποίο δόθηκαν τα παραπάνω παραδείγματα είναι για να αναδειχθεί το εύρος των δυνατοτήτων της μικτής πραγματικότητας στα βιντεοπαιχνίδια και η πληθώρα νέων εμπειριών που μπορεί να προσφέρει τόσο με νέα παιχνίδια όσο και με τροποποιήσεις παλαιότερων παιχνιδιών.

5.Δυστυχώς η πληθώρα των θετικών στοιχείων τα οποία προσφέρει η χρήση των τεχνολογιών μικτής πραγματικότητας δεν αποκλείει την ύπαρξη προβλημάτων και δυσκολιών. Το κύριο μειονέκτημα των εφαρμογών επαυξημένης πραγματικότητας είναι μεγάλη εξάρτηση τους από τις συνθήκες του πραγματικού κόσμου όπως ο φωτισμός και ο χώρος με αποτέλεσμα να υπάρχει συχνά ασυνέχεια στην λειτουργία τους. Ένα επίσης σοβαρό πρόβλημα της εν λόγω τεχνολογίας και συγκεκριμένα με την μηχανή Vuforia είναι η εκμάθηση της ιδίως σε ερασιτεχνικό επίπεδο καθώς το διαθέσιμο εκπαιδευτικό υλικό είναι σημαντικά μικρότερο σε όγκο και έχει χαμηλότερο ρυθμό ενημέρωσης συγκριτικά με άλλα αντικείμενα της πληροφορικής.

6.Η μηχανή Unity είναι ένα εξαιρετικό εργαλείο με τεράστιο εύρος επιλογών για όσους θέλουν να ασχοληθούν με την δημιουργία παιχνιδιών, είναι φιλική όσον αφορά την εκμάθηση της ακόμα και για άτομα χωρίς υπόβαθρο στον προγραμματισμό χωρίς όμως αυτό να μειώνει τις δυνατότητες για εξεζητημένες και πολύπλοκες διεργασίες.

9.2 Μελλοντική επέκταση και βελτιώσεις εφαρμογής Orbs

Παρακάτω αναφέρονται τα βήματα για την μελλοντική επέκταση της εφαρμογής Orbs και την βελτίωση της τρέχουσας έκδοσης.

- Υλοποίηση λειτουργίας 2 παικτών τοπικά ή μέσω διαδικτύου.
- Προσθήκη περισσότερων ικανοτήτων στα πιόνια.
- Αναβάθμιση του αλγορίθμου κινήσεων του υπολογιστή στην λειτουργία ενός παίκτη για μείωση της τυχαιότητας .
- Προσθήκη οπτικών και ηχητικών εφέ
- Αναβάθμιση γραφικών και οπτικής διαυγείας εφαρμογής.
- Βελτίωση δομής του κώδικα.
- Εκκαθάριση περιττών asset για μείωση του όγκου της εφαρμογής.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] "• Media revenue worldwide by category 2020 | Statista." https://www.statista.com/statistics/1132706/media-revenue-worldwide.

[2] "Gaming Industry vs. Other Entertainment Industries (2021) - Raise Your Skillz." https://raiseyourskillz.com/gaming-industry-vs-other-entertainment-industries-2021.

[3] "Video-Game Industry Revenues Exceed Sports, Film Combined in 2020." https://www.businessinsider.com/video-game-industry-revenues-exceed-sports-and-film-combined-idc-2020-12.

[4] L. Gong, Å. Fast-Berglund, and B. Johansson, "A Framework for Extended Reality System Development in Manufacturing," *IEEE Access*, vol. 9, pp. 24796–24813, 2021, doi: 10.1109/ACCESS.2021.3056752.

[5] F. Pallavicini, A. Pepe, and M. E. Minissi, "Gaming in Virtual Reality: What Changes in Terms of Usability, Emotional Response and Sense of Presence Compared to Non-Immersive Video Games?," *Simulation and Gaming*, vol. 50, no. 2, pp. 136–159, Apr. 2019, doi: 10.1177/1046878119831420.

[6] N. Atiqa Natrah Mansor, M. Herman bin Jamaluddin, A. zaki hj shukor, N. Natrah Mansor, M. Herman Jamaluddin, and A. Zaki Shukor, "Concept and application of virtual reality haptic technology: A review," *Article in Journal of Theoretical and Applied Information Technology*, vol. 13, no. 14, 2017, [Online]. Available: https://www.researchgate.net/publication/319096104

[7] "'Haptic feedback' virtual reality Teslasuit can simulate everything from a bullet to a hug - ABC News." https://www.abc.net.au/news/science/2021-04-01/vr-teslasuit-simulates-virtual-reality-touch-haptic-feedback/100030320.

[8] O. Sarıgöz, "Augmented reality, virtual reality and digital games: A research on teacher candidates," *Educational Policy Analysis and Strategic Research*, vol. 14, no. 3, pp. 41–63, Sep. 2019, doi: 10.29329/epasr.2019.208.3.

[9] D. W. F. van Krevelen and R. Poelman, "A Survey of Augmented Reality Technologies, Applications and Limitations," *International Journal of Virtual Reality*, vol. 9, no. 2, 2010, doi: 10.20870/ijvr.2010.9.2.2767.

[10] "The 51 best VR games - CNET." https://www.cnet.com/pictures/best-vr-games.

[11] "The best augmented reality games and AR games for Android." https://www.androidauthority.com/best-augmented-reality-games-ar-games-android-755298.

[12] "Unity Real-Time Development Platform | 3D, 2D VR & AR Engine." https://unity.com.

[13] "Vuforia Developer Portal |." https://developer.vuforia.com.

[14] "Complete Unity® and Android Development: Build Games & Apps | Udemy." https://www.udemy.com/course/complete-unity-and-android-development-build-games-and-apps/?src=sac&kw=complete+unity+and.

[15] “Discover Augmented Reality Games - Unity/Vuforia *Updated* course | OfCourseMe | Find online courses.” https://courses.ofcourse.me/Discover-Augmented-Reality-Games-Unity-Vuforia-*Updated*/course-247007.

[16] D. Arsenault, “Video Game Genre, Evolution and Innovation,” 2009. [Online]. Available: http:/www.eludamos.org

[17] “Brackeys - YouTube.” https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA.

[18] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, and M. Ivkovic, “Augmented reality technologies, systems and applications,” *Multimedia Tools and Applications*, vol. 51, no. 1, pp. 341–377, Jan. 2011, doi: 10.1007/s11042-010-0660-6.

ΠΑΡΑΡΤΗΜΑ Α : Ενδεικτική παρτίδα του παιχνιδιού Orbs

Terrain Positions:

- Wind Terrain – B4
- Forest Terrain – D7
- Fire Terrain – E7
- Rock Terrain – F7
- Water Terrain – G5

Enemy Positions

- Enemy Air Orb – B5
- Enemy Rock Orb – B6
- Enemy Water Orb – D5
- Enemy Plant Orb – F6
- Enemy Fire Orb – G7
- Enemy King Orb – C7

Player Positions

- Player Air Orb – C2
- Player Rock Orb – G2
- Player Water Orb – F2
- Player Plant Orb – E2
- Player Fire Orb – D2
- Player King Orb – G1

Game:

Player Turn

1. Player Air Orb used Fly, Moves from C2 to C4.
2. Player Air Orb used Air Strike, Attacks from C4 to D5 for 1+6 dmg.
3. Enemy Water Orb takes 7 dmg, Current LP →10-7=3.
4. Enemy Water Orb is Displaced, Moves from D5 to E6.

Enemy Turn

1. Enemy King Orb, Moves from C7 to B7.

Player Turn

1. Player Air Orb used Air Strike, Attacks from C4 to B5 for 1+6 dmg.

2. Enemy Air Orb takes 7 dmg Current LP → 10-7=3.
3. Enemy Air Orb is Displaced, Moves from B5 to A6.
4. Player Water Orb, Moves from F2 to F3.

Enemy Turn

1. Enemy Fire Orb, Moves from G7 to G6.

Player Turn

1. Player Rock Orb Moves from G2 to G3.

Enemy Turn

1. Enemy Air Orb, Moves from A6 to A5.

Player Turn

1. Player Air Orb used Fly, Moves from C4 to A4.
2. Player Air Orb used Air Strike, Attacks from A4 to A5 for 1+6 dmg. Enemy Air Orb takes 7 dmg, Current LP →3-7= -4.
3. Enemy Air Orb is Destroyed.

Enemy Turn

1. Enemy Plant Orb, Moves from F6 to F5.

Player Turn

1. Player Water Orb Moves from F3 to F4.
2. Player Water Orb used Shatter, Attacks from F4 to F5 for 0 dmg.

Enemy Turn

1. Enemy Rock Orb, Moves from B6 to C5.
2. Enemy Plant Orb used Plant Strike, Attacks from F5 to F4 for 4 dmg.
3. Player Water Orb takes 4 dmg, Current LP → 10-4=6.

Player Turn

1. Player Water Orb used Water Cannon, Attacks from F4 to F5 for 2+5+5-2 dmg.
2. Enemy Plant Orb takes 10 dmg, Current LP → 10-10=0.
3. Enemy Plant Orb is Destroyed.

Enemy Turn

1. Enemy Water Orb, Moves from E6 to D5.

Player Turn

1. Player Water Orb Moves from F4 to E4.
2. Player Water Orb used Shatter, Attacks from E4 to D5 for 0 dmg.

Enemy Turn

1. Enemy Rock Orb, Moves from C5 to C4.

Player Turn

1. Player Air Orb used Fly, Moves from A4 to C3.
2. Player Air Orb used Air Strike, Attacks from C3 to C4 for 1+6 dmg.
3. Enemy Rock Orb takes 7 dmg, Current LP →10-7= 3.
4. Enemy Rock Orb is Displaced, moves from C4 to C5.

Enemy Turn

1. Enemy Rock Orb, Moves from C5 to D4.

Player Turn

1. Player Air Orb used Air Strike, Attacks from C3 to D4 for 1+6 dmg.
2. Enemy Rock Orb takes 7 dmg, Current LP →3-7= -4.
3. Enemy Air Orb is Destroyed.

Enemy Turn

1. Enemy Water Orb, Moves from D5 to D4.

Player Turn

1. Player Plant Orb used Heal, Heals from E2 to E4 for 3 LP.
2. Player Water Orb takes 3 LP, Current LP →6+3=9.

Enemy Turn

1. Enemy King Orb, Moves from B7 το A7.

Player Turn

1. Player Rock Orb, Moves from G3 to F4.
2. Player King Orb used Sacrifice, Attacks from G1 to D4 for 5+5 dmg
3. Enemy Water Orb Takes 10 dmg, Current LP → 3-10=-7.
4. Enemy Water Orb is Destroyed.
5. Player King take 5 dmg, Current LP → 10-5=5.

Enemy Turn

1. Enemy Fire Orb, Moves from G6 το G7.

Player Turn

1. Player Fire Orb, Moves from D2 to D3.
2. Player Fire Orb used Combustion, Attacks from D3 to G7 for 1 dmg.
3. Enemy fire Orb takes 1 dmg, Current LP → 10-1=9.
4. Player fire Orb take 2 dmg, Current LP → 10-2=8.

Enemy Turn

1. Enemy Fire Orb, Moves from G7 το G6.

Player Turn

1. Player Rock Orb Moves from F4 to E5.

Enemy Turn

1. Enemy Fire Orb, Moves from G6 το F5.
2. Enemy Fire Orb used Fireball, Attacks from F5 to E4 for 4 dmg.
3. Player Water Orb takes 4 dmg, Current LP → 9-4=5.

Player Turn

1. Player Plant Orb used Heal, Heals from E2 to E4 for 3 LP.
2. Player Water Orb takes 3 LP, Current LP →5+3=8.

Enemy Turn

1. Enemy King Orb, moves from A7 το B7.

2. Enemy Fire Orb used Fireball, Attacks from F5 to E4 for 4 dmg.
3. Player Water Orb takes 4 dmg, Current LP → 8-4=4.

Player Turn

1. Player Water Orb, moves from E4 to D4.
2. Player Plant Orb used Heal, Heals from E2 to D4 for 3 LP.
3. Player Water Orb takes 3 LP, Current LP → 4+3=7.

Enemy Turn

1. Enemy Fire Orb, Moves from F5 to E6.

Player Turn

1. Player Rock Orb, Moves from E5 to F6.
2. Player Rock Orb used Meteor, Attacks from F6 to E6 for 2+6 dmg.
3. Enemy Fire Orb takes 8 dmg, Current LP → 9-8=1.

Enemy Turn

1. Enemy Fire Orb, Moves from E6 to F5.
2. Enemy Fire Orb used Fireball, Attacks from F5 to F6 for 4 dmg.
3. Player Rock Orb takes 4 dmg, Current LP → 10-4=6

Player Turn

1. Player Fire Orb used Combustion, Attacks from D3 to F5 for 1 dmg.
2. Enemy Fire Orb takes 1 dmg, Current LP → 1-1=0.
3. Enemy Fire Orb is Destroyed.
4. Player fire Orb takes 2 dmg, Current LP → 8-2=6.

Enemy Turn

1. Enemy King Orb, moves from B7 το B6.

Player Turn

1. Player Air Orb used Fly, Moves from C3 to B5.
2. Player Air Orb used Air Strike, Attacks from B5 to B6 for 1+6 dmg.
3. Enemy King Orb takes 7 dmg, Current LP →10-7= 3.
4. Enemy King Orb is Displaced, moves from B6 to B7.

Enemy Turn

1. Enemy King Orb, moves from B7 το A7.

Player Turn

1. Player King Orb used Sacrifice, Attacks from G1 to A7 for 5 dmg
2. Enemy King Orb Takes 5 dmg, Current LP → 3-5=-2.
3. Enemy King Orb is Destroyed.
4. Player Wins!

ΠΑΡΑΡΤΗΜΑ Β : BoardInitializer.cs

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

public class BoardInitializer : MonoBehaviour
{
    public static BoardInitializer instance;

    [Header("General use Game Objects")]
    public GameObject cursor;
    public GameObject board;
    private GameObject cursorTile;
    public GameObject pawnToPlace;



    [Header("Tile Lists")]
    public List<GameObject> tiles; // All the tiles
    public List<GameObject> potentialEnemyTiles;
    public List<GameObject> enemyKingTileList; //Tiles that the enemy king can be placed on
```

```
//Lists Required for the rest of the game
public List<GameObject> enemyTiles; //final List of enemyTiles after setup
public List<GameObject> allyTiles; // Tiles that hold the ally pawns after setup
public List<GameObject> specialTerrainTiles; //List of tiles that hold special terrain



[Header("Pieces")]
public List<GameObject> enemyPieces;
public List<GameObject> playerPieces;
public List<GameObject> terrains; // List of terrains

// Start is called before the first frame update

void Start()
{

}

public void BoardInit()
{
    spawnCursor();
    spawnSpecialTerrain();
    setupEnemyPawns();
}



public void spawnSpecialTerrain()
{
    int rndFlag = 0;
    System.Random rnd = new System.Random();

    while (rndFlag < 5)
```

```
        {
            int tileRndIndex = rnd.Next(0, 48);
            if (!specialTerrainTiles.Contains(tiles[tileRndIndex]))
            {
                specialTerrainTiles.Add(tiles[tileRndIndex]);
                rndFlag++;
            }
        }
        for (int i = 0; i < 5; i++)
        {
            GameObject spawnTile = specialTerrainTiles[i];
            Vector3 spawnPos = new Vector3(spawnTile.transform.position.x,
spawnTile.transform.position.y + 0.005f, spawnTile.transform.position.z);
            Quaternion spawnRotation = spawnTile.transform.rotation;
            //Assign tile values to the terrain prefabs
            terrains[i].GetComponent<MasterTerrain>().currentTile = spawnTile;
            terrains[i].GetComponent<MasterTerrain>().currentTileIndex = tiles.IndexOf(spawnTile);
            //-----------
            Instantiate(terrains[i], spawnPos, spawnRotation, board.transform);
        }



    }


    public void setupEnemyPawns()
    {


        for (int i = 0; i < 49; i++)
        {
            if (i % 7 >= 4)
            {
                potentialEnemyTiles.Add(tiles[i]); // get all squares behind 4th row
```

```
    }
}
foreach (GameObject specialTerrainTile in specialTerrainTiles) //remove special terrain squares
{

    if (potentialEnemyTiles.Contains(specialTerrainTile))
    {
        potentialEnemyTiles.Remove(specialTerrainTile);

    }
}
foreach (GameObject tile in potentialEnemyTiles) //Contruct the list of suitable tiles for the enemy king
{
    if (tile.name.Contains("7"))
    {
        enemyKingTileList.Add(tile);
    }

}
//King Spawn
System.Random rnd = new System.Random();
//Choose random tile of the King Tile List
int tileRndIndex = rnd.Next(0, enemyKingTileList.Count );
GameObject spawnTile = enemyKingTileList[tileRndIndex];
Vector3 spawnPos = new Vector3(spawnTile.transform.position.x, spawnTile.transform.position.y + 0.01f, spawnTile.transform.position.z);
Quaternion spawnRotation = spawnTile.transform.rotation;
spawnRotation.y = 180f;
//Assing tile values to the king
enemyPieces[0].GetComponent<MasterPawn>().currentTile = spawnTile;
enemyPieces[0].GetComponent<MasterPawn>().currentTileIndex = tiles.IndexOf(spawnTile);
Instantiate(enemyPieces[0], spawnPos, spawnRotation, board.transform);
```

```
        //Remove tile and King from list so that no other piece will accidentaly spawn on the king tile or spawn the king 2 times
        enemyTiles.Add(spawnTile);
        potentialEnemyTiles.Remove(spawnTile);
        enemyPieces.Remove(enemyPieces[0]);

        // Set Rest of the pieces
        foreach (GameObject enemyPiece in enemyPieces)
        {
            tileRndIndex = rnd.Next(0, potentialEnemyTiles.Count );
            spawnTile = potentialEnemyTiles[tileRndIndex];
            spawnPos = new Vector3(spawnTile.transform.position.x, spawnTile.transform.position.y + 0.01f, spawnTile.transform.position.z);
            spawnRotation = spawnTile.transform.rotation;
            spawnRotation.y = 180f;
            //Assign Tile Values
            enemyPiece.GetComponent<MasterPawn>().currentTile = spawnTile;
            enemyPiece.GetComponent<MasterPawn>().currentTileIndex = tiles.IndexOf(spawnTile);
            Instantiate(enemyPiece, spawnPos, spawnRotation, board.transform);
            enemyTiles.Add(spawnTile);
            potentialEnemyTiles.Remove(spawnTile);



        }




    }

    public void spawnCursor()
    {

        cursorTile = tiles[23];
```

```
        Vector3 cursorSpawnPos = new Vector3(cursorTile.transform.position.x, cursorTile.transform.position.y + 0.001f, cursorTile.transform.position.z);
        Quaternion cursorSpawnRotation = cursorTile.transform.rotation;
        Instantiate(cursor, cursorSpawnPos, cursorSpawnRotation, board.transform);

    }


    //Modified so that the player can adjust pawn position at setup phase
    public void kingPawnSelect()
    {
        GameObject pawnExists = GameObject.FindGameObjectWithTag("KingOrb");
        if (pawnExists !=null) {
            allyTiles.Remove(pawnExists.GetComponent<MasterPawn>().currentTile);
            Destroy(GameObject.FindGameObjectWithTag("KingOrb"));
        }
        pawnToPlace = playerPieces[0];

    }

    public void firePawnSelect()
    {
        GameObject pawnExists = GameObject.FindGameObjectWithTag("FireOrb");
        if (pawnExists != null)
        {
            allyTiles.Remove(pawnExists.GetComponent<MasterPawn>().currentTile);
            Destroy(GameObject.FindGameObjectWithTag("FireOrb"));
        }
        pawnToPlace = playerPieces[1];

    }

    public void plantPawnSelect()
```

```
{
    GameObject pawnExists = GameObject.FindGameObjectWithTag("PlantOrb");
    if (pawnExists != null)
    {
        allyTiles.Remove(pawnExists.GetComponent<MasterPawn>().currentTile);
        Destroy(GameObject.FindGameObjectWithTag("PlantOrb"));
    }
    pawnToPlace = playerPieces[2];

}
public void airPawnSelect()
{
    GameObject pawnExists = GameObject.FindGameObjectWithTag("AirOrb");
    if (pawnExists != null)
    {
        allyTiles.Remove(pawnExists.GetComponent<MasterPawn>().currentTile);
        Destroy(GameObject.FindGameObjectWithTag("AirOrb"));
    }
    pawnToPlace = playerPieces[3];
}



public void waterPawnSelect()
{
    GameObject pawnExists = GameObject.FindGameObjectWithTag("WaterOrb");
    if (pawnExists != null)
    {
        allyTiles.Remove(pawnExists.GetComponent<MasterPawn>().currentTile);
        Destroy(GameObject.FindGameObjectWithTag("WaterOrb"));
    }
    pawnToPlace = playerPieces[4];
}
public void rockPawnSelect()
```

```
    {
        GameObject pawnExists = GameObject.FindGameObjectWithTag("RockOrb");
        if (pawnExists != null)
        {
            allyTiles.Remove(pawnExists.GetComponent<MasterPawn>().currentTile);
            Destroy(GameObject.FindGameObjectWithTag("RockOrb"));
        }
        pawnToPlace = playerPieces[5];

    }


    public void placeSelectedPawn()
    {

        GameObject cursorInstance = GameObject.FindGameObjectWithTag("cursor");
        GameObject selectedTile = tiles[cursorInstance.GetComponent<CursorScript>().currentTileIndex];

        if (pawnToPlace.name.Contains("King") && pawnToPlace != null)
        {
            if (selectedTile.name.Contains("1") && !allyTiles.Contains(selectedTile) && !specialTerrainTiles.Contains(selectedTile))
            {
                Vector3 spawnPos = new Vector3(selectedTile.transform.position.x, selectedTile.transform.position.y + 0.01f, selectedTile.transform.position.z);
                Quaternion pawnSpawnRotation = selectedTile.transform.rotation;
                pawnToPlace.GetComponent<MasterPawn>().currentTile = selectedTile;
                pawnToPlace.GetComponent<MasterPawn>().currentTileIndex = tiles.IndexOf(selectedTile);
                Instantiate(pawnToPlace, spawnPos, pawnSpawnRotation, board.transform);
                allyTiles.Add(selectedTile);
                pawnToPlace = null;
```

```
            }
        }
        else
        {
            if (cursorInstance.GetComponent<CursorScript>().currentTileIndex % 7 < 3 &&
!specialTerrainTiles.Contains(selectedTile) && !allyTiles.Contains(selectedTile) &&
pawnToPlace!=null)
            {
                Vector3 spawnPos = new Vector3(selectedTile.transform.position.x,
selectedTile.transform.position.y + 0.01f, selectedTile.transform.position.z);
                Quaternion pawnSpawnRotation = selectedTile.transform.rotation;
                pawnToPlace.GetComponent<MasterPawn>().currentTile = selectedTile;
                pawnToPlace.GetComponent<MasterPawn>().currentTileIndex =
tiles.IndexOf(selectedTile);
                Instantiate(pawnToPlace, spawnPos, pawnSpawnRotation, board.transform);
                allyTiles.Add(selectedTile);
                pawnToPlace = null;
            }
        }
    }
    // Update is called once per frame
    void Update()
    {
    }
}
```

ΠΑΡΑΡΤΗΜΑ C : Movement.cs

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

public class Movement : MonoBehaviour
{

    public List<GameObject> tiles;
    public BoardInitializer boardInitializer;



    [Header("Position Holders")]
    public List<GameObject> enemyTiles; //final List of enemyTiles after setup
    public List<GameObject> allyTiles; // Tiles that hold the ally pawns after setup
    public List<GameObject> specialTerrainTiles; //List of tiles that hold special terrain
    public List<GameObject> availablePositions;

    // Start is called before the first frame update
    void Start()
    {

    }

    //Get the occupied tiles from the boardInitilizer after the setup
    //This method will be triggered by the finish setup button (ReadyBtn)
    public void GetInitialPositions()
    {
        enemyTiles = boardInitializer.GetComponent<BoardInitializer>().enemyTiles;
        allyTiles = boardInitializer.GetComponent<BoardInitializer>().allyTiles;
        specialTerrainTiles = boardInitializer.GetComponent<BoardInitializer>().specialTerrainTiles;
```

```
}


public List<GameObject> findNeighbors(int tileIndex)
{
    List<GameObject> neighborTiles = new List<GameObject>();

    // find north tile
    if (tileIndex % 7 != 6)
    {
        neighborTiles.Add(tiles[tileIndex + 1]);
        //find northEast tile
        if (tileIndex <= 40)
        {
            neighborTiles.Add(tiles[tileIndex + 8]);
        }
        //find northwest tile
        if (tileIndex > 5)
        {
            neighborTiles.Add(tiles[tileIndex - 6]);
        }

    }
    //find south tile
    if (tileIndex % 7 != 0 && tileIndex != 0)
    {
        neighborTiles.Add(tiles[tileIndex - 1]);
        //find south east tile
        if (tileIndex < 42)
        {
            neighborTiles.Add(tiles[tileIndex + 6]);
        }
```

```
        //find south west tile
        if (tileIndex > 7)
        {
            neighborTiles.Add(tiles[tileIndex - 8]);

        }
    }
    //find west tile
    if (tileIndex >= 7)
    {
        neighborTiles.Add(tiles[tileIndex - 7]);
    }
    //find east tile
    if (tileIndex < 42)
    {
        neighborTiles.Add(tiles[tileIndex + 7]);
    }

    return neighborTiles;
}

public void updateTileAllyDestroyed(int currentTile)
{
    allyTiles.Remove(tiles[currentTile]);

}
public void updateTileEnemyDestroyed(int currentTile)
{
    enemyTiles.Remove(tiles[currentTile]);
}

public bool isValidAttack(int tileIndexPrev, GameObject tileToAttack)
{
```

```
        availablePositions = findNeighbors(tileIndexPrev);
        if (availablePositions.Contains(tileToAttack) && !specialTerrainTiles.Contains(tileToAttack))
        {
            return true;
        }
        else
        {
            return false;
        }

    }

    public int moveEnemyPawn(GameObject enemyPawn)
    {
        System.Random rnd = new System.Random();
        availablePositions = findNeighbors(enemyPawn.GetComponent<MasterPawn>().currentTileIndex);
        foreach (GameObject tile in availablePositions) {

        }
        List<GameObject> availablePositionsTemp = new List<GameObject>();
        foreach (GameObject tile in availablePositions)
        {
            if (!enemyTiles.Contains(tile) && !allyTiles.Contains(tile) && !specialTerrainTiles.Contains(tile))
            {
                availablePositionsTemp.Add(tile);
            }
        }
        if (availablePositionsTemp.Count != 0) {
            int positionsIndex = rnd.Next(0, availablePositionsTemp.Count);
            enemyTiles.Remove(enemyPawn.GetComponent<MasterPawn>().currentTile);
            enemyTiles.Add(availablePositionsTemp[positionsIndex]);
            return tiles.IndexOf(availablePositionsTemp[positionsIndex]);
```

```
    }
    else
    {
        return -1;
    }
}

public int enemyAttack(int attackerTileIndex) {
    int positionIndex = -1;
    System.Random rnd = new System.Random();
    List<GameObject> neighborTiles = findNeighbors(attackerTileIndex);
    List<GameObject> neighborOccupiedTiles= new List<GameObject>(); ;
    foreach (GameObject tile in neighborTiles) {
        if (allyTiles.Contains(tile)) {
            neighborOccupiedTiles.Add(tile);
        }

    }
    if (neighborOccupiedTiles.Count != 0)
    {

        positionIndex = rnd.Next(0, neighborOccupiedTiles.Count);
        return tiles.IndexOf(neighborOccupiedTiles[positionIndex]);
    }
    else {

        return positionIndex;
    }



}
public bool findAvailablePositions(int tileIndexPrev, GameObject nextTile)
```

```
    {
        availablePositions = findNeighbors(tileIndexPrev);
        List<GameObject> availablePositionsTemp = new List<GameObject>();
        foreach (GameObject tile in availablePositions)
        {
            if (!enemyTiles.Contains(tile) && !allyTiles.Contains(tile) &&
!specialTerrainTiles.Contains(tile))
            {
                availablePositionsTemp.Add(tile);
            }

        }



        if (availablePositionsTemp.Contains(nextTile))
        {
            allyTiles.Remove(tiles[tileIndexPrev]);
            allyTiles.Add(nextTile);

            return true;

        }
        else
        {
            return false;
        }

    }

    public int getNorthTileIndex(int currentIndex)
    {
```

```
    if (currentIndex % 7 == 6) //last row
    {
        return currentIndex;
    }
    else
    {
        return ++currentIndex;
    }


}

public int getSouthTileIndex(int currentIndex)
{
    if (currentIndex % 7 == 0 || currentIndex == 0) // first row
    {
        return currentIndex;
    }
    else { return --currentIndex; }
}

public int getEastTileIndex(int currentIndex)
{
    if (currentIndex >= 42)  //Column G
    {
        return currentIndex;
    }
    else { return currentIndex + 7; }
}
public int getWestTileIndex(int currentIndex)
{
```

```
        if (currentIndex < 7)  //Column G
        {
            return currentIndex;
        }
        else { return currentIndex - 7; }
    }

    //For Air and Rock Displacement Effects
    public int calculateDiplacement(int attackerTileIndex , int targetTileIndex) {
        int displacementTile=-1;
        if (targetTileIndex % 7 != 6 && targetTileIndex % 7 != 0 && targetTileIndex != 0) { //is not on the edges
            if (attackerTileIndex % 7 < targetTileIndex % 7) // Attacker south Target North

            {
                if (attackerTileIndex / 7 < targetTileIndex / 7)  //Attacker Left of Target
                {
                    if (targetTileIndex <= 40)
                    {
                        displacementTile = targetTileIndex + 8;
                    }
                }
                if(attackerTileIndex / 7 > targetTileIndex / 7) // Attacker Right of Target
                {
                    if (targetTileIndex >= 6)
                    {
                        displacementTile = targetTileIndex - 6;
                    }
                }
                if (attackerTileIndex / 7 == targetTileIndex / 7) // Same Column
                {
                    if (targetTileIndex <= 47)
                    {
```

```
            displacementTile = targetTileIndex + 1;
        }
    }
}
if (attackerTileIndex % 7 > targetTileIndex % 7) // Attacker north Target South
{


    if (attackerTileIndex / 7 < targetTileIndex / 7)  //Attacker Left of Target
    {
        if (targetTileIndex <= 42)
        {
            displacementTile = targetTileIndex + 6;
        }
    }
    if (attackerTileIndex / 7 > targetTileIndex / 7) // Attacker Right of Target
    {
        if (targetTileIndex >= 8)
        {
        displacementTile = targetTileIndex - 8;
        }
    }
    if (attackerTileIndex / 7 == targetTileIndex / 7) // Same Column
    {
        if (targetTileIndex >=1) {
        displacementTile = targetTileIndex - 1;
        }
    }



}
if (attackerTileIndex % 7 == targetTileIndex % 7)  // Same row
{
```

```
            if (attackerTileIndex / 7 < targetTileIndex / 7)  //Attacker Left of Target
            {
               if (targetTileIndex <= 41) {
               displacementTile = targetTileIndex +7 ;
               }
            }
            if (attackerTileIndex / 7 > targetTileIndex / 7) // Attacker Right of Target
            {
               if (targetTileIndex >= 7) {
               displacementTile = targetTileIndex - 7;
               }
            }




         }


      }

      return displacementTile;
   }
   public bool isDisplacementPossible(int tileIndex, GameObject tileToUpdate)
   {
      if (!allyTiles.Contains(tiles[tileIndex]) && !enemyTiles.Contains(tiles[tileIndex]) &&
!specialTerrainTiles.Contains(tiles[tileIndex]))
      {
         enemyTiles.Remove(tileToUpdate);
         enemyTiles.Add(tiles[tileIndex]);
         return true;
      }
      else {
```

```
            return false;
        }
    }

    public bool isFlyPossible(int tileIndex, GameObject tileToUpdate)
    {
        if (!allyTiles.Contains(tiles[tileIndex]) && !enemyTiles.Contains(tiles[tileIndex]) && !specialTerrainTiles.Contains(tiles[tileIndex]))
        {
            allyTiles.Remove(tileToUpdate);
            allyTiles.Add(tiles[tileIndex]);
            return true;
        }
        else
        {
            return false;
        }
    }
    //Special Terrain Buffs
    public bool isFireBuffed(int pawnIndex) {

        int terrainTileIndex = GameObject.FindGameObjectWithTag("FireTerrain").GetComponent<MasterTerrain>().currentTileIndex;
        List<GameObject> neighborTiles = findNeighbors(terrainTileIndex);
        if (neighborTiles.Contains(tiles[pawnIndex])) {
            return true;
        }
        else
        {

            return false;
        }
```

```
    }
    public bool isWaterBuffed(int pawnIndex)
    {

        int terrainTileIndex = GameObject.FindGameObjectWithTag("LakeTerrain").GetComponent<MasterTerrain>().currentTileIndex;
        List<GameObject> neighborTiles = findNeighbors(terrainTileIndex);
        if (neighborTiles.Contains(tiles[pawnIndex]))
        {
            return true;
        }
        else
        {

            return false;
        }

    }
    public bool isPlantBuffed(int pawnIndex)
    {

        int terrainTileIndex = GameObject.FindGameObjectWithTag("ForestTerrain").GetComponent<MasterTerrain>().currentTileIndex;
        List<GameObject> neighborTiles = findNeighbors(terrainTileIndex);
        if (neighborTiles.Contains(tiles[pawnIndex]))
        {
            return true;
        }
        else
        {

            return false;
```

```
        }


    }
    public bool isRockBuffed(int pawnIndex)
    {

        int terrainTileIndex = GameObject.FindGameObjectWithTag("RockTerrain").GetComponent<MasterTerrain>().currentTileIndex;
        List<GameObject> neighborTiles = findNeighbors(terrainTileIndex);
        if (neighborTiles.Contains(tiles[pawnIndex]))
        {
            return true;
        }
        else
        {

            return false;
        }


    }
    public bool isAirBuffed(int pawnIndex)
    {

        int terrainTileIndex = GameObject.FindGameObjectWithTag("TornadoTerrain").GetComponent<MasterTerrain>().currentTileIndex;
        List<GameObject> neighborTiles = findNeighbors(terrainTileIndex);
        if (neighborTiles.Contains(tiles[pawnIndex]))
        {
            return true;
        }
        else
        {
```

```
            return false;
        }

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

ΠΑΡΑΡΤΗΜΑ D : UiManager.cs

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

public class UiManager : MonoBehaviour

{

    public static UiManager instance;

    public BoardInitializer boardInitializer;
    public GameManager gameManager;
    public Movement movement;



    [Header("Buttons")]
    public GameObject playButton;
    public GameObject readyButton;
```

```
public GameObject okButton;
public GameObject confirmMoveBtn;
public GameObject endTurnBtn;

[Header("selectors")]
public GameObject pawnSpawner;
public GameObject pawnSelector;
public List<GameObject> pawnSelectorList;



[Header("SpellBars")]
public List<GameObject> spellBars;



// Start is called before the first frame update

public void PlayButtonPressed()
{


    playButton.SetActive(false);


}
public void firePawnSelected()
{
    foreach (GameObject spellbar in spellBars)
    {
        if (spellbar.name.Equals("FireSpellBar"))
        {
            spellbar.SetActive(true);
```

```
        }
        else { spellbar.SetActive(false); }

    }

}
public void waterPawnSelected()
{
    foreach (GameObject spellbar in spellBars)
    {
        if (spellbar.name.Equals("WaterSpellBar"))
        {
            spellbar.SetActive(true);
        }
        else { spellbar.SetActive(false); }

    }

}
public void EarthPawnSelected()
{
    foreach (GameObject spellbar in spellBars)
    {
        if (spellbar.name.Equals("EarthSpellBar"))
        {
            spellbar.SetActive(true);
        }
        else { spellbar.SetActive(false); }

    }

}
public void PlantPawnSelected()
```

```
{
    foreach (GameObject spellbar in spellBars)
    {
        if (spellbar.name.Equals("PlantSpellBar"))
        {
            spellbar.SetActive(true);
        }
        else { spellbar.SetActive(false); }

    }

}
public void AirPawnSelected()
{
    foreach (GameObject spellbar in spellBars)
    {
        if (spellbar.name.Equals("AirSpellBar"))
        {
            spellbar.SetActive(true);
        }
        else { spellbar.SetActive(false); }

    }

}
public void KingPawnSelected()
{
    foreach (GameObject spellbar in spellBars)
    {
        if (spellbar.name.Equals("KingSpellBar"))
        {
            spellbar.SetActive(true);
        }
```

```
        else { spellbar.SetActive(false); }

    }

}

void Start()
{
    pawnSelector.SetActive(false);
    confirmMoveBtn.SetActive(false);
    endTurnBtn.SetActive(false);
    foreach (GameObject spellbar in spellBars)
    {
        spellbar.SetActive(false);
    }
}
public void setUpReady()
{
    if (boardInitializer.GetComponent<BoardInitializer>().allyTiles.Count == 6)
    {
        pawnSpawner.SetActive(false);
        pawnSelector.SetActive(true);
        okButton.SetActive(false);
        confirmMoveBtn.SetActive(true);
        readyButton.SetActive(false);
        endTurnBtn.SetActive(true);
        movement.GetComponent<Movement>().GetInitialPositions();
        gameManager.GetComponent<GameManager>().generatePawnLists();



    }
}
// Update is called once per frame
```

```
    void Update()
    {

    }
}
```

ΠΑΡΑΡΤΗΜΑ Ε : GameManager.cs

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

public class GameManager : MonoBehaviour
{



    public static GameManager instance;
    [Header("GameOver Conditions")]
    public bool gameStarted = false;
    public bool win;
    public bool lose;




    [Header("Visuals")]
    public GameObject fireBallVFX;
    public GameObject firePillarVFX;
    public GameObject meteorVFX;
    public GameObject flyCircle;
    public GameObject airStrikeVFX;
    public GameObject healVFX;
    public GameObject seedStrikeVFX;
```

```
public GameObject waterCannonVFX;
public GameObject shatterVFX;
public GameObject sacrificeVFX;
public GameObject eFireVFX;
public GameObject eRockVFX;
public GameObject eWaterVFX;
public GameObject eKingVFX;
public GameObject eAirVFX;
public GameObject ePlantVFX;



[Header("Managers")]
public GameObject Movement;
public GameObject turnManager;
public GameObject uiManager;



[Header("Game Objects")]
public List<GameObject> tiles;
public GameObject board;
public GameObject cursorObject;
GameObject nextTile;
public GameObject pawnSelected;
GameObject targetPawn;



[Header("Attacks")]
public FireBall fireball;
public Combustion combustion;
```

```
public WaterCannon waterCannon;
public Shatter shatter;
public AirStrike airStrike;
public Fly fly;
public Meteor meteor;
public Smash smash;
public Heal heal;
public SeedStrike seedStrike;
public Sacrifice sacrifice;



[Header("EnemyAttacks")]
public EnemyFireBall eFireBall;
public EnemyAirBullet eAirBullet;
public EnemyMeteor eMeteor;
public EnemySeedStrike eSeedStrike;
public EnemyWaterCannon eWaterCannon;
public EnemyKingAttack eKingAttack;



[Header("Pawns")]
public List<GameObject> allyPawns;
public GameObject firePawn;
public GameObject waterPawn;
public GameObject rockPawn;
public GameObject airPawn;
public GameObject plantPawn;
public GameObject kingPawn;
//--------------------------
public List<GameObject> enemyPawns;
```

```
public GameObject enemyFirePawn;
public GameObject enemyWaterPawn;
public GameObject enemyRockPawn;
public GameObject enemyAirPawn;
public GameObject enemyPlantPawn;
public GameObject enemyKingPawn;



[Header("Indexes")]
public int currentTileIndex;
public int nextTileIndex;

[Header("Movement")]

Vector3 nextPosition = new Vector3(0, 0, 0); //Cursor next Position
private float lerptime = 2;
private float currentLerpTime = 0;
public bool movePawn = false;

public bool flyOn = false;
public bool validMove = false;
Vector3 posA;
Vector3 posB;
private GameObject nextPawnTile;
public List<GameObject> neighborTiles;



//Track Pawns
public void generatePawnLists()
{
```

```
    firePawn = GameObject.FindGameObjectWithTag("FireOrb");
    allyPawns.Add(firePawn);
    waterPawn = GameObject.FindGameObjectWithTag("WaterOrb");
    allyPawns.Add(waterPawn);
    rockPawn = GameObject.FindGameObjectWithTag("RockOrb");
    allyPawns.Add(rockPawn);
    airPawn = GameObject.FindGameObjectWithTag("AirOrb");
    allyPawns.Add(airPawn);
    plantPawn = GameObject.FindGameObjectWithTag("PlantOrb");
    allyPawns.Add(plantPawn);
    kingPawn = GameObject.FindGameObjectWithTag("KingOrb");
    allyPawns.Add(kingPawn);

    //-------------------

    enemyFirePawn = GameObject.FindGameObjectWithTag("EnemyFireOrb");
    enemyPawns.Add(enemyFirePawn);
    enemyWaterPawn = GameObject.FindGameObjectWithTag("EnemyWaterOrb");
    enemyPawns.Add(enemyWaterPawn);
    enemyRockPawn = GameObject.FindGameObjectWithTag("EnemyRockOrb");
    enemyPawns.Add(enemyRockPawn);
    enemyAirPawn = GameObject.FindGameObjectWithTag("EnemyAirOrb");
    enemyPawns.Add(enemyAirPawn);
    enemyPlantPawn = GameObject.FindGameObjectWithTag("EnemyPlantOrb");
    enemyPawns.Add(enemyPlantPawn);
    enemyKingPawn = GameObject.FindGameObjectWithTag("EnemyKingOrb");
    enemyPawns.Add(enemyKingPawn);


    gameStarted = true;
}
//Ally Attakcs
```

```
    //Fire Casts
    public void castFireBall()
    {
        int modifiedDamage = 0;
        if (turnManager.GetComponent<TurnManager>().attackAvailable)
        {

            cursorObject = GameObject.FindGameObjectWithTag("cursor");
            int targetTile;
            targetPawn = null;
            targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
            GameObject targetTileObject = tiles[targetTile];

            if
(Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn
>().currentTileIndex, targetTileObject))
            {
                foreach (GameObject pawn in enemyPawns)
                {

                    if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                    {
                        targetPawn = pawn;
                    }

                }
                if (targetPawn != null)

                {

                    Vector3 spawnPos = new Vector3(targetPawn.transform.position.x,
targetPawn.transform.position.y + 0.1f, targetPawn.transform.position.z);
                    Quaternion spawnRot = fireBallVFX.transform.rotation;
```

```
            GameObject projectile = Instantiate(fireBallVFX, spawnPos, spawnRot, board.transform)
as GameObject;
            projectile.transform.LookAt(targetTileObject.transform.position);

            projectile.GetComponent<Rigidbody>().AddForce(projectile.transform.forward * 10f);



            if (targetPawn.tag.Contains("Rock")) { modifiedDamage -= 2; }
            if (targetPawn.tag.Contains("Plant")) { modifiedDamage += 2; }
            if (targetPawn.tag.Contains("Water")) { modifiedDamage -= 1; }
            if
(Movement.GetComponent<Movement>().isFireBuffed(pawnSelected.GetComponent<MasterPawn>(
).currentTileIndex)) { modifiedDamage += 6; }



            if
(targetPawn.GetComponent<MasterPawn>().isGonnaDie(fireball.GetComponent<FireBall>().damage
+ modifiedDamage))
            {
               Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
               enemyPawns.Remove(targetPawn);

            }

targetPawn.GetComponent<MasterPawn>().takeDamage(fireball.GetComponent<FireBall>().damage
+ modifiedDamage);
            Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with fireball ");
            turnManager.GetComponent<TurnManager>().attackAvailable = false;
         }

      }
   }


}
```

```
    public void castCombustion()
    {
        int modifiedDamage = 0;
        if (turnManager.GetComponent<TurnManager>().attackAvailable)
        {
            cursorObject = GameObject.FindGameObjectWithTag("cursor");
            int targetTile;
            targetPawn = null;
            targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
            foreach (GameObject pawn in enemyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                {
                    targetPawn = pawn;


                }


            }
            if (targetPawn != null)
            {
                Vector3 spawnPos = new Vector3(targetPawn.transform.position.x,
targetPawn.transform.position.y, targetPawn.transform.position.z);
                Quaternion spawnRot = firePillarVFX.transform.rotation;


                Instantiate(firePillarVFX, spawnPos, spawnRot, board.transform);




                if (targetPawn.tag.Contains("Rock")) { modifiedDamage -= 2; }
                if (targetPawn.tag.Contains("Plant")) { modifiedDamage += 2; }
                if (targetPawn.tag.Contains("Water")) { modifiedDamage -= 1; }
                if
(Movement.GetComponent<Movement>().isFireBuffed(pawnSelected.GetComponent<MasterPawn>(
).currentTileIndex))
```

```
        {
            modifiedDamage += 6;
        }

        if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(combustion.GetComponent<Combustion>().damage + modifiedDamage))
        {
            Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
            enemyPawns.Remove(targetPawn);

        }

targetPawn.GetComponent<MasterPawn>().takeDamage(combustion.GetComponent<Combustion>().damage + modifiedDamage);
        if (pawnSelected.GetComponent<MasterPawn>().isGonnaDie(combustion.GetComponent<Combustion>().selfInflictedDamage))
        {

Movement.GetComponent<Movement>().updateTileAllyDestroyed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex);
            allyPawns.Remove(pawnSelected);
        }

pawnSelected.GetComponent<MasterPawn>().takeDamage(combustion.GetComponent<Combustion>().selfInflictedDamage);


        Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Combustion");
        Debug.Log(pawnSelected.tag + " inficts to self " + combustion.GetComponent<Combustion>().selfInflictedDamage + " Damage ");

        turnManager.GetComponent<TurnManager>().attackAvailable = false;

    }
```

```
    }
  }
  //Water Casts
  public void castWaterCannon()
  {
    int modifiedDamage = 0;
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {

      cursorObject = GameObject.FindGameObjectWithTag("cursor");
      int targetTile;
      targetPawn = null;
      targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
      GameObject targetTileObject = tiles[targetTile];
      if
(Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn
>().currentTileIndex, targetTileObject))
      {
        foreach (GameObject pawn in enemyPawns)
        {

          if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
          {
            targetPawn = pawn;
          }

        }
        if (targetPawn != null)

        {
          Vector3 spawnPos = new Vector3(pawnSelected.transform.position.x,
pawnSelected.transform.position.y + 0.03f, pawnSelected.transform.position.z);
          Quaternion spawnRot =waterCannonVFX.transform.rotation;
```

```
            GameObject waterCannonVisual = Instantiate(waterCannonVFX, spawnPos, spawnRot, board.transform) as GameObject;
            waterCannonVisual.transform.LookAt(targetPawn.transform.position);



            if (targetPawn.tag.Contains("Plant")) { modifiedDamage -= 2; }
            if (targetPawn.tag.Contains("Fire")) { modifiedDamage += 2; }
            if (Movement.GetComponent<Movement>().isWaterBuffed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex)) { modifiedDamage += 6; }
            if (targetPawn.GetComponent<MasterPawn>().status.Equals("Frozen")) { modifiedDamage += 5; }


            if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(waterCannon.GetComponent<WaterCannon>().damage + modifiedDamage))
            {
                Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
                enemyPawns.Remove(targetPawn);


            }

targetPawn.GetComponent<MasterPawn>().takeDamage(waterCannon.GetComponent<WaterCannon>().damage + modifiedDamage);
            Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with water Cannon ");
            turnManager.GetComponent<TurnManager>().attackAvailable = false;
        }
    }
}
```

```
public void castShatter()
{
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];

        if (Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetTileObject))
        {
            foreach (GameObject pawn in enemyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile && !pawn.tag.Contains("Fire"))
                {
                    targetPawn = pawn;
                    foreach (GameObject otherPawn in enemyPawns)
                    {
                        if (pawn != otherPawn)
                        {
                            otherPawn.GetComponent<MasterPawn>().status = "";
                        }
                    }


                }

            }
            if (targetPawn != null)
```

```
            {
                Destroy(GameObject.FindGameObjectWithTag("Frozen"));
                Vector3 spawnPos = targetPawn.transform.position;
                spawnPos.y += 0.005f;
                Instantiate(shatterVFX, spawnPos, shatterVFX.transform.rotation, targetPawn.transform);

                targetPawn.GetComponent<MasterPawn>().status = "Frozen";
                Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Shatter the next attack more damage");
                turnManager.GetComponent<TurnManager>().attackAvailable = false;

            }

        }
    }

}
//Air Casts
public void castAirStrike()
{
    int modifiedDamage = 0;
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {

        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        if (Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetTileObject))
        {
            foreach (GameObject pawn in enemyPawns)
```

```
        {

            if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
            {
                targetPawn = pawn;
            }

        }
        if (targetPawn != null)

        {
            Vector3 spawnPos = new Vector3(pawnSelected.transform.position.x, pawnSelected.transform.position.y+0.03f, pawnSelected.transform.position.z);
            Quaternion spawnRot = airStrikeVFX.transform.rotation;

           GameObject airStrikeVisual= Instantiate(airStrikeVFX, spawnPos, spawnRot, board.transform) as GameObject;
            airStrikeVisual.transform.LookAt(targetPawn.transform.position);

            if (Movement.GetComponent<Movement>().isAirBuffed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex)) { modifiedDamage += 6; }


            if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(airStrike.GetComponent<AirStrike>().damage + modifiedDamage))
            {
                Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
                enemyPawns.Remove(targetPawn);

            }
            else
            {
```

```
            int displacementTileIndex = Movement.GetComponent<Movement>().calculateDiplacement(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetPawn.GetComponent<MasterPawn>().currentTileIndex);
            if (displacementTileIndex != -1 && Movement.GetComponent<Movement>().isDisplacementPossible(displacementTileIndex, targetTileObject))
            {
                currentLerpTime = 0;
                posA = targetPawn.transform.position;
                posB = new Vector3(tiles[displacementTileIndex].transform.position.x, tiles[displacementTileIndex].transform.position.y + 0.01f, tiles[displacementTileIndex].transform.position.z);
                targetPawn.GetComponent<MasterPawn>().currentTile = tiles[displacementTileIndex];
                targetPawn.GetComponent<MasterPawn>().currentTileIndex = displacementTileIndex;
                pawnSelected = targetPawn;
                movePawn = true;
                StartCoroutine(playeMovementSafety());

            }

        }

targetPawn.GetComponent<MasterPawn>().takeDamage(airStrike.GetComponent<AirStrike>().damage + modifiedDamage);
            Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Air Strike");
            turnManager.GetComponent<TurnManager>().attackAvailable = false;


        }

    }
  }

}
```

```
public void castFly()
{
    if (turnManager.GetComponent<TurnManager>().movementAvailable)
    {
        currentLerpTime = 0;
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        if (Movement.GetComponent<Movement>().isFlyPossible(targetTile, pawnSelected.GetComponent<MasterPawn>().currentTile))
        {

            Vector3 spawnPos1 = pawnSelected.transform.position;
            Quaternion spawnRot =flyCircle.transform.rotation;
            Vector3 spawnPos2 = new Vector3(targetTileObject.transform.position.x, targetTileObject.transform.position.y + 0.005f, targetTileObject.transform.position.z);
            Instantiate(flyCircle, spawnPos1, spawnRot, board.transform);
            Instantiate(flyCircle, spawnPos2, spawnRot, board.transform);

            pawnSelected.GetComponent<MasterPawn>().currentTileIndex = targetTile;
            pawnSelected.GetComponent<MasterPawn>().currentTile = targetTileObject;
            posA = pawnSelected.transform.position;
            posB = new Vector3(pawnSelected.transform.position.x, pawnSelected.transform.position.y + 0.07f, pawnSelected.transform.position.z);
            movePawn = true;
            StartCoroutine(playeMovementSafety());
            StartCoroutine(flyDelay(targetTileObject));
            turnManager.GetComponent<TurnManager>().movementAvailable = false;



        }
```

```
        }


    }
    private IEnumerator flyDelay(GameObject flyToTile)
    {
        yield return new WaitForSeconds(2);
        movePawn = false;
        currentLerpTime = 0;
        posA = pawnSelected.transform.position;
        posB = new Vector3(flyToTile.transform.position.x, flyToTile.transform.position.y + 0.01f, flyToTile.transform.position.z);
        movePawn = true;


    }
    //Rock Casts
    public void castMeteor() {
        int modifiedDamage = 0;
        if (turnManager.GetComponent<TurnManager>().attackAvailable)
        {
            cursorObject = GameObject.FindGameObjectWithTag("cursor");
            int targetTile;
            targetPawn = null;
            targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
            GameObject targetTileObject = tiles[targetTile];
            if (Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, targetTileObject))
            {
                foreach(GameObject pawn in enemyPawns)
                {
                    if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                    {
```

```
            targetPawn = pawn;

        }

    }
    if(targetPawn != null)
    {

        Vector3 spawnPos = new Vector3(targetPawn.transform.position.x, targetPawn.transform.position.y+0.01f, targetPawn.transform.position.z);
        Quaternion spawnRot = meteorVFX.transform.rotation;
        Instantiate(meteorVFX, spawnPos, spawnRot, board.transform);
        if (Movement.GetComponent<Movement>().isRockBuffed(pawnSelected.GetComponent<MasterPawn>().currentTileIndex))
        {
            modifiedDamage += 6;
        }
        if (targetPawn.GetComponent<MasterPawn>().status.Equals("Frozen")) { modifiedDamage += 5; }

        if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(meteor.GetComponent<Meteor>().damage + modifiedDamage))
        {
            Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
            enemyPawns.Remove(targetPawn);

        }

        targetPawn.GetComponent<MasterPawn>().takeDamage(meteor.GetComponent<Meteor>().damage + modifiedDamage);
        Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Meteor ");
        turnManager.GetComponent<TurnManager>().attackAvailable = false;
```

```
            }

        }


    }
}

public void castSmash() { }
//Plant Casts
public void castSeedStrike() {
    int modifiedDamage = 0;
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        int targetTile;
        targetPawn = null;
        targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        GameObject targetTileObject = tiles[targetTile];
        if
(Movement.GetComponent<Movement>().isValidAttack(pawnSelected.GetComponent<MasterPawn
>().currentTileIndex, targetTileObject))
        {
            foreach (GameObject pawn in enemyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                {
                    targetPawn = pawn;


                }


            }
            if (targetPawn != null)
            {
```

```
            Vector3 spawnPos = new Vector3(targetPawn.transform.position.x,
targetPawn.transform.position.y + 0.1f, targetPawn.transform.position.z);
            Quaternion spawnRot = seedStrikeVFX.transform.rotation;
            GameObject projectile = Instantiate(seedStrikeVFX, spawnPos, spawnRot,
board.transform) as GameObject;
            projectile.transform.LookAt(targetTileObject.transform.position);


            projectile.GetComponent<Rigidbody>().AddForce(projectile.transform.forward * 10f);


            if (targetPawn.tag.Contains("Fire")) { modifiedDamage -= 2; }
            if (targetPawn.tag.Contains("Water")) { modifiedDamage += 2; }
            if
(Movement.GetComponent<Movement>().isPlantBuffed(pawnSelected.GetComponent<MasterPawn
>().currentTileIndex))
            {
               modifiedDamage += 6;
            }
            if (targetPawn.GetComponent<MasterPawn>().status.Equals("Frozen")) {
modifiedDamage += 5; }


            if
(targetPawn.GetComponent<MasterPawn>().isGonnaDie(seedStrike.GetComponent<SeedStrike>().da
mage + modifiedDamage))
            {
               Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
               enemyPawns.Remove(targetPawn);


            }


targetPawn.GetComponent<MasterPawn>().takeDamage(seedStrike.GetComponent<SeedStrike>().da
mage + modifiedDamage);
            Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with seed Strike ");
            turnManager.GetComponent<TurnManager>().attackAvailable = false;
        }


    }
```

```
    }
  }
  public void castHeal()
  {
    if (turnManager.GetComponent<TurnManager>().attackAvailable)
    {
      cursorObject = GameObject.FindGameObjectWithTag("cursor");
      int targetTile;
      targetPawn = null;
      targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
      foreach (GameObject pawn in allyPawns)
      {

        if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
        {

          targetPawn = pawn;

        }
      }
      Vector3 spawnPos = new Vector3(targetPawn.transform.position.x,
targetPawn.transform.position.y + 0.09f, targetPawn.transform.position.z);
        Quaternion spawnRot = healVFX.transform.rotation;
        Instantiate(healVFX, spawnPos, spawnRot, board.transform);

targetPawn.GetComponent<MasterPawn>().takeHealing(heal.GetComponent<Heal>().healing);
    }
    turnManager.GetComponent<TurnManager>().attackAvailable = false;
    Debug.Log(pawnSelected.tag + " Heals " + targetPawn.tag + " for " +
heal.GetComponent<Heal>().healing);


  }
```

```
    //King Casts
    public void castSacrifice() {
        if (turnManager.GetComponent<TurnManager>().attackAvailable)
        {

            cursorObject = GameObject.FindGameObjectWithTag("cursor");
            int targetTile;
            targetPawn = null;
            targetTile = cursorObject.GetComponent<CursorScript>().currentTileIndex;
            GameObject targetTileObject = tiles[targetTile];
            foreach (GameObject pawn in enemyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == targetTile)
                {

                    targetPawn = pawn;
                }

            }
            if(targetPawn != null)
            {

                Vector3 spawnPos1 = pawnSelected.transform.position;
                Quaternion spawnRot = sacrificeVFX.transform.rotation;
                Vector3 spawnPos2 = new Vector3(targetTileObject.transform.position.x,
targetTileObject.transform.position.y + 0.005f, targetTileObject.transform.position.z);
                Instantiate(sacrificeVFX, spawnPos1, spawnRot, board.transform);
                Instantiate(sacrificeVFX, spawnPos2, spawnRot, board.transform);
                if
(targetPawn.GetComponent<MasterPawn>().isGonnaDie(sacrifice.GetComponent<Sacrifice>().dama
ge)) {

                    Movement.GetComponent<Movement>().updateTileEnemyDestroyed(targetTile);
```

```
                enemyPawns.Remove(targetPawn);
            }
            if (pawnSelected.GetComponent<MasterPawn>().isGonnaDie(sacrifice.GetComponent<Sacrifice>().damage))
            {



            }

targetPawn.GetComponent<MasterPawn>().takeDamage(sacrifice.GetComponent<Sacrifice>().damage);

pawnSelected.GetComponent<MasterPawn>().takeDamage(sacrifice.GetComponent<Sacrifice>().damage);
            Debug.Log(pawnSelected.tag + " Attacks " + targetPawn.tag + " with Sacrifice ");
            turnManager.GetComponent<TurnManager>().attackAvailable = false;

        }

        }


    }


    //Pawn Movements Methods
    public void MoveNorth()
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        currentTileIndex = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        nextTileIndex = Movement.GetComponent<Movement>().getNorthTileIndex(currentTileIndex);
        nextTile = tiles[nextTileIndex];
        nextPosition = new Vector3(nextTile.transform.position.x, nextTile.transform.position.y + 0.001f, nextTile.transform.position.z);
```

```
        cursorObject.GetComponent<CursorScript>().changeCurrentTile(nextTileIndex, nextTile.name);
        cursorObject.transform.position = nextPosition;
    }

    public void MoveSouth()
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        currentTileIndex = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        nextTileIndex = Movement.GetComponent<Movement>().getSouthTileIndex(currentTileIndex);
        nextTile = tiles[nextTileIndex];
        nextPosition = new Vector3(nextTile.transform.position.x, nextTile.transform.position.y + 0.001f,
nextTile.transform.position.z);
        cursorObject.GetComponent<CursorScript>().changeCurrentTile(nextTileIndex, nextTile.name);
        cursorObject.transform.position = nextPosition;




    }
    public void MoveEast()
    {
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        currentTileIndex = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        nextTileIndex = Movement.GetComponent<Movement>().getEastTileIndex(currentTileIndex);
        nextTile = tiles[nextTileIndex];
        nextPosition = new Vector3(nextTile.transform.position.x, nextTile.transform.position.y + 0.001f,
nextTile.transform.position.z);
        cursorObject.GetComponent<CursorScript>().changeCurrentTile(nextTileIndex, nextTile.name);
        cursorObject.transform.position = nextPosition;



    }

    public void MoveWest()
    {
```

```
        cursorObject = GameObject.FindGameObjectWithTag("cursor");
        currentTileIndex = cursorObject.GetComponent<CursorScript>().currentTileIndex;
        nextTileIndex = Movement.GetComponent<Movement>().getWestTileIndex(currentTileIndex);
        nextTile = tiles[nextTileIndex];
        nextPosition = new Vector3(nextTile.transform.position.x, nextTile.transform.position.y + 0.001f,
nextTile.transform.position.z);
        cursorObject.GetComponent<CursorScript>().changeCurrentTile(nextTileIndex, nextTile.name);
        cursorObject.transform.position = nextPosition;



    }



    //Pawn Selection Methods
    public void firePawnSelected()
    {
        movePawn = false;
        pawnSelected = GameObject.FindGameObjectWithTag("FireOrb");
    }
    public void waterPawnSelected()
    {
        movePawn = false;
        pawnSelected = GameObject.FindGameObjectWithTag("WaterOrb");
    }
    public void plantPawnSelected()
    {
        movePawn = false;
        pawnSelected = GameObject.FindGameObjectWithTag("PlantOrb");
    }
    public void airPawnSelected()
    {
        movePawn = false;
        pawnSelected = GameObject.FindGameObjectWithTag("AirOrb");
```

```
    }
    public void rockPawnSelected()
    {
        movePawn = false;
        pawnSelected = GameObject.FindGameObjectWithTag("RockOrb");
    }
    public void kingPawnSelected()
    {
        movePawn = false;
        pawnSelected = GameObject.FindGameObjectWithTag("KingOrb");
    }


    //Movement
    public void moveSelectedPawn()
    {

        if (turnManager.GetComponent<TurnManager>().movementAvailable)
        {
            nextPawnTile = tiles[cursorObject.GetComponent<CursorScript>().currentTileIndex];
            if (!pawnSelected.tag.Contains("Enemy"))
            {
                validMove = Movement.GetComponent<Movement>().findAvailablePositions(pawnSelected.GetComponent<MasterPawn>().currentTileIndex, nextPawnTile);
                if (validMove)
                {

                    currentLerpTime = 0;
                    posA = pawnSelected.transform.position;
                    posB = new Vector3(nextPawnTile.transform.position.x, nextPawnTile.transform.position.y + 0.01f, nextPawnTile.transform.position.z);
                    pawnSelected.GetComponent<MasterPawn>().currentTile = nextPawnTile;
```

```
            pawnSelected.GetComponent<MasterPawn>().currentTileIndex =
cursorObject.GetComponent<CursorScript>().currentTileIndex;
            movePawn = true;
            StartCoroutine(playeMovementSafety());
            turnManager.GetComponent<TurnManager>().movementAvailable = false;

        }
      }
    }


  }
  public void moveEnemyPawn()
  {

    System.Random rnd = new System.Random();
    string selectedTag = enemyPawns[rnd.Next(0, enemyPawns.Count)].tag;
    pawnSelected = GameObject.FindGameObjectWithTag(selectedTag);

    int nextPawnTileIndex =
Movement.GetComponent<Movement>().moveEnemyPawn(pawnSelected);
    if (nextPawnTileIndex != -1)
    {
      nextPawnTile = tiles[nextPawnTileIndex];
      currentLerpTime = 0;
      posA = pawnSelected.transform.position;
      posB = new Vector3(nextPawnTile.transform.position.x, nextPawnTile.transform.position.y +
0.01f, nextPawnTile.transform.position.z);
      pawnSelected.GetComponent<MasterPawn>().currentTile = nextPawnTile;
      pawnSelected.GetComponent<MasterPawn>().currentTileIndex = nextPawnTileIndex;
      movePawn = true;
      StartCoroutine(executeAttack());

    }
```

```
    }

    public void enemyAttack()
    {

        System.Random rnd = new System.Random();
        string selectedTag = enemyPawns[rnd.Next(0, enemyPawns.Count)].tag;
        pawnSelected = GameObject.FindGameObjectWithTag(selectedTag);
        int damage = 0;
        int tileToAttackIndex = Movement.GetComponent<Movement>().enemyAttack(pawnSelected.GetComponent<MasterPawn>().currentTileIndex);
        if (tileToAttackIndex != -1)
        {
            foreach (GameObject pawn in allyPawns)
            {
                if (pawn.GetComponent<MasterPawn>().currentTileIndex == tileToAttackIndex)
                {
                    targetPawn = pawn;
                }
            }

            if (pawnSelected.tag.Contains("Fire"))
            {
                Vector3 spawnPos1 = pawnSelected.transform.position;
                Quaternion spawnRot = eFireVFX.transform.rotation;
                Vector3 spawnPos2 = targetPawn.transform.position;
```

```
    Instantiate(eFireVFX, spawnPos1, spawnRot, board.transform);
    Instantiate(eFireVFX, spawnPos2, spawnRot, board.transform);
    damage = eFireBall.GetComponent<EnemyFireBall>().damage;
}
if (pawnSelected.tag.Contains("Air"))
{
    Vector3 spawnPos1 = pawnSelected.transform.position;
    Quaternion spawnRot = eAirVFX.transform.rotation;
    Vector3 spawnPos2 = targetPawn.transform.position;
    Instantiate(eAirVFX, spawnPos1, spawnRot, board.transform);
    Instantiate(eAirVFX, spawnPos2, spawnRot, board.transform);
    damage = eAirBullet.GetComponent<EnemyAirBullet>().damage;
}
if (pawnSelected.tag.Contains("Rock"))
{
    Vector3 spawnPos1 = pawnSelected.transform.position;
    Quaternion spawnRot =eRockVFX.transform.rotation;
    Vector3 spawnPos2 = targetPawn.transform.position;
    Instantiate(eRockVFX, spawnPos1, spawnRot, board.transform);
    Instantiate(eRockVFX, spawnPos2, spawnRot, board.transform);
    damage = eMeteor.GetComponent<EnemyMeteor>().damage;
}
if (pawnSelected.tag.Contains("Plant"))
{
    Vector3 spawnPos1 = pawnSelected.transform.position;
    Quaternion spawnRot = ePlantVFX.transform.rotation;
    Vector3 spawnPos2 = targetPawn.transform.position;
    Instantiate(ePlantVFX, spawnPos1, spawnRot, board.transform);
    Instantiate(ePlantVFX, spawnPos2, spawnRot, board.transform);
    damage = eSeedStrike.GetComponent<EnemySeedStrike>().damage;
}
if (pawnSelected.tag.Contains("Water"))
{
```

```
            Vector3 spawnPos1 = pawnSelected.transform.position;
            Quaternion spawnRot =eWaterVFX.transform.rotation;
            Vector3 spawnPos2 = targetPawn.transform.position;
            Instantiate(eWaterVFX, spawnPos1, spawnRot, board.transform);
            Instantiate(eWaterVFX, spawnPos2, spawnRot, board.transform);
            damage = eWaterCannon.GetComponent<EnemyWaterCannon>().damage;
        }
        if (pawnSelected.tag.Contains("King"))
        {
            Vector3 spawnPos1 = pawnSelected.transform.position;
            Quaternion spawnRot = eKingVFX.transform.rotation;
            Vector3 spawnPos2 = targetPawn.transform.position;
            Instantiate(eKingVFX, spawnPos1, spawnRot, board.transform);
            Instantiate(eKingVFX, spawnPos2, spawnRot, board.transform);
            damage = eKingVFX.GetComponent<EnemyKingAttack>().damage;

        }
        if (targetPawn.GetComponent<MasterPawn>().isGonnaDie(damage))
        {
            Movement.GetComponent<Movement>().updateTileAllyDestroyed(tileToAttackIndex);
            allyPawns.Remove(targetPawn);

        }

        Debug.Log(pawnSelected.tag + " Attacks Player's " + targetPawn.tag + " For " + damage + "
Damage ");
        targetPawn.GetComponent<MasterPawn>().takeDamage(damage);



    }
```

```
}
//Enemy Attacks


//Turn Methods
public void endPlayerTurn()
{

    uiManager.GetComponent<UiManager>().pawnSelector.SetActive(false);
    foreach (GameObject spellbar in uiManager.GetComponent<UiManager>().spellBars)
    {
        spellbar.SetActive(false);
    }
    turnManager.GetComponent<TurnManager>().endTurn();
    uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(false);
    GameObject[] garbageVFX = GameObject.FindGameObjectsWithTag("eAttackVFX");
    foreach (GameObject vfx in garbageVFX)
    {
        Destroy(vfx);
    }

    executeEnemyTurn();


}


public void executeEnemyTurn()
{
    moveEnemyPawn();
    newTurn();
```

```
}

IEnumerator executeAttack() {
    yield return new WaitForSeconds(2.6f);
    movePawn = false;
    enemyAttack();
}
IEnumerator playeMovementSafety()
{
    Debug.Log("Disabling bars");
    uiManager.GetComponent<UiManager>().pawnSelector.SetActive(false);
    uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(false);
     yield return new WaitForSeconds(2.1f);
    uiManager.GetComponent<UiManager>().pawnSelector.SetActive(true);
    uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(true);

}

IEnumerator playerDelay()
{
    yield return new WaitForSeconds(2.5f);
    Debug.Log("Player Turn");
    uiManager.GetComponent<UiManager>().pawnSelector.SetActive(true);
    turnManager.GetComponent<TurnManager>().newTurn();
    uiManager.GetComponent<UiManager>().endTurnBtn.SetActive(true);



}


public void newTurn()
{
    StartCoroutine(playerDelay());
```

```
    GameObject[] garbageVFX = GameObject.FindGameObjectsWithTag("AttackVFX");
    foreach (GameObject vfx in garbageVFX) {
        Destroy(vfx);
    }




}

private IEnumerator loadNextSceneWin() {

    yield return new WaitForSeconds(3.5f);
    SceneManager.LoadScene("YouWinScene");

}
private IEnumerator loadNextSceneLose()
{

    yield return new WaitForSeconds(3.5f);
    SceneManager.LoadScene("YouLoseScene");

}


// Start is called before the first frame update
void Start()
{

}

private void Awake()
```

```
{
    if (instance == null) { instance = this; }

}
// Update is called once per frame
void Update()
{
    if (movePawn)
    {

        currentLerpTime += Time.deltaTime;
        if (currentLerpTime >= lerptime)
        {
            currentLerpTime = lerptime;
        }
        float percentageTimes = currentLerpTime / lerptime;
        pawnSelected.transform.position = Vector3.Lerp(posA, posB, percentageTimes);




    }


    if (enemyPawns.Count == 0 && gameStarted)
    {
        win = true;
    }
    if (allyPawns.Count <= 3 && gameStarted)
    {
        lose = true;
    }
    if (kingPawn == null && gameStarted)
    {
```

```
            lose = true;
        }
        if (win)
        {
            StartCoroutine(loadNextSceneWin());

        }
        if (lose) {
            StartCoroutine(loadNextSceneLose());
        }

    }
}
```