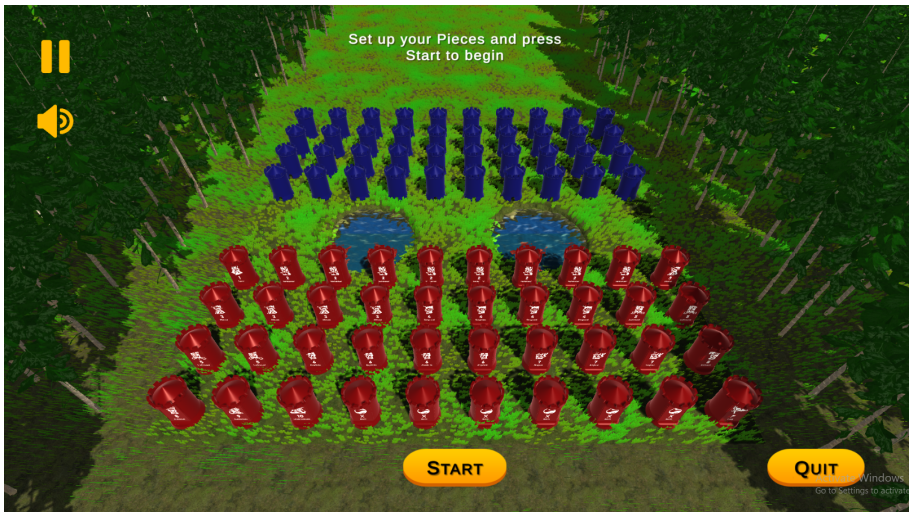
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ STRATEGO ΜΕ UNITY 3D»



Του φοιτητή
Παπαδόπουλος Κωνσταντίνος
Αρ. Μητρώου: 164821

Επιβλέπων
Δρ.Κυριάκος Τσιακμάκης

Ημερομηνία: 01-09-2023

Τίτλος Δ.Ε. Ανάπτυξη παιχνιδιού Stratego με Unity 3D
Κωδικός Δ.Ε. 23199
Ονοματεπώνυμο φοιτητή: Παπαδόπουλος Κωνσταντίνος
Ονοματεπώνυμο εισηγητή: Δρ. Κυριάκος Τσιακμάκης
Ημερομηνία ανάληψης: Δ.Ε. 30-3-2023
Ημερομηνία περάτωσης: Δ.Ε. 01-09-2023

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.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Η παρούσα πτυχιακή εργασία επιλέχθηκε διότι από πολύ μικρή ηλικία έπαιζα με βιντεοπαιχνίδια και πάντα ήθελα να δημιουργήσω ένα δικό μου. Το Game Development είναι ένας περίπλοκος και πολύ γρήγορα εξελισσόμενος κόσμος , όπου απαιτεί πολλές δεξιότητες όπως το προγραμματισμό, το design, τον ήχο, τον δοκιμαστή(tester), τον level designer κ.α. Αλλά με την βοήθεια των σύγχρονων μηχανών παιχνιδιών (game engines) η δημιουργία νέων παιχνιδιών έχει φτάσει σε νέα επίπεδα ευκολίας και αποδοτικότητας. Για την υλοποίηση της εργασίας αυτής επιλέχθηκε το εργαλείο Unity διότι η χρήση του εργαλείου αυτού είναι σχετικά απλή και δεν χρειάζεται να γίνει όλη η δουλεία προγραμματιστικά, αλλά υπάρχουν κατάλληλα εργαλεία για την διευκόλυνση και την εξοικονόμηση χρόνου του gamer developer.

Περίληψη

Στην παρούσα πτυχιακή εργασία θα παρουσιαστεί η διαδικασία ανάπτυξης ενός 3D game με χρήση της μηχανής παιχνιδιού Unity3D. Αρχικά θα γίνει μία εισαγωγή στα βιντεοπαιχνίδια, όπου θα δούμε ποιοι είναι οι διαφορετικοί τύποι του gaming αλλά και ποια είναι τα θετικά και αρνητικά του στοιχεία. Θα δούμε πως αυτά τα στοιχεία μπορούν να επηρεάσουν την καθημερινότητα μας ως προς το καλό αλλά και ως προς το κακό, ανάλογα με την χρήση τους. Στην συνέχεια θα αναφερθούμε λεπτομερώς στην δημοφιλέστερη μηχανή ανάπτυξης παιχνιδιών(Game Engine), όπου αυτή είναι η Unity. Αρχικά θα δούμε τι είναι το Unity IDE και θα μάθουμε τα βασικά χαρακτηριστικά του Interface της Unity καθώς και πως λειτουργεί το Interface αυτό. Επίσης θα μάθουμε και κάποιες από τις πιο βασικές έννοιες της Unity, όπου θα μας χρησιμεύσουν για την κατασκευή του δικού μας παιχνιδιού όπου είναι το Stratego 3D. Πριν ξεκινήσουμε να φτιάξουμε το παιχνίδι μας, θα μάθουμε τι είναι το Stratego, ποιοι είναι οι κανόνες του και πως συμπεριφέρονται τα πιόνια του. Αφού κατανοήσουμε όλα τα παραπάνω θα ξεκινήσουμε να φτιάχνουμε το παιχνίδι μας. Στην αρχή θα ξεκινήσουμε με το πως δημιουργούμε ένα καινούργιο πρότζεκτ και πως το επεξεργαζομαστε. Αφού φτιάξουμε το πρότζεκτ θα ξεκινήσουμε να δημιουργούμε το πιο βασικό πράγμα του παιχνιδιού, που είναι το Board. Στην συνέχεια θα δούμε πως φτιάξαμε το περιβάλλον του παιχνιδιού(τον χάρτη) και πως αλληλεπιδρά ο χρήστης με το board. Αμέσως μετά θα υλοποιήσουμε τα πιόνια και όλους τους κανόνες του παιχνιδιού. Τέλος θα αναλύσουμε τον τρόπο με τον οποίο δημιουργήσαμε το menu και το interface μας.

«Development of Stratego 3D game with Unity3D»

«Papadopoulos Konstantinos 164821»

Abstract

In this thesis, the process of developing a 3D game using the Unity3D game engine will be presented. First, an introduction to video games will be provided, where we will explore the different types of gaming and their positive and negative aspects. We will see how these elements can influence our daily lives for better or worse. Next we will focus on the most popular game development engine, which is Unity. We will learn about the Unity Integrated Development Environment (IDE) and its basic interface features and functionalities. Additionally, we will grasp some of the fundamental concepts of Unity that will be useful for building our own game, which is Stratego 3D. Before starting to create our game, we will learn about Stratego, its rules and how its pieces behave. Once we understand the basics of the game, we will begin creating our game. We will start by creating a new project and setting it up. After setting up the project, we will work on building the core element of the game, the board. Then, we will focus on creating the map of the game and the way the user will interact with the board. Immediately afterward, we will implement the game pieces and the rules of the game. Finally, we will take a look at how we developed our menu and interface.

Ευχαριστίες

Σε αυτήν την ενότητα ο φοιτητής/ φοιτήτρια προαιρετικά μπορεί να ευχαριστήσει όσους αισθάνεται ότι συνέβαλαν (επιστημονικά, ηθικά, οικονομικά κτλ) στην ολοκλήρωση της διπλωματικής εργασίας.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Κεφάλαιο 1ο: Εισαγωγή στα βιντεοπαιχνίδια

1.1 Τι είναι τα Βιντεοπαιχνίδια

Τα video games είναι παιχνίδια που περιλαμβάνουν αλληλεπίδραση με μία διεπαφή χρήστη ή με μία συσκευή εισόδου όπως είναι το Joystick, ένα χειριστήριο, ένα ποντίκι, ένα πληκτρολόγιο ή μία συσκευή ανίχνευσης κίνησης. Τα βιντεοπαιχνίδια ορίζονται με βάση την πλατφόρμα τους και χωρίζονται σε arcade games, παιχνίδια κονσόλας και παιχνίδια υπολογιστή. Πιο πρόσφατα επεκτάθηκαν και τα φορητά παιχνίδια μέσω φορητών συσκευών όπως είναι τα smartphones, τα tablets, τα συστήματα εικονικής επαυξημένης πραγματικότητας κ.α. Τα παιχνίδια αυτά ταξινομούνται σε ένα ευρύ φάσμα ειδών ανάλογα τον τύπο παιχνιδιού και τον σκοπό τους. Κάποια από τα είδοι είναι τα παιχνίδια δράσης (action games) , τα Multiplayer Online Battle Area (MOBA) , τα sport games , τα παιχνίδια στρατηγικής(strategy games) , τα simulation games, τα Massively Multiplayer Online Role Playing Games (MMORPGs) , τα shooters games (FPS - TPS), τα Puzzlers and Party games κ.α. [1]

1.2 Οφέλη των παιχνιδιών

Οι πιο ορατές αξίες των υπολογιστών και των βιντεοπαιχνιδιών είναι οι καλλιτεχνικές και οι ψυχαγωγικές αξίες του. Τα σύγχρονα βιντεοπαιχνίδια περιέχουν έναν εξαιρετικά μοναδικό συνδυασμό τρισδιάστατης τέχνης , εφέ CG , VFX, αρχιτεκτονικής, τεχνητής νοημοσύνης, επικής μουσικής, δραματικών παραστάσεων, αφήγησης ιστοριών και κυρίως διαδραστικότητας με τον παίκτη.[2] Αυτή η διαδραστικότητα δίνει στον παίκτη την δυνατότητα να έρθει σε επαφή με την απεικόνιση κάποιου τρισδιάστατου (πολλές φορές και δισδιάστατου) περιβάλλοντος και με τις ενέργειες του παίκτη να λειτουργεί πάνω στο περιβάλλον αυτό. Ακόμα και αν το παιχνίδι έχει περισσότερο σενάριο - ανάπτυξη ιστορίας , πάλι το παιχνίδι αυτό μπορεί να αισθάνεται σαν μια μεγάλη ποσότητα ελευθερίας για το άτομο που παίζει το παιχνίδι. Δίνει στον χρήστη μία αίσθηση αποφυγής από την καθημερινότητα του.

Τα βιντεοπαιχνίδια βοηθούν στην ανάπτυξη αντανακλαστικών του ανθρώπου αλλά και στην πιο σωστή και γρήγορη λήψη αποφάσεων στην καθημερινότητα του. Επιστήμονες από το Πανεπιστήμιο του Ρότσεστερ αναφέρουν στο περιοδικό Current Biology ότι τα βιντεοπαιχνίδια θα μπορούσαν να παρέχουν ένα ισχυρό πρόγραμμα εκπαίδευσης για την επιτάχυνση των αντιδράσεων σε πραγματικές καταστάσεις. Οι ερευνητές εξέτασαν δεκάδες νέους ηλικίας 18 έως 25 ετών που δεν έπαιζαν προηγουμένως βιντεοπαιχνίδια. Χώρισαν τα παιδιά σε δύο ομάδες όπου η μία ομάδα έπαιξε 50 ώρες από το βιντεοπαιχνίδι Call Of Duty 2 και Unreal Tournament ενώ η άλλη ομάδα έπαιξε 50 ώρες από το αργό παιχνίδι στρατηγικής The Sims 2. Μετά από την περίοδο εκπαίδευσης ζητήθηκε από όλα τα παιδιά να πάρουν γρήγορες αποφάσεις σε διάφορες εργασίες που σχεδιάστηκαν από τους ερευνητές. Στις εργασίες οι συμμετέχοντες έπρεπε να κοιτάξουν μια οθόνη, να αναλύσουν τι συνέβαινε και να απαντήσουν σε μία απλή ερώτηση σχετικά με τη δράση σε όσο τον δυνατόν

λιγότερο χρόνο. Οι παίκτες του παιχνιδιού δράσης ήταν έως και 25% πιο γρήγοροι στο να καταλήξουν σε ένα συμπέρασμα και απάντησαν σωστά στον ίδιο αριθμό απαντήσεων με τα παιδιά που έπαιζαν το παιχνίδι στρατηγικής.[3]

Σχήμα 1.1: Why playing video games is good for your brain[4]

1.3 Πόσο δημοφιλές είναι το Gaming

Το gaming είναι μία εξαιρετικά δημοφιλής δραστηριότητα σε όλο τον κόσμο. Ο αριθμός των ανθρώπων που παίζουν παιχνίδια παγκοσμίως είναι πάνω από 3 δισεκατομμύρια άτομα και συνεχώς αυξάνεται. Ορισμένες εκτιμήσεις υπολογίζουν την παγκόσμια αξία αγοράς παιχνιδιών μεταξύ 180 και 220 δισεκατομμυρίων δολαρίων το 2022. Αυτό το ποσό είναι περίπου το διπλάσιο της αγοράς για την κινηματογραφική βιομηχανία. Ένας μεγάλος αριθμός των ανθρώπων που παίζουν παιχνίδια είναι αυτοί που παίζουν στον υπολογιστή τους, αλλά η πλειοψηφία της αγοράς παιχνιδιών είναι από ανθρώπους που παίζουν παιχνίδια στα smartphone τους. [5]

1.4 Τύποι του Gaming

Το gaming χωρίζεται σε τρεις κύριες κατηγορίες, στο casual gaming, στο hardcore και στο professional gaming. Το casual gaming είναι όταν κάποιος παίζει παιχνίδια σπάνια ή παίζει παιχνίδια χαμηλής έντασης. Δηλαδή είναι κάποιος που παίζει για να περάσει την ώρα του. Ένα παράδειγμα casual gamer θα ήταν κάποιος που παίζει παιχνίδια στο κινητό του ενώ βρίσκεται μέσα σε κάποιο μέσω μαζικής μεταφοράς ή στο σπίτι του ενώ παρακολουθεί τηλεόραση. Το casual gaming είναι το μεγαλύτερο μέρος αγοράς των παιχνιδιών λόγω της εύκολης πρόσβαση που παρέχουν τα smartphone.[5]

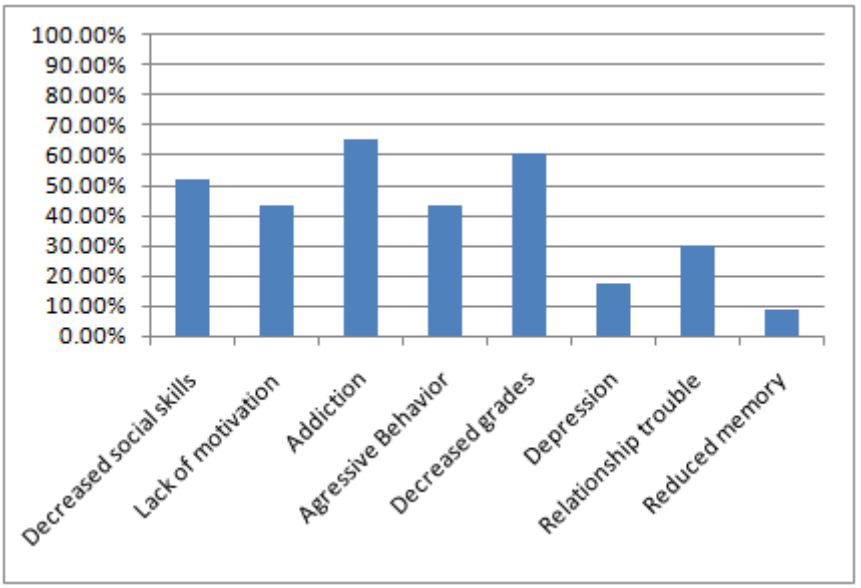
Ένας hardcore παίκτης είναι κάποιος που παίζει τακτικά βιντεοπαιχνίδια. Συνήθως κατέχουν συσκευές βιντεοπαιχνιδιών όπως κονσόλες ή gaming υπολογιστές. Ένας hardcore παίκτης μπορεί να παίζει πολλά διαφορετικά παιχνίδια αλλά εστιάζει κυρίως σε ένα παιχνίδι ή σε ένα είδος παιχνιδιού που του αρέσει ιδιαίτερα.[5] Επίσης ένας τέτοιος παίκτης αφιερώνει αρκετές ώρες την ημέρα (πάνω από 2-3) στο κύριο παιχνίδι του και τις περισσότερες φορές εντάσσεται σε gaming communities όπου και συνήθως είναι αρκετά ενεργώς σε καθημερινή βάση.

Τέλος professional παίκτης είναι κάποιος που βγάζει χρήματα παίζοντας παιχνίδια. Υπάρχουν διάφοροι τρόποι για να βγάλει χρήματα κάποιος μέσα από το gaming. Θα μπορούσε κάποιος να γίνει professional player για κάποια ομάδα και να συμμετέχει σε διάφορα τουρνουά, όπου ο παίκτης πληρώνεται και από τα συμβόλαια με την ομάδα και από τις νίκες που θα καταφέρει στα τουρνουά(θα μπορούσε κανείς να τους συγκρίνει με τους αθλητές π.χ. ποδοσφαιριστές). Ένας άλλος τρόπος είναι να γίνει κάποιος παίκτης streamer. Δηλαδή να μεταδίδει τις ώρες που θα παίζει βιντεοπαιχνίδια online, και να πληρώνεται από τους viewer τους είτε από subscribes, είτε από donations, είτε και από μεγάλους sponsors όπως είναι η red bull και η monster. Οι professionals παίκτες συνήθως εστιάζουν σε ένα μόνο παιχνίδι διότι προσπαθεί να γίνει ο καλύτερος σε αυτό το παιχνίδι, είτε γιατί παίζει σε μεγάλα τουρνουά είτε για να μαζέψει πιο εύκολα viewers στο stream του. Οι μεγαλύτερες πλατφόρμες για τους streamers είναι το twitch, το youtube και το facebook.

1.5 Αρνητικά του Gaming

Πέρα από όλα τα θετικά που μας παρέχουν τα βιντεοπαιχνίδια, υπάρχει και η αρνητική πλευρά του gaming. Ένα πολύ γνωστό και καθημερινό πρόβλημα του είναι ότι μπορεί να κάνει κάποιους ανθρώπους πιο βίαιους.[7] Πολλές φορές αν δεν υπάρχει αυτό-συγκράτηση μπορεί να μειώσει την συγκέντρωση των παικτών στα σημαντικότερα θέματα στην καθημερινή τους ζωή. Δηλαδή να παραμελήσουν σημαντικά θέματα όπως το σχολείο ή την δουλειά για να αφιερώσουν περισσότερο χρόνο στα παιχνίδια. Πολλές φορές μπορεί κάποιος και να εθιστει σε κάποιο βιντεοπαιχνίδι. Υπάρχουν κέντρα απεξάρτησης για βιντεοπαιχνίδια σε διάφορες χώρες του εξωτερικού όπως το Λονδινο [6]. Ένα άλλο αρνητικό είναι ότι πολλοί παίκτες αναφέρουν ότι έχουν αυξημένα συμπτώματα άγχους και κατάθλιψης μετά από πολλές ώρες σε κάποιο παιχνίδι. Τέλος μπορεί να υπάρξει έλλειψη κινήτρων για ενασχόληση με δραστηριότητες εκτός από τα βιντεοπαιχνίδια. Τα βιντεοπαιχνίδια είναι μια υπερδιεγερτική δραστηριότητα που μπορεί να είναι πιο ελκυστική από άλλες δραστηριότητες που μπορείτε να βρείτε στον φυσικό κόσμο. Όταν παίζουμε

βιντεοπαιχνίδια ο εγκέφαλος μας αρχίζει να παράγει σημαντικές ποσότητες ντοπαμίνης που προκαλούν το αίσθημα ευχαρίστησης. Έτσι όταν παίζουμε για παρατεταμένες περιόδους, ο ανθρώπινος εγκέφαλος μπορεί να συνηθίσει την βιασύνη της ντοπαμίνης και τα υψηλά επίπεδα διέγερσης. Όταν το παιχνίδι γίνεται η μόνη δραστηριότητα που ο εγκέφαλος μας βρίσκει ευχάριστη, η έλλειψη κινήτρων για συμμετοχή σε άλλες δραστηριότητες είναι φυσικό αποτέλεσμα.[8].



Σχήμα 1.2: Αρνητικά του Gaming[9]

1.6 Συμπέρασμα

Απ Όσο είδαμε υπάρχουν πολλά θετικά αλλά και αρνητικά πράγματα όσον αφορά τα βιντεοπαιχνίδια. Γι Αυτό τον λόγο θα πρέπει να κατανοήσουμε πολύ καλά τους κινδύνους του και να θέτουμε σοβαρά όρια όσον αφορά για το πόσες ώρες θα πρέπει να αφιερώνουμε στα βιντεοπαιχνίδια και για το ποιες είναι οι προτεραιότητες μας στην καθημερινή μας ζωή. Από την άλλη πάλι θα πρέπει να καταλάβουμε ότι υπάρχουν και πολλά θετικά που μπορούμε να κερδίσουμε από το gaming ενώ περνάμε τον χρόνο μας ευχάριστα και ψυχαγωγούμαστε. Θα πρέπει δηλαδή να υπάρχει μία σωστή ισορροπία και πειθαρχία.

Κεφάλαιο 2ο: Unity 3D engine

2.1 Εισαγωγή

Ως μηχανή ανάπτυξης παιχνιδιών (Game Engine) θα χρησιμοποιήσουμε το Unity3D όπου αυτή τη στιγμή είναι η πιο δημοφιλής στον κόσμο.Εμείς θα χρησιμοποιήσουμε την δωρεάν έκδοση της, η οποία μπορεί να την κατεβάσει ο καθένας μας. Σε περίπτωση που κάποιος θα ήθελε να χρησιμοποιήσει περισσότερες λειτουργίες της Unity, μπορεί να αγοράσει κάποια από τα premium πακέτα που προσφέρονται από αυτήν. Με τη μηχανή ανάπτυξης παιχνιδιών της Unity τα πράγματα απλοποιούνται πάρα πολύ. Η Unity κάνει την περισσότερη δουλειά στο παρασκήνιο, εκτελώντας τις περισσότερες εργασίες από μόνο της. Για αυτό τον λόγο είναι ένα πολύ χρήσιμο εργαλείο όπου μπορεί να βοηθήσει τον χρήστη στην δημιουργία μεγάλων project χωρίς την ανάγκη περισσότερων ατόμων αλλά με μόνο την βοήθεια της πλατφόρμας αυτής. Η Unity χρησιμοποιείται συνήθως για τη δημιουργία παιχνιδιών είτε για κονσόλες, είτε για κινητά ή και υπολογιστές. Επίσης δίνει την δυνατότητα στους χρήστες να δημιουργήσουν 3D, 2D, εικονικής πραγματικότητας(Virtual Reality-VR) και παιχνιδιών επαυξημένης πραγματικότητας(Augmented Reality-AR). Στην ιστοσελίδα της Unity υπάρχει και η κατηγορία “Learning” όπου εκεί μπορεί ο χρήστης να βρει πολλά εκπαιδευτικά άρθρα και βίντεο για το πως λειτουργεί η πλατφόρμα και ποιες είναι οι λειτουργίες της.

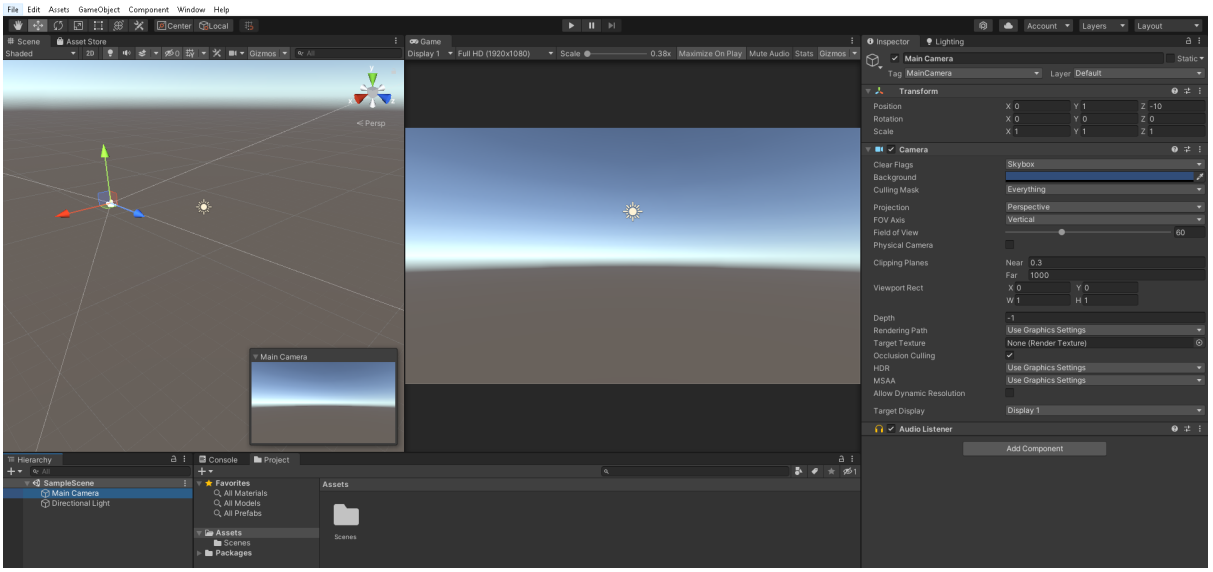
Η Unity σαν μηχανή παιχνιδιών είναι σε θέση να παρέχει πολλές από τις πιο σημαντικές ενσωματωμένες λειτουργίες που κάνουν ένα παιχνίδι να λειτουργεί. Όπως για παράδειγμα η φυσική που υπάρχει μέσα στο παιχνίδι, οι συγκρούσεις που μπορεί να γίνουν με τον παίκτη αλλά και με το περιβάλλον, το 3D rendering κ.α. Με άλλα λόγια ο προγραμματιστής - χρήστης δεν χρειάζεται να δημιουργήσει μία μηχανή φυσικής από την αρχή, υπολογίζοντας κάθε υλικό, τον τρόπο με τον οποίο θα αλληλεπιδρά το φως πάνω τους και θα αναπηδά σε διάφορες επιφάνειες ή κάθε κίνηση των υλικών αυτών κ.α Αυτό που κάνει το Unity ακόμα πιο ισχυρό είναι ότι υπάρχει το Unity Asset Store. Το Unity Asset Store είναι ένα μέρος όπου οι προγραμματιστές - χρήστες μπορούν να ανεβάσουν τις δημιουργίες τους και.να τις κάνουν διαθέσιμες προς άλλους χρήστες στην κοινότητα, ή και να κατεβάσουν δημιουργίες άλλων χρηστών για δική τους χρήση. Αρκετά από τα Asset αυτά χρειάζονται πληρωμή για να τα αποκτήσει ο χρήστης αλλά υπάρχουν και πολλά Asset που προσφέρονται δωρεάν και πολλές φορές γλιτώνουν αρκετό χρόνο από τον προγραμματιστή. Αυτό έχει ως αποτέλεσμα ο προγραμματιστής να είναι ελεύθερος να επικεντρωθεί στην αρχική του ιδέα, σε αυτό που έχει περισσότερο σημασία χωρίς να χρειάζεται να δημιουργήσει και τις λιγότερο σημαντικές λεπτομέρειες(π.χ. ένα 3D μοντέλο σαν διακοσμητικό για το περιβάλλον του) χάνοντας έτσι πολύ χρόνο[10].

2.2 Τι είναι το Unity IDE

Εκτός από μηχανή παιχνιδιών το Unity είναι και ένα IDE. Το IDE ή αλλιώς “Integrated Development Environment” σημαίνει “Ενσωματωμένο Περιβάλλον Ανάπτυξης” , το οποίο περιγράφει μια διεπαφή που μας δίνει πρόσβαση σε όλα τα εργαλεία που χρειαζόμαστε για την ανάπτυξη ενός περιβάλλοντος. Το Unity διαθέτει ένα οπτικό πρόγραμμα επεξεργασίας που επιτρέπει στους χρήστες του απλώς με drag and drop να τοποθετήσουν στοιχεία σε σκηνές και στη συνέχεια να χειριστούν τις ιδιότητες τους είτε προγραμματιστικά(μέσω των scripts) είτε πάλι μέσα από την διεπαφή. Το Unity χρησιμοποιεί C# σαν γλώσσα προγραμματισμού για την δημιουργία των scripts. Το θετικό είναι ότι μπορεί κάποιος να υλοποιήσει πολλά μέσω του Unity χωρίς να χρειάζεται να χειριστεί πολύ κώδικα. Επίσης η C# είναι μία από τις πιο φιλικές γλώσσες προγραμματισμού για αρχάριους και έχει πολλά κοινά με άλλες γλώσσες όπως η C και η Java[10].

2.3 Unity Interface

Όταν ανοίγει το Unity για πρώτη φορά μπορεί να φανεί το interface λιγάκι χαοτικό με το πόσα πολλά παράθυρα υπάρχουν και με όλα τα εικονίδια και τις επιλογές τους. Αλλά τα πράγματα είναι πιο απλά από ότι φαίνονται αρχικά. Παρακάτω θα δούμε τα βασικά παράθυρα καθώς και τις ιδιότητες τους.

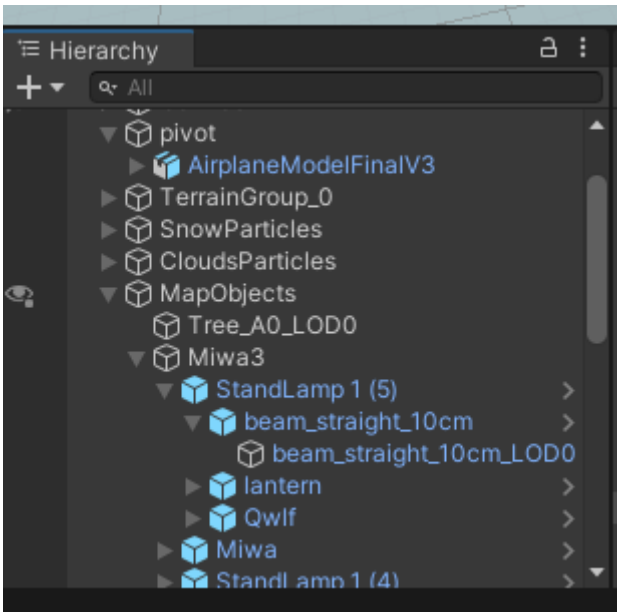


Σχήμα 2.1: Περιβάλλον του Unity

2.3.1 Hierarchy

Το πρώτο παράθυρο που παρατηρούμε όταν ανοίγουμε το Unity είναι το παράθυρο Hierarchy. Συνήθως βρίσκεται πάνω αριστερά αλλά στην φωτογραφία από πάνω(Σχήμα) το βλέπουμε

να είναι κάτω αριστερά. Αυτό συμβαίνει επειδή το άλλαξα εγώ με drag and drop επειδή με βολεύει πιο πολύ(προσωπική προτίμηση). Στο παράθυρο αυτό υπάρχει μία λίστα με όλα τα GameObjects της σκηνής μας. Μέσα από την λίστα αυτή μπορούμε να εντοπίσουμε με μεγαλύτερη ευκολία και να επιλέξουμε γρήγορα οποιαδήποτε πτυχή του παιχνιδιού μας για να αλλάξουμε τις ιδιότητες του. Όταν ο χρήστης δημιουργήσει νέα αντικείμενα στην σκηνή τότε αυτό το αντικείμενο εμφανίζεται και στο παράθυρο Hierarchy. Επίσης υπάρχει η δυνατότητα να θέσει ο χρήστης γονείς και παιδιά αντικειμένων. Για να θέσουμε κάποιο αντικείμενο ως παιδί ενός άλλου απλώς σέρνουμε το αντικείμενο που θέλουμε σαν παιδί πάνω στο αντικείμενο που θέλουμε ως γονέα[11].



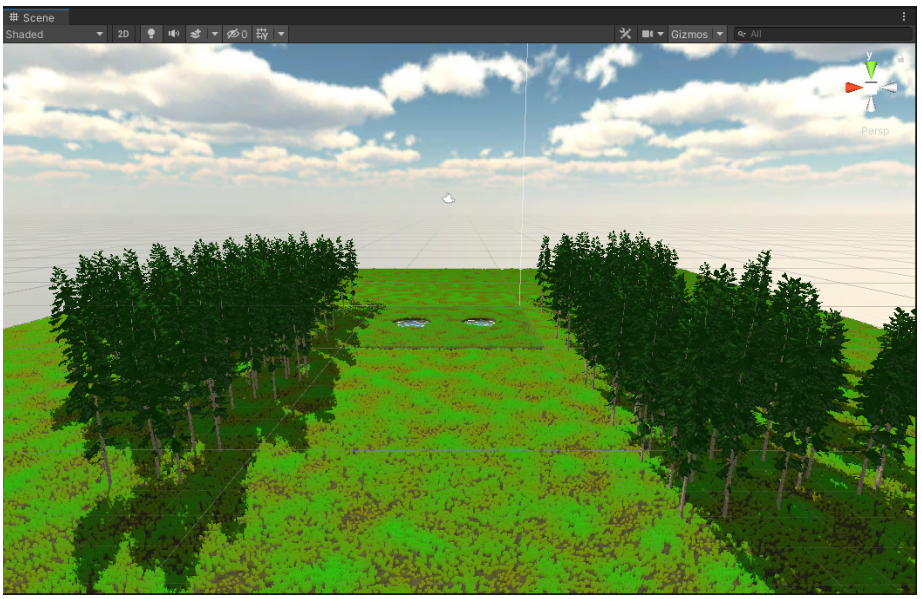
Σχήμα 2.2: Παράθυρο Hierarchy

Στο παραπάνω Σχήμα 3 μπορούμε να δούμε ότι το GameObject Pivot είναι πατέρας του AirplaneModelFinalV3 gameobject. Μπορεί να υπάρχουν περισσότεροι από 1 βαθμό γονέων-παιδιών όπως φαίνεται και στο σχήμα.

2.3.2 Scene

Το μεγαλύτερο παράθυρο που συνήθως βρίσκεται στην μέση του λογισμικού Unity και είναι αυτό του Scene. Μέσα σε αυτό το παράθυρο όπως αναφέρεται και στο όνομα του, ο χρήστης μπορεί να δει την σκηνή του παιχνιδιού του. Στην σκηνή φαίνονται όλα τα GameObjects που έχει προσθέσει ο χρήστης. Ο χρήστης μπορεί σε αυτό το παράθυρο να
επιλέξει κάποιο από τα GameObjects και να το μετακινήσει πάνω στην σκηνή,ή να το γυρίσει(rotate) ως προς τους τρεις άξονες X,Y,Z. Τα εικονίδια που βρίσκονται πάνω αριστερά στο Unity(Σχήμα 4) αλλάζουν τον τρόπο με τον οποίο αλληλεπιδρά ο χρήστης με τα αντικείμενα. Με το χέρι θα

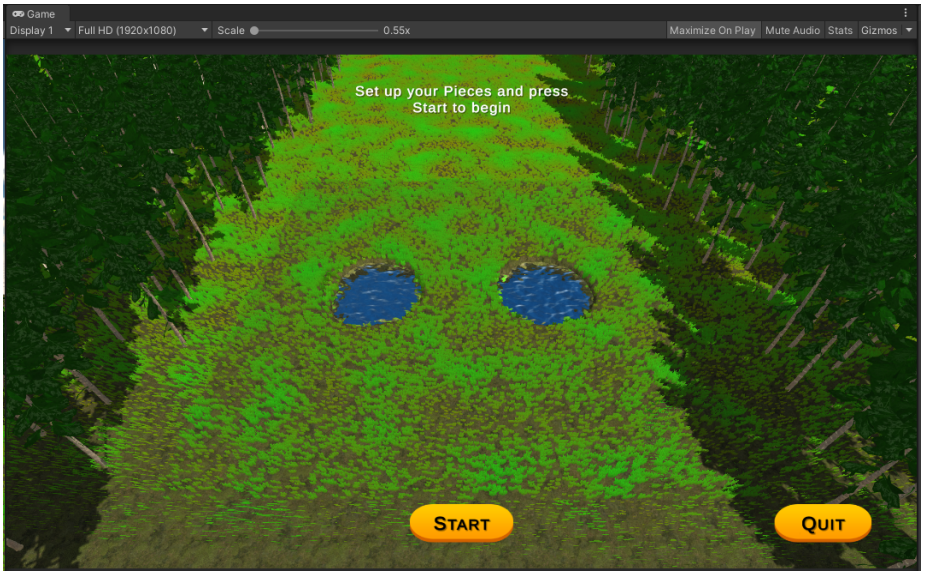
μπορέσουμε να σύρουμε την προβολή μας γύρω στην σκηνή, ενώ με τα βέλη για παράδειγμα θα μπορέσουμε να μετακινήσουμε τα αντικείμενα[12].



Σχήμα 2.3: Παράθυρο Scene

2.3.3 Game

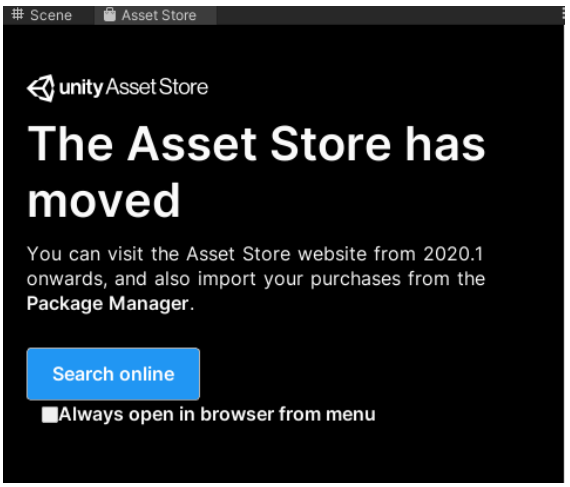
Αυτό το παράθυρο είναι συνήθως κρυμμένο πίσω από το παράθυρο Scene και μπορεί ο χρήστης να το προσπελάσει πατώντας αριστερό κλικ πάνω του. Σε αυτό το παράθυρο βλέπουμε το Game View του παιχνιδιού μας. Δηλαδή βλέπουμε πως φαίνεται το παιχνίδι μας χωρίς να μπορούμε να αλληλεπιδράσουμε με τα αντικείμενα. Η χρησιμότητα του είναι για να δοκιμάζει ο χρήστης είτε προγραμματιστικές αλλαγές είτε για να δει το τελικό αποτέλεσμα του παιχνιδιού του και να παίξει το παιχνίδι(test runs)[13].



Σχήμα 2.4: Παράθυρο Game

2.3.4 Asset Store

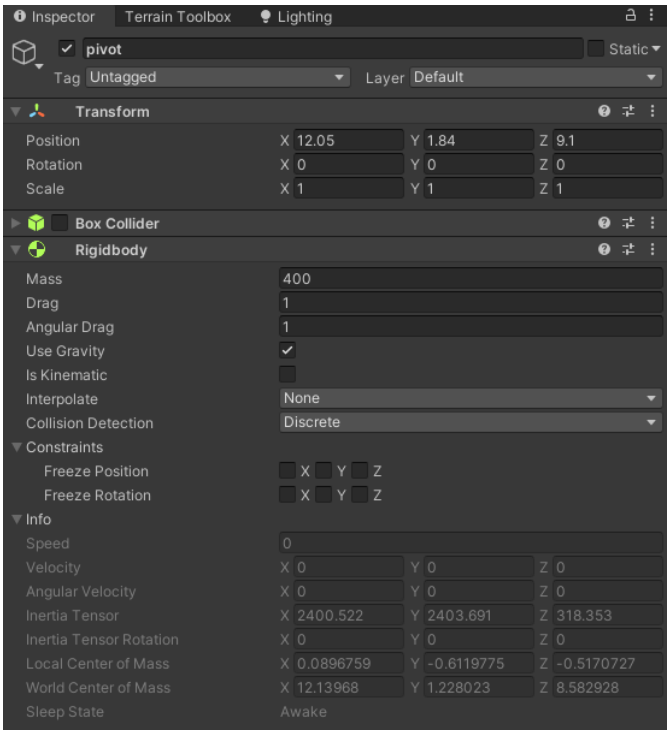
Το παράθυρο αυτό βρίσκεται συνήθως πίσω από το παράθυρο Scene. Αν όμως δεν υπάρχει εκεί, τότε μπορείτε να πάτε στο Window και θα το βρείτε εκεί. Το Asset Store όπως ήδη αναφέραμε είναι ένα μέρος όπου οι προγραμματιστές - χρήστες μπορούν να ανεβάσουν τις δημιουργίες τους και.να τις κάνουν διαθέσιμες προς άλλους χρήστες στην κοινότητα, ή και να κατεβάσουν δημιουργίες άλλων χρηστών για δική τους χρήση. Μπορείτε να κάνετε αναζήτηση στο search bar για να βρείτε το κατάλληλο asset για το project σας[14].



Σχήμα 2.5: Παράθυρο Asset Store

2.3.5 Inspector

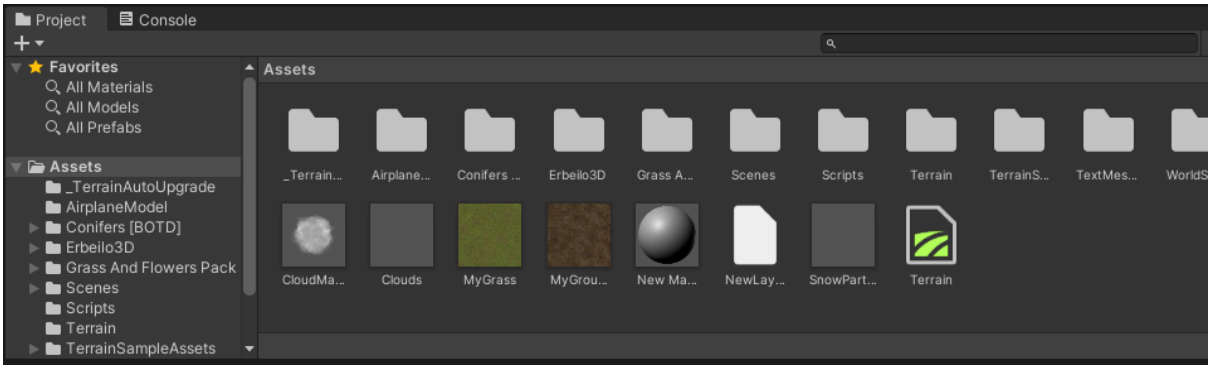
Ένα από τα πιο σημαντικά παράθυρα του λογισμικού του Unity είναι το Inspector. Αυτό το παράθυρο βρίσκεται στην πιο δεξιά πλευρά της διεπαφής του λογισμικου. Ο Inspector μας δίνει την δυνατότητα να επεξεργαστούμε τις ιδιότητες ενός επιλεγμένου GameObject. Μέσα από αυτόν μπορούμε να αλλάξουμε το μέγεθος ή την θέση ενός αντικειμένου. Επίσης μπορούμε να προσθέσουμε νέα "components" όπως είναι τα colliders , κάποιο C# script ή και ακόμη τον Rigidbody component για να δώσουμε φυσική στο αντικείμενο που θέλουμε[15].



Σχήμα 2.6: Παράθυρο Inspector

2.3.6 Project

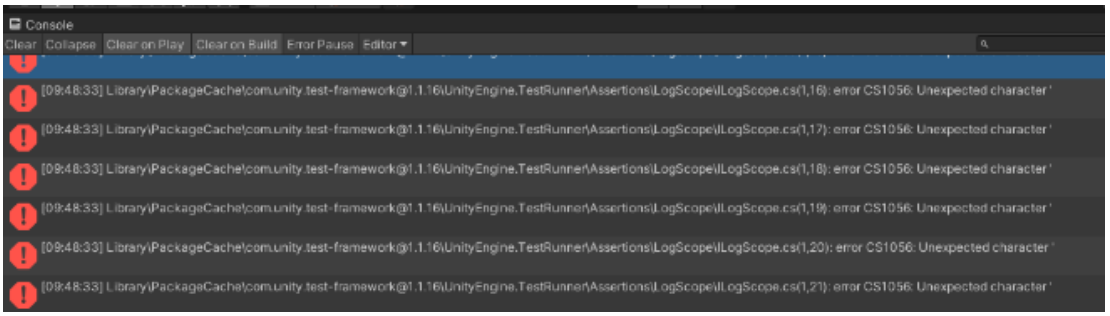
Το Project παράθυρο βρίσκεται στο κάτω μέρος της οθόνης μας και θα μας δείξει όλα τα αρχεία που απαρτίζουν το παιχνίδι μας. Ο χρήστης βρίσκει το αντικείμενο που θέλει μέσα σε αυτό το παράθυρο και το σέρνει στο Hierarchy ή στο Scene για να προστεθεί στην σκηνή του. Επίσης μπορεί να δημιουργήσει Script αρχεία και να τα φέρει με τον ίδιο τρόπο πάνω στα αντικείμενα που θέλει να ενεργήσει το αντίστοιχο Script[16].



Σχήμα 2.7: Παράθυρο Project

2.3.7 Console

Στο παράθυρο της κονσόλας μπορούμε να δούμε πληροφορίες από το ίδιο το Unity. Συνήθως αυτό θα μας ενημερώσει αν υπάρχει κάποιο πρόβλημα με τον κώδικα μας, αν υπάρχουν σφάλματα, warnings ή αν υπάρχουν προβλήματα που πρέπει να επιλυθούν με την ρύθμιση του ίδιου λογισμικού του Unity. Τα μηνύματα αυτά συνήθως μας δίνουν και επεξήγηση για το σφάλμα που υπάρχει αλλά και την σειρά με την οποία έγιναν τα λάθοι αν είναι περισσότερα από ένα[17].



Σχήμα 2.8: Παράθυρο Console

2.4 Έννοιες του Unity

Για να μπορέσει κάποιος να χρησιμοποιήσει το Unity για την παραγωγή κάποιου παιχνιδιού, θα πρέπει να έχει μία βασική ιδέα από τις πιο βασικές έννοιες του. Πολλές από αυτές τις έννοιες θα τις χρησιμοποιήσουμε και εμείς για την δημιουργία του δικού μας παιχνιδιού. Αφού κατανοήσουμε τις βασικές έννοιες του Unity τα πράγματα θα μας φανούν λιγότερο περίπλοκα, διότι θα αρχίσουμε να καταλαβαίνουμε σιγά σιγά την λογική που ακολουθεί το πρόγραμμα αλλά και χρησιμότητά του.

2.4.1 Game Object

Το Game Object είναι ίσως το πιο σημαντικό στοιχείο σε ένα παιχνίδι. Είναι όλα εκείνα τα αντικείμενα που υπάρχουν μέσα στην σκηνή του παιχνιδιού μας. Ένα Game Object μπορεί να είναι κάποιο 3D αντικείμενο που δημιουργήσαμε, η Main Camera μας ή οποιαδήποτε άλλη camera, ένα μοντέλο που κάναμε insert από κάποιο asset, ένα button, ένα text mesh, ένα εφέ (π.χ. particle system), ένα Light object , κ.α. Με άλλα λόγια τα Game Object αντιπροσωπεύουν χαρακτήρες, στηρίγματα και σκηνικά όπου δεν καταφέρνουν πολλά από μόνα τους, αλλά λειτουργούν ως δοχεία για τα Components τα οποία υλοποιούν την λειτουργικότητα πάνω σε αυτά[18].

2.4.2 Components

Όπως προαναφέραμε τα Components είναι δομικά στοιχεία των αντικειμένων, είναι δηλαδή τα μεμονωμένα κομμάτια που συνθέτουν την συμπεριφορά, την λειτουργικότητα, την εμφάνιση των αντικειμένων. Κάθε Component είναι μία αυτόνομη μονάδα που εκτελεί μια συγκεκριμένη εργασία, όπως για παράδειγμα ο Rigidbody Component όπου δίνει φυσική στο αντικείμενο, ο Collision Component όπου δίνει κάποιο είδος collision (mesh, box, circle κ.α.) στο αντικείμενο, ή κάποιος Audio Component όπου δίνει την δυνατότητα να προσθέσουμε ήχο σε κάποιο αντικείμενο. Το Unity παρέχει ένα ευρύ φάσμα ενσωματωμένων Components αλλά υπάρχει και η δυνατότητα δημιουργίας custom Components χρησιμοποιώντας τα Scripts C#. Με τον συνδυασμό δύο ή και περισσότερων Components μπορούμε να δημιουργήσουμε περίπλοκα αντικείμενα με ιδιαίτερους μηχανισμούς και ιδιότητες.[19]

2.4.3 Transform

Το Transform είναι ένας πολύ γνωστός και σημαντικός Component που καθορίζει το position, το rotation και το scale του αντικειμένου. Δηλαδή καθορίζει την θέση, την περιστροφή και την κλίμακα του αντικειμένου πάνω στον κόσμο μας. Οι ιδιότητες Position και Scale αντιπροσωπεύονται από τύπους δεδομένων Vector3, ο οποίος καθορίζει τις συντεταγμένες x, y και z του αντικειμένου. Ενώ η ιδιότητα Rotation αντιπροσωπεύεται από τύπο Quaternion ο οποίος περιγράφει την περιστροφή του αντικειμένου γύρω από τους αξονες x, y, και z. Το Component αυτό χρησιμοποιείται συνήθως για την κίνηση των χαρακτήρων και των εμποδίων ενός παιχνιδιού, για την κίνηση της κάμερας, αλλά και για την αλληλεπίδραση των αντικειμένων. Μέσα από Scripts μπορούμε να δημιουργήσουμε και δυναμικά τα στοιχεία αυτά[20].

2.4.4 Rigidbody

Όπως αναφέραμε και παραπάνω το Rigidbody είναι ένας Component που επιτρέπει στα αντικείμενα να προσομοιώσουν ρεαλιστική φυσική συμπεριφορά. Τέτοια συμπεριφορά μπορεί να είναι η βαρύτητα, η μάζα του, το Collision (ανίχνευση σύγκρουσης με άλλα αντικείμενα) κ.α. Όταν ένα αντικείμενο έχει Rigidbody Component μπορεί να επηρεαστεί από διάφορες δυνάμεις και συγκρούσεις. Σε περίπτωση που θα θελήσουμε να δημιουργήσουμε 2D game θα πρέπει να χρησιμοποιήσουμε τον 2D Rigidbody, αφού χρειάζονται μόνο 2 άξονες X και Y . Μέσα σε ένα Script για την δυναμική αλλαγή των ιδιοτήτων του Rigidbody , πρέπει να χρησιμοποιήσουμε την συνάρτηση FixedUpdate σε αντίθεση με της Update. Αυτό γίνεται διότι η FixedUpdate καλείται αμέσως πριν από κάθε ενημέρωση φυσικής, ενώ η συνάρτηση Update καλείται σε κάθε Frame που περνάει [21].

2.4.5 Collider

Ο Collider είναι ένας Component όπου επιτρέπει στα αντικείμενα να ανιχνεύουν και να ανταποκρίνονται σε συγκρούσεις με άλλα αντικείμενα. Ο Collider διατίθεται με διαφορετικές μορφές όπως ο mesh Collider, ο Box Collider, ο Cylinder Collider, ο Sphere Collider κ.α. Ο Collider συνήθως χρησιμοποιείται σε συνδυασμό με τον Component Rigidbody για την δημιουργία παιχνιδιών βασισμένα πάνω στην φυσική. Όταν δύο αντικείμενα που έχουν τον Component Collider έρθουν σε σύγκρουση, το πρόγραμμα θα το ανίχνευση αυτόματα και θα δημιουργήσει αυτό που λέμε Collision event. Μέσω των Script μπορούμε να απαντήσουμε σε αυτό το event, προσθέτοντας είτε κάποιο particles για την δημιουργία εφέ, είτε να προσθέσουμε ήχο για να κάνουμε την σύγκρουση πιο ρεαλιστική. Μία πολύ γνωστή ιδιότητα του είναι η IsTrigger. Αν επιλέξουμε την ιδιότητα αυτή μας επιτρέπει να ορίσουμε έναν collider σαν trigger και όχι ως φυσικό αντικείμενο. Δηλαδή όταν αυτό το αντικείμενο έρθει σε σύγκρουση με κάποιο άλλο δεν θα γίνει μία φυσική σύγκρουση. Αυτή η ιδιότητα συνήθως χρησιμοποιείται για την δημιουργία Checkpoints ή endgames κ.α. Η πιο γνωστή συνάρτηση για την δυναμική διαχείριση του είναι η OnCollisionEnter() όπου ο κώδικας μέσα σε αυτή εκτελείται με το που υπάρξει κάποια σύγκρουση[22].

2.4.6 Prefabs

Τα Prefabs όπου είναι συντομογραφία του “Prefabricated Object”, όπως αναφέρεται και στο όνομα του είναι ένα προκατασκευασμένο αντικείμενο παιχνιδιού όπου μπορεί να επαναχρησιμοποιηθεί πολλές φορές και σε διάφορες σκηνές. Ο προγραμματιστής μπορεί να δημιουργήσει ένα αντικείμενο το οποίο να αποτελείται είτε από ένα είτε περισσότερα άλλα αντικείμενα και να προσθέσει σε αυτό διάφορες ιδιότητες όπως είναι τα Components. Αφού δημιουργήσει το αντικείμενο μπορεί να το αποθηκεύσει σαν Prefab πολύ απλά με ένα drag and drop του αντικειμένου στο παράθυρο των Assets. Από την στιγμή που θα αποθηκευτεί το αντικείμενο μπορούμε πλέον να έχουμε πρόσβαση σε αυτό με τις αποθηκευμένες ιδιότητές του μέσα από τον κατάλογο Assets και μπορούμε να το σύρουμε στην επιθυμητή σκηνή. Για την δημιουργία παιχνιδιών αυτή η δυνατότητα είναι πολύ χρήσιμη διότι ο προγραμματιστής μπορεί να χρειαστεί να χρησιμοποιήσει το ίδιο αντικείμενο για περισσότερες από μία φορές, οπότε με το Prefab δεν χρειάζεται να το ξανα δημιουργήσει από την αρχή. Ένα άλλο θετικό των Prefabs είναι ότι αν ο προγραμματιστής σκοπεύει να κάνει μία αλλαγή στο μέλλον, μπορεί να ανοίξει το Prefab αυτό και

οποιαδήποτε αλλαγή κάνει θα επηρεάσει όλα τα αντικείμενα που προήλθαν από το Prefab αυτό. Δηλαδή δεν χρειάζεται να υλοποιήσει την ίδια αλλαγή για το κάθε ένα αντικείμενο[23].

2.4.7 Particle System

Το Particle System είναι ένα ισχυρό εργαλείο που επιτρέπει στους προγραμματιστές να προσομοιώσουν διάφορα οπτικά εφέ βασισμένα σε αυτά τα σωματίδια. Με την σωστή διαχείριση των Particle έχει την δυνατότητα ο προγραμματιστής να δημιουργήσει σχεδόν τα πάντα. Από την δημιουργία χιονιού, καπνού, βροχής, φωτιάς, μέχρι και εκρήξεις ή και εφέ σύγκρουσης. Ο τρόπος που λειτουργεί το Particle System είναι ότι σε ένα σημείο του χώρου του παιχνιδιού, το Unity δημιουργεί σωματίδια στην οθόνη μας. Μπορούμε να έχουμε πρόσβαση στις ιδιότητες των σωματιδίων αυτών μέσα από το Unity. Έχουμε την δυνατότητα να αλλάξουμε το μέγεθος, την ταχύτητα, το χρώμα, την φυσική, την διάρκεια ζωής, μέχρι και την υφή τους. Γενικά το σύστημα αυτό μας δίνει ένα υψηλό βαθμό ευελιξίας και ελέγχου για την δημιουργία εφέ στα παιχνίδια μας. Με την σωστή χρήση τους μπορούμε να διαχειριστούμε τα σωματίδια αυτά μέσα από τα Script μας και να τα εφαρμόσουμε για παράδειγμα σε κάποια σύγκρουση[24].

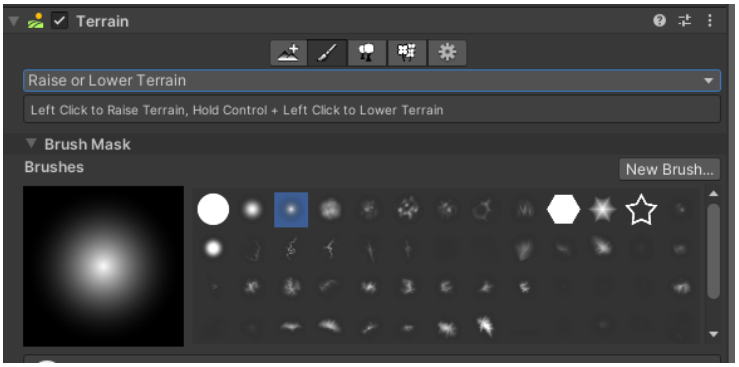
2.4.8 Public και SerializeField

Στο Unity όταν έχουμε μία Public μεταβλητή σημαίνει ότι μπορεί να προσπελαστεί και από άλλα Scripts και Components. Επίσης μία Public μεταβλητή προβάλλεται και στον Inspector, δηλαδή στον Component του ίδιου του Script με αποτέλεσμα ο προγραμματιστής να έχει πιο γρήγορη και εύκολη πρόσβαση στην αλλαγή της τιμής της. Επίσης αυτή η μεταβλητή μπορεί να μην έχει κάποια τιμή String, Integer, Float αλλά να είναι μία τιμή του τύπου Transform, GameObject κ.α. Δηλαδή δημιουργούμε την μεταβλητή μέσα στο Script και στην συνέχεια με drag and drop δίνουμε τιμή στην μεταβλητή αυτή είτε κάποιο αντικείμενο, είτε καποιο particle, είτε κάποιο sound clip κ.α. Έτσι συνδέουμε το Script με κάποιο GameObject και στην συνέχεια μπορούμε να το τροποποιήσουμε μέσα από αυτό. Αν θελήσουμε κάποια μεταβλητή να γίνει ορατή στον Inspector αλλά να μην μπορεί να έχει πρόσβαση κάποιο άλλο Component, τότε αυτή την μεταβλητή την δηλώνουμε ως Serialized Field[25].

2.4.9 Terrain

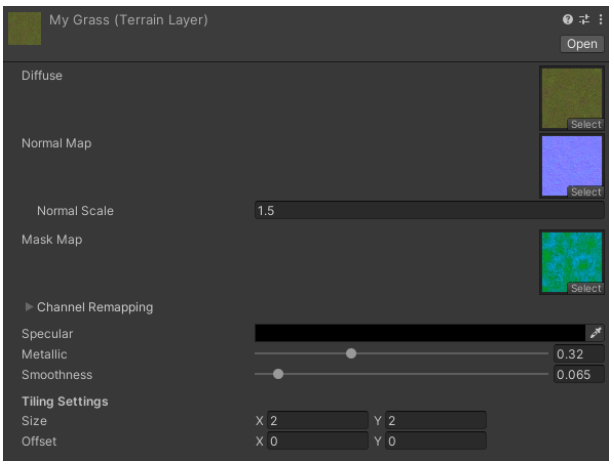
Το Terrain είναι ο Component που μας δίνει την δυνατότητα να δημιουργήσουμε γρήγορα και εύκολα το δικό μας 3D περιβάλλον. Το Terrain στο Unity μας δίνει πολλές δυνατότητες και εργαλεία για την δημιουργία εδάφους, βουνών, ερήμους και οτιδήποτε άλλο τοπίο μπορούμε να σκεφτούμε. Ένα σημαντικό εργαλείο του Terrain είναι το Terrain Editor όπου μπορούμε να δημιουργήσουμε τοπία είτε εισάγοντας χάρτες ύψους (Heightmaps) είτε ζωγραφίζοντας με αριστερό κλικ αφού διαλέξουμε το Paint Terrain. Το Paint Terrain μας δίνει και αυτό ένα μεγάλο φάσμα δυνατοτήτων.

Αρχικά μας δίνει την δυνατότητα να διαλέξουμε την επιλογή “Raise or Lower Terrain”.[26] Αφού διαλέξουμε την επιλογή αυτή διαλέγουμε τι είδος “Brush Mask” θα χρησιμοποιήσουμε. Μπορούμε να χρησιμοποιήσουμε κάποιο έτοιμο που υπάρχει ήδη μέσα στο Unity ή αλλιώς να εισάγουμε κάποιο δικό μας. Στην συνέχεια έχουμε την δυνατότητα να αλλάξουμε το μέγεθος της Brush Mask μας , την περιστροφή του, την δύναμη του αλλά και την διασπορά του. Αφού τελειώσουμε με την επεξεργασία της Brush Mask μας, είμαστε έτοιμοι να ξεκινήσουμε και να ζωγραφίσουμε το Terrain μας. Με αριστερό κλικ υψώνουμε το έδαφος ενώ με cntrl + αριστερό κλικ χαμηλώνουμε το έδαφος. Επίσης υπάρχει και η δυνατότητα να κάνουμε το έδαφος πιο λείο με shift + αριστερό κλικ[27].



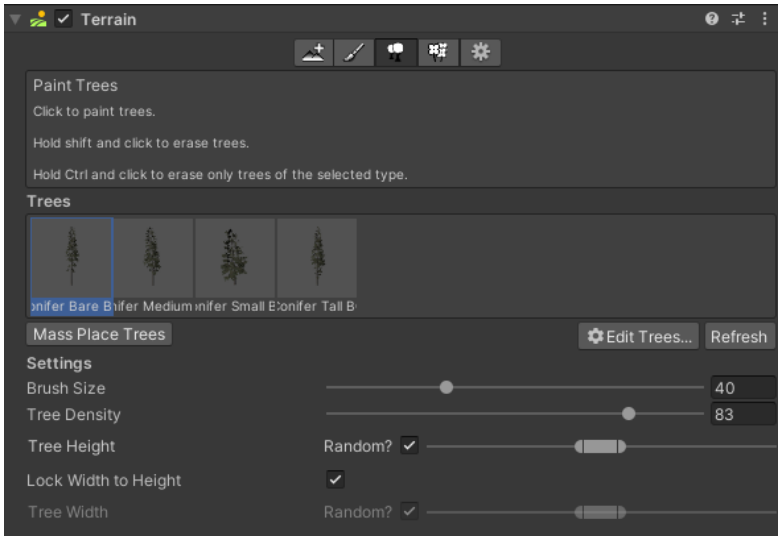
Σχήμα 2.9: Terrain Raise or Lower Component

Μία δεύτερη και πολύ ισχυρή δυνατότητα που μας δίνει ο Terrain Component είναι αυτή του Paint Texture. Με το Paint Texture μπορούμε να δώσουμε υφή στο Terrain που δημιουργήσαμε. Πάλι θα πρέπει να διαλέξουμε το Brush Mask που θέλουμε και να τροποποιήσουμε τις ιδιότητες του, με την διαφορά ότι υπάρχει η δυνατότητα να δημιουργήσουμε Layers[28]. Ας πούμε ότι δημιουργούμε ένα για την υφή του εδάφους. Μπορούμε να χρησιμοποιήσουμε κάποιο έτοιμο Layer από το Asset Store ή να δημιουργήσουμε το δικό μας. Αφού προσθέσουμε το Layer έχουμε την δυνατότητα να το επεξεργαστούμε κατάλληλα στα θέλω μας. Μπορούμε να προσθέσουμε Diffuse, Normal Map , Mask map , να παίξουμε με τις ιδιότητες Metallic και Smoothness για να προσαρμόσουμε την υφή κατάλληλα.



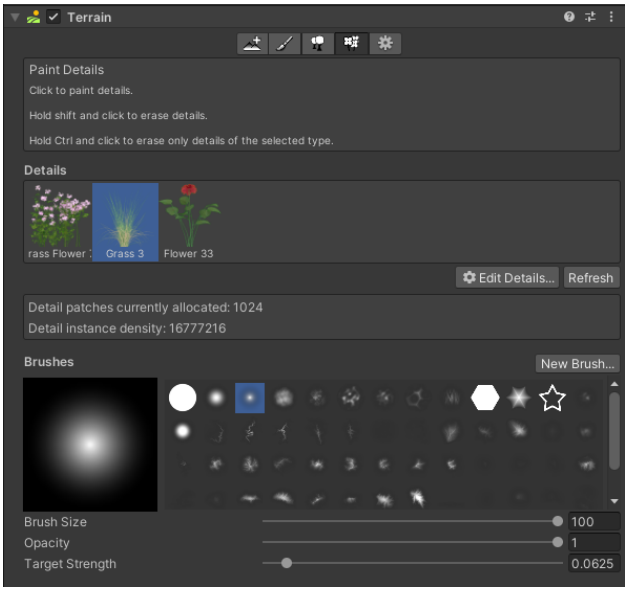
Σχήμα 2.10: Grass Texture Layer

Για να προσθέσουμε δέντρα στο περιβάλλον μας το Terrain Component μας δίνει την ιδιότητα Paint Trees. Και σε αυτήν την ιδιότητα έχουμε την επιλογή να χρησιμοποιήσουμε είτε κάποιο έτοιμο Prefab από το Asset Store του Unity είτε να προσθέσουμε ένα δικό μας μοντέλο. Στην συνέχεια επιλέγουμε το Brush Mask μας. Το Brush Mask του Paint tree μας παρέχει την δυνατότητα να μετατρέψουμε το μέγεθος της, την πυκνότητα των δέντρων και το ύψος τους. Αφού επιλέξουμε το Brush Mask που μας βολεύει μπορούμε να πάμε στην σκηνή του παιχνιδιού μας και με αριστερό κλικ να προσθέσουμε τα πρώτα δέντρα μας. Επειδη το περιβάλλον του παιχνιδιού μας μπορεί να είναι πολύ μεγάλο, το Unity μας δίνει και την επιλογή “Mass Place Trees” όπου διαλέγουμε πόσα δέντρα από το επιλεγμένο μοντέλο θέλουμε να προστεθούν αυτόματα και τυχαία μέσα σε όλο το περιβάλλον. Είναι πολύ πιθανό κάποια από αυτα τα δέντρα να τοποθετηθούν σε σημεία που δεν θέλαμε όπως δρόμους, γι αυτό μπορούμε να πάμε με το Brush Mask πάνω στα ανεπιθύμητα δέντρα και με cntrl + αριστερό κλικ να τα αφαιρέσουμε[29].



Σχήμα 2.11: Terrain Paint Trees

Για την δημιουργία μικρών αντικειμένων όπως είναι τα λουλούδια, το γρασίδι, οι βράχοι και οι θάμνοι στην επιφάνεια του Terrain μας, υπάρχει η ιδιότητα Paint Details. Με την ιδιότητα αυτή έχουμε την δυνατότητα να βελτιώσουμε την οπτική ποιότητα της σκηνής. Για την εισαγωγή κάποιου τέτοιου αντικειμένου ακολουθούμε περίπου τα ίδια βήματα που ακολουθούμε και για την εισαγωγή των δέντρων. Σε αυτή την ιδιότητα δεν μπορούμε όμως να προσθέσουμε τα αντικείμενα τυχαία στο περιβάλλον όπως κάναμε με τα δέντρα. Πρέπει δηλαδή να τα προσθέσουμε όλα με το χέρι. Αν στο μέλλον αλλάξουμε κάποιο Setting από τις ιδιότητες του αντικειμένου(π.χ. αλλάξουμε το μέγεθος από το μοντέλο ενός λουλουδιού) τότε θα αλλάξει για όλα τα αντικείμενα που προστέθηκαν από το ίδιο μοντέλο[30].



Σχήμα 2.12: Terrain Paint Details

Κεφάλαιο 3ο: Stratego - Κανόνες και Πιόνια

3.1 Εισαγωγή

Το Stratego είναι ένα επιτραπέζιο παιχνίδι που αποτελείται από 2 παίκτες, όπου ο καθένας τους έχει από ένα στρατό 40 πιονών. Ένας παίκτης παίζει με τον κόκκινο στρατό(όπου ξεκινάει και πρώτος το παιχνίδι), ενώ ο άλλος παίκτης παίζει με τον μπλε στρατό. Ένα από αυτά τα 40 πιόνια είναι και η σημαία του παίκτη, όπου σκοπός του παιχνιδιού είναι να διεκδικήσει την αντίπαλη σημαία ή να συλλάβει όλα τα αντίπαλα πιόνια(πλην της σημαίας) έτσι ώστε ο αντίπαλος να μην μπορεί να κάνει περαιτέρω κινήσεις. Επίσης αν κάποιος από τους δύο παίκτες επαναλαμβάνει την ίδια κίνηση τρεις φορές συνεχόμενα, τότε χάνει αυτόματα το παιχνίδι. Οι κανόνες του παιχνιδιού είναι αρκετά απλοί

ώστε να μπορούν να κατανοηθούν και από μικρά παιδιά, ωστόσο έχει μεγάλο βάθος στρατηγικής που προσελκύει και τους μεγαλύτερους[31].

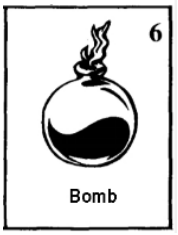
3.2 Πιόνια και ιδιότητές τους

3.2.1 Σημαία

Σχήμα 3.1: Flag Icon[32]

Υπάρχει μόνο μία σημαία σε κάθε στρατόπεδο και είναι το πολυτιμότερο πιόνι του κάθε παίκτη. Αν ένας από τους δύο παίκτες συλλάβει την αντίπαλη σημαία το παιχνίδι σταματά και ο παίκτης αυτός κερδίζει. Γι Αυτό τον λόγο ο κάθε παίκτης πρέπει να κρύψει και να προστατέψει την σημαία του από τον αντίπαλο. Συνήθως οι παίκτες τοποθετούν την σημαία τους στην τελευταία σειρά για να είναι πιο δύσκολο να εντοπιστεί από τον αντίπαλό τους και τοποθετούν γύρω της είτε βόμβες είτε πιόνια μεγάλης κατάταξης. Δεν έχουμε την δυνατότητα να κουνήσουμε την σημαία από την στιγμή που θα ξεκινήσει το παιχνίδι οπότε πρέπει να προσέξουμε που θα την τοποθετήσουμε.

3.2.2 Βόμβα



Σχήμα 3.2: Bomb Icon[32]

Όπως αναφέραμε οι βόμβες συνήθως τοποθετούνται κοντά στην σημαία για να την προστατεύσουν από τα αντίπαλα μεγάλης κατάταξης πιόνια. Υπάρχουν συνολικά 6 βόμβες σε κάθε στρατόπεδο. Ούτε οι βόμβες έχουν την δυνατότητα να μετακινηθούν από την στιγμή που θα ξεκινήσει το παιχνίδι γι αυτό θα πρέπει να τις τοποθετήσουμε έξυπνα. Οι βόμβες είναι πολύ δυνατές στο

παιχνίδι διότι παρόλο που δεν μπορούνε να κουνηθούνε, αν τις επιτεθεί οποιαδήποτε πιόνι εκτός του Miner τότε το πιόνι αυτό χάνει και φεύγει από το πεδίο μάχης(το board). Αν όμως ο αντίπαλος επιτεθεί την βόμβα μας με τον Miner τότε η βόμβα μας φεύγει από το παιχνίδι και ο Miner καταλαμβάνει το τετραγωνάκι που είχε πριν η βόμβα.

3.2.3 Κατάσκοπος

Σχήμα 3.3: Spy Icon[32]

Ο κατάσκοπος είναι και αυτός ένα από τα πιο σημαντικά πιόνια του στρατού μας. Υπάρχει μόνο ένας κατάσκοπος σε κάθε στρατόπεδο. Η κατάταξη του είναι το 1 όπου είναι και η μικρότερη και σημαίνει ότι χάνει από οποιοδήποτε άλλο πιόνι στο παιχνίδι. Ο λόγος που κάνει τον κατάσκοπο τόσο σημαντικό είναι ότι έχει την δυνατότητα αν αυτός επιτεθεί τον Marshal(αριθμός κατάταξης 10) τότε τον κερδίζει και τον βγάζει από το παιχνίδι. Είναι το μόνο πιόνι που μπορεί να κερδίσει τον Marshal αλλά πρέπει ο κατάσκοπος να επιτεθεί σε αυτόν. Αν ο Marshal επιτεθεί στον κατάσκοπο τότε ο κατάσκοπος χάνει και βγαίνει αυτός από το παιχνίδι.

3.2.4 Ανιχνευτής

Σχήμα 3.4: Scout Icon[32]

Ο ανιχνευτής(ή αλλιώς και scout) είναι το δεύτερο μικρότερο νούμερο σε κατάταξη στο παιχνίδι το 2. Υπάρχουν στο σύνολο 8 ανιχνευτές σε κάθε στρατόπεδο. Οι ανιχνευτές είναι τα μόνα πιόνια στο παιχνίδι που μπορούν να κινηθούν σε περισσότερα από ένα τετραγωνάκια μέσα σε έναν γύρο. Μπορεί να κινηθεί όσα τετράγωνα θέλει σε όποια κατεύθυνση θέλει μέχρι να συναντήσει κάποιο όριο του χάρτη ή κάποιο άλλο πιόνι(είτε δικό του είτε αντίπαλο). Επίσης μπορεί να μετακινηθεί πολλαπλά τετράγωνα αλλά και να επιτεθεί στον ίδιο γύρο κάποιο αντίπαλο πιόνι. Στόχος των ανιχνευτών είναι όπως λέει και το όνομα τους, να ανιχνεύσουν τα αντίπαλα σημαντικά πιόνια

όπως είναι βόμβες, σημαίες ή και μεγάλης κατάταξης πιόνια, ώστε να τους βοηθήσει στην αναγνώριση του αντιπάλου σχηματισμού.

3.2.5 Miner

Σχήμα 3.5: Miner Icon[32]

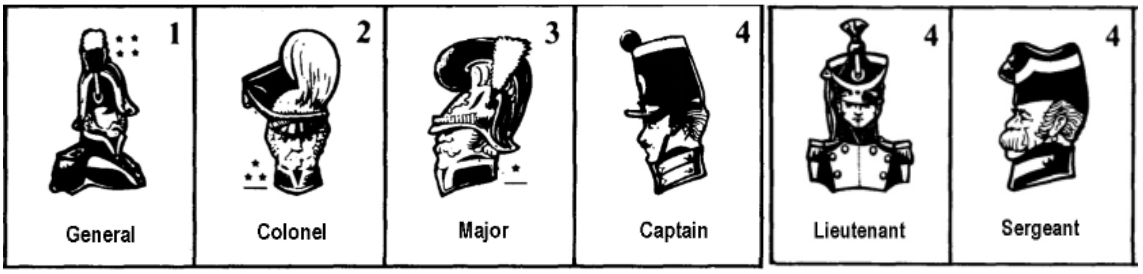
Ο Miner όπως αναφέραμε και παραπάνω είναι ο μόνος που μπορεί να κατακτήσει μία βόμβα. Υπάρχουν στο σύνολο 5 Miners σε κάθε στρατόπεδο. Σκοπός τους είναι να ανοίξουν το δρόμο για την αντίπαλη σημαία νικώντας τις αντίπαλες βόμβες. Για να το καταφέρει αυτό ο παίκτης συνήθως πρέπει πρώτα να ανιχνεύσει που βρίσκονται οι αντίπαλες βόμβες και με την κατάλληλη στρατηγική να καταφέρει να φτάσει τον Miner κοντά στην αντίπαλη βόμβα.

3.2.6 Marshal

Σχήμα 3.6: Marshal Icon[32]

Ο Marshal είναι το μεγαλύτερο σε αξία κατάταξης πιόνι και είναι ίσο με 10. Μπορεί να κερδίσει οποιοδήποτε άλλο πιόνι εκτός από το αντίπαλο 10, όπου αν επιτεθεί το ένα στο άλλο τότε χάνουν και τα δύο και απομακρύνονται από το πεδίο μάχης. Όπως αναφέραμε και παραπάνω αν ο κατάσκοπος κατάταξης 1 επιτεθεί στον Marshal τότε ο Marshal χάνει, αλλά αν επιτεθεί πρώτος ο Marshal τότε κερδίζει.

3.2.7 Όλα τα υπόλοιπα πιόνια (4,5,6,7,8,9)

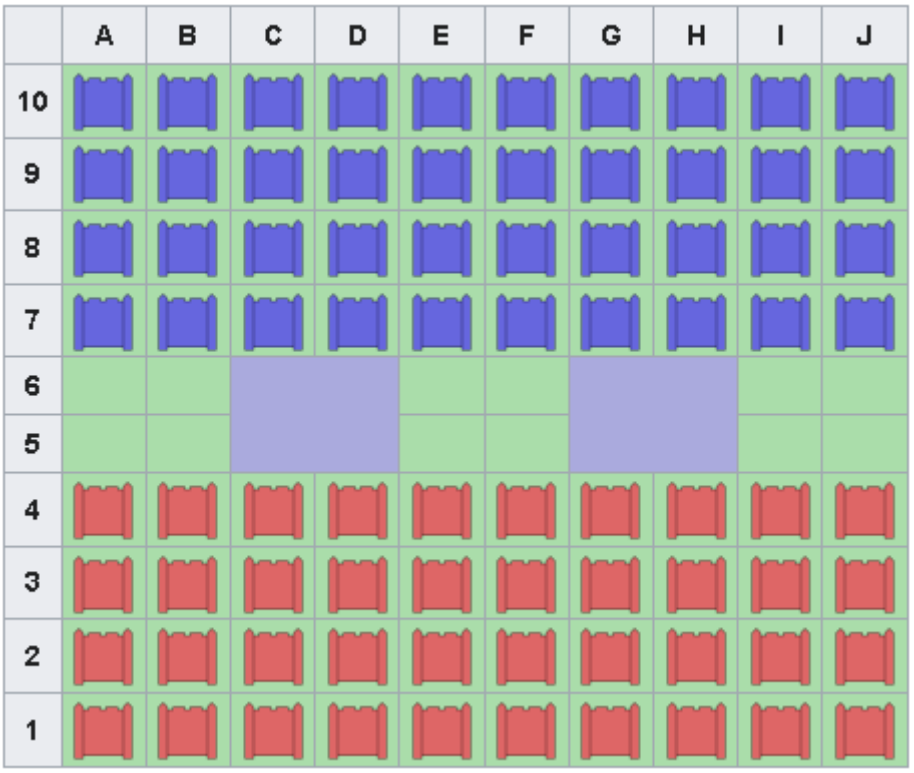


Σχήμα 3.7: General-Colonel-Major-Captain-Lieutenant-Sergeant Icons[32]

Τα υπόλοιπα πιόνια που περισσεύουν είναι ένας General κατάταξης 9, δύο Colonel κατάταξης 8, τρεις Major κατάταξης 7, τέσσερις Captain κατάταξης 6, τέσσερις Lieutenant κατάταξης 5 και τέσσερις Sergeant κατάταξης 4. Όλα αυτά τα πιόνια δεν έχουν κάποια ειδική δυνατότητα όπως είχαν το προηγούμενα που είδαμε. Κάθε φορά που κάποιος παίκτης επιτίθεται σε ένα αντίπαλο πιόνι κερδίζει αυτό με την υψηλότερη κατάταξη. Αν τα πιόνια έχουν τον ίδιο αριθμό κατάταξης τότε χάνουν και τα δύο και απομακρύνονται από το πεδίο μάχης. Επίσης τα πιόνια αυτά κινούνται με τον ίδιο ακριβώς τρόπο που κουνιέται και ο κατάσκοπος και ο Marshal. Δηλαδή ένα μόνο κουτάκι σε κάθε γύρο προς οποιαδήποτε κατεύθυνση. Δεν μπορεί κάποιο από αυτα τα πιόνια να κινηθεί και να επιτεθεί σε έναν γύρο. Για να επιτεθεί ένα πιόνι θα πρέπει να βρίσκεται ακριβώς δίπλα στο αντίπαλο του[33].

3.3 Χάρτης

Ο χάρτης του παιχνιδιού αποτελείται από 100 τετραγωνάκια(δηλαδή 10χ10). Όταν ξεκινήσει το παιχνίδι ο κάθε παίκτης καταλαμβάνει 40 τετραγωνάκια ο καθένας. Στην συνέχεια υπάρχουν άλλα 12 ελεύθερα τετραγωνάκια που χωρίζουν τα δύο στρατόπεδα και μπορούν οι παίκτες να κινηθούν πάνω σε αυτά. Τέλος υπάρχουν 8 τετραγωνάκια όπου δεν μπορεί κανένας από τους δύο παίκτες να προχωρήσει πάνω σε αυτά, χωρίζοντας το κέντρο του χάρτη σε 3 ίσα μέρη. Αυτά τα 8 τετραγωνάκια αναπαριστάνεται με μπλε χρώμα (σαν να υπάρχουν 2 λίμνες ανάμεσα στο πεδίο μάχης) και οι παίκτες θα πρέπει να κινηθούν γύρω από αυτές.



Σχήμα 3.8: Stratego Map[31]

Τα πιόνια του στρατού έχουν 2 πλευρές. Από την μία μεριά φαίνεται η κατάταξη του πιονιού ενώ από την άλλη φαίνεται απλά ένας πύργος. Ο κάθε παίκτης έχει τα πιόνια του γυρισμένα προς τον εαυτό του για να μπορεί να δει την κατάταξη του στρατού του ενώ ο αντίπαλος δεν μπορεί να δει την κατάταξη του. Στόχος του κάθε παίκτη είναι να συλλάβει την αντίπαλη σημαία αλλά και να προστατέψει και την δική του.

3.4 Συμπεράσματα

Το Stratego είναι ένα παιχνίδι όπου ο παίκτης δεν έχει πληροφορίες για τα αντίπαλα πιόνια(σε αντίθεση με το σκάκι π.χ.). Έτσι το παιχνίδι απαιτεί από τον παίκτη όχι μόνο να υπολογίζει τις επόμενες κινήσεις του και του αντιπάλου του, πριν κάνει την οποιαδήποτε κίνηση αλλά και να ψυχολογει την κάθε μάχη, να μπλοφάρει και να δημιουργεί παγίδες για τον αντίπαλο του. Είναι ένα παιχνίδι κυρίως στρατηγικής όπου για να κερδίσεις χρειάζεται να έχεις ένα ολοκληρωμένο σχέδιο παιχνιδιού για να φέρεις τον εαυτό σου σε πλεονέκτημα έναντι του αντιπάλου σου. Υπάρχουν πάρα πολλές διαφορετικές στρατηγικές που μπορεί να ακολουθήσει κανείς. Για παράδειγμα μπορείς να παγιδεύσεις μέσα σε βόμβες ένα απλό πιόνι ώστε ο αντίπαλος να νομίζει ότι προστατεύεις την σημαία σου και να επικεντρώσει όλες τις δυνάμεις του σε αυτό ενώ η σημαία σου βρίσκεται στην άλλη μεριά του χάρτη. Τέλος το Stratego είναι ένα παιχνίδι που βοηθάει τον άνθρωπο με πολλούς διαφορετικούς τρόπους όπως αναπτύσει την στρατηγική του σκέψη και την ικανότητα του στην επίλυση

προβλημάτων, βοηθάει στην γρήγορη και σωστή λήψη αποφάσεων και στην ικανότητα προσαρμογής σε νέα δεδομένα και πολλά άλλα.

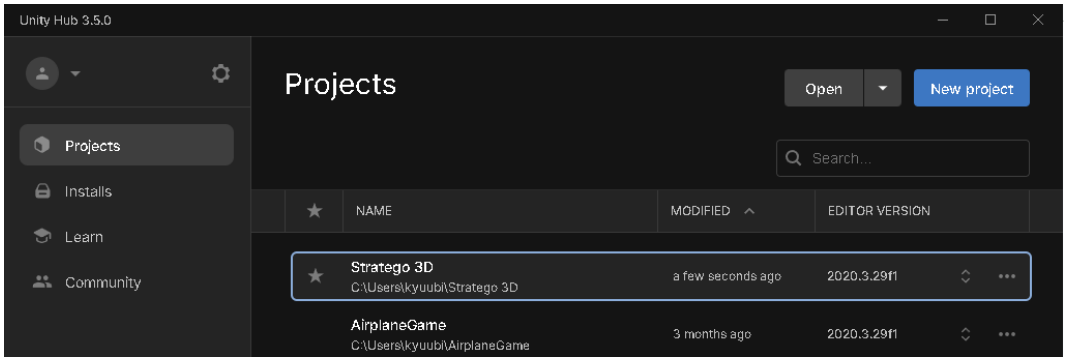
Κεφάλαιο 4ο: Ανάπτυξη του παιχνιδιού Stratego

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα δούμε τον τρόπο με τον οποίο υλοποιήσαμε το παιχνίδι μέσα από την μηχανή της Unity. Θα δείξουμε αναλυτικά όλα τα βήματα που ακολουθούμε για να φτάσουμε στο τελικό αποτέλεσμα του παιχνιδιού. Το παιχνίδι δημιουργήθηκε σε περιβάλλον 3D, δηλαδή σε τρισδιάστατο κόσμο. Επίσης ο χρήστης που θα παίζει το παιχνίδι θα ελέγχει τον κόκκινο στρατό, ενώ θα παίζει ενάντια σε ένα σχετικά εύκολο αντίπαλο BOT. Αρχικά ο χρήστης θα πρέπει να επιλέξει τον τρόπο με τον οποίο θέλει να τοποθετήσει το στρατό του. Αφού ολοκληρώσει την διαδικασία τοποθέτησης θα πρέπει να πατήσει το κουμπί “Start” για να ξεκινήσει η μάχη. Στην συνέχεια ο παίκτης έχει την πρώτη κίνηση εφόσον έχει και τον κόκκινο στρατό, οπότε θα πρέπει να διαλέξει πιο πιόνι θα κινήσει πρώτα. Εφόσον ο παίκτης κάνει την κίνηση του, το BOT με καθυστέρηση ενός δευτερολέπτου θα παίξει και αυτό μία κίνηση και η σειρά μεταξύ του παίκτη και του υπολογιστή αλλάζουνε εναλλάξ μέχρι να τελειώσει το παιχνίδι.

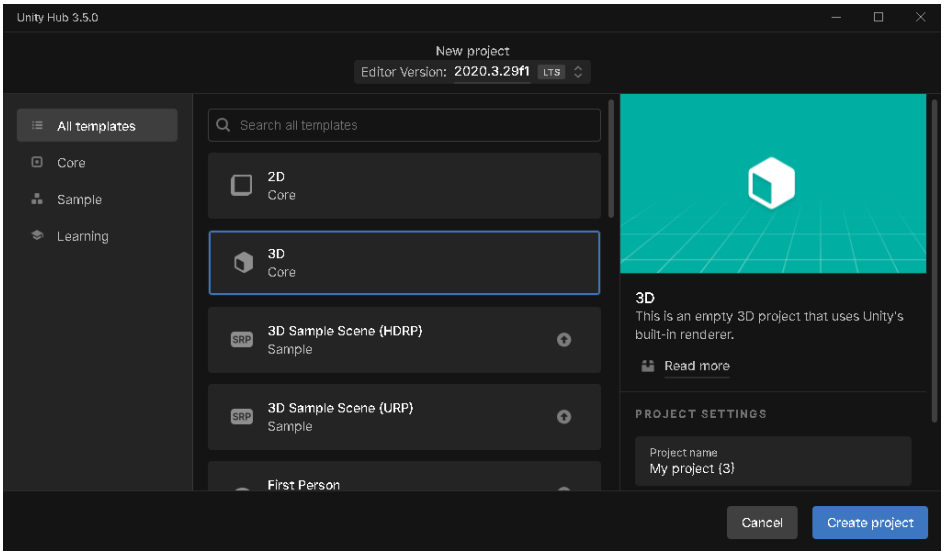
4.2 Ξεκινώντας με το Unity

Αρχικά ανοίγωντας την εφαρμογή της Unity έχουμε την δυνατότητα να πατήσουμε στο κουμπί “New project” για την δημιουργία ενός νέου project ή “Open” για να ανοίξουμε ένα ήδη υπάρχων project και να το επεξεργαστούμε από εκεί που το αφήσαμε.



Σχήμα 4.1: Unity Hub new project

Στη συνέχεια θα πρέπει να διαλέξουμε πιο Template θέλουμε για το project μας. Υπάρχουν πολλά διαφορετικά Template, όπως το 2D το οποίο αναφέρεται σε παιχνίδια δύο διαστάσεων, το 3D το οποίο αναφέρεται σε παιχνίδια τριών διαστάσεων όπου είναι και το Template που θα χρησιμοποιήσουμε για το παιχνίδι μας. Κάποια άλλα Templates είναι 3D mobile, 2D mobile, AR, VR, First Person, Third Person κ.α. Αφού διαλέξουμε το 3D Template και επιλέξουμε πιο θα είναι το όνομα του project μας, πατάμε στο κουμπί “Create project” και έχουμε τελειώσει με την δημιουργία του παιχνιδιού μας.



Σχήμα 4.2: Unity Hub create project

Αφού δημιουργήσουμε το project μας, το ανοίγουμε και εμφανίζεται στην οθόνη μας η αρχική οθόνη της εφαρμογής Unity όπου έχουμε αναλύσει τα πεδία της στο 2ο κεφάλαιο.

4.3 Δημιουργία του Board

Αρχικά το πρώτο εργαλείο που χρειαζόμαστε για ένα οποιοδήποτε επιτραπέζιο παιχνίδι είναι το Board του παιχνιδιού. Δηλαδή τον πίνακα όπου θα κάθονται αλλά και θα μετακινούνται τα πιόνια. Υπάρχουν διάφοροι τρόποι για να δημιουργήσει κανείς το Board ενός παιχνιδιού μέσα από την εφαρμογή της Unity. Τον τρόπο που επιλέξαμε εμείς είναι να δημιουργείται με την έναρξη του παιχνιδιού. Δηλαδή μέσα στην μέθοδο Awake() όπου καλείται όταν το παιχνίδι μας ξεκινα, καλούμαι την μέθοδο GenerateTiles().

```
private void GenerateTiles(float tileSize, int tileCountX, int
tileCountY)
{

      tiles = new GameObject[tileCountX, tileCountY];
      for (int x = 0; x < tileCountX; x++)
      {
            for (int y = 0; y < tileCountY; y++)
            {
                  tiles[x,y] = GenerateTile(tileSize, x, y);
            }
      }
}
```

Σχήμα 4.3: Κώδικας της μεθόδου GenerateTiles

Όπως βλέπουμε στο σχήμα 4.3 δημιουργούμε ένα νέο gameobject για κάθε ένα x και y, όπου tileCountX είναι ίσο με 10 και tileCountY είναι και αυτό ίσον με 10(δηλαδή οι διαστάσεις του Board μας) και στην συνέχεια καλούμαι την μέθοδο GenerateTile για το αντίστοιχο x και y.

```
private GameObject GenerateTile(float tileSize, int x, int y)
{
      GameObject tileObject = new GameObject(string.Format($"X:{x},
      Y:{y}"));
      tileObject.transform.parent = transform;

      Mesh mesh = new Mesh();
      tileObject.AddComponent<MeshFilter>().mesh = mesh;
      tileObject.AddComponent<MeshRenderer>().material = tileMat;

      Vector3[] vertices = new Vector3[4];
      vertices[0] = new Vector3(x * tileSize, 0.1f, y * tileSize);
      vertices[1] = new Vector3(x * tileSize, 0.1f, (y+1) * tileSize);
      vertices[2] = new Vector3((x+1) * tileSize, 0.1f, y * tileSize);
      vertices[3] = new Vector3((x+1) * tileSize, 0.1f, (y+1) *
      tileSize);
```

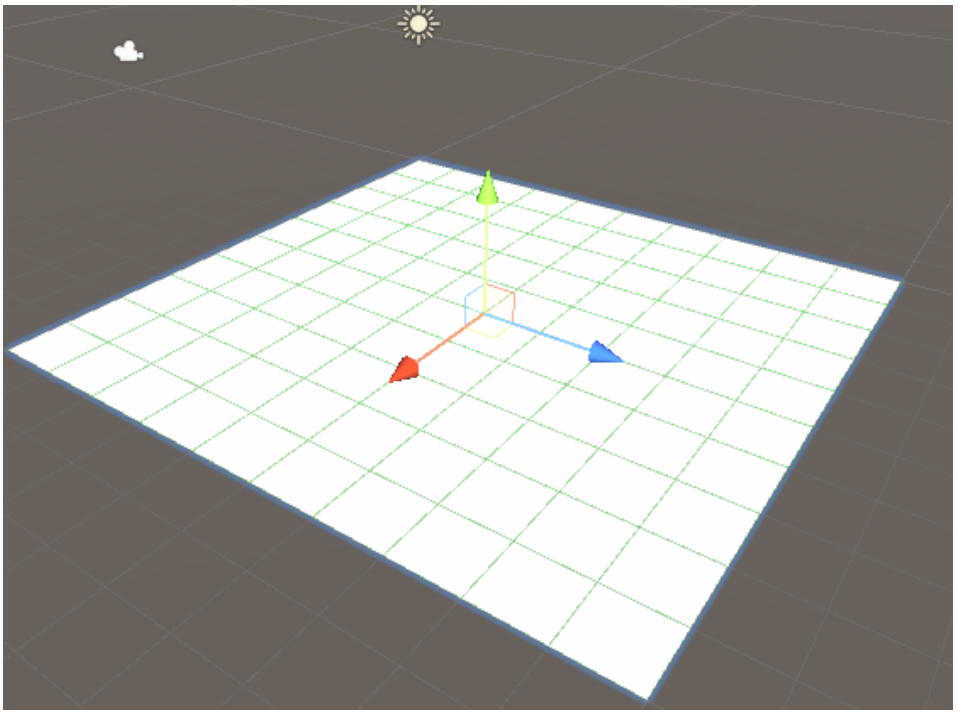
```
        int[] tris = new int[] { 0, 1, 2 ,1 ,3 ,2};

        mesh.vertices = vertices;
        mesh.triangles = tris;
        mesh.RecalculateNormals();

         tileObject.layer = LayerMask.NameToLayer("Tile");
         tileObject.AddComponent<BoxCollider>();
         return tileObject;
}
```

Σχήμα 4.4: Κώδικας της μεθόδου GenerateTile

Στην μέθοδο GenerateTile() όπως βλέπουμε και στο παραπάνω σχήμα αρχικά ονομάζουμε το κάθε ένα Tile με το αντίστοιχο x και y του. Στην συνέχεια για να μπορέσουμε να κάνουμε render τα Tiles που δημιουργήσαμε, πρέπει να τα δώσουμε ένα Mesh και ένα MeshRenderer. Δημιουργούμε τα τέσσερα Vertices του Tile μας, δηλαδή τις τέσσερις γωνίες του τετραγώνου μας. Μετά δημιουργούμε τα τρίγωνα που αποτελούν το τετράγωνο του Tile και τα αναθέτουμε στο αντίστοιχο mesh. Τέλος αναθέτουμε στο gameobject μας και ένα BoxCollider όπου θα μας χρειαστεί για να αλληλεπιδρούμε με τα Tiles που δημιουργήσαμε. Τέλος με τον κώδικα “RecalculateNormals” διορθώνουμε τον φωτισμό του παιχνιδιού που αντανακλάται πάνω στο mesh που δημιουργήσαμε. Και κάπως έτσι δημιουργήσαμε το Board του παιχνιδιού μας[34].



Σχήμα 4.5: Game Board

Τέλος διαγράφουμε τα 8 τετραγωνάκια που δεν μπορούν οι παίκτες να πατήσουν με την εντολή Destroy και το Board μας είναι πλέον έτοιμο.

4.4 Highlight Hover Tile

Έχοντας πλέον δημιουργήσει το Board μας, θα χρειαστούμε να αλληλεπιδράσουμε με κάποιο τρόπο με αυτό. Η αλληλεπίδραση αυτή θα γίνει όταν κάνουμε “mouse over” (hovering) πάνω από τα Tiles μας. Θέλουμε δηλαδή κάθε φορά που πηγαίνουμε το ποντίκι μας πάνω από κάποιο τετραγωνάκι να παίρνουμε τις αντίστοιχες συντεταγμένες του και να υπάρχει και μία οπτική διαφορά ώστε να βρίσκουμε εύκολα σε πιο Tile βρισκόμαστε. Επειδή όσο παίζουμε το παιχνίδι το ποντίκι μας θα αλλάζει θέση αρκετά συχνά, ο καλύτερος τρόπος για να υλοποιήσουμε κάτι τέτοιο είναι μέσα στην μέθοδο Update(), όπου η μέθοδος αυτή καλείται για κάθε ένα Frame που περνάει.

```
RaycastHit info;
Ray ray = currentCamera.ScreenPointToRay(Input.mousePosition);
if (Physics.Raycast(ray, out info, 100, LayerMask.GetMask("Tile",
"Hover" , "Highlight")))
{
      //Get the indexes of the tiles i hit
      Vector2Int hitPosition =
      LookUpTileIndex(info.transform.gameObject);

      //If we are hovering a tile after not hovering any tile
      if (currentHover == -Vector2Int.one)
      {
            currentHover = hitPosition;
            tiles[hitPosition.x, hitPosition.y].layer =
            LayerMask.NameToLayer("Hover");
      }

      //If we were already hovering a tile, change the previous one
      if (currentHover != hitPosition)
      {
            if(ContainsValidMoves(ref availableMoves, currentHover) &&
      preGame)
            {
```

```
                tiles[currentHover.x, currentHover.y].layer =
                LayerMask.NameToLayer("Highlight");
            }else{
                tiles[currentHover.x, currentHover.y].layer =
                LayerMask.NameToLayer("Tile");
            }
            currentHover = hitPosition;
            tiles[hitPosition.x, hitPosition.y].layer =
            LayerMask.NameToLayer("Hover");}
        }
```

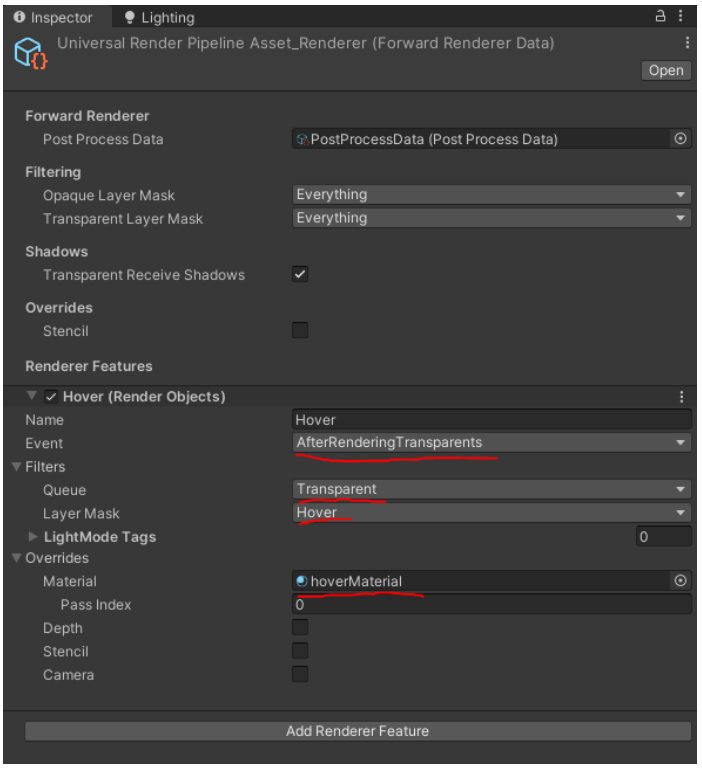
Σχήμα 4.6: Μέρος κώδικα της μεθόδου Update

```
private Vector2Int LookUpTileIndex(GameObject hitInfo)
{
      for (int x = 0; x < TILE_COUNT_X; x++)
      {
            for(int y = 0; y < TILE_COUNT_Y; y++)
            {
                if(tiles[x,y] == hitInfo)
                {
                    return new Vector2Int(x,y);
                }
            }
      }
return -Vector2Int.one;
}
```

Σχήμα 4.7: Κώδικας της μεθόδου LookUpTileIndex

Αρχικά δημιουργούμε μία ακτίνα που θα χτυπάει τα gameobjects που έχουν Layer mask είτε “Tiles” είτε “Hover”, όπου Hover γίνονται τα Tiles που έχουμε από πάνω τους το ποντίκι μας. Αν το ποντίκι βρίσκεται όντως πάνω από κάποιο Tile τότε με την μέθοδο LookUpTileIndex παίρνουμε το αντίστοιχο Index για το αντίστοιχο Tile. Και στην συνέχεια με ελέγχουμε αν κάνουμε Hover κάποιο Tile χωρίς να κάναμε πριν Hover κάποιο άλλο Tile(δηλαδή αν κάναμε mouse over πάνω από κάποιο Tile ενώ πριν είχαμε το ποντίκι μας έξω από το χάρτη) και αν ισχύει τότε αλλάζουμε το TIle Layermask σε Hover. Στην συνέχεια ελέγχουμε αν κάναμε Hover πριν κάποιο άλλο Tile και τώρα

πήγαμε σε κάποιο καινούργιο, οπότε αλλάζουμε το καινούργιο σε Tile Layermask σε Hover και κάνουμε το παλιό από Hover σε Tile. Και αν δεν ισχύει κανένα από τα δύο παραπάνω τότε αλλάζουμε το προηγούμενο Tile σε Layermask Tile. Στην συνέχεια δημιουργούμε δύο material ένα για το Hover και ένα για το Tile layermask, όπου ανάλογα πιο mask έχουμε θα υπάρχει και το αντίστοιχο χρώμα για να τα ξεχωρίζουμε. Αναθέτουμε τα δύο material μέσα από το σύστημα URP της Unity[35].



Σχήμα 4.8: Σύστημα URP

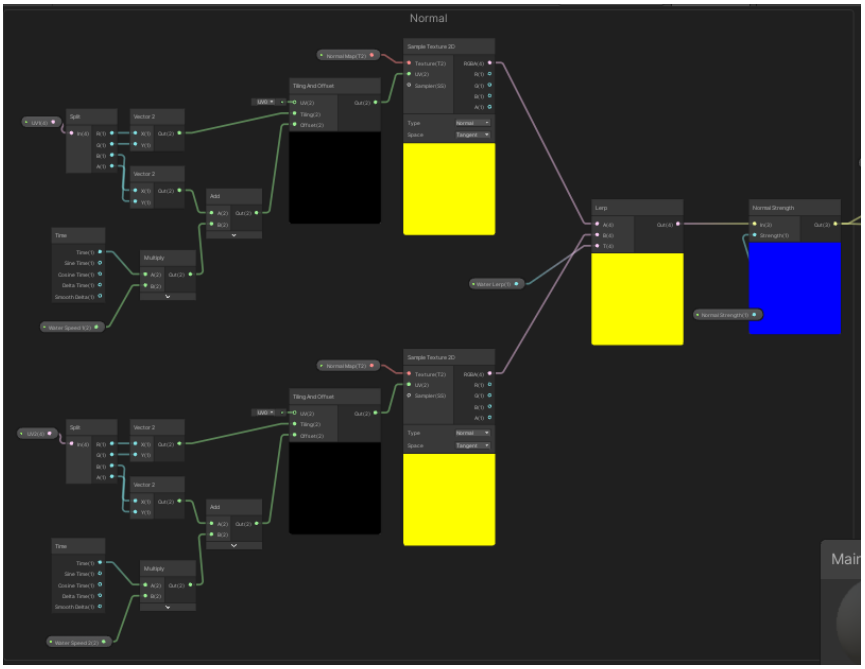
Οπότε κάθε φορά που θα κάνουμε mouse over πάνω από κάποιο Tile θα αλλάζει το layermask του σε "Hover" και θα αλλάζει και το χρώμα του από άσπρο σε κόκκινο.

4.5 Terrain Map

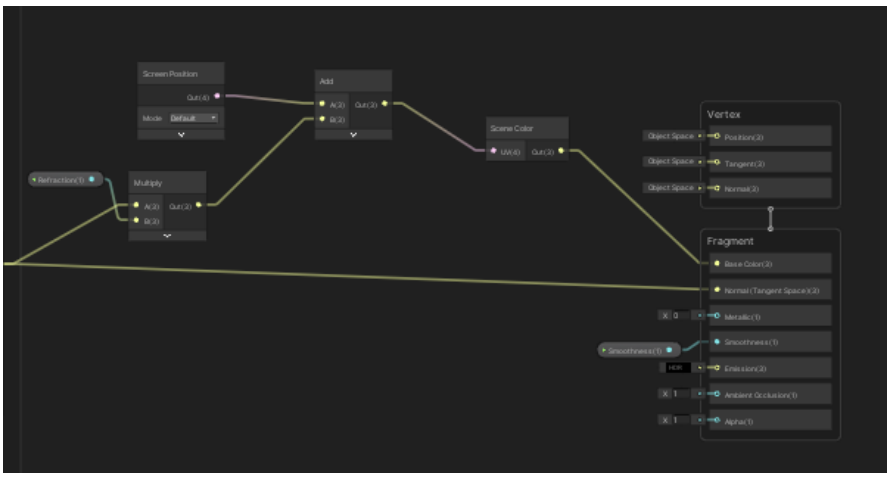
Για να κάνουμε τον χάρτη του παιχνιδιού μας πιο όμορφο θα δημιουργήσουμε ένα Terrain και θα το τοποθετήσουμε ακριβώς στο ίδιο σημείο που εμφανίζεται και το board μας για να δημιουργήσει την ψευδαίσθηση ότι τα πιόνια κινούνται πάνω στο terrain. Για να εμπλουτίσουμε το Terrain μας με όμορφα στοιχεία θα χρησιμοποιήσουμε τα "Terrain Sample Asset Pack" [36] και το "Grass And Flowers Pack 1" [37] από το Asset Store της Unity. Στην συνέχεια θα προσθέσουμε στο Terrain Layers ένα καινούργιο Layer όπου θα το ονομάσουμε "Moss_terrain" και θα του αναθέσουμε το "Grass_Moss_Terrain_Layer". Μπορούμε να πειράξουμε τα settings του layer, να δώσουμε έμφαση στο Diffuse - Color Tint ή στο Opacity as Density και να αλλάξουμε και το Normal Map Scale έτσι ώστε να παράξουμε το καλύτερο για εμάς αποτέλεσμα. Εφόσον τελειώσουμε με την δημιουργία του

Moss Layer που θα το χρησιμοποιήσουμε για να αναπαραστήσουμε το έδαφος, μετά θα προσθέσουμε μέσα από το Paint Details το μοντέλο Grass 3. Στο μοντέλο αυτό μπορούμε επίσης να πειράξουμε τα settings για το ύψος του, το πλάτος του, το πόσο πυκνά θα εμφανίζεται κ.α. Επιλέγοντας το κατάλληλο Brush Size , opacity και target Strength πατώντας αριστερό κλικ πάνω στο terrain μας τοποθετούμε και το γρασίδι.

Για τα 8 Tiles όπου οι παίκτες δεν μπορούν να προχωρήσουν θα χρησιμοποιήσουμε το Terrain Paint Texture - Raise or Lower Terrain και Set Height, για να σκάψουμε δύο λάκκους και με την βοήθεια ενός Shader θα αναπαραστήσουμε δύο λίμνες. Για την δημιουργία του Shader αρχικά θα πρέπει να δημιουργήσουμε ένα άδειο gameobject και να το τοποθετήσουμε στους δύο λάκκους. Στην συνέχεια θα δημιουργήσουμε τον Shader και θα προσθέσουμε διάφορα στοιχεία για να κάνουμε το νερό να φαίνεται αληθοφανές(να υπάρχει αντανάκλαση και να φαίνεται τρεχούμενο)[38]. Κάποια από τα στοιχεία που χρησιμοποιούμε είναι τα Time, Multiply, Tiling and Offset, Sample Texture 2D, Lerp, Normal Strength, Screen Position, Screen Color κ.α.

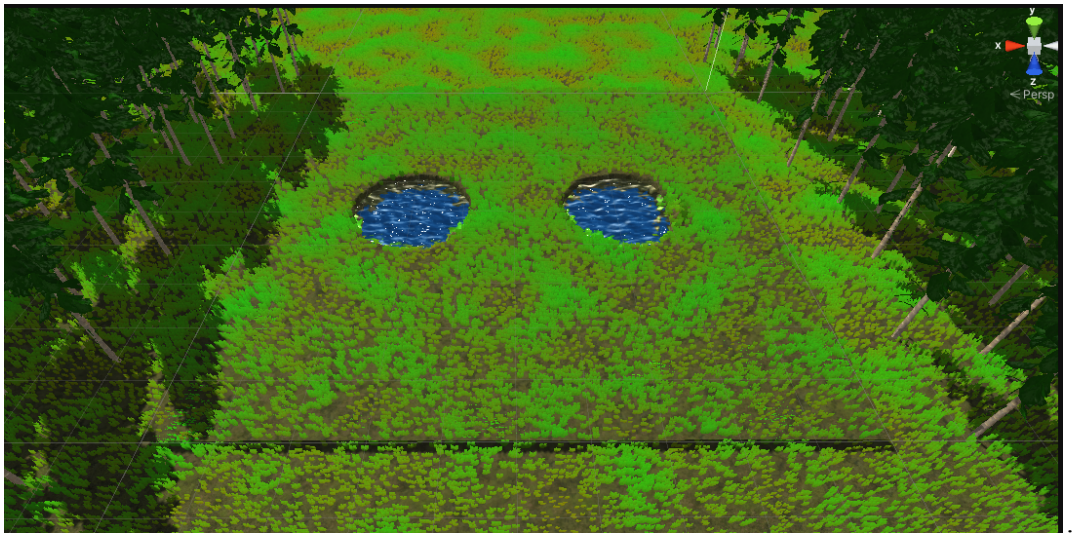


Σχήμα 4.9: Water Shader 1ο μέρος



Σχήμα 4.10: Water Shader 2ο μέρος

Τέλος θα χρησιμοποιήσουμε και το Asset Pack - Confiders[BOTD] όπου είναι και αυτό ένα free asset pack στο Asset Store της Unity. Από αυτό το pack θα πάρουμε τα 3D μοντέλα για τα δέντρα που καλύπτουν το περιβάλλον γύρω από το πεδίο μάχης. Θα πάμε πάλι στο Terrain μας και θα διαλέξουμε το Paint trees και αφού προσθέσουμε το μοντέλο δέντρου που θέλουμε και διαλέξουμε το Brush Size, Tree Density, Tree Height θα τοποθετήσουμε τα δέντρα στο περιβάλλον μας. Έχουμε πλέον τελειώσει με την επεξεργασία του Terrain μας. Το τελικό αποτέλεσμα φαίνεται στο Σχημα 4.11

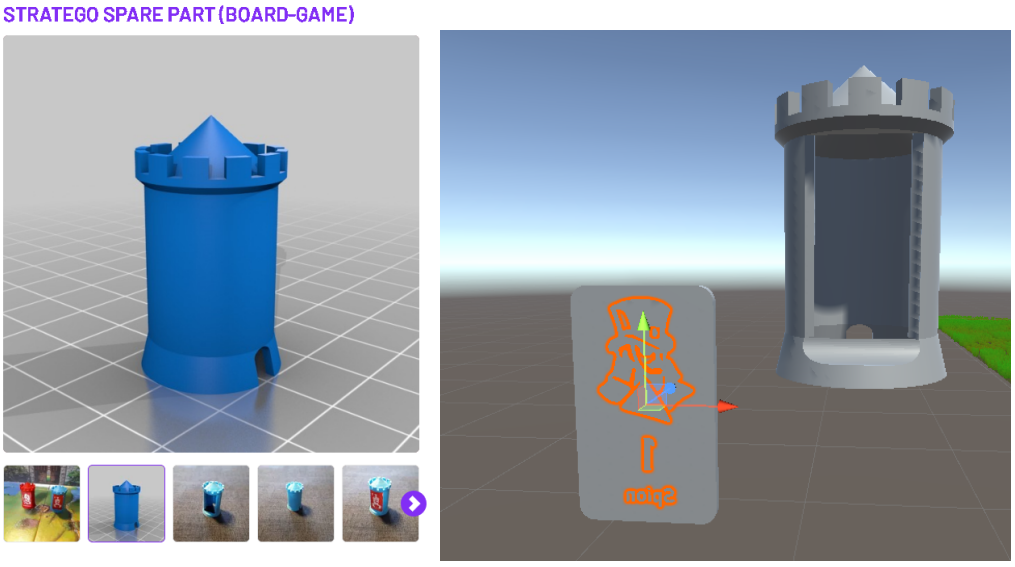


Σχήμα 4.11: Terrain τελικό αποτέλεσμα.

4.6 Πιόνια

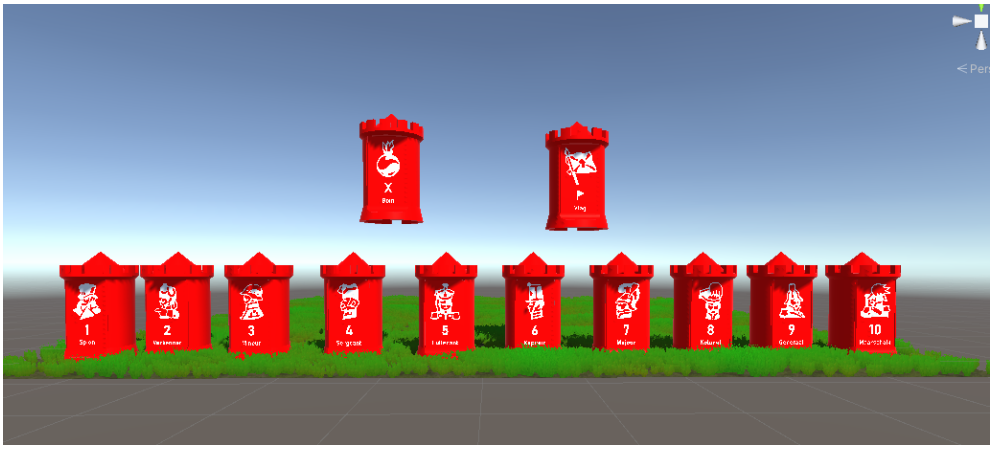
4.6.1 Τρισδιάστατα μοντέλα

Πριν ξεκινήσουμε να δημιουργούμε τα πιόνια μας θα χρειαστούμε να βρούμε 3D μοντέλα για αυτά. Τα 3D μοντέλα που χρησιμοποιούμε για την εργασία αυτή είναι 2 διαφορετικά πακέτα 3D μοντέλων που είναι δωρεάν και η αρχική τους χρήση ήτανε για 3D printing. Τα πακέτα αυτά επειδή η αρχική τους χρήση ήτανε για printing, ήτανε σε μορφή STL όπου για να τα χρησιμοποιήσουμε στο Unity τα μετατρέψαμε σε FBX μέσα από το πρόγραμμα Blender. Το πρώτο πακέτο περιέχει όλες τις κάρτες με τα ονόματα, τις κατατάξεις και τα πρόσωπα όλων των πιονιών[39], ενώ το δεύτερο πακέτο περιέχει τον τρισδιάστατο πύργο που θα χρησιμοποιήσουμε που θα σαν καλούπι του πιονιού[40].



Σχήμα 4.12: 3D Tower-Card models

Στην συνέχεια θα δημιουργήσουμε δύο καινούργια materials όπου θα τους δώσουμε τα αντίστοιχα χρώματα μπλε και κόκκινα και θα τα αναθέσουμε στον κάθε στρατό. Για τα πρόσωπα και τα ονόματα των καρτών θα χρησιμοποιήσουμε ένα τρίτο material που θα του δώσουμε το χρώμα άσπρο, για να είναι πιο διακριτά από τους παίκτες. Αφού προσθέσουμε τα μοντέλα αυτά στο πρότζεκτ μας, είμαστε έτοιμοι να δημιουργήσουμε τα πιόνια μας.



Σχήμα 4.13: Πιόνια του κόκκινου στρατού

4.6.2 Spawn Pieces

Έχοντας πλέον τα μοντέλα μέσα στο πρότζεκτ μας, μπορούμε να ξεκινήσουμε να δημιουργούμε τα πιόνια μας με την έναρξη του παιχνιδιού. Αρχικά θα δημιουργήσουμε ένα νέο script αρχείο που θα το ονομάσουμε "Pieces.cs". Το scriptaki αυτό θα το χρησιμοποιήσουμε για να αναθέτουμε σε κάθε ένα πιόνι σε ποια ομάδα ανήκει, ποια είναι η κατάταξη του, ποιες θα είναι οι κινήσεις που θα μπορεί να κάνει(π.χ. αν είναι ανιχνευτής ή όχι) και τέλος τις συντεταγμένες του. Για να πάρουμε τον τύπο του πιονιού θα δημιουργήσουμε ένα Public Enum πίνακα όπου θα περιέχει μέσα όλους τους τύπους που υπάρχουν σε αριθμητική σειρά. Αυτό θα το χρειαστούμε για να συγκρίνουμε τα πιόνια μεταξύ τους όταν θα υπάρχει σύγκρουση ανάμεσα τους. Στην συνέχεια θα δημιουργήσουμε την μέθοδο "SpawnSinglePiece()" όπου θα δημιουργεί το κάθε ένα πιόνι ξεχωριστά[41].

```
private Pieces SpawnSinglePiece(PieceType type, int team)
{
      Pieces p = Instantiate(prefabs[(int)type - 1],
      transform).GetComponent<Pieces>();
      p.type = type;
      p.team = team;
      p.GetComponent<MeshRenderer>().material = teamMaterials[team];

      p.transform.GetChild(0).GetComponent<MeshRenderer>().material =
      teamMaterials[team];

      return p;
}
```

Σχήμα 4.14: Κώδικας της μεθόδου SpawnSinglePiece

Αρχικά η μέθοδος αυτή χρησιμοποιεί το p που είναι τύπου Pieces και κάνει Instantiate δηλαδή δημιουργεί εκείνη την στιγμή το 3D μοντέλο που προαναφέραμε, ανάλογα με ποιον τύπο έχει το πιόνι. Υπάρχουν στο σύνολο 13 τέτοια prefabs όπου τα 12 από αυτά είναι για τους τύπους των πιονιών, ενώ το τελευταίο είναι το "Broken Tower Model" όπου θα το χρησιμοποιήσουμε στην συνέχεια όταν κάποιο πιόνι χάνει και εξαφανίζεται από το πεδίο μάχης. Στην συνέχεια αναθέτουμε τον τύπο και την ομάδα του πιονιού και ανάλογα με το ποια ομάδα είναι αναθέτουμε και το αντίστοιχο χρώμα material. Τέλος επιστρέφουμε το p. Αφού καταφέραμε να φτιάξουμε μία γενική μέθοδο για την δημιουργία ενός πιονιού, πλέον πρέπει να καλέσουμε αυτή την μέθοδο για κάθε ένα

πιόνι που θα υπάρχει πάνω στο board. Γι Αυτό θα δημιουργήσουμε την μέθοδο “SpawnAllPieces()” όπου θα καλείται και αυτή στην μέθοδο Awake() αμέσως μετά την μέθοδο GenerateTiles(). Επειδή στο παιχνίδι θα υπάρχει ένας παίκτης και ένα BOT θα χρειαστούμε να βάλουμε τα δικά μας πιόνια σε σειρά, για να μπορέσουμε στην συνέχεια να τα τοποθετήσουμε όπως θέλουμε, ενώ τα πιόνια από το BOT θα πρέπει να τοποθετηθούν εξαρχής σε μία τυχαίες σχετικά θέσεις καθώς δεν διαλέγει πως θα τα τοποθετήσει για κάθε ένα παιχνίδι. Πρώτα θα ασχοληθούμε με την δημιουργία και την τοποθέτηση του δικού μας στρατού δηλαδή του κόκκινου. Αρχικά θα δημιουργήσουμε έναν νέο πίνακα τύπου Pieces 10x10 που θα κρατάει τις θέσεις του κάθε πιονιού. Στην συνέχεια θα τοποθετήσουμε τα πιόνια από μικρότερα σε μεγαλύτερα σε σειρά όπως φαίνεται στο Σχήμα 4.15.

```
private void SpawnAllPieces()
    {
        pieces = new Pieces[TILE_COUNT_X,TILE_COUNT_Y];
        int redTeam = 0;
        int blueTeam = 1;

        //Spawn the Red pieces in order
        pieces[0,3] = SpawnSinglePiece(PieceType.Spy, redTeam);

        for(int j = 1; j < 9; j++)
        {
            pieces[j,3] = SpawnSinglePiece(PieceType.Scout, redTeam);
        }
        pieces[9,3] = SpawnSinglePiece(PieceType.Miner, redTeam);

        for(int j = 0; j < 4; j++)
        {
            pieces[j,2] = SpawnSinglePiece(PieceType.Miner, redTeam);
        }
        for(int j = 4; j < 8; j++)
        {
            pieces[j,2] = SpawnSinglePiece(PieceType.Sergeant,
redTeam);
        }
        pieces[8,2] = SpawnSinglePiece(PieceType.Lieutenant,
redTeam);
        pieces[9,2] = SpawnSinglePiece(PieceType.Lieutenant,
redTeam);
        pieces[0,1] = SpawnSinglePiece(PieceType.Lieutenant,
redTeam);
```

```
        pieces[1,1] = SpawnSinglePiece(PieceType.Lieutenant,
redTeam);
    }
```

Σχήμα 4.15: Κώδικας της μεθόδου SpawnAllPieces

Δημιουργούμε το πρώτο πιόνι που είναι ο κατάσκοπος στην θέση 0,3 όπου είναι και η πρώτη στο στρατόπεδο μας. Αμέσως μετά θα δημιουργήσουμε τους 8 ανιχνευτές στις επόμενες θέσεις και συνεχίζουμε μέχρι να γεμίσει όλος ο στρατός μας. Αφού τελειώσουμε με την τοποθέτηση των δικών μας πιονιών, θα ασχοληθούμε με την τυχαία τοποθέτηση του μπλε στρατού. Για να τοποθετήσουμε τυχαία τα πιόνια θα χρειαστεί να χρησιμοποιήσουμε μία λίστα Vector2 που την ονομάζουμε occupiedPositions και θα περιέχει μέσα όλες τις συντεταγμένες που είναι ελεύθερες για την τοποθέτηση των μπλε πιονιών(στο σύνολο είναι 40 τέτοιες θέσεις). Επίσης θέλουμε να τοποθετείτε πάντα η σημαία στην τελευταία σειρά για να είναι πιο ασφαλές και ο κατάσκοπος σε μία από τις δύο τελευταίες. Οπότε θα δημιουργήσουμε μία int τυχαία μεταβλητή από το 30 μέχρι το 40 και θα πάρουμε μία τυχαία θέση μέσα από τον πίνακα occupiedPositions. Σε αυτή την θέση θα τοποθετήσουμε την σημαία μας. Κάτι παρόμοιο θα κάνουμε και για τον κατάσκοπο μόνο που θα δημιουργήσουμε την τυχαία int μεταβλητή σε εύρος 20 έως 39 θέσεις. Πλέον αφού έχουμε δημιουργήσει την σημαία και τον κατάσκοπο του μπλε στρατού στις θέσεις που θέλουμε δεν μας ενδιαφέρει που θα τοποθετηθούν τα υπόλοιπα πιόνια οπότε θα τα δημιουργήσουμε εντελώς τυχαία. Θα ακολουθήσουμε ακριβώς την ίδια μέθοδο που χρησιμοποιούμε και για την σημαία και τον κατάσκοπο, απλώς το εύρος θέσεων θα είναι από το 0 έως το 38(το μέγεθος της λίστας occupiedPositions αφού δημιουργήσαμε τα 2 πιόνια).

```
//Spawn the Blue pieces random
//Spawn Flag only on the last line
int randomIndex = Random.Range(29, occupiedPositions.Count);
Vector2 randomPosition = occupiedPositions[randomIndex];
occupiedPositions.RemoveAt(randomIndex);
pieces[(int)randomPosition[0],(int)randomPosition[1]] =
SpawnSinglePiece(PieceType.Flag, blueTeam);

//Spawn Spy only on the last 2 Lines
randomIndex = Random.Range(19, occupiedPositions.Count);
randomPosition = occupiedPositions[randomIndex];
occupiedPositions.RemoveAt(randomIndex);
pieces[(int)randomPosition[0],(int)randomPosition[1]] =
SpawnSinglePiece(PieceType.Spy, blueTeam);
```

Σχήμα 4.16: Κώδικας για τοποθέτηση της σημαίας και του κατασκόπου του μπλε στρατού

```
//Spawn 8 Scouts
for(int j = 0; j < 8; j++)
{
      randomIndex = Random.Range(0, occupiedPositions.Count);
      randomPosition = occupiedPositions[randomIndex];
      occupiedPositions.RemoveAt(randomIndex);
      pieces[(int)randomPosition[0],(int)randomPosition[1]] =
      SpawnSinglePiece(PieceType.Scout, blueTeam);
}
```

Σχήμα 4.17: Κώδικας για τυχαία τοποθέτηση των 8 ανιχνευτών

Έχουμε δημιουργήσει πλέον όλα τα πιόνια και των δύο παικτών στις θέσεις που θέλουμε αλλά μόνο στην θεωρία και όχι στην πράξη. Τα πιόνια στο 3D κόσμο βρίσκονται σε λάθος θέσεις από αυτές που θέλουμε. Για να διορθώσουμε τις θέσεις τους, θα χρησιμοποιήσουμε την μέθοδο "PositionSinglePiece()" όπου θα καλείται από την μέθοδο "PositionAllPieces()" για κάθε ένα πιόνι και θα το τοποθετεί στις συντεταγμένες που του αναθέσαμε[41].

```
private void PositionSinglePiece(int x, int y, bool force = false)
{
    pieces[x,y].currentX = x;
    pieces[x,y].currentY = y;
    pieces[x,y].SetPosition(GetTileCenter(x,y), force);
    if(pieces[x,y].team == 1 && spawnBluePieces == false)
    {
        pieces[x,y].transform.Rotate(0,0,180);
    }
}
```

Σχήμα 4.18: Κώδικας της μεθόδου PositionSinglePiece

Με την βοήθεια δύο For Loop για κάθε x και y θα καλούμε την μέθοδο PositionSinglePiece και όπως βλέπουμε στο Σχήμα 4.18 θα του αναθέτουμε τις συντεταγμένες του. Ο λόγος που χρησιμοποιούμε για τις συντεταγμένες x και y το +0.5f, είναι επειδή τα tiles έχουν μέγεθος 1x1 και

θέλουμε να τοποθετηθούν στο κέντρο από τα tiles αυτά. Τέλος επειδή εμείς θα παίζουμε πάντα με τον κόκκινο στρατό, θα πρέπει να γυρίσουμε τα πιόνια του αντίπαλου στρατού 180 μοίρες έτσι ώστε να μην μπορούμε να βλέπουμε τα πιόνια του την ώρα του παιχνιδιού. Θα ελέγξουμε αν το πιόνι είναι του μπλε στρατού και θα αλλάζουμε το transform του σε transform.Rotate(0,0,180).

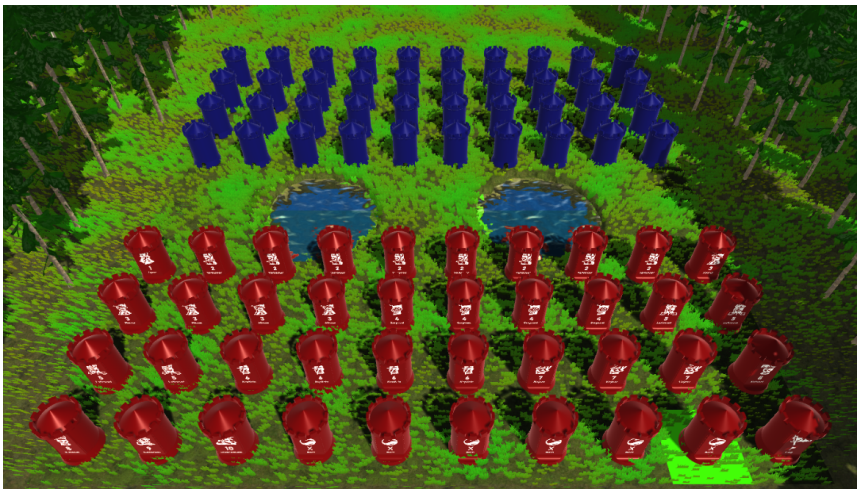
Τέλος μέσα στο Script Pieces, θα δημιουργήσουμε την μέθοδο SetPosition όπου θα την χρησιμοποιούμε κάθε φορά που κουνάμε κάποιο πιόνι. Η μέθοδος αυτή είναι αρκετά χρήσιμη, διότι θέλουμε όταν δημιουργούμε τα πιόνια στην αρχή του παιχνιδιού να τα τοποθετούμε ακαριαία στην θέση που τους αναθέσαμε. Ενώ όταν θα γίνεται μία κίνηση μέσα στο παιχνίδι(είτε μία κανονική κίνηση είτε για να στήσει το στρατόπεδο του ο παίκτης πριν ξεκινήσει το παιχνίδι), η κίνηση να γίνεται λίγο πιο αργά για φαίνεται οπτικά στον παίκτη. Οπότε μέσα στην μέθοδο SetPosition θα αναθέτουμε στην μεταβλητή desiredPosition το σημείο που θέλουμε να πάμε και στην συνέχεια αν θα θέλουμε η μεταφορά αυτή να γίνεται ακαριαία θα αναθέτουμε στην τοποθεσία του πιονιού αυτού την τιμή του desiredPosition. Τέλος μέσα στην Update μέθοδο θα αναθέτουμε στην τοποθεσία του πιονιού την θέση του με την μέθοδο της Unity Lerp ανάμεσα από τις τιμές transform.position και desiredPosition. Οπότε όταν οι τιμές αυτές είναι ίδιες η μεταφορά θα γίνεται ακαριαία, ενώ όταν δεν θα είναι ίδιες με την βοήθεια της Lerp θα φαίνεται αυτό το animation της μεταφοράς.

```
public virtual void SetPosition(Vector3 position, bool force = false)
{
      desiredPosition = position;
      if (force)
      {
            transform.position = desiredPosition;
      }
}
```

Σχήμα 4.19: Κώδικας της μεθόδου SetPosition

```
public void Update()
{
      transform.position = Vector3.Lerp(transform.position,
      desiredPosition, Time.deltaTime * 8);
}
```

Σχήμα 4.20: Κώδικας της μεθόδου Update από το script Pieces



Σχήμα 4.21: Τελικό αποτέλεσμα των πιονιών.

4.6.3 Αλληλεπίδραση με τα πιόνια

Ο τρόπος με τον οποίο θα αλληλεπιδρά ο παίκτης με τα πιόνια θα είναι με το ποντίκι του υπολογιστή. Για να το καταφέρουμε αυτό θα ελέγξουμε μέσα από την μέθοδο Update που καλείται σε κάθε frame που περνάει, πότε ο χρήστης πατάει το αριστερό κλικ του ποντικιού και την στιγμή που το πατάει σε ποιες συντεταγμένες βρίσκεται εκείνη την στιγμή, για να ξέρουμε ποιο πιόνι θα κουνήσει. Αφού ο χρήστης πατήσει το αριστερό κλικ, θα ελέγχει αν στις συντεταγμένες αυτές υπάρχει κάποιο πιόνι για να κουνήσει και αν αυτό το πιόνι είναι της κόκκινης ομάδας(δηλαδή της ομάδας του). Αν δεν υπάρχει πιόνι ή υπάρχει πιόνι της αντίπαλης ομάδας τότε δεν θα γίνεται τίποτα. Εφόσον υπάρχει δικό μας πιόνι θα ελέγχει επίσης αν είναι η δική μας σειρά για να παίξουμε. Αν ισχύουν όλα τα παραπάνω τότε θα αποθηκεύει τις συντεταγμένες στην μεταβλητή currentlyDragging και θα καλεί την μέθοδο GetMoves για να του επιστρέψει στην λίστα availableMoves όλες τις πιθανές κινήσεις που έχει το συγκεκριμένο πιόνι. Αμέσως μετά θα καλείται και η μέθοδος HighlightTiles όπου όπως λέει και το όνομα της θα κάνει highlight όλα τα Tiles που μπορεί να κινηθεί το πιόνι για να είναι πιο εμφανές οι κινήσεις στον παίκτη.

```
//If we press M1
if (Input.GetMouseButtonDown(0))
{
      if(pieces[hitPosition.x,hitPosition.y] != null &&
      pieces[hitPosition.x,hitPosition.y].team != 1)
      {
            //If is our turn
```

```
            if(turn == 0)
            {
                  currentlyDragging =
            pieces[hitPosition.x,hitPosition.y];

                  //Get Available moves
                  availableMoves = currentlyDragging.GetMoves(ref
                  pieces, TILE_COUNT_X, TILE_COUNT_Y);
                  HighlightTiles();
            }
      }
}
```

Σχήμα 4.22: Μέρος κώδικα της μεθόδου Update όταν πατάμε αριστερό κλικ

Αρχικά πάμε να δούμε πως λειτουργεί η μέθοδος GetMoves. Η μέθοδος αυτή βρίσκεται μέσα στο script Pieces και είναι μία public virtual list. Μέσα στην μέθοδο απλά δημιουργεί μία Vector2Int λίστα και την επιστρέφει. Στην συνέχεια δημιουργούμε τρία ακόμα scripts, τα BombFlag.cs, Scout.cs και Spy.cs όπου όλα τους κληρωνομούν το script Pieces άρα και την μέθοδο GetMoves. Το script BombFlag το αφήνουμε κενό καθώς αν πάμε να κουνείσουμε μία βόμβα ή την σημαία δεν θα γίνει τίποτα. Στα άλλα δύο scripts όμως θα δημιουργήσουμε την μέθοδο public override GetMoves. Η GetMoves από το script scout.cs θα δίνει όλες τις πιθανές κινήσεις που μπορεί να κάνει ο ανιχνευτής. Όπως προαναφέραμε ο ανιχνευτής μπορεί να κουνηθεί όσα τετράγωνα θέλει πάνω, κάτω, δεξιά, αριστερά μέχρι να βρεθεί είτε έξω από τον χάρτη , είτε σε κάποιο άλλο πιόνι. Οπότε μέσα σε ένα for loop θα ελέγχει αρχικά προς όλες τις κατευθύνσεις όλα τα τετραγωνάκια που μπορεί να κινηθεί και θα τα προσθέτει στην λίστα Vector2Int που δημιουργήσαμε. Όταν το πιόνι συναντήσει είτε κάποιο σύνορο του χάρτη είτε κάποιο πιόνι θα σταματάει και θα βγαίνει από το loop.

```
//Down direction
for(int i = currentY - 1; i >= 0; i--)
{
      if(Board.tiles[currentX,i].layer == 4)
      {
            break;
      }

      if(board[currentX,i] == null)
      {
            r.Add(new Vector2Int(currentX,i));
```

```
        }
                if(board[currentX,i] != null)
        {
                if(board[currentX,i].team != team)
        {
                r.Add(new Vector2Int(currentX,i));
        }
                break;
        }
 }
```

Σχήμα 4.23: Κώδικας της μεθόδου GetMoves από το Scout.cs

Στο σχήμα 4.21 βλέπουμε ότι αρχικά ελέγχει αν το τετραγωνάκι που θα κινηθούμε είναι ίσο με Layer 4, όπου το layer 4 είναι το νερό του χάρτη και αν ισχύει βγαίνει από το loop. Στην συνέχεια αν το τετραγωνάκι είναι άδειο(ίσο με null) τότε προσθέτει στην λίστα την διαθέσιμη κίνηση. Τέλος αν δεν είναι null δηλαδή αν υπάρχει κάποιο άλλο πιόνι, ελέγχει αν το πιόνι είναι από την αντίπαλη ομάδα όπου και δίνει την δυνατότητα να το επιτεθεί και μετά βγαίνει από το loop καθώς δεν θα μπορέσει να πάει παρακάτω. Αυτός ο κώδικας δίνει όλες τις διαθέσιμες κινήσεις προς τα κάτω για τον ανιχνευτή μας. Παρόμοιος κώδικας χρησιμοποιείται και για τις άλλες τρεις κατευθύνσεις. Αφού πάρουμε όλες τις διαθέσιμες κινήσεις για όλες τις κατευθύνσεις και τις προσθέσουμε στην λίστα Vector2Int, τις επιστρέφουμε στο αρχικό μας script Board.cs και καλούμε την μέθοδο HighlightTiles για να τις κάνουμε highlight. Πλέον είναι αρκετά απλό το να υπογραμμίσουμε όλες τι διαθέσιμες κινήσεις καθώς βρίσκονται όλες μέσα σε μία λίστα. Το μόνο που θα χρειαστεί να κάνουμε είναι να αλλάξουμε το Layer σε όλα τα Tiles που βρίσκονται μέσα στην λίστα αυτή με την βοήθεια ενός for loop.

```
private void HighlightTiles()
{
      if(preGame){
            for(int i = 0; i < availableMoves.Count; i++)
            {
                  tiles[availableMoves[i].x,
                  availableMoves[i].y].layer =
                  LayerMask.NameToLayer("Highlight");
            }
      }
}
```

Σχήμα 4.24: Κώδικας της μεθόδου HighlightTiles

Η boolean μεταβλητή preGame την χρησιμοποιούμε για να ξέρουμε αν έχει ξεκινήσει το παιχνίδι, διότι πριν την έναρξή του δεν θέλουμε να γίνονται highlight τα Tiles αφού δεν μετακινούμε τα πιόνια αλλά απλώς τα αλλάζουμε μεταξύ τους για να στήσουμε τον στρατό μας. Άρα αφού πατήσει ο παίκτης το αριστερό κλικ πάνω σε κάποιο δικό του πιόνι πλέον πέρνουμε όλες τις διαθέσιμες κινήσεις και τις κάνουμε highlight για να φαίνονται πιο εύκολα σε αυτόν. Πάμε τώρα να δούμε τι θα γίνεται όταν αφήνουμε το αριστερό κλικ.

Αρχικά το πρώτο πράγμα που θέλουμε να ελέγξουμε είναι να δούμε αν βρισκόμαστε στο στάδιο που στέλνουμε τον στρατό μας(δηλαδή το preGame) ή αν το παιχνίδι έχει ήδη ξεκινήσει.Αν βρισκόμαστε στο στάδιο που το παιχνίδι έχει ήδη ξεκινήσει τότε όταν ο παίκτης αφήσει το αριστερό κλικ του ποντικιού, θα καλείται η μέθοδος MoveTo όπου είναι μία μέθοδος boolean και επιστρέφει true ή false αναλόγως αν η κίνηση που θέλουμε να κάνουμε είναι έγκυρη ή όχι. Ενώ αν βρισκόμαστε στο αρχικό στάδιο του παιχνιδιού τότε θα καλείται η μέθοδος Swap αντί της MoveTo.

```
//If we release M1
if (currentlyDragging != null && Input.GetMouseButtonUp(0) &&
preGame)
{
      Vector2Int prevPosition = new
      Vector2Int(currentlyDragging.currentX,currentlyDragging.currentY
      );

      bool validMove = MoveTo(currentlyDragging, hitPosition.x,
      hitPosition.y);
      if (!validMove)
      {

            currentlyDragging.SetPosition(GetTileCenter(prevPosition.x
            , prevPosition.y));
       }

       currentlyDragging = null;
       RemoveHighlightTiles();

}
else if(currentlyDragging != null && Input.GetMouseButtonUp(0))
{
```

```
        Vector2Int prevPosition = new
        Vector2Int(currentlyDragging.currentX,currentlyDragging.currentY
        );

        bool validSwap = Swap(currentlyDragging, hitPosition.x,
        hitPosition.y);
        if (!validSwap)
        {

              currentlyDragging.SetPosition(GetTileCenter(prevPosition.x
              , prevPosition.y));
         }

         currentlyDragging = null;
         RemoveHighlightTiles();
}
```

Σχήμα 4.25: Μέρος κώδικα της μεθόδου Update όταν αφήνουμε το αριστερό κλικ

4.6.3.1 Move pawns

Έστω ότι βρισκόμαστε στο στάδιο όπου έχει ξεκινήσει το παιχνίδι. Για να είναι έγκυρη η κίνηση που θέλουμε να κάνουμε, πρέπει να ελέγξουμε κάποιους παράγοντες πάλι. Αρχικά πρέπει να ελέγξουμε αν είναι η σειρά μας και αν η κίνηση που πάμε να κάνουμε βρίσκεται μέσα στην λίστα με τα availableMoves. Αν δεν ισχύει κάποιο από αυτά τότε θα επιστρέφει false. Στην συνέχεια πρέπει να ελέγξουμε αν εκεί που θέλουμε να μετακινήσουμε το πιόνι μας υπάρχει άλλο δικό μας πιόνι όπου και πάλι θα επιστρέφει false εάν υπάρχει[42]. Επίσης πρέπει να δούμε αν το πιόνι που πάμε να κουνήσουμε είναι είτε βόμβα είτε σημαία διότι κανένα από τα δύο πιόνια δεν έχει την δυνατότητα να κινηθεί.

```
private bool MoveTo(Pieces cd, int x, int y)
{
      if(turn == 0){
            if(!ContainsValidMoves(ref availableMoves, new
      Vector2(x,y)))
            {
                  return false;
```

```
            }
      }

      Vector2Int prevPosition = new
Vector2Int(cd.currentX,cd.currentY);

      Pieces p = pieces[x,y];

      //Checking if there is another piece
      if(pieces[x,y] != null)
      {

            if(p.team == cd.team)
            {
                  return false;
            }
      }

      //Checking if flag or bomb, then dont move
      if(pieces[prevPosition.x,prevPosition.y].type == PieceType.Flag
      || pieces[prevPosition.x,prevPosition.y].type == PieceType.Bomb)
      {
            return false;
      }
```

Σχήμα 4.26: Μέρος κώδικα της μεθόδου MoveTo 1

Μία ιδιαιτερότητα που πρέπει να ελέγξουμε είναι αν το πιόνι που πάμε να κουνήσουμε είναι ο κατάσκοπος και επιτίθεται στον Marshal(αριθμός κατάταξης 10) όπου αν ισχύει τότε ο κατάσκοπος κερδίζει και αφαιρείται ο Marshal από το πεδίο μάχης. Μία δεύτερη ιδιαιτερότητα είναι αν κάνουμε επίθεση με κάποιον Miner σε μία βόμβα, όπου και πάλι η βόμβα χάνει και αφαιρείται από το πεδίο μάχης. Τέλος αφού τελειώσαμε με όλες τις ιδιαιτερότητες το μόνο που έμεινε είναι να ελέγξουμε την ιεραρχία των πιονιών. Δηλαδή αν το πιόνι που κινούμαι είναι μεγαλύτερο, μικρότερο ή ίσο με αυτό που κάνουμε επίθεση. Αν είναι μεγαλύτερο τότε το αντίπαλο αφαιρείται και του πέρνουμε την θέση, αν είναι μικρότερο τότε αφαιρείται το δικό μας και δεν κουνιέται κανένα πιόνι και τέλος αν είναι ίδια τότε αφαιρούνται και τα δύο από το πεδίο μάχης.

```
//Checking if Spy attacks Marshall
if(p != null && pieces[prevPosition.x,prevPosition.y].type ==
PieceType.Spy && p.type == PieceType.Marshall)
```

```
{
      SpawnBrokenTowerAttacking(cd.currentX ,cd.currentY ,x ,y);
      p.gameObject.SetActive(false);
      p = null;
      attack = true;
}

//Checking if Miner attacks Bomb
if(p != null && pieces[prevPosition.x,prevPosition.y].type ==
PieceType.Miner && p.type == PieceType.Bomb)
{
      SpawnBrokenTowerAttacking(cd.currentX ,cd.currentY ,x ,y);
      p.gameObject.SetActive(false);
      p = null;
      attack = true;
}
```

Σχήμα 4.27: Μέρος κώδικα της μεθόδου MoveTo 2

```
//Checking the hierarchy of the pieces
if(p != null && pieces[prevPosition.x,prevPosition.y].type > p.type)
{
      SpawnBrokenTowerAttacking(cd.currentX ,cd.currentY ,x ,y);
      p.gameObject.SetActive(false);
      p = null;
      attack = true;
}
else if(p != null && pieces[prevPosition.x,prevPosition.y].type <
p.type)
{
      MovePawnLog(cd,x,y,true);

      cd.SetPosition(GetTileCenter(x,y));
      StartCoroutine(WaitToRemoveBothPieces(x ,y ,cd.currentX
,cd.currentY , false));
      SpawnBrokenTowers(x,y,cd.currentX,cd.currentY,false);

      QueueThreeMovesDraw(x,y,prevPosition.x, prevPosition.y);
      if(turn == 1){
```

```
            turn = 0;
      }else if(turn == 0){
            turn = 1;
      }
      turnTimer = 0;
      OutOfMoves();
      return true;
}
//Checking if the pieces have the same value
else if(p != null && pieces[prevPosition.x,prevPosition.y].type ==
p.type)
{
      MovePawnLog(cd,x,y,true);

      cd.SetPosition(GetTileCenter(x,y));
      StartCoroutine(WaitToRemoveBothPieces(cd.currentX ,cd.currentY
,x ,y , true));
      SpawnBrokenTowers(x,y,cd.currentX,cd.currentY,true);

      QueueThreeMovesDraw(x,y,prevPosition.x, prevPosition.y);
      if(turn == 1){
            turn = 0;
      }else if(turn == 0){
            turn = 1;
      }
      turnTimer = 0;
      OutOfMoves();
      return true;
}
```

Σχήμα 4.28: Μέρος κώδικα της μεθόδου MoveTo 3

Όπως βλέπουμε από τα δύο σχήματα 4.25 και 4.26, όταν κάνουμε επίθεση σε κάποιο αντίπαλο πιόνι καλείται αμέσως μετά η μέθοδος SpawnBrokenTowers και σε κάποια σημεία καλείται και η μέθοδος StartCoroutine(RemoveBrokenTower). Στην συνέχεια ελέγχει ποιανού σειρά είναι και την αλλάζει ώστε να πάρει σειρά ο άλλος παίκτης. Τέλος καλούνται και οι δύο μέθοδοι QueueThreeMovesDraw και OutOfMoves. Πάμε να δούμε πρώτα τι κάνουν οι πρώτες δύο μέθοδοι που καλούνται. Η μέθοδος SpawnBrokenTowers όπως φαίνεται και από την ονομασία της, αυτό που κάνει είναι να εμφανίζει ένα μοντέλο 3D σπασμένου πύργου. Στην ουσία αυτό που θέλουμε να κάνουμε είναι όταν κάποιο πιόνι χάνει και αποσύρετε από το πεδίο μάχης, να εμφανίζεται αυτό το μοντέλο του σπασμένου πύργου στην θέση του κανονικού πύργου. Αυτό το κάνουμε για να είναι πιο εμφανές στον παίκτη πότε καταστρέφεται ένα πιόνι και σε ποιο σημείο του χάρτη.

Όταν επιτίθεται ένα πιόνι υπάρχουν δύο διαφορετικές πιθανότητες των κατατάξεων των πιονιών (είτε να είναι ίδιας αξίας, είτε το ένα μεγαλύτερο από το άλλο). Για αυτό χρησιμοποιούμε δύο μεθόδους που κάνουν σχεδόν ολόιδια δουλειά. Η μία είναι η SpawnBrokenTowerAttacking όπου καλείται όταν το πιόνι που επιτίθεται είναι μεγαλύτερο από αυτό που δέχεται την επίθεση και η άλλη μέθοδος είναι η SpawnBrokenTowers όπου καλείται όταν τα πιόνια είναι είτε ίδιας κατάταξης είτε αυτό που επιτίθεται είναι μικρότερο από αυτό που δέχεται την επίθεση. Πρώτα θα χρειαστούμε να έχουμε τις συντεταγμένες και των δύο πιονιών μέσα στην μέθοδο, που τις περνάμε παραμετρικά. Επίσης θα περάσουμε παραμετρικά και μία boolean μεταβλητή που όταν θα της δίνουμε την τιμή true θα ξέρουμε ότι τα δύο πιόνια είναι ίδιας αξίας και θα πρέπει να δημιουργήσουμε δύο μοντέλα κατεστραμμένων πυργών αντί για ένα. Αφού πάρουμε όλες τις τιμές αυτές θα δημιουργήσουμε μία μεταβλητή Pieces και θα την αναθέτουμε το μοντέλο του broken tower στις συντεταγμένες του πρώτου πιονιού. Στην συνέχεια θα ελέγχουμε σε ποια ομάδα ανήκει το πιόνι αυτό και θα αναθέτουμε το ίδιο χρώμα και στο σπασμένο πύργο. Τέλος θα καλούμε την μέθοδο RemoveBrokenTower όπου η μέθοδος αυτή είναι τύπου IEnumerator και θα περιμένει δύο δευτερόλεπτα για να εξαφανίσει τα συντρίμμια του κατεστραμμένου πύργου από τον χάρτη. Μετά θα ελέγχουμε αν η boolean μεταβλητή brokenTowers έχει την τιμή true, για να ξέρουμε αν θα ξανά ακολουθήσουμε ακριβώς τα ίδια βήματα για το δεύτερο πιόνι(επειδή αν είναι true σημαίνει ότι καταστρέφονται και τα δύο πιόνια). Επίσης αν τα δύο πιόνια έχουν την ίδια αξία κατάταξης ή αν το πιόνι που επιτίθεται είναι μικρότερης αξίας, θα καλείται και η μέθοδος WaitToRemoveBothPieces όπου και αυτή είναι τύπου IEnumerator και θα περιμένει 0.2 δευτερόλεπτα για να αφαιρέσει τα αρχικά μοντέλα των πιονιών από τον χάρτη. Ο λόγος που χρειαζόμασταν τα 0.2 δευτερόλεπτα είναι επειδή χωρίς αυτή την καθυστέρηση, η αφαίρεση των πιονιών ήτανε ακαριαία και δεν προλάβαιναν τα πιόνια να έρθουν σε σύγκρουση οπτικά για να αντιληφθεί ο παίκτης τι ακριβώς έγινε.

```
IEnumerator RemoveBrokenTower(Pieces p)
{
      yield return new WaitForSeconds(2f);
      p.gameObject.SetActive(false);
      for(int i = 0; i < 7; i++){
          p.transform.GetChild(i).gameObject.SetActive(false);
      }
 }
```

Σχήμα 4.29: Κώδικας της μεθόδου RemoveBrokenTower

```
void SpawnBrokenTowers(int x, int y, int cx, int cy, bool
brokenTowers)
{
```

```
      Pieces p = Instantiate(prefabs[12],
      pieces[x,y].transform.position,
      pieces[x,y].transform.rotation).GetComponent<Pieces>();
      p.SetPosition(GetTileCenter(x,y),false);
      int teamBroken = 0;
      if(pieces[x,y].team == 1)
      {
            teamBroken = 0;
      }
      else if(pieces[x,y].team == 0)
      {
            teamBroken = 1;
      }
      p.team = teamBroken;
      p.GetComponent<MeshRenderer>().material =
teamMaterials[teamBroken];
      for(int i = 0; i < 7; i++)
      {

            p.transform.GetChild(i).GetComponent<MeshRenderer>().mater
            ial = teamMaterials[teamBroken];
      }
      StartCoroutine(RemoveBrokenTower(p));

      if(brokenTowers){
            Pieces p2 = Instantiate(prefabs[12],
            pieces[x,y].transform.position,
            pieces[x,y].transform.rotation).GetComponent<Pieces>();
            p2.SetPosition(GetTileCenter(x,y),false);
            if(pieces[cx,cy].team == 1)
            {
                  teamBroken = 0;
            }
            else if(pieces[cx,cy].team == 0)
            {
                  teamBroken = 1;
            }
            p2.team = teamBroken;
            p2.GetComponent<MeshRenderer>().material =
            teamMaterials[teamBroken];
           for(int i = 0; i < 7; i++){
```

```
                p2.transform.GetChild(i).GetComponent<MeshRenderer>(
                ).material  = teamMaterials[teamBroken];
            }
            StartCoroutine(RemoveBrokenTower(p2));
        }
}
```

Σχήμα 4.30: Κώδικας της μεθόδου SpawnBrokenTowers

```
IEnumerator WaitToRemoveBothPieces(int x ,int y ,int cx ,int cy ,bool
removePieces)
{
      yield return new WaitForSeconds(0.2f);
      pieces[cx,cy].gameObject.SetActive(false);
      pieces[cx,cy] = null;

      if(removePieces){
          pieces[x,y].gameObject.SetActive(false);
          pieces[x,y] = null;
      }
}
```

Σχήμα 4.31: Κώδικας της μεθόδου WaitToRemoveBothPieces

Πάμε τώρα να δούμε τις επόμενες δύο μεθόδους που καλούνται αμέσως μετά την κίνηση ενός πιονιού. Η πρώτη είναι η μέθοδος QueueThreeMovesDraw και με αυτή θα ελέγχουμε αν ο παίκτης που μόλις κούνησε το πιόνι του, έκανε τρεις ίδιες συνεχόμενες κινήσεις όπου και απαγορεύεται και θα χάνει το παιχνίδι.Ο τρόπος που θα υλοποιήσουμε αυτό τον έλεγχο είναι δημιουργώντας δύο λίστες Vector2Int και αποθηκεύοντας εκεί μέσα τις τέσσερις τελευταίες κινήσεις του κάθε παίκτη. Στην συνέχεια θα ελέγχουμε αν η πρώτη κίνηση τις λίστας είναι ίδια με την τρίτη και αν η δεύτερη είναι ίδια με την τέταρτη. Αν ισχύει κάτι τέτοιο τότε το παιχνίδι θα σταματάει και ο παίκτης που έκανε τρεις ίδιες κινήσεις θα χάνει το παιχνίδι[43].

```
void QueueThreeMovesDraw(int x, int y, int cx, int cy)
{
    if(pieces[cx,cy].team == 0 && pieces[cx,cy] != null){
        threeMovesDraw.Add(new Vector2Int(cx,cy));
        threeMovesDraw.Add(new Vector2Int(x,y));
        if(threeMovesDraw.Count >= 6){
            if(threeMovesDraw[1] == threeMovesDraw[2] &&
            threeMovesDraw[2] == threeMovesDraw[5])
             {
                   if(threeMovesDraw[0] == threeMovesDraw[3] &&
                   threeMovesDraw[3] == threeMovesDraw[4])
                   {
                         DeathPanel();
                         deathText.text = "You Lost!!!";
                         Debug.Log("GAME ENDS RED LOSES");
                  }
             }
        }
        if(threeMovesDraw.Count >= 6)
        {
            threeMovesDraw.RemoveAt(0);
            threeMovesDraw.RemoveAt(0);
        }
    }
```

Σχήμα 4.32: Κώδικας της μεθόδου QueueThreeMovesDraw

Τέλος καλείται και η μέθοδος OutOfMoves. Όπως μπορούμε να καταλάβουμε και από το όνομα της, η μέθοδος αυτή θα ελέγχει αν ο παίκτης μετά την τελευταία κίνηση του δεν έχει κάποια άλλη κίνηση να παίξει, όπου και θα χάνει το παιχνίδι. Η μέθοδος αυτή είναι αρκετά απλή, το μόνο πράγμα που κάνουμε είναι να πάρουμε με την βοήθεια ενός for loop, για όλα τα πιόνια του στρατού που μας περισσεύουν, όλες τις διαθέσιμες κινήσεις και να τις αποθηκεύσουμε στην λίστα outOfMoves. Αν η λίστα αυτή περιέχει έστω και μία κίνηση για κάθε ένα πιόνι, τότε θα αυξάνουμε την μεταβλητή ofm κατά ένα. Μόλις βγούμε από το for loop θα ελέγχουμε αν η μεταβλητη αυτή είναι ίση με το 0. Αν είναι ίση με τότε σημαίνει ότι δεν υπάρχει καμία διαθέσιμη κίνηση και θα χάνει ο παίκτης αυτός.

```
void OutOfMoves()
{
        for(int i = 0; i < TILE_COUNT_X; i++)
```

```
        {
            for(int j = 0; j < TILE_COUNT_Y; j++)
            {
                if(pieces[i,j] != null && pieces[i,j].team == 0)
                {
                        outOfMoves = pieces[i,j].GetMoves(ref pieces,
                        TILE_COUNT_X, TILE_COUNT_Y);
                        if(outOfMoves.Count > 0)
                        {
                                ofm += 1;
                        }
                }
            }
        }
        if(ofm == 0)
        {
            DeathPanel();
            deathText.text = "You Lost!!!";
            Debug.Log("LITOURGEIII RED TEAM LOSES");
        }
        ofm = 0;
}
```

Σχήμα 4.33: Κώδικας της μεθόδου OutOfMoves

Μετά από όλους αυτούς τους ελέγχους, αν η boolean μέθοδος επιστρέψει true, τότε η κίνηση θα γίνει κανονικά και η μεταβλητή currentlyDragging θα πάρει την τιμή null για να είναι έτοιμη για την επόμενη κίνηση. Μετά την ολοκλήρωση της κίνησης, θα καλέσουμε την μέθοδο RemoveHighLightTiles, όπου είναι ακριβώς αντίθετη από την μέθοδο HighlightTiles. Αυτό που κάνει δηλαδή είναι να αλλάζει τα layers σε “Tile layer” από όλα τα tiles που βρίσκονται μέσα στην λίστα availableMoves. Τέλος θα καθαρίζει την λίστα availableMoves για να είναι και αυτή έτοιμη για την επόμενη κίνηση.

```
private void RemoveHighlightTiles()
{
    for(int i = 0; i < availableMoves.Count; i++)
    {
            tiles[availableMoves[i].x, availableMoves[i].y].layer =
            LayerMask.NameToLayer("Tile");
```

```
    }

    availableMoves.Clear();
}
```

Σχήμα 4.34: Κώδικας της μεθόδου RemoveHighLightTiles

4.6.3.2 Swap pawns

Πάμε τώρα να δούμε τι θα γίνεται αν βρισκόμαστε στο αρχικό στάδιο του παιχνιδιού, όπου ο παίκτης πρέπει να στήσει το στρατόπεδο του και όχι να κουνήσει και να επιτεθεί με τα πιόνια του. Όπως αναφέραμε και παραπάνω, έχουμε δημιουργήσει την boolean μεταβλητή preGame για να ξέρουμε αν βρισκόμαστε στο αρχικό στάδιο του παιχνιδιού ή όχι. Εφόσον βρισκόμαστε στο αρχικό στάδιο, θα ακολουθούμε παρόμοια διαδικασία απλά θα καλούμε την boolean μέθοδο Swap αντί της boolean μεθόδου MoveTo. Η μέθοδος Swap χρησιμοποιείται για να διαμορφώσει ο παίκτης το στρατόπεδο του πριν από την μάχη. Είναι πολύ πιο απλή από την μέθοδο της MoveTo, καθώς πλέον δεν θα χρειάζεται να ελέγχουμε όλους τους κανονισμούς του παιχνιδιού(για παράδειγμα να συγκρίνουμε τις κατατάξεις των πιονιών ή να ελέγξουμε αν κάποιος παίκτης έμεινε από κινήσεις) αλλά μόνο αν τα δύο πιόνια που επιλέξαμε είναι της ίδιας ομάδας και μπορούν να αλλάξουν θέση μεταξύ τους.

Δεν χρειάζεται να ξανά ελέγξουμε αν το πιόνι που διαλέγουμε με αριστερό κλικ είναι της δικιάς μας ομάδας και αν όντως υπάρχει κάποιο πιόνι εκεί, επειδή αυτός ο έλεγχος είναι ολόιδιος με αυτόν που κάνουμε όταν θέλουμε να μετακινήσουμε ένα πιόνι εφόσον το παιχνίδι έχει ξεκινήσει και γίνεται ήδη μέσα στην Update μέθοδο. Οπότε το μόνο που έμεινε να ελέγξουμε, είναι αν το δεύτερο πιόνι που διαλέγουμε για να αλλάξουν θέσεις μεταξύ τους(στο σημείο που αφήνουμε το αριστερό κλικ) δεν έχει την τιμή null και είναι της δικιάς μας ομάδας. Εφόσον ισχύουν και οι δύο προϋποθέσεις, αρκεί πλέον να αλλάξουμε τις τιμές των δύο πιονιών. Για να το καταφέρουμε αυτό, θα χρησιμοποιήσουμε το σημείο pieces[2,4] σαν μία τρίτη μεταβλητή, που θα αποθηκεύσει τις τιμές του ενός πιονιού για να γίνει πιο εύκολη η αλλαγή των τιμών. Ο λόγος που διαλέγουμε την θέση 2,4 από την λίστα pieces, είναι επειδή σε εκείνο το σημείο βρίσκεται η μία από τις δύο λίμνες του χάρτη μας και ο χρήστης απαγορεύεται να πατήσει εκεί πάνω, οπότε το σημείο αυτό θα είναι πάντα null και άχρηστο για εμάς. Οπότε αποθηκεύουμε τις συντεταγμένες του πρώτου πιονιού στο σημείο 2,4 και δίνουμε στο πρώτο πιόνι τις συντεταγμένες του δεύτερου πιονιού. Μετά αναθέτουμε τις συντεταγμένες του σημείου 2,4 στο δεύτερο πιόνι και τέλος ξαναδίνουμε στο σημείο 2,4 την τιμή null. Πλέον έχουν αλλάξει οι τιμές των δύο πιονιών στην λίστα μας και μένει να καλέσουμε την μέθοδο PositionSinglePiece και για τα δύο πιόνια ώστε να αλλάξουν θέση και τα δύο τρισδιάστατα μοντέλα τους.

Έχοντας υλοποιήσει πλέον και την μέθοδο της Swap, μας έχει μείνει μόνο να υλοποιήσουμε και το κώδικα για το BOT που θα παίζουμε εναντίον για να τελειώσουμε με την αλληλεπίδραση των πιονιών.

```
bool Swap(Pieces cd, int x, int y)
{
    Vector2Int prevPosition = new
Vector2Int(cd.currentX,cd.currentY);

    if(pieces[x,y] != null)
    {

        if(pieces[x,y].team != cd.team)
        {
            return false;
        }
    }else if(pieces[x,y] == null)
    {
        return false;
    }

    pieces[2,4] = pieces[x,y];
    pieces[x,y] = pieces[prevPosition.x,prevPosition.y];
    pieces[prevPosition.x,prevPosition.y] = pieces[2,4];

    pieces[2,4] = null;

    PositionSinglePiece(x,y);
    PositionSinglePiece(prevPosition.x,prevPosition.y);
    return true;
}
```

Σχήμα 4.35: Κώδικας της μεθόδου Swap

4.6.3.3 BOT playing

Ο κώδικας για το BOT βρίσκεται όλος μέσα στην μέθοδο AIPlaying όπου καλείται και αυτή μέσα από την μέθοδο Update. Αρχικά ελέγχουμε αν είναι η σειρά για να παίξει το BOT και έχουμε

και μία δεύτερη μεταβλητή την turnTimer, όπου θα πρέπει να είναι μεγαλύτερη από την τιμή 1 για να μπορέσουμε να καλέσουμε την AI Playing. Κάθε ένα δευτερόλεπτο που περνάει η τιμή της turnTimer αυξάνεται κατά ένα. Ο λόγος που χρειαζόμαστε αυτήν την καθυστέρηση του ενός δευτερολέπτου για να παίξει το BOT, είναι επειδή όταν κάνουμε εμείς μία κίνηση, πριν καν τελειώσει η δικιά μας κίνηση το BOT παίζει πολύ γρήγορα την δικιά του. Έτσι οπτικά φαίνεται η ψευδαίσθηση ότι κινούνται δύο πιόνια σχεδόν ταυτόχρονα, όπου και δεν ισχύει. Για αυτό τον λόγο χρησιμοποιούμε την καθυστέρηση αυτή για να είναι οπτικά πιο όμορφο το παιχνίδι μας.

```
//AI PROGRAMMING
if(turn == 1 && turnTimer >= 1)
{
      AIPlaying();
}
```

Σχήμα 4.36: Μέρος κώδικα της μεθόου Update για να καλέσουμε την AIPlaying

Το μέρος του κώδικα που αυξάνει την μεταβλητή turnTimer βρίσκεται μέσα στην μέθοδο Update και η μέθοδος Update καλείται για κάθε ένα Frame που περνάει, αυτό σημαίνει ότι η μεταβλητή θα φτάνει την τιμή ένα μέσα σε πολύ λιγότερο χρόνο από το ένα δευτερόλοπτο που θέλουμε. Για αυτόν τον λόγο χρησιμοποιούμε το Time.deltaTime για να επηρεάζεται η μεταβλητή σε κάθε δευτερόλεπτο που περνάει και όχι κάθε frame.

Πάμε να δούμε τι ακριβώς γίνεται μέσα στην μέθοδο AIPlaying. Ο τρόπος με τον οποίο θα πέζει το BOT είναι αρκετά απλός καθώς θέλουμε να φτιάξουμε ένα εύκολο αντίπαλο για το παιχνίδι μας. Αρχικά θα χρειαστούμε μία λίστα Vector4 που θα αποθηκεύει όλες τις πιθανές κινήσεις που θα έχει το BOT σε κάθε γύρο. Ο λόγος που η λίστα αυτή θα πρέπει να είναι Vector4, είναι επειδή στην λίστα αυτή θα αποθηκεύουμε κινήσεις(δηλαδή τα προηγούμενα x,y και τα καινούργια x,y). Στην συνέχεια θα ελέγχει για όλα τα πιόνια που του έχουν απομείνει, οποιαδήποτε κίνηση μπορεί να κάνει το καθένα με την βοήθεια της GetMovesAI όπου είναι σχεδόν ολόιδια με την GetMoves που είδαμε παραπάνω και θα προσθέτει όλες αυτές τις κινήσεις στην λίστα που δημιουργήσαμε. Αμέσως μετά θα ελέγχουμε αν η λίστα αυτή είναι άδεια και αν όντως είναι θα σταματάει το παιχνίδι καθώς δεν θα έχει το BOT καμία κίνηση να παίξει. Εφόσον δεν θα είναι άδεια, θα περιέχει δηλαδή τουλάχιστον μία κίνηση, θα δημιουργεί μία τυχαία μεταβλητή int όπου θα έχει εύρος από το 0 έως το μέγεθος της λίστας αυτής. Στην συνέχεια θα χρησιμοποιούμε το νούμερο αυτό για να πάρουμε μία κίνηση μέσα από την λίστα που περιέχει όλες τις διαθέσιμες κινήσεις. Εφόσον η κίνηση δεν είναι παράνομη θα υλοποιεί την κίνηση αυτή και θα τοποθετεί και τα πιόνια στις σωστές συντεταγμένες τους. Τέλος θα αδειάζει την λίστα Vector 4 για να είναι έτοιμη για τον επόμενη φορά που θα παίξει το BOT.

```
void AIPlaying()
{
    for(int i = 0; i < 10; i++)
```

```
        {
            for(int j = 0; j < 10; j++)
            {
                if(pieces[i,j] != null && pieces[i,j].team == 1)
                {

                    availableMovesAI.AddRange(pieces[i,j].GetMovesA
                    I(ref pieces, TILE_COUNT_X, TILE_COUNT_Y));
                }
            }
        }

        if (availableMovesAI.Count > 0)
        {
            int randomMove = Random.Range(0, availableMovesAI.Count);
            Vector4 randomPos = availableMovesAI[randomMove];
           bool validMoveAI =
           MoveTo(pieces[(int)randomPos[2],(int)randomPos[3]],
           (int)randomPos[0], (int)randomPos[1]);
            if (!validMoveAI)
            {

                pieces[(int)randomPos[2],(int)randomPos[3]].SetPosit
                ion(GetTileCenter((int)randomPos[2],(int)randomPos[3
                ]));
            }

        }
        else
        {
            DeathPanel();
            deathText.text = "You Won!!!";
            Debug.Log("Red Wins");
        }
        availableMovesAI.Clear();
}
```

Σχήμα 4.37: Κώδικας της μεθόδου AIPlaying

4.6.3.4 Κίνηση έξω από το board

Όλες οι παραπάνω κινήσεις που είδαμε ότι μπορεί να κάνει ο παίκτης αλλά και το BOT γίνονται εφόσον βρίσκεται το ποντίκι μας πάνω σε ένα από τα τρία Layers: Tile, Hover και Highlight. Πρέπει να δούμε τι θα γίνεται αν το ποντίκι μας βρεθεί κάπου εκτός του χάρτη, δηλαδή σε κάποιο σημείο που το Layer mask θα είναι διαφορετικό από τα παραπάνω. Συνήθως όταν το ποντίκι μας βρίσκεται εκτός από τον board μας θα του αναθέτουμε την τιμή -Vector2Int.one όπου είναι η τιμή (-1,-1). Αν όμως το ποντίκι μας βρισκόταν πρώτα μέσα στο board και μετά μετακινήθηκε κάπου έξω από το board, θα πρέπει να ελέγξουμε αν η μεταβλητή currentHover έχει διαφορετική τιμή από την -Vector2Int.one. Στην συνέχεια θα ελένξουμε αν υπάρχει κάποια διαθέσιμη κίνηση μέσα στην λίστα availableMoves, για τις συντεταγμένες της μεταβλητής currentHover. Εάν υπάρχει τότε θα κάνουμε Highlight την κίνηση αυτή. Αλλιώς θα αφαιρούμε τα higlighted tiles. Τέλος θα αναθέτουμε την τιμή -Vector2Int.one στην μεταβλητή currentHover. Τέλος θα ελέγξουμε αν η μεταβλητή currentlyDragging δεν είναι null(δηλαδή έχουμε κάποιο πιόνι στο ποντίκι μας) και εάν αφήσουμε το αριστερό κλικ κάπου έξω από το board και θα τοποθετούμε το πιόνι αυτό με την μέθοδο SetPosition στις συντεταγμένες που είχε εξαρχής. Μετά θα αναθέτουμε στο currentlyDragging την τιμή null και θα καλούμε την μέθοδο RemoveHighlightTiles.

```
else //When the Mouse is out of the board
{
      if (currentHover != -Vector2Int.one)
      {
            if(ContainsValidMoves(ref availableMoves, currentHover))
            {
                  tiles[currentHover.x, currentHover.y].layer =
                  LayerMask.NameToLayer("Highlight");
            }
            else
            {

                  tiles[currentHover.x, currentHover.y].layer =
                  LayerMask.NameToLayer("Tile");
            }
                  currentHover = -Vector2Int.one;
       }

       if(currentlyDragging && Input.GetMouseButtonUp(0))
       {
currentlyDragging.SetPosition(GetTileCenter(currentlyDragging.current
X, currentlyDragging.currentY));
                    currentlyDragging = null;
            RemoveHighlightTiles();
       }
```

```
}
```

Σχήμα 4.38: Μέρος κώδικα της μεθόδου Update όταν το ποντίκι βρίσκεται εκτός του Board

4.6.4 Move Pawn Log

Έχοντας πλέον υλοποιήσει όλες τις κινήσεις των πιονιών, θα χρειαστούμε να κρατάμε ένα TextMeshPro ως αρχείο καταγραφής των κινήσεων των πιονιών. Ο λόγος που χρειαζόμαστε το αρχείο καταγραφής είναι επειδή τα πιόνια βρίσκονται γυρισμένα από τον αντίπαλο, οπότε δεν είναι δυνατόν ο παίκτης να ξέρει την κατάταξη του αντίπαλου όταν αυτός κάνει επίθεση σε κάποιο δικό μας. Για αυτό τον λόγο το αρχείο καταγραφής θα καταγράφει τις τελευταίες 12 κινήσεις. Θα αποθηκεύει τις παλιές συντεταγμένες του πιονιού αλλά και τις καινούργιες μετά την ολοκλήρωση της κίνησης και σε περίπτωση που κάποιο πιόνι επιτίθεται κάποιο άλλο, θα καταγράφει και τις κατατάξεις των δύο πιονιών αυτών. Αρχικά θα χρειαστούμε δύο TextMeshPro - Text, ένα που θα καταγράφει τις κινήσεις της κόκκινης ομάδας και ένα για τις κινήσεις της μπλε ομάδας. Ο λόγος που χρησιμοποιούμε δύο Text αντί για ένα, είναι επειδή είναι πιο εύκολη η χρήση τους καθώς θα αλλάζουμε το χρώμα ως προς το χρώμα της ομάδας που κάνει την κίνηση, για να γίνεται πιο εύκολη η παρακολούθηση του αρχείου. Έτσι το ένα αρχείο θα έχει μόνιμα το μπλε χρώμα, ενώ το άλλο θα έχει μόνιμα το κόκκινο χρώμα στα γράμματα του.

Αρχικά μέσα στο Script Board θα δημιουργήσουμε την μέθοδο MovePawnLog, όπου θα πάρει παραμετρικά τις συντεταγμένες του πιονιού που κινείται, τις νέες συντεταγμένες και την boolean μεταβλητή attack που θα μας χρησιμεύσει για να ελέγξουμε εάν το πιόνι επιτίθεται ή όχι. Θα χρειαστούμε δύο λίστες String μία για την μπλε ομάδα και μία για την κόκκινη. Πρώτα θα ελέγχει ποιος παίκτης παίζει, για να ξέρουμε σε ποια λίστα θα προσθέσουμε την κίνηση. Στην συνέχεια θα ελέγχουμε εάν η μεταβλητή attack έχει την τιμή true, όπου εάν ισχύει θα αποθηκεύουμε την κίνηση αλλά και τους δύο τύπους των πιονιών στην λίστα του String. Αμέσως μετά θα καθαρίζουμε την λίστα και τυπώνουμε κάθε στοιχείο της λίστας σε διαφορετική γραμμή, και μετά από κάθε γραμμή θα αφήνουμε κενό μία άδεια γραμμή για την κίνηση της αντίπαλης ομάδας. Επειδή θέλουμε να καταγράφουμε μόνο τις τελευταίες 12 κινήσεις των παικτών, θα ελέγχουμε εάν η λίστα της κόκκινης ομάδας είναι μεγαλύτερη ή ίση με 6 στοιχεία και αν ισχύει θα αφαιρούμε ένα στοιχείο από την κόκκινη λίστα και ένα από την μπλε. Έχουμε τελειώσει και με την καταγραφή των κινήσεων των πιονιών, μένει μόνο μετά από κάθε κίνηση να καλούμε την μέθοδο MovePawnLog και να δίνουμε την τιμή True στην μεταβλητή attack εάν στην κίνηση αυτή επιτίθεται σε κάποιο πιόνι. Το αρχείο καταγραφής που δημιουργήσαμε θα τα τοποθετήσουμε στο δεξιό-πάνω μέρος της οθόνης του παιχνιδιού.

```
public void MovePawnLog(Pieces cd,int x,int y, bool attack)
{
    if(turn == 0)
    {
        if(attack)
        {
            redList.Add($"X:{cd.currentX}, Y:{cd.currentY} to X:{x},
Y:{y}, {cd.type} attacking " +
$"<color=blue>{pieces[x,y].type}</color>");
        }
        else
        {
            redList.Add($"X:{cd.currentX}, Y:{cd.currentY} to X:{x},
Y:{y}");
        }
        pawnMoveText.text = "";
        if(blueList.Count >= 6)
        {
            blueList.RemoveAt(0);
            redList.RemoveAt(0);
            pawnMoveTextBlue.text = "";
            foreach (string i in blueList)
            {
                pawnMoveTextBlue.text += i + "\n\n";
            }
        }
        foreach (string i in redList)
        {
            pawnMoveText.text += i + "\n\n";
        }
    }
```

Σχήμα 4.39: Κώδικας της μεθόδου MovePawnLog

Σχήμα 4.40: Ολοκληρωμένη εικόνα του αρχείου καταγραφής

Κεφάλαιο 5ο: Ανάπτυξη του Menu - Interface

5.1 Εισαγωγή

Η ανάπτυξη του μενού ενός παιχνιδιού παίζει καθοριστικό ρόλο στην δημιουργία ενός παιχνιδιού, καθώς είναι ο τρόπος με τον οποίο ο χρήστης μπορεί να πλοηγηθεί στα χαρακτηριστικά του παιχνιδιού αυτού. Πέρα από το χαρακτηριστικό πλοήγησης, η διεπαφή μενού λειτουργεί και ως το πρώτο σημείο επαφής του χρήστη με το παιχνίδι. Μια καλοφτιαγμένη διεπαφή μενού βελτιώνει τόσο την αισθητική του παιχνιδιού όσο και την ικανοποίηση του παίκτη που θα πάρει από την εμπειρία του με το παιχνίδι. Κύρια συστατικά ενός μενού είναι ότι περιέχει βασικές λειτουργίες όπως το κουμπί έναρξης του παιχνιδιού, όταν ο παίκτης είναι έτοιμος να ξεκινήσει να παίζει, την προσαρμογή των ρυθμίσεων του παιχνιδιού, όπως των γραφικών του, της ανάλυσης του, της έντασης της μουσικής του αλλά και την επιλογή διαφόρων επιπέδων του. Ο σχεδιασμός του μενού για το κάθε παιχνίδι είναι εντελώς διαφορετικώς, καθώς θα πρέπει το μενού αυτό να συμπίπτει με το θέμα και το είδος του παιχνιδιού, διασφαλίζοντας την αισθητική πλοήγηση και περιήγηση του. Με άλλα λόγια η διεπαφή μενού λειτουργεί ως κρίσιμη γέφυρα μεταξύ του παίκτη και του παιχνιδιού, διαμορφώνοντας τη συνολική εμπειρία του και συμβάλλοντας σημαντικά στην απόλαυση του παιχνιδιού[44].

5.2 Canvas

Για να μπορέσουμε να υλοποιήσουμε το μενού μας, το πρώτο στοιχείο που θα χρειαστούμε είναι ένας καμβάς που θα περιέχει όλο το interface μας. Ο καμβάς είναι ένας component της μηχανής Unity που δίνει την δυνατότητα στους προγραμματιστές να δημιουργήσουν δυναμικά και οπτικά διεπαφές μενού. Με την βοήθεια του καμβά ο προγραμματιστής μπορεί να δημιουργήσει διαδραστικά μενού χρησιμοποιώντας τα στοιχεία UI (User Interface), όπως είναι τα κουμπιά, οι πίνακες, στοιχεία κειμένου κ.α. Επίσης ο καμβάς είναι και ο τρόπος με τον οποίο οι προγραμματιστές ανταποκρίνονται στις αλληλεπιδράσεις των παικτών για να παρέχουν την καλύτερη συναρπαστική εμπειρία ενός μενού. Ο καμβάς έχει την δυνατότητα και την ευελιξία να προσαρμόζει όλα τα στοιχεία που περιέχει, ανάλογα με τις διαφορετικές αναλύσεις και τις αναλογίες διαστάσεων του παιχνιδιού, διασφαλίζοντας ότι η διεπαφή θα είναι σωστά υλοποιημένη για κάθε συσκευή και πλατφόρμα. Για τους παραπάνω λόγους ο καμβάς είναι ένα ισχυρό εργαλείο για την βελτίωση της αισθητικής ενός παιχνιδιού[45].

5.3 Start and Quit

Εφόσον δημιουργήσουμε τον καμβά μας όπου θα περιέχει όλα τα στοιχεία του μενού μας, είμαστε έτοιμοι να ξεκινήσουμε με την υλοποίηση του μενού. Το πρώτο στοιχείο που θα χρειαστούμε για την εφαρμογή μας, θα είναι ένα κουμπί που θα αλλάζει την κατάσταση του παιχνιδιού, θα αλλάζει δηλαδή την μεταβλητή boolean PreGame που αναφέραμε στο κεφάλαιο 4ο. Όπως είδαμε στο κεφάλαιο 4 , η μεταβλητή αυτή μας χρησιμεύει για να ελέγχουμε εάν βρισκόμαστε στην αρχική φάση του παιχνιδιού, την τοποθέτηση του στρατού μας, ή εάν βρισκόμαστε στην δεύτερη φάση του παιχνιδιού, την μετακίνηση και την επίθεση των πιονιών μας.
Αρχικά θα πάμε πάνω στο καμβά που δημιουργήσαμε και θα πατήσουμε δεξί κλικ, θα επιλέξουμε την επιλογή UI και στην συνέχεια το button - TextMeshPro. Ο λόγος που θα δημιουργήσουμε button - TextMeshPro αντί για ένα απλό button, είναι επειδή το TexhMeshPro μας δίνει την δυνατότητα για την δημιουργία πιο εξελιγμένων στοιχείων διεπαφής χρήστη και πιο προηγμένων στοιχείων κειμένου. Το start button αυτό περιέχει ένα text όπου θα θέσουμε το κείμενο σε “start” και θα επιλέξουμε την κατάλληλη γραμματοσειρά και τις αντίστοιχες ιδιότητες του που θέλουμε. Στην συνέχεια θα χρησιμοποιήσουμε ένα ακόμη asset από το Unity Asset Store, για να ομορφύνουμε το μενού διεπαφής μας. Το asset αυτό είναι το “Simple button set 1” [46]. Θα επιλέξουμε στο component του button “image”, την εικόνα που θέλουμε (την button_100 από το package) και θα αλλάξουμε και τα αντίστοιχα χρώματα Highlighted Color, Pressed Color, και Normal Color για να φαίνεται η αλληλεπίδραση του χρήστη με το κουμπί(όταν ο χρήστης κάνει mouse over, ή όταν πατιέται το κουμπί). Επίσης θα χρειαστούμε και ένα TextMeshPro - text όπου θα γράφει “Setup your pieces and press Start to begin”. Οπτικά έχουμε τελειώσει με το κουμπί Start. Πάμε να υλοποιήσουμε και τον κώδικα που θα καλείται με το πάτημα του κουμπιού.

```
public void StartGame()
{
    Board.preGame = true;
    Destroy(startButton);
```

```
    setUpText.fontSize = 42;
    setUpText.text = "Game Started!!!";
    Destroy(setUpTxt,3);
}
```

Σχήμα 5.1: Κώδικας της μεθόδου StartGame

Όπως βλέπουμε και στο σχήμα 5.1, θα χρειαστούμε τρεις SerializedField μεταβλητές για το κουμπί Start. Στην πρώτη μεταβλητή θα αντιστοιχήσουμε το ίδιο το κουμπί για να μπορέσουμε να το καταστρέψουμε από το παιχνίδι όταν πατηθεί, καθώς μετά δεν θα μας είναι χρήσιμο. Για την δεύτερη και τρίτη μεταβλητή θα αντιστοιχίσουμε το GameObject SetUptxt για να μπορέσουμε να το καταστρέψουμε και αυτό, αλλά και το ίδιο το TextMeshPro για να μπορέσουμε να αλλάξουμε το περιεχόμενο του και το fontSize του. Έτσι με το που πατηθεί το κουμπί Start, θα αναθέτουμε την τιμή true στην boolean μεταβλητή preGame όπου είδαμε στο 4ο κεφάλαιο ότι την χρησιμοποιούμε για να ελέγξουμε αν βρισκόμαστε στο αρχικό στάδιο του παιχνιδιού ή όχι. Αμέσως μετά θα καταστρέψουμε το κουμπί Start και θα αλλάξουμε το fontSize αλλά και το περιεχόμενο του setUpText σε “Game Started!!!”. Τέλος θα καταστρέψουμε με χρονοκαθυστέρηση τριών δευτερολέπτων ολόκληρο το setUpText.

Για το παιχνίδι μας θα χρειαστούμε και ένα δεύτερο βασικό κουμπί, το QuitGutoon, για να κλείνει το παιχνίδι όταν ο παίκτης θελήσει να σταματήσει να παίζει. Παρόμοια θα χρησιμοποιήσουμε ένα κουμπί που θα το ονομάσουμε QuitButton και θα του αναθέτουμε την ίδια εικόνα button_100 από το package που προαναφέραμε. Επίσης θα αντιγράψουμε και τις ίδιες τιμές για το κουμπί, για την περίπτωση που θα κάνουμε mouse over, όταν πατήσουμε το κουμπί αλλά και για το normal Color, καθώς θέλουμε να δίνουν την αίσθηση ενός ίδιου θέματος για το μενού.

```
public void Quit()
{
    #if UNITY_STANDALONE
        Application.Quit();
    #endif
    #if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
    #endif
}
```

Σχήμα 5.2: Κώδικας της μεθόδου Quit[47]

Για την μέθοδο quit θα ελέγξουμε εάν το παιχνίδι μας τρέχει σε Unity Editor όπου εάν ισχύει θα σταματάει το παιχνίδι. Ενώ εάν το παιχνίδι τρέχει κανονικά σε build in περιβάλλον πάλι θα σταματάει και θα κλείνει. Ο λόγος που κάνουμε τον έλεγχο αυτό είναι επειδή μόνο με το Application.Quit το παιχνίδι μέσα στον Editor του Unity δεν θα σταματούσε να παίζει και για αυτό τον λόγο ελέγχουμε εάν βρισκόμαστε στον Editor ή όχι για να κλείνει η εφαρμογή σε οποιαδήποτε περίπτωση και εάν βρισκόμαστε. Ουσιαστικά το δεύτερο κομμάτι του κώδικα χρησιμεύει μόνο στον προγραμματιστή για να του δώσει μία πιο ολοκληρωμένη εμπειρία κατά την διάρκεια της δημιουργίας του παιχνιδιού του. Τέλος για να συνδέσουμε τα δύο κουμπιά αυτά με τις δύο μεθόδους που δημιουργήσαμε, θα χρειαστεί να πάμε στον component button και να προσθέσουμε στην λίστα On Click() ένα καινούργιο αντικείμενο. Στην συνέχεια θα πρέπει να αντιστοιχίσουμε το GameObject που αναθέσαμε το Script όπου υπάρχουν μέσα οι μέθοδοι. Τέλος διαλέγουμε μέσα από το Script την αντίστοιχη μέθοδο για το αντίστοιχο κουμπί. Πλέον τα κουμπιά Start και Quit είναι έτοιμα.

5.4 Pause Menu

Ένα δεύτερο σημαντικό κομμάτι του menu - interface μας είναι το Pause Menu. Δηλαδή ένα menu όπου θα σταματάει τον χρόνο μέσα στο παιχνίδι και θα διακόπτονται όλες οι αλληλεπιδράσεις με αυτό. Επίσης μέσα από το μενού αυτό θα μπορεί ο χρήστης να πλοηγηθεί στο επόμενο Settings Menu που θα δούμε στο 5.5. Αρχικά θα χρειαστούμε να δημιουργήσουμε ένα GameObject που θα το ονομάσουμε Pause Menu και θα περιέχει όλα τα κουμπιά, τα texts και οτιδήποτε άλλο θα χρειαστούμε για το μενού αυτό. Κάτω από το PauseMenu, θα δημιουργήσουμε το PauseMenuPanel και όπου είναι ένα απλό Panel. Στο Panel αυτό θα προσθέσουμε την εικόνα background όπου θα δώσει μορφή και σχήμα στο Panel αυτό. Επίσης θα αλλάξουμε το χρώμα του σε μαύρο και θα επιλέξουμε το Transparency του στο 40% περίπου. Με αυτό τον τρόπο κάθε φορα που θα εμφανίζεται το Panel αυτό μαζί με το μενού, θα αλλάζει το χρώμα του παιχνιδιού σε λίγο πιο σκούρο και θα δίνει την αίσθηση ότι κάτι έχει αλλάξει στο παιχνίδι, δηλαδή ότι βρίσκεται σε Pause Mode. Πέρα από το PauseMenuPanel μέσα στο Pause Menu θα βρίσκονται και άλλα τρία κουμπιά. Τα κουμπιά αυτά είναι το ResumeButton όπου όπως λέει και το όνομα του θα επαναφέρει το παιχνίδι σε τρέχουσα μορφή και θα κλείνει το Pause Menu. Το δεύτερο κουμπί θα είναι το SettingsButton όπου θα μεταφέρει τον παίκτη στο Settings Menu. Τέλος θα υπάρχει και το RestartButton όπου θα επαναφέρει το παιχνίδι στην αρχική του κατάσταση για να μπορέσει ο παίκτης να ξεκινήσει ένα νέο παιχνίδι από την αρχή.

Ο τρόπος με τον οποίο το μενού αυτό θα καλείται, θα γίνεται με δύο τρόπους. Ο πρώτος είναι ότι θα υπάρχει ένα κουμπί με το εικονίδιο του Pause όπου αν πατηθεί θα εμφανίζεται το μενού αυτό, ενώ ο δεύτερος είναι ο κλασικός τρόπος αν πατηθεί το κουμπί του πληκτρολογίου Escape button. Αρχικά θα χρειαστούμε μία μεταβλητή GameObject για το Pause Menu για να μπορέσουμε να το εμφανίζουμε και να το εξαφανίζουμε. Στην συνέχεια θα χρειαστούμε μία boolean μεταβλητή την isPaused για να ξέρουμε εάν το παιχνίδι είναι σταματημένο ή όχι. Τέλος θα χρειαστούμε ένα ακόμα GameObject για την τιμή του pauseMenuPanel. Μέσα στην μέθοδο Start όπου καλείται με το που ξεκινάει να τρέχει το παιχνίδι, θα δίνουμε τα δύο GameObject menuPause και pausePanel την τιμή SetActive(false). Στην συνέχεια μέσα από την μέθοδο Update που καλείται για κάθε ένα frame που περνάει, θα ελέγχουμε εάν έχει πατηθεί το κουμπί Escape button από το πληκτρολόγιο. Όπου και εάν έχει πατηθεί τότε θα καλείται η μέθοδος mPause.

```
void Update()
{
      if(Input.GetKeyDown(KeyCode.Escape) && !isSettings)
       {
            mPause();
       }
}

public void mPause()
{
    if(isPaused)
    {
        Resume();
    }
    else
    {
        Pause();
    }
}
```

Σχήμα 5.3: Κώδικας της μεθόδου Update και mPause

Μέσα στην μέθοδο mPause θα ελέγχουμε εάν η μεταβλητή isPaused έχει την τιμή True ή False, δηλαδή εάν το παιχνίδι βρίσκεται ήδη σε κατάσταση παύσης ή όχι. Εάν έχει την τιμή False τότε θα καλείται η μέθοδος Pause όπου και θα εμφανίζει το menuPause. Στην συνέχεια θα αλλάζει την τιμή της μεταβλητής boolean σε true και θα αλλάζει και την εικόνα από το κουμπί του Pause σε μία άλλη εικόνα με το σχήμα του κουμπιού Play. τέλος θα εμφανίζει και το pausePanel για να γίνεται αισθητή η διαφορά στην παύση του παιχνιδιού.

```
public void Pause()
{
      menuPause.SetActive(true);
       isPaused = true;
       button.image.sprite = buttonImage;
       pausePanel.SetActive(true);
}
```

```
public void Resume()
{
      menuPause.SetActive(false);
      //Time.timeScale = 1f;
      isPaused = false;
      button.image.sprite = buttonImage2;
      pausePanel.SetActive(false);
}
```

Σχήμα 5.4: Κώδικας της μεθόδου Pause και Resume

Στην περίπτωση που η μεταβλητή isPaused έχει την τιμή true θα καλείται η μέθοδος Resume όπου θα κάνει ακριβώς το αντίστροφο από την μέθοδο Pause. Δηλαδή θα εξαφανίζει το menuPause και το pausePanel, μετά θα δίνει την τιμή false στην boolean μεταβλητή isPaused και τέλος θα αλλάζει πάλι την εικόνα του κουμπιού του pause menu σε μία εικόνα παύσης. Τέλος για το RestartButton θα χρειαστεί να δημιουργήσουμε μία int μεταβλητή για να πάρουμε την τιμή της τρέχουσας σκηνής. Στην συνέχεια θα ξανακάνουμε Load την σκηνή αυτή από την αρχή και θα ορίσουμε τον χρόνο σε ένα. Επίσης θα αλλάξουμε τις boolean μεταβλητές preGame και isPaused σε false(δηλαδή τις αρχικές τους τιμές). Τέλος θα ορίσουμε και τα γραφικά του παιχνιδιού στο default τους(θα δούμε για τα γραφικά του παιχνιδιού στο κεφάλαιο 5.5) και θα αλλάξουμε και την μεταβλητή Board.turn στην αρχική της τιμή, για να έχει την πρώτη κίνηση πάντα ο κόκκινος παίκτης.

```
public void Restart()
{
    int scene = SceneManager.GetActiveScene().buildIndex;
    SceneManager.LoadScene(scene, LoadSceneMode.Single);
    Time.timeScale = 1;
    Board.preGame = false;
    isPaused = false;
    QualitySettings.SetQualityLevel(0);
    Board.turn = 0;
}
```

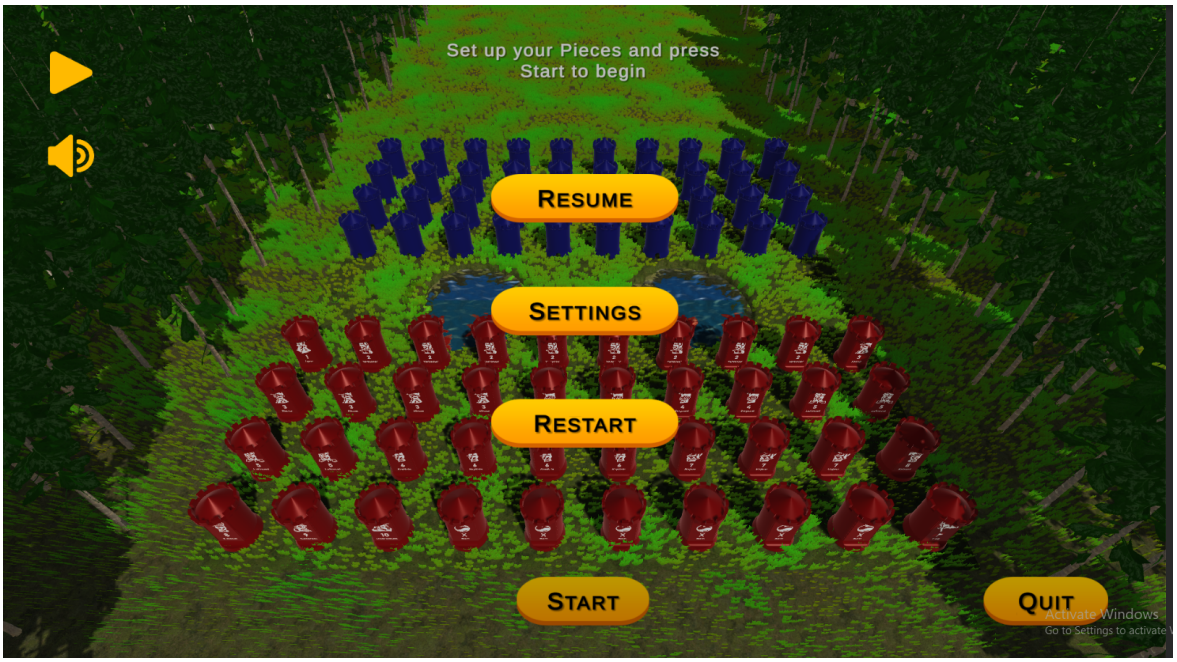
Σχήμα 5.5: Κώδικας της μεθόδου Restart

Όταν θα πατιέται το κουμπί Settings θα εμφανίζεται το Menu Settings και το Settings Panel, θα δίνουμε στην boolean μεταβλητή isPaused την τιμή true. Επίσης θα αλλάζουμε την εικόνα του κουμπιού που καλεί το Settings Menu ξανά στην εικόνα Play και θα δίνουμε και την τιμή True στην μεταβλητή isSettings όπου θα δηλώνει ότι το Settings Menu είναι ενεργό.

```
public void Settings()
{
    menuSettings.SetActive(true);
    settingsPanel.SetActive(true);
    isPaused = true;
    button.image.sprite = buttonImage;
    isSettings = true;
}
```

Σχήμα 5.6: Κώδικας της μεθόδου Settings

Πλέον έχουμε ολοκληρώσει το Pause Menu μας. Όπως βλέπουμε και στο σχήμα 5.7 όταν το μενού παύσης είναι ενεργό, τα κουμπιά Start και Quit, αλλά και το παιχνίδι από πίσω βρίσκεται εκτός λειτουργίας. Πάνω αριστερά στην εικόνα βλέπουμε το κουμπί που εμφανίζει το Pause Menu πέρα από το κουμπί πληκτρολογίου Escape button. Κάτω από το κουμπί αυτό βρίσκεται το κουμπί Mute της έντασης του παιχνιδιού, που θα το δούμε και αυτό στο 5.5 κεφάλαιο.



Σχήμα 5.7: Ολοκληρωμένη εικόνα του Pause Menu

5.5 Settings Menu

Για την κατασκευή του Settings Menu θα ακολουθήσουμε μία παρόμοια διαδικασία με αυτή που ακολουθήσαμε και για την κατασκευή του Pause Menu. Πρώτα θα χρειαστούμε ένα νέο GameObject το Settings Menu, που θα λειτουργήσει ως γονέας όλων των στοιχείων που θα υλοποιήσουμε για το μενού μας. Στην συνέχεια θα χρησιμοποιήσουμε ένα Panel το SettingsMenuPanel όπου θα του αναθέτουμε μία εικόνα, για να δώσουμε πάλι μορφή στο χρώμα του και θα ρυθμίσουμε κατάλληλα το transparency του για να γίνεται αισθητή η παρουσία του. Στην συνέχεια θα δημιουργήσουμε δύο νέα GameObjects όπου θα τα ονομάσουμε Borders. Θα τα αναθέσουμε μέσα από το package 3D Modern Menu UI[48] του Asset Store, δύο εικόνες τις Handle solid και Button Frame Press. Με αυτό τον τρόπο θα δώσουμε οπτική μορφή στο περίγραμμα του μενού μας. Θα ρυθμίσουμε τις διαστάσεις τους ως προς τις δικές μας ανάγκες.
Το υπόλοιπο μενού θα είναι λίγο διαφορετικό από το Pause Menu που ήδη φτιάξαμε. Μέσα στο μενού αυτό θα υπάρχει ένα DropDown - TextMeshPro για την επιλογή των γραφικών του παιχνιδιού, ένα δεύτερο DropDown - TextMeshPro για την επιλογή του Resolution του, ένα Toggle button για την επιλογή να είναι το παιχνίδι Fullscreen ή όχι και τέλος ένας Slider που θα ρυθμίζει την ένταση του παιχνιδιού. Επίσης θα υπάρχει και ένα απλό κουμπί που θα δίνει την δυνατότητα στον παίκτη να γυρίσει πίσω στο Pause Menu και να κλείσει το Settings Menu. Τέλος για κάθε ένα κουμπί - DropDown ή Slider θα τοποθετήσουμε και ένα Text που θα λειτουργεί σαν τίτλο του κάθε κουμπιού.

5.5.1 Ρύθμιση των γραφικών

Αρχικά θα δούμε τον τρόπο με τον οποίο θα φτιάξουμε τις ρυθμίσεις των γραφικών μας. Όπως όλα τα σωστά φτιαγμένα παιχνίδια, έτσι και το δικό μας, θα δίνει την δυνατότητα στον χρήστη να ρυθμίσει τα γραφικά του παιχνιδιού, ανάλογα με τις δυνατότητες του υπολογιστή του, για να μπορέσει να ανταπεξέλθει στις απαιτήσεις του καλύτερα. Για την δημιουργία των γραφικών θα χρησιμοποιήσουμε ένα DropBox, όπου μέσα σε αυτό θα αναθέτουμε τα διαφορετικά γραφικά του παιχνιδιού. Αφού δημιουργήσουμε ένα DropDown - TextMeshPro κάτω από τον γονέα Settings Menu, θα δώσουμε στο μενού αυτό μία παρόμοια εικόνα για να ταιριάζει με το θέμα μας, και θα δώσουμε στο DropBox τις τιμές , Ultra, Very High, High, Medium, Low και Very Low όπου θα είναι οι πέντε διαφορετικές τιμές των γραφικών μας. Στην συνέχεια δίπλα από το DropBox μας θα δημιουργήσουμε και ένα Text, όπου θα το ονομάσουμε Graphics και θα λειτουργεί σαν οδηγός του DropDown προς τον παίκτη. Στην συνέχεια θα πάμε στα Project Settings του project μας και θα διαλέξουμε την κατηγορία Quality. Εκεί πέρα θα δημιουργήσουμε ακριβώς τις ίδιες πέντε διαφορετικές επιλογές για τα γραφικά μας. Έχοντας πλέον τις πέντε αυτές επιλογές, θα επιλέξουμε την κάθε μία ξεχωριστά και μας δίνεται η δυνατότητα να διαλέξουμε τις τιμές των γραφικών σε διάφορα πεδία όπως τα: LOD Bias, Maximum LOD level, Particle Raycast Budget, Texture Quality, Anisotropic Textures, Realtime Reflection Probes, Billboards Face Camera Position κ.α. Για κάθε μία επιλογή θα ρυθμίσουμε τις τιμές ανάλογα με την αντίστοιχη επιλογή των γραφικών (από Ultra έως Very Low σε φθίνουσα σειρά). Το μόνο που μένει τώρα είναι να δώσουμε στον παίκτη την δυνατότητα να αλλάζει μεταξύ αυτών των πέντε επιλογών γραφικών μέσα από το παιχνίδι.

```
public void SetQuality(int qualityIndex)
{
      QualitySettings.SetQualityLevel(qualityIndex);
}
```

Σχήμα 5.8: Κώδικας της μεθόδου SetQuality

Ο κώδικας της μεθόδου SetQuality που θα ορίζει την επιλογή των γραφικών του παιχνιδιού είναι αρκετά απλός. Έχοντας πλέον τις ίδιες επιλογές και στο DropDown - TextMeshPro αλλά και στο Project Settings του παιχνιδιού μας, το μόνο που μένει είναι να αντιστοιχίσουμε την int μεταβλητή του DropDown με αυτή από το Project Settings[49]. Για αυτό τον λόγο είναι σημαντικό οι επιλογές αυτές να είναι ολόιδιες και στις δύο μεριές. Έτσι απλά δίνουμε την δυνατότητα στον χρήστη να ρυθμίσει τα γραφικά του παιχνιδιού ως προς τα δικά του θέλω.

5.5.2 Fullscreen Mode

Το δεύτερο κομμάτι που θα εξετάσουμε για την κατασκευή του Settings Menu, είναι η επιλογή FullScreen του παιχνιδιού. Όταν τρέχουμε το παιχνίδι η Default τιμή για το παιχνίδι μας είναι να βρίσκεται σε FullScreen. Ο παίκτης όμως πολλές φορές μπορεί να χρειάζεται την αντίθετη επιλογή, δηλαδή το παιχνίδι να βρίσκεται σε Windowed mode. Η υλοποίηση του είναι και αυτή πάρα πολύ απλή, αρχικά στο Toggle button που χρησιμοποιήσαμε θα ορίσουμε πάλι τις κατάλληλες τιμές αλλά και την εικόνα για να ταιριάζει και αυτή με το θέμα μας. Στην συνέχεια θα φτιάξουμε την μέθοδο SetFullscreen όπου θα πάρει παραμετρικά μία μεταβλητή Boolean και θα ανάλογα με το αν το κουμπί έχει πατηθεί η όχι θα ορίζει το παιχνίδι σε FullScreen ή windowed mode[50].

```
public void SetFullScreen(bool fullScreen)
{
      Screen.fullScreen = fullScreen;
}
```

Σχήμα 5.9: Κώδικας της μεθόδου SetFullScreen

5.5.3 Ρύθμιση της ανάλυσης του παιχνιδιού

Η υλοποίηση της ανάλυσης του παιχνιδιού είναι λίγο πιο περίπλοκη από τις άλλες επιλογές που είδαμε για το Settings Menu. Ο κύριος λόγος που είναι πιο περίπλοκη η υλοποίηση της ανάλυσης, είναι επειδή κάθε συσκευή που θα παίζει το παιχνίδι, θα έχει και διαφορετικές επιλογές ανάλογα την συσκευή και τις δυνατότητές της. Για το καταφέρουμε αυτό, θα χρειαστούμε να δημιουργήσουμε την λίστα resolutions όπου θα περιέχει όλα τα resolutions που μπορεί να υποστηρίξει η συσκευή. Στην συνέχεια για κάθε ένα από τα στοιχεία της λίστας resolution, θα δημιουργούμε ένα String Option και θα του δίνουμε τις τιμές width και height του αντίστοιχου resolution και θα το προσθέτουμε στο resolution DropDown - TextMeshPro. Στην συνέχεια θα ελέγχουμε εάν το width και το height του αντίστοιχου resolution έχει τις ίδιες τιμές με το τρέχων resolution και αν ισχύει θα αποθηκεύει τις τιμές αυτές για να μπορέσει να κάνει refresh την τιμή του DropDown ως προς το τρέχων resolution. Όλα τα παραπάνω θα γίνονται μέσα στην μέθοδο Start όπου καλείται με το που ξεκινάει το παιχνίδι. Δηλαδή θα δίνουμε όλα τα resolution μέσα στο DropDown resolution μας και θα του αναθέτουμε και το τρέχων resolution για να διορθώσουμε την τιμή του. Το μόνο που μένει τώρα, είναι να δημιουργήσουμε και την μέθοδο SetResolution όπου θα συνδέσουμε την μέθοδο αυτή με το DropDown - TexhMeshPro και θα πάρει παραμετρικά την τιμή int όταν θα αλλάζουμε την επιλογή του DropDown. Έχοντας την τιμή της επιλογής θα αλλάζουμε το width και το height τους ανάλογα με την επιλογή που κάναμε, μέσα από την λίστα resolutions[51].

```
List<string> options = new List<string>();

int currentResolution = 0;
for (int i = 0; i < resolutions.Length; i++)
{
      string option = resolutions[i].width + " x " +
resolutions[i].height;
      options.Add(option);

      if(resolutions[i].width == Screen.currentResolution.width &&
            resolutions[i].height == Screen.currentResolution.height)
      {
            currentResolution = i;
      }
}

resolutionDropdown.AddOptions(options);
resolutionDropdown.value = currentResolution;
```

```
resolutionDropdown.RefreshShownValue();
```

Σχήμα 5.10: Μέρος κώδικα της μεθόδου Start

```
public void SetResolution(int resolutionIndex)
{
      Resolution resolution = resolutions[resolutionIndex];
      Screen.SetResolution(resolution.width, resolution.height,
Screen.fullScreen);
}
```

Σχήμα 5.11:Κώδικας της μεθόδου SetResolution

5.5.4 Ρύθμιση έντασης

Για την ρύθμιση της έντασης του παιχνιδιού μας, αρχικά θα χρειαστεί να δημιουργήσουμε ένα καινούργιο Audio Mixer, που θα το ονομάσουμε Main Mixer. Θα συνδέσουμε το Audio Mixer που δημιουργήσαμε με το Script μας για να μπορέσουμε να επεξεργαστούμε την τιμή του. Στον Slider που έχουμε θα ορίσουμε πάλι την κατάλληλη εικόνα αλλά και χρώματα και ιδιότητες για να ταιριάζει με το υπόλοιπο μενού μας. Στην συνέχεια, επειδή ο Audio Mixer μπορεί να πάρει τις τιμές από 0 έως -80, για αυτό τον λόγο θα πάμε στον Volume Slider που δημιουργήσαμε και θα ορίσουμε τις τιμές Max Value ίσο με 0 και Min Value ίσον με -80. Τώρα πλέον όταν θα αλλάζουμε τον Slider θα αλλάζει και η τιμή του Audio Mixer μας. Το μόνο που μένει είναι να γράψουμε στο script τον τρόπο με τον οποίο θα αλλάζει η τιμή του. Θα δημιουργήσουμε την μέθοδο SetVolume, που θα περνάει μία float μεταβλητή(την τιμή του slider) και θα την ορίζει μέσα στο Audio Mixer. Έτσι απλά δημιουργήσαμε και το Slider μας[52].

```
public void SetVolume(float volume)
{
      audioMixer.SetFloat("volume", volume);
}
```

Σχήμα 5.12:Κώδικας της μεθόδου SetVolume

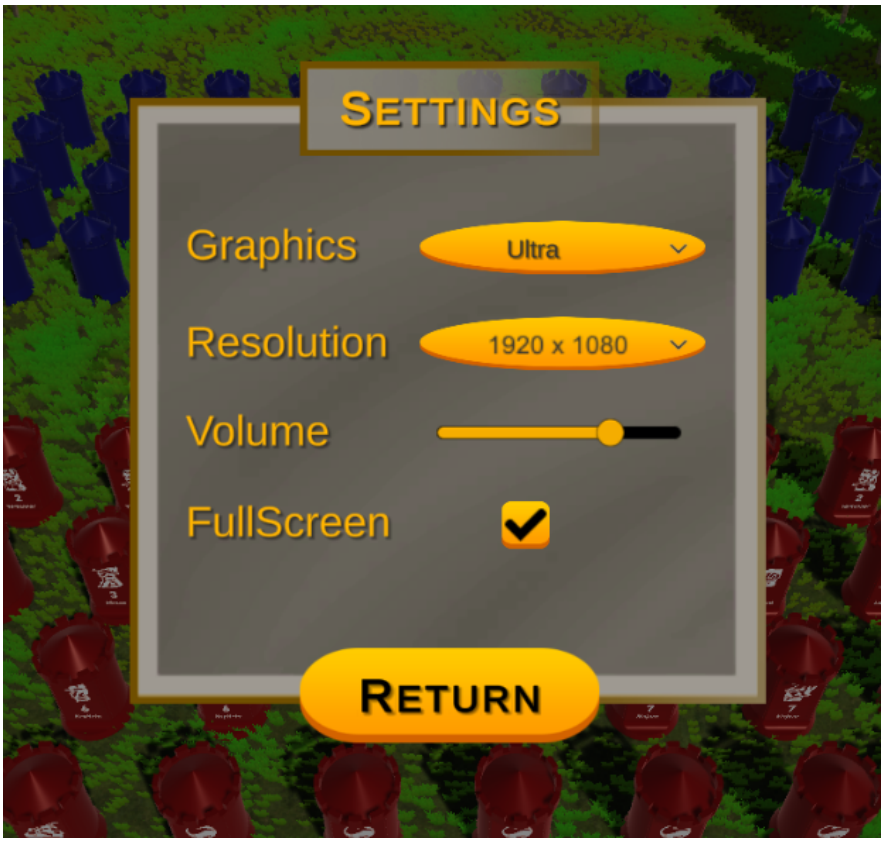
Για την ρύθμιση της έντασης, έχουμε να υλοποιήσουμε και το κουμπί που θα βρίσκεται στην επιφάνεια του παιχνιδιού. Δηλαδή θα μπορεί ο χρήστης να χαμηλώσει τελείως το παιχνίδι ή να δώσει την default τιμή στην ένταση, χωρίς να χρειαστεί να πλοηγηθεί στο Settings Menu, απλά με το πάτημα του κουμπιού. Θα δημιουργήσουμε αρχικά το κουμπί Music Button όπου θα είναι ξεχωριστά από το Settings Menu καθώς θέλουμε να λειτουργεί και όταν το Settings Menu είναι ανενεργό. Στο κουμπί αυτό θα αναθέσουμε την εικόνα speaker, όπου είναι μία εικόνα που δείχνει ένα ηχείο με την ένταση ανοιχτή. Στην συνέχεια θα δώσουμε τα ίδια χρώματα και ιδιότητες με τα υπόλοιπα κουμπιά που έχουμε ήδη δημιουργήσει. Όταν θα πατιέται το κουμπί αυτό, θα κλείνει τελείως την ένταση του παιχνιδιού. Επίσης θα αλλάζει την εικόνα του ηχείου, με ένα άλλο ηχείο που θα δείχνει ότι δεν υπάρχει ένταση. Για να αλλάζουμε την εικόνα και να αυξάνουμε ή να χαμηλώνουμε την ένταση, θα χρησιμοποιήσουμε την boolean μεταβλητή isVolume. Όταν η μεταβλητή αυτή έχει την τιμή true, θα αναθέτει στην ένταση του Audio Mixer την τιμή -20(λίγο πιο κάτω από το μέγιστο), θα αλλάζει την εικόνα του ηχείου σε αυτή με την ένταση και θα δίνει την τιμή false στην μεταβλητή isVolume. Αντιθέτως θα αναθέτει στην ένταση του Audio Mixer την τιμή -80(δηλαδή θα είναι τέρμα χαμηλωμένη η ένταση), θα αλλάζει την εικόνα του ηχείου σε αυτή χωρίς ένταση και θα δίνει την τιμή true στην μεταβλητή isVolume.

```
public void SetVolume2()
{
      if(isVolume)
      {
            audioMixer.SetFloat("volume", -20);
            buttonVolume.image.sprite = buttonVolumeImage2;
            isVolume = false;
      }
      else
      {
            audioMixer.SetFloat("volume", -80);
            buttonVolume.image.sprite = buttonVolumeImage;
            isVolume = true;
      }
}
```

Σχήμα 5.13:Κώδικας της μεθόδου SetVolume2

Τέλος θα υπάρχει και ένα κουμπί στο κάτω μέρος του μενού που θα επιτρέπει στον χρήστη να γυρίσει πίσω στο Pause Menu και να κλείσει το Settings Menu. Θα δημιουργήσουμε ένα ακόμη

κουμπί που θα έχει την ονομασία Return και θα το συνδέσουμε με την μέθοδος Return. Η μέθοδος αυτή όταν θα καλείται θα εξαφανίζει το Settings Menu μαζί με το settings panel και θα αλλάζει την τιμή της boolean μεταβλητής isPaused σε true. Αυτόματα με την αλλαγή της τιμής της μεταβλητής, το Pause Menu θα γίνεται πάλι ενεργό(ενώ πριν βρισκόταν πίσω στο παρασκήνιο) και ο παίκτης θα έχει την δυνατότητα είτε να συνεχίσει το παιχνίδι του, είτε να ξεκινήσει ένα καινούργιο από την αρχή. Τέλος θα προσθέσουμε ένα τραγούδι στο παρασκήνιο του παιχνιδιού, που ο παίκτης θα μπορεί με τον Volume Slider να το δυναμώσει ή χαμηλώσει. Αρχικά θα πάρουμε το τραγούδι "Warrior's Tribe"[53] θα το κατεβάσουμε και θα το προσθέσουμε μέσα στα Assets του project μας. Στην συνέχεια θα δημιουργήσουμε ένα νέο Script που θα το ονομάσουμε AudioManager και θα συνδέσουμε το τραγούδι αυτό με το Script στην μεταβλητή backgroundMusic. Στην μέθοδο Start του Script θα βάλουμε το τραγούδι αυτό να ξεκινήσει να παίζει και θα ρυθμίσουμε από τις ρυθμίσεις του Audio Source να είναι On Loop, δηλαδή εφόσον φτάσει στο τέλος του να ξαναπαίζει από την αρχή. Πλέον έχουμε τελειώσει και με το Settings Menu μας.



Σχήμα 5.14:Ολοκληρωμένη εικόνα του Settings Menu

5.6 Death Menu

Το τελευταίο στοιχείο για το παιχνίδι μας είναι το να φτιάξουμε ένα μικρό μενού για όταν το παιχνίδι φτάσει στο τέλος του. Το μενού αυτό θα είναι πολύ πιο απλό από τα δύο προηγούμενα που

φτιάξαμε. Το μόνο που θα κάνει είναι να εμφανίζει ένα μήνυμα στον παίκτη, που θα λέει εάν κέρδισε ή έχασε το παιχνίδι. Κάτω από το μήνυμα αυτό θα ρωτάει εάν θέλει να ξαναπαίξει ένα νέο παιχνίδι από την αρχή ή εάν επιθυμεί να κλείσει το παιχνίδι. Αρχικά θα δημιουργήσουμε πάλι ένα GameObject που θα το ονομάσουμε Death Menu, και θα λειτουργεί ως γονέας του μενού. Στην συνέχεια θα δημιουργήσουμε ένα Panel, το Death Menu Panel όπου θα χρησιμοποιήσουμε το μαύρο χρώμα και θα θέσουμε το Transparency στο 75% για να φαίνεται πιο σκοτεινή η εικόνα. Θα χρειαστούμε δύο TextMeshPro - text, ένα όπου θα εμφανίζει το μήνυμα You Won!!! ή You Lost!!! ανάλογα εάν έχασε ή κέρδισε το παιχνίδι. Το δεύτερο text θα γράφει Do You Wanna Play Again?, όπου θα ρωτάει τον χρήστη εάν επιθυμεί να παίξει ξανά. Τέλος θα δημιουργήσουμε δύο κουμπιά το RestartButton και το QuitButton, όπου το ένα θα ξαναξεκινάει το παιχνίδι μας από την αρχή ενώ το αλλό θα κλείνει το παιχνίδι. Στην μέθοδο Start από το Script Board αρχικά θα απενεργοποιούμε το

Death Panel και το Death Menu. Όταν ο χρήστης κερδίσει ή χάσει το παιχνίδι, μέσα από το Script Board θα καλεί την μέθοδο DeathPanel. Η μέθοδος DeathPanel θα ενεργοποιεί ξανά το Death Menu και το Death Panel, θα αλλάζει την τιμή της boolean μεταβλητής isPaused σε true και θα αλλάζει και την σειρά των παικτών σε περίπτωση που ο χρήστης ξαναζητήσει να παίξει. Για το κουμπί QuitButton έχουμε ήδη υλοποίηση την μέθοδο Quit οπότε το μόνο που μένει είναι να συνδέσουμε την μέθοδο με το κουμπί αυτό. Για το Restart Button όμως θα δημιουργήσουμε την μέθοδος Restart στο Script Interface. Αυτό που θα κάνει η μέθοδος αυτή, είναι να αποθηκεύει το νούμερο της τρέχουσας σκηνή σε μία μεταβλητή int και με την βοήθεια του SceneManager να ξαναφορτώνει την σκηνή αυτή. Στην συνέχεια ξανά ρυθμίζουμε την τιμή Time.timeScale σε 1 και αλλάζουμε την τιμή της boolean μεταβλητής preGame σε false(καθώς ξεκινάμε ένα νέο παιχνίδι). Επίσης θα αλλάξουμε και την τιμή της μεταβλητής isPaused σε false και θα αλλάζει και την σειρά των παικτών. Κάπως έτσι έχουμε τελειώσει με την υλοποίηση του Interface - Menu του παιχνιδιού μας.

```
public void Restart()
{
      int scene = SceneManager.GetActiveScene().buildIndex;
      SceneManager.LoadScene(scene, LoadSceneMode.Single);
      Time.timeScale = 1;
      Board.preGame = false;
      isPaused = false;
      QualitySettings.SetQualityLevel(0);
      Board.turn = 0;
}
```

Σχήμα 5.15:Κώδικας της μεθόδου Restart

```
public void DeathPanel()
{
	Interface.isPaused = true;
	menuDeath.SetActive(true);
	deathPanel.SetActive(true);
	turn = 1;
}
```

Σχήμα 5.16:Κώδικας της μεθόδου DeathPanel

Σχήμα 5.17:Ολοκληρωμένη εικόνα του Death Menu

Κεφάλαιο 6ο: Συμπεράσματα

Στην παρούσα πτυχιακή εργασία είδαμε τον τρόπο ανάπτυξης του επιτραπέζιου παιχνιδιού Stratego σε περιβάλλον 3D με το πρόγραμμα Unity. Κατά την διάρκεια της εργασίας αντιμετωπίσαμε

πολλές προκλήσεις, διερευνήσαμε διάφορες πτυχές του σχεδιασμού και της υλοποίησης της τρισδιάστατης εφαρμογής και μάθαμε πως να ανταπεξέλθουμε στα διάφορα προβλήματα που προέκυψαν στην πορεία υλοποίησης της εργασίας. Ο αρχικός στόχος αυτού του έργου ήταν να φέρει μία νέα διάσταση στο παραδοσιακό επιτραπέζιο παιχνίδι 2D, προσφέροντας στους παίκτες μία συναρπαστική και καθηλωτική εμπειρία, διατηρώντας το στρατηγικό του στοιχείο. Επίσης μέσα από την εργασία αυτή μαθαίνουμε για το πόσο σημαντικές είναι οι δυνατότητες των εφαρμογών ανάπτυξης παιχνιδιών και πως μπορούν να δώσουν μία νέα πνοή στα κλασικά παιχνίδια της παιδικής μας ηλικίας, καθιστώντας τα πιο προσιτά και ελκυστικά στο σύγχρονο κοινό.

Η εργασία μας ξεκίνησε με μία ολοκληρωμένη εισαγωγή στα βιντεοπαιχνίδια και στην

επιρροή τους στην καθημερινότητα των ανθρώπων στην κοινωνία μας. Στην συνέχεια μάθαμε για την μηχανή ανάπτυξης Unity 3D όπου και χρησιμοποιήσαμε για την ανάπτυξη του δικού μας παιχνιδιού. Εξετάσαμε αναλυτικά τα κυριότερα μέρη της διεπαφής της εφαρμογής, τις ιδιότητές τους και την χρήση τους. Τέλος αναλύσαμε τις σημαντικότερες έννοιες της, όπου και τις χρησιμοποιούμε για την υλοποίηση της εργασίας. Στο αμέσως επόμενο κεφάλαιο κατανοήσαμε τους κανόνες του παιχνιδιού Stratego καθώς και όλες τις ιδιότητες των πιονιών αλλά και τις ιδιαιτερότητες τους. Έχοντας πλέον μάθει όλα τα σημαντικά στοιχεία του παιχνιδιού,
εξετάσαμε βήμα-βήμα τον τρόπο με τον οποίο αναπτύξαμε το παιχνίδι. Αρχικά είδαμε πως υλοποιήσαμε το Board και την αλληλεπίδραση του με τον παίκτη. Αμέσως μετά μάθαμε πως να δημιουργούμε και να ομορφαίνουμε το δικό μας περιβάλλον μέσα στον 3D κόσμο του παιχνιδιού. Τέλος αναλύσαμε την διαδικασία υλοποίησης των πιονιών και των κανόνων που μάθαμε στο κεφάλαιο 3. Έχοντας ολοκληρώση με την ανάπτυξη του παιχνιδιού, μάθαμε πως να δημιουργούμε το δικό μας μενού και να ρυθμίζουμε τις λειτουργίες του και πως να το συνδέουμε με το παιχνίδι αλλά και με τον παίκτη. Είδαμε δηλαδή μία ολοκληρωμένη διαδικασία ανάπτυξης παιχνιδιού, από την κατανόηση των εργαλείων που χρησιμοποιήσαμε καθώς και την κατανόηση του ίδιου του παιχνιδιού μέχρι τον τρόπο υλοποίησης του.

Καθώς ολοκληρώνουμε την πτυχιακή εργασία, είναι σημαντικό να αναγνωρίσουμε ότι η ανάπτυξη των παιχνιδιών είναι μία συνεχής διαδικασία. Υπάρχουν πάντα τρόποι για περαιτέρω βελτίωση, επέκταση και καινοτομία του παιχνιδιού. Το μόνο που χρειάζεται κάποιος για να τα πετύχει όλα αυτά είναι αφοσίωση, δημιουργικότητα και πάθος για τον σχεδιασμό και την ανάπτυξη παιχνιδιών.

Κλείνοντας, αυτή η εργασία ήτανε μία ανταποδοτική εμπειρία στην ανάπτυξη παιχνιδιών, όπου όχι μόνο δημιουργήσαμε το δικό μας παιχνίδι Stratego 3D αλλά επεκτείναμε και τις γνώσεις μας και τις δεξιότητες μας πάνω στον τομέα αυτόν. Ελπίζω η εργασία αυτή να εμπνεύσει τους μελλοντικούς προγραμματιστές παιχνιδιών και να τους δώσει μία ώθηση για να συνεχίσουν να πιέζουν τα όρια τους για να υλοποιήσουν το δικό τους παιχνίδι.

Βιβλιογραφία

[1]Wikipedia, “Video Games” , 30 ιουλίου 2023 from https://en.wikipedia.org/wiki/Video_game
[2]Computer and Video Games, Widipedia from https://www.cs.mcgill.ca/~rwest/wikispeedia/wpcd/wp/c/Computer_and_video_games.htm
[3]Alan Blank-Rochester, “Video games speed up reaction time”, FUTURITY, 13 Σεπτεμβρίου 2010 from https://www.futurity.org/video-games-speed-up-reaction-time
[4]Record Head, “10 Reasons Why Playing Video Games is Good for Your Brain”, from https://recordhead.biz/10-reasons-video-games-is-good-for-you/
[5]Gavin Wright, “Gaming”, TechTarget, Νοέμβριο 2022 from https://www.techtarget.com/whatis/definition/gaming
[6]John Stafford, “London hospital to help teens kick that World Of Warcraft habit”, Scope - Stanford Medicine, 19 Μαρτίου 2010 from https://scopeblog.stanford.edu/2010/03/19/london_hospital/
[7]Market Business News, “What is gaming” , 2023 from https://marketbusinessnews.com/financial-glossary/gaming-definition/
[8]Cam Adair, “The negative effects of Video Games - 12 Symptoms”, Game Quitters, from https://gamequitters.com/negative-effects-of-video-games/
[9]Adolescent Gaming, “Physical and Mental Effects”, from https://adolescentgaming.wordpress.com/physical-and-mental-health/
[10]Adam Sinicki, “What is Unity? Everything you need to know”, Android Authority, 20 Μαρτίου 2021 from https://www.androidauthority.com/what-is-unity-1131558/
[11]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/Hierarchy.html
[12]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/UsingTheSceneView.html
[13]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/GameView.html
[14]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/AssetStore.html
[15]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/UsingTheInspector.html
[16]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/ProjectView.html
[17]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/Console.html
[18]Unity Documentation, 2023 from]https://docs.unity3d.com/ScriptReference/GameObject.html
[19]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/Components.html
[20]Unity Documentation, 2023 from https://docs.unity3d.com/ScriptReference/Transform.html

[21]Unity Documentation, 2023 from
https://docs.unity3d.com/ScriptReference/Rigidbody.html
[22]Unity Documentation, 2023 from
https://docs.unity3d.com/ScriptReference/Collider.html
[23]Unity Documentation, 2023 from
https://docs.unity3d.com/Manual/Prefabs.html
[24]Unity Documentation, 2023 from
https://docs.unity3d.com/ScriptReference/ParticleSystem.html
[25]Unity Documentation, 2023 from
https://docs.unity3d.com/ScriptReference/SerializeField.html
[26]Unity Documentation, 2023 from
https://docs.unity3d.com/Manual/TerrainTools.html
[27]Unity Documentation, 2023 from
https://docs.unity3d.com/Manual/script-Terrain.html
[28]Unity Documentation, 2023 from
https://docs.unity3d.com/Manual/class-TerrainLayer.html
[29]Unity Documentation, 2023 from
https://docs.unity3d.com/Manual/class-Tree.html
[30]Unity Documentation, 2023 from
https://docs.unity3d.com/Manual/terrain-Grass.html
[31]Wikipedia, “Stratego”, 2023 from
https://en.wikipedia.org/wiki/Stratego
[32]Stratego, “Ranks”, from
https://speluitslagen.nl/spelregels/7?lang=en
[33]Fortra Digital Defense, “Learning Cyber Defense Strategies from Stratego Strategy”, Fortra, from
https://www.digitaldefense.com/blog/cyber-defense-stratego/
[34]Mercenary Camp (30 Μαρτίου 2021). Tutorial για 3D σκάκι. [Video]. Youtube.
https://www.youtube.com/watch?v=FtGy7J8XD90&ab_channel=MercenaryCamp
[35]Unity Documentation, 2023 from
https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@16.0/manual/
[36]Unity Asset Store , 2023 from
https://assetstore.unity.com/packages/3d/environments/landscapes/terrain-sample-asset-pack-145808
[37]Unity Asset Store, 2023 from
https://assetstore.unity.com/packages/2d/textures-materials/nature/grass-and-flowers-pack-1-17100
[38] 帕羅 Paro (2 Ιανουαρίου 2023). Tutorial για water shader [Video]. Youtube.
https://www.youtube.com/watch?v=91XXV6WtEbQ&ab_channel=%E5%B8%95%E7%BE%85Paro
[39] Levend Stratego Cards(Stratego Alive, dual color),
https://www.printables.com/en/model/377438-levend-stratego-cards-stratego-alive-dual-color
[40]Stratego Spare Part(Board-Game),
https://cults3d.com/en/3d-model/game/stratego-spare-part-board-game

[41]Mercenary Camp (1 Απριλίου 2021). Tutorial για 3D σκάκι. [Video]. Youtube. https://www.youtube.com/watch?v=INcZNkU-XUw&ab_channel=MercenaryCamp
[42]Mercenary Camp (1 Απριλίου 2021). Tutorial για 3D σκάκι. [Video]. Youtube. https://www.youtube.com/watch?v=3kW54hU98os&ab_channel=MercenaryCamp
[43]Chess.com , The Rules of Chess, 5 Απριλίου 2022 from https://support.chess.com/article/1042-i-got-a-draw-by-repetition-how-did-that-happen
[44]Max Pears , Setting the Tone: Main Menus are the game, Game Developer, 21 Απριλίου 2016 from https://www.gamedeveloper.com/design/setting-the-tone-main-menus-are-the-game
[45]Unity Documentation, 2023 from https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html
[46]Unity Asset Store, 2023 from https://assetstore.unity.com/packages/2d/gui/icons/simple-button-set-01-153979
[47]KillerFirefly, How to Close a Game, StackOverFlow, 21 Δεκεμβρίου 2021 from https://stackoverflow.com/questions/70437401/cannot-finish-the-game-in-unity-using-application-quit
[48]Unity Asset Store, 2023 from https://assetstore.unity.com/packages/tools/gui/3d-modern-menu-ui-116144
[49]Unity Documentation, 2023 from https://docs.unity3d.com/Manual/class-QualitySettings.html
[50]Unity Documentation, 2023 from https://docs.unity3d.com/ScriptReference/Screen-fullScreen.html
[51]Unity Documentation, 2023 from https://docs.unity3d.com/ScriptReference/Screen-resolutions.html
[52]Unity Documentation, 2023 from https://docs.unity3d.com/ScriptReference/AudioSource-volume.html
[53]Rotem Moav, “Warrior’s Tribe”, Relevator Ltd, 7 Μαρτίου 2021 from https://www.youtube.com/watch?v=2DrKF5X6v5E&ab_channel=RotemMoav-Topic