ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ FPS ΜΕ ΧΡΗΣΗ ΤΗΣ ΜΗΧΑΝΗΣ Unity3D

Των φοιτητών Ιωαννίδης Ευστάθιος
και Ιορδανίδης Ιορδάνης
Αρ. Μητρώου: 144282 & 144271

Επιβλέπων
Δεληγιάννης Ιγνάτιος
Βαθμίδα Καθηγητής

Ημερομηνία 15-06-2021

Τίτλος Δ.Ε: Ανάπτυξη παιχνιδιού FPS με Unity3D
Κωδικός Δ.Ε: 20205
Ονοματεπώνυμο φοιτητών: Ιωαννίδης Ευστάθιος & Ιορδανίδης Ιορδάνης
Ονοματεπώνυμο εισηγητή: Δεληγιάννης Ιγνάτιος
Ημερομηνία ανάληψης Δ.Ε 03-11-2020
Ημερομηνία περάτωσης Δ.Ε 15-06-2021

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.

Περίληψη

Στην παρούσα πτυχιακή εργασία θα παρουσιαστεί η διαδικασία ανάπτυξης ενός προϊόντος ψυχαγωγικού λογισμικού με χρήση της μηχανής παιχνιδιών Unity3D. Αρχικά, γίνεται μια ιστορική αναδρομή της πορείας εξέλιξης των βιντεοπαιχνιδιών στην πάροδο των χρόνων. Ακόμη, παρουσιάζονται και αναλύονται οι βασικές κατηγορίες βιντεοπαιχνιδιών αλλά και οι ομάδες ατόμων που συμμετέχουν στην ανάπτυξη τους. Επιπρόσθετα θα αναλυθούν τα στάδια ανάπτυξης της δημιουργίας ενός βιντεοπαιχνιδιού και θα αναφερθούν κάποια βασικά χαρακτηριστικά των πιο δημοφιλών μηχανών δημιουργίας. Συγκεκριμένα παραθέτονται οι μηχανές: Game Maker Studio 2, Unreal Engine, Phase, Godot και CryEngine.

Στην πορεία θα αναφερθούμε λεπτομερώς στην δημοφιλέστερη μηχανή δημιουργίας, την προσωπική μας επιλογή για την κατασκευή του βιντεοπαιχνιδιού, την Unity3D. Έτσι, θα απαριθμήσουμε τα μοναδικά χαρακτηριστικά της και θα μνημονεύσουμε την πορεία εξέλιξης της εταιρίας από τα πρωταρχικά της στάδια μέχρι και σήμερα. Επιπλέον, θα εξετάσουμε αναλυτικά όλα τα βασικά παράθυρα αλλά και τις βασικές έννοιες από τις οποίες αποτελείται η Unity3D που θα πρέπει ο αναγνώστης να αντιληφθεί για την ευκολότερη κατανόηση της πρακτικής υλοποίησης του παιχνιδιού.

Εν συνεχεία, θα γίνει βήμα προς βήμα παρουσίαση και επεξήγηση όλων των σταδίων που ακολουθήθηκαν μέχρι την πλήρη ανάπτυξη του παιχνιδιού μας. Τέλος, θα επιδείξουμε μια εναλλακτική μορφή Έγγραφου Σχεδίασης Παιχνιδιών, ειδικά σχεδιασμένη για μικρότερες ομάδες ανάπτυξης, στην οποία βασιστήκαμε και υλοποιήσαμε το συγκεκριμένο βιντεοπαιχνίδι.

Development of FPS game using the Unity3D engine

IOANNIDIS EFSTATHIOS

IORDANIDIS IORDANIS

Abstract

This thesis will present the process of developing an entertainment software product (a FPS game) using the Unity3D game engine. Firstly, a historical overview of the evolution of video games will be introduced. Then, the main categories of video games and the people involved in their development will be presented. In addition, the development stages of creating a video game and the key features of the most popular game engines will be discussed. Specifically, this study will reference game engines such as the Game Maker Studio 2, Unreal Engine, Phase, Godot, CryEngine and then will thoroughly discuss the choice of utilising the Unity3D game engine to develop our FPS game. Moreover, all the basic windows and concepts that Unity3D consists of will be examined and explained, which will allow the reader to acquire a basic understanding of the practical implementation of the game. Then, there will be a step by step presentation and explanation of all the steps that were followed detailing the full development of our game. Finally, we will demonstrate an alternative form of the Game Design Document, specifically designed for smaller development teams, based on which the current video game was implemented.

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε θερμά τον επιβλέπων καθηγητή μας που μας έδωσε την ευκαιρία να ασχοληθούμε με αυτό το θέμα και τις οικογένειες μας που στάθηκαν δίπλα μας στον αγώνα αυτό.

Τέλος ευχαριστούμε την σχολή και όλο το προσωπικό της, για την βοήθεια και τις γνώσεις που μας έδωσαν και αυτά που μας δίδαξαν.

Περιεχόμενα

Κατάλογος Σχημάτων και Εικόνων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
FPS	First Person Shooter
TPS	Third Person Shooter
MOBA	Multiplayer Online Battle Area
MMORGPs	Massively Multiplayer Online Role-Playing Games
VR	Virtual Reality
AR	Augmented Reality
GDD	Game Design Document
SGDD	Short Game Design Document
SDK	Software Development Kit

Κεφάλαιο 1ο: Βιντεοπαιχνίδια

1.1 Εισαγωγή

"Ενα βιντεοπαιχνίδι είναι ένα ηλεκτρονικό παιχνίδι το οποίο περιλαμβάνει αλληλεπίδραση με μια διεπαφή χρήστη για την παραγωγή οπτικής ανάδρασης σε μια συσκευή βίντεο" [1]. Το πρώτο συνθετικό της λέξης βιντεοπαιχνίδι αναφέρεται σε μία συσκευή που προβάλλει γραφικά τόσο σε δισδιάστατη μορφή όσο και σε τρισδιάστατη. Τα ηλεκτρονικά συστήματα που χρησιμοποιούνται για να εμφανίζουν τα βιντεοπαιχνίδια αναφέρονται ως πλατφόρμες με κάποια παραδείγματα αυτών να είναι οι ηλεκτρονικοί υπολογιστές, οι κονσόλες βιντεοπαιχνιδιών και τα κινητά τηλέφωνα [1]. Σήμερα, τα βιντεοπαιχνίδια είναι ένα διαδεδομένος τρόπος τόσο ψυχαγωγίας όσο και επικοινωνίας για πολλούς ανθρώπους με τις δυνατότητες τους να πληθαίνουν συνεχώς.

1.2 Ιστορική Αναδρομή Βιντεοπαιχνιδιών

Η ιστορία των βιντεοπαιχνιδιών ξεκινάει στα τέλη της δεκαετίας του 1940. Περίπου στα τέλη του 1960 στην Αμερική άρχισαν να εισβάλουν στην καθημερινή ζωή των ανθρώπων οι ηλεκτρονικοί υπολογιστές. Από εκεί και έπειτα, έκαναν εμφάνιση τα πρώτα βιντεοπαιχνίδια στα φλίπερ, στους υπολογιστές και στις κονσόλες που ήταν συνήθως φορητές [1].

Η αρχαιότερη ηλεκτρονική συσκευή ήρθε στο φώς το 1947 από τους Thomas T. Goldsmith Jr και Estle Ray Mann και ήταν εμπνευσμένη στην οθόνη των ραντάρ. Η συσκευή αυτή επέτρεπε στον χρήστη να ελέγχει μία κουκίδα με σκοπό να προσομοιώσει ένα βλήμα που έριχνε στόχους, οι οποίοι απεικονίζονταν ως σταθερά σχέδια στην οθόνη [2]. Το πρώτο βιντεοπαιχνίδι ονομάστηκε Tennis for Two και αναπτύχθηκε από τον William Higginbotham το 1958 ενώ το παιχνίδι που ήταν ορόσημο για τα βιντεοπαιχνίδια ηλεκτρονικών υπολογιστών ήταν το SpaceWar που δημιουργήθηκε το 1961 [3],[4]. Τα βιντεοπαιχνίδια άρχισαν να γίνονται γνωστά την δεκαετία του 1970 μέσω της δημιουργίας των Arcade παιχνιδιών, τα οποία τοποθετούνταν σε μεγάλα εμπορικά κέντρα και λειτουργούσαν με κερματοδέκτες. Παράλληλα την ίδια δεκαετία δημιουργήθηκε η πρώτη γενιά κονσολών με κυρίαρχο και πιο δημοφιλές το Pong. Αυτά αποτελούσαν ευχάριστο τρόπο διασκέδασης για πολλούς ανθρώπους εκείνης της εποχής.

Έτσι ο δρόμος της εμπορικότητας των μηχανών Arcade άνοιξε το 1971 όταν δημιουργήθηκε το πρώτο βιντεοπαιχνίδι που πωλήθηκε στο εμπόριο και λειτουργούσε με κέρματα, με όνομα "Computer Space" [4]. Για την προβολή του συγκεκριμένου βιντεοπαιχνιδιού χρησιμοποιούνταν μία ασπρόμαυρη τηλεόραση καθώς και ένα σύστημα ηλεκτρονικών υπολογιστών. Το επόμενο έτος, το 1972, ακολούθησε η πρώτη έκδοση του Pong, το Pong της Atari που ήταν μία Arcade έκδοση. Η δεύτερη έκδοση του Pong ήρθε το 1975 και ήταν για οικιακή χρήση. Η μεγάλη εμπορική επιτυχία του Pong έδωσε το έναυσμα σε πολλές εταιρίες να δημιουργήσουν διάφορους κλώνους αυτού, με αποτέλεσμα της έναρξης της βιομηχανίας των βιντεοπαιχνιδιών [6].

Το τέλος του Pong μετά από έναν καταιγισμό κλώνων του στην αγορά ήρθε το 1977, με την επιτυχία του παιχνιδιού "Space Invaders" της εταιρίας Taito την επόμενη κιόλας χρονιά. Το παιχνίδι αυτό έδωσε το κίνητρο για την έναρξη της χρυσής εποχής των Arcade Games καθώς έδωσε έμπνευση σε πολλούς κατασκευαστές να εισέλθουν στο εμπόριο αυτών. Το "Space Invaders" στα επόμενα χρόνια

ξεκίνησε την συνεργασία του με το Atari VCS μέσω της άδειας που του παρείχε και έδωσε τρομερή άνοδο στις πωλήσεις της κονσόλας. Ωστόσο η κυριαρχία της βιομηχανίας των βιντεοπαιχνιδιών στράφηκε στο "στρατόπεδο" της Nintedo Entertainment System, η οποία μετά από λίγα χρόνια έδωσε μία ακόμα άνοδο στην βιομηχανία. Στις αρχές του 18ου αιώνα δημιουργήθηκαν και άλλες μεγάλες εταιρίες, με τις τρεις πιο δημοφιλής να είναι η Activision, η EA και η Sega. Από εκείνη την περίοδο και μετά η βιομηχανία των βιντεοπαιχνιδιών δέχεται όλο και μεγαλύτερη αύξηση, με νέες εταιρίες να μπαίνουν στο εμπορικό παιχνίδι [6].

Ερχόμενοι στην τρίτη γενιά κονσολών ή αλλιώς στην «Εποχή των 8-bit», το 1983 η Nintendo κυκλοφόρησε το Family Computer, το οποίο υποστήριζε παιχνίδια με ποικιλία χρωμάτων χάρη στα sprites υψηλής ανάλυσης καθώς η βελτίωση και των γραφικών ήταν εμφανής. Η Nintendo αργότερα με σκοπό να διαφοροποιήσει τα προϊόντα της από τις παλαιότερες αποτυχημένες κονσόλες, πρόσθεσε σε αυτές μία μπροστινή υποδοχή για τις κασέτες παιχνιδιών, ενώ περιείχε και στη συσκευασία το παιχνίδι Super Mario Brothers που πολύ γρήγορα έγινε ο μεγαλύτερος τίτλος παιχνιδιού στην αγορά [4].

Η Sega θέλοντας να κερδίσει μεγαλύτερο μερίδιο στην αγορά, ήταν η πρώτη που δημιούργησε τη δική της καινούργια κονσόλα (Mega Drive), στις αρχές της τέταρτης γενιάς ή αλλιώς της «Εποχής των 16-bit» όπως αυτή ονομάστηκε. Η εταιρία πέτυχε τον στόχο της με πολύ μεγάλο μερίδιο αγοράς και πιο επιτυχημένο παιχνίδι της να είναι το Sonic The Hedgehog [3].

Η Πέμπτη γενιά ουσιαστικά ξεκίνησε την περίοδο που αναπτύχθηκαν το Nintendo 64 και το Sony Playstation, καθώς πιο πριν οι κονσόλες που λανσαρίστηκαν δεν είχαν ιδιαίτερη επιτυχία. Μετά από μία ανεπιτυχή συνεργασία των εταιριών Sony και Nintendo, η Sony πήρε την μεγάλη απόφαση να εξάγει στην αγορά την πρώτη της κονσόλας που ονομάστηκε Sony PlayStation. Η κονσόλα αυτή κατάφερε να πουλήσει πάνω από 100.000.000 τεμάχια σε όλο τον κόσμο, δημιουργώντας έναν φόβο στην Nintendo που έβλεπε τα νούμερα να αυξάνονται κατακόρυφα στο αντίπαλο τους "στρατόπεδο". Η Nintendo έτσι θέλοντας να απαντήσει παρουσίασε το Nintendo 64 που αποδείχτηκε μία ποιοτική κονσόλα, η οποία όμως μέτρησε κάτω από 40.000.000 με αποτέλεσμα η Sony να κατέχει τα ηνία της κορυφής [3].

Μπαίνοντας στην έκτη γενιά πλέον η Sony μετά την μεγάλη επιτυχία της, κυκλοφόρησε στις αρχές του 2000 το PlayStation 2 που ήταν μια 128 bit κονσόλα. Η συγκεκριμένη κονσόλα επέτρεπε στους developers να δημιουργούν δικούς τους servers, κάτι αρκετά πρωτότυπο για την τότε εποχή. Η επιτυχία του PlayStation 2 ήταν ακόμα μεγαλύτερη, με την απάντηση να έρχεται από την Microsoft μέσω της κονσόλας Xbox η οποία περιείχε πολλές καινοτομίες. Με αυτόν τον τρόπο η εταιρία κατάφερε να κερδίσει ένα μερίδιο στην αγορά, χωρίς όμως να καταφέρει και πάλι να μπει στην πρώτη θέση των παγκόσμιων πωλήσεων. Όσον αφορά την Sega και την Nintendo, η προσπάθεια τους δεν ήταν επαρκής για να "χτυπήσουν" τις αντίπαλες εταιρίες τους, με τις κονσόλες τους να κερδίζουν μικρό ποσοστό της αγοράς [3].

Κλείνοντας και μπαίνοντας πλέον στην έβδομη γενιά, οι δύο μεγάλες εταιρίες κυκλοφόρησαν δύο πολύ δυνατές φορητές κονσόλες. Από την πλευρά της η Sony δημιούργησε το PSP, το οποίο άγγιζε το επίπεδο των γραφικών του PlayStation 2. Η συσκευή μπορούσε να παρέχει ασύρματη σύνδεση στο διαδίκτυο καθώς και σύνδεση με τηλεοράσεις. Από την άλλη πλευρά η Nintendo δημιούργησε το DS που αποτελούνταν από δύο οθόνες εκ των οποίων η μία αφής. Η συσκευή παρείχε σύνδεση με άλλες συσκευές DS, εφόσον ήταν σε κοντινή απόσταση. Έτσι δόθηκε η δυνατότητα στους παίκτες να παίζουν παράλληλα μαζί, κάτι που κέρδισε το ενδιαφέρον του κόσμου. Μέσω της αναβάθμισης της Nintendo με το DSi, οι χρήστες πλέον μπορούσαν να συνδεθούν στον παγκόσμιο ιστό μέσω WiFi και

να κατεβάσουν παιχνίδια από το Store που δημιούργησε η εταιρία αποκλειστικά για την συγκεκριμένη κονσόλα. Λίγο αργότερα και συγκεκριμένα το 2005 η Microsoft κυκλοφόρησε το Xbox 360 που ήταν μία οικιακή κονσόλα με πολλές νέες δυνατότητες. Τέλος την απάντηση της δύο χρόνια αργότερα έδωσε η Sony με την δημιουργία του PlayStation 3 που αποτέλεσε μία κονσόλα με πολλές καινούργιες τεχνολογίες στον χώρο αυτό [2-3].

Επί του παρόντος, βρισκόμαστε πλέον στην όγδοη γενιά κονσολών και οι δυνατότητες γραφικών συνδυάζονται με πολυμέσα και αλληλεπίδραση κάτι που οφείλεται στην ευρεία χρήση του διαδικτύου στα περισσότερα παιχνίδια που κυκλοφορούν στην αγορά αυτή τη στιγμή.

1.3 Κατηγορίες Βιντεοπαιχνιδιών

Στο σημείο αυτό θα μιλήσουμε για τις διάφορες κατηγορίες παιχνιδιών που υπάρχουν και έχουν αναπτυχθεί με την πάροδο των χρόνων. "Από το 1958 έγινε μία προσπάθεια ταξινόμησης των ηλεκτρονικών παιχνιδιών. Τότε είχαν ταξινομηθεί σε τέσσερις βασικές κατηγορίες: παιχνίδια ανταγωνισμού, τύχης, προσομοίωσης και κίνησης."[8] Ακολούθησαν πολλές κατηγοριοποιήσεις παιχνιδιών από ερευνητές κάτι που όπως αποδείχθηκε είναι αρκετά δύσκολο, καθώς συνεχώς προκύπτουν νέα είδη παιχνιδιών. Τα ηλεκτρονικά παιχνίδια στοιβάζονται κυρίως με βάση το gameplay τους, καθώς δε παίζει ρόλο το περιβάλλον που εξελίσσονται. Υπάρχουν ωστόσο παιχνίδια που δεν ανήκουν μόνο σε μία κατηγορία, αλλά περιέχουν στοιχεία από δύο ή περισσότερα είδη ταυτόχρονα όπως για παράδειγμα το GTA που συνδυάζει στοιχεία από παιχνίδια αγώνων, ρόλων και Shooting. Αρχικά μια πολύ σημαντική διαχώριση των παιχνιδιών είναι ως προς τις διαστάσεις που έχουν στα γραφικά τους. Οι δύο πολύ γνωστές σε όλους κατηγορίες είναι τα Δισδιάστατα παιχνίδια (2D) και Τρισδιάστατα (3D). Τα τελευταία χρόνια το μεγαλύτερο ποσοστό των παιχνιδιών έχουν τρεις διαστάσεις καθώς είναι πολύ πιο εντυπωσιακά τα γραφικά τους και προτιμούνται περισσότερο από τους χρήστες.

Πέρα από την διαχώριση των παιχνιδιών ως προς τις διαστάσεις στα γραφικά τους, τα παιχνίδια διαχωρίζονται και ως προς το είδος τους. Παρακάτω θα αναφερθούν κάποιες βασικές κατηγορίες: [6-10]

1.3.1 Παιχνίδια Δράσης-Action Games

Το μεγαλύτερο ποσοστό των παιχνιδιών που κυκλοφορούν βρίσκονται σε αυτή την κατηγορία. Στα παιχνίδια αυτού του είδους ο χρήστης χρειάζεται να έχει γρήγορα αντανακλαστικά και σωστό συγχρονισμό για να μπορέσει να επιτύχει τις δοκιμασίες που του θέτει το παιχνίδι. Ο στόχος είναι συνήθως το πιο σημαντικό κομμάτι καθώς και η ταχύτητα αυτού, όσο ανεβαίνει το επίπεδο. Ο χρήστης στην κατηγορία αυτή έρχεται σε αλληλεπίδραση τόσο με το περιβάλλον, όσο και με τα όντα που βρίσκονται σε αυτό, με σκοπό να βγάλει εις πέρας όλα όσα του ζητούνται . Τα Action Games, χωρίζονται σε δύο βασικές κατηγορίες. Συγκεκριμένα διακρίνουμε τα Shooters παιχνίδια και τα Fighting. Ξεκινώντας από το πρώτο αξίζει να σημειωθεί ότι είναι από τα δημοφιλέστερα παιχνίδια τα τελευταία χρόνια. Η διάκριση των Shooters Games γίνεται με βάση την θέση της κάμερας ενώ υπάρχει δυνατότητα και αλληλοεπικάλυψης καθώς πολλοί σύγχρονοι τίτλοι επιτρέπουν την εναλλαγή ανάμεσα στις δύο αυτές περιπτώσεις. Έτσι στην πρώτη περίπτωση ο χρήστης βλέπει μέσα από τα μάτια του παίκτη που διαχειρίζεται και το παιχνίδι ονομάζεται **First Person Shooter (FPS)** και στην δεύτερη περίπτωση ο χρήστης βλέπει ολόκληρο τον χαρακτήρας που διαχειρίζεται και το παιχνίδι ονομάζεται **Third Person Shooter (TPS)**. Σε όλες τις καταστάσεις που αναφέραμε παραπάνω, ο

παίκτης είτε λειτουργεί αυτόνομα και προσπαθεί να σκοτώσει τους αντιπάλους του, είτε λειτουργεί στα πλαίσια μιας ομάδας, προσπαθώντας ή να πετύχει κάποιον στόχο, ή συλλογικά να σκοτωθούν οι παίκτες της αντίπαλης ομάδας. Κάποια παραδείγματα αυτού του είδους είναι το Call Of Duty και το Counter Strike.

Εικόνα 1 - 1: Παιχνίδι Shooter

Από την άλλη πλευρά στα **Fighting Games** ο παίκτης ελέγχει έναν μόνο χαρακτήρα. Στις περισσότερες περιπτώσεις αυτής της κατηγορίας ο παίκτης μπαίνει σε μια διαδικασία επιλογής του χαρακτήρα που θα χειριστεί. Ο κάθε χαρακτήρας έχει τις δικές του πολεμικές ικανότητες και τη δική του αμφίεση. Όταν ξεκινάει το παιχνίδι απέναντι από τον χαρακτήρα που διαχειρίζεται ο παίκτης υπάρχει ένας αντίπαλος, τον οποίο είτε διαχειρίζεται κάποιος άλλος παίχτης, είτε έχει προστεθεί κατά την δημιουργία του παιχνιδιού και έχει αυτοματοποιημένες κινήσεις. Ο κάθε χαρακτήρας έχει ζωή και νικητής του παιχνιδιού είναι αυτός που θα καταφέρει πρώτος να τελειώσει την ζωή του αντιπάλου του. Κάποια πολύ γνωστά παιχνίδια αυτού του είδους είναι το WWE και το Mortal Kombat.

Εικόνα 1 - 2: Παιχνίδι Fighting

1.3.2 Multiplayer Online Battle Area (MOBA)

Τα MOBA Games είναι πλήρως βασισμένα στο multiplayer, που σημαίνει ότι παίκτες από όλο τον κόσμο έχουν την δυνατότητα να παίζουν ταυτόχρονα. Στην συγκεκριμένη κατηγορία οι παίκτες παίζουν συνήθως σε ομάδες ελέγχοντας έναν χαρακτήρα μέσα από μία μεγάλη γκάμα που τους προσφέρετε. Σκοπός του παιχνιδιού είναι η καταστροφή της βάσης της αντίπαλης ομάδας και η εξόντωση των αντιπάλων τους. Ο παίκτης επίσης πρέπει να αγοράσει αντικείμενα με τα οποία γίνεται

πιο δυνατός τόσο στην αμυντική ικανότητα του, όσο και στην επιθετική. Η αναγνώριση του παίκτη προκύπτει από μία κατάταξη που μπορούν να δούν οι παίκτες τόσο κατά την διάρκεια του παιχνιδιού, όσο και στο τέλος αυτού. Το κάθε παιχνίδι συνήθως διαρκεί από 15 ως 45 λεπτά, χωρίς αυτό να αποτελεί κανόνα. Στο τέλος του παιχνιδιού ο παίκτης ανταμείβεται με κάποιους πόντους αναλόγως με το αποτέλεσμα αυτού. Πολύ γνωστά παραδείγματα αυτής της κατηγορίας είναι το Dota και το League of Legends.

Εικόνα 1 - 3: Παιχνίδι MOBA

1.3.3 Massively Multiplayer Online Role-Playing Games (MMORPGs)

Τα παιχνίδια που απαρτίζουν αυτή την κατηγορία έχουν βάση στην multiplayer μορφή. Ο παίκτης διαχειρίζεται έναν χαρακτήρα σε ένα περιβάλλον που μπορούν να παίξουν ταυτόχρονα άτομα από όλο τον κόσμο. Κάθε παίκτης ξεκινάει από το πρώτο επίπεδο και σκοπός του είναι τόσο να δυναμώσει τον χαρακτήρα του μέσω των αντικειμένων που αποκτάει, όσο και να ανεβαίνει επίπεδα μέσω του XP όπως είναι γνωστό στα άτομα που ασχολούνται με τα συγκεκριμένα παιχνίδια. Μία ακόμα σημαντική πτυχή του είδους αυτού είναι ότι προσφέρει την επιλογή δύο διαφορετικών game play. Στην πρώτη περίπτωση ο παίκτης αντιμετωπίζει όντα που βρίσκονται στο περιβάλλον τα οποία διαχειρίζεται το σύστημα και το game play ονομάζεται Person Versus Environment. Στην δεύτερη περίπτωση ο παίκτης έρχεται αντιμέτωπος με παίκτες που διαχειρίζονται άλλα άτομα και το game play ονομάζεται Person Versus Person. Υπάρχουν πολλά παιχνίδια από τα οποία αποτελείται η συγκεκριμένη κατηγορία, με πιο δημοφιλή να είναι το World of Warcraft και το 4Story.

Εικόνα 1 - 4: Παιχνίδι MMORPGs

1.3.4 Παιχνίδια Αθλητισμού-Sport Games

Η συγκεκριμένη κατηγορία περιλαμβάνει όλα τα παιχνίδια που αναπαριστούν κάθε είδους σπορ-άθλημα και παίζονται είτε μεταξύ παικτών, είτε μεταξύ ενός παίκτη και ενός χρήστη που δημιουργεί και διαχειρίζεται η μηχανή. Σε αντίθεση με πολλά παιχνίδια που λαμβάνουν χώρα σε ένα μέρος που είναι άγνωστο προς τον παίκτη, εδώ απεικονίζεται ένας αληθινός κόσμος που είναι γνώριμος στα μάτια αυτού. Στην κατηγορία αυτή ο παίκτης έχει να διαχειριστεί είτε έναν χαρακτήρα, είτε ένα σύνολο χαρακτήρων. Σε μερικά παιχνίδια της κατηγορίας αυτής ο χρήστης έχει τον ρόλο του οργανωτή-προπονητή μίας ομάδας και προσπαθεί να την διαχειριστεί έτσι ώστε να αποδώσει όσο το δυνατόν καλύτερα. Στόχος του παιχνιδιού είναι διαφορετικός κάθε φορά, ανάλογα με το άθλημα που εξελίσσεται στο παιχνίδι. Παραδείγματα αυτής της κατηγορίας είναι το FIFA και το NBA 2K.

Εικόνα 1 - 5: Παιχνίδι Αθλητισμού

1.3.5 Παιχνίδια Στρατηγικής–Strategy Games

Τα παιχνίδια στρατηγικής αποσκοπούν στο να προκαλούν τον παίκτη να κερδίσει το παιχνίδι μέσω της οργάνωσης και του σχεδιασμού διαφόρων ενεργειών που θα τον βοηθήσουν είτε να κατακτήσει την βάση του αντιπάλου του είτε να μειώσει τις δυνάμεις αυτού. Σε αυτή την περίπτωση ο χρήστης χειρίζεται μια ομάδα χαρακτήρων-ατόμων. Η πιο γνωστή μορφή της κατηγορίας αυτής, είναι τα παιχνίδια **Real Time Strategy** στα οποία οι παίκτες χρειάζεται να παίρνουν συνεχώς τις κατάλληλες

αποφάσεις για να πετύχουν τον σκοπό του παιχνιδιού. Επιπρόσθετα τους δίνεται η δυνατότητα να χτίζουν στις βάσεις τους κτίρια και να εκμεταλλεύονται διάφορους πόρους με σκοπό να υπερέχουν σε σχέση με τους αντίπαλους παίκτες. Μοναδικός παράγοντας για την επιτυχία των παιχνιδιών στρατηγικής είναι η σωστή οργάνωση του παίκτη και η επιλογή των καλύτερων αποφάσεων κάθε φορά. Γνωστά παραδείγματα αυτού του είδους είναι το Age Of Mythology και το Age Of Empires.

Εικόνα 1 - 6: Παιχνίδι Στρατηγικής

1.3.6 Παιχνίδια Προσομοίωσης–Simulation Games

Τα παιχνίδια αυτά βασίζονται στην όσο το δυνατόν ρεαλιστικότερη μεταφορά μίας συγκεκριμένης κατάστασης με κοινό χαρακτηριστικό των υποκατηγοριών αυτών να είναι η προσομοίωση πραγματικών ή φανταστικών κόσμων. Έτσι αναλύοντας τα παιχνίδια προσομοίωσης διακρίνονται οι εξής υποκατηγορίες: Social Simulation Games, Vehicle Simulation και Construction and Management Simulation. Έτσι αρχικά στα **Social Simulation Games** δίνεται συνήθως παραπάνω έμφαση στις κοινωνικές σχέσεις του χαρακτήρα που διαχειρίζεται ο παίκτης, είτε με άλλους παίκτες σε Online μορφή, είτε με προκατασκευασμένους χαρακτήρες από το παιχνίδι χωρίς την ανάγκη σύνδεσης στο διαδίκτυο. Επίσης σε μερικές περιπτώσεις αυτής της κατηγορίας ο χρήστης βιώνει μια ζωή στα πλαίσια του παιχνιδιού παίρνοντας αποφάσεις και επιλέγοντας τον τρόπο που αυτός θέλει να ενεργεί κάθε στιγμή. Στην δεύτερη περίπτωση, στα **Vehicle Simulation** ο παίκτης έχει την δυνατότητα να χειριστεί κάτι που στην πραγματικότητα ίσως ήταν δύσκολο, όπως για παράδειγμα ένα αεροπλάνο. Τέλος στα **Construction and Management Simulation** παιχνίδια, ο παίκτης καλείται να κατασκευάσει και να διαχειριστεί πόλεις και κοινότητες, μέσω των πόρων που του παρέχονται. Σκοπός του παιχνιδιού εδώ είναι να ολοκληρωθούν πλήρως οι στόχοι που θέτονται από το σύστημα. Παραδείγματα των παιχνιδιών προσομοίωσης είναι το Sims,το ThemePark και το Flight Simulator που τον τελευταίο καιρό έχει ενθουσιάσει πολύ μεγάλο ποσοστό του κόσμου που ασχολείται με τα βιντεοπαιχνίδια.

Εικόνα 1 - 7: Παιχνίδι Προσομοίωσης

1.4 Ομάδα Ανάπτυξης Βιντεοπαιχνιδιών

Για να θεωρείται επιτυχημένη μία εταιρία ανάπτυξης παιχνιδιών θα πρέπει να διαθέτει μία ολοκληρωμένη και έμπειρη ομάδα ανάπτυξης. Σε πιο ερασιτεχνικά επίπεδα η δημιουργία παιχνιδιού μπορεί να γίνει ακόμα και από ένα μόνο άτομο, το οποίο όμως έχει να φέρει εις πέρας πολλές ενέργειες για την υλοποίηση του παιχνιδιού. Ωστόσο υπάρχουν εταιρίες-ομάδες των οποίων τα άτομα δουλεύουν συλλογικά, ο κάθε ένας στο πόστο που γνωρίζει και ειδικεύεται, με απώτερο σκοπό την υλοποίηση ενός παιχνιδιού. Κάτι πολύ σημαντικό που αξίζει να σημειωθεί είναι ότι θα πρέπει τα μέλη της ομάδας να έχουν κάποια πειθαρχικά χαρακτηριστικά, συγκεκριμένα να είναι επίμονοι, εργατικοί και ευέλικτοι. Έτσι καταλαβαίνουμε ότι στην ομάδα ανάπτυξη ενός παιχνιδιού διακρίνονται κάποιοι ρόλοι. Οι πιο γνωστοί και παράλληλα σημαντικοί είναι ο Σχεδιαστής, ο Καλλιτέχνης, ο Προγραμματιστής, ο Μηχανικός/Σχεδιαστής Ήχου, ο Δοκιμαστής, ο Μηχανικός/Σχεδιαστής Μουσικής και ο Διαχειριστής Έργου. [11-14]

1.4.1 Σχεδιαστής (Game Designer)

Ο Σχεδιαστής μαζί με τον Προγραμματιστή αποτελούν τους πιο σημαντικούς ρόλους στον τομέα της ανάπτυξης παιχνιδιών. Ο πρώτος είναι το άτομο που ουσιαστικά φέρει όλες τις ιδέες πάνω στις οποίες θα πατήσουν όλα τα υπόλοιπα μέλη της ομάδας και θα δουλέψουν. Στην δημιουργία παιχνιδιών όλα ξεκινούν από την ιδέα, η οποία όμως μπορεί να φανεί ‘περιττή’ αν δεν υπάρξει η σωστή υλοποίηση. Τα άτομα που ασχολούνται με αυτό τον ρόλο πρέπει να είναι αρκετά δημιουργικά και να έχουν οργανωτικές δεξιότητες καθώς πρέπει να μοιράσουν αυτή την ιδέα σε όλα τα μέλη της ομάδας. Τέλος εκτός από την ιδέα που αναφέραμε, ο Σχεδιαστής είναι φέρει ευθύνη και για την δημιουργία της δομής αλλά και των κανόνων του παιχνιδιού.

1.4.2 Καλλιτέχνης (Artist)

Ο ρόλος του Καλλιτέχνη είναι πολύ σημαντικός αλλά και δύσκολος παράλληλα. Τα άτομα που ασχολούνται με αυτό τον τομέα χρειάζεται να έχουν πολύ μεγάλη εμπειρία καθώς το αποτέλεσμα της δουλειάς του είναι αυτό που θα δουν οι παίκτες στην οθόνη τους. Έχουν να διαχειριστούν πάρα πολλές υφές και να τις ενώσουν ομοιόμορφα έτσι ώστε να φαίνεται πιο ελκυστικό το παιχνίδι στον

χρήστη. Τέλος ξοδεύουν πολύ χρόνο αλληλεπιδρώντας με άλλα μέλη της ομάδας καθώς πρέπει να βρίσκονται σε συνεχή επικοινωνία τόσο με τον σχεδιαστή για να λαμβάνουν οδηγίες και να μοιράζονται ιδέες για την εμφάνιση της σκηνής, όσο και με τον προγραμματιστή καθώς τα αντικείμενα που δημιουργεί θα πρέπει να μπορούν να δεχτούν κάποιες ιδιότητες.

1.4.3 Προγραμματιστής (Programmer)

"Είναι ο πρώτος ρόλος που πρέπει να σκεφτεί μία ομάδα ανάπτυξης παιχνιδιών".[12] Είναι το πρόσωπο που δίνει σάρκα και οστά στις ιδέες των σχεδιαστών, οπότε η επιλογή του είναι πολύ δύσκολη καθώς πρέπει πάντα να προτιμούνται έμπειρα άτομα. Και αυτό γιατί ακόμα και αν υπάρχει μία πολύ μεγάλη ιδέα για κάτι, αυτή μπορεί να χαθεί αν δεν υπάρχει το σωστό άτομο να την υλοποιήσει. Ο Προγραμματιστής απαιτείται να έχει γνώση της γλώσσας προγραμματισμού που χρησιμοποιείται από την μηχανή που δημιουργείται το παιχνίδι καθώς θα κληθεί και να επιλύσει με ταχύ ρυθμό σφάλματα που πιθανώς θα προκύψουν. Αυτός μέσω του κώδικα προγραμματισμού που αναπτύσσει, επιτρέπει στον χρήστη να αλληλεπιδρά με το περιβάλλον αλλά και τους χαρακτήρες του παιχνιδιού. Τέλος αξίζει να σημειωθεί ότι οι δύο πιο δημοφιλής γλώσσες δημιουργίας παιχνιδιών είναι η C++ και η C# .

1.4.4 Μηχανικός / Σχεδιαστής Ήχου (Sound Engineer)

Υπεύθυνος για το κομμάτι ήχου του παιχνιδιού είναι ο Sound Engineer. Μέσω των ήχων και των ηχητικών εφέ ο Μηχανικός / Σχεδιαστής Ήχου προσφέρει περισσότερη ρεαλιστικότητα στο παιχνίδι και βελτιώνει την εμπειρία του χρήστη. Σε πολλές περιπτώσεις πρέπει να δημιουργήσει ήχους που θα αντιστοιχίζονται σε αλληλεπιδράσεις του παιχνιδιού.

1.4.5 Δοκιμαστής (Tester)

Είναι το άτομο που δοκιμάζει το παιχνίδι σε όλη την διάρκεια της δημιουργίας του. Ελέγχει τόσο την ποιότητα των γραφικών και επικοινωνεί με τον Σχεδιαστή για πιθανά σφάλματα και βελτιώσεις, όσο και την ροή του παιχνιδιού και συνομιλεί με το αντίστοιχο πρόσωπο, ανάλογα με το κομμάτι του παιχνιδιού που προέκυψε το σφάλμα. Είναι πολύ σημαντικό να υπάρχει ένα πολύ έμπειρο άτομο σε αυτή τη θέση για να μπορέσει να βρίσκει τα σφάλματα στα διάφορα κομμάτια του παιχνιδιού καθώς και να υπάρχει άψογη επικοινωνία ανάμεσα σε αυτόν και στα υπόλοιπα μέλη της ομάδας.

1.4.6 Συγγραφέας (Writer)

Η ύπαρξη αυτού του ατόμου δεν είναι απαραίτητη σε όλα τα είδη των παιχνιδιών, είναι όμως σε παιχνίδια που περιέχουν διαλόγους και κείμενα καθώς είναι ο υπεύθυνος για την ιστορία του παιχνιδιού. Μέσω των διαλόγων και των κειμένων ο Συγγραφέας δίνει ρεαλιστικότητα στο παιχνίδι καθώς κρατάει και την αγωνία του χρήστη για την συνέχεια αυτού. Έτσι θα λέγαμε ότι το κείμενο και οι διάλογοι είναι ζωτικής σημασίας για την δημιουργία παιχνιδιών.

1.4.7 Διαχειριστής Έργου (Project Manager)

Ο Διαχειριστής Έργου είναι ο υπεύθυνος για την οργάνωση των μελών του Project καθώς βασικός του ρόλος είναι να παρακολουθεί την ομάδα. Είναι το άτομο που φροντίζει να τηρούνται τα χρονοδιαγράμματα, παρακινώντας τα άτομα που καθυστερούν την ολοκλήρωση του έργου. Πρέπει να έχει πολύ υψηλές οργανωτικές δεξιότητες και να μπορεί να ηγηθεί στα μέλη της ομάδας, έτσι ώστε να λύνονται πιθανά θέματα που προκύπτουν. Έτσι μπορούμε να πούμε ότι η δουλειά αυτού του ατόμου είναι εξαιρετικά σημαντική για την πορεία του έργου. Σε αρκετές περιπτώσεις κυρίως μικρών Project, δεν υπάρχει ξεχωριστό άτομο για αυτή την θέση, οπότε τον ρόλο αυτό τον αναλαμβάνει συνήθως ο Σχεδιαστής του παιχνιδιού.

Καταλαβαίνουμε έτσι ότι για να υπάρχει επιτυχία σε ένα Project, η ομάδα πρέπει να είναι ισχυρή, με όρεξη και να υπάρχει καλή επικοινωνία και συνεργασία ανάμεσα στα μέλη αυτής. Θα πρέπει όλα τα μέλη να ακολουθούν τα χρονοδιαγράμματα που έχουν τεθεί, καθώς ακόμα και ένα μόνο κομμάτι να λείπει, αρκεί για να χαλάσει η συνολική εικόνα. Τα μελή και ο αριθμός αυτών από τα οποία αποτελείται κάθε ομάδα, εξαρτώνται από τις απαιτήσεις του κάθε παιχνιδιού. Σε κάθε περίπτωση όμως η ανάπτυξη παιχνιδιών απαιτεί μια ποικιλία διαφορετικών δεξιοτήτων.

1.5 Στάδια Ανάπτυξης Βιντεοπαιχνιδιών

Η ανάπτυξη παιχνιδιών δεν είναι εύκολη ούτε απλή διαδικασία. Κρύβει πάρα πολλές δυσκολίες, οι οποίες εμφανίζονται σε όλη την περίοδο που διαρκεί η ανάπτυξη ενός παιχνιδιού. Κατά την διάρκεια του κύκλου ανάπτυξης τα μέλη της ομάδας έχουν να αντιμετωπίσουν θέματα όπως είναι οι προθεσμίες για περάτωση εργασιών, τα σημεία συμφόρησης στην παραγωγή καθώς και οι πολλές ώρες εργασίας προκειμένου να βγει ένα σωστό και αρμονικό αποτέλεσμα. Η διαδικασία αυτή περνάει από κάποια στάδια, έτσι ώστε τελικά να φτάσει τελικά το παιχνίδι στα χέρια του κάθε παίκτη. Παρακάτω θα αναφέρουμε και θα αναλύσουμε όλα τα στάδια ανάπτυξης ενός παιχνιδιού.[15-17]

1.5.1 Σχεδιασμός Βιντεοπαιχνιδιού

Το πρώτο στάδιο είναι η ρίζα της ανάπτυξης βιντεοπαιχνιδιών. Αρχικά θα χρειαστεί να δημιουργηθεί ένα έγγραφο σχεδίασης παιχνιδιού το οποίο θα είναι το σχεδιάγραμμα που θα ακολουθηθεί για την παραγωγή αυτού, δημιουργώντας έτσι μια γενική ιδέα του. Στο στάδιο αυτό θέτονται αρχικά κάποιες πολύ γενικές ερωτήσεις μέσω των οποίων αποκτάται μία αντίληψη για το πως θα είναι η τελική μορφή του παιχνιδιού. Κάποια βασικά ερωτήματα είναι :

-Ποια είναι η βασική ιδέα πίσω από το παιχνίδι;

-Θα είναι δύο ή τριών διαστάσεων;

-Ποιο θα είναι το είδος του βιντεοπαιχνιδιού;

-Πόσους χαρακτήρες θα έχει και ποιοι θα είναι αυτοί;

-Ποιος θα είναι ο στόχος του παιχνιδιού;

-Μέσω ποιας μηχανής θα δημιουργήσουμε το παιχνίδι;

Εφόσον απαντηθούν τα ερωτήματα αυτά, πλέον θα έχει δημιουργηθεί μία ραχοκοκαλιά του εγγράφου σχεδίασης παιχνιδιού. Επίσης οι εκδότες θα μπορούν πλέον και αυτοί να ξέρουν τι θα περιμένουν σαν αποτέλεσμα. Επόμενο κομμάτι αυτού του σταδίου είναι να ελέγξουμε κατά πόσο είναι εφικτά αυτά που έχουμε αποφασίσει καθώς και να αλλάξουμε κάτι αν αυτό μας φαίνεται ευκολότερα εφικτό. Από κει και πέρα προκύπτουν κάποια ακόμα σημαντικά ερωτήματα που έχουν να κάνουν με το κομμάτι της παραγωγής του παιχνιδιού. Για παράδειγμα :

-Από πόσα άτομα θα περιλαμβάνεται η ομάδα;

-Πόσα άτομα από κάθε ρόλο χρειαζόμαστε;

-Ποιος είναι ο εκτιμώμενος χρόνος μέχρι την τελική κυκλοφορία;

-Τι κόστος θα έχει το project μας;

-Πως θα δημιουργηθούν έσοδα από το παιχνίδι;

-Έχουμε τα κατάλληλα μηχανήματα για να υποστηρίξουν αυτό το έργο;

Άρα όπως καταλαβαίνουμε οι αποφάσεις που πρέπει να παρθούν είναι πολλές και σημαντικές. Όταν το project που θέλουμε να δημιουργήσουμε είναι υπό την εποπτεία ενός εκδότη, τότε τα χρονοδιαγράμματα και οι προϋπολογισμοί είναι πολύ πιο αυστηροί.

1.5.2 Προ-Παραγωγή Βιντεοπαιχνιδιού

Στο στάδιο αυτό όλα τα μέλη της ομάδας έρχονται σε άμεση επικοινωνία με σκοπό να καθορίσουν το πως θα διαμορφωθεί κάθε κομμάτι του παιχνιδιού, αλλά και το πως θα συνδυαστούν αυτά μεταξύ τους. Συγκεκριμένα ο σχεδιαστής επικοινωνεί με τα υπόλοιπα μέλη για να δει αν είναι εφικτά και υλοποιήσιμα όλα όσα έχει σκεφτεί. Έτσι για παράδειγμα συσκέπτεται με τον προγραμματιστή, για το αν μπορούν να πραγματοποιηθούν μέσω του κώδικα, κάποιες ενέργειες του χαρακτήρα και με τον συγγραφέα για το πως θα γραφτεί το σενάριο και οι διάλογοι του παιχνιδιού. Επίσης στο στάδιο αυτό ελέγχεται το περιβάλλον, οι διεπαφές και άλλα στοιχεία του παιχνιδιού τόσο για την εμφάνιση τους όσο και για την πρακτικότητα τους. Εφόσον έχουν γίνει οι απαραίτητες συζητήσεις και όλοι οι έλεγχοι, το έργο περνάει στο επόμενο στάδιο που είναι η παραγωγή του.

1.5.3 Παραγωγή Βιντεοπαιχνιδιού

Η παραγωγή είναι το πιο περίπλοκο αλλά και προκλητικό στάδιο στην ανάπτυξη βιντεοπαιχνιδιών, καθώς εδώ παίρνει ζωή η ιδέα των σχεδιαστών. Κάθε μέλος της ομάδας ξεκινάει και εργάζεται με σκοπό να βγάλει εις πέρας την δουλειά που του έχει τεθεί. Για παράδειγμα οι σχεδιαστές δημιουργούν το περιβάλλον στα πλαίσια που αυτό τους έχει ζητηθεί. Οι προγραμματιστές γράφουν πλήθος γραμμών κώδικα, έτσι ώστε να δώσουν λειτουργικότητα σε όλους τους χαρακτήρες και τα κομμάτια του περιβάλλοντος. Ο Διαχειριστής Έργου φροντίζει την ομαλή λειτουργία και τον καλό συντονισμό όλων των μελών αφού η επικοινωνία τους είναι θα παίξει μεγάλο ρόλο στο τελικό αποτέλεσμα. Όπως καταλαβαίνουμε στο κομμάτι αυτό η ομάδα καταναλώνει τον περισσότερο χρόνο της καθώς κατά την διάρκεια της παραγωγής προκύπτουν πολλά σφάλματα τα οποία καλείται να λύσει. Επίσης προσφέρεται το μεγαλύτερο μέρος των πόρων της, αφού αυτοί είναι θα που θα δώσουν και τα έσοδα που είναι και το ζητούμενο. Τέλος πολλές φορές τυχαίνει μερικά κομμάτια του έργου να μην

λειτουργούν σωστά με αποτέλεσμα να φτάνουν ακόμα και στο σημείο της απόρριψης, κάτι που είναι λυπηρό για τα άτομα που έχουν δαπανήσει ώρες δουλειάς για την υλοποίηση του.

1.5.4 Στάδιο Δοκιμών

Στο σημείο αυτό αφού η ομάδα έχει ολοκληρώσει το έργο της, περνάμε στην διαδικασία των δοκιμών. Τα ανώτερα στελέχη της ομάδας αφιερώνουν κάποιες ώρες, προσπαθώντας να ελέγξουν αν όλα τα κομμάτια του έργου αλληλεπιδρούν αρμονικά. Έτσι ελέγχονται όλα τα σημεία του περιβάλλοντος τόσο για την εμφάνιση τους, όσο για την λειτουργικότητα τους. Επίσης παίζοντας το παιχνίδι, ελέγχουν αν αυτό φαίνεται διασκεδαστικό και αν κρατάει την αγωνία του παίκτη. Στην περίπτωση που υπάρχουν διάλογοι, επαληθεύεται η σωστή ροή τους, διότι ένα κακογραμμένο κείμενο παιχνιδιού μπορεί να το κάνει βαρετό στα μάτια των παικτών.

1.5.5 Προ-Λανσάρισμα

Εδώ πλέον το παιχνίδι είναι υλοποιημένο με τις αμφιβολίες για σφάλματα να είναι ελάχιστες, καθώς έχουν γίνει πολλές δοκιμές για την σωστή λειτουργία του. Στο στάδιο αυτό οι εκδότες μαζί με τα ανώτερα στελέχη του έργου, δίνουν έμφαση στο κομμάτι του μάρκετινγκ το οποίο ξεκινάει από τα τελευταία κιόλας στάδια της παραγωγής. Συγκεκριμένα προσπαθούν να δημιουργήσουν μια διαφήμιση του παιχνιδιού, η οποία θα κεντρίσει το ενδιαφέρον του κόσμου, έτσι ώστε να έχει επιτυχία κατά το λανσάρισμα του. Μεγάλες εταιρίες προβάλλουν τα παιχνίδια που πρόκειται να κυκλοφορήσουν, σε μεγάλες εκθέσεις που αφορούν τον συγκεκριμένο κλάδο. Στις εκθέσεις αυτές συμμετέχουν άτομα που ασχολούνται ακόμα και επαγγελματικά με τον χώρο αυτό και έτσι είναι ίσως η καλύτερη διαφήμιση που μπορεί να επιτευχθεί.

1.5.6 Λανσάρισμα – Κυκλοφορία Παιχνιδιού

Το πιο συναρπαστικό κομμάτι για όλα τα μέλη της ομάδας που δούλεψαν σκληρά για να βγει ένα σωστό αποτέλεσμα. Παράλληλα όμως είναι και ένα αγχωτικό κομμάτι για την ομάδα καθώς έχει αγωνία για το πόσο επιτυχημένο θα είναι το παιχνίδι στο θέμα πωλήσεων, αλλά και για τα πιθανά σφάλματα που μπορεί να προκύψουν στους χρήστες. Έτσι η ομάδα πρέπει να είναι σε ετοιμότητα να λύσει όλα τα σφάλματα που αναφέρουν οι χρήστες στα διαδικτυακά φόρουμ, μέσω ενημερώσεων που προσφέρονται στα άτομα που έχουν στην κατοχή τους το παιχνίδι.

1.6 Μηχανές Δημιουργίας Βιντεοπαιχνιδιών

Η επιλογή της μηχανής δημιουργίας παιχνιδιών είναι μια από τις πιο καθοριστικές αποφάσεις που πρέπει να πάρει ένα άτομο ή μια ομάδα ατόμων που θέλει να δημιουργήσει ένα παιχνίδι. Πολλές φορές χρειάζεται αναλυτική έρευνα των ατόμων για να ελέγξουν αν η μηχανή τους παρέχει τις υπηρεσίες και τους πόρους που θα χρειαστούν. Ένα ακόμα κριτήριο για την σωστή επιλογή της μηχανής είναι το είδος του παιχνιδιού που πρόκειται να δημιουργηθεί, αλλά και ο αριθμός των διαστάσεων που θα περιέχει αυτό. Σε περιπτώσεις που τα άτομα είναι άπειρα στον τομέα δημιουργίας παιχνιδιών, είναι σημαντικό να επιλεγεί μια σχετικά απλή μηχανή η οποία θα του δώσει ποικίλες

γνώσεις πάνω στο αντικείμενο. Παρακάτω θα μιλήσουμε για τις πιο γνωστές μηχανές δημιουργίας παιχνιδιών, αναφέροντας για κάθε μια από αυτές κάποια πλεονεκτήματα αλλά και μειονεκτήματα που είναι απαραίτητο να έχουν γνώση οι χρήστες. Συγκεκριμένα θα αναφερθούμε στις εξής μηχανές: GameMaker Studio 2, Unreal Engine, Phaser, Godot, CryEngine και Unity.[18-22]

1.6.1 GameMaker Studio 2

Το GameMaker Studio αναπτύχθηκε και διανέμεται από την εταιρία YoYoGames και θεωρείται μία πολύ καλή επιλογή από άτομα που έχουν ακόμα και μηδαμινή εμπειρία πάνω στην δημιουργία παιχνιδιών. Αυτό οφείλεται στην τεχνική drag-and-drop, η οποία χρησιμοποιείται κατά κύριο λόγο. Ωστόσο για άτομα που προτιμούν την κωδικοποίηση και θέλουν να προσθέσουν στο project τους κάποιες επιπλέον λειτουργίες, οι οποίες δεν μπορούν να προστεθούν μέσω του drag-and-drop, η μηχανή προσφέρει μία δική της γλώσσα προγραμματισμού (GameMaker). Αξίζει να σημειωθεί ότι η συγκεκριμένη μηχανή συνίσταται κυρίως για παιχνίδια δύο διαστάσεων, καθώς στην τρισδιάστατη μορφή παρέχει αρκετά περιορισμένες δυνατότητες στους χρήστες. Επίσης το πρόγραμμα παρέχει πολύ καλή λειτουργία εντοπισμού σφαλμάτων, κάτι που βοηθά ακόμα περισσότερο τους άπειρους χρήστες της, αλλά και τους έμπειρους στην εξοικονόμηση χρόνου. Ένα ακόμα θετικό στοιχείο της μηχανής είναι η συνεργασία που έχει με μεγάλες εταιρίες όπως είναι το Steam, το Amazon και το Twitch. Το GameMaker Studio 2 υποστηρίζεται από τα περισσότερα λειτουργικά συστήματα, συγκεκριμένα από τα Windows, macOs, Android, Playstation4 και το Xbox One. Ένα από τα αρνητικά της συγκεκριμένης μηχανής είναι ότι παρέχει την δυνατότητα κωδικοποίησης μόνο μέσω της δικής της γλώσσας προγραμματισμού. Ένα ακόμα μειονέκτημα είναι ότι η επίσημη υποστήριξη της μηχανής έχει αρκετά προβλήματα και ανταποκρίνεται με καθυστέρηση. Κλείνοντας, κάποια πολύ γνωστά παιχνίδια που κυκλοφορούν και οφείλουν την δημιουργία τους στο GameMaker Studio είναι το Katana Zero και το Undertable.

1.6.2 Unreal Engine

Η Unreal Engine αναπτύχθηκε από την Epic Games με πρώτη κυκλοφορία παιχνιδιού το 1998. Από το 2015 διατίθεται μία δωρεάν έκδοση της μηχανής, καθώς υπάρχει και εμπορική έκδοση στην οποία η εταιρία ζητάει το 5% των εσόδων των πωλήσεων του εκάστοτε παιχνιδιού, κάτι που μπορεί να αλλάξει αν το παιχνίδι δημοσιευθεί μέσω της Epic Games. Η μηχανή προσφέρει πάρα πολύ μεγάλες γραφικές δυνατότητες και για αυτό τον λόγο μετράει στην αγορά πολλά AAA παιχνίδια. Αυτός όμως είναι και ο λόγος που το Unreal απαιτεί ο χρήστης να χρησιμοποιεί ένα ισχυρό μηχάνημα, σε σχέση με μηχανές όπως το Unity που μπορούν να τρέχουν και σε πιο αδύναμα μηχανήματα . Όπως θα δούμε στην συνέχεια και στο Unity, η συγκεκριμένη μηχανή προσφέρει ένα Marketplace, από το οποίο μπορούν όλοι οι χρήστες να αποκτήσουν ακόμα και δωρεάν πόρους, με σκοπό να τους χρησιμοποιήσουν στα Project τους. Η συγκεκριμένη μηχανή μπορεί να δημιουργήσει παιχνίδια 2D, 3D αλλά θεωρείται και μία από τις κορυφαίες επιλογές για παιχνίδια Virtual Reality. Η ύπαρξη Blueprints βοηθάει πολύ τους νέους χρήστες. Κάποια αρνητικά που μπορεί κανείς να διακρίνει είναι ότι συνίσταται για μεγάλα έργα πολλών ατόμων και όχι για ατομικά project, καθώς και η εκμάθηση της θεωρείται αρκετά δύσκολη. Τέλος κάποιες πολύ γνωστές περιπτώσεις παιχνιδιών που εντοπίζονται στην αγορά χάρη στην Unreal Engine είναι το Tekken7 και το NBA 2K.

1.6.3 Phaser

Το Phaser δημιουργήθηκε το 2013 και είναι μια εξαιρετική επιλογή για παιχνίδια που προορίζονται είτε για κινητές συσκευές είτε για προγράμματα περιήγησης. Προτείνεται κυρίως για παιχνίδια δύο διαστάσεων, καθώς δεν είναι και η καλύτερη επιλογή στην περίπτωση των τριών διαστάσεων. Καθώς η τεχνολογία της συγκεκριμένης μηχανής βασίζεται στην ίδια τεχνολογία που χρησιμοποιείται για την ανάπτυξη ιστού, διατηρεί ένα πολύ σταθερό πλαίσιο, ακόμα και όταν ενημερώνεται με νέες εκδόσεις. Αξίζει να σημειωθεί ότι έχει περισσότερους και πιο αυστηρούς περιορισμούς από τις υπόλοιπες μηχανές, αφού σε πολλές περιπτώσεις τα παιχνίδια προορίζονται για προγράμματα περιήγησης. Εν κατακλείδι αρκετά γνωστοί τίτλοι παιχνιδιών που αναπτύχθηκαν μέσω στο Phaser είναι το Bayou Island και το Elf Runner.

1.6.4 Godot

Η Godot κυκλοφόρησε πρώτη φορά το 2014 και είναι μία επιλογή που διατίθεται δωρεάν στους χρήστες χωρίς τέλη δικαιωμάτων. Ένα σημαντικό στοιχείο της είναι ο ανοιχτός κώδικας που προσφέρει, με αποτέλεσμα να δίνει στους χρήστες της μεγάλη ευελιξία. Διακρίνεται από άλλες μηχανές λόγω της αρχιτεκτονικής που χρησιμοποιεί (κόμβος - σκηνή) η οποία δε συναντάται σε άλλη μηχανή. Επιπρόσθετα χρησιμοποιεί μία δική της γλώσσα κωδικοποίησης η οποία ονομάζεται GDScript και δημιουργήθηκε αποκλειστικά για την μηχανή Godot. Αυτό είναι και ένα από τα αρνητικά που μπορεί να διακρίνει κανείς, καθώς ενώ μοιάζει με την Python, χρειάζεται έναν επιπλέον χρόνο από όλους τους χρήστες προκειμένου να προσαρμοστούν και να μάθουν να τη χρησιμοποιούν. Μέσω της Godot οι χρήστες μπορούν να δημιουργήσουν τόσο παιχνίδια δύο διαστάσεων (2D), όσο και τριών (3D). Η συγκεκριμένη μηχανή υποστηρίζεται σε μία μεγάλη γκάμα λειτουργικών συστημάτων, αναλυτικά στα Windows, Linux, MacOS, iOS, Android, HTML5, BSD και στο WebAss Assembly. Τέλος όπως είναι απόλυτα λογικό, καθώς δεν είναι και η πιο δημοφιλής μηχανή για δημιουργία παιχνιδιών, έχει αρκετά μειωμένους πόρους σε σχέση με πιο γνωστές μηχανές όπως είναι το Unity. Αρκετά είναι τα παιχνίδια που οφείλουν την δημιουργία τους στην Godot με πιο γνωστά να είναι το BoomTown και το Gun-Toting Cats.

1.6.5 CryEngine

Η CryEngine αναπτύχθηκε το 2002 από την Crytek με την πρώτη επιτυχία του να πραγματοποιείται το 2004 μέσω του Farcry που ήταν ένα από τα δημοφιλέστερα παιχνίδια της εποχής. Μέσω του ερχομού νέων εκδόσεων, η CryEngine έφτασε σε επίπεδο να υποστηρίζει την δημιουργία ακόμα και Virtual Reality παιχνιδιών. Η ολοκληρωμένη έκδοσης της μηχανής διατίθεται σε δωρεάν μορφή από το 2016 με την εταιρία να ζητάει μερίδιο των εσόδων σε περίπτωση εμπορικότητας. Όπως αναφέραμε και σε άλλες περιπτώσεις, η μηχανή δεν έχει τους ίδιους πόρους που μπορεί να προσφέρει μία μηχανή όπως το Unity και η Unreal Engine. Αξίζει να σημειωθεί ότι είναι πολύ προσιτή μηχανή ακόμα και για άτομα με μικρή εμπειρία, καθώς διαθέτει πολλά σεμινάρια στον ιστότοπο της που μπορούν να δώσουν αρκετές γνώσεις στους χρήστες της. Επιπρόσθετα η μηχανή προσφέρει καταπληκτικά γραφικά και οπτικά στοιχεία, καθώς και ένα από τα καλύτερα εργαλεία για ήχους παιχνιδιών, το Fmod. Η συγκεκριμένη μηχανή υποστηρίζεται από πολλά λειτουργικά συστήματα, αναλυτικά από τα iOS, Windows, Android, Linux, Playstation, Xbox και Wii. Κλείνοντας η συγκεκριμένη μηχανή μας έχει

προσφέρει απολαυστικά και εντυπωσιακά παιχνίδια όπως είναι το Crysis και το Sniper Ghost Warrior 2.

1.7 Επίλογος

Στο κεφάλαιο αυτό εξηγήσαμε την έννοια του βιντεοπαιχνιδιού και παραθέσαμε μία αναλυτική ιστορική αναδρομή αυτών. Εν συνεχεία αριθμήσαμε τις κυρίαρχες κατηγορίες βιντεοπαιχνιδιών και αναφερθήκαμε στα άτομα που αποτελούν τους βασικούς ρόλους της ομάδας ανάπτυξης. Επιπρόσθετα παραθέσαμε την ενδεικτική σειρά των βημάτων που πρέπει να ακολουθήσει η ομάδα αυτή και τέλος αναφερθήκαμε στις πιο δημοφιλείς μηχανές δημιουργίας βιντεοπαιχνιδιών καταλήγοντας στην δική μας επιλογή , το Unity, το οποίο θα αναλυθεί στο προσεχές κεφάλαιο.

Κεφάλαιο 2ο: Unity3D Engine

2.1 Εισαγωγή

Το Unity3D είναι η πιο δημοφιλής μηχανής παιχνιδιών στον κόσμο αυτή τη στιγμή. Υπάρχει δωρεάν έκδοση την οποία μπορούν να προμηθευτούν όλοι οι χρήστες, καθώς σε αυτήν παρέχονται πολλές δυνατότητες. Αν κάποιος χρήστης θέλει περισσότερες λειτουργίες, υπάρχει η δυνατότητα αγοράς κάποιων πιο premium πακέτων, για τα οποία πληρώνει συνδρομή. Χρησιμοποιώντας ένας χρήστης ή προγραμματιστής το Unity3D, έχει την ευκαιρία να δημιουργήσει ότι παιχνίδι μπορεί να φανταστεί καθώς επίσης του προσφέρεται ένα πολύ μεγάλο εύρος δυνατοτήτων. Ο προγραμματισμός για τα scripts της εφαρμογής γίνεται κυρίως μέσω της C#. Το Unity3D δίνει την δυνατότητα στον χρήστη να δημιουργήσει εφαρμογές 2D, 3D, Virtual Reality (VR) αλλά και Augmented Reality (AR) [24]. Αξίζει να σημειωθεί ότι στην ιστοσελίδα της Εφαρμογής μπορεί κάποιος να βρει εκπαιδευτικά βίντεο και άρθρα, τα οποία υπάρχουν για να βοηθούν τους χρήστες τόσο στο να μάθουν καλύτερα την εφαρμογή, όσο και στο να βοηθήσουν αυτούς σε περίπτωση που δε γνωρίζουν πως χρησιμοποιείται κάποιο εργαλείο της εφαρμογής. Πολύ σημαντικό επίσης για τους χρήστες της εφαρμογής είναι το Unity Asset Store [24], το οποίο προσφέρει μια τεράστια ποικιλία από assets. Το asset είναι η αναπαράσταση ενός στοιχείου στο παιχνίδι. Έτσι ένας χρήστης μπορεί να πάρει είτε ολόκληρο terrain είτε κάποια αντικείμενα τα οποία θα προσθέσει ο ίδιος χειροκίνητα στο παιχνίδι του. Τα assets αυτά διατίθενται είτε δωρεάν σε μερικές περιπτώσεις, είτε επί πληρωμή. Όλοι οι χρήστες μπορούν να δημιουργήσουν τα δικά τους assets και να τα διαθέσουν ή να τα πουλήσουν στους υπόλοιπους χρήστες του Unity3D. Τέλος αξίζει να σημειωθεί ότι μπορεί να εγκατασταθεί σε όλα τα λειτουργικά σύστημα, καθώς και να δημιουργήσει παιχνίδια για όλα τα είδη κονσόλας.

2.2 Ιστορία Unity3D

Ο Nicolas Francis, ο Joachim Ante και ο Devid Helgason είναι τα τρία πρόσωπα πίσω από την δημιουργία του Unity. Όλα ξεκίνησαν το 2002 μετά από ένα post του πρώτου ο οποίος αναζητούσε συνεργάτη για την δημιουργία μίας Game Engine . Έτσι αρχίζοντας να αναπτύσσουν με σκοπό να βγάλουν τα προς το ζην, είδαν την ανάγκη να δημιουργήσουν κάτι ιδανικότερο έτσι ώστε να αναδείξουν την τεχνολογία [23-24]. Τελικά είχαμε την ίδρυση της εταιρίας το 2004 με του τρεις αυτούς να είναι οι επικεφαλείς και οι προγραμματιστές. Καθοριστικό παράγοντα στην αρχική επιτυχία της εταιρίας έπαιξε το γεγονός ότι βοηθούσε ανεξάρτητους Developers, οι οποίοι δεν είχαν την δυνατότητα να δημιουργήσουν δική τους μηχανή, να αναπτύξουν ένα βιντεοπαιχνίδι. Η ομάδα ενσωματώθηκε και το αρχικό όνομα της εταιρίας ήταν Over the Edge Entertainment (OTEE) και βασικός σκοπός της να δημιουργηθούν μεγάλοι τίτλοι παιχνιδιών στην αγορά [23]. Δύο χρόνια μετά η εταιρία αποφάσισε να δημιουργήσει ένα παιχνίδι για να ελέγξει τις αντοχές και τις δυνάμεις της μηχανής. Το πρώτο παιχνίδι που αναπτύχθηκε ήταν το Gooball το 2005 και μέσω αυτού η εταιρία ανακάλυψε πολλά σφάλματα στην μηχανή της. Τελικά μέσα από τα ελάχιστα κέρδη του Goobal, η OTEE προσέλαβε και άλλους προγραμματιστές και κυκλοφόρησε το Unity για πρώτη φορά το 2005. Στην αρχή η μηχανή δημιουργούσε παιχνίδια που υποστηρίζονταν μόνο σε Mac Os X, ενώ λίγο αργότερα έδωσε δικαίωμα υποστήριξης σε Microsoft Windows και σε browsers, με το τελευταίο να χτυπάει την μοναδικότητα του Adobe’s Flash στο κομμάτι αυτό [24]. Περίπου στο 2007 και εκεί που

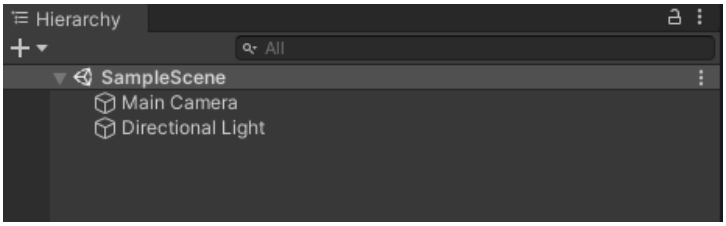
τα smartphones δέχονταν μεγάλη άνοδο καθώς είχαμε και την δημιουργία του πρώτου Apple Iphone, η εταιρία αποφάσισε να παρέχει μία έκδοση της μηχανής για την ανάπτυξη παιχνιδιών στο AppStore. Το 2009 και μετά την άνοδο των πωλήσεων των υπολογιστών, η Unity παρουσίασε την πρώτη έκδοση που προσέφερε υποστήριξη σε Windows. Το 2010 και ενώ μόλις είχε κυκλοφορήσει η τρίτη έκδοση του, υπήρχαν πάνω από 200.000 εγγεγραμμένοι προγραμματιστές στο Unity με την εταιρία να προσφέρει συνεχώς καινούργιες τεχνολογίες στην μηχανή της. Η εταιρία έτσι μέσα στα επόμενα χρόνια έδινε συνεχώς νέες εκδόσεις με όλο και περισσότερες παροχές και έτσι μέχρι και σήμερα αποτελεί την πιο δημοφιλή πλατφόρμα ανάπτυξης παιχνιδιών.[24]

2.3 Βασικά Παράθυρα Unity3D

Το Unity3D περιέχει κάποια πολύ βασικά παράθυρα, τα οποία εμφανίζονται στην οθόνη του χρήστη με το πρώτο άνοιγμα της εφαρμογής. Μέσα από αυτά τα παράθυρα ο χρήστης αναπτύσσει το παιχνίδι του. Τα πιο βασικά παράθυρα είναι το Hierarchy, το Project, το Scene, το Game, το Console, το Inspector το Animation και το Animator.

2.3.1 Hierarchy

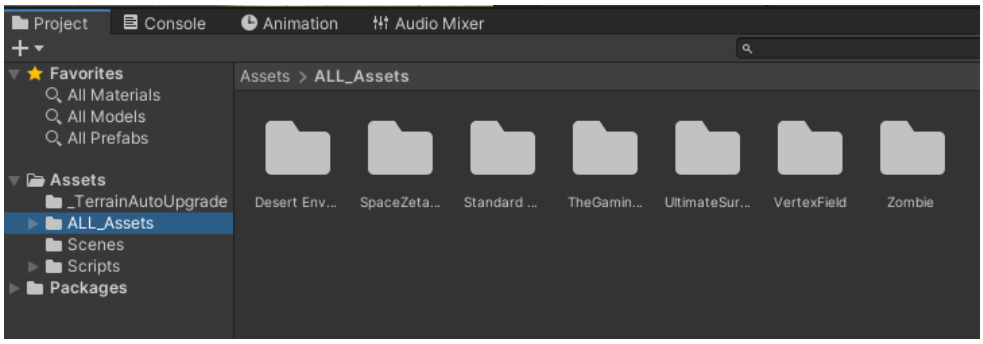
Στο συγκεκριμένο παράθυρο εμπεριέχονται όλα τα αντικείμενα που βρίσκονται στην σκηνή μας, τα οποία μπορούμε να ταξινομήσουμε και να ομαδοποιήσουμε. Όταν ένας χρήστης προσθέτει ένα νέο αντικείμενο στην σκηνή μας, τότε αυτό εμφανίζεται στο Hierarchy. Επιπρόσθετα ο χρήστης μπορεί να θέσει παιδιά και γονείς αντικειμένων. Ο πιο εύκολος τρόπος για να θέσουμε ένα αντικείμενο ως παιδί ενός άλλου αντικειμένου, είναι να το σύρουμε πάνω στο αντικείμενο που θέλουμε να είναι ο πατέρας.[25]



Εικόνα 2 - 1: Παράθυρο Hierarchy

2.3.2 Project

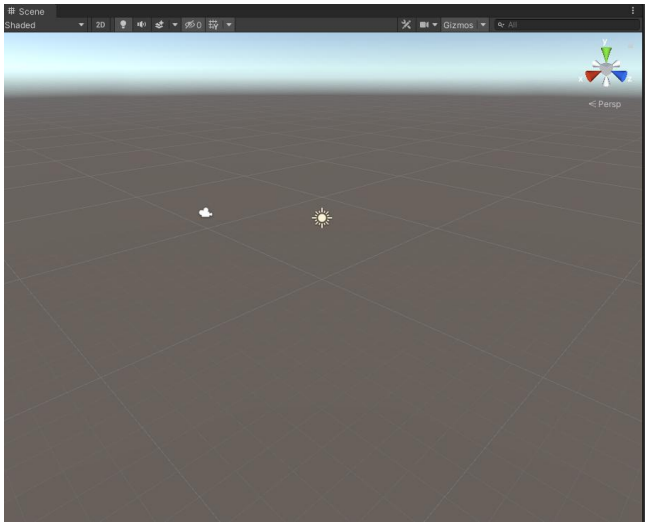
Εδώ ο χρήστης μπορεί να βρει όλα τα αντικείμενα που έχει κάνει εισαγωγή στην εφαρμογή του. Έτσι βρίσκει όποιο αντικείμενο αυτός θέλει και το σέρνει στο Hierarchy και με αυτό τον τρόπο αυτό προστίθεται στην σκηνή του. Όπως απεικονίζεται και παρακάτω, όταν επιλέγουμε έναν φάκελο στο πεδίο αυτό, ακριβώς δίπλα εμφανίζονται όλα τα στοιχεία που τον απαρτίζουν. Επίσης διακρίνουμε μία μπάρα αναζήτησης στο δεξιό πάνω μέρος, από την οποία ο χρήστης μπορεί να αναζητήσει ονομαστικά τα αντικείμενα.[26]



Εικόνα 2 - 2: Παράθυρο Project

2.3.3 Scene

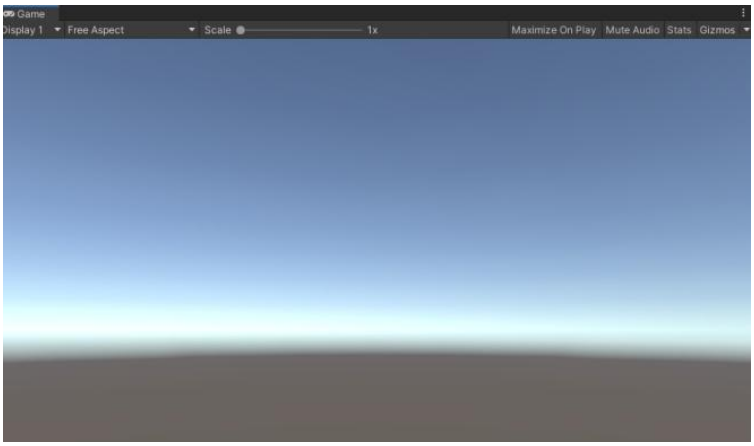
Ανοίγοντας αυτό το παράθυρο ο χρήστης μπορεί να δει την σκηνή του παιχνιδιού του, η οποία περιέχει όλα τα game objects που έχει προσθέσει. Με πολύ εύκολο και γρήγορο τρόπο μπορεί να επιλέξει ένα αντικείμενο και να αλλάξει την θέση του ως προς τρεις άξονες. Οι τρείς άξονες της εφαρμογής είναι ο X (κόκκινο χρώμα) που κατευθύνει δεξιά-αριστερά, ο Y (πράσινο χρώμα) που κατευθύνει πάνω-κάτω και ο Z (μπλε χρώμα) που κατευθύνει μπροστά-πίσω.[27]



Εικόνα 2 - 3: Παράθυρο Scene

2.3.4 Game

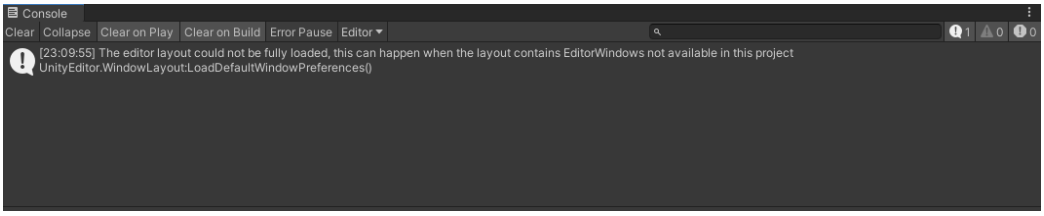
Στο παράθυρο αυτό ο χρήστης μπορεί ανά πάσα στιγμή να δει μία προεπισκόπηση του παιχνιδιού που δημιουργεί. Μπορεί επίσης να δοκιμάσει την λειτουργία κάποιων ενεργειών που έχει προσθέσει με την χρήση κώδικα. Το παράθυρο αυτό ανοίγει αυτόματα όταν πατηθεί το κουμπί Play που βρίσκεται στην μέση της εφαρμογής.[28]



Εικόνα 2 - 4: Παράθυρο Game

2.3.5 Console

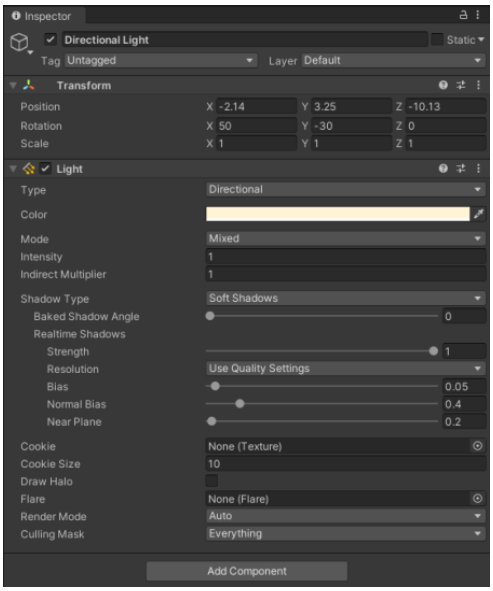
Το παράθυρο αυτό είναι πάρα πολύ σημαντικό καθώς βοηθάει τον χρήστη να βρει κάποια πιθανά σφάλματα που έχουν προκύψει είτε κατά την χρήση των εργαλείων της Unity3D, είτε στον κώδικα που έχει αναπτυχθεί. Συγκεκριμένα και στις δύο περιπτώσεις μας δίνει μία επεξήγηση του λάθους, ενώ στην περίπτωση του κώδικα, δίνει και την ακριβή σειρά που έχει πραγματοποιηθεί το λάθος. Σε περίπτωση που ο χρήστης δεν μπορεί να διορθώσει το λάθος που έχει εμφανιστεί, το πρόγραμμα δε τον αφήνει να δει το παιχνίδι από την όψη του Game. Τα μηνύματα λαθών είναι τρία και συγκεκριμένα έχουμε το Message, το Warning και το Error με το τρίτο να χρίζει άμεση επίλυση.[29]



Εικόνα 2 - 5: Παράθυρο Console

2.3.6 Inspector

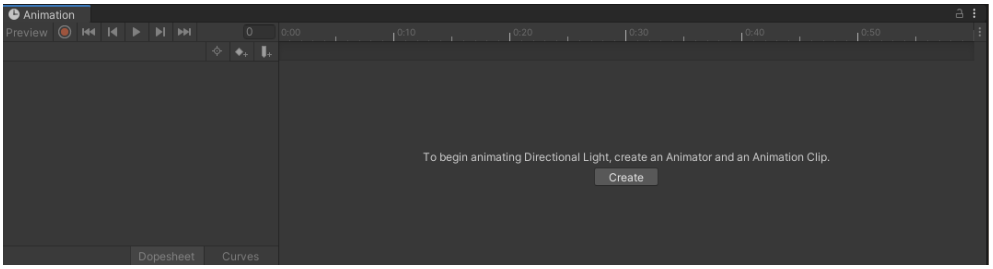
Σε αυτό το παράθυρο ο χρήστης μπορεί να δει τις συνιστώσες κάθε αντικειμένου. Επιλέγοντας έτσι ένα αντικείμενο από το πεδίο Hierarchy, του δίνεται η δυνατότητα να αλλάξει, είτε ακόμα και να προσθέσει συνιστώσες, οι οποίες επηρεάζουν την λειτουργία του αντικειμένου στο παιχνίδι μας.[30]



Εικόνα 2 - 6: Παράθυρο Inspector

2.3.7 Animation

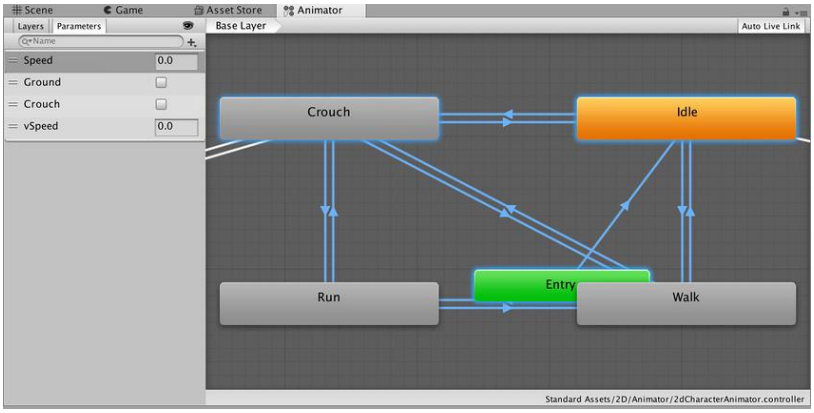
Ένα επίσης πολύ σημαντικό παράθυρο της Unity3D είναι το Animation. Εδώ ο χρήστης μπορεί να δει την κίνηση ενός αντικειμένου του. Έτσι του δίνεται η δυνατότητα είτε να αλλάξει, είτε να προσθέσει μια συνθήκη στην κίνηση του αντικειμένου σε κάθε καρέ, καθώς στο δεξιό μέρος μπορεί να δει αναλυτικά την κίνηση όπως αυτή εξελίσσεται μέσα στον χρόνο ύπαρξης της. [31]



Εικόνα 2 - 7: Παράθυρο Animation

2.3.8 Animator

Τέλος σημαντική είναι η ύπαρξη του Animator. Στο παράθυρο αυτό ο χρήστης μπορεί να δημιουργήσει την ‘επικοινωνία’ ανάμεσα στα Animations που έχει προσθέσει στο παιχνίδι του. Μέσω ενός διαγράμματος, τοποθετεί μια σειρά στον τρόπο που θα διαδέχεται τα Animations το παιχνίδι του καθώς έχει την δυνατότητα να ενεργοποιήσει αλλά και να απενεργοποιήσει κάποιες συνθήκες όταν ξεκινάει η αναπαραγωγή ενός Animation.[32]



Εικόνα 2 - 8: Παράθυρο Animator

2.4 Βασικές έννοιες στο Unity3D

Ένα άτομο που ενδιαφέρεται να ασχοληθεί με την παραγωγή παιχνιδιών μέσω Unity, πρέπει να γνωρίζει και να έχει κατανοήσει σωστά τις παρακάτω έννοιες. Τις βασικές αυτές έννοιες θα χρησιμοποιήσουμε και εμείς αργότερα όταν θα αναλύσουμε το παιχνίδι που δημιουργήσαμε. Έτσι έχουμε :

2.4.1 Game Object

Το Game Object είναι ένα από τα βασικά δομικά στοιχεία ενός παιχνιδιού και χαρακτηρίζει όλα τα αντικείμενα που υπάρχουν μέσα σε μία σκηνή. Συγκεκριμένα μπορεί να είναι ένα εφέ, ένας παίκτης, ένα κουτί, μία κάμερα ή ακόμα και ένα κουμπί στο μενού του παιχνιδιού. Η δημιουργία του μπορεί να προκύπτει εύκολα, πατώντας δεξί κλικ στο πεδίο Hierarchy και επιλέγοντας το Create Empty. Τέλος το Unity μας δίνει την επιλογή να αλλάξουμε κάποιες ρυθμίσεις μέσω του πεδίου Inspector.[33]

2.4.2 GetComponent

Το GetCompenent χρησιμοποιείται στην κωδικοποίηση του Unity μέσω C# και ουσιαστικά επιστρέφει το στοιχείο του τύπου. Όπως βλέπουμε και παρακάτω δημιουργείται μία αναφορά τύπου AnotherScript η οποία αντιστοιχίζει στο Component AnotherScript που έχει το αντικείμενο του Script. Ουσιαστικά "τραβάει" την επιλογή που έχουμε τοποθετήσει στο πεδίο AnotherScript και την αντιστοιχίζει με το anotherScript που δημιουργήσαμε.[34]

```
using UnityEngine;
using System.Collections;

public class UsingOtherComponents : MonoBehaviour
{
    private AnotherScript anotherScript;

    void Awake ()
    {
        anotherScript = GetComponent<AnotherScript>();
    }
```

Εικόνα 2 - 9: Χρήση GetComponent

2.4.3 Πεδία Public και Private

Τα δύο αυτά πεδία χρησιμοποιούνται στην κωδικοποίηση του Unity. Στην περίπτωση που δημιουργείται ένα πεδίο με την επιλογή public, αυτό το πεδίο θα εμφανιστεί αυτόματα στο Component του αντίστοιχου Script. Οπότε θα μπορεί πολύ εύκολα να μεταβληθεί η επιλογή του συγκεκριμένου πεδίου. Από την άλλη, όταν δημιουργείται ένα πεδίο με την επιλογή private δεν εμφανίζεται στο Component του Script και έτσι δεν μπορεί ο χρήστης να επιλέξει αυτό που θέλει να αντιστοιχίσει.

2.4.4 Συναρτήσεις Start – Update – Awake

Στις συγκεκριμένες συναρτήσεις μπορεί κανείς να ρυθμίσει τον χρόνο που θα εκτελούνται κάποιες ενέργειες. Έτσι έχουμε:

- Start ( ) → Οτιδήποτε βρίσκεται μέσα στην συνάρτηση αυτή θα εκτελείται μετά την εκκίνηση του παιχνιδιού και τη στιγμή που το script είναι ενεργό.
- Update( ) →Οτιδήποτε βρίσκεται μέσα στην συνάρτηση θα εκτελείται σε κάθε frame. Στις περισσότερες περιπτώσεις χρησιμοποιείται για να κάνει έλεγχο εισόδων του χρήστη.
- Awake( ) → Οτιδήποτε βρίσκεται μέσα στην συνάρτηση θα εκτελεστεί με την εκκίνηση του παιχνιδιού και μόνο μία φορά ακόμα και αν το script είναι απενεργοποιημένο. Χρησιμοποιείται συνήθως για να αρχικοποιήσει πεδία. [35]

2.4.5 Transform

Το συγκεκριμένο component υπάρχει για κάθε αντικείμενο που βρίσκεται στο περιβάλλον του παιχνιδιού. Μέσω αυτού μπορεί να αλλάξει η θέση, η περιστροφή και η κλίμακα του αντικειμένου. Οι τιμές αυτές μετριούνται σε σχέση με τον γονέα του transform. Εάν δεν έχει γονέα, αυτές μετριούνται στον παγκόσμιο χώρο. Το συγκεκριμένο πεδίο μπορεί να τροποποιηθεί είτε μέσω του component που βρίσκεται στην εφαρμογή Unity, είτε μέσω του κώδικα. Συγκεκριμένα τα παραπάνω αναφέρονται ως: transform.position, transform.rotation και transform.lossyScale. Στην περίπτωση που το αντικείμενο που θέλουμε να τροποποιήσουμε είναι παιδί ενός άλλου, τα πεδία αναφέρονται ως εξής : transform.localPosition, transform.localRotation και transform.localScale.[36]

2.4.6 Vector3

Το Vector3 είναι η αναπαράσταση τρισδιάστατων διανυσμάτων και σημείων. Στο Vector3(0,0,0) το πρώτο μηδενικό αναφέρεται στον άξονα X, το δεύτερο στον Y και το τρίτο στον Z. Μπορούμε να το χρησιμοποιήσουμε για να ορίσουμε κάποιο GameObject σε μία ακριβή θέση ή ακόμα και να μετακινήσουμε αυτό με την πάροδο του χρόνου στον άξονα που εμείς επιθυμούμε. [37]

2.4.7 Prefabs

Το Prefab είναι ένας τύπος στοιχείου που επιτρέπει την πλήρη αποθήκευση ενός GameObject με σκοπό την επαναχρησιμοποίηση του. Μέσω του prefab διατηρούνται αυτόματα όλα τα αντίγραφα συγχρονισμένα, κάτι που δε μπορούμε να πετύχουμε με το copy paste. Για να δημιουργηθεί ένα Prefab αρκεί να μεταφέρουμε το GameObject από το πεδίο Hierarchy στο πεδίο Project. Σε περίπτωση

που κάνουμε τροποποιήσεις σε ένα Prefab, αυτές αντικατοπτρίζονται σε όλα τα συγχρονισμένα GameObject και με τον τρόπο αυτό γλιτώνουμε τόσο χρόνο, όσο και πιθανά σφάλματα που θα προέκυπταν χωρίς την χρήση αυτού. Για να εφαρμόσουμε τις αλλαγές του αρχικού Prefab σε όλα τα Game Object, μεταβαίνουμε στο πεδίο Prefab στην καρτέλα Inspector και στο Overrides επιλέγουμε το Apply All. [38]

2.4.8 Collider

Το Collider χρησιμοποιείται για να προσομοιώσει τη σύγκρουση δύο φυσικών αντικειμένων στο περιβάλλον. Ουσιαστικά το collider λειτουργεί σαν «αόρατος τοίχος», εμποδίζοντας ένα αντικείμενο με collider να προσπελάσει ένα άλλο που επίσης διαθέτει αυτή την ιδιότητα. Τα colliders διατίθενται σε διάφορα σχήματα και τύπους. Συγκεκριμένα τα σχήματα που μπορεί να έχουν είναι η σφαίρα, η κάψουλα και το κουτί. Τώρα για πιο σύνθετα σχήματα μπορούμε είτε να συνδυάσουμε πολλά από αυτά, είτε να χρησιμοποιήσουμε το Mesh Collider το οποίο θα ταιριάζει ακριβώς με το σχήμα που θέλουμε να καλύψουμε. Κλείνοντας το collider έχει τρία event που συμβαίνουν σε κάθε φάση της σύγκρουσης. Έχουμε το OnCollisionEnter( ) που εκτελείται την στιγμή που συμβαίνει η σύγκρουση, το OnCollisionStay( ) που πραγματοποιείται όταν το ένα αντικείμενο είναι για κάποιο χρονικό διάστημα σε σύγκρουση με ένα άλλο και το OnCollisionExit( ) που εκτελείται από την στιγμή που τελειώνει η σύγκρουση.[39]

2.4.9 Rigidbody

Ένα Rigidbody είναι μια κλάση η οποία δίνει την ιδιότητα στο αντικείμενο να αλληλεπιδρά με τον φυσικό κόσμο. Συγκεκριμένα θα επηρεάζεται από την βαρύτητα και θα δέχεται δυνάμεις από το περιβάλλον του. Ουσιαστικά το χρησιμοποιούμε όταν θέλουμε ένα GameObject να έχει μάζα με ότι αυτό συνεπάγεται. Στην περίπτωση των παιχνιδιών δύο διαστάσεων, πρέπει να χρησιμοποιείται το Rigidbody 2D καθώς αυτό αλληλεπιδρά μόνο στον άξονα X και Y. Τέλος αξίζει να σημειωθεί ότι ο χρήστης μπορεί να ρυθμίζει την κλίμακα της βαρύτητας του κάθε αντικειμένου μέσω του πεδίου Gravity Scale στο component Rigidbody. [40]

2.4.10 Animator Controller

Το Animator Controller επιτρέπει στους χρήστες να οργανώσουν και να διατηρήσουν ένα σύνολο Animation Clip και συναφών μεταβάσεων για τον χαρακτήρα ή αντικείμενο που έχουν στην σκηνή τους. Στις περισσότερες περιπτώσεις είναι φυσιολογικό να υπάρχουν πολλά Animation Clip και να εναλλασσόμαστε μεταξύ τους όταν συμβαίνουν ορισμένες συνθήκες παιχνιδιού. Για παράδειγμα θα μπορούσε ο χρήστης να μεταβεί από μία κατάσταση σε μία άλλη με την ενεργοποίηση μίας συνθήκης ή με την χρήση ενός πλήκτρου. [41]

2.4.11 Coroutine

Το Coroutine είναι μία συνάρτηση που έχει την ιδιότητα να σταματάει την εκτέλεση της και να τη συνεχίζει στο επόμενο frame. Είναι πολύ χρήσιμη σε περίπτωση που θέλει κάποιος να μεταβάλει μία

τιμή σταδιακά ή να εκτελέσει κάποια εντολή μετά το πέρασμα ορισμένων δευτερολέπτων. Υπάρχουν ορισμένοι τύποι δήλωσης που θέτουν σε παύση ένα Coroutine. Η επιλογή για το ποιον θα χρησιμοποιήσουμε επηρεάζεται από το χρονικό διάστημα που θέλουμε να είναι σε παύση. Έτσι θα έχουμε :

- Τον yield return null στον οποίο θα περιμένει μέχρι το επόμενο frame.

```
IEnumerator MyCoroutine() {
    int i=0;

    while (i<10){
        i++;
        yield return null;
    }
}
```

Εικόνα 2 - 10: Yield return null σε Coroutine

- Τον WaitForSeconds(X), στον οποίο θα περιμένει X δευτερόλεπτα.

```
IEnumerator MyCoroutine() {
    print("Wait 5 sec");
    yield return new WaitForSeconds(5);
    print("5 seconds has passed");
}
```

Εικόνα 2 - 11: WaitForSeconds(X) σε Coroutine

- Τον WaitForSecondsRealTime(X), στον οποίο εκτελεί την ίδια λειτουργία με το WaitForSeconds(X) με την διαφορά ότι συνεχίζει να λειτουργεί ακόμα και όταν κάνουμε παύση του παιχνιδιού.

- Τον WaitUntil(X) όπου περιμένει η συνθήκη X να είναι false πριν συνεχίσει.

- Και τέλος την περίπτωση που περιμένουμε μία άλλη Coroutine να εκτελεστεί.[42]

```
IEnumerator MyCoroutine()
    {
        print("Coroutine has started");
        yield return StartCoroutine(MyOtherCoroutine());
        print("Coroutine has ended");
    }
```

Εικόνα 2 - 12: Wait For Another Coroutine

2.4.12 Layers

Τα επίπεδα στο Unity καθορίζουν ποια GameObjects μπορούν να αλληλεπιδράσουν μεταξύ τους. Χρησιμοποιούνται κατά βάση για την διαφορετική προβολή αντικειμένων από τις κάμερες της σκηνής αλλά και για την ανίχνευση σύγκρουσης ανάμεσα στα διάφορα αντικείμενα. Το Unity μας δίνει την δυνατότητα δημιουργίας και ανάθεσης νέων Layers, που μπορούμε να χρησιμοποιήσουμε στις περιπτώσεις που δεν αρκούν τα έτοιμα Layers αυτού.[43]

2.4.13 Tags

"Η ετικέτα είναι μια λέξη αναφοράς που μπορείτε να αντιστοιχίσετε σε ένα ή περισσότερα GameObjects"[44]. Όπως και στην περίπτωση των Layers , πέραν των προκαθορισμένων Tags του Unity οι χρήστες μπορούν να δημιουργήσουν και να αναθέσουν τα δικά τους Tags. Χρησιμοποιούνται περισσότερο στην κωδικοποίηση του παιχνιδιού , αφού προσφέρουν έναν εύκολο τρόπο εντοπισμού ενός συγκεκριμένου GameObject με την ενσωματωμένη μέθοδο GameObject.FindWithTag( ).

2.4.14 Particle System

"Ένα σύστημα σωματιδίων προσομοιώνει και αποδίδει πολλές μικρές εικόνες ή πλέγματα, που ονομάζονται σωματίδια, για να παράγει ένα οπτικό εφέ"[45].Το σύνολο των σωματιδίων αυτών όταν ενωθούν προκαλούν την τελική όψη και εμφάνιση του εφέ. Τα συστήματα σωματιδίων χρησιμοποιούνται επί τω πλείστο για την οπτική αναπαράσταση δυναμικών αντικειμένων όπως για παράδειγμα το νερό, η φωτιά, ο καπνός κλπ.

2.5 Φωτισμός στο Unity3D

Ένα πολύ σημαντικό και παράλληλα ιδιαίτερο κομμάτι στο Unity είναι ο φωτισμός. Παίζει πρωταγωνιστικό ρόλο στην εμφάνιση και στην ζωντάνια του περιβάλλοντος του παιχνιδιού. Έτσι καταλαβαίνουμε ότι πρέπει να δίνεται έμφαση στην σωστή διαχείριση του, ειδικά σε παιχνίδια στα οποία μας ενδιαφέρει να δώσουμε μία πιο ρεαλιστική αναπαράσταση του περιβάλλοντος στον οποίο θα αλληλεπιδρά ο παίκτης. Υπάρχουν διάφορα είδη φωτισμού ανάλογα με την έκταση που θέλουμε αυτός να καλύπτει. Πολύ εύκολα μπορούμε να διακρίνουμε τα 4 είδη φωτισμού που μας προσφέρονται από το Unity, τα οποία είναι τα εξής: Point Light, Directional Light, Spot Light και Area Light.[46]

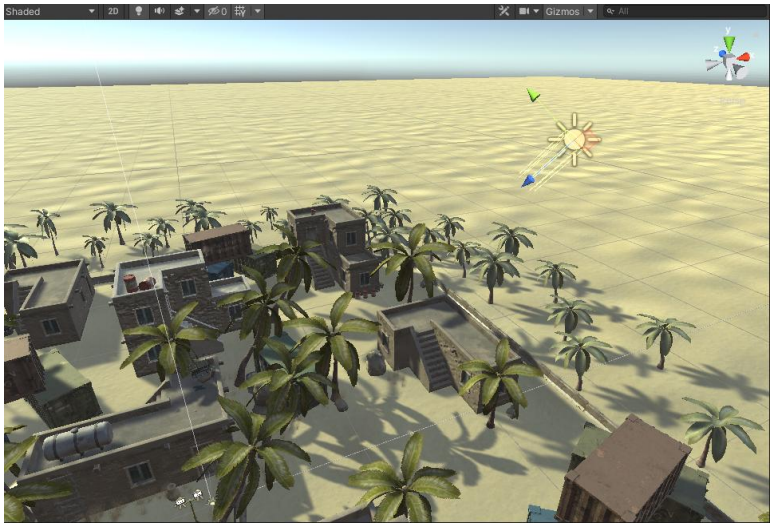
2.5.1 Point Light

Το Point Light είναι μία σφαίρα φωτός που ακτινοβολεί με την ίδια ένταση προς όλες τις κατευθύνσεις. Το Unity μας δίνει την δυνατότητα να τροποποιήσουμε το χρώμα που προβάλλεται μέσω της επιλογής Color, την ένταση αυτού μέσω της επιλογής Intensity, καθώς και το πόσο επεκτείνεται στον χώρο μέσω της επιλογής Range. Προφανώς υπάρχουν και άλλες ρυθμίσεις που μπορεί να τροποποιήσει κανείς, για να προσαρμόσει το φως καλύτερα στο περιβάλλον του. Το Point Light στις περιπτώσεις περιπτώσεις χρησιμοποιείται για τη δημιουργία εφέ όπως είναι μία φωτιά, μία έκρηξη ή ακόμα και μια λάμπα εσωτερικού-εξωτερικού χώρου. Για την δημιουργία ενός Point Light, αρκεί να ακολουθήσουμε την εξής διαδικασία GameObject → Light → Point Light.

Εικόνα 2 - 13: Point Light

2.5.2 Directional Light

Το Directional Light είναι μία δέσμη φωτός που υπάρχει σε κάθε νέα σκηνή στο Unity από προεπιλογή. Στις περισσότερες περιπτώσεις αναπαριστά το φως του ηλίου, για αυτό και τοποθετείται αρκετά πιο μακριά από τον κόσμο που αλληλεπιδρά ο χρήστης. Είναι ένα μέσο δημιουργίας σκιών σε ακίνητα αντικείμενα, που μπορεί να καλύψει ακόμα και όλο το περιβάλλον που κινείται ο παίκτης. Και σε αυτήν την περίπτωση φωτός, μπορεί κανείς πολύ εύκολα να αλλάξει την κατεύθυνση που προέρχεται το φως αλλά και να προσαρμόσει τις σκιές που δημιουργούνται στα αντικείμενα όταν το φώς πέφτει επάνω. Για την δημιουργία ενός Directional Light αρκεί να ακολουθήσουμε την εξής διαδικασία GameObject → Light → Directional Light.

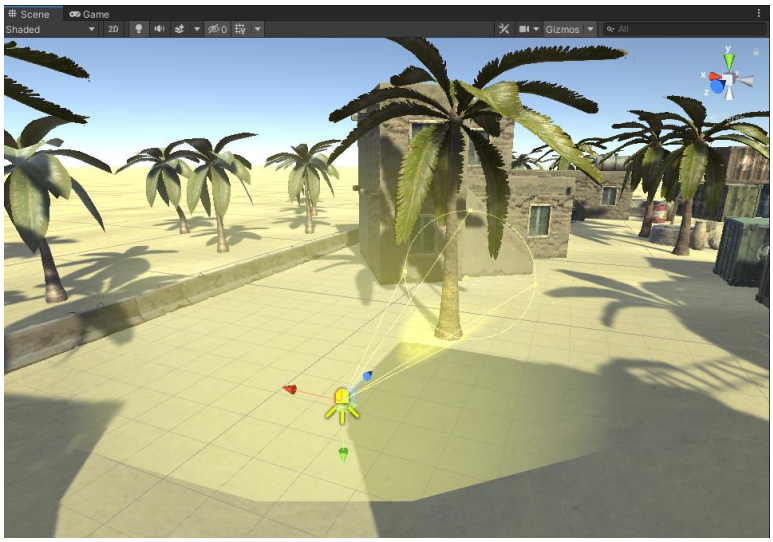


Εικόνα 2 - 14: Directional Light

2.5.3 Spot Light

Το Spot Light αποτελεί μια δέσμη φωτός στην οποία οι ακτίνες φωτός διαχέονται σε κωνικό σχήμα. Πολύ εύκολα μπορεί κανείς να το παρομοιάσει με προβολέα, για αυτό και χρησιμοποιούνται για φακούς, για φωτεινούς σηματοδότες και γενικά για περιπτώσεις στις οποίες το φώς θέλουμε να προβάλλεται όπως απεικονίζει στην φωτογραφία παρακάτω. Μέσω των ρυθμίσεων του Spot Light

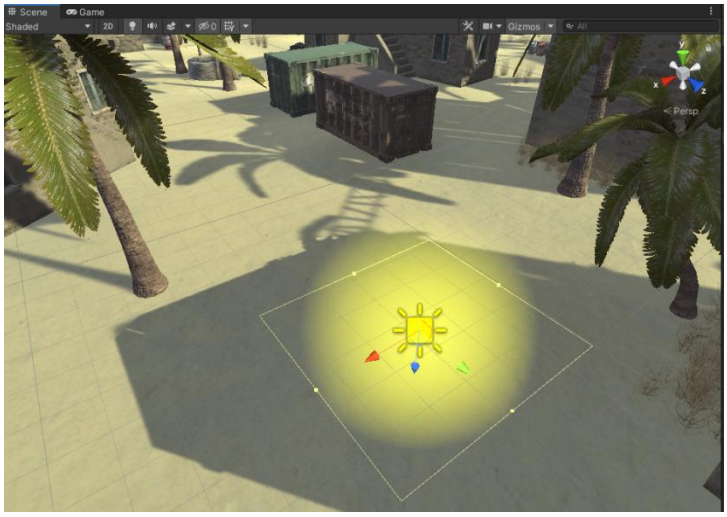
μπορεί κανείς να προσαρμόσει το μέγεθος του κώνου που φέγγει το φως, καθώς και την δύναμη που αυτό ακτινοβολεί. Για την δημιουργία ενός Spot Light αρκεί να ακολουθήσουμε την εξής διαδικασία GameObject → Light → Spot Light.



Εικόνα 2 - 15: Spot Light

2.5.4 Area Light

Το Area Light είναι μια δέσμη φωτός στην οποία οι ακτίνες φωτός διαχέονται σε ορθογώνιο σχήμα, "με την ίδια ένταση προς όλες τις κατευθύνσεις αλλά μόνο από την μία πλευρά αυτού."[46] Σε πολλές περιπτώσεις χρησιμοποιείται είτε για φωτεινούς σηματοδότες, είτε για τον φωτισμό δρόμου έτσι ώστε να κατευθύνει τον παίκτη προς μία κατεύθυνση. Αξίζει να σημειωθεί ότι προσφέρει μικρότερη σκίαση από τα υπόλοιπα Lights του Unity.Όπως και στα υπόλοιπα είδη φωτισμού, μπορεί κανείς να τροποποιήσει το χρώμα που προβάλλεται μέσω της επιλογής Color, το μέγεθος των πλευρών του ορθογωνίου μέσω των επιλογών Width και Height, το εύρος του φωτός στον χώρο αλλά και αρκετές άλλες ρυθμίσεις που προσφέρονται από το Unity.



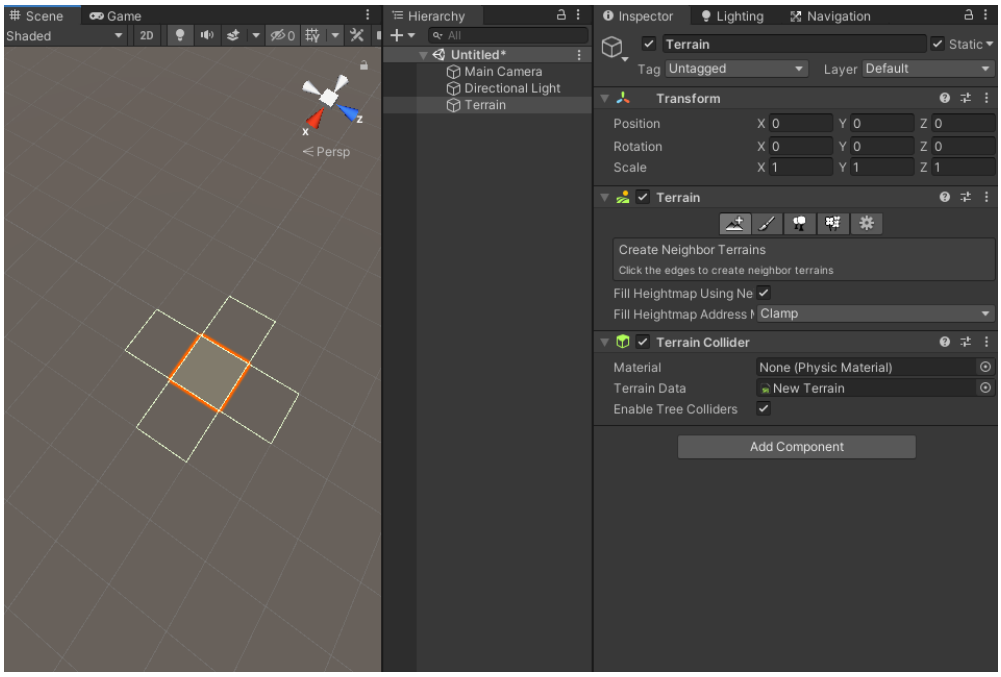
Εικόνα 2 - 16: Area Light

2.6 Terrain στο Unity3D

Το Terrain είναι το περιβάλλον στο οποίο διαδραματίζεται το παιχνίδι. Έτσι παίζει πολύ σημαντικό ρόλο στην τελική επιτυχία του έργου, καθώς είναι αυτό που συνήθως εντυπωσιάζει τον παίκτη, ενώ είναι και η πρώτη εικόνα που δέχεται κάποιος στην πρώτη του επαφή με το παιχνίδι. Η ομάδα ανάπτυξης ενός παιχνιδιού έχει την δυνατότητα να δημιουργήσει από το μηδέν ένα δικό της terrain έτσι όπως το θέλει και το φαντάζεται. Από την άλλη υπάρχουν πολλά έτοιμα terrain στο Unity Asset Store ή ακόμα και έτοιμα αντικείμενα τα οποία μπορεί είτε να αγοράσει είτε να προμηθευτεί κάποιος δωρεάν, στην περίπτωση που αυτό υποστηρίζεται και να τα προσθέσει στο έργο του όπως αυτός επιθυμεί. Στην ενότητα αυτή θα δείξουμε όσο πιο απλά γίνεται την δημιουργία ενός Terrain, καθώς είναι και το πιο ενδιαφέρον σχεδιαστικό κομμάτι για πολλούς. [47]

Αρχικά θα χρειαστεί να δημιουργηθεί ένα νέο Terrain με τα εξής βήματα : GameObject → 3D Object → Terrain. Έτσι πλέον στην σκηνή μας εμφανίζεται ένα Terrain, πάνω στο οποίο θα δημιουργήσουμε όλα τα υπόλοιπα.

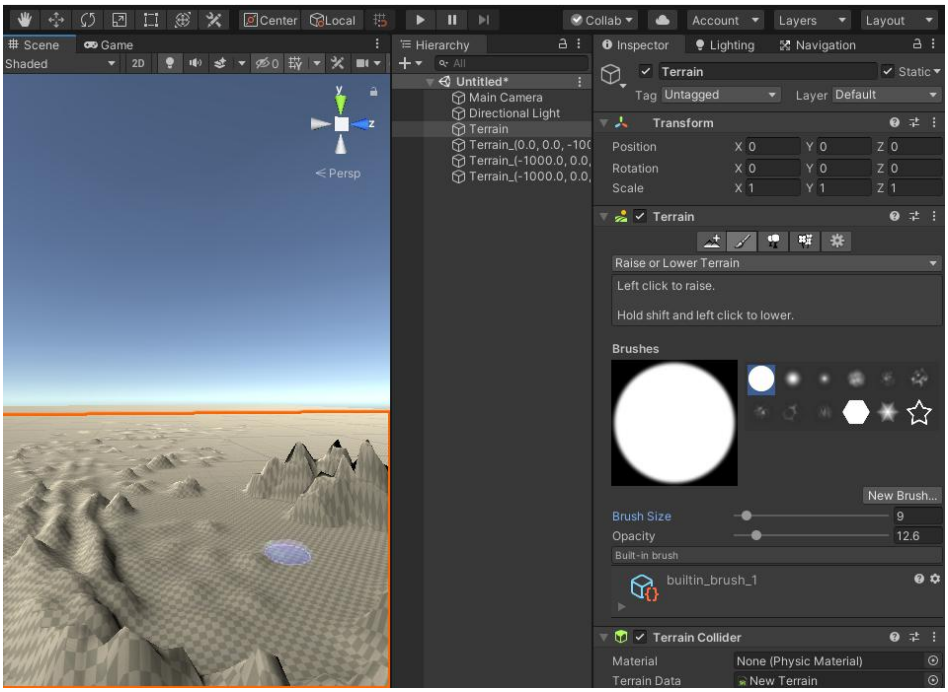
Έχοντας επιλεγμένο το Terrain, στην καρτέλα Inspector παρατηρούμε 5 εικονίδια τα οποία συμβάλλουν στην διαμόρφωση αυτού. Έτσι όταν δημιουργούμε ένα Terrain είναι από προεπιλογή 1000x1000. Στην περίπτωση που θέλει κανείς να αυξήσει το μέγεθος της επιφάνειας αρκεί να πατήσει το πρώτο κουμπί που ονομάζεται Create Neighbor Terrain, όπως φαίνεται στην εικόνα. Έπειτα εμφανίζεται στην σκηνή ένα πλέγμα τετραγώνων, όπου ο χρήστης επιλέγοντας περισσότερα τετράγωνα, προσαρμόζει το μέγεθος που θέλει να έχει τελικά το Terrain αυτό. Όταν επιλέγουμε ένα τετράγωνο, προστίθεται ένα επιπλέον GameObject στο πεδίο Hierarchy, καθώς στην συνέχεια μπορούμε να επεξεργαστούμε ακόμα και διαφορετικά το κάθε ένα.



Εικόνα 2 - 17: Create Neighbor Terrains

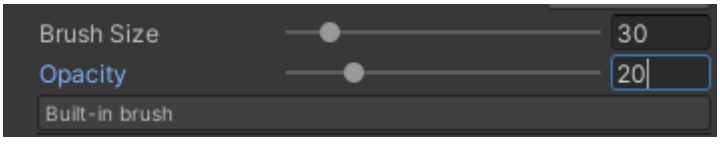
Εφόσον έχουμε ολοκληρώσει την διαμόρφωση του μεγέθους της επιφάνειας, περνάμε στο κομμάτι της γλυπτικής του Terrain. Έτσι επιλέγουμε το δεύτερο κουμπί στο πεδίο Inspector, που ονομάζεται Paint Terrain. Αφού το έχουμε επιλέξει θα παρατηρήσουμε ένα drop down μενού στο οποίο έχουμε

έξι επιλογές. Η πρώτη επιλογή ονομάζεται Raise or Lower Terrain και αφορά την ανύψωση ή την βύθιση του εδάφους. Υπάρχουν αρκετά Brushes με διαφορετικά σχήματα που μπορεί να χρησιμοποιήσει κανείς για την γλυπτική του εδάφους της σκηνής του . Κάνοντας αριστερό κλικ στην σκηνή παρατηρούμε το έδαφος να παίρνει ύψος και αυτό να αυξάνεται όσο εμείς σέρνουμε το ποντίκι προς τα πάνω. Αν τώρα θέλουμε να μειώσουμε κάπου το ύψος αρκεί να έχουμε πατημένο το πλήκτρο Shift όταν πατάμε αριστερό κλικ. Με τον τρόπο αυτό το ύψος μπορεί να αφαιρεθεί τελείως, επαναφέροντας το έδαφος στο αρχικό σημείο.



Εικόνα 2 - 18: Paint Terrain- Raise or Lower Terrain

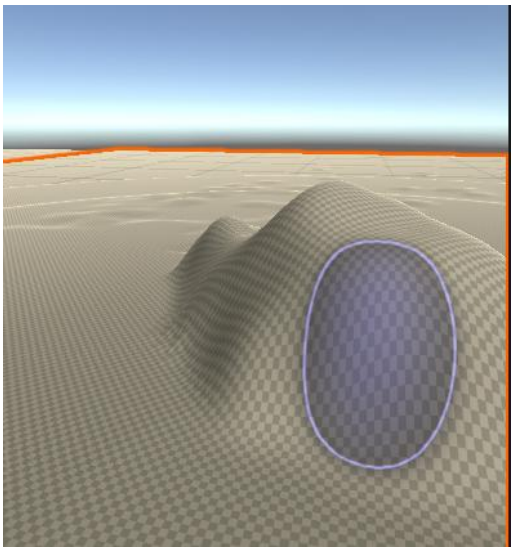
Πριν όμως ξεκινήσει κανείς να διαμορφώνει το έδαφος του Terrain καλό θα ήταν να ασχοληθεί με τις ρυθμίσεις του πινέλου για να μπορεί να το διαμορφώνει με το τρόπο που επιθυμεί. Έτσι τα πεδία που πρέπει να αλλάξει είναι το Brush Size το οποίο διαμορφώνει το μέγεθος του πινέλου μας και το Opacity το οποίο καθορίζει πόσο γρήγορα θα υψώνεται το έδαφος. Τέλος να σημειωθεί ότι το μέγεθος του πινέλου (Brush Size) μπορεί να αλλάξει και από το πληκτρολόγιο μας με το πλήκτρο "]" για αύξηση και το πλήκτρο "[" για μείωση αυτού.



Εικόνα 2 - 19: Ρυθμίσεις Brush

Μία ακόμα επιλογή στο drop down μενού του Paint Terrain είναι το Set Height. Μέσω αυτού μπορεί κανείς αν θέσει το ανώτερο ύψος που μπορεί να έχει το Terrain του παιχνιδιού του. Έτσι στο πεδίο Height επιλέγουμε το ύψος που επιθυμούμε και παρατηρούμε ότι αν προσπαθήσουμε να δημιουργήσουμε έναν λόφο με ύψος μεγαλύτερο από το όριο που έχουμε θέσει, το Unity δε μας το επιτρέπει.

Η επόμενη επιλογή του drop down μενού είναι το Smooth Height με το οποίο μπορεί κανείς να "εξομαλύνει τα άκρα του εδάφους που είναι τραχιά."[47] Εφαρμόζοντας το παρατηρούμε ότι το έδαφος παίρνει μία πιο ομαλή και κυκλική μορφή.



Εικόνα 2 - 20: Smooth Height

Η τελευταία επιλογή από το drop down μενού είναι το Stamp Terrain μέσω του οποίου μπορούμε να δημιουργήσουμε ύψη, προσαρμόζοντας από το πεδίο με το αντίστοιχο όνομα, το ύψος αυτών.

Στο σημείο αυτό και αφού έχει ολοκληρωθεί η γλυπτική του εδάφους, περνάμε στην ζωγραφική αυτού. Συγκεκριμένα θα χρωματίσουμε όλο το έδαφος για να πάρει μία πιο όμορφη και ζωντανή μορφή. Για τον σκοπό αυτό στο Asset Store του Unity υπάρχει μία τεράστια γκάμα από assets που μπορούν να χρησιμοποιήσουν οι χρήστες για να δώσουν το χρώμα που επιθυμούν στις υφές που ανέπτυξαν. Αφού έχουμε εισάγει στο Unity τα Assets που θέλουμε να χρησιμοποιήσουμε, επιλέγουμε το Paint Texture και στο Edit προσθέτουμε τις επιλογές που θέλουμε να έχουμε για την διαμόρφωση του εδάφους.

Εφόσον έχουμε αποφασίσει για τον χρωματισμό του εδάφους που θέλουμε να εφαρμόσουμε, επιλέγουμε αρχικά τα τεταρτημόρια του εδάφους που αυτό θα εφαρμοστεί και προσθέτουμε ένα νέο επίπεδο μέσω της επιλογής στο αντίστοιχο Inspector. Από εδώ και πέρα όλος ο χρωματισμός τους εδάφους θα υλοποιηθεί με την προαναφερθείσα διαδικασία.

Συνεχίζοντας, περνάμε στην δημιουργία των δέντρων μέσω των οποίων θα γεμίσει περισσότερο η σκηνή μας. Για τον σκοπό αυτό θα χρησιμοποιηθεί το δωρεάν πακέτο που προσφέρει το Unity και συγκεκριμένα το Free SpeedTrees. Στο πακέτο αυτό μπορεί κανείς να βρει διαφόρων ειδών δέντρα για όλες τις περιπτώσεις που θέλει να καλύψει. Έτσι επιλέγουμε το πεδίο Paint Trees από το πεδίο Terrain στο αντίστοιχο Inspector και στην συνέχεια αιτούμε την προσθήκη ενός νέου δέντρου μέσω της επιλογής Add Tree στο πεδίο Edit Trees. Τέλος στο αναδυόμενο παράθυρο προσθέτουμε το δέντρο που επιθυμούμε μέσω του πεδίου Tree Prefab.

Αφού έχουμε ολοκληρώσει τα παραπάνω, μπορούμε πολύ εύκολα να επεξεργαστούμε την εμφάνιση των δέντρων που έχουμε προσθέσει στην σκηνή μας, μέσω των ρυθμίσεων που μας προσφέρονται από το Unity.

Εν συνεχεία το Unity μας δίνει την δυνατότητα λεπτομερειακής βαφής κάπως σημείων της σκηνής μας. Για παράδειγμα μας επιτρέπει να ζωγραφίσουμε με λεπτομέρεια το γρασίδι. Αυτό επιτυγχάνεται

μέσω του εργαλείου Paint Details, που είναι το τέταρτο αντίστοιχα εργαλείο στο πεδίο Terrain. Το μόνο που θα χρειαστεί να κάνουμε είναι να επιλέξουμε το Add Grass Texture, από το αναδυόμενο παράθυρο Edit Details και να προσθέσουμε το γρασίδι που θα εφαρμοστεί. Από εδώ και πέρα η διαδικασία είναι πλέον γνωστή, αφού ουσιαστικά κάνουμε ότι κάναμε και στις υπόλοιπες περιπτώσεις ζωγραφικής του εδάφους.

Κλείνοντας το Unity μας παρέχει την επιλογή Terrain Settings, η οποία επιτρέπει στον χρήστη να τροποποιήσει πολλές πρόσθετες ρυθμίσεις που αφορούν το έδαφος της σκηνής του.

2.7 Επίλογος

Στο κεφάλαιο αυτό αναφερθήκαμε στην Unity3D Engine, την δημοφιλέστερη μηχανή δημιουργίας βιντεοπαιχνιδιού, που θα χρησιμοποιήσουμε εμείς. Αρχικά εξηγήσαμε τα δομικά χαρακτηριστικά του και την πορεία που ακολούθησαν οι δημιουργοί του μέχρι και σήμερα. Εν κατακλείδι, αναφερθήκαμε στα βασικά παράθυρα και έννοιες που θα πρέπει να γνωρίζει ο αναγνώστης για να κατανοήσει πλήρως την διαδικασία ανάπτυξης που θα ακολουθήσουμε στο επόμενο κεφάλαιο.

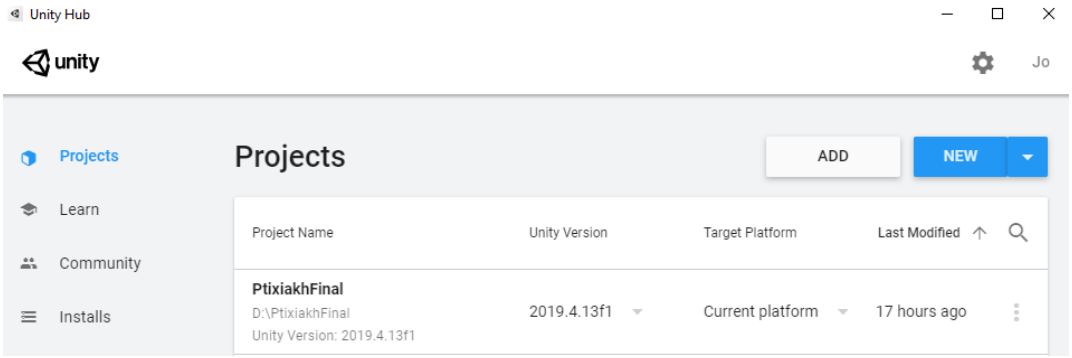
Κεφάλαιο 3ο: Ανάπτυξη Παιχνιδιού με Unity

3.1 Εισαγωγή

Στο σημείο αυτό θα ξεκινήσουμε να κάνουμε την ανάλυση του παιχνιδιού του οποίου υλοποιήσαμε. Θα παραθέσουμε όλη την διαδικασία που εμείς ακολουθήσαμε, έτσι ώστε αυτό να προσφέρει όσες περισσότερες γνώσεις είναι δυνατόν στους αναγνώστες. Το παρακάτω παιχνίδι είναι ένα First Person Shooter παιχνίδι, όπου ο χρήστης διαχειρίζεται έναν χαρακτήρα και με τα όπλα που είναι διαθέσιμα, προσπαθεί να εξοντώσει τους αντιπάλους που στην προκειμένη περίπτωση είναι τα Ζόμπι. Χαρακτηριστικό αυτού του είδους παιχνιδιού είναι ότι η κάμερα που προβάλει τον παίκτη, αναπαριστά τα μάτια αυτού, κάνοντας έτσι τον χρήστη να νιώθει πιο ρεαλιστικά το παιχνίδι.

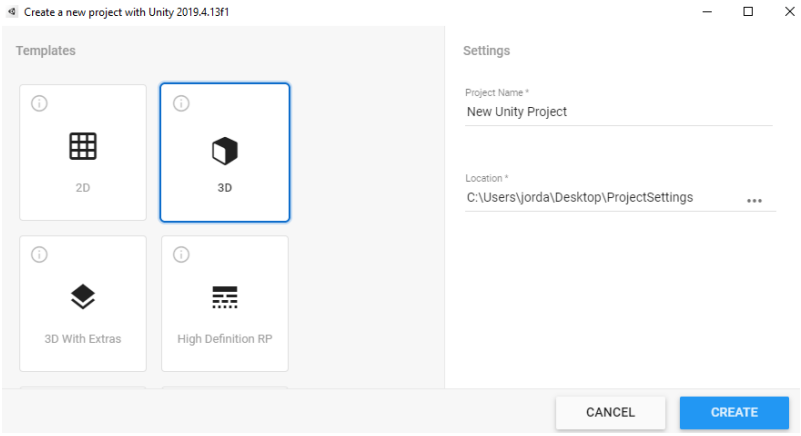
3.2 Ξεκινώντας με το Unity

Αρχικά ανοίγοντας το Unity πατάμε την επιλογή "NEW" η οποία μας δίνει την δυνατότητα δημιουργίας ενός νέου Project. Στην περίπτωση που έχουμε ήδη δημιουργήσει ένα Project, απλά επιλέγουμε αυτό από την λίστα, έτσι ώστε να συνεχίσουμε την επεξεργασία του από εκεί που έχει γίνει η τελευταία αποθήκευση.



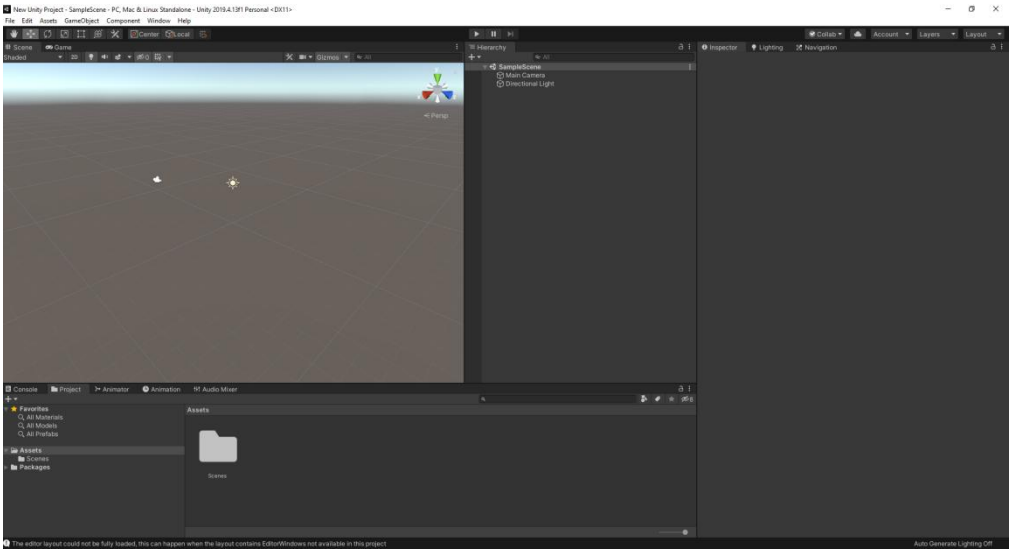
Εικόνα 3 - 1: Άνοιγμα Unity

Στην συνέχεια το Unity μας δίνει την δυνατότητα να επιλέξουμε το Template του Project μας. Τα δύο πιο γνωστά είναι το 2D, το οποίο αναφέρεται σε παιχνίδια δύο διαστάσεων και το 3D, που είναι για παιχνίδια τριών διαστάσεων και είναι και αυτό που εμείς θα χρησιμοποιήσουμε. Εκτός όμως από το Template μας δίνεται η δυνατότητα να δηλώσουμε το όνομα του Project μας, αλλά και τον φάκελο στον οποίο αυτό θα αποθηκεύεται. Αφού ρυθμίσουμε όλα τα παραπάνω, επιλέγοντας το "CREATE" θα έχουμε ολοκληρώσει την έναρξη της δημιουργίας του παιχνιδιού.[48]



Εικόνα 3 - 2: Έναρξη/Δημιουργία παιχνιδιού

Μετά τις παραπάνω ρυθμίσεις, έχουμε την πρώτη εικόνα του Unity. Πρόκειται για το περιβάλλον στο οποίο θα δημιουργήσουμε το παιχνίδι μας. Στην Ενότητα 2.3 έχει γίνει πλήρης αναφορά για όλα τα πεδία από τα οποία αποτελείται το Unity.



Εικόνα 3 - 3: Περιβάλλον Unity

3.3 Menus

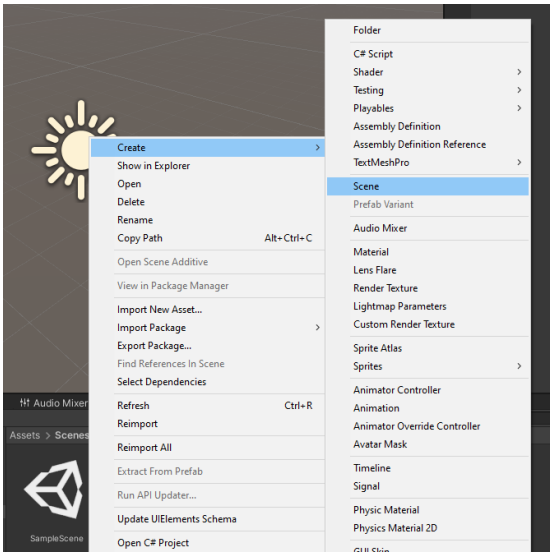
Τα Menus σε ένα παιχνίδι καθορίζουν σε μεγάλο βαθμό την αλληλεπίδραση του χρήστη με το παιχνίδι. Η εμφάνιση τους οφείλει να είναι απλή, κατανοητή και φιλική προς τους χρήστες καθώς αυτά αποτελούν ένα κομμάτι του Project που θα επαναπροβληθεί αρκετές φορές. Επιπλέον, ο κύριος σκοπός τους είναι να επιτρέπουν στον χρήστη να ρυθμίσει κατάλληλα και με βάση τις δικές του προτιμήσεις διάφορες παραμέτρους του παιχνιδιού αλλά και παρέχει την δυνατότητα εναλλαγής μεταξύ των προκαθορισμένων σκηνών αυτού.

3.3.1 Main Menu

Το Main Menu είναι ένα πολύ σημαντικό κομμάτι του Project, καθώς είναι η πρώτη εικόνα που βλέπει ο παίκτης και μέσω αυτού αρχίζει να βιώνει την ατμόσφαιρα του παιχνιδιού. Σε πολλές περιπτώσεις το μενού περιέχει ένα βίντεο ή μια κινούμενη εικόνα, ενώ άλλες φορές χρησιμοποιείται απλά μια εικόνα, όπως θα δούμε και στο δικό μας Project. Στο αρχικό μενού δίνεται η δυνατότητα

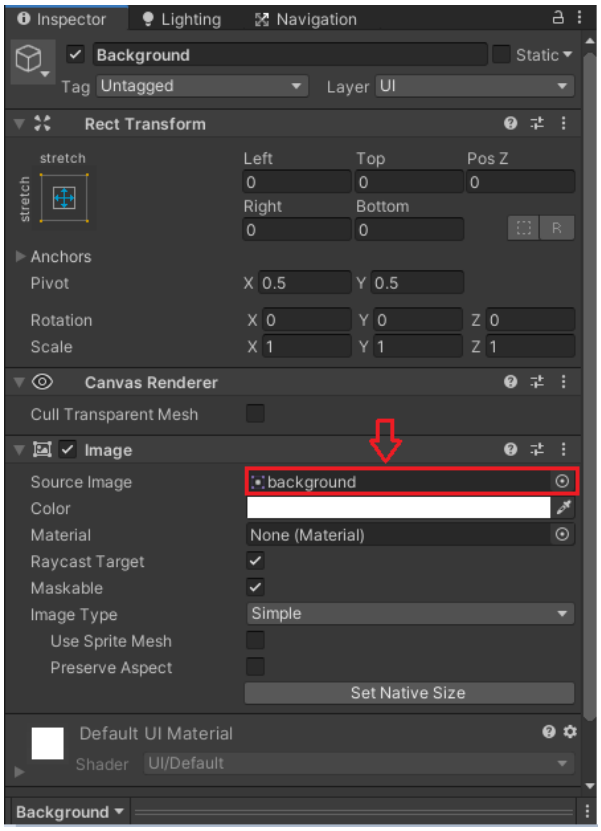
στον παίκτη είτε να ξεκινήσει το παιχνίδι, είτε να αλλάξει κάποιες αρχικές ρυθμίσεις ή να αποχωρήσει από το παιχνίδι.

Για να δημιουργήσουμε ένα Main Menu, αρχικά δημιουργούμε μία σκηνή η οποία θα είναι διαφορετική από αυτήν που θα υλοποιήσουμε το παιχνίδι μας. Έτσι πατάμε δεξί κλίκ στο πεδίο "Project" και στην συνέχεια "Create" → "Scene", όπως φαίνεται και στην Εικόνα 3-4.



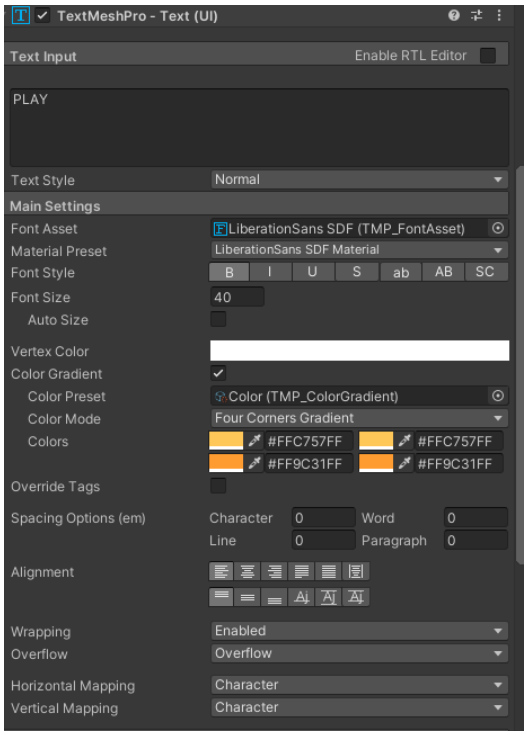
Εικόνα 3 - 4: Δημιουργία σκηνής

Αφού έχουμε δημιουργήσει την σκηνή στην οποία θα υλοποιηθεί το βασικό μενού μας, στην συνέχεια δημιουργούμε έναν καμβά πατώντας δεξί κλικ στο παράθυρο "Hierarchy" και μετά "UI" → "Canvas". Μέσα σε αυτόν τον καμβά θα δημιουργήσουμε ένα "Panel" στο οποίο θα τοποθετήσουμε την εικόνα που θέλουμε να απεικονίζεται σαν background στο Main Menu μας. Έτσι έχοντας επιλεγμένο το "Panel" στο πεδίο "Inspector", τοποθετούμε την εικόνα μας στο Source Image, όπως φαίνεται στην Εικόνα 3-5. Τέλος στο πεδίο Color ρυθμίζουμε την αντίθεση των χρωμάτων έτσι ώστε να μην φαίνεται διαφανής η εικόνα μας.



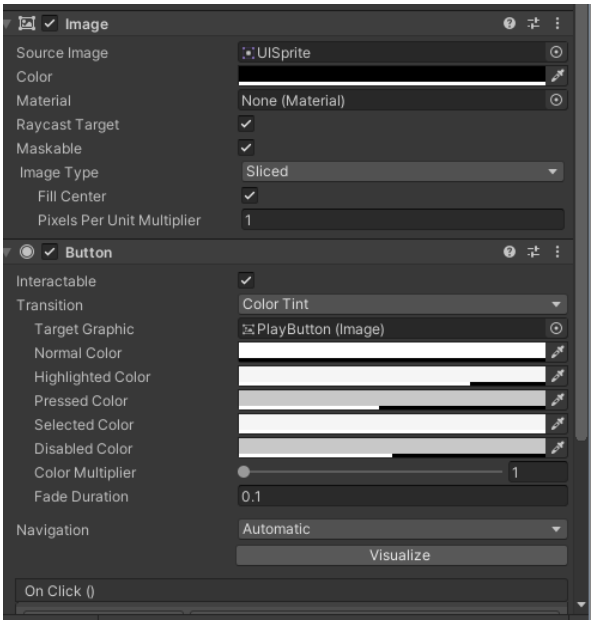
Εικόνα 3 - 5: Background εικόνα

Στην συνέχεια θα χρειαστεί να προσθέσουμε ένα Text ως εξής Canvas → UI → Text-TextMeshPro . Η επιλογή αυτή διαφέρει από την επιλογή του απλού Text, καθώς εδώ μας επιτρέπεται η μορφοποίηση του κειμένου, μέσω του TextMeshPro το οποίο κάνουμε προσθήκη στο Project μας. Αφού έχουμε δημιουργήσει ένα text area, μπορούμε να εφαρμόσουμε κάποιες ρυθμίσεις στο πεδίο TextMeshPro για να βελτιώσουμε την εμφάνιση του Text. Επίσης με την επιλογή του πεδίου Underlay, προσθέτουμε σκιές. Στην Εικόνα 3-6 φαίνονται οι ρυθμίσεις που έχουμε επιλέξει.



Εικόνα 3 - 6: Ρυθμίσεις Text

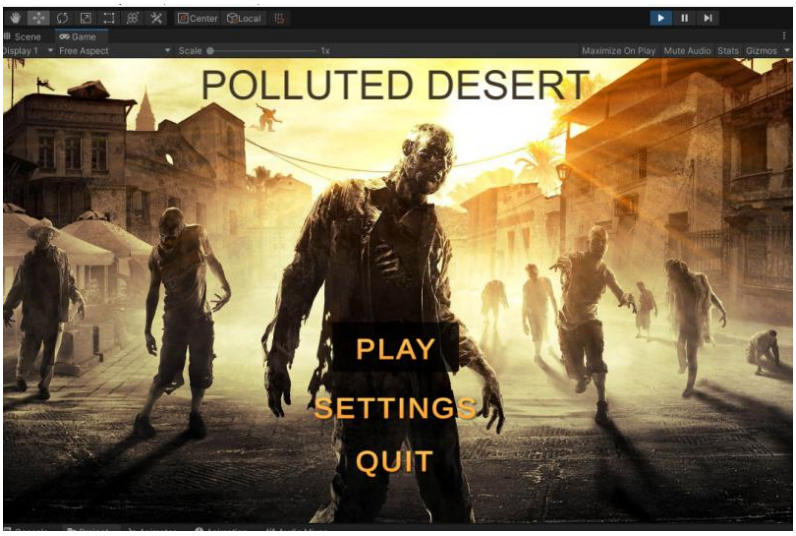
Τώρα θα δημιουργήσουμε το κουμπί που αναπαριστά το Play Button. Έτσι επιλέγοντας ως εξής Canvas → UI → Button δημιουργούμε ένα κουμπί. Μέσα σε αυτό το κουμπί, εισάγουμε το Text που δημιουργήσαμε προηγουμένως, οπότε πλέον έχουμε δημιουργήσει το πρώτο κουμπί με όνομα PLAY. Εφαρμόζοντας τις ρυθμίσεις που παρουσιάζονται στην Εικόνα 3- 7 μεταβάλλουμε την εμφάνιση του, έτσι ώστε αυτό να σκιάζεται όταν εμείς περνάμε τον κέρσορα από επάνω.



Εικόνα 3 - 7: Ρυθμίσεις Play Button

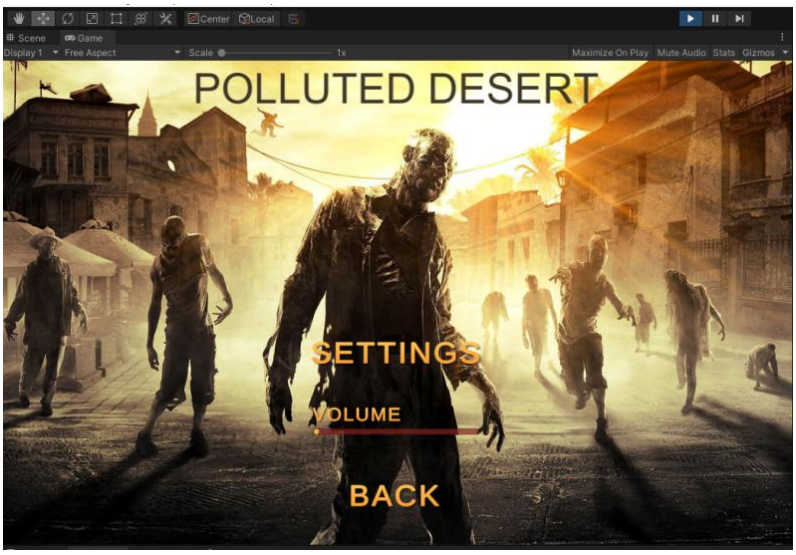
Πατώντας στο πληκτρολόγιο μας Ctrl + D και έχοντας επιλεγμένο το κουμπί, το κάνουμε duplicate έτσι ώστε να δημιουργήσουμε το δεύτερο κουμπί μας (SETTINGS), το οποίο θα είναι στα ίδια

πλαίσια με το προηγούμενο. Αντίστοιχα εφαρμόζουμε και για το τρίτο κουμπί(QUIT). Η τελική εμφάνιση απεικονίζεται στην Εικόνα 3-8.



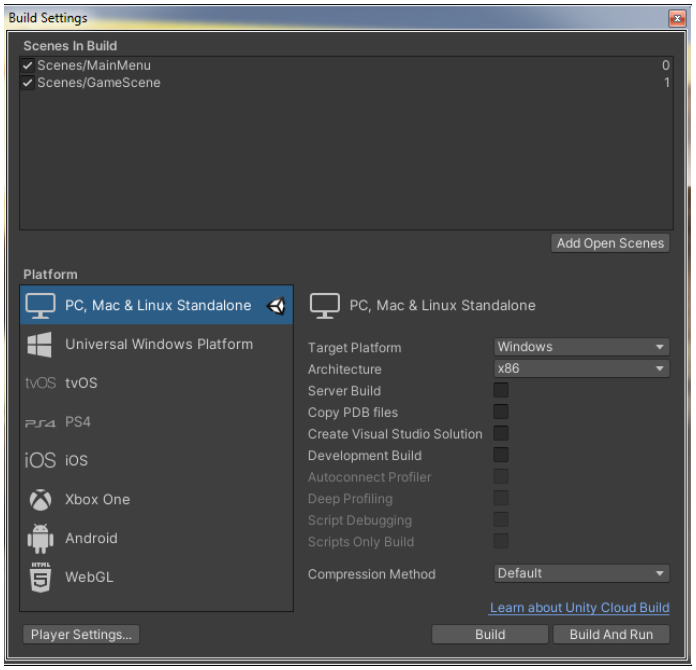
Εικόνα 3 - 8: Main Menu

Ακολουθώντας την ίδια φιλοσοφία θα δημιουργήσουμε το Settings Menu μέσω του οποίου ο χρήστης θα έχει την δυνατότητα να αυξομειώσει την ένταση του ήχου του παιχνιδιού. Δημιουργούμε έτσι δύο Text, ένα με όνομα SETTINGS και ένα με όνομα VOLUME, ένα κουμπί με όνομα BACK και ένα Slider όπως φαίνεται στην Εικόνα 3-9.



Εικόνα 3 - 9: Settings Menu

Για να δώσουμε λειτουργικότητα σε όλα αυτά που δημιουργήσαμε, προσθέτουμε ένα script ως νέο Component στο Main Menu μας. Ο κώδικας που εμφανίζεται στην Εικόνα 3-11 αφορά την λειτουργία του κουμπιού PLAY, μέσω του οποίου αλλάζουμε σκηνή και μεταβαίνουμε στην αντίστοιχη που απεικονίζει το παιχνίδι μας. Για να γίνει αυτή η αλλαγή της σκηνής θα χρειαστεί να δηλώσουμε στο πεδίο Build Settings (File → Build Settings) όλες τις σκηνές που συμπεριλαμβάνονται στο παιχνίδι μας(Εικόνα 3-10). Τοποθετούμε στην πρώτη επιλογή το Main Menu και στην δεύτερη το Game Scene και έτσι μέσα από τον κώδικα της Εικόνας 3-11 μπορούμε να εναλλασσόμαστε μεταξύ αυτών των σκηνών. Στην μέθοδο QuitGame τερματίζουμε το παιχνίδι μας.

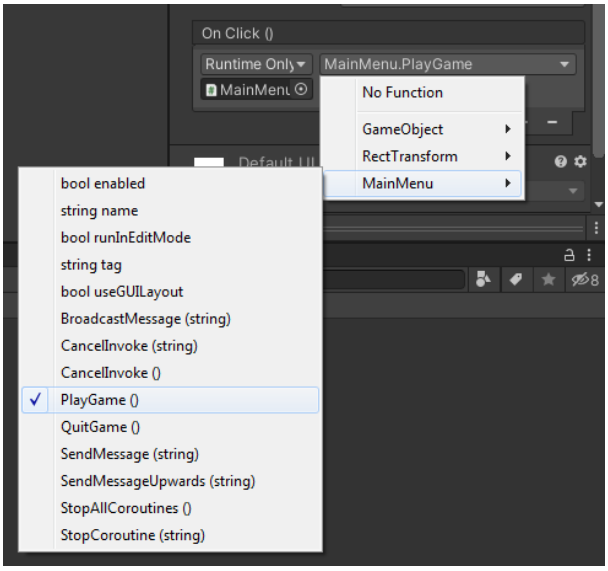


Εικόνα 3 - 10: Build Settings

```
MainMenu.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class MainMenu : MonoBehaviour
7  {
8      public void PlayGame()
9      {
10         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
11     }
12
13     public void QuitGame()
14     {
15         Debug.Log("QuitedGame"); //για λόγους testing
16         Application.Quit();
17     }
18 }
19
```

Εικόνα 3 - 11: Τμήμα Κώδικα από Main Menu Script

Αυτό όμως δεν είναι αρκετό για να λειτουργήσουν τα παραπάνω. Μέσω του event OnClick() μπορούμε να συνδέσουμε κάθε κουμπί με την κλήση της αντίστοιχης μεθόδου. Έτσι στο παράδειγμα της Εικόνα 3-12 ενεργοποιείται η μέθοδος PlayGame( ) στην περίπτωση που ο παίκτης επιλέξει το κουμπί PLAY.



Εικόνα 3 - 12: OnClick( )

Για την μουσική υπόκρουση που θα συναντήσει ο χρήστης κατά την είσοδο του στο μενού θα χρησιμοποιήσουμε μια απαλή μελωδία αντλούμενη από έτοιμο πακέτο assets [50].Στην πορεία θα χρειαστεί να προσαρμόσουμε το Slider μας. Αρχικά θα δημιουργήσουμε το νέο Game Object στο οποίο θα προσθέσουμε ένα Component (Add Component) το Audio Source, στο Output του οποίου θα βάλουμε ένα Music Mixer. Εφόσον έχουμε ολοκληρώσει τα παραπάνω, θα δημιουργήσουμε το script με το οποίο θα δώσουμε την δυνατότητα στον παίκτη να ελέγχει τον ήχο του παιχνιδιού. Έτσι, όπως προβάλλεται στην Εικόνα 3-13, προσθέτουμε ένα mixer, το οποίο μας δίνει την δυνατότητα να κάνουμε αλλαγές πάνω στην ένταση του ήχου. Στην συνέχεια δημιουργούμε μία μέθοδο στην οποία δίνουμε τιμή στο mixer με όνομα “MusicVol” που δημιουργήσαμε πριν, έτσι ώστε τα Desi Belle που αναπαράγεται η μουσική, να είναι συνάρτηση της ρύθμισης από το slider. Τέλος στο Unity είναι απαραίτητο να δηλώσουμε το SetLevel ως On Value Changed γεγονός του Slider.

SetVolume.cs

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

public class SetVolume : MonoBehaviour
{
    public AudioMixer mixer;

    public void SetLevel(float sliderValue)
    {
        mixer.SetFloat("MusicVol", Mathf.Log10(sliderValue) * 20);
        //μετατροπη του sliderValue σε λογαριθμικη τιμή ωστε
        //να ταιριαζει με την τιμη του mixer
    }
}
```

Εικόνα 3 - 13: Τμήμα Κώδικα από SetVolume Script

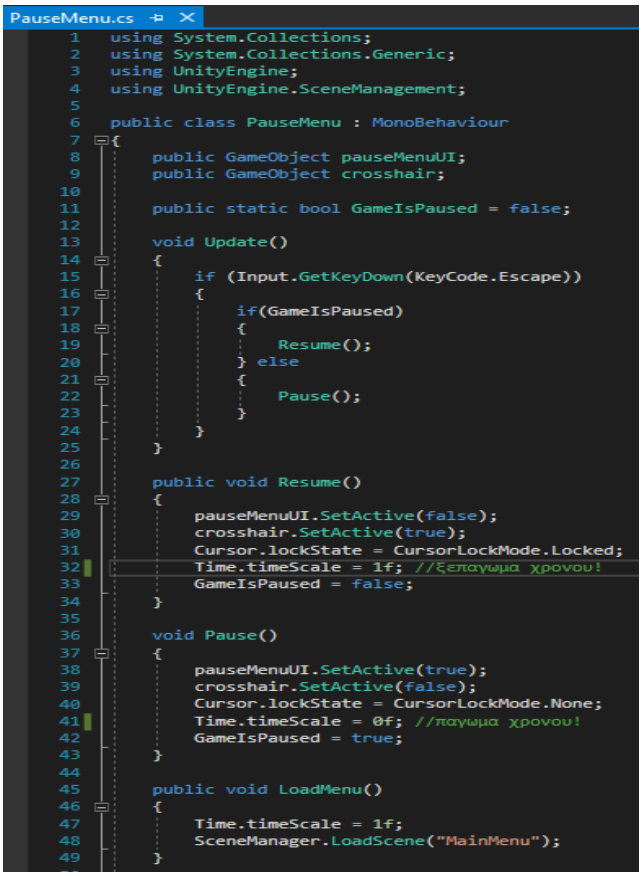
3.3.2 Pause/GameOver Menu

Για να δώσουμε στον χρήστη την δυνατότητα να σταματήσει το παιχνίδι σε οποιαδήποτε στιγμή θελήσει θα κατασκευάσουμε και ένα Pause Menu το οποίο είναι αρκετά πιο απλό στην δημιουργία του. Το Pause Menu δεν σχεδιάζεται σε διαφορετική σκηνή (Scene) αλλά στην ίδια σκηνή που θα αναπτύξουμε το παιχνίδι. Τελώντας μια παρόμοια διαδικασία με το Main Menu, δημιουργούμε αρχικά έναν καμβά στο παράθυρο Hierarchy. Στις περιπτώσεις που δημιουργούμε Menu, είναι προτιμότερο

να επιλέξουμε την 2D απεικόνιση, για να έχουμε πιο κατανοητή όψη αυτών που δημιουργούμε. Στο background image αυτή τη φορά επιλέγουμε το None για να έχουμε φόντο από πίσω το παιχνίδι με μία πιο σκοτεινή απόχρωση, την στιγμή που γίνει η διακοπή μέσω ESC. Το τελικό αποτέλεσμα παρουσιάζεται στην Εικόνα 3-14.

Εικόνα 3 - 14: Pause Menu

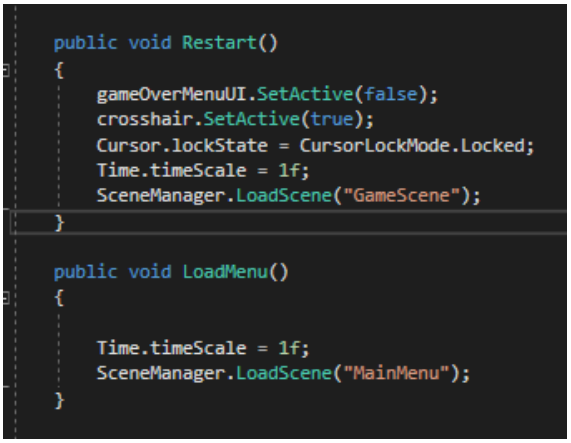
Ήρθε η στιγμή να κάνουμε λειτουργικά όλα τα κουμπιά που έχουμε δημιουργήσει. Αρχικά χρησιμοποιώντας ξανά το event OnClick( ), υλοποιούμε τις αντίστοιχες συνδέσεις μεταξύ των μεθόδων και των κουμπιών. Στην συνέχεια μέσω του script PauseMenu(Εικόνα 3-15), ορίζουμε μία μεταβλητή GameIsPaused η οποία μας βοηθάει στην εναλλαγή μεταξύ του Run Time Environment και του Pause Menu. Επιπρόσθετα χρησιμοποιώντας τα GameObjects του PauseMenu από το παράθυρο Hierarchy τα ενεργοποιούμε και τα απενεργοποιούμε σε κάθε περίπτωση, καθώς επίσης παγώνουμε και ξεπαγώνουμε τον χρόνο, αναλόγως την μέθοδο στην οποία βρισκόμαστε.



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

public class PauseMenu : MonoBehaviour
{
    public GameObject pauseMenuUI;
    public GameObject crosshair;

    public static bool GameIsPaused = false;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if(GameIsPaused)
            {
                Resume();
            } else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        crosshair.SetActive(true);
        Cursor.lockState = CursorLockMode.Locked;
        Time.timeScale = 1f; //ξεπαγωμα χρονου!
        GameIsPaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        crosshair.SetActive(false);
        Cursor.lockState = CursorLockMode.None;
        Time.timeScale = 0f; //παγωμα χρονου!
        GameIsPaused = true;
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }
```

Εικόνα 3 - 15: Τμήμα Κώδικα από PauseMenu Script

Κλείνοντας την ενότητα αυτή, θα γίνει αναφορά του Game Over Menu. Το Menu αυτό αποτελείται από δύο Text και δύο Buttons. Ακολουθώντας την ίδια φιλοσοφία με το Pause Menu και όταν ο παίκτης ηττηθεί, εμφανίζεται το Menu αυτό και του δίνεται η επιλογή είτε να ξαναπαίξει το παιχνίδι, είτε να μετακινηθεί στο αρχικό μενού. Παρατηρούμε ότι υπάρχουν πολλές ομοιότητες μεταξύ των δύο scripts καθώς και στην περίπτωση αυτή εναλλασσόμαστε μεταξύ των βασικών σκηνών όπως φαίνεται και στην Εικόνα 3-16.

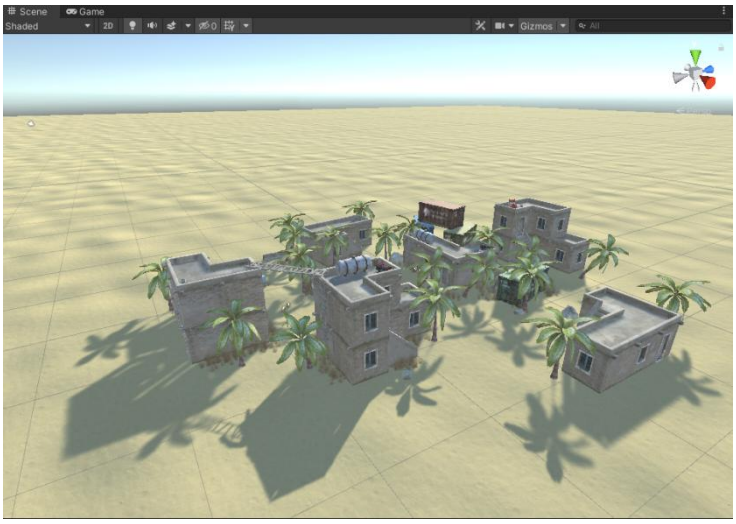


```
public void Restart()
{
    gameOverMenuUI.SetActive(false);
    crosshair.SetActive(true);
    Cursor.lockState = CursorLockMode.Locked;
    Time.timeScale = 1f;
    SceneManager.LoadScene("GameScene");
}

public void LoadMenu()
{

    Time.timeScale = 1f;
    SceneManager.LoadScene("MainMenu");
}
```

Εικόνα 3 - 16: Τμήμα Κώδικα από Game Over Script

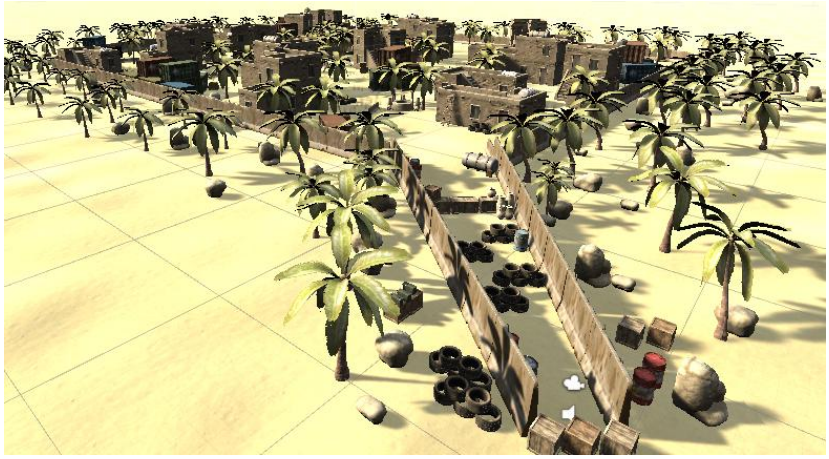
3.4 Terrain – Φωτισμός

Παραπάνω έχουμε αναλύσει με λεπτομέρεια την δημιουργία Terrain από το μηδέν. Ωστόσο υπάρχουν πολλά έτοιμα Assets στο διαδίκτυο, τα οποία μπορεί να χρησιμοποιήσει κανείς ακόμα και με μηδενικό κόστος. Στην περίπτωση μας, για την αναπαράσταση του περιβάλλοντος του παιχνιδιού, χρησιμοποιήσαμε πακέτα από έτοιμα σχεδιαστικά μοντέλα (rocks – plantation – houses – props – street lamps)[49][51-54]. Έχοντας ως βάση το Sample Scene της Εικόνα 3-17



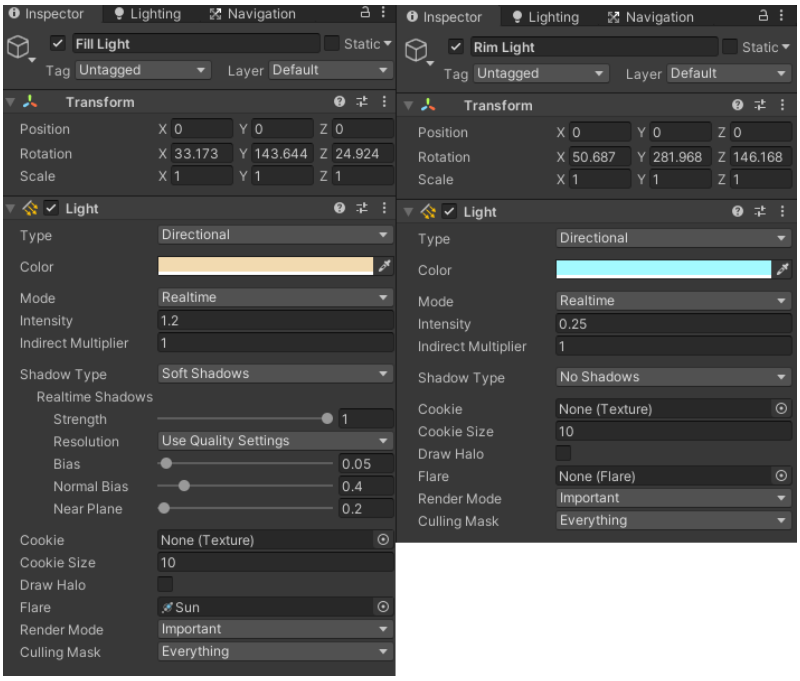
Εικόνα 3 - 17: Sample Scene

προσθέσαμε μία ποικιλία σχεδιαστικών μοντέλων, επεκτείνοντας τις διαστάσεις του και δίνοντας την αίσθηση μίας κατεκτημένης πλέον πόλης από υπερφυσικά όντα. Ο συνδυασμός των εγκαταλελειμμένων κτηρίων και αντικειμένων με την περιφρούρηση της περιοχής από έναν υψηλό τοίχο δίνει στον χρήστη την αίσθηση του κινδύνου και της επιβίωσης που προσπαθούμε να επιτύχουμε. Η τελική μορφή του περιβάλλοντος απεικονίζεται στην Εικόνα 3-18.



Εικόνα 3 - 18: Final Scene

Συνεχίζοντας με τον φωτισμό του περιβάλλοντος χρησιμοποιήσαμε δύο Directional Lights. Το πρώτο ονομάστηκε Fill Light και αναπαριστά το φώς του ηλίου και το δεύτερο ονομάστηκε Rim Light και εκτός από τα οφέλη που προσδίδει στην αίσθηση της αληθοφάνειας της ατμόσφαιρας, τονίζει τα μέρη και τα σχεδιαστικά μοντέλα που δεν εκτίθενται απευθείας στο FillLight.

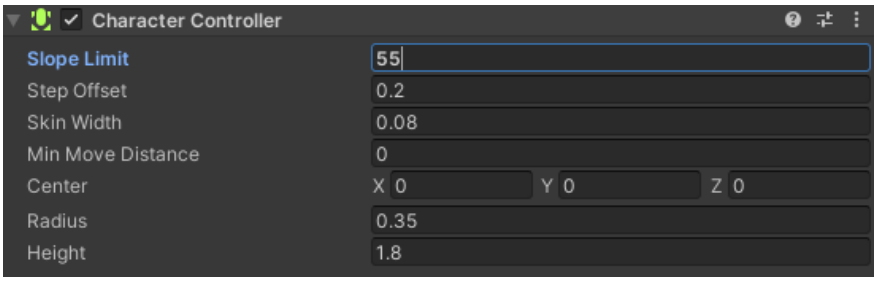


Εικόνα 3 - 19: Fill & Rim Light

Όπως φαίνεται και στην Εικόνα 3-19, στην περίπτωση του Fill Light η επιλογή Shadow Type έχει την τιμή Soft Shadows ενώ αντιθέτως στην περίπτωση του RimLight η επιλογή αυτή είναι απενεργοποιημένη ώστε να μην υπάρχουν διπλές σκιάσεις σε κανένα αντικείμενο. Επιπρόσθετα διακρίνουμε ότι στο FillLight η επιλογή του Flare έχει σαν αποδέκτη το Sun, με σκοπό να προσομοιώνει την επίδραση των διαθλάσεων του φωτός μέσα σε ένα φακό κάμερας .Τέλος στην καρτέλα Lighting και συγκεκριμένα στο πεδίο Environment Lighting → Source έχουμε χρησιμοποιήσει την επιλογή Gradient, αυξάνοντας την τιμή του Sky Color κατά ένα και έτσι δίνεται περισσότερος φωτισμός στο περιβάλλον μας ακόμα και στα σημεία που δεν ακτινοβολεί το φως το ηλίου.

3.5 Player

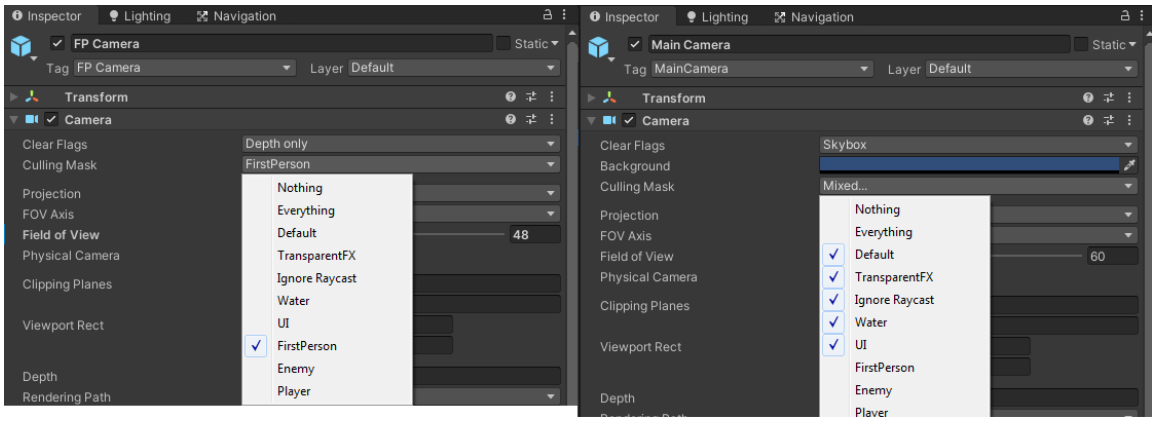
Εφόσον έχουμε τελειώσει με το περιβάλλον και τον φωτισμό αυτού, μπορούμε να δημιουργήσουμε τον χαρακτήρα που θα διαχειρίζεται ο χρήστης. Το πρώτο πράγμα που έχουμε να κάνουμε είναι να φτιάξουμε ένα καινούργιο Game Object το οποίο θα ονομάσουμε Player και θα το τοποθετήσουμε στο σημείο που θέλουμε να βρίσκεται όταν πραγματοποιείται η έναρξη του παιχνιδιού. Έχοντας ολοκληρώσει τα παραπάνω, θα χρειαστεί να προσθέσουμε ένα Component που ονομάζεται Character Controller και θα υπεύθυνο για την κίνηση και για την αλληλεπίδραση του χαρακτήρα με το περιβάλλον(Εικόνα 3-20). Συγκεκριμένα το πεδίο Slope Limit είναι υπεύθυνο στο να περιορίσει τον Collider να ανεβαίνει μόνο στις πλαγιές που είναι λιγότερο απότομες από την υποδεικνυόμενη τιμή. Το Radius είναι το πλάτος του χαρακτήρα μας και το Height είναι το ύψος του.



Εικόνα 3 - 20: Character Controller

3.5.1 Player Camera

Για να συνεχίσουμε την επεξεργασία του χαρακτήρα μας, θα χρειαστεί πρώτα να δημιουργήσουμε δύο κάμερες. Η πρώτη ονομάζεται Main Camera και προβάλει το περιβάλλον του παιχνιδιού μέσα από τα μάτια του χαρακτήρα, ενώ η δεύτερη ονομάζεται FP Camera και είναι υπεύθυνη για την οπτική αναπαράσταση των χεριών και των όπλων που έχει στην διάθεση του. Αυτό επιτυγχάνεται μέσα από την επιλογή Culling Mask και η μόνη διαφορά μεταξύ τους είναι ότι η Main Camera αγνοεί τα GameObjects που βρίσκονται σε layer First Person, ενώ η FP Camera εμφανίζει μόνο τα GameObjects που βρίσκονται στο layer αυτό(Εικόνα 3-21).



Εικόνα 3 - 21: Camera Settings

Τέλος από την FP Camera θα αφαιρέσουμε το component AudioListener, έτσι ώστε να μην υφίσταται διπλή αναπαραγωγή των ήχων του παιχνιδιού.

3.5.2 Free Look

Στο σημείο αυτό θα προγραμματίσουμε την κίνηση της κάμερας μέσω του κέρσορα. Έτσι δημιουργούμε ένα script που ονομάζουμε MouseLook. Όπως παρουσιάζεται στην Εικόνα 3-22, μέσω της μεθόδου LockAndUnlockCursor( ), πατώντας το κουμπί TAB δίνουμε το δικαίωμα στον χρήστη να ξεκλειδώσει-εξαφανίσει και να κλειδώσει-εμφανίσει αντίστοιχα τον κέρσορα στο κέντρο της οθόνης. Επιπλέον με την μέθοδο LookAround( ), η οποία καλείται από την Update( ) και επομένως εκτελείται σε κάθε frame, αντλούμε τις συντεταγμένες του κέρσορα και τροφοδοτούμε αυτές στην σχετική θέση του Player. Οι τιμές αυτές οριοθετούνται από -70f έως +80f στον άξονα του Y.

```
    void LockAndUnlockCursor() {

        if(Input.GetKeyDown(KeyCode.Tab)) {

            if(Cursor.lockState == CursorLockMode.Locked) {

                Cursor.lockState = CursorLockMode.None; //κλειδωμα-ξεκλειδωμα cursor

            } else {

                Cursor.lockState = CursorLockMode.Locked;
                Cursor.visible = false;

            }

        }

    } // lock and unlock

    void LookAround() {

        current_Mouse_Look = new Vector2( Input.GetAxis("Mouse Y"), Input.GetAxis("Mouse X"));//MOUSE_Y-κάθετα MOUSE_X οριζόντια

        look_Angles.x += current_Mouse_Look.x * sensivity * (invert ? 1f : -1f);
        look_Angles.y += current_Mouse_Look.y * sensivity;

        look_Angles.x = Mathf.Clamp(look_Angles.x, default_Look_Limits.x, default_Look_Limits.y);//θέτουμε τα όρια(apo -70 - +80)

        lookRoot.localRotation = Quaternion.Euler(look_Angles.x, 0f, 0f);
        playerRoot.localRotation = Quaternion.Euler(0f, look_Angles.y, 0f);

    } // look around
}
```

Εικόνα 3 - 22: Τμήμα Κώδικα από MouseLook Script

3.5.3 Player's Movement

Για την κίνηση του παίκτη μας, θα χρησιμοποιήσουμε δυο διαφορετικά scripts τα οποία και θα του προσθέσουμε ως components στο παράθυρο Inspector.

Το πρώτο script ονομάζεται Movement και είναι υπεύθυνο για την βασική κίνηση του χαρακτήρα μας (Εικόνα 3-23). Μέσω της μεθόδου Update( ) καλούμε την συνάρτηση MoveThePlayer( ), όπου βασισμένοι στην ίδια λογική με την κίνηση του κέρσορα ανατροφοδοτούμε την μεταβλητή move_Direction με τιμές που αναπαριστούν το πάτημα των πλήκτρων (A,D,W,S). Στην συνέχεια, αφού ελέγξουμε εάν ο χρήστης πατάει το πλήκτρο Space οπότε και προσομοιώνουμε το άλμα του χαρακτήρα, χρησιμοποιούμε την ενσωματωμένη συνάρτηση Move( ) του Unity ώστε να κινηθούμε με επιτυχία προς οποιαδήποτε κατεύθυνση.

```
22      void Update ()
23      {
24          MoveThePlayer();
25      }
26
27      void MoveThePlayer() {
28                                          //A-D -> Οριζόντια                  //W-S -> Κάθετα
29          move_Direction = new Vector3(Input.GetAxis("Horizontal"), 0f, Input.GetAxis("Vertical"));
30
31          move_Direction = transform.TransformDirection(move_Direction);
32          move_Direction *= speed * Time.deltaTime;
33
34          ApplyGravity();
35
36          character_Controller.Move(move_Direction);
37
38      } // move player
39
40      void ApplyGravity() {
41
42          vertical_Velocity -= gravity * Time.deltaTime;
43          // jump
44          PlayerJump();
45
46          move_Direction.y = vertical_Velocity * Time.deltaTime;
47
48      } // apply gravity
49
50      void PlayerJump() {
51
52          if(character_Controller.isGrounded && Input.GetKeyDown(KeyCode.Space)) {
53              vertical_Velocity = jump_Force;
54          }
55
56      }
57  }
```

Εικόνα 3 - 23: Τμήμα Κώδικα από Movement Script

Το δεύτερο script ονομάζεται DifferentTypesOfMovement (Εικόνα 3-24) και αφορά κάποιους συμπληρωματικούς τρόπους κίνησης του χαρακτήρα μας (Sprint, Crouch, Prone).

Κρατώντας πατημένο το πλήκτρο SHIFT και μόνο εάν ο παίκτης έχει επαρκή ποσότητα από Stamina, κάτι που θα εξηγηθεί λεπτομερώς αργότερα, αλλάζουμε την τιμή της ταχύτητας του χαρακτήρα θέτοντας την σε μια αρκετά υψηλότερη, sprint_Speed.

Πατώντας το πλήκτρο C, και μόνο εάν ο χαρακτήρας δεν βρίσκεται ήδη σε αυτήν την κατάσταση (Crouching), αλλάζουμε την οπτική της κάμερας και την θέτουμε ίση με την μεταβλητή crouch_Height για να προσομοιώσουμε το σκύψιμο αυτού. Επίσης, μεταβάλλουμε την τιμή με την οποία κινούμαστε και την θέτουμε ίση με την μεταβλητή crouch_Speed, η οποία έχει οριστεί προηγουμένως.

Τέλος , με το πλήκτρο Z προσομοιώνεται το ‘ξάπλωμα’ του παίκτη ακολουθώντας τα ίδια βήματα με την παραπάνω συνάρτηση.

```
159     void Prone(){
160 
161         if (Input.GetKeyDown(KeyCode.Z)){
162 
163             if (is_Proning){ // if we are proning - stand up
164 
165                 look_Root.localPosition = new Vector3(0f, stand_Height, 0f);
166                 movement.speed = move_Speed;
167 
168                 footsteps.step_Distance = walk_Step_Distance;
169                 footsteps.volume_Min = walk_Volume_Min ;
170                 footsteps.volume_Max = walk_Volume_Max ;
171 
172                 is_Proning = false;
173             }
174             else { // if we re not proning -- prone !
175 
176                 look_Root.localPosition = new Vector3(0f, prone_Height, 0f);
177                 movement.speed = prone_Speed;
178 
179                 footsteps.step_Distance = prone_Step_Distance;
180                 footsteps.volume_Min = prone_Volume;
181                 footsteps.volume_Max = prone_Volume;
182 
183                 is_Proning = true;
184             }
185         }
186     }//prone
187 }
188 
```

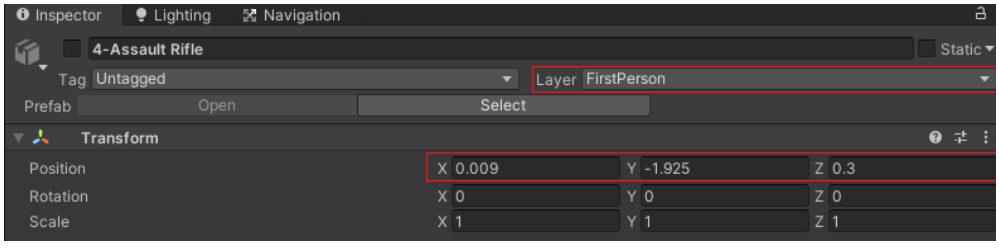
Εικόνα 3 - 24: Τμήμα Κώδικα από DifferentTypeOfMovement Script

3.6 Weapons

Συνεχίζοντας θα αναφερθούμε στην δημιουργία των όπλων που έχει στην κατοχή του ο παίκτης. Θα χρησιμοποιήσουμε ένα πακέτο, σχεδιασμένο ειδικά για εκπαιδευτικούς σκοπούς, από assets που περιλαμβάνει έτοιμα τρισδιάστατα μοντέλα όπλων καθώς και εύκολη χρήση των materials που έχει διαθέσιμα. [55]

Αρχικά ,στην καρτέλα Hierarchy και συγκεκριμένα μέσα στον Player που έχουμε ήδη συνθέσει, θα δημιουργήσουμε ένα νέο Game Object ως παιδί του που θα ονομαστεί FP Weapons. Μέσα σε αυτόν τον «υποφάκελο» (FP Weapons) θα τοποθετήσουμε τέσσερα όπλα(Axe, Revolver, Shotgun, Assault Rifle) και χρησιμοποιώντας το κατάλληλο material για κάθε περίπτωση θα δώσουμε την τελική μορφή για την οπτική αναπαράσταση αυτών.

Αφού τελειώσαμε με την απεικόνιση των όπλων θα πρέπει να επανατοποθετηθούν σε σχέση με την θέση του Player. Αυτό θα επιτευχθεί μέσω του Component Transform → Position. Παράλληλα θέτουμε σε όλα τα διαθέσιμα όπλα την επιλογή FirstPerson στο πεδίο Layer που προαναφέραμε, έτσι ώστε να εμφανίζονται μονάχα στην FP Camera μας, όπως προβάλλεται στην Εικόνα 3-25.



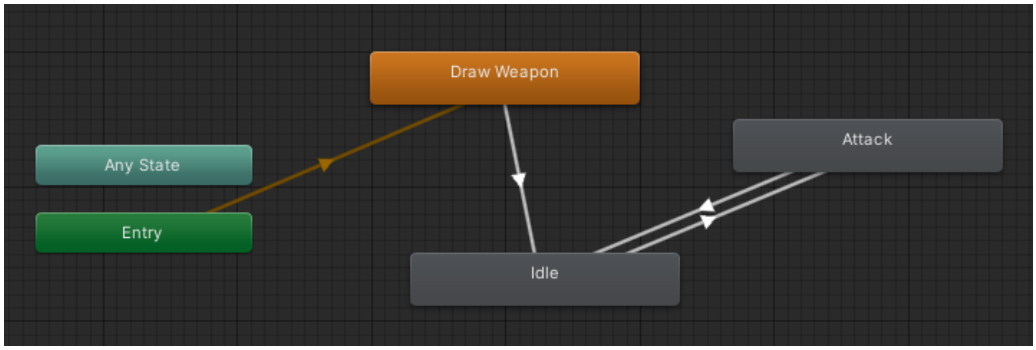
Εικόνα 3 - 25: Assault Rifle Settings

3.6.1 Weapons Animator Controllers

Στην πορεία έχοντας τελειώσει με το σχεδιαστικό κομμάτι θα προσπαθήσουμε να δώσουμε «ζωή» στα όπλα μας. Θα πρέπει να κατασκευάσουμε τέσσερεις διαφορετικούς Animator Controller , έναν για κάθε ξεχωριστό όπλο και να συνδέσουμε αυτούς με το αντίστοιχο πεδίο στο Component Animator → Controller.

Ανοίγοντας πλέον το παράθυρο Animator και πατώντας δεξί κλικ Create State→ Empty, μας δίνεται η δυνατότητα να δημιουργήσουμε ένα νέο Animation State. Έτσι , φτιάχνουμε τρία καινούργια Empty States τα οποία μετονομάζουμε σε DrawWeapon , Idle , Attack θέτοντας παράλληλα σαν Default State το Draw Weapon καθώς αυτή είναι το αρχική κατάσταση που θέλουμε να εμφανίζεται. Σε κάθε State προσθέτουμε το αντίστοιχο Animation και συγκεκριμένα στο πεδίο Motion επιλέγουμε τα αντίστοιχα Animations για τα Draw, Idle και Attack.

Επιπλέον, πρέπει να ενώσουμε τα States μεταξύ τους με μεταβάσεις που ονομάζονται transitions (δεξί κλικ →Make Transition) και αναπαριστούν τον τρόπο και τη σειρά με την οποία θα προβάλλονται τα διάφορα States. Η διαδικασία αυτή φαίνεται στην Εικόνα 3-26 :



Εικόνα 3 - 26: Weapons Animator Tab

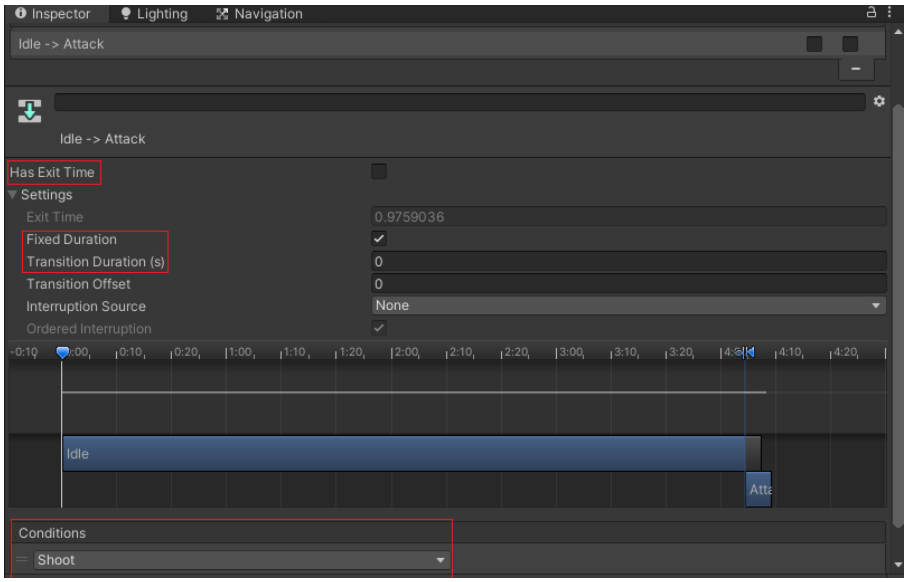
Προτού εξηγήσουμε τις διαφορετικές ρυθμίσεις των transitions , σημαντικό είναι να αναφερθούμε στην ύπαρξη και τη δημιουργία της trigger παραμέτρου “Shoot” η οποία αποτελεί τον συνδετικό κρίκο μεταξύ του State Idle και του State Attack(Εικόνα 3-27). Ουσιαστικά για να κριθεί επιτυχής η μετάβαση αυτή θα πρέπει να ικανοποιηθεί η συνθήκη “Shoot” μέσω του κώδικα που θα εξηγηθεί στην πορεία.

Οι ρυθμίσεις των transitions που θα ασχοληθούμε είναι οι εξής [56]:

HasExitTime → Ο χρόνος εξόδου είναι μια ειδική μετάβαση που δεν βασίζεται σε κάποια παράμετρο. Αντ 'αυτού, βασίζεται στον κανονικοποιημένο χρόνο του State. Θέτοντας για παράδειγμα την τιμή 1, επιτρέπουμε στο αντίστοιχο animation να παρουσιαστεί ολοκληρωμένο πριν μεταβούμε στο επόμενο.

Fixed Duration → Εάν είναι επιλεγμένο το πλαίσιο αυτό, ο χρόνος μετάβασης ερμηνεύεται σε δευτερόλεπτα.

Transition Duration → Η διάρκεια της μετάβασης, σε κανονικοποιημένο χρόνο ή δευτερόλεπτα ανάλογα με το πλαίσιο σταθερής διάρκειας, σε σχέση με τη διάρκεια της τρέχουσας κατάστασης. Καθορίζει ουσιαστικά τον χρόνο στον οποίο θα πραγματοποιηθεί το transition μεταξύ δυο καταστάσεων.

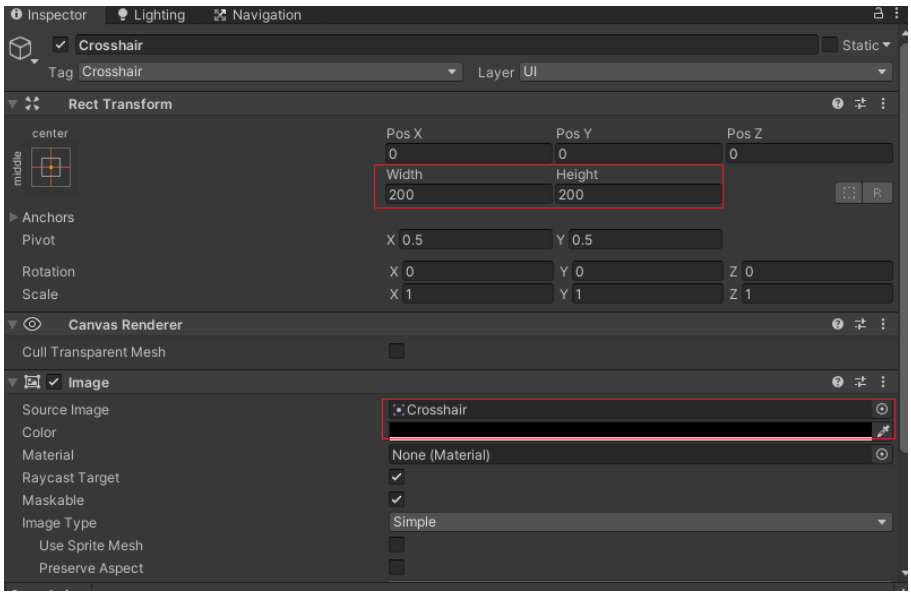


Εικόνα 3 - 27: Idle-Attack transition settings

3.6.2 Zoom Effect

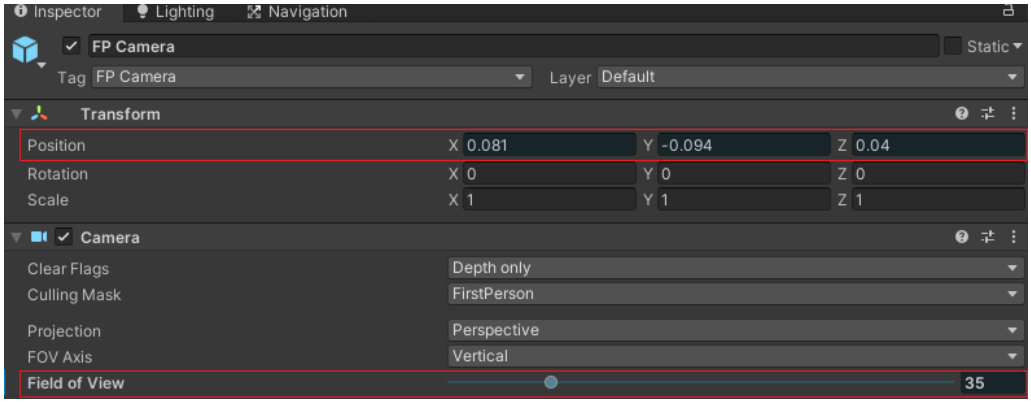
Για να προσδώσουμε ακόμη περισσότερο αληθοφάνεια στην χρήση των όπλων μας, θα δημιουργήσουμε ένα νέο Animation κλιπ που προσομοιάζει την μεγέθυνση της στόχευσης.

Αρχικά, όπως και σε όλα τα υπόλοιπα Action Games χρειαζόμαστε μια ένδειξη για τον χρήστη που εμφανίζει το σημείο στο οποίο στοχεύει το εκάστοτε όπλο. Προσθέτουμε λοιπόν στην καρτέλα Hierarchy έναν καινούργιο καμβά και μέσα σε αυτόν μια άδεια εικόνα. Στην πορεία, επιλέγουμε την εικόνα του στόχου που θέλουμε να χρησιμοποιήσουμε και αφού τροποποιήσουμε τον τύπο υφής της (Inspector-Texture type →Sprite 2D and UI) ώστε να μπορεί να χρησιμοποιηθεί κατάλληλα , προστίθεται αυτή σαν πηγή εικόνας . Ακόμη, μεταβάλλουμε το χρώμα σε μαύρο και μεγεθύνουμε τις διαστάσεις τις όπως εμφανίζεται στην Εικόνα 3-28.



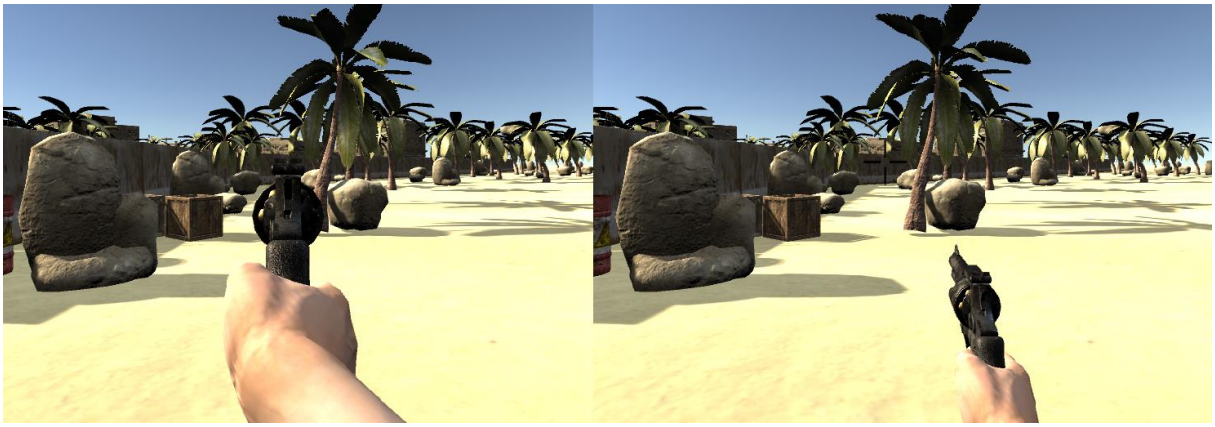
Εικόνα 3 - 28: Crosshair Settings

Έχοντας ολοκληρώσει την επεξεργασία του στόχου, μπορούμε να προχωρήσουμε στην προσομοίωση της επιμέρους στόχευσης. Επιλέγουμε την FP Camera, μιας και αυτή μονάχα εμφανίζει τα όπλα και προσθέτουμε ως νέο Component ένα Animator. Ακολουθώντας την πλέον γνωστή διαδικασία, κατασκευάζουμε ένα Animator Controller και το συνδέουμε με το Component Animator →Controller που μόλις δημιουργήσαμε. Χρησιμοποιούμε για πρώτη φορά το παράθυρο Animation για την σύνθεση ενός καινούργιου Animation Clip. Η διαδικασία αυτή είναι αρκετά απλή. Διαλέγουμε το κουμπί Create και αποθηκεύουμε σε χώρο της αρεσκείας μας το Animation Clip με όνομα "ZoomIn". Για να καθορίσουμε τον τρόπο αναπαραγωγής του, επιλέγουμε το κόκκινο κουμπί Record και μετακινούμαστε στο Sample 60. Τότε, μέσω της καρτέλας Inspector και πιο συγκεκριμένα του Component Transform μεταβάλουμε την θέση αλλά και το οπτικό πεδίο της FP Cameras. Οι αλλαγές αυτές παρουσιάζονται στην Εικόνα 3-29.



Εικόνα 3 - 29: ZoomIn Settings

Τελικά, μέσω του παραθύρου Animator δημιουργούμε ένα νέο Animation State με το όνομα Idle που θα έχει τον ρόλο του Default State έτσι ώστε να αποτρέπει την αδιάκοπη επανάληψη του ZoomIn. Δεν θα χρειαστεί να ενώσουμε τα States μεταξύ τους με μεταβάσεις, διότι αυτό θα υλοποιηθεί με τον κώδικα. Το μόνο που έχει απομείνει είναι να κατασκευάσουμε ένα αντίγραφο του Animation που συνθέσαμε , να το ονομάσουμε ZoomOut και να θέσουμε αρνητική τιμή στην ταχύτητα του ώστε αυτό να προβληθεί αντίθετα. Η τελική όψη του φαίνεται στην Εικόνα 3-30 :



Εικόνα 3 - 30: ZoomIn –ZoomOut Effect

3.6.3 Weapons Scripts

Στην συνέχεια, για να καθορίσουμε τον τρόπο εμφάνισης, διαχείρισης και σωστής λειτουργίας των παραπάνω θα δημιουργήσουμε τρία νέα scripts.

Το πρώτο script ονομάζεται Organizer (Εικόνα 3-31) και θα προστεθεί ως νέο Component σε κάθε όπλο που δημιουργήσαμε ανεξαιρέτως. Στο script αυτό για να μπορέσουμε να κατηγοριοποιήσουμε τα επιμέρους στοιχεία του θα χρησιμοποιήσουμε την έννοια του enum. Το enum είναι μια ειδική "κλάση" που αντιπροσωπεύει μια ομάδα σταθερών (μεταβλητές που δεν αλλάζουν τιμή/ μεταβλητές μόνο για ανάγνωση)[57]. Επιπλέον, αυτό είναι υπεύθυνο για όλες τις «δευτερεύουσες» λειτουργίες των όπλων. Επεξεργάζεται την κατάλληλη προβολή και μετάβαση μεταξύ των διαφορετικών State χρησιμοποιώντας την παράμετρο “Shoot” που δημιουργήσαμε στην προηγούμενη ενότητα αλλά και την αναπαραγωγή των ήχων που θα εξηγηθεί αργότερα.

Organizer.cs

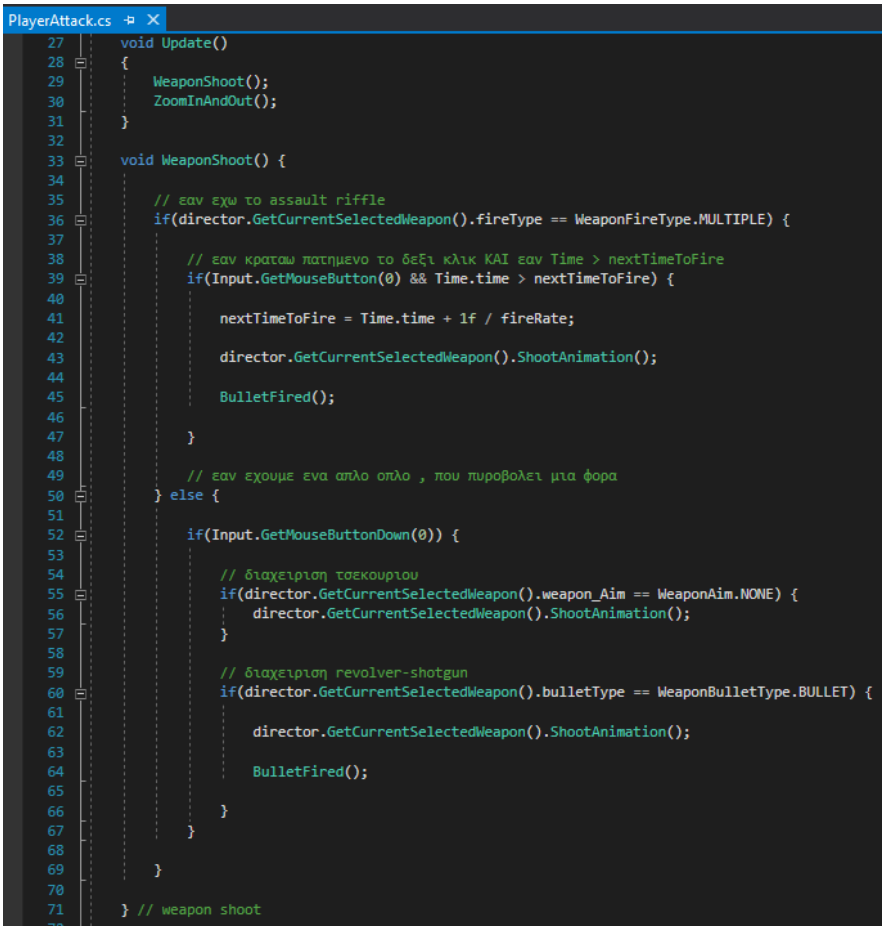
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public enum WeaponAim {
    NONE,
    AIM
}

public enum WeaponFireType {
    SINGLE,
    MULTIPLE
}

public enum WeaponBulletType {
    BULLET,
    NONE
}

public class Organizer : MonoBehaviour
{
    public WeaponAim weapon_Aim; //τροπος στοχευσης οπλου--τσεκουρι μονο none.
    public WeaponFireType fireType;//τροπος πυροβολισμου--1 σφαιρα ή πολλες
    public WeaponBulletType bulletType;//τυπος σφαιρας
    public GameObject attack_Point;//για το τσεκουρι μονο , ανιχνευει την συγκρουση με αντιπαλο

    private Animator animator;
    [SerializeField]
    private GameObject muzzleFlash; // για shotgun και revolver
    [SerializeField]
    private AudioSource shootSound, reload_Sound;

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

    public void ShootAnimation() {
        animator.SetTrigger("Shoot");
    }
```

Εικόνα 3 - 31: Τμήμα Κώδικα από Organizer Script

Το δεύτερο script ονομάζεται Director (Εικόνα 3-32) και θα προστεθεί ως νέο Component στον ίδιο τον Player. Ο κύριος στόχος αυτού του script είναι η επιτυχής εμφάνιση και ενεργοποίηση του επιλεγμένου όπλου από τον χρήστη. Αυτό πραγματοποιείται μέσω της μεθόδου Update( ), οπού ελέγχοντας σε κάθε frame του παιχνιδιού την αλληλεπίδραση του χρήστη με τους αριθμούς του πληκτρολόγιου(1–4, αναπαριστούν τα διάφορα όπλα) καλείται η μέθοδος TurnOnSelectedWeapon(int weaponIndex). Με τη σειρά της η μέθοδος αυτή, απενεργοποιεί το τρέχον όπλο και ενεργοποιεί – εμφανίζει το πλέον επιλεγμένο.

Εικόνα 3 - 32: Τμήμα Κώδικα από Director Script

Τέλος, το τρίτο script ονομάζεται PlayerAttack (Εικόνα 3-33) και θα προστεθεί όπως και το προηγούμενο ως νέο Component στον Player. Το συγκεκριμένο script αφορά την επιτυχημένη λειτουργία-χρήση του κάθε όπλου αλλά και την υλοποίηση του ZoomEffect που προαναφέραμε. Η μέθοδος WeaponShoot( ), εκμεταλλευόμενη τα enums του script Organizer ελέγχει τον τύπο του όπλου που ετοιμαζόμαστε να χρησιμοποιήσουμε και ανάλογα την περίσταση καλεί το αντίστοιχο animation αλλά και πυροβολεί μέσω της μεθόδου BulletFired( ) που θα εξηγηθεί αργότερα. Παρομοίως η μέθοδος ZoomInAndOut( ), ελέγχει τον τύπο του όπλου και μονάχα στην περίπτωση που αυτό μπορεί να κάνει χρήση του ZoomEffect καλεί το αντίστοιχο animation.



```
PlayerAttack.cs
27      void Update()
28      {
29          WeaponShoot();
30          ZoomInAndOut();
31      }
32
33      void WeaponShoot() {
34
35          // εαν εχω το assault riffle
36          if(director.GetCurrentSelectedWeapon().fireType == WeaponFireType.MULTIPLE) {
37
38              // εαν κραταω πατημενο το δεξι κλικ ΚΑΙ εαν Time > nextTimeToFire
39              if(Input.GetMouseButton(0) && Time.time > nextTimeToFire) {
40
41                  nextTimeToFire = Time.time + 1f / fireRate;
42
43                  director.GetCurrentSelectedWeapon().ShootAnimation();
44
45                  BulletFired();
46
47              }
48
49              // εαν εχουμε ενα απλο οπλο , που πυροβολει μια φορα
50          } else {
51
52              if(Input.GetMouseButtonDown(0)) {
53
54                  // διαχειριση τσεκουριου
55                  if(director.GetCurrentSelectedWeapon().weapon_Aim == WeaponAim.NONE) {
56                      director.GetCurrentSelectedWeapon().ShootAnimation();
57                  }
58
59                  // διαχειριση revolver-shotgun
60                  if(director.GetCurrentSelectedWeapon().bulletType == WeaponBulletType.BULLET) {
61
62                      director.GetCurrentSelectedWeapon().ShootAnimation();
63
64                      BulletFired();
65
66                  }
67              }
68
69          }
70
71      } // weapon shoot
```

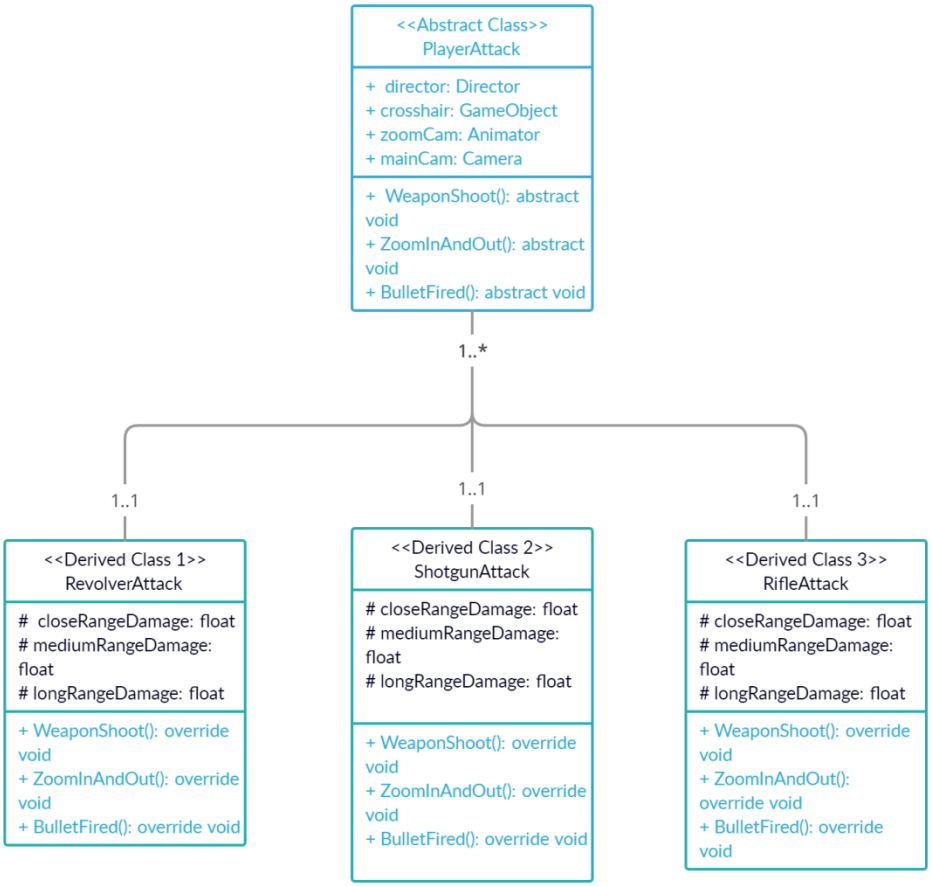
Εικόνα 3 - 33: Τμήμα Κώδικα από PlayerAttack Script

Με στόχο την υλοποίηση του παραπάνω script με αντικειμενοστρεφή τρόπο έτσι ώστε να απλοποιείται, να επαναχρησιμοποιείται ο κώδικας μας αλλά και να προσδίδει επιπλέον λειτουργίες στα ήδη υπάρχοντα στοιχεία μαζί με την δυνατότητα διαφορετικής εφαρμογής της ίδιας μεθόδου στο κάθε όπλο πραγματοποιήσαμε τα παρακάτω :

- Μετατροπή του script PlayerAttack σε μια abstract-γονική κλάση οπού θα αποτελεί τον πυλώνα της επίθεσης του παίκτη.
- Τρείς διαφορετικές υποκλάσεις (RevolverAttack, ShotgunAttack, RifleAttack) οι οποίες κληρονομούν την abstract κλάση PlayerAttack, μία για το κάθε όπλο για την ορθή χρησιμοποίηση τους από τον παίκτη.
- Μία ανεξάρτητη κλάση (AxeAttack) υπεύθυνη για την ανίχνευση και περάτωση της χρήσης του τσεκουριού.

Όπως προαναφέρθηκε, παρουσιάζεται στο Σχήμα 3-1 η αντικειμενοστρεφής δομή που δημιουργήθηκε για να καλύψει τις παραπάνω ανάγκες. Έτσι, η abstract κλάση PlayerAttack περιέχει τις τρείς μεθόδους υπεύθυνες για την επιτυχημένη επίθεση του παίκτη χωρίς όμως σώμα μεθόδου, δηλαδή πραγματική υλοποίηση στην ίδια την κλάση. Οι αντίστοιχες υλοποιήσεις γίνονται κάθε φορά στις υποκλάσεις των όπλων, οπού εκεί μπορούν και ελέγχονται διαφορετικές συνθήκες κάθε φορά εξειδικευμένες για το αντίστοιχο όπλο. Με τον τρόπο αυτό μπορούμε να πραγματοποιήσουμε διαφορετική ζημιά με τον ίδιο τρόπο επίθεσης (πυροβολισμός) εξαρτώμενη από το όπλο που χρησιμοποιεί την τρέχουσα στιγμή ο παίκτης αλλά και μείωση ή αύξηση της

ζημίας που θα δεχόταν υπό κανονικές συνθήκες ο αντίπαλος σε σχέση με την απόσταση του από τον παίκτη (π.χ το Shotgun έχει μεγαλύτερο closeRangeDamage απ' ότι το Rifle). Τέλος, η κλάση AxeAttack στέκεται μόνη της και δεν αποτελεί κομμάτι της αντικειμενοστρεφούς αυτής δομής, καθώς δεν θα μπορούσε να υλοποιήσει τις απαραίτητες μεθόδους της κλάσης PlayerAttack αφού δεν πρόκειται για κάποιο όπλο που μπορεί να πυροβολήσει ή να χρησιμοποιήσει για να μεγεθύνει την οπτική του ο χρήστης (ZoomInAndOut).



Σχήμα 3 - 1: Διάγραμμα Κλάσεων της Αντικειμενοστρεφούς Δομής

3.7 Enemies

Αφού έχουμε ολοκληρώσει και την δημιουργία των όπλων θα προχωρήσουμε στο κομμάτι υλοποίησης των αντιπάλων του παίκτη μας. Για αυτό το κομμάτι της εργασίας θα χρησιμοποιήσουμε ένα ευρέως διαδεδομένο και άκρως λεπτομερές πακέτο από assets που περιλαμβάνει το τρισδιάστατο μοντέλο του αντιπάλου (Zombie) αλλά και διάφορα animations.[58]

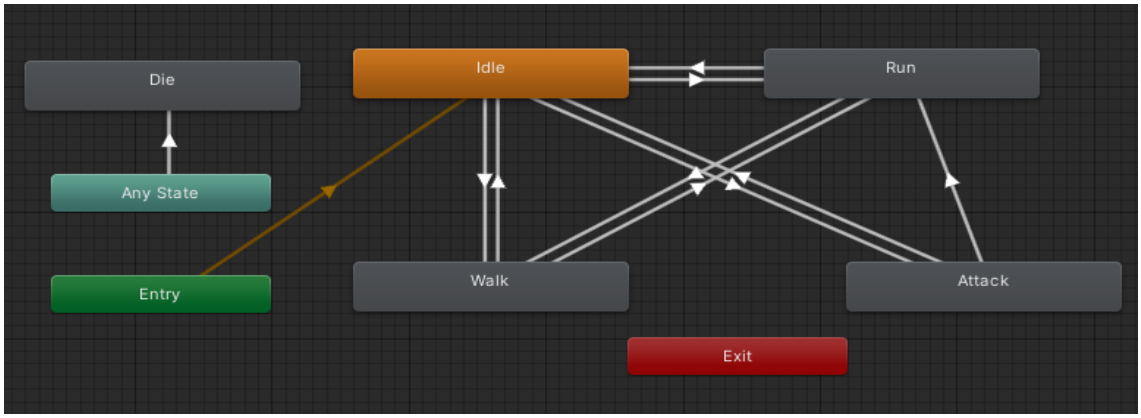
Ξεκινάμε τοποθετώντας το ζόμπι στην καρτέλα Hierarchy και του προσθέτουμε τρία νέα Components για να καθορίσουμε τον τρόπο κίνησης και αλληλεπίδρασής του με το υπόλοιπο περιβάλλον. Το πρώτο Component θα είναι ένα Character Controller με παρόμοιες επιμέρους ρυθμίσεις με αυτές που θέσαμε στον Player για την κίνηση του, το δεύτερο θα είναι ένα BoxCollider διαμορφώνοντας τις ρυθμίσεις Center και Size ώστε να καλύπτεται ολόκληρη η επιφάνεια του Ζόμπι για την επιτυχή ανίχνευση συγκρούσεων με αλλά GameObjects και τέλος ένα NavMeshAgent για την πλοήγηση του

στην σκηνή μας. Εδώ πρέπει να προσθέσουμε ότι για να επιτευχθεί σωστά η πλοήγηση ενός Agent σε μια οποιαδήποτε σκηνή θα πρέπει να «ορίσουμε» τον χώρο στον οποίο του δίνεται το δικαίωμα να κινηθεί.[59] Αυτό συμβαίνει μέσω της καρτέλας Navigation όπου επιλέγοντας το terrain, του προσδίδουμε την ιδιότητα Walkable και στην πορεία το «αποθηκεύουμε» πατώντας Bake.

3.7.1 Enemies Animator Controllers

Ακολουθώντας την πλέον γνωστή διαδικασία, θα κατασκευάσουμε ένα Animator Controller το οποίο θα συνδέσουμε με το Component Animator→Controller του ζόμπι μας.

Στο παράθυρο Animator , θα δημιουργήσουμε αυτήν την φορά πέντε διαφορετικά Empty States που θα μετονομάσουμε σε Idle, Walk, Run, Attack και Die συνδέοντας τα παράλληλα με το αντίστοιχο Animation για κάθε κατάσταση μέσω του πεδίου Motion. Επιπλέον , θέτουμε σαν Default State την κατάσταση Idle. Εδώ χρειάζεται να αναφέρουμε την χρήση της έτοιμης κατάστασης από το Unity, Any State, η οποία είναι μια κατάσταση που είναι πάντα παρούσα και υπάρχει για την περίπτωση οπού θέλουμε να μεταβούμε σε μια συγκεκριμένη κατάσταση, όπως για παράδειγμα το Die, ανεξάρτητα από την κατάσταση στην οποία βρισκόμαστε.[60]



Εικόνα 3 - 34: Zombie Animator Tab

Για την ορθή λειτουργία και εναρμόνιση των πέντε αυτών καταστάσεων, είναι απαραίτητο πέραν των transitions που εμφανίζονται στην Εικόνα 3-34 να δημιουργήσουμε και τέσσερις επιπλέον παραμέτρους. Οι δύο εξ αυτών θα είναι τύπου Boolean (walk, run) και οι δύο υπόλοιπες (attack, die) θα είναι τύπου trigger. Σκοπός ύπαρξης και των τεσσάρων παραμέτρων είναι η επιτυχημένη μετάβαση και επικοινωνία μεταξύ των διαφορετικών καταστάσεων και αυτό αντικατοπτρίζεται στον κώδικα που θα αναλυθεί παρακάτω.

3.7.2 Enemies Scripts

Στη συνέχεια, για να μπορέσουμε να επεξεργαστούμε κατάλληλα τα animations που μόλις συνθέσαμε αλλά και την συνολική λειτουργία των αντιπάλων μας, θα δημιουργήσουμε δυο ακόμη scripts.

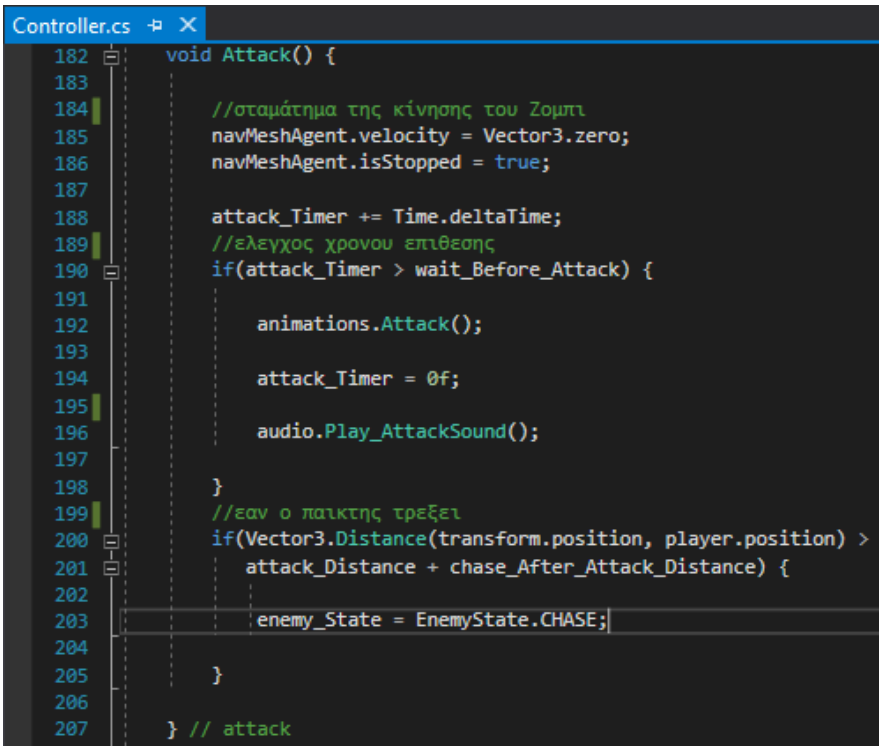
Το πρώτο script ονομάζεται Animations και θα προστεθεί ως νέο Component στο ζόμπι μας. Το script αυτό διαχειρίζεται την προβολή των διαφόρων καταστάσεων και όπως φαίνεται στην Εικόνα 3-35 χρησιμοποιεί τις τέσσερις παραμέτρους που κατασκευάσαμε για να καλέσει την αντίστοιχη μέθοδο κάθε φορά με την τιμή όπου εμείς θα εισάγουμε.

Animations.cs

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

public class Animations : MonoBehaviour
{
    private Animator animator;

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

    public void Walk(bool walk) {
        animator.SetBool("Walk", walk);
    }

    public void Run(bool run) {
        animator.SetBool("Run", run);
    }

    public void Attack() {
        animator.SetTrigger("Attack");
    }

    public void Dead() {
        animator.SetTrigger("Die");
    }

}
```

Εικόνα 3 - 35: Animations Script

Το δεύτερο script ονομάζεται Controller και θα προστεθεί και αυτό ως νέο Component στο ζόμπι μας. Το συγκεκριμένο script διαχειρίζεται πλήρως τις λειτουργίες κίνησης και αλληλεπίδρασης του ζόμπι με το περιβάλλον αλλά και με τον χαρακτήρα μας. Αρχικά, χρησιμοποιείται ξανά η έννοια του enum, αυτήν την φορά για να κατηγοριοποιήσει τις καταστάσεις ύπαρξης του αντιπάλου μας (Patrol, Chase, Attack). Οι διαθέσιμες αυτές καταστάσεις ελέγχονται στην μέθοδο Update( ) και αναλόγως την κατάσταση στην οποία βρισκόμαστε , καλείται η αντίστοιχη μέθοδος. Όπως παρουσιάζεται για παράδειγμα στην Εικόνα 3-36, η μέθοδος Attack( ) σταματάει την κίνηση του αντιπάλου που έχει ξεκινήσει στη μέθοδο Chase( ) και ελέγχει τον χρόνο επίθεσης για την ομαλή διαδοχή μεταξύ των χτυπημάτων. Επιπρόσθετα, αναπαράγει τόσο το κατάλληλο animation όσο και τον κατάλληλο ήχο έτσι ώστε να προκληθεί και η οπτικοακουστική αναπαράσταση του γεγονότος. Τέλος, προβλέπεται η περίπτωση που ο χαρακτήρας απομακρυνθεί από το ζόμπι σε απόσταση μεγαλύτερη του attack_Distance και έτσι επανερχόμαστε στην κατάσταση Chase( ). Ακολουθώντας την ίδια φιλοσοφία υλοποιούμε και τις υπολειπόμενες καταστάσεις-μεθόδους.



Εικόνα 3 - 36: Τμήμα Κώδικα από Controller Script

Κλείνοντας την υποενότητα αυτή, θα αναφερθούμε στον τρόπο ένταξης και επαναχρησιμοποίησης των αντιπάλων στη σκηνή μας. Για τον λόγο αυτό θα δημιουργήσουμε ένα τρίτο script που ονομάζεται EnemyManager.

Αρχικά, θα κατασκευάσουμε μέσω της καρτέλας Hierarchy δέκα GameObjects σε θέσεις τέτοιες ώστε να καλύπτουν όλο το φάσμα της πόλης και αυτά θα αναπαριστούν τα δέκα σημεία (SpawnPoints) που θα πραγματοποιείται η αναγέννηση των αντιπάλων. Όπως φαίνεται στην Εικόνα 3-37, μέσω της μεθόδου SpawnEnemies( ), χρησιμοποιούμε το Prefab του ζόμπι που έχουμε αναπτύξει και δημιουργούμε κλώνους του στα δέκα σημεία που μόλις ορίσαμε. Εν κατακλείδι, εκμεταλλευόμαστε την έννοια του Coroutine και οριοθετούμε τον ρυθμό επανένταξης των αντιπάλων στην σκηνή αλλά και προσφέρουμε έναν τρόπο αδιάκοπης αναπαραγωγής διπλοτύπων.

```
EnemyManager.cs
30      void Start()
31      {
32          initial_Zombie_Count = zombie_Enemy_Count;
33
34          SpawnEnemies();
35
36          StartCoroutine("CheckToSpawnEnemies");
37      }
38
39      void SpawnEnemies() {
40
41          int index = 0;
42          for (int i = 0; i < zombie_Enemy_Count; i++) {
43
44              Instantiate(zombie_Prefab, zombie_SpawnPoints[index].position, Quaternion.identity);
45              index++;
46          }
47
48          zombie_Enemy_Count = 0;
49
50      }
51
52      IEnumerator CheckToSpawnEnemies() {
53          yield return new WaitForSeconds(wait_Before_Spawn_Enemies_Time);
54
55          SpawnEnemies();
56
57          StartCoroutine("CheckToSpawnEnemies"); //αδιάκοπη coroutine
58
59      }
```

Εικόνα 3 - 37: Τμήμα Κώδικα από EnemyManager Script

3.8 Sounds

Μεγάλο κομμάτι στην ανάπτυξη και επιτυχία των Action Games είναι η ρεαλιστική αναπαράσταση των ηχητικών εφέ. Συγκεκριμένα, παίζει σημαντικό ρόλο στην συνολική εμπειρία και αλληλεπίδραση του χρήστη με το περιβάλλον του παιχνιδιού. Οι διάφοροι ήχοι που χρησιμοποιήθηκαν στο παιχνίδι αποκτήθηκαν μέσω του Unity Asset Store, που παρέχει μια ποικιλία για κάθε είδους περίπτωση.

Πρωτίστως, για την μουσική υπόκρουση που αναπαριστά το γενικό περιβάλλον της ερήμου, κατασκευάζουμε ένα GameObject με όνομα SoundEffects στο οποίο προσθέτουμε ένα νέο Component, το AudioSource. Στο πεδίο AudioClip αυτού, αντλούμε τον επιλεγμένο ήχο [61] και θέτουμε την επιλογή Play On Awake σε αληθή ώστε να γίνει αναπαραγωγή κατά την είσοδο του χρήστη στο παιχνίδι. Ακόμη, ενεργοποιούμε την επιλογή Loop μέσω της οποίας ορίζουμε την αδιάκοπη επανάληψη του συγκεκριμένου ήχου.

3.8.1 Player's Sounds

Οι ήχοι που θέλουμε να αναπαράγονται σχετικά με τον Player είναι οι βηματισμοί του καθώς κινείται στην σκηνή μας[55]. Ξεκινάμε δημιουργώντας ένα καινούργιο GameObject ως παιδί του, που ονομάζουμε Audio και παράλληλα προσθέτουμε σε αυτό ένα νέο Component τύπου AudioSource και ένα νέο script Footsteps.

```
Footsteps.cs
26      void Update ()
27      {
28          CheckToPlayFootstepSound();
29      }
30
31      void CheckToPlayFootstepSound() {
32
33          // Εαν δεν ειμαστε στο εδαφος
34          if (!character_Controller.isGrounded)
35              return;
36
37          if(character_Controller.velocity.sqrMagnitude > 0) { //Εαν κινούμαστε
38
39              // accumulated distance ειναι η τιμη του ποσο μακρια μπορει να παει ο παικτης μεχρι να αναπαραχθει ο ηχος
40              accumulated_Distance += Time.deltaTime;
41
42              if(accumulated_Distance > step_Distance) {
43                  footstep_Sound.volume = Random.Range(volume_Min, volume_Max);
44                  footstep_Sound.clip = footstep_Clip[Random.Range(0, footstep_Clip.Length)];
45                  footstep_Sound.Play();
46                  accumulated_Distance = 0f;
47              }
48          } else {
49              accumulated_Distance = 0f; //επαναφορά τις τιμής για να μην παιχτει την επομενη φορα απευθείας ο ηχος!!
50          }
51
52      }
53
54  }
55
```

Εικόνα 3 - 38: Τμήμα Κώδικα από Footsteps Script

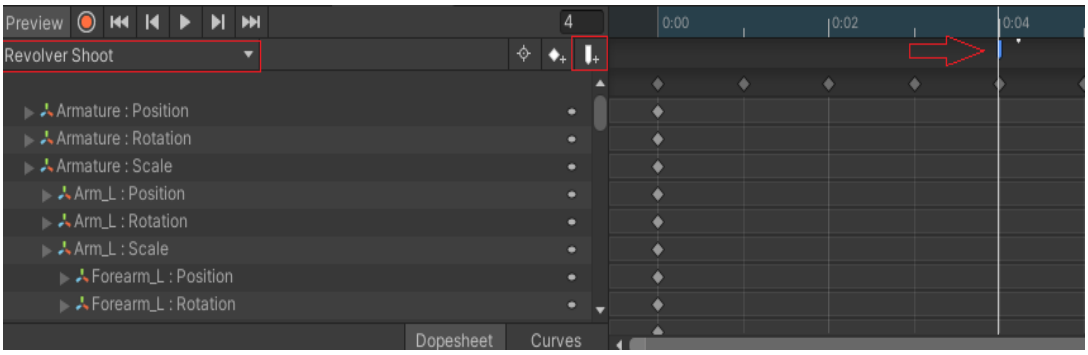
Όπως παρατηρούμε στην Εικόνα 3-38, μέσω της μεθόδου CheckToPlayFootstepSound( ) ελέγχουμε εάν ο παίκτης βρίσκεται στο έδαφος, δηλαδή δεν έχει κάνει χρήση του Space=Jump και επίσης εάν αυτός κινείται. Μόνο αν ικανοποιούνται και οι δύο συνθήκες και ταυτόχρονα η απόσταση που έχει καλυφθεί είναι μεγαλύτερη της απαιτούμενης step_Distance (ορίζεται στο DifferentTypesOfMovement script και είναι διαφορετική για κάθε είδος κίνησης) θα γίνει αναπαραγωγή του ήχου.

Επιστρέφοντας στο DifferentTypesOfMovement script και συγκεκριμένα στην Εικόνα 3-24, μπορούμε να καταλάβουμε πλέον πως κάθε είδος ξεχωριστής κίνησης έχει πέραν της εφαρμογής διαφορετικής ταχύτητας για τον παίκτη, και διαφορετικό ρυθμό αναπαραγωγής αλλά και έντασης του ήχου.

3.8.2 Weapons' Sounds

Στο σημείο αυτό, θα προσθέσουμε ένα AudioSource component σε κάθε διαθέσιμο όπλο, εκτός από την περίπτωση του Shotgun που θα έχει αντίστοιχα δύο. Τα component αυτά αναπαριστούν τον πυροβολισμό των όπλων ενώ το δεύτερο του Shotgun προσομοιώνει το γέμισμα των σφαιρών αυτού [55]. Αυτό όμως δεν είναι αρκετό, θα πρέπει να συνδέσουμε τα συγκεκριμένα AudioSources με τα κατάλληλα πεδία του script Organizer για να πραγματοποιηθεί επιτυχώς η αναπαραγωγή των ήχων.

Όπως παρουσιάζεται στην Εικόνα 3-39, από το παράθυρο Animation επιλέγω κάθε φορά το επιθυμητό όπλο και το Shoot Animation αυτού και προσθέτω ένα νέο Animation Event στο ιδανικό frame και συγκεκριμένα για την στιγμή που ο χρήστης θα επιτεθεί. Στη συνέχεια, μέσω της καρτέλας Inspector μπορούμε να διαλέξουμε για το κάθε Animation Event που συνθέσαμε την μέθοδο που θέλουμε να κληθεί.



Εικόνα 3 - 39: Προσθήκη AnimationEvent για ShootSound

Για την προσομοίωση των διαφορετικών ήχων που μπορούν να προκληθούν από ένα τσεκούρι, δημιουργούμε ένα βοηθητικό script με όνομα AxeSound που προστίθεται σαν Component σε αυτό. Το script αυτό, όπως φαίνεται και στην Εικόνα 3-40, επιλέγει τυχαία ανάμεσα από τρείς διαφορετικούς ήχους με μεταχείριση της ενσωματωμένης συνάρτησης Random.Range, τους οποίους και αναπαράγει κατά την χρησιμοποίηση του από τον χρήστη.

AxeSound.cs

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

public class AxeSound : MonoBehaviour
{

    [SerializeField]
    private AudioSource audioSource;

    [SerializeField]
    private AudioClip[] woosh_Sounds;

    void PlayWooshSound() {
        audioSource.clip = woosh_Sounds[Random.Range(0, woosh_Sounds.Length)];
        audioSource.Play();
    }
}

```

Εικόνα 3 - 40: AxeSound script

3.8.3 Enemies' Sounds

Βασιζόμενοι στην ίδια φιλοσοφία θα υλοποιήσουμε τα ηχητικά εφέ για τους αντιπάλους. Επιλέγουμε το ζόμπι και δημιουργούμε ένα καινούργιο GameObject ως παιδί του με όνομα Zombie Sound. Σε αυτό θα προσθέσουμε ένα AudioSource component αλλά και ένα επιπλέον script με όνομα Audio.

Όπως φαίνεται στην Εικόνα 3-41, με το συγκεκριμένο script απλώς συνδέουμε τα διάφορα ηχητικά κλιπ [55] με τις κατάλληλες μεθόδους. Η αναπαραγωγή και κλήση τους όμως θα προκληθεί από το προαναφερθέν Controller script στα κατάλληλα σημεία κάθε φορά. Για παράδειγμα, ξανακοιτώντας την Εικόνα 3-36, παρατηρούμε πως πέραν όλων των υπολοίπων, η γραμμή κώδικα 196 εκμεταλλεύεται την αναφορά στο script Audio και προκαλεί την ενεργοποίηση της μεθόδου Play_AttackSound( ).

Audio.cs

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

public class Audio : MonoBehaviour
{
    private AudioSource audioSource;

    [SerializeField]
    private AudioClip scream_Clip, die_Clip;
    [SerializeField]
    private AudioClip[] attack_Clips;

    void Awake ()
    {
        audioSource = GetComponent<AudioSource>();
    }

    public void Play_AttackSound() {
        audioSource.clip = attack_Clips[Random.Range(0, attack_Clips.Length)];
        audioSource.Play();
    }

    public void Play_ScreamSound() {
        audioSource.clip = scream_Clip;
        audioSource.Play();
    }

    public void Play_DeadSound() {
        audioSource.clip = die_Clip;
        audioSource.Play();
    }

}
```

Εικόνα 3 - 41: Audio script

3.9 Damage Detection

Έχοντας ολοκληρώσει με την δημιουργία του χαρακτήρα μας, των όπλων αλλά και των αντιπάλων αυτού θα αναλύσουμε τον τρόπο αλληλεπίδρασης τους. Για τον λόγο αυτό πρέπει να εξηγήσουμε την διαδικασία με την οποία ανιχνεύεται η σύγκρουση μεταξύ αυτών των GameObjects.

Αρχικά, επιλέγοντας το τσεκούρι κατασκευάζουμε ένα καινούργιο GameObject ως παιδί του με όνομα AttackPoint. Ορίζουμε το transform του μέσω του παραθύρου Inspector έτσι ώστε να βρίσκεται στο άκρο του τσεκουριού. Στο σημείο αυτό, θα δημιουργήσουμε ένα νέο script Attack, που θα είναι υπεύθυνο για την ανίχνευση της ζημιάς και θα προστεθεί σαν νέο Component στο AttackPoint μας.

Όπως φαίνεται στην Εικόνα 3-42, χρησιμοποιούμε την ενσωματωμένη μέθοδο του Unity, Physics.OverlapSphere για να ανιχνεύσουμε την σύγκρουση του AttackPoint που δημιουργήσαμε με όλα τα GameObjects που βρίσκονται σε layerMask ίσο με Enemy.

```
Attack.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Attack : MonoBehaviour
6  {
7      public float damage = 2f;
8      public float range = 1f;
9      public LayerMask layerMask;
10
11     void Update ()
12     {
13         Collider[] strikes = Physics.OverlapSphere(transform.position, range, layerMask);
14             if(strikes.Length > 0) {
15                 strikes[0].gameObject.GetComponent<Health>().ApplyDamage(damage);
16                 gameObject.SetActive(false); // ανιχνευση ζημιας μονο μια φορα και απενεργοποιηση του attackPoint.
17             }
18     }
19 }
```

Εικόνα 3 - 42: Attack script

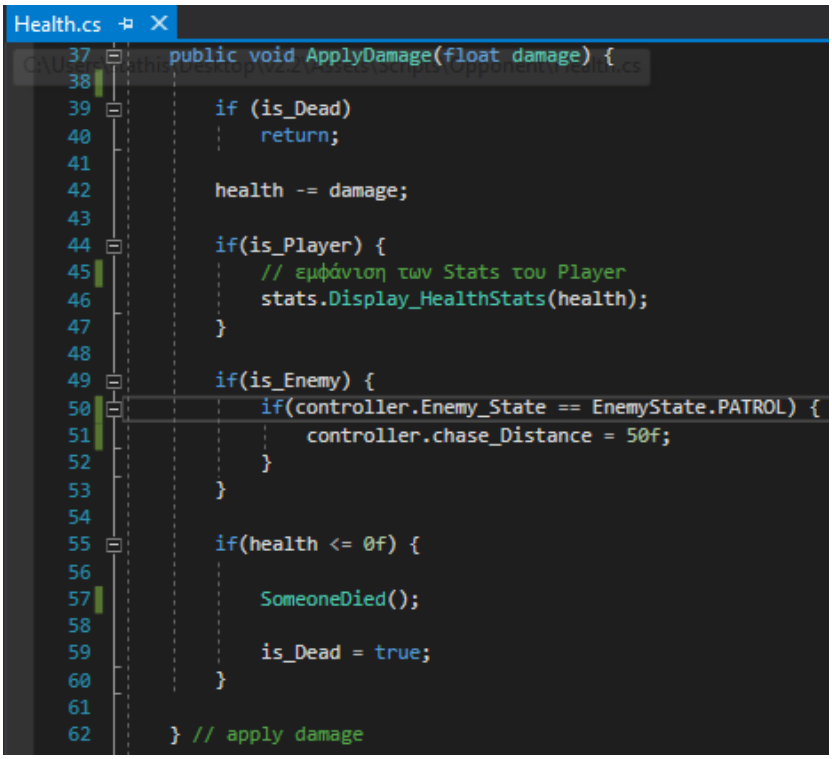
Εν συνεχεία, θα πρέπει να συνδέσουμε το AttackPoint με το αντίστοιχο πεδίο στο Organizer script και να προσθέσουμε τις κατάλληλες μεθόδους ενεργοποίησης και απενεργοποίησης του. Ακολουθώντας την ίδια προσέγγιση με παραπάνω (Εικόνα 3-37), επιλέγουμε το AttackAnimation και προσθέτουμε στο επιθυμητό frame ενεργοποίησης την μέθοδο TurnOnAttackPoint( ) και αντίστοιχα στο frame απενεργοποίησης την μέθοδο TurnOffAttackPoint( ).

Για την ανίχνευση της ζημιάς από την πλευρά του ζόμπι ως προς τον χαρακτήρα μας θα ακολουθήσουμε την ιδία σειρά ενεργειών. Θα επαναχρησιμοποιήσουμε το Attack script που δημιουργήσαμε με μοναδική διαφορά την τιμή του layerMask η οποία θα τεθεί ίση με Player.

3.10 Health-Applying Damage

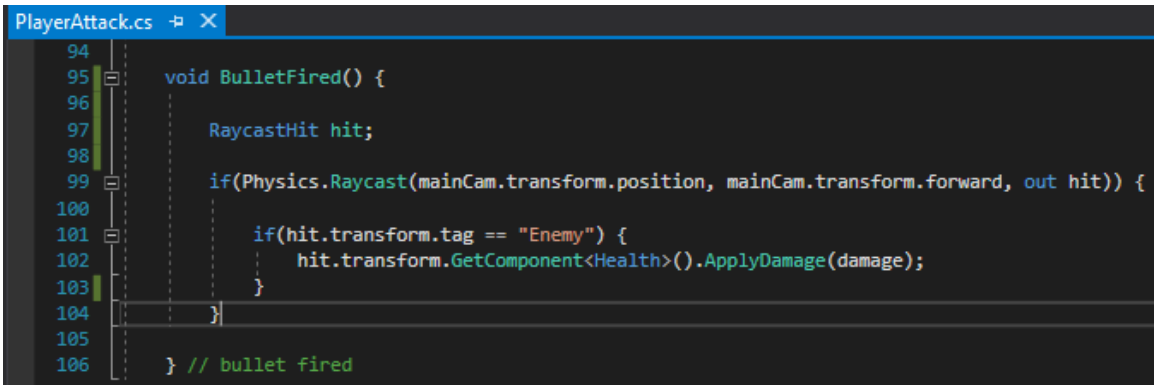
Μετά την επιτυχημένη ανίχνευση ζημιάς, στόχος μας είναι να εφαρμοστεί κατάλληλα και στα θεμιτά GameObjects. Για τον λόγο αυτό, θα δημιουργήσουμε ένα επαναχρησιμοποιήσιμο script που θα προσαρτηθεί και στον Player αλλά και στους αντίπαλους. Η διαφοροποίηση αυτή θα γίνει μέσω μιας Boolean τιμής (is_Player / is_Enemy) η οποία θα αληθεύει κάθε φορά για το αντίστοιχο GameObject.

Όπως απεικονίζεται στην Εικόνα 3-43, ανεξάρτητα από την παραπάνω διάκριση του script, αφαιρείται πάντα από την προκαθορισμένη τιμή του health=100f η ζημιά που δέχεται σε κάθε περίπτωση το επικείμενο GameObject. Στο ενδεχόμενο που η τιμή health μηδενιστεί ή αγγίξει αρνητικές τιμές, μεταβαίνουμε στην μέθοδο SomeoneDied( ).



Εικόνα 3 - 43: Τμήμα Κώδικα από Health script

Επιστρέφοντας στην Εικόνα 3-42, παρατηρούμε πως η γραμμή κώδικα 15 καλεί μέσω του script Health την μέθοδο ApplyDamage(damage) που μόλις εξηγήσαμε. Με τον ίδιο τρόπο , η μέθοδος BulletFired( ) που αναφέρθηκε στην Ενότητα 3.6.3, χρησιμοποιεί την προκαθορισμένη μέθοδο του Unity, Physics.Raycast για να πυροβολήσει από την θέση που βρίσκεται ο χαρακτήρας και προς την κατεύθυνση που στοχεύει αποθηκεύοντας στην μεταβλητή hit το GameObject που θα έρθει σε επαφή. Έτσι, στην Εικόνα 3-44, η γραμμή κώδικα 102 , καλεί και πάλι μέσω του script Health την προαναφερθείσα μέθοδο.



Εικόνα 3 - 44: Τμήμα Κώδικα από PlayerAttack script

Health.cs

```
64    void SomeoneDied() {
65
66        if(is_Enemy) {
67
68            ScoreManager.instance.AddPoint();
69            GameOverMenu.instance.AddPoint2();
70            navMeshAgent.velocity = Vector3.zero;
71            navMeshAgent.isStopped = true;
72            controller.enabled = false;
73
74            animations.Dead();
75
76            StartCoroutine(DeadSound()); // για να καθυστερήσει ελαχιστα η εκτέλεση
77
78            // αναγέννησε κι αλλα ζομπι
79            EnemyManager.instance.EnemyDied();
80        }
81
82        if(is_Player) {
83
84            GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
85
86            for (int i = 0; i < enemies.Length; i++) {
87                enemies[i].GetComponent<Controller>().enabled = false;
88            }
89
90            // σταματημα αναγεννησης
91            EnemyManager.instance.StopSpawning();
92
93            GetComponent<Movement>().enabled = false;
94            GetComponent<PlayerAttack>().enabled = false;
95            GetComponent<Director>().GetCurrentSelectedWeapon().gameObject.SetActive(false);
96
97        }
98
99        if(tag == "Player") {
100
101           Invoke("ShowGameOverMenu", 3f);
102
103       } else {
104
105           Invoke("TurnOffGameObject", 3f);
106       }
107   } // SomeoneDied
```

Εικόνα 3 - 45: Τμήμα Κώδικα από Health script

Στην Εικόνα 3-45 απεικονίζεται η μέθοδος SomeoneDied( ), η οποία χωρίζει την εκτέλεση των ενεργειών σε δυο περιπτώσεις. Στην περίπτωση που το GameObject είναι αντίπαλος, θα απενεργοποιήσουμε ορισμένες ενέργειες του, θα αναπαράγουμε το αντίστοιχο Animation αλλά και θα εκμεταλλευτούμε το script EnemyManager για να αναγεννηθούν περισσότεροι εχθροί. Στην αντίθετη περίπτωση, θα σταματήσουμε την λειτουργία και αναγέννηση των εχθρών και παράλληλα την κίνηση και την επίθεση του χαρακτήρα μας.

3.11 User Interface

Για τη γραφική απεικόνιση της ζωής και αντοχής του χαρακτήρα μας θα δημιουργήσουμε ένα καινούργιο Canvas στο παράθυρο Hierarchy. Στον Canvas αυτό, θα προσθέσουμε μια εικόνα που θα εξυπηρετεί τον σκοπό του φόντου και ακόμη δυο επιμέρους εικόνες ως παιδιά της πρώτης, οι οποίες θα αναπαριστούν το σύμβολο της ζωής και την μπάρα κύλισης που αντιστοιχεί στο ποσοστό της. Στις προστιθέμενες εικόνες θα ρυθμίσουμε κατάλληλα την πηγή εικόνας τους [55],[62], τη θέση, το μέγεθος και το χρώμα τους. Επιπλέον, στην μπάρα κύλισης θα μεταβάλλουμε το πεδίο Image Type και Fill Method σε Filled και Horizontal αντίστοιχα. Την ίδια διαδικασία θα ακολουθήσουμε και για την υλοποίηση του ποσοστού αντοχής (Stamina).

Για να προσδώσουμε λειτουργικότητα στα παραπάνω, θα συνθέσουμε ένα script με όνομα Stats το οποίο θα προσαρτηθεί στον Player. Όπως φαίνεται στην Εικόνα 3-46, θα πρέπει να μετατρέψουμε το ποσοστό ζωής στην δεκαδική μορφή του και να το προωθήσουμε στην μπάρα κύλισης μέσω του πεδίου Fill Amount.

Stats.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class Stats : MonoBehaviour
7  {
8
9      [SerializeField]
10     private Image health_Stats, stamina_Stats;
11
12     public void Display_HealthStats(float healthValue) {
13
14         healthValue /= 100f;
15         health_Stats.fillAmount = healthValue;
16
17     }
18
19     public void Display_StaminaStats(float staminaValue) {
20
21         staminaValue /= 100f;
22         stamina_Stats.fillAmount = staminaValue;
23
24     }
25 }
```

Εικόνα 3 - 46: Stats script

Επιστρέφοντας στην Εικόνα 3-43, παρατηρούμε πως η γραμμή κώδικα 46 καλεί μέσω του script Stats την μέθοδο Display_HealthStats(float healthValue) για την αναπαράσταση της ζημιάς που έχει δεχτεί ο χαρακτήρας μας. Ανατρέχοντας ακόμη πιο πίσω , στην Ενότητα 3.5.3, θα χρειαστεί να προσθέσουμε τους απαραίτητους ελέγχους στο DifferentTypesOfMovement script για την χρησιμοποίηση και την κατανάλωση της αντοχής. Όπως παρατηρούμε στην Εικόνα 3-47, ελέγχουμε εάν η τιμή του stamina είναι μεγαλύτερη του μηδενός και ταυτόχρονα αν ο χαρακτήρας βρίσκεται σε κίνηση, τότε μειώνουμε κατάλληλα την τιμή της. Στην αντίθετη περίπτωση θα αυξήσουμε την τιμή του stamina με ρυθμό μικρότερο από το προηγούμενο ενδεχόμενο. Οι αυξομειώσεις αυτές παρουσιάζονται στον χρήστη με χρησιμοποίηση της μεθόδου Display_StaminaStats στις γραμμές κώδικα 81 κι 89.

```
DifferentTypesOfMovement.cs*
62    void Sprint() {
63
64        //εαν εχουμε stamina μπορουμε να κανουμε sprint
65        if(stamina > 0f) {
66                            //καταναλωση του stamina μονο αν ο χαρακτήρας κινειται ΚΑΙ ταυτοχρονα κραταμε πατημενο το Shift
67        if(Input.GetKey(KeyCode.LeftShift) && !is_Crouching && !is_Proning && character_Controller.velocity.sqrMagnitude > 0) {
68
69            stamina -= sprint_Treshold * Time.deltaTime;
70
71            if(stamina <= 0f) {
72
73                stamina = 0f;
74
75                // επαναφορα ηχου και ταχυτητας
76                movement.speed = move_Speed;
77                footsteps.step_Distance = walk_Step_Distance;
78                footsteps.volume_Min = walk_Volume_Min;
79                footsteps.volume_Max = walk_Volume_Max;
80            }
81            stats.Display_StaminaStats(stamina);
82
83        } else {
84
85            if(stamina != 100f) {
86
87                stamina += (sprint_Treshold / 2f) * Time.deltaTime;
88
89                stats.Display_StaminaStats(stamina);
90
91                if(stamina > 100f) {
92                    stamina = 100f;
93                }
94            }
95        }
96    } // sprint
```

Εικόνα 3 - 47: Τμήμα Κώδικα από DifferentTypesOfMovement script

3.12 Extras

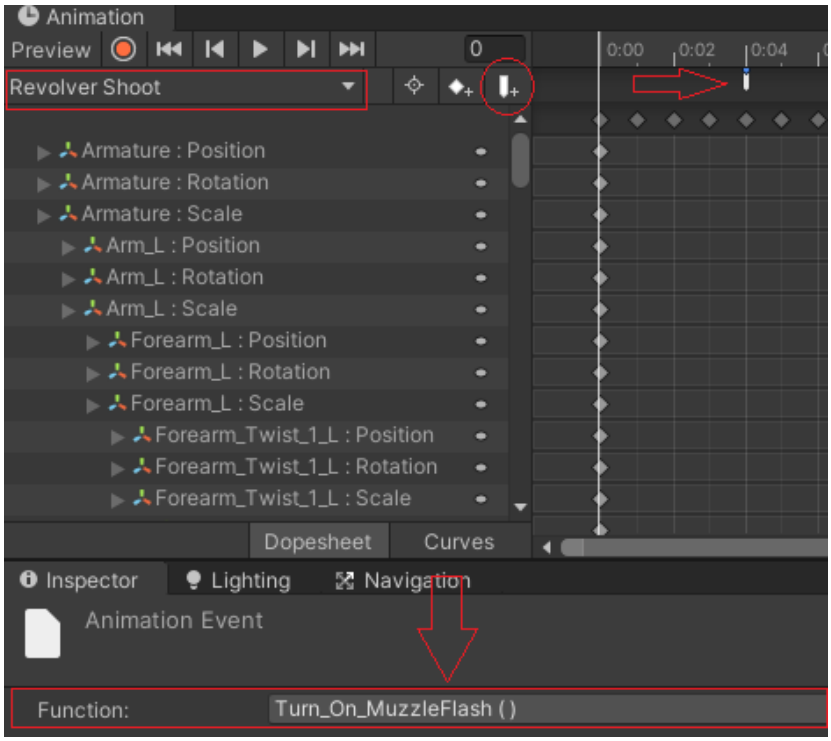
Στην τελευταία ενότητα, θα αναφερθούμε στην χρήση των Particle Effects του παιχνιδιού μας αλλά και στον τρόπο μέτρησης και αποθήκευσης των πόντων του χρήστη.

Τα Particle Effects που χρησιμοποιούνται για την αναπαράσταση των εφέ όσον αφορά το περιβάλλον της ερήμου προέρχονται από τους δημιουργούς του Terrain [49] και είναι τα εξής :

- DustStorm για την προσομοίωση της αμμοθύελλας,
- WaterProDaytime για την ρεαλιστική προβολή του νερού,
- Smoke, Fire, Flames, Glow, SmokeDark, SmokeLit, SparksFalling, SparksRising για την απεικόνιση του κατεστραμμένου – φλεγόμενου βαρελιού.

Τα Particle Effects των όπλων αφορούν την προσομοίωση του φωτός από την εκπυρσοκρότηση τους (Revolver, Shotgun) υλοποιήθηκαν ως εξής :

Χρησιμοποιούμε το έτοιμο Particle Effect [55] για το MuzzleFlash και το προσαρμόζουμε στο κατάλληλο σημείο για κάθε όπλο που θα εφαρμοστεί. Εν συνεχεία, θα πρέπει να συνδέσουμε τα συγκεκριμένα GameObjects με τα κατάλληλα πεδία του script Organizer καθώς και να δημιουργήσουμε τις αντίστοιχες μεθόδους για να πραγματοποιηθεί επιτυχώς η εμφάνιση του εφέ. Όπως φαίνεται στην Εικόνα 3-48, επιλέγουμε το ShootAnimation για κάθε όπλο και προσθέτουμε στο επιθυμητό frame ενεργοποίησης την μέθοδο TurnOnMuzzleFlash( ) και αντίστοιχα στο frame απενεργοποίησης την μέθοδο TurnOffMuzzleFlash ( ).



Εικόνα 3 - 48: Προσθήκη AnimationEvent για MuzzleFlash

Για την επιτυχημένη καταμέτρηση των πόντων που συγκεντρώνει ο χρήστης σκοτώνοντας τους εχθρούς του θα δημιουργήσουμε δυο καινούργια Text στον ήδη υπάρχον Canvas μας. Το ένα θα αντιπροσωπεύει το τρέχον σκορ του χρήστη (score) και το δεύτερο το χρονικά υψηλότερο σκορ του (highscore). Έτσι θα κατασκευάσουμε ένα νέο script με όνομα ScoreManager το όποιο θα προστεθεί σαν καινούργιο component στον Canvas μας.

Όπως παρατηρούμε στην Εικόνα 3-49, θα εκμεταλλευτούμε την έννοια του Instance και παράλληλα θα δημιουργήσουμε την μέθοδο AddPoint( ) η οποία θα καλείται κάθε φορά που πεθαίνει ένα ζόμπι για να προσθέτει έναν πόντο στην βαθμολογία του χαρακτήρα μας (Εικόνα 3-45, γραμμή κώδικα 68).

Τέλος , σημαντική είναι η ύπαρξη και χρησιμοποίηση της ενσωματωμένης κλάσης Player.Prefs για την επιτυχημένη αποθήκευση της υψηλότερης ιστορικά επίδοσης του χρήστη.

ScoreManager.cs

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

public class ScoreManager : MonoBehaviour
{
    public static ScoreManager instance ;

    public Text scoreText;
    public Text highscoreText;

    int score = 0;
    int highscore = 0;

    void Awake()
    {
        instance = this ;
    }

    void Start()
    {
        highscore = PlayerPrefs.GetInt("highscore", 0);
        scoreText.text = "SCORE:" + score.ToString() ;
        highscoreText.text = "HIGHSCORE:" + highscore.ToString();
    }

    public void AddPoint(){

        score += 1;
        scoreText.text = "SCORE:" + score.ToString() ;

        if (highscore < score )
            PlayerPrefs.SetInt("highscore", score);
    }
}
```

Εικόνα 3 - 49: ScoreManager script

3.13 Επίλογος

Στο κεφάλαιο αυτό, αναλύσαμε λεπτομερώς την σειρά ενεργειών που ακολουθήσαμε για την πλήρη υλοποίηση του παιχνιδιού μας. Αρχικά, παρουσιάσαμε την οπτική αναπαράσταση για τα κύρια και δευτερεύοντα μενού, την δημιουργία του χαρακτήρα, των εχθρών και των όπλων. Στην συνέχεια, προσαρτήσαμε τα ηχητικά εφέ και προσδώσαμε λειτουργικότητα στα παραπάνω μέσω κωδικοποίησης. Στο προσεχές κεφάλαιο θα προβληθεί το τελικό Έγγραφο Σχεδίασης Παιχνιδιού που προέκυψε με την περάτωση του Project.

Κεφάλαιο 4ο: Έγγραφο Σχεδίασης Παιχνιδιού

4.1 Εισαγωγή

"Το έγγραφο σχεδίασης παιχνιδιών (GDD) είναι το προσχέδιο από το οποίο ένα βιντεοπαιχνίδι πρόκειται να χτιστεί."[63]. Το GDD είναι ένα "έγγραφο αναφοράς, το οποίο τα μέλη της ομάδας ανάπτυξης πρέπει να συμβουλεύονται συχνά"[63], που δημιουργείται κατά το στάδιο του σχεδιασμού του παιχνιδιού, πολύ νωρίτερα δηλαδή από την πραγματική παραγωγή του. Το έγγραφο αυτό περιέχει όλες τις πληροφορίες για το παιχνίδι που επρόκειτο να δημιουργηθεί από την αρχική-πρώιμη σχεδίαση του μέχρι και το τελικό αποτέλεσμα. Ένα σωστά δομημένο GDD μπορεί να παρέχει υποστήριξη στην μετάβαση μεταξύ του σταδίου της προ-παραγωγής και του σταδίου παραγωγής παίζοντας έτσι πολύ σημαντικό ρόλο σε ολόκληρη την διαδικασία ανάπτυξης παιχνιδιού[65].

Ωστόσο, επειδή αποτελεί ένα μεγάλο σε έκταση έγγραφο που προβάλει λεπτομερώς όλα τα βασικά στοιχεία ενός παιχνιδιού, αποδεικνύεται ότι χρησιμοποιείται ελάχιστα, ειδικά όταν το ζητούμενο είναι ο σχηματισμός και η επεξεργασία μικρών παιχνιδιών. Λόγω αυτού, υπάρχει το SGDD (ShortGameDesignDocument), το οποίο είναι ένας τύπος εγγράφου που απεικονίζει τα συστατικά του παιχνιδιού με πιο απλοποιημένο τρόπο[64]. Μια τέτοια μορφή εγγράφου θα χρησιμοποιήσουμε και εμείς στα πλαίσια της εργασίας μας.

Ανεξαρτήτως όμως του μεγέθους και της μορφής του έγγραφου σχεδίασης παιχνιδιού αυτό θα πρέπει σίγουρα να περιέχει [65-66]:

- Περίληψη των κυριότερων σημείων (τίτλος, είδος παιχνιδιού, γενικά χαρακτηριστικά)
- Gameplay (στόχοι, χειρισμός παίκτη)
- Βασικά στοιχεία παιχνιδιού (ιστορία, χαρακτήρες, εχθροί)
- Assets (μουσική, 2D/3D models)

4.2 Περιεχόμενα Εγγράφου Σχεδίασης

- Περίληψη Παιχνιδιού
- Σύντομη Περιγραφή Παιχνιδιού
- Gameplay
- Βασικοί Χαρακτήρες
- Menu Screens / Διάγραμμα Ροής
- Μουσική / Ήχοι
- 2D/3D Models

4.2.1 Περίληψη Παιχνιδιού

Τίτλος: Polluted Desert

Ονόματα Ομάδας Ανάπτυξης Παιχνιδιού: Ιορδανίδης Ιορδάνης – Ιωαννίδης Ευστάθιος

Είδος: 3D First Person Shooter – Survival Game

Τεχνική Περιγραφή: Windows PC

Ελάχιστες απαιτήσεις:

Λειτουργικό Σύστημα: Windows 7+

Κάρτα Γραφικών: Direct X11 - NVDIA GeForce GT730

Ενδιαφερόμενοι Παίκτες: Άτομα με ηλικία 16+

Ημερομηνία Κυκλοφορίας: Ιούνιος 2021

4.2.2 Σύντομη Περιγραφή Παιχνιδιού

Το παιχνίδι Polluted Desert είναι ένα τρισδιάστατο παιχνίδι Shooting. Ο χαρακτήρας αρχικά βρίσκεται στην είσοδο μίας απομονωμένης πόλης στην μέση της ερήμου που έχει κατακτηθεί από υπερφυσικά όντα, τα οποία συνεχώς του επιτίθενται. Στην κατοχή του διαθέτει μονάχα τέσσερα όπλα και σκοπός του είναι να επιβιώσει όσο το δυνατόν περισσότερο.

4.2.3 Gameplay

Χειρισμός:

I. Πληκτρολόγιο

- W-S-A-D → Βασική κίνηση χαρακτήρα προς όλες τις κατευθύνσεις (W - Μπροστά, S - Πίσω, A - Αριστερά, D - Δεξιά)
- C → Crouch - "Σκύψιμο" χαρακτήρα
- Z → Prone - "Ξάπλωμα" χαρακτήρα
- 1-2-3-4 → Εναλλαγή διαθέσιμου οπλισμού
- Shift → Sprint - Γρήγορη κίνηση χαρακτήρα
- Esc → Παύση παιχνιδιού - Εμφάνιση Pause Menu
- Tab → Εμφάνιση κέρσορα

II. Ποντίκι

- Mouse → Αλλαγή οπτικής θέσης της κάμερας και περιστροφή της προβολής του χαρακτήρα του παίκτη
- Left Mouse Click → Χρήση τρέχοντος όπλου – εκπυρσοκρότηση
- Right Mouse Click → Zoom In – Zoom Out – Εστίαση όπλου

III. Κάμερα

Η κάμερα, όπως και σε όλα τα FPS παιχνίδια, είναι εγκατεστημένη πάνω στον χαρακτήρα και προβάλει αυτό που θα έβλεπε με τα μάτια του.

Στόχος:

Βασικός στόχος του χρήστη είναι να καταφέρει να επιβιώσει όσο το δυνατόν περισσότερο, σκοτώνοντας παράλληλα όσο περισσότερους εχθρούς μπορέσει. Το άθροισμα αυτών αποτελεί το τρέχον σκορ του παίκτη, επιδιώκοντας ακόμη να ξεπεράσει την υψηλότερη χρονικά βαθμολογία που έχει επιτύχει ο ίδιος.

4.2.4 Βασικοί Χαρακτήρες

I. Χαρακτήρας Παίκτη

Αποτελεί τον κύριο χαρακτήρα του παιχνιδιού και συγκεκριμένα είναι αυτός που διαχειρίζεται ο χρήστης για την επιτυχημένη αλληλεπίδραση του με το περιβάλλον. Του δίνεται η δυνατότητα κίνησης στις τρείς διαστάσεις σε όλη την έκταση της σκηνής καθώς και η δυνατότητα εναλλαγής και χρήσης του διαθέσιμου οπλισμού. Ακολουθώντας τον κανόνα των FPS παιχνιδιών, προβάλλεται ο χαρακτήρας με την απεικόνιση των χεριών του σε συνδυασμό με το επιλεγμένο όπλο κάθε χρονική στιγμή.

II. Εχθροί

Οι εχθρικές μονάδες του παιχνιδιού είναι υπερφυσικά όντα (Zombie). Κινούνται και αυτά στις τρείς διαστάσεις της σκηνής, αλλά η ταχύτητα τους είναι αρκετά μικρότερη από αυτήν του παίκτη. Περιπολούν σε διαφορετικές εκτάσεις του περιβάλλοντος και σε περίπτωση αντίληψης του παίκτη σε κοντινή απόσταση, του επιτίθενται. Για να προκαλέσουν όμως ζημιά στον χαρακτήρα χρειάζεται να τον πλησιάσουν αρκετά αφού η επίθεση τους πραγματοποιείται με τα χέρια. Σε περίπτωση εξόντωσης τους, επανέρχονται αδιάκοπα στην σκηνή σε προκαθορισμένα σημεία.

4.2.5 Menu Screens / Διάγραμμα Ροής

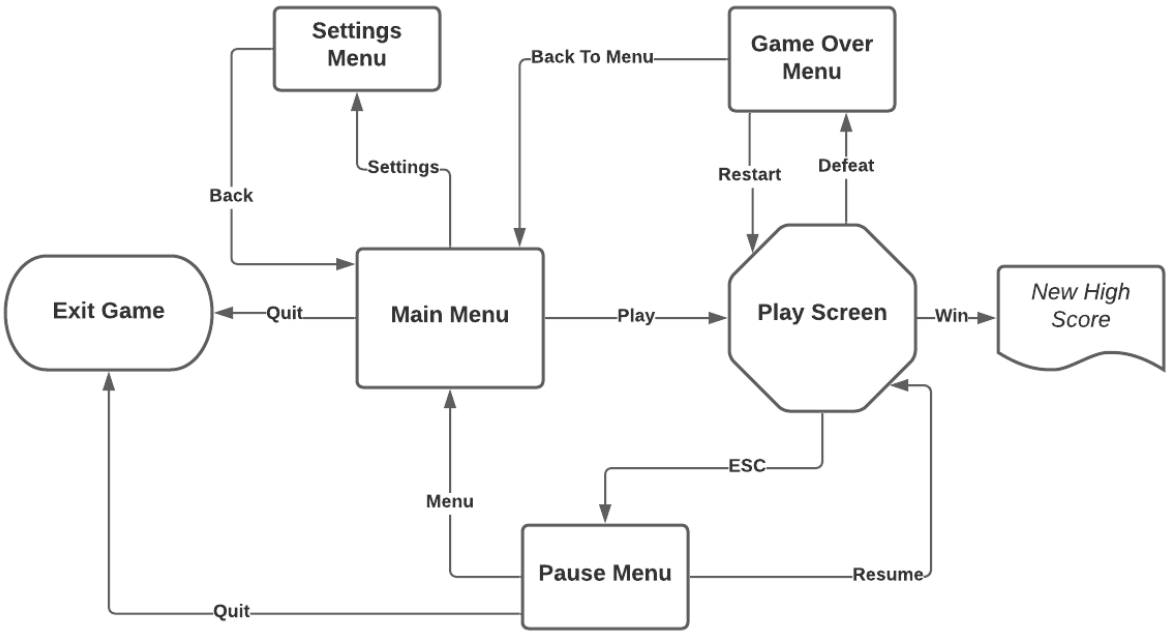
Η πρώτη εικόνα που εμφανίζεται στον χρήση κατά την εκκίνηση του παιχνιδιού είναι αυτή του Main Menu. Ο χρήστης έχει στην διάθεση του τρεις επιλογές, εκ των οποίων η πρώτη ξεκινάει το παιχνίδι, η δεύτερη τον ανακατευθύνει στο μενού των ρυθμίσεων, όπου δίνεται η δυνατότητα ρύθμισης του ήχου και η τρίτη πραγματοποιεί έξοδο από το παιχνίδι.

Σε οποιαδήποτε στιγμή του χρόνου εκτέλεσης του παιχνιδιού, πατώντας το πλήκτρο ESC, εμφανίζεται στο προσκήνιο το Pause Menu. Ο χρήστης έχει στην διάθεση του τρείς επιλογές εκ των οποίων η πρώτη συνεχίζει το παιχνίδι από την χρονική στιγμή που αυτό διακόπηκε, η δεύτερη τον ανακατευθύνει στο Main Menu και η Τρίτη τερματίζει το παιχνίδι.

Στην περίπτωση που οι εχθροί επικρατήσουν και η ζωή του χρήστη μηδενιστεί, το τρέχον παιχνίδι ολοκληρώνεται και εμφανίζεται το Game Over Menu στο οποίο προβάλλεται και το τελικό σκορ του. Επιπλέον του δίνεται η δυνατότητα, είτε να ξεκινήσει έναν νέο γύρο παιχνιδιού ή να μεταβεί στο αρχικό μενού.

MENUS	BUTTONS
Main Menu	➢ Play ➢ Settings ➢ Quit
Settings Menu	➢ Sound Slider ➢ Back
Pause Menu	➢ Resume ➢ Menu ➢ Quit
Game Over Menu	➢ Restart ➢ Back To Menu

Πίνακας 4 - 1: Menus & Buttons



Σχήμα 4 - 1: Διάγραμμα Ροής μεταξύ των διαφόρων Menu Screens

4.2.6 Μουσική / Ήχοι

Οι διάφοροι ήχοι που χρησιμοποιούνται στο παιχνίδι προκειμένου να συμβάλουν στην αληθοφάνεια του, παρουσιάζονται αναλυτικά στον Πίνακα 4.2.

ΗΧΟΣ	ΠΕΡΙΓΡΑΦΗ
Ήχος για Main Menu	Μουσική που αναπαράγεται κατά την έναρξη του παιχνιδιού.
Ήχος ερήμου	Επαναλαμβανόμενος ήχος για αναπαράσταση των φυσικών ήχων του περιβάλλοντος.
Ήχος για πατημασιές	Τυχαία επιλογή ανάμεσα σε τέσσερα ηχητικά εφέ για την προσομοίωση των διαφόρων ήχων από τα βήματα του παίκτη στην άμμο της ερήμου.
Ήχος για επίθεση με τσεκούρι	Τυχαία επιλογή ανάμεσα σε τρία ηχητικά εφέ για την προσομοίωση των διαφόρων ήχων από την επίθεση με το τσεκούρι.
Ήχος για επίθεση με Revolver	Ηχητικό κλιπ που μοιάζει με ένα περίστροφο όπλο που πυροβολεί.
Ήχος για επίθεση με Shotgun	Ηχητικό κλιπ που μοιάζει με μία καραμπίνα που πυροβολεί.
Ήχος για γέμισμα σφαιρών σε Shotgun	Ηχητικό κλιπ που αναπαριστά το γέμισμα των σφαιρών μίας καραμπίνας.
Ήχος για επίθεση με Assault Rifle	Ηχητικό κλιπ που μοιάζει με ένα αυτόματο όπλο που πυροβολεί.
Ήχος εχθρού όταν εντοπίζει αντίπαλο	Ηχητικό εφέ που προσομοιώνει τον εντοπισμό του παίκτη σε κοντινή απόσταση.
Ήχος για επίθεση από εχθρό	Ηχητικό εφέ που προσομοιώνει την επίθεση του εχθρού προς τον παίκτη.

Ήχος θανάτου του εχθρού	Ηχητικό εφέ που προσομοιώνει την εξόντωση του εχθρού.

Πίνακας 4 - 2: Ήχοι & Περιγραφή

4.2.7 2D / 3D Models

Τα διάφορα μοντέλα που χρησιμοποιούνται στο παιχνίδι αναφέρονται στον παρακάτω πίνακα.

MODELS	ΠΕΡΙΓΡΑΦΗ
Desert Environment Packet	Το κύριο μέρος του terrain. Πάνω σε αυτό βασίστηκε ο σχεδιασμός του περιβάλλοντος.
Street Lamps Packet	Ποικιλία από λάμπες δρόμου για τον εμπλουτισμό αντικειμένων της σκηνής του παιχνιδιού.
Barrels Packet	Ποικιλία από πολύχρωμα βαρέλια για τον εμπλουτισμό αντικειμένων της σκηνής του παιχνιδιού.
Shed Tools Packet	Ποικιλία από αποθήκες και εργαλεία για τον εμπλουτισμό αντικειμένων της σκηνής του παιχνιδιού.
Zombie	Η εχθρική μονάδα του περιβάλλοντος. Κινείται σχετικά αργά μέχρι να εντοπίσει τον παίκτη. Όταν τον εντοπίσει του επιτίθεται προκαλώντας μικρή ζημιά σε κάθε του χτύπημα.
Axe	Το πρώτο διαθέσιμο όπλο του παίκτη. Προκαλεί τεράστια ζημιά και θανατώνει τον αντίπαλο με ένα του χτύπημα. (Πλήκτρο → 1)
Revolver	Το δεύτερο διαθέσιμο όπλο του παίκτη. Εξαιτίας της ισχύς του εμφανίζει κρότο λάμψης και ο ρυθμός επίθεσης (firerate) του είναι αρκετά μικρός. (Πλήκτρο → 2)

Shotgun	Το τρίτο διαθέσιμο όπλο του παίκτη. Εξαιτίας της ισχύς του εμφανίζει κρότο λάμψης και επαναφορτίζει σφαίρες πριν από κάθε του χτύπημα. (Πλήκτρο → 3)
Assault Rifle	Το τέταρτο διαθέσιμο όπλο του παίκτη. Διαθέτει τον ταχύτερο ρυθμό επίθεσης (firerate) και μπορεί να πυροβολεί συνεχώς. (Πλήκτρο → 4)

Πίνακας 4 - 3: Models & Περιγραφή

4.2.8 Game Logic

INTERACTIONS	ΠΕΡΙΓΡΑΦΗ
Score & HighScore	Ο παίκτης κάθε φορά που σκοτώνει ένα Ζόμπι, παίρνει έναν βαθμό στο score του. Το highscore δηλώνει το ανώτερο score που έχει πετύχει ο παίκτης από όλους τους γύρους που έχει παίξει.
Stamina	Η stamina του χαρακτήρα δηλώνει τον βαθμό που κουράζεται ο παίκτης καθώς τρέχει. Όταν ο χρήστης έχει πατημένο το shift, ο χαρακτήρας κινείται πιο γρήγορα και το stamina μειώνεται αναλογικά με τον χρόνο που είναι πατημένο το κουμπί. Όταν το stamina μηδενίσει, ο παίκτης δεν έχει δυνατότητα να κινείται γρήγορα. Η αναπλήρωση του γίνεται όταν ο παίκτης δεν χρησιμοποιεί το πλήκτρο shift.
Ζωή χαρακτήρα	Η ζωή του χαρακτήρα μας αρχικά είναι ίση με 100. Καθώς δεχόμαστε επίθεση από τα Ζόμπι, αυτή μειώνεται μέχρι να φτάσει στο μηδέν, οπότε και ηττούμαστε. Η ζημιά που προκαλούν τα ζόμπι με κάθε τους χτύπημα είναι ίση με 5.

Ζωή Ζόμπι	Η ζωή των αντιπάλων του χαρακτήρα μας αρχικά είναι ίση με 100. Καθώς δέχονται επίθεση από τον χαρακτήρας μας, αυτή μειώνεται μέχρι να φτάσει στο μηδέν, όπου εξοντώνονται. Η ζημιά που προκαλείται στα Ζόμπι είναι διαφορετική για κάθε όπλο που έχει στην διάθεση του ο χρήστης.
Ζημιά Τσεκουριού	Με το τσεκούρι ο χαρακτήρας μας μπορεί να επιτύχει μείωση της ζωή του αντιπάλου του, μόνο αν βρίσκεται σε πολύ κοντινή απόσταση, έτσι ώστε να υπάρχει επαφή ανάμεσα στο τσεκούρι και στον αντίπαλο. Η ζημιά που δέχεται ο αντίπαλος από αυτό είναι ίση με 100, με αποτέλεσμα να θανατώνεται με μόνο μία επίθεση.
Ζημιά Revolver	Η ζημιά του revolver είναι διαφορετική, ανάλογα με την απόσταση από την οποία απέχει ο αντίπαλος από τον χαρακτήρα μας. Συγκεκριμένα διακρίνονται τρεις περιπτώσεις. Αν η απόσταση αυτή είναι μικρότερη από πέντε μέτρα, η ζημιά που προκαλείται είναι 40. Αν η απόσταση είναι από πέντε έως δέκα μέτρα η ζημιά που προκαλείται είναι 30 και τέλος στην περίπτωση που η απόσταση είναι πάνω από δέκα μέτρα, η ζημιά είναι 15.
Ζημιά Shotgun	Η ζημιά του shotgun είναι διαφορετική, ανάλογα με την απόσταση από την οποία απέχει ο αντίπαλος από τον χαρακτήρα μας. Συγκεκριμένα διακρίνονται τρεις περιπτώσεις. Αν η απόσταση αυτή είναι μικρότερη από πέντε μέτρα, η ζημιά που προκαλείται είναι 50. Αν η απόσταση είναι από πέντε έως δέκα μέτρα η ζημιά που προκαλείται είναι 30 και τέλος στην περίπτωση που η απόσταση είναι πάνω από δέκα μέτρα, η ζημιά είναι 10.
Ζημιά Assault Rifle	Η ζημιά του assault rifle είναι διαφορετική, ανάλογα με την απόσταση από την οποία απέχει ο αντίπαλος από τον χαρακτήρα μας. Συγκεκριμένα διακρίνονται τρεις περιπτώσεις. Αν η απόσταση αυτή είναι μικρότερη από πέντε μέτρα, η ζημιά που προκαλείται είναι 35. Αν η απόσταση είναι από πέντε έως δέκα μέτρα η ζημιά που προκαλείται είναι

	25 και τέλος στην περίπτωση που η απόσταση είναι πάνω από δέκα μέτρα, η ζημιά είναι 20.

Κεφάλαιο 5ο: Συμπεράσματα και προτάσεις βελτίωσης

5.1 Συμπεράσματα

Συνοψίζοντας, είναι εμφανές πως η διαδικασία ανάπτυξης ενός προϊόντος ψυχαγωγικού λογισμικού αποτελεί εκτός από μια πολύ ενδιαφέρουσα και παραγωγική διαδικασία, έναν πρωτότυπο τρόπο εκμάθησης όσον αφορά το κομμάτι της σύνθεσης κώδικα. Ακολουθώντας τα αμέτρητα tutorials που υπάρχουν στο διαδίκτυο και εκμεταλλευόμενοι τους πλέον εύκολα προσπελάσιμους πόρους, η ανάπτυξη ενός απλού βιντεοπαιχνιδιού είναι πλέον εφικτή ακόμη και για μη επαγγελματίες χρήστες.

Η τρέχουσα πτυχιακή εργασία υλοποιήθηκε με μηδενικό αρχικό κεφάλαιο, χρησιμοποιώντας μονάχα δωρεάν διαθέσιμα προϊόντα από το κατάστημα της μηχανής δημιουργίας, το Unity Asset Store. Συνεπώς, είναι εύκολα κατανοητό πως παρότι οι μηχανές δημιουργίας βιντεοπαιχνιδιών είναι πολλές σε πλήθος, η επιλογή της καταλληλότερης μπορεί να παίξει καθοριστικό ρόλο στην ολοκλήρωση των στόχων της ομάδας.

Τέλος, είναι ξεκάθαρη η σημασία της ύπαρξης ενός προσχέδιου (GDD) ως τεχνική ανάπτυξης οποιουδήποτε βιντεοπαιχνιδιού ανεξάρτητα από το μέγεθος του ίδιου του παιχνιδιού ή και της ομάδας.

5.2 Προτάσεις βελτίωσης

Οι προτάσεις βελτίωσης της τρέχουσας Πτυχιακής Εργασίας είναι οι παρακάτω :

- Προσθήκη επιμέρους οπτικών εφέ κατά την διάρκεια εκτέλεσης του παιχνιδιού (π.χ. εφέ από αίμα όταν δέχεται ζημιά ένας αντίπαλος ή ο ίδιος ο παίκτης).
- Προσθήκη επιμέρους ακουστικών εφέ κατά τη διάρκεια εκτέλεσης του παιχνιδιού (π.χ. ήχος λαχανιάσματος όταν ξοδεύεται όλο το stamina του παίκτη).
- Προσθήκη ενός η παραπάνω διαθέσιμων σκηνών-terrain μετά από την επίτευξη μιας αρκετά υψηλής βαθμολογίας στην τρέχουσα σκηνή.
- Προσθήκη ποικιλίας διαφορετικών εχθρών με επαυξημένη δυσκολίας εξόντωσης (π.χ. εχθρός που επίσης κουβαλάει όπλο και έχει περισσότερους πόντους ζωής).
- Χρησιμοποίηση του Facebook SDK για Unity. Το Facebook SDK για Unity παρέχει μια ολοκληρωμένη συλλογή των κοινωνικών δυνατοτήτων του Facebook, δίνοντας στους παίκτες του παιχνιδιού Unity τη δυνατότητα να μοιράζονται περιεχόμενο με τους φίλους τους και τους επιτρέπει να δημιουργήσουν μια προσωπική εμπειρία παιχνιδιού.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1]Παπαδάκης, Η Βιομηχανία των Βιντεοπαιχνιδιών και η εξέλιξη τους , 2016 from https://docplayer.gr/2418861-I-viomihania-ton-vinteopaihnidion-kai-i-exelixi-toys.html

[2]Νίκος Κριπιντίρης, Η ιστορία των βιντεοπαιχνιδιών,Dec 2014 from https://unboxholics.com/stories/4829-i-istoria-ton-videopaixnidion

[3] V4T, Βιντεοπαχνίδια, Nov 2017 from https://v4t.pixel-online.org/manual_pdf/10/gr.pdf

[4]Rachel Kowert & Thorsten Quandt, The Video Game Debate, Aug 2015 from https://books.google.gr/books?hl=el&lr=&id=Y-JzCgAAQBAJ&oi=fnd&pg=PA1&dq=history+of+video+games&ots=HNeTPjalqj&sig=oNZ4lDGSKt-0as0uKBQm-lR7p04&redir_esc=y#v=onepage&q=history%20of%20video%20games&f=false

[5]Vince, The Many Different Types Of Video Games & Their Subgenres, Apr 2018 from https://www.idtech.com/blog/different-types-of-video-game-genres

[6]Eludamos. Journal for Computer Game Culture. 2009 from https://www.eludamos.org/index.php/eludamos/article/view/vol3no2-3/126

[7]Calliope, Οι κατηγορίες παιχνιδιών με Competitive mode, Mac 2019 from http://www.ifreegames.gr/2019/03/23/%CE%BF%CE%B9-%CE%BA%CE%B1%CF%84%CE%B7%CE%B3%CE%BF%CF%81%CE%AF%CE%B5%CF%82-%CF%80%CE%B1%CE%B9%CF%87%CE%BD%CE%B9%CE%B4%CE%B9%CF%8E%CE%BD-%CE%BC%CE%B5-competitive-mode/

[8]Φωκίδης, Εικονικά Περιβάλλοντα Μάθησης και Πολυμέσα, Fe 2015 from https://eclass.aegean.gr/modules/document/file.php/TPDE106/%CE%97%CE%BB%CE%B5%CE%BA%CF%84%CF%81%CE%BF%CE%BD%CE%B9%CE%BA%CE%AC%20%CF%80%CE%B1%CE%B9%CF%87%CE%BD%CE%AF%CE%B4%CE%B9%CE%B1.pdf

[9]Dwight Pavlovic, Video Game Genres, July 2020 from https://store.hp.com/us/en/tech-takes/video-game-genres

[10] Wendy Despain, Writing for Video Game Genres, Feb 2009 from https://books.google.gr/books?id=J-XqBgAAQBAJ&dq=genres+of+video+games&hl=el

[11]Pavel Shylenok, Understanding the Roles of Game Dev Professional, Jan 2019 from https://www.gamasutra.com/blogs/PavelShylenok/20190115/334322/Understanding_the_Roles_of_Game_Dev_Professionals.php

[12]Chandan Lunthi, Different Roles In Game Development, Feb 2018 from https://www.oodlestechnologies.com/blogs/Different-Roles-In-Game-Development/

[13] Nurlan Suleymanov, The 9 Roles You Need to Build a Great GDT,Au 2019 from https://www.stalliongaming.com/blog/9-roles-you-need-to-build-game-development-team/

[14]Ual, Game Development roles from https://www.futurelearn.com/info/courses/introduction-to-indie-games/0/steps/96373

[15]Starloop Studios, Game Development Stages, Sep 2020 from https://starloopstudios.com/game-development-stages/

[16]Devin Pickell, The 7 Stages of Game Development, Oct 2019 from https://learn.g2.com/stages-of-game-development

[17]Nuclino, Video Game Development Process from https://www.nuclino.com/articles/video-game-development-process

[18]Andrade,Game engines: a survey, Nov 2015 from https://pdfs.semanticscholar.org/b656/0c35ce1f0484cc2fc75cada34b580953c9ff.pdf

[19]Lindsay Schardon, Best Game Engines of 2021, Jan 2021 from https://gamedevacademy.org/best-game-engines/

[20]Hady EIHady, Top Game Engines from https://instabug.com/blog/game-engines/

[21]Παπαδόπουλος Δημήτρης, Ανάπτυξη ηλεκτρονικού παιχνιδιού στρατηγικής, Se 2019 from https://hellanicus.lib.aegean.gr/bitstream/handle/11610/19525/PapadopoulosDimitrisThesis.pdf?sequence=1

[22]Νικόλαος Μίντζας, Σχεδίαση ενός εικονικού χώρου σε περιβάλλον Blender, De 2017 from https://nemertes.lis.upatras.gr/jspui/bitstream/10889/11380/3/Nemertes_Mintzas%28ele%29.pdf

[23] Ed Fear, 2021 from http://www.develop-online.net/analysis/united-theystand/0116643.

[24]John K.Haas, A History of the Unity Game Engine, March 2014 from https://core.ac.uk/download/pdf/212986458.pdf

[25]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/Hierarchy.html

[26]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/ProjectView.html

[27]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/UsingTheSceneView.html

[28]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/GameView.html

[29]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/Console.html

[30]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/UsingTheInspector.html

[31]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/animeditor-UsingAnimationEditor.html

[32]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/AnimatorWindow.html

[33]Unity Documentation, 2020 from https://docs.unity3d.com/ScriptReference/GameObject.html

[34]Unity Documentation, 2020 from https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html

[35]Needone App, Jun 15 2020 from https://needoneapp.medium.com/unity-are-you-familiar-with-awake-onenable-start-update-fixedupdate-and-ondestroy-3e535617762f

[36]Unity Documentation, 2020 from https://docs.unity3d.com/ScriptReference/Transform.html

[37]Unity Documentation, 2020 from https://docs.unity3d.com/ScriptReference/Vector3.html

[38]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/Prefabs.html

[39]Unity Documentation, 2020 from https://docs.unity3d.com/ScriptReference/Collider.html

[40]Unity Documentation, 2020 from https://docs.unity3d.com/ScriptReference/Rigidbody.html

[41]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/class-AnimatorController.html

[42]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/Coroutines.html

[43 Unity Documentation, 2020 from https://docs.unity3d.com/Manual/Layers.html

[44] Unity Documentation, 2020 from https://docs.unity3d.com/Manual/Tags.html

[45] Unity Documentation, 2020 from https://docs.unity3d.com/Manual/ParticleSystems.html

[46]Unity Documentation, 2020 from https://docs.unity3d.com/Manual/Lighting.html

[47]Unity Technologies, Dec 2020 from https://learn.unity.com/tutorial/working-with-the-terrain-editor-1#

[48]Unity Technologies, 2020 from https://learn.unity.com/tutorial/creating-new-projects#5d92399bedbc2a05c1835459

[49] Brackeys, 2017 from https://devassets.com/assets/desert-environment/

[50]Unity Asset Store, 2020 from https://assetstore.unity.com/packages/audio/music/orchestral/free-game-music-pack-176757

[51]Unity Asset Store, Standard Assets April 2020 from https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351

[52] Unity Asset Store, June 2019 from https://assetstore.unity.com/packages/3d/props/shed-tools-bridge-and-fences-104216

[53] Unity Asset Store, March 2019 from https://assetstore.unity.com/packages/3d/props/exterior/street-lamps-165658

[54] Unity Asset Store, June 2019 from 4 Industrial barrels | 3D Props | Unity Asset Store

[55] Unity Asset Store, July 2018 from https://assetstore.unity.com/packages/templates/systems/ultimate-survival-80368

[56] Unity Documentation, 2020 from https://docs.unity3d.com/Manual/class-Transition.html

[57]Microsoft Documentation in C# ,2020 from https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum

[58]Unity Asset Store, October 2019 from https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232

[59]Unity Documentation , 2020 from https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

[60] Unity Documentation ,2020 from https://docs.unity3d.com/Manual/class-State.html

[61]Unity Asset Store, 2020 from https://assetstore.unity.com/packages/audio/music/orchestral/free-game-music-pack-176757

[62]Icons.db.com, 2021 from https://www.iconsdb.com/white-icons/

[63]Mark Baldwin, Game Design Document Outline, October 2005 from https://people.ucalgary.ca/~jparker/art503/BaldwinGameDesignDocumentTemplate.pdf

[64] RS Martins, FCP Raulino, AM Filgueira, SGDDEdu – “Model of short game design document for digital educational games”, February 2019 from https://repositorio.ufrn.br/bitstream/123456789/29746/1/AModelofShort_BULAMARQUI_2019.pdf

[65] M. G. Salazar, H. A. Mitre, C. L. Olalde and J. L. G. Sánchez, "Proposal of Game Design Document from software engineering requirements perspective", 2012, from https://ieeexplore.ieee.org/document/6314556

[66] DML de Carvalho, FJL Gomes, “Simple Game Design Document Focused on Gameplay Features” ,2016 from http://www.sbgames.org/sbgames2016/downloads/anais/157232.pdf

ΠΑΡΑΡΤΗΜΑ Α : Σύνδεσμος για εκτελέσιμο παιχνίδι

https://drive.google.com/file/d/1qkzN6od7TDKlwgvYQjHV7V7R3t419ncB/view?usp=sharing

ΠΑΡΑΡΤΗΜΑ Β : Κώδικας παιχνιδιού

Β.1 Κλάση MainMenu

Η κλάση που είναι υπεύθυνη για την διαχείριση του κεντρικού-αρχικού μενού.

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

public class MainMenu : MonoBehaviour
{
	public void PlayGame()
	{
		Cursor.lockState = CursorLockMode.Locked;
		Cursor.visible = false;
		SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
	}

	public void QuitGame()
	{
		Debug.Log("QuitedGame");
		Application.Quit();
	}
}
```

Β.2 Κλάση SetVolume

Η κλάση που επιτρέπει στον χρήστη να αυξομειώσει την ένταση ήχου του παιχνιδιού.

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

public class SetVolume : MonoBehaviour
{
	public AudioMixer mixer;

	public void SetLevel(float sliderValue)
	{
		mixer.SetFloat("MusicVol", Mathf.Log10(sliderValue) * 20);
		//μετατροπη του sliderValue σε λογαριθμικη τιμή ωστε
		//να ταιριαζει με την τιμη του mixer
	}
}
```

B.3 Κλάση PauseMenu

Η κλάση που είναι υπεύθυνη για την διαχείριση του μενού που εμφανίζεται κατά τη διάρκεια του παιχνιδιού όταν ο χρήστης πιέσει το πλήκτρο Escape.

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

public class PauseMenu : MonoBehaviour
{
    public GameObject pauseMenuUI;
       public GameObject crosshair;

    public static bool GameIsPaused = false;

    void Update()
       {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if(GameIsPaused)
            {
                Resume();
            } else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
              crosshair.SetActive(true);
              Cursor.lockState = CursorLockMode.Locked;
              Cursor.visible = false ;
        Time.timeScale = 1f; //ξεπαγωμα χρονου!
        GameIsPaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
              crosshair.SetActive(false);
              Cursor.lockState = CursorLockMode.None;
              Cursor.visible = true ;
        Time.timeScale = 0f; //παγωμα χρονου!
        GameIsPaused = true;
    }

    public void LoadMenu()
    {
              Cursor.lockState = CursorLockMode.None;
              Cursor.visible = true ;
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }
```

```
    public void QuitGame()
    {
        Debug.Log("QUIT!");
        Application.Quit();
    }
}
```

B.4 Κλάση GameOverMenu

Η κλάση που είναι υπεύθυνη για την διαχείριση του μενού που εμφανίζεται με τον τερματισμό του παιχνιδιού, όταν ο χρήστης ηττηθεί.

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

public class GameOverMenu : MonoBehaviour
{
       public static GameOverMenu instance ;
    public GameObject gameOverMenuUI;
       public GameObject crosshair;

       public Text scoreText2;
       int score = 0;

       void Awake()
       {
              instance = this ;
       }

    public void Restart()
    {
        gameOverMenuUI.SetActive(false);
              crosshair.SetActive(true);
              Cursor.lockState = CursorLockMode.Locked;
              Cursor.visible = false ;
        Time.timeScale = 1f;
              SceneManager.LoadScene("GameScene");
    }

    public void LoadMenu()
    {
              Cursor.lockState = CursorLockMode.None;
              Cursor.visible = true ;
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }

       public void AddPoint2()
       {
              score += 1;
              scoreText2.text = "YOU SCORED:"+ score.ToString() ;
       }
}
```

B.5 Κλάση ScoreManager

Η κλάση που είναι υπεύθυνη για την αύξηση του τρέχοντος σκορ του παίκτη αλλά και την αποθήκευση της χρονικά υψηλότερης βαθμολογίας.

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

public class ScoreManager : MonoBehaviour
{
	public static ScoreManager instance ;

	public Text scoreText;
	public Text highscoreText;

	int score = 0;
	int highscore = 0;

	void Awake()
	{
		instance = this ;
	}

    void Start()
    {
		highscore = PlayerPrefs.GetInt("highscore", 0);
	 scoreText.text = "SCORE:" + score.ToString() ;
		highscoreText.text = "HIGHSCORE:" + highscore.ToString();
    }

	public void AddPoint(){

		score += 1;
		scoreText.text = "SCORE:" + score.ToString() ;

		if (highscore < score )
			PlayerPrefs.SetInt("highscore", score);
	}
}
```

B.6 Κλάση MouseLook

Η κλάση που είναι υπεύθυνη για την επιτυχημένη κίνηση της κάμερας σε σχέση με το ποντίκι.

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

public class MouseLook : MonoBehaviour {

    [SerializeField]
    private Transform playerRoot, lookRoot;

    [SerializeField]
    private bool invert;

    [SerializeField]
    private bool can_Unlock = true;

    [SerializeField]
```

```
    private float sensivity = 5f;

    [SerializeField]
    private int smooth_Steps = 10;

    [SerializeField]
    private float smooth_Weight = 0.4f;

    [SerializeField]
    private float roll_Angle = 10f;

    [SerializeField]
    private float roll_Speed = 3f;

    [SerializeField]
    private Vector2 default_Look_Limits = new Vector2(-70f, 80f);

    private Vector2 look_Angles;
    private Vector2 current_Mouse_Look;
    private Vector2 smooth_Move;
    private float current_Roll_Angle;
    private int last_Look_Frame;

    void Start ()
       {
        Cursor.lockState = CursorLockMode.Locked;
       }

       void Update ()
       {
        LockAndUnlockCursor();
        if(Cursor.lockState == CursorLockMode.Locked) {
            LookAround();
        }
       }

    void LockAndUnlockCursor() {

        if(Input.GetKeyDown(KeyCode.Tab)) {

            if(Cursor.lockState == CursorLockMode.Locked) {

                Cursor.lockState = CursorLockMode.None; //κλειδωμα-ξεκλειδωμα cursor

            } else {

                Cursor.lockState = CursorLockMode.Locked;
                Cursor.visible = false;

            }

        }

    } // lock and unlock

    void LookAround() {

        current_Mouse_Look = new Vector2( Input.GetAxis("Mouse Y"),
Input.GetAxis("Mouse X"));//MOUSE_Y-κάθετα MOUSE_X οριζόντια

        look_Angles.x += current_Mouse_Look.x * sensivity * (invert ? 1f : -1f);
        look_Angles.y += current_Mouse_Look.y * sensivity;
```

```
        look_Angles.x = Mathf.Clamp(look_Angles.x, default_Look_Limits.x,
default_Look_Limits.y);//θέτουμε τα όρια(apo -70 - +80)

        lookRoot.localRotation = Quaternion.Euler(look_Angles.x, 0f, 0f);
        playerRoot.localRotation = Quaternion.Euler(0f, look_Angles.y, 0f);

    } // look around
}
```

B.7 Κλάση Movement

Η κλάση που είναι υπεύθυνη για την βασική κίνηση του χαρακτήρα μας.

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

public class Movement : MonoBehaviour
{

    private CharacterController character_Controller;

    public float speed = 5f;
    public float jump_Force = 10f;

    private float gravity = 20f;
    private Vector3 move_Direction;
    private float vertical_Velocity;

    void Awake()
       {
        character_Controller = GetComponent<CharacterController>();
    }

       void Update ()
       {
        MoveThePlayer();
       }

    void MoveThePlayer() {
                                                                  //A-D = Horizontal
                        //W-S = Vertical
        move_Direction = new Vector3(Input.GetAxis("Horizontal"), 0f,
Input.GetAxis("Vertical"));

        move_Direction = transform.TransformDirection(move_Direction);
        move_Direction *= speed * Time.deltaTime;

        ApplyGravity();

        character_Controller.Move(move_Direction);

    } // move player

    void ApplyGravity() {

        vertical_Velocity -= gravity * Time.deltaTime;

        PlayerJump();
```

```
        move_Direction.y = vertical_Velocity * Time.deltaTime;

    } // apply gravity

    void PlayerJump() {

        if(character_Controller.isGrounded && Input.GetKeyDown(KeyCode.Space)) {
            vertical_Velocity = jump_Force;
        }

    }
}
```

B.8 Κλάση DifferentTypesOfMovement

Η κλάση που είναι υπεύθυνη για τους συμπληρωματικούς τρόπους κίνησης του χαρακτήρα.

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

public class DifferentTypesOfMovement : MonoBehaviour
{

    private Movement movement;
    private Footsteps footsteps;
    private Stats stats;
    private Transform look_Root;
    private CharacterController character_Controller;

    public float sprint_Speed = 10f;
    public float move_Speed = 5f;
    public float crouch_Speed = 2f;
       public float prone_Speed = 1f;
    public float sprint_Treshold = 10f;

    private float stand_Height = 1.6f;
    private float crouch_Height = 1.1f;
       private float prone_Height = 0.9f;

    private bool is_Crouching;
       private bool is_Proning;

    private float sprint_Volume = 1f;
    private float crouch_Volume = 0.1f;
       private float prone_Volume =0.05f;
    private float walk_Volume_Min = 0.2f, walk_Volume_Max = 0.6f;

    private float walk_Step_Distance = 0.4f;
    private float sprint_Step_Distance = 0.25f;
    private float crouch_Step_Distance = 0.5f;
       private float prone_Step_Distance = 0.6f;

    private float stamina = 100f;

    void Awake()
    {
        movement = GetComponent<Movement>();
        footsteps = GetComponentInChildren<Footsteps>();
        stats = GetComponent<Stats>();
```

```
        look_Root = transform.GetChild(0);
        character_Controller = GetComponent<CharacterController>();
    }

       void Start()
       {
        footsteps.volume_Min = walk_Volume_Min;
        footsteps.volume_Max = walk_Volume_Max;
        footsteps.step_Distance = walk_Step_Distance;
    }

    void Update()
    {
        Sprint();
              Crouch();
              Prone();
    }

    void Sprint() {

        //εαν εχουμε stamina μπορουμε να κανουμε sprint
        if(stamina > 0f) {

            if (Input.GetKeyDown(KeyCode.LeftShift) && !is_Crouching && !is_Proning) {

                movement.speed = sprint_Speed;

                footsteps.step_Distance = sprint_Step_Distance;
                footsteps.volume_Min = sprint_Volume;
                footsteps.volume_Max = sprint_Volume;
            }
        }

        if(Input.GetKeyUp(KeyCode.LeftShift) && !is_Crouching && !is_Proning) {

            movement.speed = move_Speed;

            footsteps.step_Distance = walk_Step_Distance;
            footsteps.volume_Min = walk_Volume_Min;
            footsteps.volume_Max = walk_Volume_Max;

        }
                                                //καταναλωση του stamina μονο αν ο
χαρακτήρας κινειται ΚΑΙ ταυτοχρονα κραταμε πατημενο το Shift
        if(Input.GetKey(KeyCode.LeftShift) && !is_Crouching && !is_Proning &&
character_Controller.velocity.sqrMagnitude > 0) {

            stamina -= sprint_Treshold * Time.deltaTime;

            if(stamina <= 0f) {

                stamina = 0f;

                // επαναφορα ηχου και ταχυτητας
                movement.speed = move_Speed;
                footsteps.step_Distance = walk_Step_Distance;
                footsteps.volume_Min = walk_Volume_Min;
                footsteps.volume_Max = walk_Volume_Max;
            }
            stats.Display_StaminaStats(stamina);

        } else {
```

```
            if(stamina != 100f) {

                stamina += (sprint_Treshold / 2f) * Time.deltaTime;

                stats.Display_StaminaStats(stamina);

                if(stamina > 100f) {
                    stamina = 100f;
                }
            }
        }
    } // sprint

       void Crouch(){

             if(Input.GetKeyDown(KeyCode.C)) {

                   //if we are crouching stand up!
                   if (is_Crouching){

               look_Root.localPosition = new Vector3(0f, stand_Height,
0f);//.localPosition --> so its relative to its parent , not the Unity world position
.
                movement.speed = move_Speed;
                is_Crouching = false;

                footsteps.step_Distance = walk_Step_Distance;
                footsteps.volume_Min = walk_Volume_Min;
                footsteps.volume_Max = walk_Volume_Max;

                   }
                   else {

                         //if we are not crouching - Crouch !!
                look_Root.localPosition = new Vector3(0f, crouch_Height, 0f);
                movement.speed = crouch_Speed;
                is_Crouching = true;

                footsteps.step_Distance = crouch_Step_Distance;
                footsteps.volume_Min = crouch_Volume;
                footsteps.volume_Max = crouch_Volume;

                   }
             }//if we press c

       }//crouch

       void Prone(){

             if (Input.GetKeyDown(KeyCode.Z)){

                   if (is_Proning){ // if we are proning - stand up

                         look_Root.localPosition = new Vector3(0f, stand_Height,
0f);
                         movement.speed = move_Speed;

                         footsteps.step_Distance = walk_Step_Distance;
                         footsteps.volume_Min = walk_Volume_Min ;
                         footsteps.volume_Max = walk_Volume_Max ;
```

```
                        is_Proning = false;
                    }
                    else { // if we re not proning -- prone !

                        look_Root.localPosition = new Vector3(0f, prone_Height,
0f);
                        movement.speed = prone_Speed;

                        footsteps.step_Distance = prone_Step_Distance;
                        footsteps.volume_Min = prone_Volume;
                        footsteps.volume_Max = prone_Volume;

                        is_Proning = true;
                    }
                }
        }//prone
}
```

B.9 Κλάση Organizer

Η κλάση που είναι υπεύθυνη για την διαχείριση και κατηγοριοποίηση των όπλων του χρήστη.

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

public enum WeaponAim {
    NONE,
    AIM
}

public enum WeaponFireType {
    SINGLE,
    MULTIPLE
}

public enum WeaponBulletType {
    BULLET,
    NONE
}

public class Organizer : MonoBehaviour
{
    public WeaponAim weapon_Aim; //τροπος στοχευσης οπλου--τσεκουρι μονο none.
    public WeaponFireType fireType;//τροπος πυροβολισμου--1 σφαιρα ή πολλες
    public WeaponBulletType bulletType;//τυπος σφαιρας
    public GameObject attack_Point;//για το τσεκουρι μονο , ανιχνευει την συγκρουση με
αντιπαλο

    private Animator animator;
    [SerializeField]
    private GameObject muzzleFlash; // για shotgun και revolver
    [SerializeField]
    private AudioSource shootSound, reload_Sound;

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

    public void ShootAnimation() {
```

```
        animator.SetTrigger("Shoot");
    }

    void Play_ShootSound() {
        shootSound.Play();
    }

    void Play_ReloadSound() {
        reload_Sound.Play();
    }

    void Turn_On_AttackPoint() {
        attack_Point.SetActive(true);
    }

    void Turn_Off_AttackPoint() {
        if(attack_Point.activeInHierarchy) {
            attack_Point.SetActive(false);
        }
    }

    void Turn_On_MuzzleFlash() {
        muzzleFlash.SetActive(true);
    }

    void Turn_Off_MuzzleFlash() {
        muzzleFlash.SetActive(false);
    }
}
```

B.10 Κλάση Director

Η κλάση που είναι υπεύθυνη για την επιτυχημένη εμφάνιση και ενεργοποίηση των όπλων του χρήστη.

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

public class Director : MonoBehaviour
{
       [SerializeField] //editing size at inspector
       private Organizer[] weapons;

       private int current_Weapon;

    void Start()
    {
        current_Weapon = 0;
              weapons[current_Weapon].gameObject.SetActive(true);
    }

       void Update ()
       {
                                                  //no1 στο πληκτρολογιο
        if(Input.GetKeyDown(KeyCode.Alpha1)) {
            TurnOnSelectedWeapon(0);
        }
                                                  //no2 στο πληκτρολογιο
        if (Input.GetKeyDown(KeyCode.Alpha2)) {
            TurnOnSelectedWeapon(1);
        }
```

```
                                                    //no3 στο πληκτρολογιο
        if (Input.GetKeyDown(KeyCode.Alpha3)) {
            TurnOnSelectedWeapon(2);
        }
                                                    //no4 στο πληκτρολογιο
        if (Input.GetKeyDown(KeyCode.Alpha4)) {
            TurnOnSelectedWeapon(3);
        }

    }

    void TurnOnSelectedWeapon(int weaponIndex) {

              //αποτρέπει να ΕΠΑΝΑφορτωθει το ιδιο οπλο
        if (current_Weapon == weaponIndex)
            return;

        // απενεργοποιηση τρεχοντος οπλου
        weapons[current_Weapon].gameObject.SetActive(false);

        // ενεργοποιηση επιλεγμενου οπλου
        weapons[weaponIndex].gameObject.SetActive(true);

        // αποθηκευση τρεχουσας τιμης
        current_Weapon = weaponIndex;

    }

    public Organizer GetCurrentSelectedWeapon() {
        return weapons[current_Weapon];
    }

}
```

B.11 Κλάση PlayerAttack

Η abstract κλάση που αποτελεί τον γονιό της αντικειμενοστρεφούς δομής σε σχέση με την επίθεση του χρήστη.

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

public abstract class PlayerAttack : MonoBehaviour
{

    public Director director;
    public GameObject crosshair;
    public Animator zoomCam;
    public Camera mainCam;

    void Awake()
    {
        director = GetComponent<Director>();
        zoomCam = GameObject.FindWithTag("FP Camera").GetComponent<Animator>();
        crosshair = GameObject.FindWithTag("Crosshair");
        mainCam = Camera.main;
    }

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

```
    {
        WeaponShoot();
              ZoomInAndOut();
    }

   public abstract void WeaponShoot();
   public abstract void ZoomInAndOut();
   public abstract void BulletFired();

}
```

B.12 Κλάση RevolverAttack

Η κλάση αυτή κληρονομεί την κλάση PlayerAttack και προσδίδει επιπλέον λειτουργίες στο οπλο νο2-revolver.

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

public class RevolverAttack : PlayerAttack
{
       protected float closeRangeDamage = 40f;
       protected float mediumRangeDamage = 30f;
       protected float longRangeDamage = 15f;

       public override void WeaponShoot(){

       if(Input.GetMouseButtonDown(0)) {

        if(director.GetCurrentSelectedWeapon().bulletType == WeaponBulletType.BULLET
&&
                    director.GetCurrentSelectedWeapon().tag == "Revolver") {

            director.GetCurrentSelectedWeapon().ShootAnimation();

            BulletFired();
        }
       }
       }
       public override void ZoomInAndOut(){
              // we are going to aim with our camera on the weapon
                    if(director.GetCurrentSelectedWeapon().weapon_Aim ==
WeaponAim.AIM) {

                           // if we press and hold right mouse button
                           if(Input.GetMouseButtonDown(1)) {

                                  zoomCam.Play("ZoomIn");
                                  crosshair.SetActive(false);
                           }

                           // when we release the right mouse button click
                           if (Input.GetMouseButtonUp(1)) {

                                  zoomCam.Play("ZoomOut");
                                  crosshair.SetActive(true);
                           }

                    }
```

```
        }
        public override void BulletFired(){
                RaycastHit hit;

         if(Physics.Raycast(mainCam.transform.position, mainCam.transform.forward, out
hit)) {

              if(hit.transform.tag == "Enemy") {

                                        if(Vector3.Distance(hit.transform.position,
transform.position) <= 5f){


        hit.transform.GetComponent<Health>().ApplyDamage(closeRangeDamage);
                                        }

                                        else if (Vector3.Distance(hit.transform.position,
transform.position) <= 10f){


        hit.transform.GetComponent<Health>().ApplyDamage(mediumRangeDamage);
                                        }

                                        else { //long range


        hit.transform.GetComponent<Health>().ApplyDamage(longRangeDamage);

                                        }
              }
         }
        }
}
```

B.13 Κλάση ShotgunAttack

Η κλάση αυτή κληρονομεί την κλάση PlayerAttack και προσδίδει επιπλέον λειτουργίες στο οπλο νο3-shotgun.

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

public class ShotgunAttack : PlayerAttack
{
        protected float closeRangeDamage = 50f;
        protected float mediumRangeDamage = 30f;
        protected float longRangeDamage = 10f;

        public override void WeaponShoot(){

                if(Input.GetMouseButtonDown(0)) {

                        if(director.GetCurrentSelectedWeapon().bulletType ==
WeaponBulletType.BULLET &&
                                director.GetCurrentSelectedWeapon().tag == "Shotgun") {

                                director.GetCurrentSelectedWeapon().ShootAnimation();

                                BulletFired();
                        }
```

```
                }
        }
        public override void ZoomInAndOut(){
                // we are going to aim with our camera on the weapon
                        if(director.GetCurrentSelectedWeapon().weapon_Aim ==
WeaponAim.AIM) {

                                // if we press and hold right mouse button
                                if(Input.GetMouseButtonDown(1)) {

                                        zoomCam.Play("ZoomIn");
                                        crosshair.SetActive(false);
                                }

                                // when we release the right mouse button click
                                if (Input.GetMouseButtonUp(1)) {

                                        zoomCam.Play("ZoomOut");
                                        crosshair.SetActive(true);
                                }

                        }
        }
        public override void BulletFired(){
                RaycastHit hit;

         if(Physics.Raycast(mainCam.transform.position, mainCam.transform.forward, out
hit)) {

             if(hit.transform.tag == "Enemy") {

                                        if(Vector3.Distance(hit.transform.position,
transform.position) <= 5f){


        hit.transform.GetComponent<Health>().ApplyDamage(closeRangeDamage);
                                        }

                                        else if (Vector3.Distance(hit.transform.position,
transform.position) <= 10f){


        hit.transform.GetComponent<Health>().ApplyDamage(mediumRangeDamage);
                                        }

                                        else { //long range


        hit.transform.GetComponent<Health>().ApplyDamage(longRangeDamage);

                                        }
             }
         }
        }
}
```

B.14 Κλάση RifleAttack

Η κλάση αυτή κληρονομεί την κλάση PlayerAttack και προσδίδει επιπλέον λειτουργίες στο οπλο νο4-assault rifle.

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

public class RifleAttack : PlayerAttack
{
	protected float closeRangeDamage = 35f;
	protected float mediumRangeDamage = 25f;
	protected float longRangeDamage = 20f;
	protected float fireRate = 4f;
	protected float nextTimeToFire;

	public override void WeaponShoot(){

		if(director.GetCurrentSelectedWeapon().fireType ==
WeaponFireType.MULTIPLE) {

			// εαν κραταω πατημενο το δεξι κλικ ΚΑΙ εαν Time >
nextTimeToFire
			if(Input.GetMouseButton(0) && Time.time > nextTimeToFire) {

				nextTimeToFire = Time.time + 1f / fireRate;


	director.GetCurrentSelectedWeapon().ShootAnimation();

				BulletFired();
			}
		}
	}
		public override void ZoomInAndOut(){
		// we are going to aim with our camera on the weapon
			if(director.GetCurrentSelectedWeapon().weapon_Aim ==
WeaponAim.AIM) {

			// if we press and hold right mouse button
			if(Input.GetMouseButtonDown(1)) {

				zoomCam.Play("ZoomIn");
				crosshair.SetActive(false);
			}

			// when we release the right mouse button click
			if (Input.GetMouseButtonUp(1)) {

				zoomCam.Play("ZoomOut");
				crosshair.SetActive(true);
			}

		}
	}
	public override void BulletFired(){
		RaycastHit hit;

	 if(Physics.Raycast(mainCam.transform.position, mainCam.transform.forward, out
hit)) {

		if(hit.transform.tag == "Enemy") {

				if(Vector3.Distance(hit.transform.position,
transform.position) <= 5f){
```

```
        hit.transform.GetComponent<Health>().ApplyDamage(closeRangeDamage);
                                    }

                                    else if (Vector3.Distance(hit.transform.position,
transform.position) <= 10f){


        hit.transform.GetComponent<Health>().ApplyDamage(mediumRangeDamage);
                                    }

                                    else { //long range


        hit.transform.GetComponent<Health>().ApplyDamage(longRangeDamage);

                                    }
             }
         }
        }
}
```

B.15 Κλάση AxeAttack

Η κλάση που είναι υπεύθυνη για την επίθεση με το οπλο νο1-axe.

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

public class AxeAttack : MonoBehaviour
{
    private Director director;

    void Awake()
    {
        director = GetComponent<Director>();
    }
    void Update()
    {
        AxeShoot();
    }
       void AxeShoot(){
              if(Input.GetMouseButtonDown(0)) {

                    // διαχειριση τσεκουριου
                    if(director.GetCurrentSelectedWeapon().weapon_Aim ==
WeaponAim.NONE) {
                           director.GetCurrentSelectedWeapon().ShootAnimation();
                    }
       }
   }
}
```

B.16 Κλάση Animations

Η κλάση που είναι υπεύθυνη για την διαχείριση και την κατάλληλη προβολή των διαφορετικών καταστάσεων του αντιπάλου.

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

public class Animations : MonoBehaviour
{
    private Animator animator;

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

    public void Walk(bool walk) {
        animator.SetBool("Walk", walk);
    }

    public void Run(bool run) {
        animator.SetBool("Run", run);
    }

    public void Attack() {
        animator.SetTrigger("Attack");
    }

    public void Dead() {
        animator.SetTrigger("Die");
    }

}
```

B.17 Κλάση Controller

Η κλάση που είναι υπεύθυνη για τις λειτουργίες κίνησης και αλληλεπίδρασης του αντιπάλου με το περιβάλλον.

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

public enum EnemyState {
    PATROL,
    CHASE,
    ATTACK
}

public class Controller : MonoBehaviour
{

    private Animations animations;
    private NavMeshAgent navMeshAgent;
    private EnemyState enemy_State;
    private Transform player;
    private Audio audio;
    public GameObject attack_Point;

    public float walk_Speed = 0.5f;
    public float run_Speed = 4f;

    public float chase_Distance = 20f;
```

```
public float attack_Distance = 2f;
public float chase_After_Attack_Distance = 2f;

public float patrol_Radius_Min = 20f, patrol_Radius_Max = 60f;
public float patrol_For_This_Time = 15f;
public float wait_Before_Attack = 2f;

private float current_Chase_Distance;
private float patrol_Timer;
private float attack_Timer;

void Awake()
   {
    animations = GetComponent<Animations>();
    navMeshAgent = GetComponent<NavMeshAgent>();
    player = GameObject.FindWithTag("Player").transform;
    audio = GetComponentInChildren<Audio>();
}

void Start ()
   {
    enemy_State = EnemyState.PATROL;
    patrol_Timer = patrol_For_This_Time;

    // when the enemy first gets to the player,attack right away
    attack_Timer = wait_Before_Attack;

    // memorize the value of chase distance,so that we can put it back
    current_Chase_Distance = chase_Distance;

   }

   void Update ()
   {

    if(enemy_State == EnemyState.PATROL) {
        Patrol();
    }

    if(enemy_State == EnemyState.CHASE) {
        Chase();
    }

    if (enemy_State == EnemyState.ATTACK) {
        Attack();
    }

}

public EnemyState Enemy_State {
    get; set;
}

void Turn_On_AttackPoint() {
    attack_Point.SetActive(true);
}

void Turn_Off_AttackPoint() {
    if (attack_Point.activeInHierarchy) {
        attack_Point.SetActive(false);
    }
}
```

```
        void Patrol() {

         // enable the agent to move again
         navMeshAgent.isStopped = false;
         navMeshAgent.speed = walk_Speed;

         // add to the patrol timer
         patrol_Timer += Time.deltaTime;

         if(patrol_Timer > patrol_For_This_Time) {

             SetNewRandomDestination();

             patrol_Timer = 0f; //resetin to be able to use again

         }

         if(navMeshAgent.velocity.sqrMagnitude > 0) { // if moving

             animations.Walk(true);

         } else {

             animations.Walk(false);

         }

         // test the distance between the player and the enemy
         if(Vector3.Distance(transform.position, player.position) <= chase_Distance) {

             animations.Walk(false);

             enemy_State = EnemyState.CHASE;

             // play spotted audio
             audio.Play_ScreamSound();

         }


    } // patrol

    void Chase() {

         // enable the agent to move again
         navMeshAgent.isStopped = false;
         navMeshAgent.speed = run_Speed;

         // set the player's position as the destination,because we are chasing(running
towards) the player
         navMeshAgent.SetDestination(player.position);

         if (navMeshAgent.velocity.sqrMagnitude > 0) {

             animations.Run(true);

         } else {

             animations.Run(false);

         }
```

```
        // if the distance between enemy and player is less than the attack distance
        if(Vector3.Distance(transform.position, player.position) <= attack_Distance) {

            // stop the animations
            animations.Run(false);
            animations.Walk(false);
            enemy_State = EnemyState.ATTACK;

            // reset the chase distance to previous
            if(chase_Distance != current_Chase_Distance) {
                chase_Distance = current_Chase_Distance;
            }

        } else if(Vector3.Distance(transform.position, player.position) >
chase_Distance) {
            // player run away from enemy

            // stop running
            animations.Run(false);

            enemy_State = EnemyState.PATROL;

            // reset the patrol timer so that the function,can calculate the new
patrol destination right away
            patrol_Timer = patrol_For_This_Time;

            // reset the chase distance to previous
            if (chase_Distance != current_Chase_Distance) {
                chase_Distance = current_Chase_Distance;
            }
        }

    } // chase

    void Attack() {

          //σταμάτημα της κίνησης του Ζομπι
        navMeshAgent.velocity = Vector3.zero;
        navMeshAgent.isStopped = true;

        attack_Timer += Time.deltaTime;
          //ελεγχος χρονου επιθεσης
        if(attack_Timer > wait_Before_Attack) {

            animations.Attack();

            attack_Timer = 0f;

            audio.Play_AttackSound();

        }
          //εαν ο παικτης τρεξει
        if(Vector3.Distance(transform.position, player.position) >
           attack_Distance + chase_After_Attack_Distance) {

            enemy_State = EnemyState.CHASE;

        }

    } // attack
```

```
    void SetNewRandomDestination() {

        float rand_Radius = Random.Range(patrol_Radius_Min, patrol_Radius_Max);

        Vector3 randDir = Random.insideUnitSphere * rand_Radius;
        randDir += transform.position;

        NavMeshHit navHit;

        NavMesh.SamplePosition(randDir, out navHit, rand_Radius, -1); // ελεγχει την
τιμη του randDir για να δει αν ειναι μεσα στην navigationable περιοχη,

                                    // αν ειναι εκτος, βρισκει νεα τιμη απο το
rand_Radius και θα το αποθηκευσει στην τιμη navHit,το -1 = ολα τα layers.
        navMeshAgent.SetDestination(navHit.position);

    }

}
```

B.18 Κλάση EnemyManager

Η κλάση που είναι υπεύθυνη για τον τρόπο ένταξης και επαναχρησιμοποίησης των αντιπάλων στο παιχνίδι.

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

public class EnemyManager : MonoBehaviour {

    public static EnemyManager instance;
    public Transform[] zombie_SpawnPoints;
    public float wait_Before_Spawn_Enemies_Time = 10f;

    [SerializeField]
    private GameObject zombie_Prefab;
    [SerializeField]
    private int zombie_Enemy_Count;
    private int initial_Zombie_Count;

    void Awake ()
       {
        MakeInstance();
       }

    void Start()
       {
        initial_Zombie_Count = zombie_Enemy_Count;

        SpawnEnemies();

        StartCoroutine("CheckToSpawnEnemies");
    }

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

```
    void SpawnEnemies() {

        int index = 0;
        for (int i = 0; i < zombie_Enemy_Count; i++) {

            Instantiate(zombie_Prefab, zombie_SpawnPoints[index].position,
Quaternion.identity);
            index++;
        }

        zombie_Enemy_Count = 0;

    }

    IEnumerator CheckToSpawnEnemies() {
        yield return new WaitForSeconds(wait_Before_Spawn_Enemies_Time);

        SpawnEnemies();

        StartCoroutine("CheckToSpawnEnemies"); //αδιάκοπη coroutine

    }

    public void EnemyDied() {

            zombie_Enemy_Count++;

            if(zombie_Enemy_Count > initial_Zombie_Count) {
                zombie_Enemy_Count = initial_Zombie_Count;
            }
    }

    public void StopSpawning() {
        StopCoroutine("CheckToSpawnEnemies");
    }

}
```

B.19 Κλάση Footsteps

Η κλάση που είναι υπεύθυνη για την αναπαραγωγή του ήχου των βηματισμών του παίκτη.

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

public class Footsteps : MonoBehaviour
{
    private AudioSource footstep_Sound;
    private CharacterController character_Controller;

    [HideInInspector]//we need them to be public but i dont want them to be visible in
inspector pannel
    public float volume_Min, volume_Max;
    [HideInInspector]
    public float step_Distance;

    [SerializeField]
    private AudioClip[] footstep_Clip;

    private float accumulated_Distance;
```

```
        void Awake ()
        {
         footstep_Sound = GetComponent<AudioSource>();
         character_Controller = GetComponentInParent<CharacterController>(); // getting
the charcontroller from the top parent
        }

        void Update ()
        {
         CheckToPlayFootstepSound();
        }

    void CheckToPlayFootstepSound() {

        // Εαν δεν ειμαστε στο εδαφος
        if (!character_Controller.isGrounded)
            return;

        if(character_Controller.velocity.sqrMagnitude > 0) { //Εαν κινούμαστε

            // accumulated distance ειναι η τιμη του ποσο μακρια μπορει να παει ο
παικτης μεχρι να αναπαραχθει ο ηχος
            accumulated_Distance += Time.deltaTime;

            if(accumulated_Distance > step_Distance) {
                footstep_Sound.volume = Random.Range(volume_Min, volume_Max);
                footstep_Sound.clip = footstep_Clip[Random.Range(0,
footstep_Clip.Length)];
                footstep_Sound.Play();
                accumulated_Distance = 0f;
            }
        } else {
            accumulated_Distance = 0f; //επαναφορά τις τιμής για να μην παιχτει την
επομενη φορα απευθείας ο ηχος!!
        }

    }

}
```

B.20 Κλάση AxeSound

Η κλάση που είναι υπεύθυνη για την αναπαραγωγή των ήχων του τσεκουριού κατά την επίθεση με αυτό.

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

public class AxeSound : MonoBehaviour
{

    [SerializeField]
    private AudioSource audioSource;

    [SerializeField]
    private AudioClip[] woosh_Sounds;

    void PlayWooshSound() {
        audioSource.clip = woosh_Sounds[Random.Range(0, woosh_Sounds.Length)];
        audioSource.Play();
```

```
    }
}
```

B.21 Κλάση Audio

Η κλάση που είναι υπεύθυνη για την αναπαραγωγή των ήχων που προκαλούνται από τον αντίπαλο.

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

public class Audio : MonoBehaviour
{
    private AudioSource audioSource;

    [SerializeField]
    private AudioClip scream_Clip, die_Clip;
    [SerializeField]
    private AudioClip[] attack_Clips;

    void Awake ()
       {
        audioSource = GetComponent<AudioSource>();
       }

    public void Play_AttackSound() {
        audioSource.clip = attack_Clips[Random.Range(0, attack_Clips.Length)];
        audioSource.Play();
    }

    public void Play_ScreamSound() {
        audioSource.clip = scream_Clip;
        audioSource.Play();
    }

    public void Play_DeadSound() {
        audioSource.clip = die_Clip;
        audioSource.Play();
    }

}
```

B.22 Κλάση Attack

Η κλάση που είναι υπεύθυνη για την ανίχνευση της ζημιάς με το οπλο νο1-axe.

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

public class Attack : MonoBehaviour
{
    public float damage = 2f;
    public float range = 1f;
    public LayerMask layerMask;

       void Update ()
       {
        Collider[] strikes = Physics.OverlapSphere(transform.position, range,
layerMask);
                    if(strikes.Length > 0) {
```

```
        strikes[0].gameObject.GetComponent<Health>().ApplyDamage(damage);
                            gameObject.SetActive(false); // ανιχνευση ζημιας μονο μια φορα
και απενεργοποιηση του attackPoint.
                        }
        }
}
```

B.23 Κλάση Health

Η κλάση που είναι υπεύθυνη για την κατάλληλη εφαρμογή της ζημιάς σε όλα τα gameobjects.

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

public class Health : MonoBehaviour
{
        public GameObject gameOverMenuUI;
        public GameObject crosshair;

    private Animations animations;
    private Audio audio;
    private Controller controller;
    private NavMeshAgent navMeshAgent;
       private Stats stats;

    public float health = 100f;
    public bool is_Player, is_Enemy;

    private bool is_Dead;

       void Awake ()
       {
        if(is_Enemy) {
            animations = GetComponent<Animations>();
            controller = GetComponent<Controller>();
            navMeshAgent = GetComponent<NavMeshAgent>();
            audio = GetComponentInChildren<Audio>();
        }

        if(is_Player) {
            stats = GetComponent<Stats>();
        }

       }

    public void ApplyDamage(float damage) {

        if (is_Dead)
            return;

        health -= damage;

        if(is_Player) {
            // εμφάνιση των Stats του Player
            stats.Display_HealthStats(health);
        }

        if(is_Enemy) {
```

```
            if(controller.Enemy_State == EnemyState.PATROL) {
                controller.chase_Distance = 50f;
            }
        }

        if(health <= 0f) {

            SomeoneDied();

            is_Dead = true;
        }

    } // apply damage

    void SomeoneDied() {

        if(is_Enemy) {

                    ScoreManager.instance.AddPoint();
                    GameOverMenu.instance.AddPoint2();
            navMeshAgent.velocity = Vector3.zero;
            navMeshAgent.isStopped = true;
            controller.enabled = false;

            animations.Dead();

            StartCoroutine(DeadSound()); // για να καθυστερήσει ελαχιστα η εκτέλεση

            // αναγέννησε κι αλλα ζομπι
            EnemyManager.instance.EnemyDied();
        }

        if(is_Player) {

            GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");

            for (int i = 0; i < enemies.Length; i++) {
                enemies[i].GetComponent<Controller>().enabled = false;
            }

            // σταματημα αναγεννησης
            EnemyManager.instance.StopSpawning();

            GetComponent<Movement>().enabled = false;
            GetComponent<PlayerAttack>().enabled = false;

GetComponent<Director>().GetCurrentSelectedWeapon().gameObject.SetActive(false);

        }

        if(tag == "Player") {

            Invoke("ShowGameOverMenu", 3f);

        } else {

            Invoke("TurnOffGameObject", 3f);
        }
    } // SomeoneDied

    void TurnOffGameObject() {
        gameObject.SetActive(false);
```

```
    }
    IEnumerator DeadSound() {
        yield return new WaitForSeconds(0.3f);
        audio.Play_DeadSound();
    }

    void ShowGameOverMenu() {
              gameOverMenuUI.SetActive(true);
              crosshair.SetActive(false);
              Cursor.lockState = CursorLockMode.None;
              Cursor.visible = true ;
    }

}
```

B.24 Κλάση Stats

Η κλάση που είναι υπεύθυνη για την εμφάνιση και διαχείριση του ποσοστού ζωής και αντοχής του παίκτη.

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

public class Stats : MonoBehaviour
{

    [SerializeField]
    private Image health_Stats, stamina_Stats;

    public void Display_HealthStats(float healthValue) {

        healthValue /= 100f;
        health_Stats.fillAmount = healthValue;

    }

    public void Display_StaminaStats(float staminaValue) {

        staminaValue /= 100f;
        stamina_Stats.fillAmount = staminaValue;

    }
}
```