ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΥΛΟΠΟΙΗΣΗ 2D PLATFORM ΠΑΙΧΝΙΔΙΟΥ ΜΕΣΩ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ ΑΝΑΠΤΥΞΗΣ UNITY»

Αχιλλέας Μικρώνης Λούντζης – Γεώργιος Νικολακόπουλος

Επιβλέπων: Δρ. Χριστοδούλου Σωτήριος, Επίκουρος Καθηγητής

Πάτρα 2020

Περίληψη

Λαμβάνοντας υπόψιν την ραγδαία εξέλιξη της τεχνολογίας τα τελευταία χρόνια, οι δημιουργοί βιντεοπαιχνιδιών έχουν αποκτήσει την ευχέρεια να επιλέγουν τον τρόπο ανάπτυξής τους, μέσα από μια μεγάλη ποικιλία εργαλείων. Μερικά από αυτά, μπορεί να παρέχονται δωρεάν για οποιονδήποτε έχει την επιθυμία να αναπτύξει κάποιο βιντεοπαιχνίδι και από την άλλη, στις περιπτώσεις μεγάλων παραγωγών μπορεί να αποτελούν ιδιοκτησία της εκάστοτε εταιρείας.

Η πλατφόρμα που χρησιμοποιείται στην προκειμένη περίπτωση ονομάζεται Unity Engine και αποτελεί μια από τις δημοφιλέστερες επιλογές για τους developers. Το συγκεκριμένο εργαλείο δίνει τη δυνατότητα στους χρήστες να δημιουργήσουν σχεδόν οποιοδήποτε είδος παιχνιδιού θελήσουν, παρέχοντας συμβατότητα για όλες τις δημοφιλείς κονσόλες, smartphones, headsets εικονικής πραγματικότητας και ηλεκτρονικούς υπολογιστές. Επίσης, παρέχει την επιλογή για κατασκευή παιχνιδιών είτε δύο διαστάσεων (2D Games), είτε τριών διαστάσεων (3D Games).

Στην παρούσα διπλωματική εργασία, παρουσιάζεται μια οπτική της διαδικασίας που θα μπορούσε κάποιος να ακολουθήσει, σε μορφή ενός οδηγού, για την δημιουργία ενός βιντεοπαιχνιδιού το οποίο κατατάσσεται στην κατηγορία των δύο διαστάσεων. Θα ακολουθήσουμε το ταξίδι του βασικού μας χαρακτήρα σε πέντε διαφορετικά επίπεδα, στα οποία θα κληθεί να αντιμετωπίσει τις προκλήσεις που θα βρεθούν στον δρόμο του, μέχρι την ολοκλήρωση της περιπέτειάς του.

Λέξεις Κλειδιά

Unity Engine, εργαλεία, ανάπτυξη παιχνιδιών, οδηγός, τεχνολογία, βιντεοπαιχνίδια, παιχνίδια δύο διαστάσεων, παιχνίδια τριών διαστάσεων

Abstract

Given the rapid development of technology in recent years, video game developers have the capability to choose how to develop them, through a wide variety of tools. Some of them can be provided free of charge for anyone who has the desire to create a video game, on the other hand, in case of large video game productions they can be owned by each company.

The platform used in our case is called Unity Engine and forms one of the most popular options for developers. This tool allows users to create almost any type of game they want, providing compatibility for all popular consoles, smartphones, virtual reality headsets and computers. It also provides the option to build either two-dimensional (2D Games) or three-dimensional (3D Games) games.

In this dissertation, we present an overview of the process that someone could follow, in the form of a guide, to create a video game that belongs into the two-dimensional category. We will follow the journey of our main character through five different levels, in which he will be called to face the challenges that may appear on his way, until the end of his adventure.

Keywords

Unity Engine, tools, game development, guide, technology, video games, two-dimensional games, three-dimensional games

Περιεχόμενα

Πίνακας Εικόνων

Περιεχόμενα Source codes

Περιεχόμενα Πινάκων

Κεφάλαιο 1ο: Εισαγωγή

Η Ιστορία των Βιντεοπαιχνιδιών, αρχίζει στα τέλη της δεκαετίας του '40. Προς τα τέλη του '50 και στα μέσα του '60, στην Αμερική, αρχίζουν να μπαίνουν στην καθημερινή μας ζωή, οι υπολογιστές. Για την ακρίβεια, οι κεντρικοί υπολογιστές. Από εκείνη την περίοδο, τα βιντεοπαιχνίδια έκαναν την εμφάνιση τους, στις κονσόλες, στα φλίπερ, στους υπολογιστές, αλλά και στις φορητές κονσόλες.

Η ιστορία των κονσολών ξεκινάει ουσιαστικά όταν χρονολογικά ξεκινάει και των βιντεοπαιχνιδιών, κάπου στη δεκαετία του 1940, όταν το 1947 ο Thomas T. Goldsmith, και ο νεότερος Estle Ray Mann κατέθεσαν αίτηση στις Ηνωμένες Πολιτείες ευρεσιτεχνίας για μια εφεύρεση που περιγράφεται ως "μια συσκευή καθοδικού σωλήνα διασκέδασης".

Σήμερα τα βιντεοπαιχνίδια έχουν εξελιχτεί σε μια από τις μεγαλύτερες βιομηχανίες διασκέδασης μαζί με τον κινηματογράφο και την μουσική. Με το πέρασμα των χρόνων και την εξέλιξη της τεχνολογίας καταφέρνουν να αναπαριστούν όλο και πιο πολύπλοκα και εντυπωσιακά γραφικά. Χαρακτηριστικό παράδειγμα αποτελεί η πέμπτη γενιά κονσόλων η οποία επέτρεψε την εύκολη αναπαράσταση τρισδιάστατων γραφικών. Μέχρι τότε οι κονσόλες έπαιζαν κυρίως παιχνίδια δισδιάστατων γραφικών εξαιτίας των περιορισμών τους λόγω του hardware. Αν και τα πρώτα τρισδιάστατα γραφικά ήταν αρκετά “τετραγωνισμένα”, με την βελτίωση του hardware σήμερα, μπορούμε να αναπαραστίσουμε αρκετά ρεαλιστικά σκηνικά και μοντέλα στις οθόνες μας.

Αξίζει να σημειωθεί ότι εξελίσσεται συνεχώς ολόκληρη η εμπειρία των βιντεοπαιχνιδιών και όχι μόνο τα γραφικά. Τα τελευταία χρόνια για παράδειγμα έχουμε δεί την ραγδαία εξέλιξη των virtual reality headsets. Πρόκειται για κράνη που φοράει στο κεφάλι του ο χρήστης τα οποία περιέχουν τριγύρω μια ενιαία οθόνη, η οποία καλύπτει ολόκληρο το οπτικό πεδίο του χρήστη. Με αυτόν τον τρόπο μπορεί να προσομοιώνει ολόκληρους τεχνητούς κόσμους και να μπερδεύει τον χρήστη ότι όντως βρίσκεται μέσα σε αυτόν, γιατί δεν καλύπτει περιορισμένο κομμάτι του οπτικού πεδίου όπως μια τηλεόραση. Μια ακόμα σημαντική εξέλιξη ήταν ότι με την εδραίωση του ίντερνετ τα παιχνίδια απέκτησαν περισσότερα κοινωνικά χαρακτηριστικά. Πλέον μπορεί κάποιος να παίζει από το σπίτι του ένα παιχνίδι με αντιπάλους από όλο τον κόσμο χωρίς να χρειάζεται να βρίσκεται στο ίδιο δωμάτιο με αυτούς ή να μοιράζεται μια τηλεόραση.

Τα βιντεοπαιχνίδια καθώς και η ανάπτυξη παιχνιδιών για κινητές συσκευές αυξάνονται. Μέρα με τη μέρα, τα παιχνίδια για σταθερούς υπολογιστές αλλά και για κονσόλες αντικαθίστανται από παιχνίδια που εκτελούνται σε κινητά τηλέφωνα και tablet. Επιπλέον,

άτομα που ασχολούνται με το σχεδιασμό και την ανάπτυξη παιχνιδιών προέρχονται από διάφορους τομείς. Για παράδειγμα, παιδαγωγοί και ειδικοί στον τομέα με περιορισμένες, εάν υπάρχουν, δεξιότητες προγραμματισμού εμπλέκονται στο σχεδιασμό και την ανάπτυξη βιντεοπαιχνιδιών. Επομένως, η επιλογή ανάπτυξης ενός παιχνιδιού χρησιμοποιώντας αποκλειστικά μια γλώσσα προγραμματισμού, όπως C++, C# ή Java, δεν είναι πραγματικά εφικτή για όλους. Αυτό έκανε τη χρήση μηχανών παιχνιδιών ακόμη πιο σημαντική. Οι μηχανές ανάπτυξης παιχνιδιών επιταχύνουν τη διαδικασία ανάπτυξης ενός παιχνιδιού μέσω υπαρχόντων προτύπων και στοιχείων που μπορούν να επαναχρησιμοποιηθούν, ελαχιστοποιώντας ή εξαλείφοντας εντελώς την ανάγκη να έχουμε μια βαθιά γνώση του προγραμματισμού. Επιπλέον, οι μηχανές ανάπτυξης παιχνιδιών δίνουν την ευκαιρία να αναπτύξουν ένα παιχνίδι μία φορά και να το εξαγάγουν σε διάφορες πλατφόρμες, συμπεριλαμβανομένων των κινητών συσκευών, κάνοντας μερικές μόνο αλλαγές στην αρχική έκδοση.

Ωστόσο, υπάρχουν πολλές μηχανές ανάπτυξης παιχνιδιών που μοιράζονται ορισμένα κοινά χαρακτηριστικά, αλλά έχουν επίσης και πολλές διαφορές. Στην πραγματικότητα, κάθε μηχανή έχει διαφορετική φιλοσοφία στην ανάπτυξη παιχνιδιών και με αποτέλεσμα να στοχεύουν σε ένα ευρύ φάσμα διαφορετικών αναγκών: μηχανές που δεν απαιτούν γνώσεις προγραμματισμού, μηχανές που βασίζονται σε δημοφιλείς τεχνολογίες Ιστού, μηχανές που είναι ανοιχτού κώδικα και μπορούν να προσαρμοστούν / επεκταθούν από έμπειρους χρήστες και επαγγελματικές μηχανές ανάπτυξης παιχνιδιών. Η σωστή επιλογή μιας μηχανής παιχνιδιών είναι μια απόφαση πολλαπλών παραγόντων και δεν είναι εύκολη. Παρόλο που υπάρχουν πολλά άρθρα κριτικών σχετικά με τις μηχανές ανάπτυξης, στις περισσότερες περιπτώσεις αυτά τα άρθρα επικεντρώνονται σε έναν συγκεκριμένο τύπο μηχανών (π.χ. μηχανές παιχνιδιών 3D για κινητά) ή σε έναν συγκεκριμένο τομέα (π.χ. προσομοίωση χειρουργικής εκπαίδευσης). Η παροχή μιας επισκόπησης των μηχανών ανάπτυξης που καλύπτουν ένα ευρύ φάσμα διαφορετικών προφίλ χρήστη και αναγκών με βάση μια εκτεταμένη λίστα χαρακτηριστικών θεωρείται σημαντική. Κατά συνέπεια, ο σκοπός αυτής της διπλωματικής εργασίας είναι να ενημερώσει όσους ενδιαφέρονται για: όλες τις δυνατότητες που ενδέχεται να υπάρχουν σε μια μηχανή ανάπτυξης που οργανώνονται σε κατηγορίες υψηλότερου επιπέδου. Τους κύριους τύπους μηχανών ανάπτυξης, τις δυνατότητες που προσφέρονται από αντιπροσωπευτικά παραδείγματα μηχανών ανάπτυξης που εμπίπτουν στις προαναφερθείσες κατηγορίες.

Ο απώτερος στόχος είναι η παροχή υποστήριξης στην επιλογή της καταλληλότερης μηχανής παιχνιδιών με βάση τις ανάγκες ενός συγκεκριμένου βιντεοπαιχνιδιού και το προφίλ των ατόμων που πρόκειται να το εφαρμόσουν.

Κεφάλαιο 2ο: Μηχανές Ανάπτυξης

2.1. Μηχανές Ανάπτυξης Παιχνιδιών

Οι παραγωγοί μηχανών παιχνιδιών αποφασίζουν πώς επιτρέπουν στους χρήστες να χρησιμοποιούν τα προϊόντα τους. Ακριβώς όπως το gaming είναι μια βιομηχανία, έτσι και οι μηχανές παιχνιδιών που κατασκευάζονται. Οι κυριότερες μηχανές παιχνιδιών διατίθενται σε διάφορες τιμές, είτε πρόκειται για συνδρομή είτε για πληρωμές αδειών.

Η Unreal Engine 4 [1], είναι μια από τις σημαντικότερες μηχανές παιχνιδιών και χρησιμοποιείται για τη δημιουργία πολλών αξιοσημείωτων παιχνιδιών, όπως το Fortnite, το PlayerUnknown's Battlegrounds και το Life Is Strange 2, υιοθέτησε μια δομή ελεύθερης χρήσης με δικαιώματα εκμετάλλευσης όλων τον πωλήσεων παιχνιδιών που χρησιμοποιούν αυτή την μηχανή.

Μια άλλη μηχανή παιχνιδιών που φέρνει επί του παρόντος ένα αξιοσημείωτο εισόδημα είναι το Unity [2], το οποίο υιοθετεί μια παρόμοια τακτική πληρωμής με την Unreal Engine. Το Unity βρίσκεται πίσω από παιχνίδια όπως το Rust, το Subnautica και το Life Is Strange: Before the Storm.

Εικόνα 1. Οι πιο Γνωστές Μηχανές Ανάπτυξης

Το GameMaker [3] κυκλοφόρησε για πρώτη φορά με το όνομα Animo το 1991 και ήταν κατάλληλο για τη δημιουργία παιχνιδιών 2D. Παρόλο που αυτή η μηχανή παιχνιδιών είναι κατάλληλη για 2D παιχνίδια, επιτρέπει στον χρήστη να προσθέσει 3D γραφικά και physics. Το GameMaker είναι εύκολο στη χρήση για αρχάριους χρήστες, καθώς δεν απαιτεί γνώσεις προγραμματισμού. Ωστόσο, η λειτουργικότητα της κάμερας 3D σε σύγκριση με τα γραφικά 3D είναι περιορισμένη.

Το JMonkey [4] κυκλοφόρησε για πρώτη φορά το 2003. Πρόκειται για μια δωρεάν μηχανή παιχνιδιών και εξάγει παιχνίδια χωρίς κόστος σε όλες τις πλατφόρμες, συμπεριλαμβανομένων των κινητών συσκευών. Το JMonkey υποστηρίζει τρισδιάστατα γραφικά. Αν και είναι απαραίτητο να έχουμε εμπειρία στην γλώσσα προγραμματισμού Java, είναι μια μηχανή ανοιχτού κώδικα που επιτρέπει στους χρήστες να την επεκτείνουν και να την προσαρμόσουν στις ανάγκες τους.

Η μηχανή παιχνιδιών 3D Ogre [5] κυκλοφόρησε για πρώτη φορά τον Νοέμβριο του 2013. Το Ogre3D είναι μια δωρεάν μηχανή παιχνιδιών ανοιχτού κώδικα. Επιτρέπει την εξαγωγή ενός παιχνιδιού σε απόδοση γραφικών iOS, Android και Windows Phone 8.

Η μηχανή παιχνιδιών Sio2 [6] κυκλοφόρησε για πρώτη φορά το 2009 και μέσα σε ένα χρόνο έγινε μία από τις 3 πιο δημοφιλείς μηχανές παιχνιδιών που χρησιμοποιήθηκαν στο App Store. Το Sio2 είναι μια μηχανή παιχνιδιών που προηγουμένως ήταν δωρεάν και ανοιχτού κώδικα. Ωστόσο, είναι πλέον διαθέσιμο στους χρήστες μέσω πληρωμής. Κάποιος μπορεί επίσης να αποκτήσει πρόσβαση στον πηγαίο κώδικα, αγοράζοντας την έκδοση Certified Developer. Μια δοκιμαστική έκδοση παρέχεται στους χρήστες με τις περισσότερες διαθέσιμες δυνατότητες, όπως η εξαγωγή για κινητά. Ωστόσο, η προσομοίωση παιχνιδιών παρέχεται στις διάφορες συσκευές iOS και είναι μια από τις πιο δημοφιλείς μηχανές παιχνιδιών για εξαγωγή σε πλατφόρμες iOS παρέχοντας προηγμένα 3D γραφικά.

Εικόνα 2. Μηχανές Ανάπτυξης Παιχνιδιών

2.2. Τεχνική Ανάλυση Μηχανών Ανάπτυξης

Η οπτικοακουστική πιστότητα αναφέρεται στο πόσο ρεαλιστικός φαίνεται ο κόσμος του παιχνιδιού οπτικά και ακουστικά [7]. Η λειτουργική πιστότητα ορίζεται από το βαθμό στον οποίο η προσομοίωση λειτουργεί σωστά όταν οι μηχανές παιχνιδιών ανταποκρίνονται σε διεργασίες του χρήστη [8]. Η έννοια της συνθεσιμότητας χρησιμοποιείται για να περιγράψει την επαναχρησιμοποίηση περιεχομένου που δημιουργήθηκε σε μια μηχανή παιχνιδιών, αλλά και τη δυνατότητα της μηχανής να εισάγει και να χρησιμοποιεί δεδομένα από γνωστές ή εμπορικές πηγές [7]. Τα σετ εργαλείων προγραμματιστών αναφέρονται σε SDK και GDK στα οποία έχουν πρόσβαση οι προγραμματιστές προκειμένου να συνδέσουν τις μηχανές παιχνιδιών με περιφερειακές συσκευές ή άλλα API λογισμικού [7]. Η προσβασιμότητα των χρηστών αναφέρεται στη χρηστικότητα και την τιμή μιας μηχανής παιχνιδιών και είναι ένα πολύ σημαντικό κριτήριο για την επιλογή της ως κατάλληλη για έναν συγκεκριμένο σκοπό. Η δικτύωση είναι ζωτικής σημασίας σε ορισμένους τύπους παιχνιδιών, όπως το MMORPG, επειδή πολλοί χρήστες έχουν διαφορετικά χαρακτηριστικά που επηρεάζουν τη φύση του παιχνιδιού και ορίζονται στα πρώτα στάδια της ανάπτυξής του [9]. Η κατηγορία χαρακτηριστικών ανάπτυξης αναφέρεται στο λειτουργικό σύστημα στο οποίο μπορεί να χρησιμοποιηθεί [9] μια μηχανή παιχνιδιών και τα API γραφικών που υποστηρίζει [10]. Τέλος, οι πλατφόρμες ανάπτυξης, ή αλλιώς η ετερογένεια, είναι μια θεμελιώδης πρόκληση για τους χρήστες του σχεδιασμού εικονικών περιβαλλόντων, καθώς η ικανότητα ανάπτυξης παιχνιδιών σε πλατφόρμες για ένα ευρύ φάσμα υλικού και λογισμικού είναι ένα αξιοσημείωτο πλεονέκτημα [7].

2.2.1. Οπτικοακουστική Πιστότητα

Το Game Maker είναι η λιγότερο ανεπτυγμένη μηχανή παιχνιδιών όσον αφορά την οπτικοακουστική πιστότητα, καθώς η μόνη κατηγορία στην οποία υποστηρίζει όλες τις δυνατότητες είναι ο ήχος. Το JMonkey είναι μια από τις καλύτερες μηχανές παιχνιδιών στην κατηγορία του στατικού global illumination μαζί με την Unreal Engine 4 που είναι ακόμα καλύτερο και το Ogre3D, αλλά δεν υποστηρίζει φωτισμό ανά pixel, ροή ήχου και occlusion culling. Το Ogre3D έχει ένα πλεονέκτημα στην κατηγορία των σκιών μαζί με την Unreal Engine. Παρατηρούμε ότι το Sio2 έχει μεγάλο μειονέκτημα στα τρισδιάστατα γραφικά για κινητές συσκευές και στο editor mapping, καθώς δεν υποστηρίζει καμία από τις σχετικές

δυνατότητες που εξετάζουμε σε αυτές τις κατηγορίες. Ταυτόχρονα, δεν υποστηρίζει ήχο 2D, μορφοποίηση και ανάμειξη των animations.

Το Unity δεν υποστηρίζει μόνο τα πολλαπλά textures, ενώ από την άλλη υποστηρίζει όλες τις άλλες δυνατότητες και φαίνεται να ξεπερνά τις υπόλοιπες μηχανές παιχνιδιών σε animations μαζί με την Unreal Engine 4. Η Unreal Engine 4 φαίνεται να προηγείται σε οπτικοακουστική πιστότητα καθώς υποστηρίζει όλες τις λειτουργίες και έχει ένα πλεονέκτημα στα animations (μαζί με το Unity όπως αναφέρθηκε), τις σκιές και το global illumination.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
1. Rendering						
1.1. Texture						
1.1.1. Basic	✓	✓	✓	✓	✓	✓
1.1.2. Multi-textural	✗	✓	✓	✓	✗	✓
1.1.3. Procedural	✗	✓	✓	✗	✓	✓
1.2. Φωτισμοί						
1.2.1. Per-Vertex	✓	✓	✓	✓	✓	✓
1.2.2. Per-Pixel	✗	✗	✓	✓	✓	✓
1.2.3. Light Mapping	✗	✓	✓	✓	✓	✓
1.2.4. Gloss/Specular Maps	✗	✓	✗	✗	✓(alpha channel)	✓
1.2.5. Anisotropic Reflection	✗	✓	✗(normal reflection)	✓	✓	✓
1.3. Σκιές						
1.3.1. Shadows Volume	✗	✓	✓	✗	✗	✓
1.3.2. Shadow Mapping	✗	✓	✓	✓	✓	✓
1.3.3. Projected	✗	✗	✓	✓	✗	✓
1.3.4. Projected Planar	✗	✗	✓	✗	✓	✓
1.4. Ειδικά Εφέ						
1.4.1. Environmental Mapping	✓	✓	✓	✓	✓	✓
1.4.2. Particle System	✓	✓	✓	✓	✓	✓
1.4.3. Billoarding	✓(3D)	✓	✓	✓	✓	✓
1.4.4. Lens Flare	✗	✓	✓	✓	✓	✓
2. Animation						
2.1. Forwards/ Kinematics	✗	✗	✗	✗	✓	✓
2.2. Inverse Kinematics	✗	✗	✓	✗	✓	✓
2.3. KeyFrame Animation	✗	✓	✗	✓	✓	✓
2.4. Skeletal Animation	✓	✓	✓	✓	✓	✓
2.5. Morphing Animation	✓(2D)	✗	✓	✗	✓	✓
2.6. Blending	✗	✓	✓	✗	✓	✓
3. Ήχος						
3.1. 2D Sound	✓	✓	✓	✗	✓	✓
3.2. 3D Sound	✓	✓	✓	✓	✓	✓
3.3. Streaming Sound	✓	✗	✓	✓	✓	✓
4. Γραφικά 3D για Κινητές Συσκευές						
4.1. Map Editor						
4.1.1. Bump Mapping	✗	✓	✓	✗	✓	✓
4.1.2. Parallax Mapping	✗	✓	✓	✗	✓	✓
4.1.3. Normal Mapping	✗	✓	✓	✗	✓(not on mobile)	✓
4.2. Global Illumination						
4.2.1. Ray Tracing	✗	✗	✓	✗	✓	✓
4.2.2. Ambient Occlusion	✗	✓	✓	✗	✓(not on mobile)	✓
4.2.2.1. Cut Scene Editor	✗	✓	✗	✗	✓(not free)	✓(Matinee)
5. Επεξεργασία Σκηνών						
5.1. Occlusion Culling	✗	✗	✓	✓	✓	✓

Πίνακας 1. Οπτικοακουστική Πιστότητα

2.2.2. Λειτουργική Πιστότητα

Το Unity και η Unreal Engine 4 υποστηρίζουν όλα τα χαρακτηριστικά λειτουργικής πιστότητας και διαθέτουν το PhysX SDK για φυσική προσομοίωση που είναι κατάλληλη για αυτόν τον σκοπό [11]. Από την άλλη το Game Maker, η Ogre3D, η JMonkey και το Sio2 έχουν μερικούς περιορισμούς πάνω στην υποστήριξη λειτουργιών.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
Προγραμματισμός						
Γλώσσα Προγραμματισμού	GameMaker Language	Java	C++	C, C++, Lua	C#, JavaScript, Boo	C++, Blueprint
Programmaing Editor	✓	✓	✓	✗	✓	✓
GUI Editor	✗	✓	✓	✗	✓	✓
Debugger	✓	✓	✓	✓	✓	✓
Profiler	✓	✓	✓	✓(not in free version)	✓	✓
Τεχνητή Νοημοσύνη						
Πρόγραμμα Επεξεργασίας AI	✗	✓	✗	✓	✓	✓
Εντοπισμός Μονοπατιού	✓	✓	✓	✓	✓	✓
Λήψη Αποφάσεων	✗	✓	✗	✗	✓	✓
Ανίχνευση Συγκρούσεων	✓(2D)	✓	✓	✓	✓	✓
Physics						
Animation Editor	✓	✗	✓	✗	✓	✓
Material Editor	✗	✓	✓	✗	✓	✓
Particle Effect Editor	✗	✓	✓	✗	✓	✓
3D Audio Editor	✓	✗	✗	✗	✓	✓

Πίνακας 2. Λειτουργική Πιστότητα

2.2.3. Συνθετικότητα

Όπως φαίνεται στον Πίνακα 5 όλες οι μηχανές παιχνιδιών, με εξαίρεση το GameMaker που είναι μια μηχανή για 2D παιχνίδια, υποστηρίζουν εισαγωγή / εξαγωγή περιεχομένου από / προς τις πιο γνωστές πλατφόρμες CAD. Το GameMaker εισάγει μοντέλα 3D animation μόνο από το λογισμικό Misfit Model 3D, το οποίο είναι παλιό και χωρίς υποστήριξη.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
1. Εισαγωγή/Εξαγωγή Περιεχομένου						
1.1 MAYA	✗	✓	✓	✓	✓	✓
1.2 3D Studio MAX	✗	✓	✓	✓	✓	✓
1.3 Blender	✗	✓	✓	✓	✓	✓

Πίνακας 3. Συνθετικότητα

2.2.4. Εργαλεία Προγραμματιστών

Στον Πίνακα 6 παρουσιάζονται τα ενσωματωμένα SDK και GDK που περιλαμβάνονται σε κάθε μηχανή παιχνιδιών. Τα Unity, GameMaker και Sio2 απαιτούν την ξεχωριστή εγκατάσταση SDK για εξαγωγή σε iOS, Android ή Windows Phone, ενώ οι άλλες μηχανές παιχνιδιών περιλαμβάνουν αυτόματα τα απαιτούμενα SDK.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
1. Εργαλεία Προγραμματιστών						
1.1 Windows Phone SDK	✗	-	✓	-	✓	✓
1.2 iOS SDK	✓	✓	✓	✗	✗	✓
1.3 Android SDK	✗	✓	✓	✗	✗	✓
1.4 Android GDK	✓	✓	✓	✗	✗	✓

Πίνακας 4. Εργαλεία Προγραμματιστών

2.2.5. Προσβασιμότητα

Σύμφωνα με τον Πίνακα 7, το Unity φαίνεται να είναι η πιο χρησιμοποιούμενη μηχανή παιχνιδιών, παρέχοντας τα περισσότερα δωρεάν εκπαιδευτικά προγράμματα, παραδείγματα και assets, ενώ διαθέτει και μια πολύ μεγάλη κοινότητα. Ωστόσο, η τεχνική υποστήριξη δεν προσφέρεται στη δωρεάν έκδοση, σε αντίθεση με την Unreal Engine 4 στην οποία παρέχεται δωρεάν. Επιπλέον, οι μηχανές παιχνιδιών JMonkey, Ogre3D και Unreal Engine 4 είναι εντελώς δωρεάν και ανοιχτού κώδικα. Η μηχανή Sio2, παρόλο που προηγουμένως παρεχόταν δωρεάν στους χρήστες, η σημερινή all-in-one έκδοση με πρόσβαση στον πηγαίο κώδικα της μηχανής διατίθεται στα 1999$, καθιστώντας την πιο ακριβή από οποιαδήποτε άλλη.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
1. Χρηστικότητα						
1.1. Ευκολία Μάθησης						
1.1.1. Tutorials	3	4	4	3	5	4
1.1.2. Παραδείγματα	2	4	4	3	5	4
1.2. Υποστήριξη						
1.2.1. Τεχνική Υποστήριξη	✗	✗	✗	✓ (not in free version)	✓ (Professional)	✓
1.2.2. Υποστήριξη Κοινότητας	3	3	4	3	5	4
2. Κόστος						
2.1. Δωρεάν	Studio Free	✓	✓	Trial Version	Personal Version	✓
2.2. Ανοιχτού Κώδικα	✗	✓	✓	Certified Developer	Professional	✓
2.3. Κόστος/Ετος	Studio Professional Version: 149.99$ + 299.99$	-	-	Certified Developer Version: 1999$	Plus Version: 395$ Professional: 1500$	5% Commision after first 3000$ of the first game

Πίνακας 5. Προσβασιμότητα

2.2.6. Δικτύωση

Όπως φαίνεται στον Πίνακα 8, όλες οι μηχανές παιχνιδιών ακολουθούν το ίδιο μοντέλο δικτύου και μπορούν να εξάγουν παιχνίδια που απευθύνονται σε πολλούς χρήστες.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
1. Δικτύωση						
1.1 Client-Server	✓	✓	✓	✓	✓	✓
1.2 Peer-to-Peer	✗	✗	✗	✗	✗	✗
1.3 Multiplayer	✓	✓	✓	✓	✓	✓

Πίνακας 6. Δικτύωση

2.2.7. Λειτουργίες Ανάπτυξης

Όπως μπορούμε να δούμε στον Πίνακα 6, το GameMaker και το Unity δεν μπορούν να εγκατασταθούν και να εκτελεστούν σε πλατφόρμες Linux, ενώ το GameMaker δεν μπορεί να εγκατασταθεί σε MacOSX. Όλες οι μηχανές παιχνιδιών υποστηρίζουν το DirectX (ή το Direct3D) και το OpenGL, εκτός από τα JMonkey και Sio2 που υποστηρίζουν μόνο τη βιβλιοθήκη OpenGL. Επιπλέον το GameMaker έχει το μειονέκτημα ότι τρέχει μόνο στα Windows.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
1. Λειτουργικά Συστήματα						
1.1 Windows	✓	✓	✓	✓	✓	✓
1.2 Mac OSX	✗	✓	✓	✓	✓	✗
1.3 Linux	✗	✓	✓	✓	✗	✗
2. Graphics API	Direct X 9, OpenGL	OpenGL	OpenGL, Direct3D	OpenGL	Direct X, OpenGL	Direct X 9, OpenGL

Πίνακας 7. Λειτουργίες Ανάπτυξης

2.2.8. Πλατφόρμες Ανάπτυξης

Από τον Πίνακα 10, μπορούμε να συμπεράνουμε ότι οι JMonkey και Unreal Engine 4 δεν είναι τόσο δυνατές στην ανάπτυξη παιχνιδιών σε κινητά τηλέφωνα. Όσον αφορά τις πλατφόρμες επιτραπέζιων υπολογιστών, τα JMonkey, Ogre3D και Sio2 δεν υποστηρίζουν όλες τις διαθέσιμες πλατφόρμες. Το Unity έχει ένα σημαντικό πλεονέκτημα στην ανάπτυξη παιχνιδιών για κονσόλες, καθώς υποστηρίζει όλες τις κονσόλες που εξετάσαμε.

Λειτουργίες/ Μηχανές Παιχνιδιών	Game Maker	JMonkey	Ogre3D	Sio2	Unity	Unreal Engine 4
1. Κινητές Συσκευές						
1.1. iOS	✓ (not in free version)	✓	✓	✓	✓	✓
1.2. Android	✓ (not in free version)	✓	✓	✓	✓	✓
1.3. WebOS(Palm)	✗	✗	✗	✓	✓	✗
1.4. Windows Mobile 6	✗	✗	✗	✗	✗	✗
1.5. Windows Mobile 7	✓ (not in free version)	✗	✗	✗	✗	✗
1.6. Windows Phone 8	✓ (not in free version)	✗	✓	✗	✓	✗
1.7. Bada	✗	✗	✗	✓	✗	✗
1.8. Symbian	✗	✗	✗	✗	✗	✗
2. Desktop						
2.1. Windows	✓	✓	✓	✓	✓	✓
2.2. Mac OSX	✓	✓	✓	✓	✓	✓
2.3. Linux	✓	✓	✓	✗	✓(Linux/Steam OS)	✓
2.4. Web	HTML5 (not in free version)	✗	✗	✗	WebGL	HTML5
3. Consoles						
3.1. Playstation	✓ (not in free version)	✗	✓	✗	✓	✓
3.2. Wii	✗	✗	✗	✗	✓	✗
3.3. XBOX	✓ (not in free version)	✗	✓	✗	✓	✓

Πίνακας 8. Πλατφόρμες Ανάπτυξης

2.2.9. Ανακεφαλαίωση

Με βάση τα αποτελέσματα της συγκριτικής ανάλυσης μπορούμε να βγάλουμε μερικά σημαντικά συμπεράσματα σχετικά με την αποτελεσματικότητα κάθε μηχανής παιχνιδιών για τις κατηγορίες χαρακτηριστικών που διερευνώνται σε αυτή την διπλωματική εργασία.

Παρατηρήθηκε ότι το Game Maker δεν είναι κατάλληλο για την ανάπτυξη παιχνιδιών με τρισδιάστατα γραφικά. Ωστόσο, είναι ιδανικό για την εύκολη δημιουργία παιχνιδιών με 2D γραφικά, παρέχοντας ένα φιλικό προς το χρήστη περιβάλλον και τη δυνατότητα ανάπτυξης παιχνιδιών χωρίς καμία εμπειρία στον προγραμματισμό.

Το JMonkey υπολείπεται ορισμένων πτυχών της οπτικοακουστικής πιστότητας, όπως η απουσία occlusion culling, animation και τρισδιάστατων εργαλείων ήχου. Επιπλέον, δεν μπορεί να εξάγει παιχνίδια σε κονσόλες. Από την άλλη πλευρά, είναι μια μηχανή παιχνιδιών ανοιχτού κώδικα, επιτρέποντας στους χρήστες να επεκτείνουν τη λειτουργικότητά του.

Το Ogre3D, όπως και το JMonkey, δεν υποστηρίζει πολλά χαρακτηριστικά της οπτικοακουστικής και λειτουργικής κατηγορίας πιστότητας. Από την άλλη, το Ogre3D είναι επίσης μια μηχανή παιχνιδιών ανοιχτού κώδικα και ως εκ τούτου μπορεί να προσαρμοστεί στις ανάγκες ενός έμπειρου χρήστη. Επιπλέον, το Ogre3D είναι ιδανικό για δημιουργία σκιών και είναι κατάλληλο για αρχάριους που παρέχοντας αρκετά καλή υποστήριξη.

Το Sio2 είναι κατάλληλο για τη δημιουργία εφαρμογών σε λογισμικό iOS, καθώς ο πυρήνας του βασίζεται σε αυτό το λειτουργικό σύστημα. Είναι ανεπαρκές σε πολλά χαρακτηριστικά οπτικοακουστικής και λειτουργικής πιστότητας. Επίσης, δεν είναι τόσο

προσιτό στους χρήστες, επειδή παρόλο που προηγουμένως προσφερόταν δωρεάν και ως μηχανή ανοιχτού κώδικα, τώρα είναι η πιο ακριβή μηχανή παιχνιδιών.

Το Unity και η Unreal Engine 4 είναι οι πιο ισχυρές μηχανές παιχνιδιών, οι οποίες υποστηρίζουν σχεδόν όλες τις λειτουργίες που περιλαμβάνονται στο πλαίσιο που χρησιμοποιούνται. Η Unreal Engine 4 παρουσιάζει καλύτερη οπτικοακουστική απόδοση από το Unity, καθώς υποστηρίζει περισσότερες δυνατότητες ενώ ταυτόχρονα είναι μια μηχανή ανοιχτού κώδικα.

2.3. Unity Εναντίων Unreal

Το Unity και η Unreal Engine 4 επιλέχθηκαν από εμάς για περαιτέρω σύγκριση, επειδή είναι σαφώς οι δύο πιο ανεπτυγμένες μηχανές παιχνιδιών. Στο πλαίσιο της διπλωματικής εργασίας αναζητήσαμε δύο παρόμοια παιχνίδια shooter για υπολογιστές και κινητές συσκευές, ώστε να αξιοποιηθούν καλύτερα τα διάφορα χαρακτηριστικά των δύο μηχανών. Συγκεκριμένα, το Survival Shooter Game [7] για το Unity και το Twin Stick Shooter [8] στην Unreal Engine 4.

Τα χαρακτηριστικά που εξετάστηκαν ήταν η χρηστικότητα της μηχανής παιχνιδιών, η καμπύλη εκμάθησης, η διαδικασία εξαγωγής για κινητές συσκευές, καθώς και η ποιότητα και το μέγεθος της εφαρμογής του παιχνιδιού.

2.3.1. Ευχρηστία

Όσον αφορά την καμπύλη χρηστικότητας και εκμάθησης τις μηχανής παιχνιδιού, το Unity φαίνεται να είναι καλύτερο όσον αφορά μια φιλική προς το χρήστη διεπαφή, καθώς επίσης προσφέρει και δωρεάν assets για την ανάπτυξη παιχνιδιών, ενώ έχει λιγότερες απαιτήσεις για υλικό. Ειδικότερα:

- Το Unity και η Unreal Engine 4 ενσωματώνουν και τα δύο απαραίτητα εργαλεία για την ανάπτυξη παιχνιδιών, αλλά η Unreal Engine 4 έχει μια πιο περίπλοκη διεπαφή σε αντίθεση με το Unity που προσφέρει όλα τα απαραίτητα εργαλεία σε ένα μόνο παράθυρο.
- Στο Unity ο χρήστης πρέπει να έχει γνώση (ή να μάθει) της γλώσσας αντικειμενοστραφή προγραμματισμού C# για τον προγραμματισμό των απαραίτητων σεναρίων. Από την άλλη πλευρά, η Unreal Engine χρησιμοποιεί ένα σύστημα Blueprint

Visual Scripting που στηρίζεται σε μια διεπαφή που βασίζεται σε κόμβο για τον καθορισμό αντικειμένων και κλάσεων. Αυτός ο επεξεργαστής γραφικών μπορεί να βοηθήσει κάποιον που δεν ασχολείται με τον προγραμματισμό ή έναν αρχάριο σε αυτό το πεδίο, αλλά για πολλούς προγραμματιστές μπορεί να είναι προτιμότερο να γράφετε σε πηγαίο κώδικα.

- Και οι δύο μηχανές έχουν μια ευρεία κοινότητα με πολλά εκπαιδευτικά προγράμματα. Στο Unity τα περισσότερα από αυτά είναι σε μορφή κειμένου, ενώ στην Unreal Engine 4 σε μορφή βίντεο. Ωστόσο, στην Unreal Engine 4 δεν υπάρχουν τόσα πολλά δωρεάν εκπαιδευτικά προγράμματα.
- Και οι δύο μηχανές ανάπτυξης παιχνιδιών έχουν ένα μεγάλο ηλεκτρονικό κατάστημα με assets, αλλά στην Unreal Engine 4 δεν υπάρχουν πολλά δωρεάν assets σε αντίθεση με το Unity.
- Τέλος, το Unity έχει μικρότερες απαιτήσεις σε υλικό.

2.3.2. Εξαγωγή Παιχνιδιού για Κινητές Συσκευές

Η διαδικασία εξαγωγής του παιχνιδιού σε κινητά τηλέφωνα είναι αρκετά απλή και στις δύο μηχανές ανάπτυξης παιχνιδιών. Μερικές σημειώσεις είναι οι εξής:

- Στο Unity, δεν υποστηρίζεται το normal mapping στην πραγματικότητα, όπως υποδηλώνεται στη συγκριτική ανάλυση των μηχανών ανάπτυξης παιχνιδιών. Ωστόσο, υπάρχουν πολλά διαθέσιμα shaders που μπορούν να χρησιμοποιηθούν για την αποφυγή σφαλμάτων. Ένα άλλο πρόβλημα που μπορεί να εμφανιστεί είναι ένα προβληματικό αρχείο κώδικα για τον χειρισμό των χαρακτήρων από το assets store. Αυτό το πρόβλημα μπορεί να αντιμετωπιστεί γράφοντας κώδικα για τον προγραμματισμό του αρχείου για τον χειρισμό χρησιμοποιώντας τα υπάρχοντα εκπαιδευτικά προγράμματα.
- Στην Unreal Engine 4 δεν υπάρχει ποικιλία mobile shaders και παρουσιάζονται συχνά σφάλματα. Τα shaders πρέπει να τροποποιούνται χρησιμοποιώντας το πρόγραμμα επεξεργασίας υλικού, ενώ πρέπει να γίνονται νέες ρυθμίσεις για τον φωτισμό. Επιπλέον, δεν υπάρχει δυνατότητα προσομοίωσης και ο χρήστης πρέπει είτε να χρησιμοποιήσει ένα πρόγραμμα προεπισκόπησης για κινητά είτε να ξεκινήσει το παιχνίδι ενώ αναπτύσσεται. Τέλος, η εγκατάσταση δεν είναι απλή και απαιτεί την προετοιμασία και εκτέλεση του κατάλληλου αρχείου bat χρησιμοποιώντας υπολογιστή.

2.3.3. Ποιότητα και Μέγεθος Παιχνιδιού

Κάνοντας μια αναζήτηση στα Προφίλ ενός Παιχνιδιού καθώς και παρακολούθηση του αποτυπώματος της μνήμης της συσκευής κατά τη διάρκεια του παιχνιδιού, παρατηρήσαμε τις ακόλουθες πληροφορίες για την ποιότητα και το μέγεθος του τελικού παιχνιδιού. Η συσκευή που χρησιμοποιήθηκε είχε τα ακόλουθα χαρακτηριστικά: Android 4.4, RAM 1GB, ROM 8GB, 1.2GHz Quad Core. Τα αποτελέσματα συνοψίζονται στους Πίνακες 11 και 12.

	Unity	Unreal Engine 4
Προβλήματα μνήμης συσκευών	Όχι	Ναι
Αποτύπωμα μνήμης	Μικρότερο από την Unreal Engine	Τέσσερις φορές μεγαλύτερο από το Unity
Profiler		
Draw Calls	28	27
Μνήμη	Light 1.3 KB	Light 16 KB
Χρόνος	Animation σε μέγιστο χρόνο 2.43ms και μέσο χρόνο < 2ms	Animation σε 8.03ms και μέσο χρόνο < 1.43ms

Πίνακας 9. Ποιότητα Παιχνιδιού

	Unity	Unreal Engine 4
Αρχικό μέγεθος textures	84.8 MB	54.8 MB
Συνολικό μέγεθος παιχνιδιού μετά την συμπίεση	63.9 MB	135 MB
Συμπίεση	Αποτελεσματική Συμπίεση	Πρόβλημα στην συμπίεση παιχνιδιού

Πίνακας 10. Μέγεθος Παιχνιδιού

Στον Πίνακα 10, παρατηρούμε ότι στο Unity η συμπίεση οδηγεί σε ένα παιχνίδι με συνολικό μέγεθος που είναι ακόμη μικρότερο από το αρχικό μέγεθος των textures που χρησιμοποιούνται στο παιχνίδι. Από την άλλη πλευρά, η συμπίεση του παιχνιδιού στην Unreal Engine 4 οδηγεί σε ένα παιχνίδι που έχει στην πραγματικότητα το ίδιο μέγεθος του αντίστοιχου project. Είναι προφανές ότι η συμπίεση παιχνιδιών είναι πολύ πιο αποτελεσματική στο Unity και αυτό είναι πολύ σημαντικό για παιχνίδια που εκτελούνται σε κινητές συσκευές.

2.3.3. Εμπειρία χρήστη

Η εμπειρία των χρηστών ήταν αρκετά διαφορετική στα δύο παιχνίδια. Στον Πίνακα 13 παρουσιάζεται μια περίληψη της εμπειρίας χρήστη και για τις δύο μηχανές ανάπτυξης.

Unity	Unreal Engine 4
Κίνηση με χειριστήριο και shooting με την αφή	Κίνηση και shooting μέσω χειριστηρίου
Κίνηση μόνο με ταυτόχρονη λήψη	Ευκολία στην κίνηση και το shooting ταυτόχρονα
Πιο ακριβής στόχευση	Όχι τόσο ακριβές στο shooting
Δυσκολία σε ταυτόχρονη κίνηση και shooting	Δυσκολία στην χρήση χειριστηρίου μερικές φορές
Πιο ρεαλιστικά γραφικά	

Πίνακας 11. Εμπειρία χρήστη

2.4. Συμπεράσματα και Προτάσεις

Καταλήγοντας, δεν μπορούμε να προτείνουμε μια μηχανή παιχνιδιών τόσο καλύτερη από οποιαδήποτε άλλη, αφού κάθε μηχανή έχει τα πλεονεκτήματα και τα μειονεκτήματά της. Η επιλογή εξαρτάται από το προφίλ και τη γνώση του χρήστη (π.χ. παιδαγωγούς, ειδικούς σε κάποιον τομέα, προγραμματιστές), το αποτέλεσμα που θέλει να επιτύχει, καθώς και τους πόρους και το χρόνο του. Αυτή η διπλωματική εργασία, αναδεικνύει τη βιβλιογραφία στον τομέα παρέχοντας μια ολιστική επισκόπηση των μηχανών παιχνιδιών για παιχνίδια που καλύπτουν διαφορετικά προφίλ χρηστών και ανάγκες. Σε αντίθεση με τις υπάρχουσες συγκριτικές αναλύσεις και κριτικές των μηχανών παιχνιδιών που βασίζονται σε συγκεκριμένες δυνατότητες, όπως η παροχή της δυνατότητας ανάπτυξης παιχνιδιών 3D ή ανοιχτού κώδικα, η συγκριτική μας ανάλυση λαμβάνει υπόψιν αντιπροσωπευτικά παραδείγματα μηχανών παιχνιδιών που εμπίπτουν σε διαφορετικές κατηγορίες που καλύπτουν ένα ευρύ φάσμα απαιτήσεων. Η ανάλυση πραγματοποιείται χρησιμοποιώντας εκτεταμένη λίστα σημαντικών χαρακτηριστικών που έχουν καταγραφεί σε ένα πλαίσιο επιλογής μηχανών παιχνιδιών για σοβαρές εφαρμογές υψηλής πιστότητας [7].

Συγκεκριμένα, η παρούσα διπλωματική στοχεύει:

- Στην υποστήριξη του χρήστη στην επιλογή της σωστής μηχανής, θέτοντας ορισμένα κριτήρια που πρέπει να ληφθούν υπόψιν.
- Στην παρουσίαση μιας ολοκληρωμένης επισκόπησης ενός αριθμού μηχανών που εμπίπτουν σε διαφορετικές κατηγορίες, όπως αυτές που δεν απαιτούν γνώσεις προγραμματισμού, μηχανές που βασίζονται σε δημοφιλείς σύγχρονες τεχνολογίες Ιστού, μηχανές ανοιχτού κώδικα που μπορούν να προσαρμοστούν / επεκταθούν από έμπειρους χρήστες και επαγγελματικές μηχανές παιχνιδιών.

Η ανασκόπηση της βιβλιογραφίας και η συγκριτική ανάλυση των μηχανών παιχνιδιών, απέδειξαν ότι οι κυρίαρχες μηχανές παιχνιδιών είναι το Unity και η Unreal Engine 4. Μπορούμε να συμπεράνουμε ότι οι πληροφορίες που συνοψίζονται για όλες τις μηχανές παιχνιδιών είναι αρκετά αξιόπιστες. Φυσικά για να είμαστε σίγουροι, οι μηχανές παιχνιδιών πρέπει να χρησιμοποιούνται στην πράξη.

Όσον αφορά τις μηχανές παιχνιδιών που έχουν εξεταστεί, τα πιο σημαντικά συμπεράσματα μπορούν να συνοψιστούν ως εξής:

- Το Unity θεωρείται πιο κατάλληλο για αρχάριους επειδή: Έχει απλούστερη διεπαφή χρήστη. Παρέχει πολλά tutorials και παραδείγματα. Υπάρχουν πολλά διαθέσιμα assets. Επιπλέον, η εγκατάστασή του δεν απαιτεί υλικό υψηλών επιδόσεων. Όσον αφορά την ανάπτυξη παιχνιδιών για κινητά Android, η διαδικασία είναι αρκετά απλή και η διαδικασία εξαγωγής είναι πιο εύκολη. Ωστόσο, απαιτείται γνώση της γλώσσας προγραμματισμού C#.
- Η Unreal Engine 4 είναι πιο κατάλληλη για έμπειρους χρήστες, καθώς: υποστηρίζει Visual Scripting και έχει ένα πιο περίπλοκο γραφικό περιβάλλον. Έχει μια πιο απότομη καμπύλη μάθησης. Αυτή η μηχανή απαιτεί υλικό υψηλών επιδόσεων, αλλά από την άλλη πλευρά τα γραφικά του είναι αξιοσημείωτα. Η Unreal Engine 4 είναι κατάλληλη για παιχνίδια επιτραπέζιου υπολογιστή, και παρέχει περισσότερες δυνατότητες για απόδοση γραφικών. Ωστόσο, εντοπίζονται σφάλματα κατά τη διαδικασία μεταφοράς των παιχνιδιών στο Android, ενώ αντιμετωπίζονται δυσκολίες στη συμπίεση του παιχνιδιού, με αποτέλεσμα να υπάρχουν προβλήματα απόδοσης σε συσκευές που έχουν λίγη διαθέσιμη μνήμη.

Είναι σαφές ότι αυτές οι πληροφορίες δεν είναι άκρως αντικειμενικές. Πρώτα απ 'όλα, η συγκριτική ανάλυση περιλάμβανε μικρό αριθμό μηχανών παιχνιδιών σε σύγκριση με τον μεγάλο αριθμό που είναι διαθέσιμοι. Ωστόσο, κάναμε μια προσπάθεια να συμπεριλάβουμε αντιπροσωπευτικά παραδείγματα γνωστών μηχανών παιχνιδιών που εμπίπτουν σε διαφορετικές κατηγορίες, όπως μηχανές παιχνιδιών για αρχάριους και έμπειρους χρήστες, μηχανές παιχνιδιών που βασίζονται σε τεχνολογίες ιστού, μηχανές παιχνιδιών ανοιχτού κώδικα που μπορούν να προσαρμοστούν, μηχανές παιχνιδιών που χρησιμοποιούνται στη βιομηχανία παιχνιδιών και μηχανές που στοχεύουν κυρίως σε κινητές συσκευές. Το κοινό χαρακτηριστικό όλων αυτών που αναλύθηκαν είναι ότι υποστηρίζουν την εξαγωγή ενός παιχνιδιού τουλάχιστον στις δύο πιο χρησιμοποιούμενες πλατφόρμες για κινητά, δηλαδή το Android και το iOS. Αυτή η συγκριτική ανάλυση που βασίζεται σε μια εκτενή λίστα ή χαρακτηριστικά μπορεί να βοηθήσει οποιονδήποτε ενδιαφέρεται στον τομέα να σχεδιάσει μια σαφή εικόνα για τις διάφορες κατηγορίες των υφιστάμενων μηχανών και επίσης να κάνει μια συνετή επιλογή με βάση τις γνώσεις του, τη φύση του παιχνιδιού που θα αναπτυχθεί και τα πιο σημαντικά χαρακτηριστικά του παιχνιδιού που πρέπει να υποστηρίζονται από την επιλεγμένη μηχανή παιχνιδιών.

Ένας άλλος περιορισμός έγκειται στο γεγονός ότι η συγκριτική ανάλυση των μηχανών παιχνιδιών βασίστηκε σε πληροφορίες από τους επίσημους ιστότοπούς τους και τη διαθέσιμη βιβλιογραφία. Καταβλήθηκε μεγάλη προσπάθεια να διασταυρωθούν οι πληροφορίες που παρουσιάζονται από διάφορες πηγές. Επιπλέον, η μελέτη με τις δύο μηχανές παιχνιδιών επιβεβαίωσε τις πληροφορίες που παρουσιάστηκαν στο πλαίσιο της ανάλυσης και αυτή είναι μια καλή ένδειξη ότι η συγκεκριμένη κριτική είναι έγκυρη.

Οι προαναφερθέντες περιορισμοί παρέχουν έναν καλό οδηγό για μελλοντική έρευνα στον τομέα. Η ανασκόπηση της βιβλιογραφίας και η συγκριτική ανάλυση μπορούν να επεκταθούν προκειμένου να συμπεριληφθούν περισσότερες μηχανές παιχνιδιών, οι οποίες θα μπορούσαν επίσης να διερευνηθούν με τη χρήση τους για την ανάπτυξη πρωτότυπων παιχνιδιών. Επιπλέον, τα παιχνίδια μπορούν να εξαχθούν σε άλλες πλατφόρμες εκτός από το Android με σκοπό να εξεταστούν περισσότερα χαρακτηριστικά διαφορετικών μηχανών. Αυτό που θα ήταν ακόμη πιο σημαντικό είναι η ανάπτυξη μιας διαδικτυακής εφαρμογής για την παρουσίαση όλων αυτών των πληροφοριών σε άτομα που ενδιαφέρονται να χρησιμοποιήσουν μηχανές παιχνιδιών.

Αυτή η εφαρμογή θα μπορούσε ακόμη και να υποβάλει προτάσεις στους χρήστες σχετικά με τις καλύτερες επιλογές μηχανών ανάπτυξης για ένα συγκεκριμένο project με βάση τις πληροφορίες που παρέχονται από τους χρήστες σε μια εξειδικευμένη μορφή σχετικά με τις γνώσεις τους πάνω στον προγραμματισμό, τις προτιμήσεις τους για την ανάπτυξη ενός παιχνιδιού (π.χ. κωδικοποίηση, οπτική δέσμη ενεργειών, καθορισμός ενεργειών μέσω μιας διεπαφής μεταφοράς και απόθεσης και ούτω καθεξής), τα χαρακτηριστικά που θεωρούνται σημαντικά για τις πλατφόρμες παιχνιδιών, ανάπτυξης και ανάπτυξής τους.

2.5. Γιατί Επιλέξαμε το Unity

Στην παρούσα διπλωματική εργασία, με βάση την παραπάνω σύγκριση των χαρακτηριστικών που διαθέτει η κάθε μηχανή ανάπτυξης, επιλέξαμε το Unity για την δημιουργία του δικού μας 2D παιχνιδιού, έχοντας υπόψιν τις λειτουργίες και τα χαρακτηριστικά που θα εξυπηρετούσαν τις ανάγκες του project. Συγκεκριμένα, ένας από τους παράγοντες επιλογής ήταν η ύπαρξη της γλώσσας αντικειμενοστραφούς προγραμματισμού C#, στην οποία διαθέταμε μια μεγαλύτερη οικειότητα. Επιπροσθέτως, η μεγάλη ποικιλία από δωρεάν παραδείγματα και tutorials όσον αφορά το ξεκίνημα πάνω στην ανάπτυξη βιντεοπαιχνιδιών καθώς και η ύπαρξη ενός asset store το οποίο περιέχει τεράστιο αριθμό από δωρεάν assets αλλά και επί πληρωμή, έπαιξαν πολύ καθοριστικό ρόλο. Ακόμα, κάποια χαρακτηριστικά που μας οδήγησαν στην επιλογή του, ήταν η απλούστερη διεπαφή χρήστη, καθώς επίσης και οι χαμηλές απαιτήσεις εγκατάστασης, καθιστώντας το καταλληλότερο για την δημιουργία του πρώτου μας βιντεοπαιχνιδιού.

Κεφάλαιο 3ο: Σχεδιασμός Παιχνιδιού

3.1. Βήματα Υλοποίησης Παιχνιδιού

Παρακάτω περιγράφεται σε συντομία η σειρά βημάτων που ακολουθήσαμε για τον σχεδιασμό του παιχνιδιού που δημιουργήθηκε στα πλαίσια της παρούσας διπλωματικής.

3.1.1. Στάδιο Σχεδιασμού

- Προσδιορισμός σεναρίου – Ανάλυση τις πλοκής του παιχνιδιού
- Σχεδιασμός των βασικών οθονών (Storyboard) – Δημιουργία σκίτσων με βάση τα επιμέρους στάδια του παιχνιδιού
- Αναζήτηση και δημιουργία βασικών στοιχείων του παιχνιδιού (Game Assets)

3.1.2. Στάδιο Υλοποίησης στο Unity

- Δημιουργία animations κινήσεων του παίκτη και των εχθρών
- Επεξεργασία των στοιχείων του παιχνιδιού με το πρόγραμμα Adobe Photoshop
- Δημιουργία βασικών σκηνών
- Προγραμματισμός των διαφόρων λειτουργιών

3.2. Περιγραφή Σεναρίου Παιχνιδιού

Παρακάτω, θα περιγράψουμε το σενάριο που δημιουργήσαμε όσον αφορά το παιχνίδι μας, παρουσιάζοντάς το σε τρία κεφάλαια τα οποία αποτελούν και την αρχική ιδέα πίσω από το gameplay, τις σκηνές και περιήγηση του χαρακτήρα μέσα στον κόσμο του παιχνιδιού.

3.2.1. Κεφάλαιο 1

Το παιχνίδι διαδραματίζεται σε ένα ψυχρό μέρος που έχει γίνει μια πυρηνική καταστροφή. Ο κόσμος που έχει απομείνει προσπαθεί να επιβιώσει υπό πολύ δύσκολες συνθήκες σε καταφύγια που έχουν δημιουργηθεί προκειμένου να επιζήσουν.

Ο παίκτης ζει σε ένα από τα καταφύγια για αρκετά χρόνια. Λόγω κάποιας βλάβης που έχει προκύψει εκεί, αποφασίζει να αναζητήσει τον τρόπο για να την λύσει στο πλησιέστερο καταφύγιο. Στον δρόμο για αυτό, θα ανακαλύψει ένα πεσμένο αεροπλάνο το οποίο θα

εξερευνήσει. Έξω από το αεροπλάνο θα βρει κάποια αντικείμενα που θα του κινήσουν την περιέργεια, αλλά λόγω του κρύου που επικρατεί θα συνεχίσει τον δρόμο του για τον αρχικό του προορισμό, το κοντινότερο καταφύγιο. Μετά από κάποιες μέρες που θα μείνει εκεί και αφού έχει συλλέξει τα αντικείμενα που χρειαζόταν, παίρνει τον δρόμο της επιστροφής για το αρχικό καταφύγιο. Φτάνοντας εκεί θα διορθώσει το πρόβλημα για το οποίο έφυγε, όσο όμως περνούσαν οι μέρες σκεφτόταν όλο και πιο πολύ αυτά που είχε βρει στο αεροπλάνο με αποτέλεσμα να πάρει την απόφαση, παρά τους κινδύνους που υπήρχαν στην περιοχή, το υπερβολικό κρύο αλλά και την ραδιενέργεια στην ατμόσφαιρα να ξεκινήσει την εξερεύνηση για να ανακαλύψει τι σχέση έχουν το πεσμένο αεροπλάνο καθώς και τα ευρήματα που ανακάλυψε, με την καταστροφή που έχει συμβεί στον κόσμο.

3.2.2. Κεφάλαιο 2

Ο λόγος που τον έκανε να ξεκινήσει το ταξίδι ήταν τα ευρήματα στο αεροπλάνο. Εκεί βρίσκονταν δύο νεκροί άνδρες με μερικά αντικείμενα στην κατοχή τους. Ένας χάρτης της περιοχής στα ρούχα του οδηγού, καθώς και ένα μισοτελειωμένο γράμμα στον δεύτερο άνδρα. Αυτό όπως φαινόταν από το περιεχόμενό του, γράφτηκε λίγο πριν τον θάνατό του κατά την διάρκεια της πτώσης. Μερικά από αυτά που έλεγε είχαν να κάνουν με την περιγραφή ενός σχεδίου που είχε εφαρμοστεί στην περιοχή και πως ο κόσμος που ζούσε εκεί είχε ξεγελαστεί. Επίσης ανέφερε πως για να μαθευτεί η αλήθεια έπρεπε αυτός που θα βρει το γράμμα να πάει στην τοποθεσία που ήταν σημειωμένη στον χάρτη. Κατά πάσα πιθανότητα, ο άνδρας λίγες στιγμές πριν τον θάνατό του αποφάσισε να αποκαλύψει την αλήθεια.

Ο Ted έχοντας τον χάρτη με την τοποθεσία στην κατοχή του και εφόσον μάζεψε τις απαραίτητες προμήθειες για το ταξίδι, ξεκίνησε για την απαγορευμένη περιοχή όπως αυτή αναφερόταν στον χάρτη. Στον δρόμο για αυτήν θα συναντήσει αρκετά εμπόδια αλλά παρά όλες τις δυσκολίες θα καταφέρει να φτάσει στην περιοχή που αναφερόταν στο γράμμα. Εκεί θα ανακαλύψει πως η ατμόσφαιρα είναι μολυσμένη από ραδιενέργεια. Για να μπορέσει να συνεχίσει τον δρόμο του θα πρέπει να χρησιμοποιήσει την μάσκα που βρήκε νωρίτερα στο προηγούμενο καταφύγιο. Βέβαια ο χρόνος που θα έχει θα είναι περιορισμένος για να φτάσει στο πλησιέστερο μέρος που θα είναι ασφαλής και στην συνέχεια στον τελικό του στόχο, την τοποθεσία που του υπέδειξε ο άνδρας στο αεροπλάνο μέσα από το γράμμα που είχε γράψει. Εκεί θα μαζέψει ότι προμήθειες του είναι απαραίτητες για να συνεχίσει το ταξίδι.

Αργότερα θα φύγει από το καταφύγιο, θα συνεχίσει για μερικά ακόμα χιλιόμετρα βρίσκοντας διάφορους εχθρούς στον δρόμο του. Μετά από λίγο θα καταφέρει να φτάσει στο

σημείο που έδειχνε ο χάρτης, εκεί όμως δεν θα αντικρίσει κάτι προφανές και για αυτό το λόγο θα πρέπει να εξερευνήσει το μέρος περισσότερο. Λίγη ώρα αργότερα και αφού είχε ψάξει όλη την περιοχή θα βρει μια μικρή αποθήκη με το σύμβολο που είχε δει για πρώτη φορά στο αεροπλάνο. Πριν μπορέσει να περάσει την είσοδο θα αντικρίσει ένα μεγάλο τέρας που θα τον εμποδίσει το πέρασμά του.

Αφού καταφέρει να νικήσει το τέρας, θα μπει μέσα και εκεί θα ανακαλύψει μια κρυφή είσοδο που θα τον οδηγήσει σε ένα διαφορετικό μέρος. Εκεί θα βρει αυτά που έγραφε ο άνδρας στο αεροπλάνο. Αυτό το δωμάτιο θα έχει διάφορα αντικείμενα που υποδηλώνουν ότι πρόκειται για χώρο μιας μυστικής οργάνωσης, καθώς υπήρχαν σύμβολα, χάρτες της περιοχής, έγγραφα, αλλά και όπως φαινόταν, η περιγραφή ενός σχεδίου που έδειχνε ένα πυρηνικό εργοστάσιο έξω από την πόλη. Αφού θα διαβάσει τα έγγραφα θα αρχίσει να κατανοεί το σχέδιο της μυστικής αυτής οργάνωσης. Αυτό είχε να κάνει με την καταστροφή της μικρής πόλης του Ted, έτσι ώστε να την αποκόψει από τον υπόλοιπο κόσμο και να δημιουργήσει ένα μηχάνημα όπου θα καταφέρνει να ελέγξει όλα τα πυρηνικά εργοστάσια ανά τον κόσμο με σκοπό να πετύχει το ίδιο αποτέλεσμα με αυτή την μικρή πόλη. Το σχέδιο είχε να κάνει με την καταστροφή του σύγχρονου τρόπου ζωής της κοινωνίας, με τα άτομα της οργάνωσης που θα καταφέρουν να επιζήσουν στα καταφύγια που έχουν δημιουργήσει, να αρχίσουν να δημιουργούν έναν νέο τρόπο διαβίωσης μακριά από την τεχνολογία, καθώς η άποψή τους είναι πως καταστρέφει την ελεύθερη βούληση του ανθρώπου. Καθώς όλα αυτά που έμαθε ο Ted τον σόκαραν αφού κατάλαβε πως θα πεθάνουν πολλοί συνάνθρωποί του και επίσης ξεγελάστηκαν όλα αυτά τα χρόνια από μια μυστική οργάνωση, αποφασίζει να συνεχίσει τον δρόμο του για το πυρηνικό εργοστάσιο το οποίο φαινόταν στα σχέδια που ανακάλυψε σε μια μυστική τοποθεσία. Ο σκοπός του ήταν να αποτρέψει το μηχάνημα να καταστρέψει τον κόσμο.

3.3.3. Κεφάλαιο 3

Η πορεία που θα έπρεπε να ακολουθήσει για να φτάσει στο εργοστάσιο θα ήταν μέσα από το δάσος. Πρόκειται για μια πολύ επικίνδυνη διαδρομή καθώς ήταν το πρώτο μέρος που μολύνθηκε από την ραδιενέργεια στην αρχική καταστροφή. Εκεί θα συναντήσει διάφορους εχθρούς, προκλήσεις και ίσως μερικούς γρίφους. Καθώς θα σκοτώσει έναν άνθρωπο της οργάνωσης θα ακούσει από τον ασύρματό του πως έχει ξεκινήσει η αντίστροφη μέτρηση για την καταστροφή των εργοστασίων σε όλο τον κόσμο. Στο τέλος θα καταφέρει να φτάσει στην είσοδο του εργοστασίου όπου θα πρέπει αντιμετωπίσει έναν δυνατό εχθρό για να μπορέσει να μπει στο εσωτερικό του εργοστασίου.

Καθώς θα περάσει την πόρτα, θα συναντήσει τον αρχηγό της μυστικής οργάνωσης ο οποίος θα τον περιμένει και θα του πει διάφορες ιδέες για να υποστηρίξει το σχέδιο του. Ο Ted θα αποφασίσει να τον σταματήσει και να καταστρέψει το μηχάνημα, ο αρχηγός θα αποχωρήσει στην αίθουσα του μηχανήματος δίνοντας εντολή στους ακόλουθούς του να τον σκοτώσουν. Αυτός θα τους αντιμετωπίσει και θα καταφέρει να τους εξουδετερώσει. Θα προχωρήσει λίγο στον χώρο του εργοστασίου και θα μπει στο δωμάτιο που βρίσκεται το μηχάνημα που θα καταστρέψει τον κόσμο. Εκεί θα βρει τον αρχηγό της οργάνωσης που θα προσπαθήσει να το προστατέψει και θα ξεκινήσει η τελική μάχη. Αφού ο Ted καταφέρει να τον νικήσει θα πρέπει να βάλει τον κωδικό αυτοκαταστροφής του εργοστασίου για να μπορέσει να σώσει τον κόσμο. Ξέρει πως αν το κάνει αυτό η πόλη του θα καταστραφεί ολοσχερώς. Θα πάρει όμως την απόφαση να το κάνει αφού θα επιλέξει την σωτηρία της ανθρωπότητας.

3.4. Σχεδιασμός Βασικών Οθονών (Storyboard)

Παρακάτω παρουσιάζονται οι βασικές οθόνες (Storyboard) του παιχνιδιού βάση του παραπάνω σεναρίου.

Εικόνα 3. Storyboard

Εικόνα 4. Storyboard

Εικόνα 5. Storyboard

3.5. Χειρισμός Παιχνιδιού

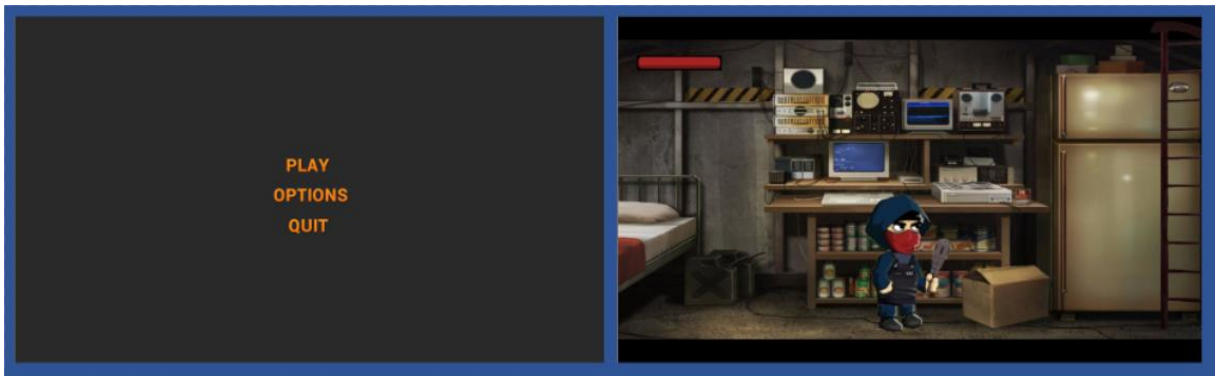
Για την χρήση του βασικού μενού του παιχνιδιού ο χρήστης χρησιμοποιεί το ποντίκι για να κάνει την επιλογή του. Κατά την διάρκεια όμως του παιχνιδιού χρησιμοποιεί αποκλειστικά το πληκτρολόγιο.

Τα πλήκτρα χειρισμού αποτελούνται από τα παρακάτω:

- Πάνω βελάκι → Άλμα
- Δεξί/Αριστερό βελάκι → Κίνηση μπροστά/πίσω
- Spacebar → Επίθεση
- Esc → Σταμάτημα του παιχνιδιού (pause game)

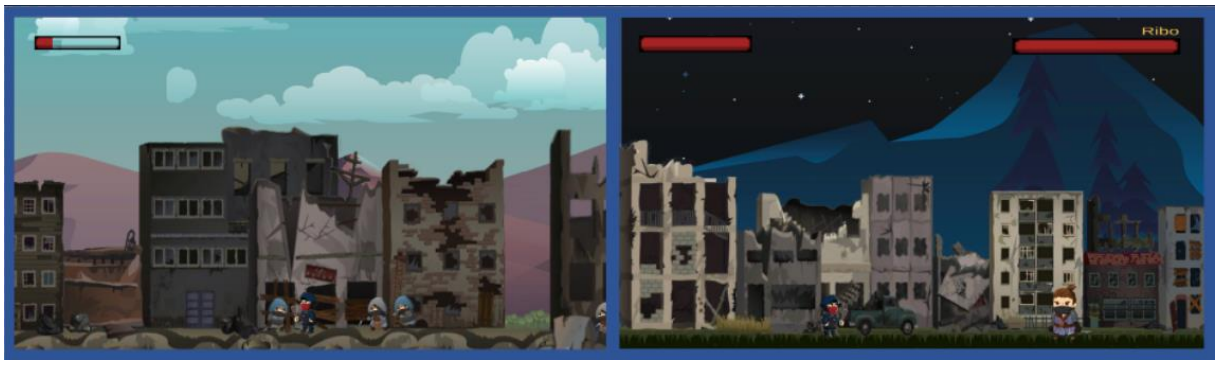
3.6. Σκηνές Παιχνιδιού

Κατά την εκκίνηση της εφαρμογής, εμφανίζεται το αρχικό μενού, με επιλογές για έναρξη του παιχνιδιού, ρυθμίσεις καθώς και έξοδο στην επιφάνεια εργασίας. Δίνεται η επιλογή στον παίκτη, να ξεκινήσει από την αρχή ή να συνεχίσει από το σημείο που σταμάτησε. Αν επιλεγεί το New Game, θα εμφανιστεί η πρώτη σκηνή η οποία είναι το καταφύγιο. Εκεί, πατώντας τα πλήκτρα της κίνησης ο παίκτης μπορεί να περιηγηθεί στον χώρο. Φτάνοντας στο σημείο που βρίσκεται η σκάλα, πηγαίνει στην πόλη η οποία αποτελεί και την δεύτερη σκηνή του παιχνιδιού.



Εικόνα 6. Αρχικό Μενού και Πρώτη Σκηνή

Από αυτή την σκηνή και μετά, δίνεται η δυνατότητα να εφαρμοστεί το σύστημα craft για δημιουργία αντικειμένων που θα χρησιμεύσουν στην συνέχεια. Αυτοί οι μηχανισμοί θα μπορούν να εφαρμοστούν και στις υπόλοιπες τέσσερις σκηνές σε συνδυασμό με το σύστημα μάχης για την αντιμετώπιση των εχθρών.



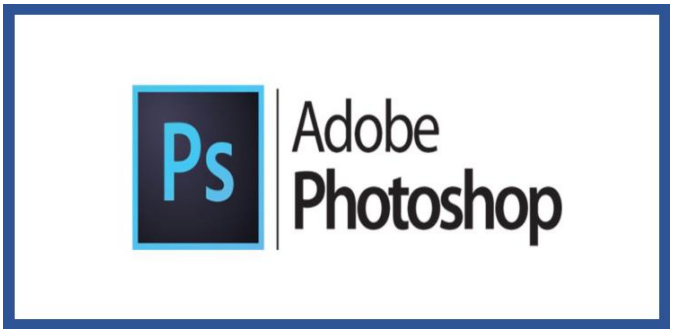
Εικόνα 7. Δεύτερη και Τρίτη Σκηνή

Εικόνα 8. Τέταρτη και Πέμπτη Σκηνή

3.7. Ανάλυση Προγραμμάτων

3.7.1. Adobe Photoshop

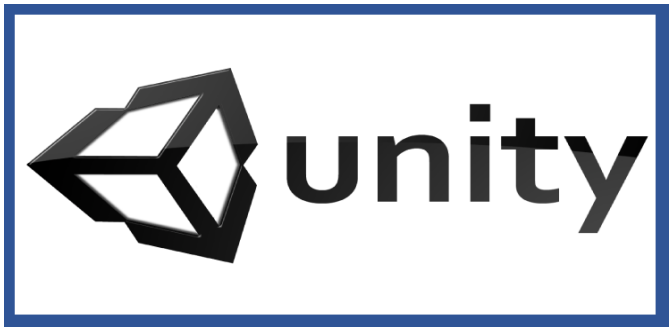
Ένα πρόγραμμα επεξεργασίας εικόνας. Χρησιμοποιήθηκε κυρίως για την επεξεργασία και την διόρθωση των assets.



Εικόνα 9. Adobe Photoshop

3.7.2. Unity

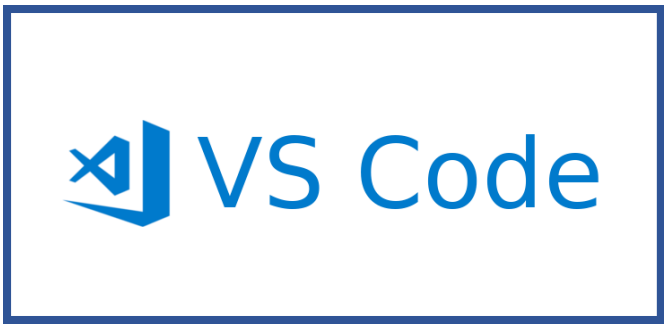
Το Unity είναι μια δωρεάν μηχανή γραφικών που χρησιμοποιείται για την δημιουργία παιχνιδιών για υπολογιστές, κονσόλες κινητά και ιστοσελίδες. Η πρώτη έκδοση βγήκε το 2005 και από τότε κάθε χρόνο βγαίνουν νέες εκδόσεις και προστίθενται συνέχεια νέες δυνατότητες. Το Unity είναι κατάλληλο τόσο για την δημιουργία τρισδιάστατων όσο και δισδιάστατων παιχνιδιών. Η μεγάλη δύναμή του έγκειται στην ευκολία να δημιουργήσεις ένα παιχνίδι μια φορά και να το τρέξεις εύκολα σε όποια κονσόλα ή υπολογιστή θέλεις.



Εικόνα 10. Unity Engine

3.7.3 Visual Studio Code

Το Visual Studio Code είναι ένας δωρεάν επεξεργαστής πηγαίου κώδικα που δημιουργήθηκε από τη Microsoft για Windows, Linux και macOS. Οι λειτουργίες του περιλαμβάνουν υποστήριξη για εντοπισμό σφαλμάτων, επισήμανση σύνταξης, έξυπνη ολοκλήρωση κώδικα, αποσπάσματα, αναδιαμόρφωση κώδικα και ενσωματωμένο Git. Οι χρήστες μπορούν να αλλάξουν το θέμα, τις συντομεύσεις πληκτρολογίου, τις προτιμήσεις και να εγκαταστήσουν επεκτάσεις που προσθέτουν επιπλέον λειτουργικότητα.



Εικόνα 11. Visual Studio Code

Κεφάλαιο 4ο: Υλοποίηση

Στο κεφάλαιο αυτό, θα αναλύσουμε τις ενότητες που ακολουθήσαμε και θα παρουσιάσουμε τον πηγαίο κώδικα που χρησιμοποιήσαμε σε κάθε μια από αυτές.
Συγκεκριμένα, θα μελετήσουμε τα παρακάτω:

- Assets & Animations
- Λειτουργίες Κύριου Χαρακτήρα & Εχθρών
- Δημιουργία Menu
- Λειτουργίες Σκηνών
- Δημιουργία Health Bar Παίκτη και Εχθρών
- Λειτουργία Παύσης Παιχνιδιού
- Λειτουργία Game Over
- Δημιουργία & Ενσωμάτωση Sound Effects
- Λειτουργίες Κέρσορα
- Δημιουργία Credits

4.1. Assets

Για την δημιουργία των σκηνών χρησιμοποιήσαμε το asset store που παρέχεται από το Unity καθώς επίσης και την ιστοσελίδα “CraftPix.net”. Πιο ειδικά, όσον αφορά τους εχθρούς του παιχνιδιού ενσωματώσαμε το “Mighty Heroes (Rogue) 2D Fantasy Characters Pack”. Για την επεξεργασία και εμφάνιση κειμένου στις σκηνές του Game Over και του Pause, χρησιμοποιήθηκε το “Textmesh Pro”. Τα backgrounds των κύριων σκηνών δημιουργήθηκαν από τα assets packs που ακολουθούν: “Free Forest Pack”, “Free Asset – 2D Handcrafted Art”, “Free War Pixel Art 2D Backgrounds”, “War 2D Game Backgrounds”, “Destroyed City Parallax Backgrounds”, “Free Horizontal 2D Game Backgrounds”, “Free Pixel Art Forest”.

4.2. Σκηνές

Στην συγκεκριμένη ενότητα θα εξετάσουμε τις λειτουργίες που εφαρμόσαμε κατά την δημιουργία των σκηνών, με σκοπό την σωστή αλληλεπίδραση όλων των αντικειμένων τα οποία βρίσκονται σε αυτές μεταξύ τους.

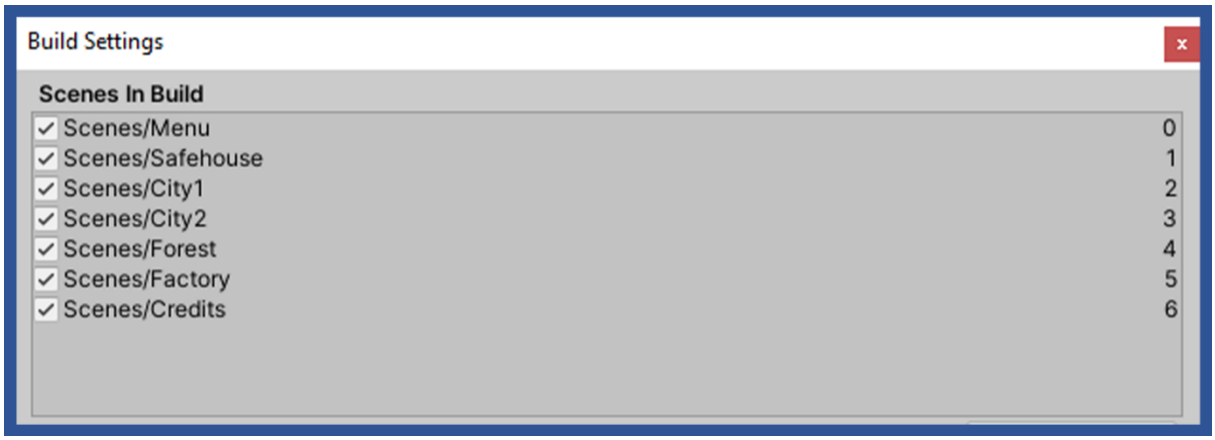
4.2.1. LevelLoader

Η λειτουργία του “LevelLoader” έχει σκοπό κατά την ολοκλήρωση τις κάθε σκηνής, οπότε το ορίζουν οι developers να γίνεται μετάβαση στην επόμενη σκηνή. Για να συμβεί αυτό συμμετέχει και η ιδιότητα “TriggerEnd” που θα εξηγηθεί παρακάτω.

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

public class LevelLoader : MonoBehaviour
{
    private int currentSceneIndex;
    private int sceneToContinue;
    public GameObject completeLevelUI;

    public void CompleteLevel() {
        completeLevelUI.SetActive(true);
    }

    public void LoadNextLevel() {
        currentSceneIndex = SceneManager.GetActiveScene().buildIndex+1;
        PlayerPrefs.SetInt("SavedScene", currentSceneIndex);
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void ContinueGame() {
        sceneToContinue = PlayerPrefs.GetInt("SavedScene");

        if(sceneToContinue != 0)
            SceneManager.LoadScene(sceneToContinue);
        else
            return;
    }
}
```

Source Code 1. Script “LevelLoader”

Στην συνάρτηση “CompleteLevel()” γίνεται ενεργοποίηση του UI της εναλλαγής σκηνής, ενώ στην συνάρτηση “LoadNextLevel()” γίνεται η μεταφορά στην επόμενη σκηνή με το “buildIndex” το οποίο είναι μια λειτουργία του unity που επιτρέπει στους προγραμματιστές να δηλώνουν ποιες σκηνές θα χρησιμοποιηθούν. Η συνάρτηση “ContinueGame()” δίνει την δυνατότητα επαναφοράς του παιχνιδιού στην σκηνή που βρισκόταν ο παίκτης μετά από έξοδο στο αρχικό μενού.



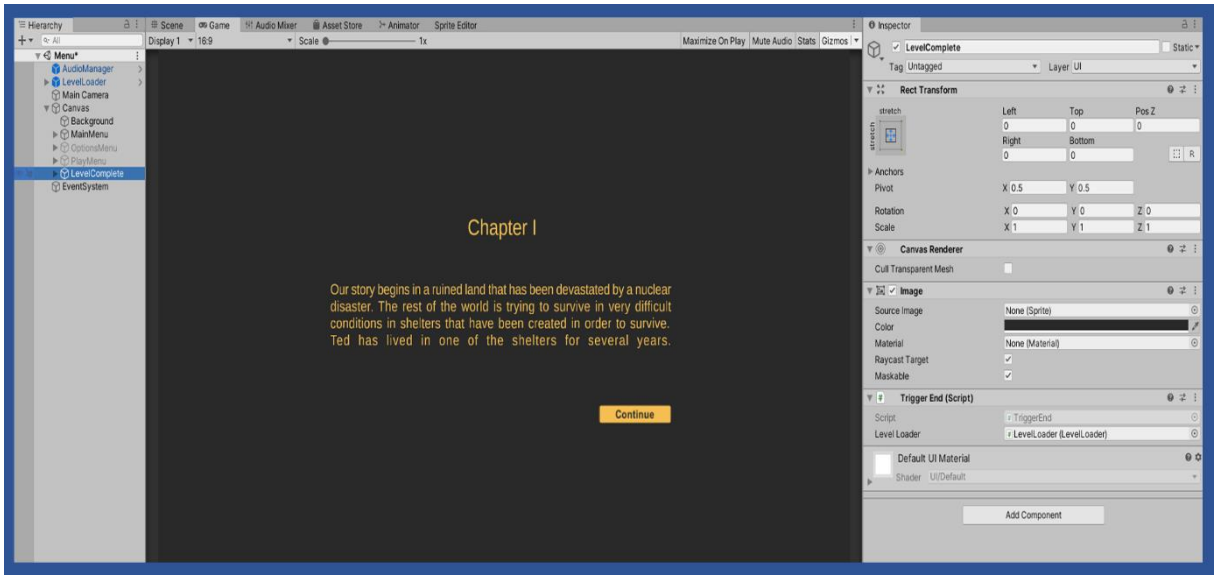
Εικόνα 12. BuildIndex

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

public class TriggerEnd : MonoBehaviour
{
    public LevelLoader levelLoader;
    public static bool GameIsPaused = false;

    void OnTriggerEnter2D(){
        Cursor.visible = true;
        levelLoader.CompleteLevel();
        Pause();
    }

    public void Resume () {
        Time.timeScale = 1f;
        GameIsPaused = false;
        levelLoader.LoadNextLevel();
    }

    public void Pause () {
        Time.timeScale = 0f;
        GameIsPaused = true;
    }
}
```

Source Code 2. Script "TriggerEnd"

Το Script “TriggerEnd” δίνει την δυνατότητα σε ένα αντικείμενο που δημιουργείται μέσω του Unity και τοποθετείται σε ένα συγκεκριμένο σημείο μέσα στις σκηνές να ενεργοποιείται όταν το διαπερνά οποιοδήποτε αντικείμενο, στην δικιά μας περίπτωση ο κύριος χαρακτήρας.

Πιο συγκεκριμένα στον κώδικα, η συνάρτηση “OnTriggerEnter2D()” χρησιμοποιώντας το “LevelLoader” που η λειτουργία του εξετάστηκε παραπάνω, ενεργοποιεί την σκηνή που μεσολαβεί αναμεσά από τις κυρίες σκηνές, η οποία περιγράφει το σενάριο και κάνει παύση του παιχνιδιού με την συνάρτηση “Pause()”, μέχρι ο χρήστης να επιλέξει να μεταβεί στην επόμενη σκηνή μέσω του κουμπιού “Continue”. Όταν το κουμπί πατηθεί, καλείται η συνάρτηση “Resume()” η οποία κάνει ψευδή την λειτουργία παύσης και στην συνέχεια μέσω της συνάρτησης “LoadNextLevel()” η οποία βρίσκεται στο “evelLoader” πραγματοποιεί την αλλαγή σκηνών.



Εικόνα 13. TransitionScene

4.2.2. CameraFollow

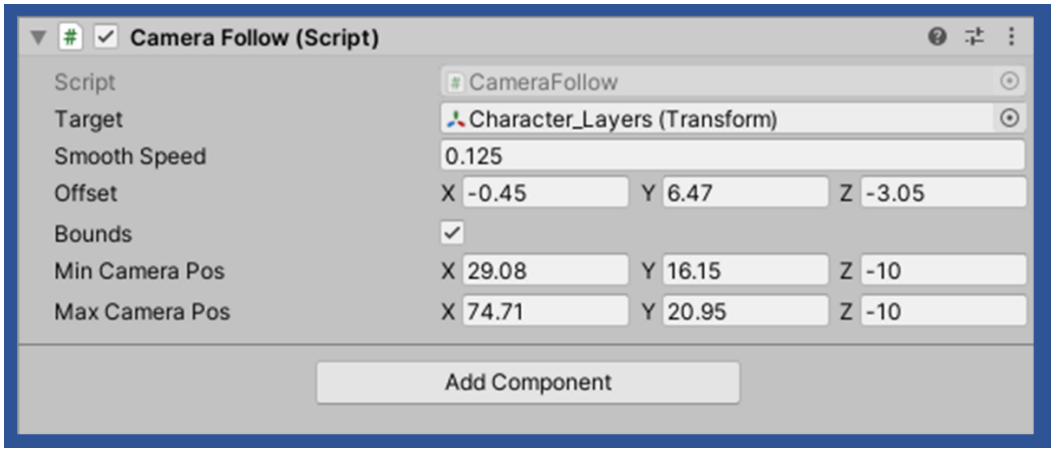
Η λειτουργία CameraFollow δίνει την δυνατότητα στην κάμερα που προσφέρεται ως ενσωματωμένο εργαλείο του Unity, να ακολουθεί τον κύριο χαρακτήρα όπου και αν βρίσκεται μέσα στις σκηνές.

```
using UnityEngine;

public class CameraFollow : MonoBehaviour {
    public Transform target;
    public float smoothSpeed = 0.125f;
    public Vector3 offset;
    public bool bounds;
    public Vector3 minCameraPos;
    public Vector3 maxCameraPos;

    void FixedUpdate() {
        Vector3 desiredPosition = target.position + offset;
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
        transform.position = smoothedPosition;

        if (bounds) {
            transform.position = new Vector3(Mathf.Clamp(transform.position.x, minCameraPos.x, maxCameraPos.x),
                Mathf.Clamp(transform.position.y, minCameraPos.y, maxCameraPos.y),
                Mathf.Clamp(transform.position.z, minCameraPos.z, maxCameraPos.z));
        }
    }
}
```

Source Code 3. Script "CameraFollow"

Μέσω του κώδικα σε αυτό το Script, δημιουργούνται οι απαραίτητες μεταβλητές που θα χρησιμοποιηθούν ως όρια για την παραμονή της κάμερας στην επιθυμητή θέση μέσα στην σκηνή, καθώς επίσης ορίζεται και η ταχύτητα με την οποία θα κινείται. Τέλος, δημιουργείται και η μεταβλητή "target" που θα εξηγηθεί παρακάτω.

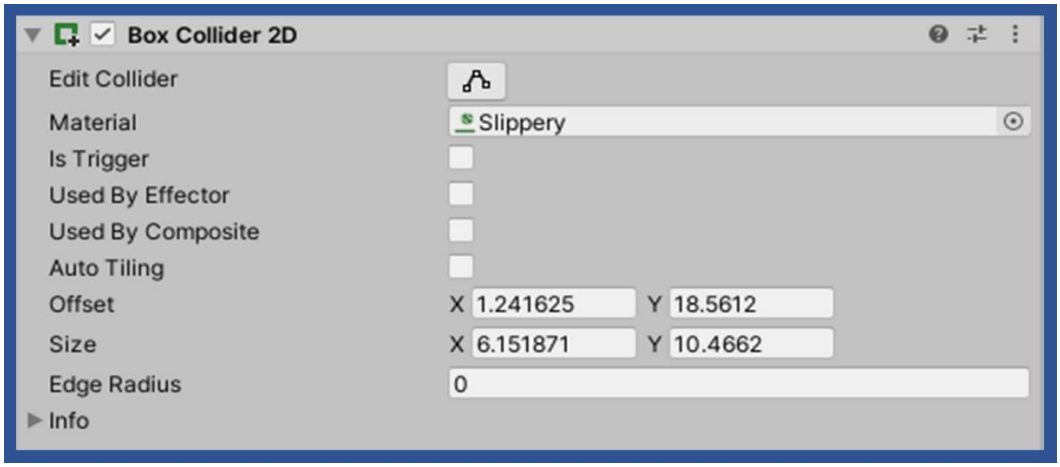
Η συνάρτηση "FixedUpdate()" η οποία αποτελεί μια built-in συνάρτηση του Unity, προσφέρεται για να χρησιμοποιηθεί για τις ανάγκες των developers δίνοντας την δυνατότητα να καλείται από μόνη της. Στην δική μας περίπτωση, χρησιμοποιείται η μεταβλητή "target" για να εντοπιστεί η θέση του χαρακτήρα και περνώντας τις τιμές στην δομή ελέγχου που ακολουθεί, οι οποίες έχουν οριστεί στο Unity όπως φαίνεται στην «Εικόνα 14», δίνεται η δυνατότητα στην κάμερα να βρίσκεται πάντα πάνω στον κύριο χαρακτήρα του παιχνιδιού.



Εικόνα 14. CameraFollow Properties

4.2.3. Box Collider 2D

Η ιδιότητα του Unity Box Collider 2D, χρησιμοποιεί ένα αόρατο σχήμα με σκοπό τον χειρισμό φυσικών συγκρούσεων μεταξύ αντικειμένων. Ένα collider δεν χρειάζεται να έχει ακριβώς το ίδιο σχήμα με τα αντικείμενα για τα οποία θα χρησιμοποιηθεί.



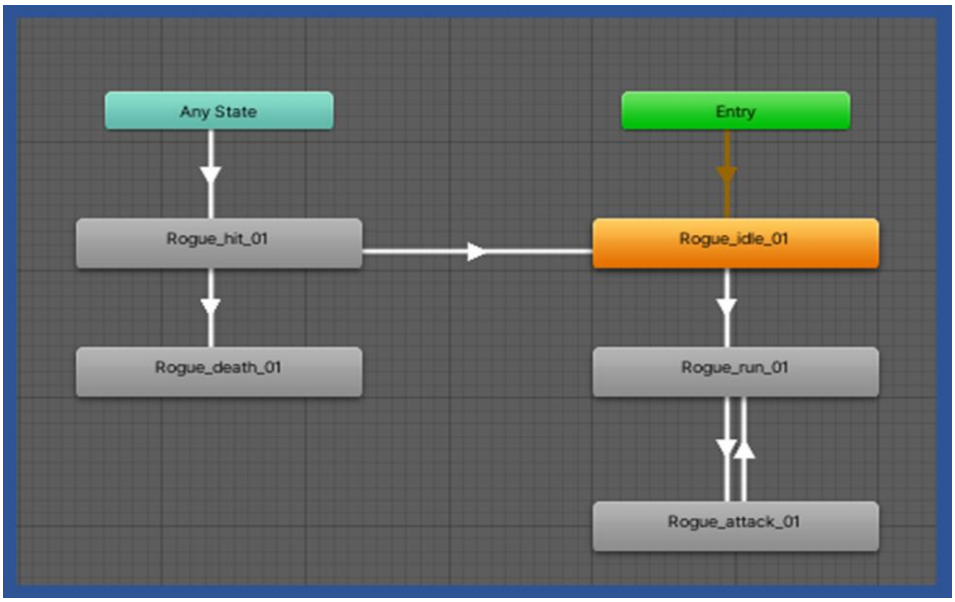
Εικόνα 15. Box Collider 2D

4.3. Εχθροί

Όπως είπαμε παραπάνω, τα Assets και τα Animations των εχθρών τα επιλέξαμε από το Asset Store του Unity. Παρακάτω θα αναλύσουμε την δομή που εφαρμόσαμε για την προσαρμογή τους στις ανάγκες του δικού μας project.

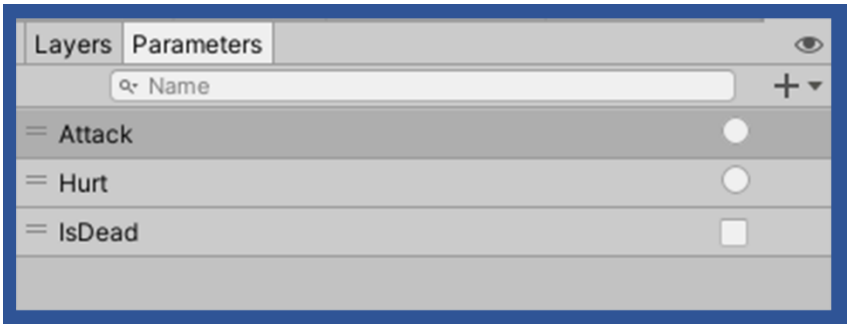
4.3.1. Animator Controller

Ο Animator Controller επιτρέπει την οργάνωση και την διατήρηση ενός συνόλου από κλιπ Animations και των σχετικών Animation Transitions για έναν χαρακτήρα η αντικείμενο. Στις περισσότερες περιπτώσεις είναι φυσιολογικό να υπάρχουν πολλά Animations και να εναλλάσσονται μεταξύ τους όταν συμβαίνουν ορισμένες καταστάσεις κατά την διάρκεια ενός παιχνιδιού. Για παράδειγμα, θα μπορούσε να γίνει μετάβαση από ένα Walk Animation κλιπ σε ένα Jump Animation κλιπ όταν πατηθεί το πλήκτρο διαστήματος.



Εικόνα 16. Animator Controller Εχθρών

Όπως φαίνεται στην «Εικόνα 16» αυτό που χρειάστηκε να κάνουμε στην περίπτωση των Animations των εχθρών ήταν να χρησιμοποιήσουμε την λειτουργία του Animator Controller η οποία περιγράφεται παραπάνω για να δημιουργηθούν οι μεταβάσεις μεταξύ των Animatıon κλιπ. Για παράδειγμα από την κατάσταση αδράνειας του χαρακτήρα "Rogue_idle_01", να μπορεί να γίνει μετάβαση στις καταστάσεις τρεξίματος "Rogue_run_01" και λήψης χτυπήματος "Rogue_hit_01". Με τον ίδιο τρόπο έγιναν οι συσχετίσεις μεταξύ των υπόλοιπων Animations με αποτέλεσμα να δημιουργηθεί η τελική μορφή κινήσεων των εχθρών.

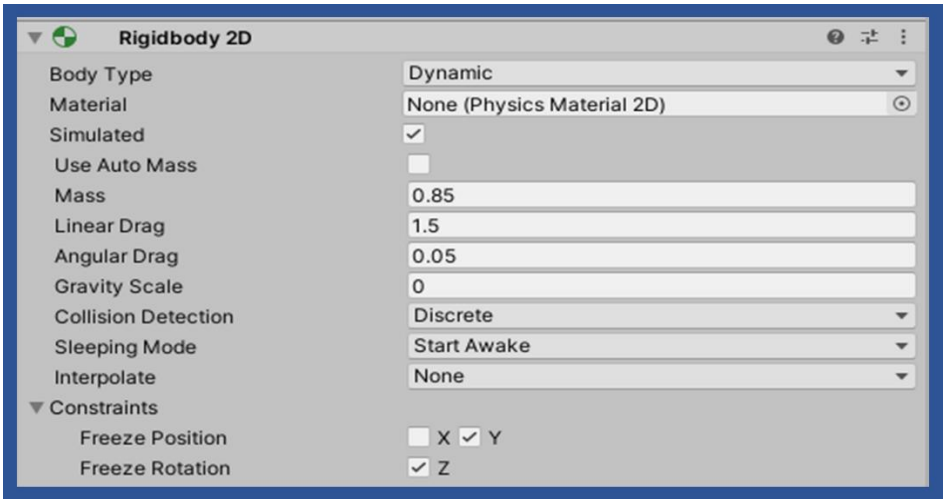


Εικόνα 17. Παράμετροι Animator Controller

Πιο τεχνικά, ορίσαμε κάποιες παραμέτρους στα Animations οι οποίες αποτελούν μεταβλητές που εφαρμόζονται σε έναν Animator Controller και μπορούν να δεχτούν τιμές μέσα από ένα Script κώδικα. Αυτός είναι ο τρόπος με τον οποίο ένα Script μπορεί να ελέγξει ή να επηρεάσει τη ροή του State Machine.

4.3.2. Rigidbody 2D

Η ιδιότητα Rigidbody 2D τοποθετεί ένα αντικείμενο υπό τον έλεγχο της μηχανής φυσικής. Οι διαφορές από ένα απλό Rigidbody είναι ότι στο 2D, τα αντικείμενα μπορούν να κινηθούν μόνο στο επίπεδο X,Y και μπορούν να περιστραφούν μόνο σε έναν άξονα κάθετο προς αυτό το επίπεδο. Αυτός ήταν και ο λόγος που επιλέχθηκε στην δικιά μας περίπτωση.



Εικόνα 18. Rigidbody 2D

4.3.3. Κίνηση Εχθρού

Στην ενότητα αυτή, αυτό που χρειάστηκε ήταν να δημιουργηθούν μερικές συναρτήσεις που θα υποστηρίξουν τις απαραίτητες λειτουργίες για να αποκτήσουν οι εχθροί τις δυνατότητες κίνησης και ακολούθησης του χαρακτήρα μέσα στις σκηνές. Παρακάτω ο κώδικας.

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

public class Enemy_Run : StateMachineBehaviour {
    public float speed = 2.5f;
    public float attackRange = 3f;
    Transform player;
    Rigidbody2D rb;
    Enemy enemy;

    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {
        player = GameObject.FindGameObjectWithTag("Player").transform;
        rb = animator.GetComponent<Rigidbody2D>();
        enemy = animator.GetComponent<Enemy>();
    }

    // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {
        enemy.LookAtPlayer();
        Vector2 target = new Vector2(player.position.x, rb.position.y);
        Vector2 newPos = Vector2.MoveTowards(rb.position, target, speed * Time.fixedDeltaTime);
        rb.MovePosition(newPos);

        if (Vector2.Distance(player.position, rb.position) <= attackRange) {
            animator.SetTrigger("Attack");
        }
    }

    // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {
        animator.ResetTrigger("Attack");
    }
}
```

Source Code 4. Script “Enemy Run”

Όπως φαίνεται στον κώδικα, αρχικοποιούνται οι τιμές των μεταβλητών που θα χρησιμοποιηθούν στη συνέχεια. Στην συνέχεια, όταν καλείται η συνάρτηση “OnStateEnter()”, αποθηκεύεται η θέση του χαρακτήρα στην μεταβλητή “player” καθώς και τα Components Rigidbody2D και Enemy στις μεταβλητές “rb” και “enemy” αντίστοιχα.

Πηγαίνοντας στην συνάρτηση “OnStateUpdate()”, μέσω του Script “Enemy” καλείται η συνάρτηση “LookAtPlayer()” η οποία φαίνεται στην εικόνα “Source Code 5”. Η χρήση αυτής γίνεται με σκοπό οι εχθροί να κοιτούν πάντα προς την θέση που βρίσκεται ο χαρακτήρας. Αυτό επιτυγχάνεται εντοπίζοντας την θέση του χαρακτήρα μέσα στην σκηνή και χρησιμοποιώντας την δομή ελέγχου που ακολουθεί, δίνονται οι επιθυμητές τιμές έτσι ώστε ο εχθρός να κάνει rotate προς την σωστή κατεύθυνση. Έπειτα, ενημερώνεται η θέση του χαρακτήρα και οι εχθροί κινούνται προς αυτήν την κατεύθυνση. Ακόμα, λόγω της δομής ελέγχου που βρίσκεται στην συνάρτηση, αν η απόσταση του εχθρού από τον χαρακτήρα είναι μικρότερη ή ίση από την τιμή που έχει αποθηκευτεί στην μεταβλητή “attackRange”, ενεργοποιείται το Animation της επίθεσης.

Τέλος, στην συνάρτηση “OnStateExit()”, το Animation της επίθεσης γίνεται Reset ώστε να γίνει επαναφορά του εχθρού στην κατάσταση αδράνειας.

```
37      public void LookAtPlayer() {
38          Vector3 flipped = transform.localScale;
39          flipped.z *= -1f;
40
41          if (transform.position.x > player.position.x && isFlipped) {
42              transform.localScale = flipped;
43              transform.Rotate(0f, 180f, 0f);
44              isFlipped = false;
45          } else if (transform.position.x < player.position.x && !isFlipped) {
46              transform.localScale = flipped;
47              transform.Rotate(0f, 180f, 0f);
48              isFlipped = true;
49          }
50      }
```

Source Code 5. Συνάρτηση “LookAtPlayer()”

4.3.4. Σύστημα Μάχης

Παρακάτω θα περιγράψουμε τον τρόπο που ακολουθήσαμε για να δημιουργήσουμε την λειτουργία επίθεσης των εχθρών. Ακολουθεί ο κώδικας.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnemyWeapon : MonoBehaviour {
6       public int attackDamage = 20;
7       public Vector3 attackOffset;
8       public float attackRange = 1f;
9       public LayerMask attackMask;
10
11      public void Attack() {
12          FindObjectOfType<AudioManager>().Play("Enemyattack");
13          Vector3 pos = transform.position;
14          pos += transform.right * attackOffset.x;
15          pos += transform.up * attackOffset.y;
16
17          Collider2D colInfo = Physics2D.OverlapCircle(pos, attackRange, attackMask);
18          if (colInfo != null) {
19              colInfo.GetComponent<Player>().TakeDamage(attackDamage);
20          }
21      }
22
23      void OnDrawGizmosSelected() {
24          Vector3 pos = transform.position;
25          pos += transform.right * attackOffset.x;
26          pos += transform.up * attackOffset.y;
27
28          Gizmos.DrawWireSphere(pos, attackRange);
29      }
30  }
```

Source Code 6. Script “Enemy Weapon”

Στο συγκεκριμένο Script δηλώνονται οι απαραίτητες τιμές στις μεταβλητές οι οποίες θα χρειαστούν στις παρακάτω συναρτήσεις. Συγκεκριμένα η “OnDrawGizmosSelected()” σχεδιάζει ένα κύκλο γύρω από τον εχθρό ο οποίος αποτελεί το Attack Range. Βάση αυτού, στην συνάρτηση “Attack()”, όταν ο παίχτης βρεθεί στην εμβέλεια της μεταβλητής “attackRange”, γίνεται Damage στον παίχτη εάν ο εχθρός καταφέρει να τον χτυπήσει.

Στο Script που ακολουθεί θα αναλυθούν και κάποια αλλά χαρακτηριστικά των εχθρών. Όπως αυτά της ζωής, του χτυπήματος και του θανάτου.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Enemy : MonoBehaviour {
6       public Transform player;
7       public bool isFlipped = false;
8       public Animator animator;
9       public int maxHealth = 100;
10      int currentHealth;
11      public HealthBar healthBar;
12
13      // Start is called before the first frame update
14      void Start() {
15          currentHealth = maxHealth;
16      }
17
18      // Update is called once per frame
19      public void TakeDamage(int damage) {
20          currentHealth -= damage;
21          animator.SetTrigger("Hurt");
22          healthBar.SetHealth(currentHealth);
23
24          if(currentHealth <= 0) {
25              Die();
26          }
27      }
28
29      void Die() {
30          Debug.Log("Enemy died!");
31          animator.SetBool("IsDead", true);
32          this.enabled = false;
33          GetComponent<BoxCollider2D>().enabled = false;
34          GetComponent<CircleCollider2D>().enabled = false;
35      }
```

Source Code 7. Script “Enemy”

Όσον αφορά την αρχική ζωή των εχθρών, αυτή ορίζεται στην μεταβλητή “maxHealth”. Στην συνέχεια, στην συνάρτηση “Start()” αποθηκεύεται στην μεταβλητή “currentHealth” η ζωή του εχθρού η οποία όμως θα μεταβάλλεται. Έπειτα, όταν καλείται η “TakeDamage()” η ζωή μειώνεται ανάλογα με την τιμή που έχει δοθεί στην μεταβλητή “damage” και μέσω της δομής ελέγχου που βρίσκεται στην συνέχεια, αν η ζωή πάρει τιμή ίση ή μικρότερη από το 0, καλείται η συνάρτηση “Die()” και ο εχθρός πεθαίνει. Οι λειτουργίες αυτής, είναι να ενεργοποιήσει το Animation του θανάτου και να απενεργοποιήσει το αντικείμενο και τις ιδιότητες του εχθρού από την σκηνή.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5
6   public class HealthBar : MonoBehaviour {
7       public Slider slider;
8       public Gradient gradient;
9       public Image fill;
10
11      public void SetMaxHealth(int health) {
12          slider.maxValue = health;
13          slider.value = health;
14          fill.color = gradient.Evaluate(1f);
15      }
16
17      public void SetHealth(int health) {
18          slider.value = health;
19      }
20  }
```

Source Code 8. Script “Health Bar”

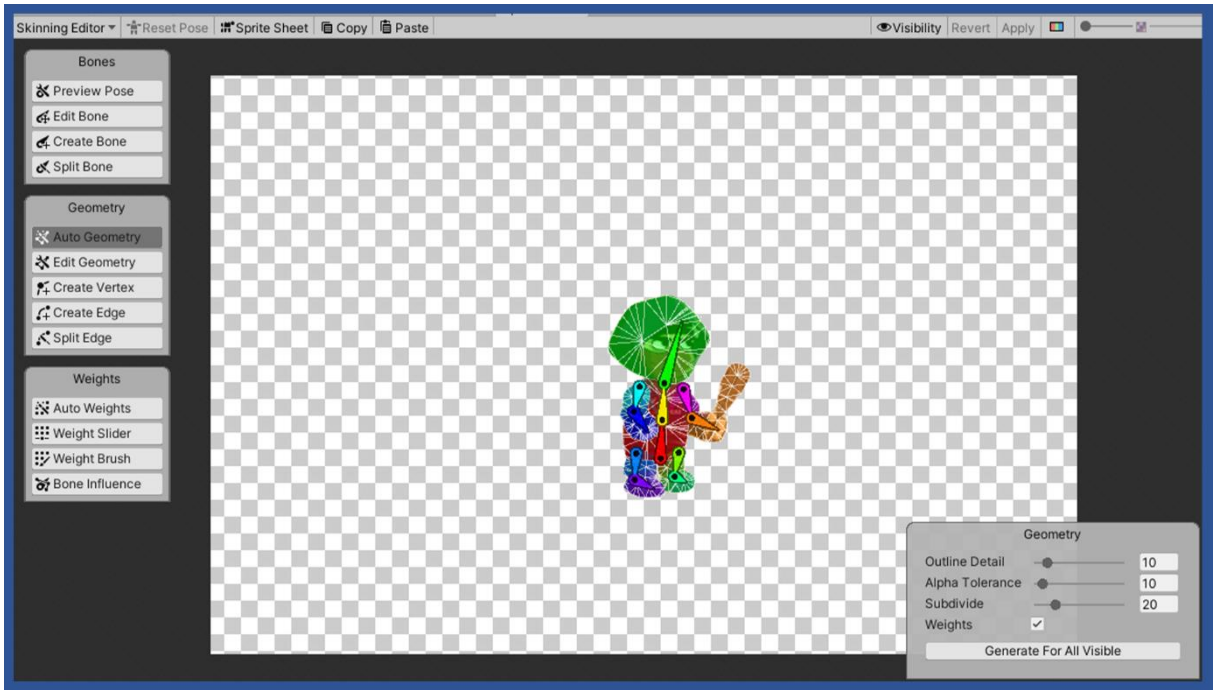
Συνεχίζοντας στο κομμάτι της ζωής των εχθρών, μέσω του Unity δημιουργήσαμε γραφικά μια μπάρα ζωής η οποία μέσω του κώδικα όποτε γίνεται damage σε αυτούς, το κόκκινο χρώμα που επιλέξαμε μειώνεται προς τα αριστερά. Η λειτουργία αυτή χρησιμοποιήθηκε αποκλειστικά για τα δύο bosses του παιχνιδιού και όχι για τους απλούς εχθρούς.

4.4. Κύριος Χαρακτήρας

Περνώντας στην δημιουργία του χαρακτήρα του παιχνιδιού, τα assets, τα animations, καθώς και οι λειτουργίες του, σχεδιάστηκαν εξ ολοκλήρου από εμάς. Παρακάτω θα εξηγηθεί με μεγαλύτερη λεπτομέρεια η διαδικασία που ακολουθήσαμε για τα animations και τις βασικές λειτουργίες που πρέπει να υποστηρίζει ο βασικός χαρακτήρας.

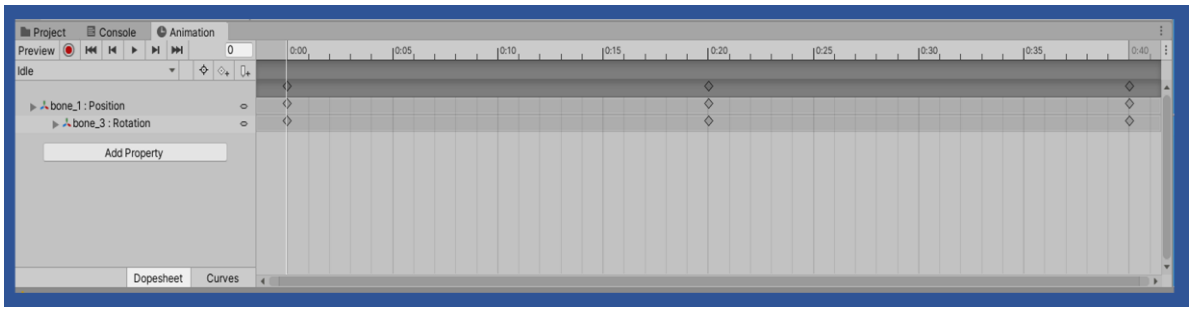
4.4.1. Animations

Αρχικά, χρησιμοποιήθηκαν κάποια πακέτα του Unity που δίνουν τη δυνατότητα στην μηχανή να τα δημιουργεί Animations. Απαραίτητα για να μπορέσει να συμβεί αυτό ήταν τα εξής πακέτα: “2D Animation”, “2D IK”, “2D Sprite”, “2D Tilemap Editor”.



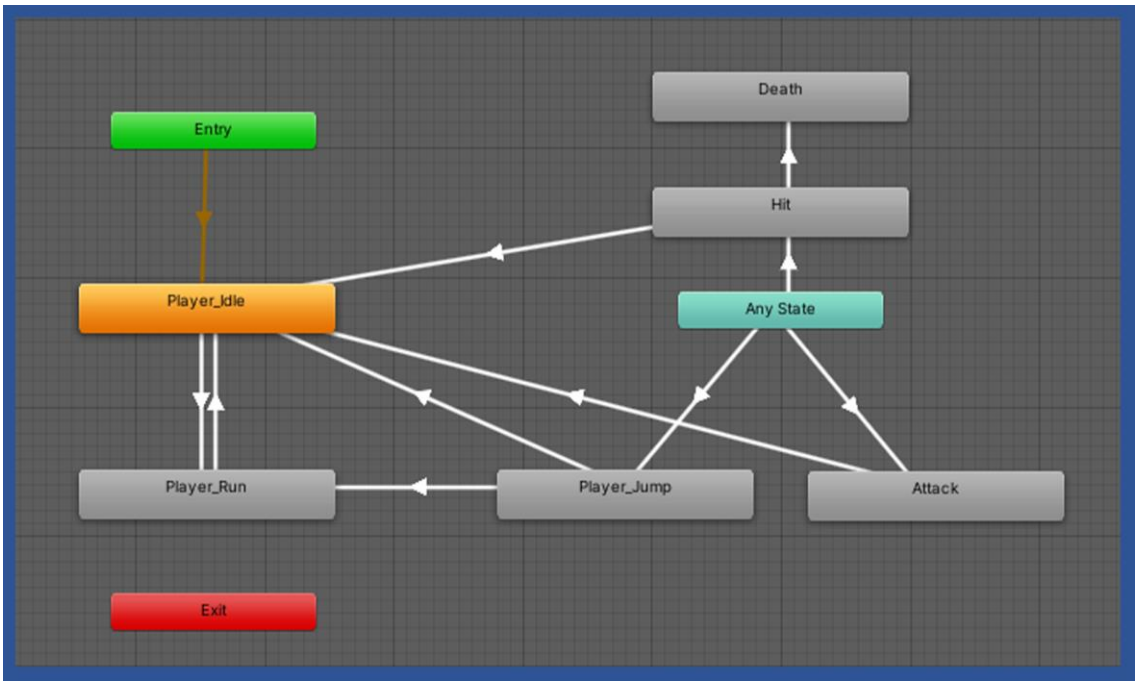
Εικόνα 19. Skinning Editor

Κάτι πολύ σημαντικό που προσφέρεται από τα παραπάνω πακέτα, είναι το εργαλείο “Skinning Editor” το οποίο φαίνεται στην «Εικόνα 19». Αυτό το εργαλείο το χρησιμοποιήσαμε για να δημιουργήσουμε τα Animations του χαρακτήρα. Κάτι πολύ χρήσιμο μέσα από αυτό, είναι η δημιουργία εικονικών Bones τα οποία συνδέουν τα διαφορετικά Layers του χαρακτήρα με σκοπό να υπάρξει ένας εικονικός σκελετός. Τέλος, αφού δόθηκε η δυνατότητα δημιουργίας Animations μετά από αυτό το βήμα, μέσω του εργαλείου που βρίσκεται στην στήλη Animation του Unity, όπως φαίνεται στην «Εικόνα 20», δημιουργήσαμε και τα απαραίτητα Animations.



Εικόνα 20. Στήλη Animation

Με αυτόν τον τρόπο, φτιάξαμε τα κλιπ “Player_Idle” για την κατάσταση αδράνειας, “Player_Run” για το τρέξιμο του χαρακτήρα, “Player_Jump” για το άλμα, “Attack” για την επίθεση, “Hit” για την λήψη χτυπήματος και “Death” για τον θάνατο του χαρακτήρα. Όλα αυτά συσχετίστηκαν με την χρήση του Animator το οποίο αναλύεται στην ενότητα των εχθρών.



Εικόνα 21. Animator Χαρακτήρα

4.4.2. Κίνηση Χαρακτήρα

Σε αυτήν την ενότητα, της λειτουργίας κίνησης του βασικού χαρακτήρα, θα εξετάσουμε τον τρόπο που χρησιμοποιήσαμε για να δώσουμε την δυνατότητα στον χαρακτήρα μας να περιηγείται στις σκηνές. Οι ιδιότητες που χρειάζονται είναι αυτές της κίνησης στον οριζόντιο άξονα των σκηνών αλλά και το άλμα στον κάθετο άξονα, με σκοπό την αποφυγή των εχθρών. Στο πλαίσιο αυτών των δύο λειτουργιών εμπεριέχονται και άλλες ιδιότητες οι οποίες θα παρουσιαστούν στην συνέχεια.

```
using UnityEngine;
using UnityEngine.Events;

public class CharacterController2D : MonoBehaviour {
    [SerializeField] private float m_JumpForce = 400f;
    [Range(0, .3f)] [SerializeField] private float m_MovementSmoothing = .05f;
    [SerializeField] private bool m_AirControl = false;
    [SerializeField] private LayerMask m_WhatIsGround;
    [SerializeField] private Transform m_GroundCheck;
    [SerializeField] private Transform m_CeilingCheck;

    const float k_GroundedRadius = .2f;
    private bool m_Grounded;
    const float k_CeilingRadius = .2f;
    private Rigidbody2D m_Rigidbody2D;
    private bool m_FacingRight = true;
    private Vector3 m_Velocity = Vector3.zero;

    [Header("Events")]
    [Space]

    public UnityEvent OnLandEvent;

    [System.Serializable]
    public class BoolEvent : UnityEvent<bool> { }

    private void Awake() {
        m_Rigidbody2D = GetComponent<Rigidbody2D>();

        if (OnLandEvent == null)
            OnLandEvent = new UnityEvent();
    }

    private void FixedUpdate() {
        bool wasGrounded = m_Grounded;
        m_Grounded = false;
        Collider2D[] colliders = Physics2D.OverlapCircleAll(m_GroundCheck.position, k_GroundedRadius, m_WhatIsGround);
        for (int i = 0; i < colliders.Length; i++) {
            if (colliders[i].gameObject != gameObject) {
                m_Grounded = true;
                if (!wasGrounded)
                    OnLandEvent.Invoke();
            }
        }
    }
```

Source Code 9. Script "Character Controller"

Οι προαναφερθείσες λειτουργίες υλοποιήθηκαν μέσω του Script "CharacterController2D". Όπως φαίνεται στο "Source Code 9" αρχικοποιούνται οι βασικές μεταβλητές. Κάποιες που αξίζει να αναφερθούν είναι η μεταβλητή "m_JumpForce" από την οποία ορίζεται το ύψος του άλματος του χαρακτήρα αλλά και η "m_MovementSmoothing" μέσω της οποίας ομαλοποιείται η κίνησή του. Στην συνέχεια, οι συναρτήσεις "Awake()" και "FixedUpdate()" συμβάλουν ώστε ο χαρακτήρας μέσω των Colliders που αποτελούν Component του Unity αλλά και της μεταβλητής "OnLandEvent", να παραμένει στα όρια του εδάφους που έχουν οριστεί μέσα στην σκηνή και να μην χάνεται από την οθόνη.

```
47      public void Move(float move, bool jump) {
48          if (m_Grounded || m_AirControl) {
49              Vector3 targetVelocity = new Vector2(move * 10f, m_Rigidbody2D.velocity.y);
50              m_Rigidbody2D.velocity = Vector3.SmoothDamp(m_Rigidbody2D.velocity, targetVelocity, ref m_Velocity, m_MovementSmoothing);
51              if (move > 0 && !m_FacingRight) {
52                  Flip();
53              }
54              else if (move < 0 && m_FacingRight) {
55                  Flip();
56              }
57          }
58          if (m_Grounded && jump) {
59              m_Grounded = false;
60              m_Rigidbody2D.AddForce(new Vector2(0f, m_JumpForce));
61          }
62      }
63
64      private void Flip() {
65          m_FacingRight = !m_FacingRight;
66          Vector3 theScale = transform.localScale;
67          theScale.x *= -1;
68          transform.localScale = theScale;
69      }
70  }
```

Source Code 10. Script "Character Controller"

Το Script "CharacterController2D" συνεχίζεται με την συνάρτηση "Move()" η οποία δίνει την δυνατότητα στον παίκτη να κινείται δεξιά, αριστερά αλλά και να κάνει άλμα. Επίσης, μέσω των δομών ελέγχου που φαίνονται στο "Source Code 10", ο χαρακτήρας αν η τιμή της παραμέτρου "move" είναι μεγαλύτερη του μηδέν και ο παίκτης δεν κοιτάζει προς τα δεξιά, γίνεται στροφή 180 μοιρών. Το ίδιο ισχύει και για την περίπτωση από δεξιά προς τα αριστερά με τον αντίστοιχο τρόπο. Η στροφή του χαρακτήρα γίνεται καλώντας την συνάρτηση "Flip()".

Τέλος, στο Script "PlayerMovement" που φαίνεται στην συνέχεια, αποθηκεύονται οι είσοδοι που δίνει ο χρήστης, οι οποίες χρησιμοποιούνται ως παράμετροι στην συνάρτηση "Move()" του "Character Controller", με αποτέλεσμα ο χαρακτήρας να κινείται στον οριζόντιο άξονα, αλλά και να κάνει άλμα στον κάθετο αντίστοιχα. Ακόμα όταν το πάνω βέλος πατηθεί, ενεργοποιείται το Animation του άλματος, καθώς επίσης και του τρεξίματος μέσω της εναλλαγής ταχύτητας.

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

public class PlayerMovement : MonoBehaviour {
    public CharacterController2D controller;
    public Animator animator;
    public float runSpeed = 40f;
    float horizontalMove = 0f;
    bool jump = false;
    bool isMoving = false;

    // Update is called once per frame
    void Update () {
        horizontalMove = Input.GetAxisRaw("Horizontal") * runSpeed;

        if (Input.GetKey (KeyCode.RightArrow) || Input.GetKey (KeyCode.LeftArrow)) {
            GetComponent<AudioSource>().UnPause();
        } else {
            GetComponent<AudioSource>().Pause();
        }

        if (Input.GetKey (KeyCode.UpArrow)) {
            FindObjectOfType<AudioManager>().Play("PlayerJump");
        }

        animator.SetFloat("Speed", Mathf.Abs(horizontalMove));

        if (Input.GetButtonDown("Jump")) {
            jump = true;
            animator.SetBool("isJumping", true);
        }
    }

    public void OnLanding () {
        animator.SetBool("isJumping", false);
    }

    void FixedUpdate () {
        // Move our character
        controller.Move(horizontalMove * Time.fixedDeltaTime, jump);
        jump = false;
    }
}
```

Source Code 11. Script “Player Movement”

4.4.4. Σύστημα Μάχης

Παρακάτω, θα παρουσιαστεί η μέθοδος που ακολουθήσαμε για να υπάρξει ένα ολοκληρωμένο σύστημα μάχης για το παιχνίδι μας. Αυτό που χρειάστηκε ήταν να δημιουργήσουμε δύο βασικά Scripts τα οποία θα εξηγηθούν στην συνέχεια.

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

public class PlayerCombat : MonoBehaviour {
    public Animator animator;
    public Transform attackPoint;
    public float attackRange = 0.5f;
    public LayerMask enemyLayers;
    public int attackDamage = 40;
    public float attackRate = 2f;
    float nextAttackTime = 0f;

    // Update is called once per frame
    void Update() {
        if(Time.time >= nextAttackTime) {
            if (Input.GetKeyDown(KeyCode.Space)) {
                Attack();
                FindObjectOfType<AudioManager>().Play("Playermelee");
                nextAttackTime = Time.time + 1f / attackRate;
            }
        }
    }

    void Attack() {
        // Play an attack animation
        animator.SetTrigger("Attack");
        // Detect enemies in range of aatack
        Collider2D[] hitEnemies = Physics2D.OverlapCircleAll(attackPoint.position, attackRange, enemyLayers);
        // Damage them
        foreach(Collider2D enemy in hitEnemies) {
            enemy.GetComponent<Enemy>().TakeDamage(attackDamage);
        }
    }

    void OnDrawGizmosSelected() {
        if (attackPoint == null)
            return;
        Gizmos.DrawWireSphere(attackPoint.position, attackRange);
    }
}
```

Source Code 12. Script “Player Combat”

Ένα από αυτά τα δύο Scripts, είναι αυτό του “PlayerCombat”. Όπως φαίνεται στο “Source Code 12”, γίνονται οι απαραίτητες αρχικοποιήσεις μεταβλητών, ενώ στην συνέχεια στην συνάρτηση “Update()” η οποία καλείται μια φορά σε κάθε ένα Frame, μέσω των δομών ελέγχου όταν το πλήκτρο διαστήματος πατηθεί, καλείται η συνάρτηση “Attack()”, εφαρμόζοντας και μια καθυστέρηση μεταξύ των επιθέσεων μέσω του υπολογισμού που αποθηκεύεται στην μεταβλητή “nextAttackTime”. Στην συνάρτηση “Attack()” ενεργοποιείται το Animation της επίθεσης, καθώς επίσης εντοπίζονται και οι εχθροί που βρίσκονται στην επιθυμητή εμβέλεια. Η δομή επανάληψης που ακολουθεί εφαρμόζει Damage στους εχθρούς. Τέλος, η συνάρτηση “OnDrawGizmosSelected()” σχεδιάζει έναν εικονικό κύκλο περιμετρικά του όπλου του χαρακτήρα, ο οποίος αποτελεί την εμβέλεια που έχει το όπλο για να χτυπήσει κάποιον εχθρό.

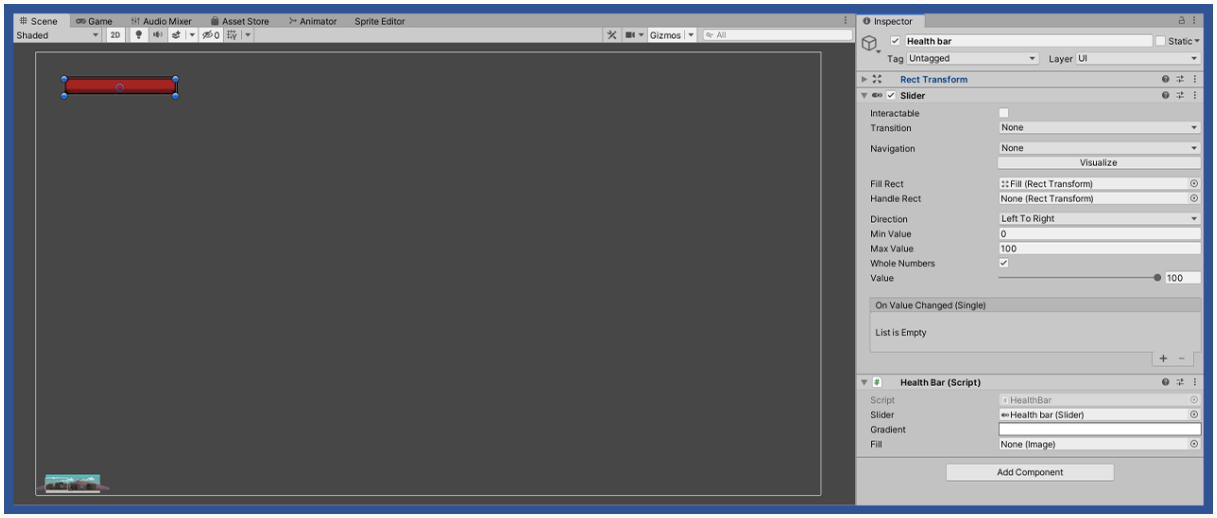
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class Player : MonoBehaviour {
    public int health = 40;
    public int maxHealth = 100;
    public int currentHealth;
    public Animator animator;
    public HealthBar healthBar;

    [SerializeField]
    private GameObject GameOverUI;

    // Start is called before the first frame update
    void Start() {
        currentHealth = maxHealth;
    }

    public void EndGame() {
        Debug.Log("GAME OVER!");
        GameOverUI.SetActive(true);
        Time.timeScale = 0;
    }

    public void TakeDamage(int damage) {
        currentHealth -= damage;
        animator.SetTrigger("Hurt");
        healthBar.SetHealth(currentHealth);

        if(currentHealth <= 0) {
            Die();
        }
    }

    void Die() {
        Debug.Log("Character died!");
        animator.SetBool("IsDead", true);
        this.enabled = false;
        GetComponent<BoxCollider2D>().enabled = false;
        GetComponent<CircleCollider2D>().enabled = false;
        EndGame();
    }
}
```

Source Code 13. Script "Player"

Το δεύτερο βασικό Script του συστήματος μάχης, είναι αυτό του "Player". Σε αυτό, ορίζονται οι απαραίτητες μεταβλητές καθώς και οι τιμές τους. Συνεχίζοντας, στην συνάρτηση "Start()" αρχικοποιείται η τιμή της ζωής του χαρακτήρα πριν από το πρώτο Frame της σκηνής. Πηγαίνοντας στην συνάρτηση "EndGame()", δίνεται η δυνατότητα κατά τον θάνατο του χαρακτήρα να εμφανιστεί το Interface του Game Over. Μια ακόμα σημαντική λειτουργία είναι αυτή της "TakeDamage()", μέσα από την οποία όταν αυτή κληθεί μειώνεται η ζωή του παίκτη. Ο τρόπος που καλείται είναι μέσω της συνάρτησης "Attack()" του Script "EnemyWeapon" το

οποίο έχει αναφερθεί παραπάνω. Όταν η τιμή της ζωής του παίκτη γίνει μικρότερη ή ίση με το μηδέν, καλείται η συνάρτηση “Die()” από την οποία το Animation θανάτου ενεργοποιείται, τα Components του παίκτη απενεργοποιούνται και τέλος το Interface του Game Over εμφανίζεται από την “EndGame()” που εξηγήθηκε παραπάνω.

Κάτι ακόμα όσον αφορά το σύστημα μάχης, είναι ότι ο χαρακτήρας διαθέτει και αυτός όπως και τα bosses του παιχνιδιού μια μπάρα ζωής. Ο κώδικας που χρειάστηκε είναι ο ίδιος με αυτόν που χρειάστηκε για τους εχθρούς. Αυτός φαίνεται στο “Source Code 8”.



Εικόνα 22. Health Bar Χαρακτήρα

4.5. Άλλες Σημαντικές Λειτουργίες

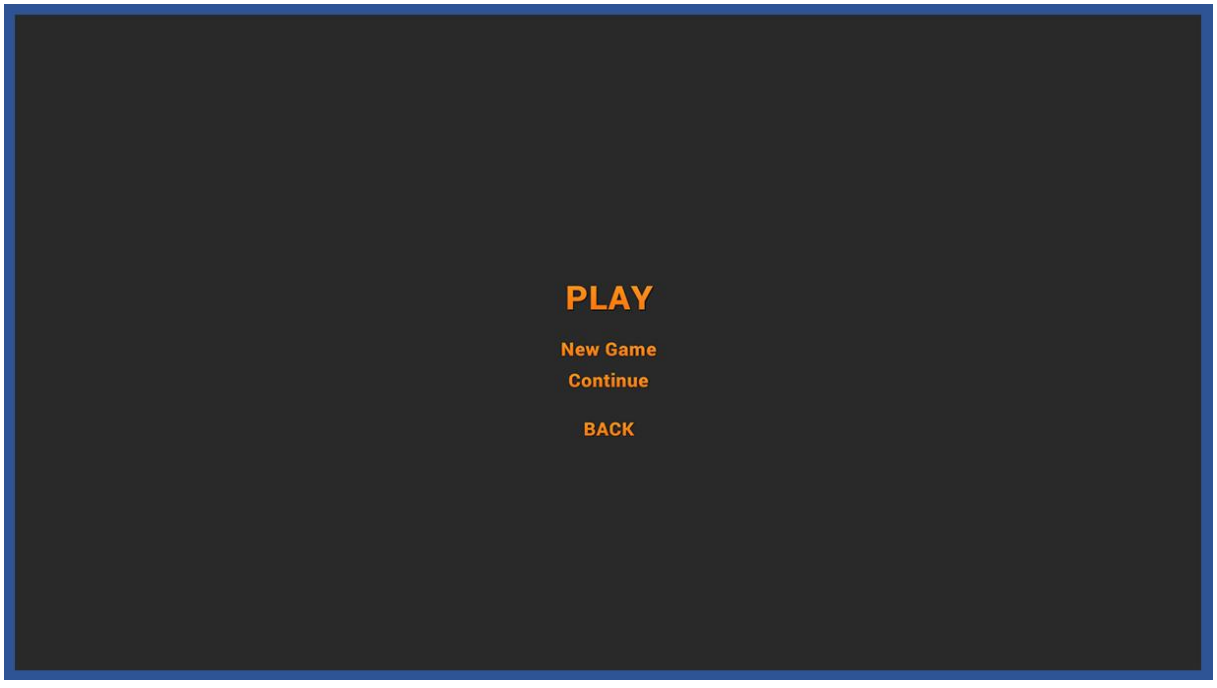
Περνώντας στην τελευταία ενότητα της υλοποίησης του παιχνιδιού, θα αναλύσουμε μερικές ακόμα σημαντικές λειτουργίες οι οποίες ήταν απαραίτητες για την δημιουργία ενός λειτουργικού παιχνιδιού.

4.5.1. Δημιουργία Menu Παιχνιδιού

Μια από αυτές τις λειτουργίες είναι το βασικό μενού, μέσα από το οποίο δίνεται η δυνατότητα στον χρήστη να ξεκινήσει το παιχνίδι, να επιστρέψει στην σκηνή που βρισκόταν μετά από έξοδο του παιχνιδιού, να παραμετροποιήσει τις ρυθμίσεις καθώς επίσης να κάνει και έξοδο από το παιχνίδι.

Εικόνα 23. Main Menu

Στο κύριο μενού, όπως φαίνεται στην «Εικόνα 23», όταν το “Play” πατηθεί εμφανίζεται μια δεύτερη οθόνη στην οποία του δίνεται η δυνατότητα ξεκινήσει το παιχνίδι από την αρχή ή να συνεχίσει από εκεί που σταμάτησε στην περίπτωση που έχει ξαναπαίξει. Αυτή η ιδιότητα φαίνεται παρακάτω.



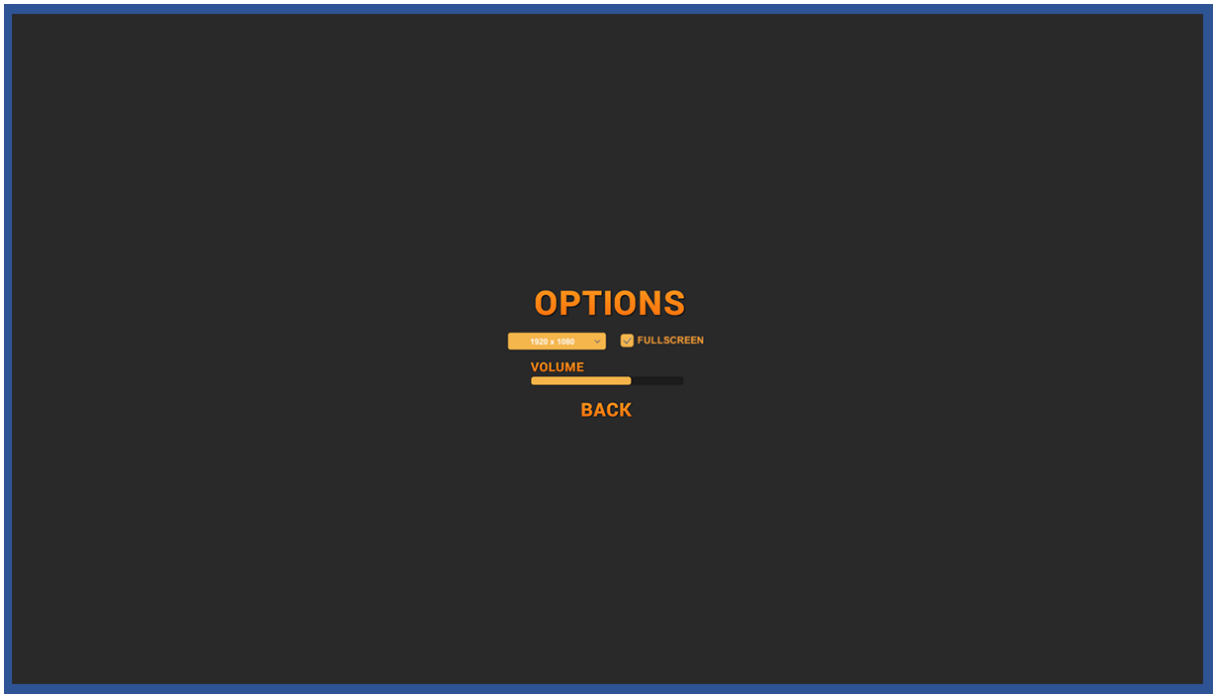
Εικόνα 24. Play Menu

Πιο τεχνικά, όταν πατηθεί το κουμπί “New Game” ή το “Continue”, καλείται το Script “LevelLoader” το οποίο έχει εξηγηθεί παραπάνω και ο κώδικας φαίνεται στο “Source Code 1”. Η λειτουργία εξόδου από το παιχνίδι πραγματοποιείται από τον παρακάτω κώδικα μέσω της συνάρτησης “QuitGame()”.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6   public class MainMenu : MonoBehaviour {
7       public void QuitGame() {
8           Debug.Log ("QUIT!");
9           Application.Quit();
10      }
11  }
```

Source Code 14. Script “Main Menu”

Όσον αφορά τις ρυθμίσεις, προσφέρονται μερικές σημαντικές παραμετροποιήσεις. Δίνεται η δυνατότητα αλλαγής της ανάλυσης, εκτέλεση του παιχνιδιού σε Fullscreen ή Windowed Mode, αλλά και αυξομείωση της έντασης του ήχου.



Εικόνα 24. Options Menu

Για να επιτευχθούν οι λειτουργίες που αναφέρθηκαν παραπάνω χρειάστηκε να δημιουργήσουμε το Script “SettingsMenu”. Σε αυτό η συνάρτηση “Start()” εφαρμόζει την ανάλυση του παιχνιδιού ενώ παρακάτω έχουν δημιουργηθεί τρείς ακόμα συναρτήσεις. Η “SetResolution()” που καθορίζει την τιμή της ανάλυσης. Η “SetVolume()” που ορίζει την τιμή της έντασης και η “SetFullscreen()” που τρέχει το παιχνίδι σε πλήρη οθόνη. Οι συναρτήσεις αυτές καλούνται μέσω Event που ενεργοποιείται στα αντίστοιχα Components του Unity, όταν τα γραφικά αντικείμενα που δημιουργήθηκαν αλλάζουν μορφή.

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

public class SettingsMenu : MonoBehaviour {
    public AudioMixer audioMixer;
    public Dropdown resolutionDropdown;
    Resolution[] resolutions;

    void Start() {
        resolutions = Screen.resolutions;
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        int currentResolutionIndex = 0;

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

            if (resolutions[i].width == Screen.currentResolution.width && resolutions[i].height == Screen.currentResolution.height) {
                currentResolutionIndex = i;
            }
        }
        resolutionDropdown.AddOptions(options);
        resolutionDropdown.value = currentResolutionIndex;
        resolutionDropdown.RefreshShownValue();
    }

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

        audioMixer.SetFloat("volume", volume);
    }

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

Source Code 15. Script “Settings Menu”

4.5.2. Δημιουργία Pause Menu

Ένα ακόμα σημαντικό interface που έπρεπε να φτιάξουμε, είναι αυτό της παύσης του παιχνιδιού. Μέσα από αυτό, ο παίκτης μπορεί όπως είναι προφανές να σταματήσει το παιχνίδι όποια στιγμή θέλει, να κάνει επαναφορά σε αυτό όταν το θελήσει, να επιστρέψει στο αρχικό μενού αλλά και να κάνει έξοδο από αυτό.

Εικόνα 25. Pause Menu

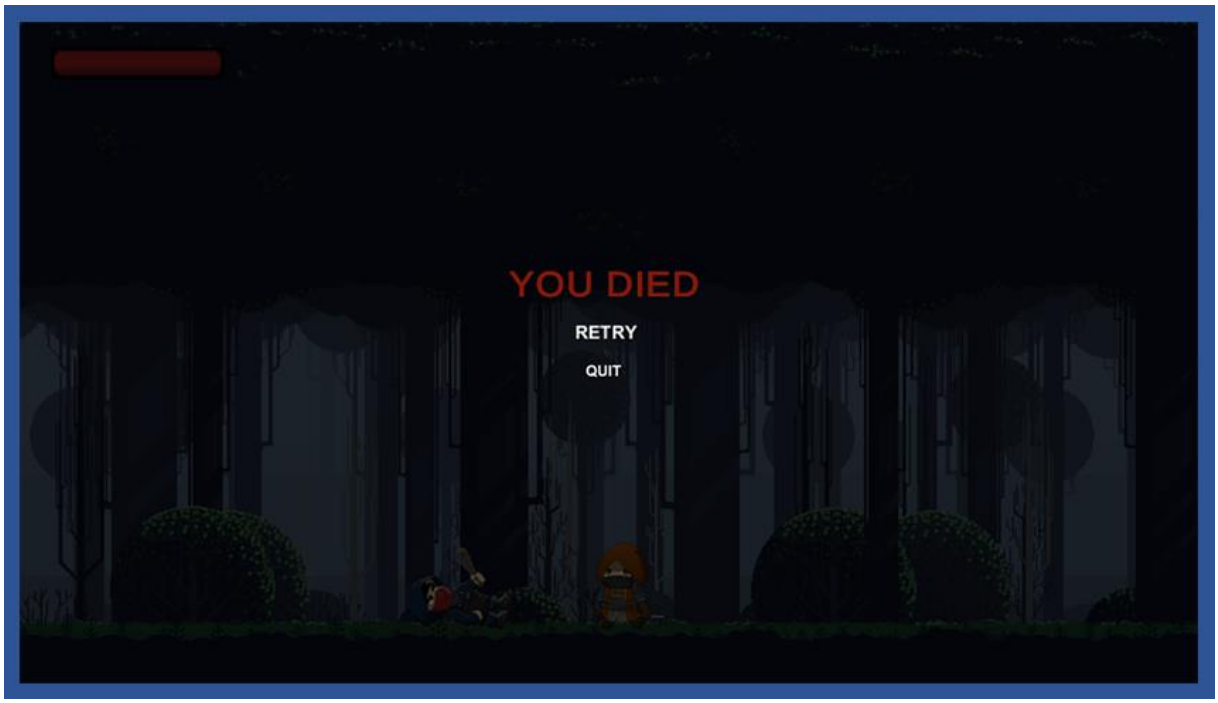
Ο κώδικας που χρειάστηκε για αυτές τις λειτουργίες είναι αυτός του Script "PauseMenu". Σε αυτόν υπάρχουν τέσσερις συναρτήσεις που υποστηρίζουν τις λειτουργίες που αναφέρθηκαν. Η "Resume()" από την οποία απενεργοποιείται το Interface της παύσης και γίνεται ψευδές το Boolean "GameIsPaused". Η "Pause()" που έχει την ίδια λειτουργία, κάνοντας όμως τις παραπάνω τιμές αληθείς. Η "LoadMenu()" που μέσω του "SceneManager" εμφανίζει το αρχικό μενού όταν το κουμπί "MENU" πατηθεί και η "QuitGame()" που πραγματοποιεί την έξοδο του παιχνιδιού. Οι συναρτήσεις έχουν ρυθμιστεί να καλούνται μέσα από τα Components του Unity όταν γίνεται κλικ στο παραπάνω Interface της παύσης. Τέλος η συνάρτηση "Update()", χρησιμοποιώντας το Boolean "GameIsPaused" σε μια δομή ελέγχου, όταν το πλήκτρο escape πατηθεί, αν ο χρήστης παίζει το παιχνίδι εμφανίζεται το μενού παύσης, ενώ αν βρίσκεται ήδη εκεί γίνεται επαναφορά στο παιχνίδι.

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

public class PauseMenu : MonoBehaviour {
    public static bool GameIsPaused = false;
    public GameObject pauseMenuUI;

    // Update is called once per frame
    void Update() {
        if (Input.GetKeyDown(KeyCode.Escape)) {
            if (GameIsPaused) {
                Resume();
            } else {
                Pause();
            }
        }
    }

    public void Resume () {
        Cursor.visible = false;
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    void Pause () {
        Cursor.visible = true;
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }

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

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

Source Code 16. Script “Pause Menu”

4.5.3. Δημιουργία Game Over

Μια ακόμα ιδιότητα που χρειάστηκε ήταν αυτή ενός UI για την περίπτωση που ο παίκτης πεθάνει, εννοώντας την οθόνη για το Game Over. Το συγκεκριμένο Interface ενεργοποιείται από την συνάρτηση “EndGame()” του Script “Player” το οποίο φαίνεται παραπάνω στο “Source Code 13”.



Εικόνα 26. Game Over

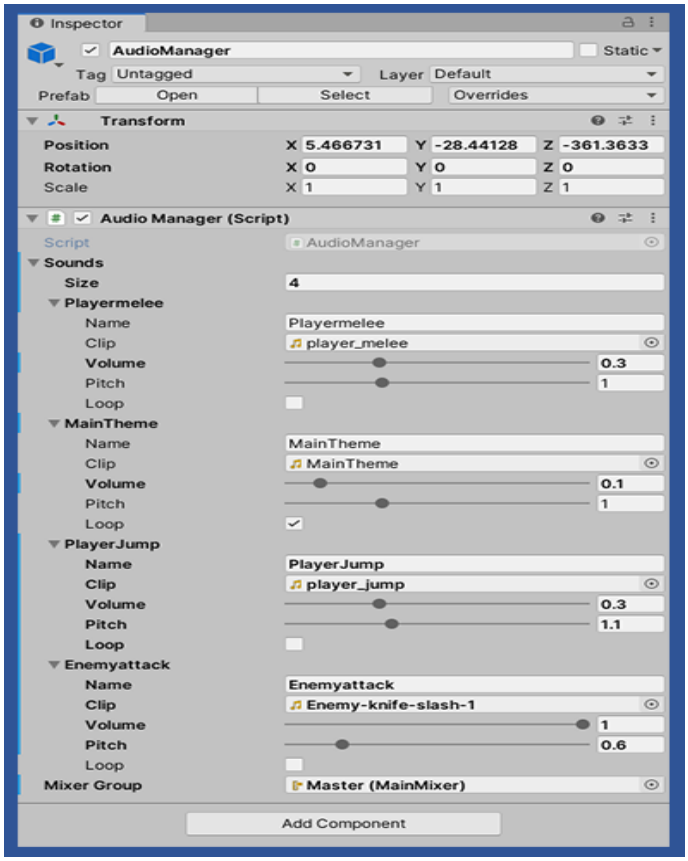
Για να αποκτήσουν κάποιες ιδιότητες τα περιεχόμενα της συγκεκριμένης διεπαφής, έπρεπε να δημιουργήσουμε ακόμα ένα Script. Σε αυτό υπάρχουν οι δύο συναρτήσεις που χρειάστηκαν για να δοθεί η δυνατότητα στον χρήστη να προσπαθήσει ξανά, αλλά και να κάνει έξοδο από το παιχνίδι. Όσον αφορά την πρώτη λειτουργία, αυτή υποστηρίζεται από την συνάρτηση "Retry()" η οποία μέσω του "Scene Manager" εντοπίζει την παρούσα σκηνή και την εκτελεί ξανά. Για την δεύτερη λειτουργία δημιουργήθηκε η συνάρτηση "Quit()" από την οποία γίνεται η έξοδος. Το Script φαίνεται στην συνέχεια.

```
1   using UnityEngine;
2   using UnityEngine.SceneManagement;
3
4   public class GameOver : MonoBehaviour {
5       public void Start(){
6           Cursor.visible = true;
7       }
8
9       // Start is called before the first frame update
10      public void Quit() {
11          Debug.Log("Application quit!");
12          Application.Quit();
13      }
14
15      // Update is called once per frame
16      public void Retry() {
17          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
18          Time.timeScale = 1;
19      }
20  }
```

Source Code 17. Script "Game Over"

4.5.4. Ενσωμάτωση Sound Effects

Σε αυτή την ενότητα θα αναλύσουμε τον τρόπο που ενσωματώσαμε τα εφέ ήχων, τα οποία προστέθηκαν για να κάνουν περισσότερο διαδραστικό το παιχνίδι μας. Αυτό έγινε με την βοήθεια του Unity το οποίο προσφέρει ένα χρήσιμο εργαλείο που ονομάζεται Audio Manager.



Εικόνα 27. Audio Manager

Για να λειτουργήσει το συγκεκριμένο εργαλείο είναι απαραίτητη η δημιουργία κώδικα που θα προσθέσει τα απαιτούμενα πεδία στο Unity. Στην συνάρτηση “Awake()”, αυτό που αξίζει να αναφερθεί είναι η δομή επανάληψης “Foreach” μέσα από την οποία γίνεται αυτό που αναφέρθηκε παραπάνω, η δημιουργία των πεδίων στο Unity. Από αυτά γίνεται με εύκολο τρόπο να ρυθμιστεί το κάθε εφέ ήχου ξεχωριστά. Επίσης, υπάρχει η συνάρτηση “Play()” η οποία δέχεται ως παράμετρο το όνομα του αρχείου ήχου. Αν το όνομα αρχείου βρεθεί, εκτελείται το αρχείο. Στην περίπτωση που αυτό δεν βρεθεί, παρουσιάζεται ένα μήνυμα που αναφέρει ότι το αρχείο δεν βρέθηκε. Η “Play()” στο συγκεκριμένο Script καλείται στην συνάρτηση “Start()” με σκοπό να παίζει το βασικό θέμα ήχου στο παρασκήνιο.

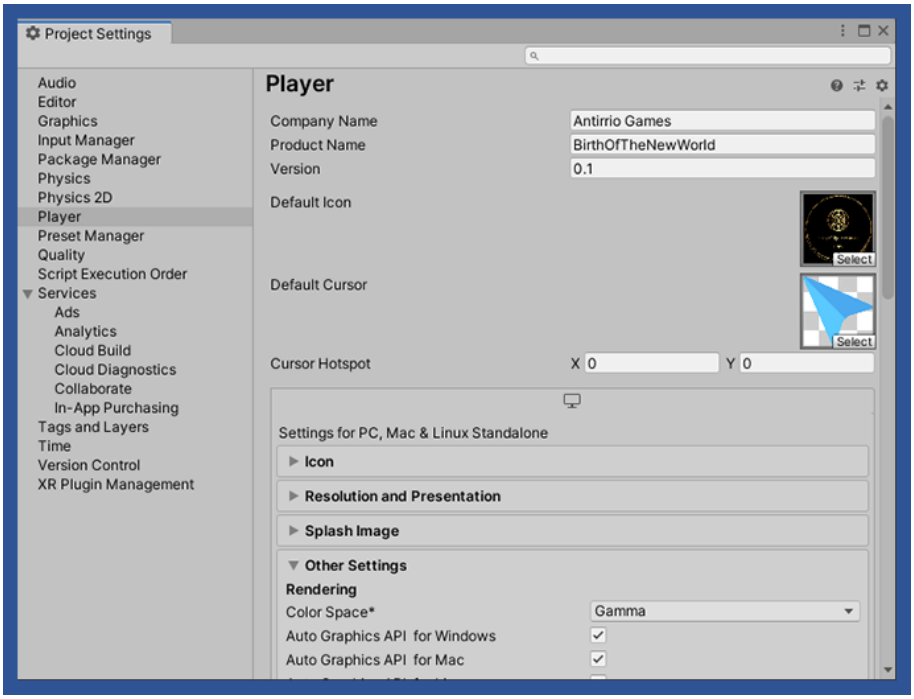
```
using UnityEngine.Audio;
using System;
using UnityEngine;

public class AudioManager : MonoBehaviour {
    public Sound[] sounds;
    public static AudioManager instance;
    public AudioMixerGroup MixerGroup;

    void Awake() {
        if (instance == null)
            instance = this;
        else {
            Destroy(gameObject);
            return;
        }

        DontDestroyOnLoad(gameObject);

        foreach (Sound s in sounds) {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;
            s.source.outputAudioMixerGroup = MixerGroup;

            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
            s.source.loop = s.loop;
        }
    }

    void Start() {
        Play("MainTheme");
    }

    // Update is called once per frame
    public void Play (string name) {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        if(s == null) {
            Debug.LogWarning("Sound: " + name + " not found!");
            return;
        }
        s.source.Play();
    }
}
```

Source Code 18. Script "Audio Manager"

Κάτι ακόμα που αξίζει να αναφερθεί στο κομμάτι των εφέ ήχων, είναι ο τρόπος που ενεργοποιούνται τα εφέ επίθεσης του χαρακτήρα και των εχθρών. Αυτό συμβαίνει από τις συναρτήσεις που εκτελούν την επίθεσή τους από το "Source Code 6" για τους εχθρούς και το "Source Code 12" για τον χαρακτήρα, μέσω της εντολής "FindObjectOfType" η οποία χρησιμοποιεί το "Audio Manager" για να γίνει αναπαραγωγή του εκάστοτε ήχου. Με τον ίδιο τρόπο εκτελούνται και τα υπόλοιπα εφέ ήχου από τις συναρτήσεις για την κίνηση και το άλμα.

4.5.5. Λειτουργίες Κέρσορα

Σε αυτή την ενότητα θα αναφέρουμε μια ακόμα λεπτομέρεια που προσθέσαμε. Αυτή αφορά τον κέρσορα κατά την διάρκεια εκτέλεσης του παιχνιδιού. Οι ενέργειες που κάναμε ήταν να αλλάξουμε τον τρόπο που θα εμφανίζεται από τις ρυθμίσεις του Unity, αλλά και η εφαρμογή κώδικα για την απόκρυψη και την εμφάνισή του στα σημεία που αυτό κρίνεται απαραίτητο.



Εικόνα 28. Ρύθμιση Αλλαγής Κέρσορα

Για το κομμάτι της εμφάνισης και απόκρυψης του κέρσορα, για αρχή χρησιμοποιήσαμε το Script “CursorHide” από το οποίο κατά την εκκίνηση του παιχνιδιού γίνεται μη ορατός κάνοντας ψευδές το Boolean “Cursor.visible”. Το συγκεκριμένο Boolean εφαρμόζεται στο “Source Code 16” για την παύση και την επαναφορά του παιχνιδιού αλλά και στο “Source Code 17” για το Game Over, κάνοντάς το αληθές ή ψευδές ανάλογα την περίσταση.

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

public class CursorHide : MonoBehaviour {
    // Start is called before the first frame update
    void Start() {
        Cursor.visible = false;
    }
}
```

Source Code 19. Script "Cursor Hide"

4.5.6. Δημιουργία Credits

Φτάνοντας στο τέλος του κεφαλαίου της υλοποίησης, παρακάτω θα παρουσιαστεί η δημιουργία του Interface για τα Credits του παιχνιδιού καθώς και μια απλή λειτουργία μέσω ενός Script κώδικα.

Εικόνα 29. Credits

Η λειτουργία αυτή, έχει να κάνει με τον τερματισμό εκτέλεσης του παιχνιδιού πατώντας το κουμπί “Quit”. Αυτό συμβαίνει από την συνάρτηση “Quit()” που φαίνεται στην συνέχεια.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Credits : MonoBehaviour {
6       // Start is called before the first frame update
7       public void Quit() {
8           Debug.Log("QUIT");
9           Application.Quit();
10      }
11  }
```

Source Code 20. Script “Credits”

Κεφάλαιο 5ο: Συμπεράσματα

Στο πλαίσιο της παρούσας διπλωματικής εργασίας, κατά την διάρκεια του σχεδιασμού και της υλοποίησης της πρώτης μας προσπάθειας να δημιουργήσουμε ένα παιχνίδι δύο διαστάσεων χρησιμοποιώντας την μηχανή ανάπτυξης Unity, μέσα από την εμπειρία που αποκτήσαμε πάνω σε αυτή περνώντας από τα διαφορετικά στάδια του σχεδιασμού και της υλοποίησης, διαπιστώσαμε πέρα από την θεωρητική ανάλυση και στην πράξη, τα χαρακτηριστικά και τις ευκολίες που προσφέρει για κάποιον που βρίσκεται σε ένα πρώιμο στάδιο πάνω στον τομέα της ανάπτυξης βιντεοπαιχνιδιών. Οι ιδιότητες αυτές που μας διευκόλυναν ήταν το μεγάλο πλήθος παραδειγμάτων και tutorials, η ευχρηστία του User Interface και η ύπαρξη γλώσσας αντικειμενοστραφούς προγραμματισμού στον τομέα της ανάπτυξης.

Κάποιοι προβληματισμοί που μας απασχόλησαν κατά την διάρκεια της ανάπτυξης, ήταν ο μεγάλος όγκος γνώσεων σε πολλούς διαφορετικούς τομείς. Αυτούς αποτέλεσαν ο σχεδιασμός σεναρίου, ο σχεδιασμός των κατάλληλων assets και animations, τα ηχητικά εφέ και η δημιουργία κώδικα. Σε ένα project όπως αυτό αλλά και σε κάτι ακόμα πιο μεγάλο, όπως γίνεται σε μεγάλα Studios ανάπτυξης παιχνιδιών, υπάρχουν συγκεκριμένες ομάδες ώστε να μοιράζεται ο όγκος δουλειάς και να γίνονται οι κατάλληλες εργασίες από τους μηχανικούς που κατέχουν γνώσεις πάνω στον συγκεκριμένο κλάδο. Παρότι σαν εμπειρία ήταν πολύ ενδιαφέρουσα και μας βοήθησε να δούμε και να εφαρμόσουμε όλα αυτά και στην πράξη, καταλήξαμε στο ότι ο κάθε επιμέρους τομέας στην διαδικασία δημιουργίας ενός παιχνιδιού απαιτεί και μεγάλη ενασχόληση για την απόκτηση μιας ικανοποιητικής εμπειρίας.

Μελλοντικά, θα θέλαμε να δουλέψουμε πάνω σε ένα πιο μεγάλο Project, τόσο προσωπικά αλλά και ως μέρος μιας μεγάλης ομάδας σε ένα συγκεκριμένο τομέα, αυτόν του προγραμματισμού, καθώς εκεί κατέχουμε και την μεγαλύτερη εμπειρία.

Βιβλιογραφία

[1] “Unreal Engine,” https://www.unrealengine.com/

[2] “Unity3d,” https://unity3d.com/

[3] “GameMaker,” https://www.yoyogames.com/

[4] “JMonkey,” https://jmonkeyengine.org/

[5] “OGRE 3D,” http://www.ogre3d.org/

[6] “SIO2,” http://www.sio2interactive.com/index.php

[7] Petridis, P., Dunwell, I., Panzoli, D., Arnab, S., Protopsaltis, A., Hendrix, M. and Freitas, S., Game Engines Selection Framework for High-Fidelity Serious Applications, International Journal of Interactive Worlds, Vol 2012 (2012), 19 pages.

[8] Alexander, A.L., Brunyé, T., Sidman, J. and Weil, S.A., “From Gaming to Training: A Review of Studies on Fidelity, Immersion, Presence, and Buy-in and Their Effects on Transfer in PC-Based Simulations and Games”, Aptima, Inc., Woburn, MA, DARWARS Training Impact Group, 2012.

[9] Pattrasitidecha, A., “Comparison and evaluation of 3D mobile game engines”, Master of Science Thesis in the Program International Design, Department of Computer Science and Engineering, University of Gothenburg, 2014.

[10] Rocha, R. V., Rocha, R. V. and Araújo, R., “Selecting the best open source 3D games engines”, In Proceedings of the brazilian Symposium on Games and Digital Entertainment, Florianópolis, Santa Catarina, Brazil, 2010.

[11] Harris, A. C., “Design and implementation of an autonomous robotics simulator”, Doctoral dissertation, The University of North Carolina at Charlotte, 2011.

[12] “Survival Shooter Tutorial- Unity 3D,” https://unity3d.com/learn/tutorials/projects/survival-shooter-tutorial

[13] “Blueprint Twin Stick Shooter- Unreal Engine 4,” https://docs.unrealengine.com/latest/INT/Videos/PLZlv_N0_O1gb5sdygbSiEU7hb0eomNLdq/