`

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών
«ΠΜΣ ΠΛΗΡΟΦΟΡΙΚΗ»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Μαθηματικός Λαβύρινθος** **Math Maze**
Ονοματεπώνυμο Φοιτητή	**Σκολαρίκος Στέργιος**
Πατρώνυμο	**Μαρίνος**
Αριθμός Μητρώου	**ΜΠΠΛ / 18061**
Επιβλέπων	**Παναγιωτόπουλος Θεμιστοκλής, Καθηγητής**

Ημερομηνία Παράδοσης **Ιούλιος 2022**

Τριμελής Εξεταστική Επιτροπή

Θεμιστοκλής Παναγιωτόπουλος	Διονύσιος Σωτηρόπουλος	Ιωάννης Τασούλας
Καθηγητής	Επίκουρος Καθηγητής	Επίκουρος Καθηγητής

Περιεχόμενα

Περίληψη

Σκοπός της εργασίας είναι η υλοποίηση ενός 3D Role Playing Game για παιδιά 6ης Δημοτικού, μέσα από το οποίο θα διασκεδάσουν, θα δοκιμαστούν σε ερωτήσεις Μαθηματικών και θα αποκτήσουν μια πρώτη επαφή με τον κόσμο των παιχνιδιών σε ηλεκτρονικό υπολογιστή. Στο **κεφάλαιο 1** θα γίνει μια σύντομη αναδρομή στην ιστορία των ηλεκτρονικών παιχνιδιών, την εξέλιξή τους, καθώς και στα παιχνίδια Role Playing. Στο **κεφάλαιο 2** θα δοθεί βάση στην πλατφόρμα Unity και στις λειτουργίες της με σκοπό την κατανόηση της. Στο **κεφάλαιο 3** θα παρουσιαστεί μία αναλυτική περιγραφή του παιχνιδιού (Σενάριο, Οντότητες-Κόσμος, Κανόνες και Ροή Παιχνιδιού). Στο **κεφάλαιο 4** θα μας απασχολήσει ο σχεδιασμός και η ανάλυση του παιχνιδιού. Στο **κεφάλαιο 5** θα παρουσιάσουμε τον τρόπο υλοποίησης του παιχνιδιού μέσα από εικόνες και αναλύοντας τον κώδικα που χρησιμοποιήσαμε. Η υλοποίηση της παρούσας εργασίας πραγματοποιήθηκε χρησιμοποιώντας την **πλατφόρμα Unity 2021.1.5f1** για το στήσιμο του γραφικού περιβάλλοντος του παιχνιδιού, καθώς και την υλοποίηση βασικών λειτουργιών, το **Visual Studio 2019 (IDE)** μέσα στο οποίο στήθηκε η αλληλεπίδραση των χαρακτήρων με το περιβάλλον του παιχνιδιού μέσα από Scripts που γράφτηκαν σε γλώσσα προγραμματισμού C# και το **DB Browser For SQLite** στο οποίο στήθηκε η βάση δεδομένων του παιχνιδιού.

Abstract

The purpose of this project is to implement a 3D Role Playing Game for 6th graders, through which they will have fun, will be tested in Math questions and will gain a first contact with the world of computer games. **Chapter 1** will give a brief overview of the history of electronic games, their evolution, as well as Role Playing games. **Chapter 2** will provide an overview of the Unity platform and its functions to understand it. **Chapter 3** will present a detailed description of the game (Scenario, Entities-World, Rules, and Game Flow). In **chapter 4** we will deal with the design and analysis of the game. In **chapter 5** we will present how to implement the game through images and analyzing the code we used. This work was carried out using the **Unity 2021.1.5f1** platform to set up the game's graphical interface, as well as the implementation of key functions, **Visual Studio 2019 (IDE)** in which the characters interacted with the game environment in from Scripts written in C # programming language and **DB Browser For SQLite** in which the game database was set up.

Πίνακας Εικόνων

Αναδρομή στην Ιστορία των Βιντεοπαιχνιδιών

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 ζωή των ανθρώπων οι υπολογιστές, συγκεκριμένα οι κεντρικοί υπολογιστές.

Από εκείνη την περίοδο, τα βιντεοπαιχνίδια έκαναν την εμφάνιση τους, στις κονσόλες, στα φλίπερ, στους υπολογιστές, αλλά και στις φορητές κονσόλες.

Από τη στιγμή που οι υπολογιστές και οι κονσόλες ήρθαν σε επαφή μεγάλο κοινό, αποφάσισαν να κάνουν κάτι πρωτότυπο. Δημιούργησαν τις λεγόμενες φορητές κονσόλες. Αυτή η κίνηση δεν ήταν καλοδεχούμενη από τις αμερικάνικες εταιρίες, καθώς μεγαλύτερη επιρροή στη βιομηχανία των παιχνιδιών είχαν οι ιαπωνικές εταιρείες. Αυτές κατασκεύασαν δημοφιλείς κονσόλες όπως το PlayStation της Sony και το NES της Nintendo. Σήμερα, βιντεοπαιχνίδια εμφανίζονται και στα κινητά τηλέφωνα, τα tablets και τους προσωπικούς ψηφιακούς οδηγούς.

Στις μέρες μας οι πωλήσεις των video games ανταγωνίζονται δυναμικά τις κινηματογραφικές ταινίες, έχοντας εξελιχθεί σε μια από τις πιο επικερδείς βιομηχανίες παγκοσμίως.

Η εξέλιξη των video games

Το πρώτο παιχνίδι - μηχάνημα που διαφημίστηκε επίσημα, ήταν το **Computer Space**, το 1971, όπου τα video games ξεκίνησαν την μακρόχρονη πορεία τους σαν ένα μέσο εμπορικής διασκέδασης.

Το παιχνίδι αυτό αποτέλεσε την αφετηρία των arcade παιχνιδομηχανών που λειτουργούσαν με κέρματα με μεγαλύτερη επιτυχία το παιχνίδι PONG το 1972 από την Atari. Μέσα σε ένα χρόνο τα έσοδα της Atari ήταν τεράστια και ενέπνευσε τη δημιουργία αμέτρητων αντιγραφών ή παρόμοιων παιχνιδιών. Για μεγάλο διάστημα, το να έλεγε κανείς ότι παίζει βίντεο-παιχνίδια σήμαινε ότι παίζει PONG.

Μεγάλη επιτυχία γνώρισαν, επίσης, κι άλλα παιχνίδια τύπου arcade, όπως το **Asteroids** και το **Space Invaders (1978),** αλλά και το **Pac-Man (1980)**, το οποίο πέρασε στην ιστορία ως το δεύτερο παιχνίδι που προκάλεσε έλλειψη νομισμάτων στην Ιαπωνία, ενώ ήταν το πρώτο παιχνίδι που έγινε δημοφιλές και στο γυναικείο φύλο. Παράλληλα το 1980 η Atari λανσάρει το **Battlezone**, το πρώτο βιντεοπαιχνίδι τύπου arcade τριών διαστάσεων.

Στις αρχές της δεκαετίας του 1980 παρατηρείται η πτώση των βιντεοπαιχνιδιών ή, όπως ονομάστηκε, **The great videogame crash**. Οι λόγοι συναντώνται στην εμφάνιση των προσωπικών υπολογιστών μέσα στο σπίτι και σε κάποια λάθη των εταιριών κονσόλας στην αγορά. Από το ΄80 και μετά εμφανίζονται ηλεκτρονικά παιχνίδια τα οποία είναι σχεδιασμένα, για να παίζονται στους υπολογιστές, όπως το **ZX Spectrum** από την εταιρία Sinclair Research και το **Commodore 64** από την Commodore International το 1982.

Εκείνη την χρονιά έκανε την εμφάνισή του το παιχνίδι **Zork** για τον υπολογιστή Apple II. Βέβαια, το 1984 οι κονσόλες επέστρεψαν στην αγορά με την Nintendo με τα πασίγνωστα παιχνίδια, όπως **Super Mario Bros**, **The Legend of Zelda** και η Sega λανσάρει την κονσόλα Genesis, την οποία ακολούθησε το γνωστό Game Boy. Ένα από τα πιο δημοφιλή και εθιστικά παιχνίδια ήταν το **Tetris** που δημιουργήθηκε από τον Alexei Pajitnov το 1985, όταν εργαζόταν στην Ακαδημία των Επιστημών στην Μόσχα. Εμφανιζόταν σε πολλές πλατφόρμες, αλλά κυριότερα στο Game Boy της Nintendo το 1989, πουλώντας 33 εκατομμύρια αντίγραφα και καθιστώντας το ως το πιο γνωστό παιχνίδι στον κόσμο.

Οι τόσες πολλές επιλογές παιχνιδιών έθεσαν στο περιθώριο της arcade παιχνιδομηχανές, αλλά η εμφάνιση του Street Fighter το 1991 και του Mortal Kombat το 1992 τόνωσαν ξανά αυτές τις παιχνιδομηχανές. Τα παιχνίδια αυτά παρουσίαζαν βίαιο υλικό και από το 1993 ξεκινά μία συζήτηση σχετικά με τη βία στα βίντεο-παιχνίδια και την επίδραση που έχει στα παιδιά.

Η ανάπτυξη των ηλεκτρονικών παιχνιδιών στους υπολογιστές συνέχιζε, αλλά η εγκατάσταση των παιχνιδιών με όρους σημερινούς ήταν δύσκολη, καθώς τα windows δεν ήταν διαδεδομένα και οι χρήστες έπρεπε να δουλεύουν με DOS.

Από τα τέλη της δεκαετίας του 1980 και τις αρχές του ΄90 η βιομηχανία του παιχνιδιού μέσω υπολογιστή εξελίσσεται και γίνεται πιο επαγγελματική δουλειά, καθώς η τεχνολογία προχωράει, με τον ήχο, τα γραφικά, τον προγραμματισμό, τον σχεδιασμό-animation, το μάρκετινγκ και την παραγωγή.

Το 1990 η επιτυχία του **SimCity**, ήταν τεράστια και από εκείνο το διάστημα και μετά εμφανίζονται πληθώρα παιχνιδιών. Κυριότερα και πιο δημοφιλή ήταν το **Prince of Persia**(1989), το Myst (1993), το **Doom** καθώς και το **Elder Scrolls** (1994).

Η ανάπτυξη των υπολογιστών, όσον αφορά τα βελτιωμένα γραφικά τους και τον ήχο, σε συνδυασμό με την σύνδεση στο διαδίκτυο κάνει πολύ γνωστά τα παιχνίδια MMORPG (Massively Multiplayer Online Role Playing Games) τα οποία παίζονται αποκλειστικά στο διαδίκτυο από πολλούς παίκτες που συνδέονται σε έναν σέρβερ.

Από τα πρώτα παιχνίδια αυτού του τύπου ήταν το **Meridian 59** το 1996 και το **World of Warcraft**, με πάνω από 8 εκατομμύρια παίκτες ανά τον κόσμο.

Παράλληλα, οι κονσόλες αναβαθμίζονται εκ νέου με την εισαγωγή στην αγορά του PlayStation από την Sony και του Xbox , τα οποία πλέον μπορούν να καλύψουν ορισμένες δυνατότητες του υπολογιστή, όπως αναπαραγωγή ταινιών, μουσικής και σύνδεση στο ίντερνετ. Το 2006 η Nintendo εισήγαγε το Wii, ένα παιχνίδι με τηλεχειριστήριο με αισθητήρα κινήσεων και παιχνίδια που σχετίζονται με τα αθλήματα (exergames).

Τέλος, τα τελευταία χρόνια τα έξυπνα τηλέφωνα ή smartphones με σύνδεση στο διαδίκτυο έχουν ενσωματώσει πολλά γνωστά παιχνίδια, ενώ πολλά έχουν δημιουργηθεί για τα κινητά τηλέφωνα. Πολλά από αυτά είναι το **Angry Birds**, το **Clash of Clans** και το **Clash Royale**, με τα οποία ασχολούνται εκατομμύρια ανθρώπων σε καθημερινή βάση, μιας και αυτά τα παιχνίδια έχουν το χαρακτηριστικό να χρειάζονται μία καθημερινή διαχείριση, ώστε να μαζεύεις πόντους και να προχωράς στο παιχνίδι.

Role Playing Games (RPG)

Role-Playing Game ή RPG ονομάζεται το είδος βιντεοπαιχνιδιού στο οποίο η ιστορία εξελίσσεται με την εξερεύνηση του κόσμου του παιχνιδιού και οι μάχες διαδραματίζονται με τη μορφή εντολών. Δηλαδή, είναι ένα είδος παιχνιδιού στο οποίο οι παίκτες αναλαμβάνουν το ρόλο φανταστικών χαρακτήρων και μέσω συνεργασίας δημιουργούν ή παρακολουθούν ιστορίες.

Το είδος παιχνιδιών ρόλων ξεκίνησε το μέσα της δεκαετίας του '70 στους κεντρικούς υπολογιστές, εμπνευσμένη από τα επιτραπέζια (pen-and-paper) παιχνίδια ρόλων όπως το Dungeons & Dragons. Άλλες πηγές έμπνευσης για τα πρώιμα παιχνίδια του είδους περιλαμβάνουν τα επιτραπέζια παιχνίδια πολέμου, τα παιχνίδια εξομοίωσης σπορ, τα παιχνίδια περιπέτειας, όπως το Colossal Cave Adventure, παραδοσιακά παιχνίδια στρατηγικής όπως το σκάκι, αλλά και τα έργα του Άγγλου συγγραφέα φαντασίας, Τζ. Ρ. Ρ. Τόλκιν.

Ενώ τα πρώτα παιχνίδια ρόλων προσέφεραν την εμπειρία ενός παίκτη, στις αρχές με μέσα της δεκαετίας του '90 γνώρισαν άνθιση τα παιχνίδια ρόλων με πολλούς παίκτες, όπως τα Diablo και Secret of Mana. Με την έλευση του ίντερνετ, τα παιχνίδια με πολλούς παίκτες δημιούργησαν ένα νέο είδος βιντεοπαιχνιδιών, τα μαζικά παιχνίδια ρόλων πολλαπλών παικτών, όπως τα Final Fantasy XI, Lineage και World of Warcraft.

Άλλο ένα είδος που προέκυψε είναι τα παιχνίδια δράσης ρόλων, όπου ο παίκτης χειρίζεται άμεσα τον βασικό χαρακτήρα, δηλαδή χωρίς τη χρήση εντολών, και δίνει έμφαση στις μάχες που διεξάγονται πάντα σε πραγματικό χρόνο και τη δράση, ενώ η αλληλεπίδραση των χαρακτήρων είναι μικρότερη σε σχέση με τα παραδοσιακά παιχνίδια μάχης. Τα πρώιμα παιχνίδια δράσης ρόλων ακολούθησαν το πρότυπο που δημιουργήθηκε τη δεκαετία του '80 από την ιαπωνική εταιρία ανάπτυξης βιντεοπαιχνιδιών Nihon Falcom, με τίτλους όπως τα Dragon Slayer και τη σειρά Ys, όπου η μάχη διεξάγεται στο στυλ hack and slash και οι κινήσεις και οι δράσεις του χαρακτήρα γίνονται απευθείας από τον παίκτη με τη χρήση του χειριστηρίου ή του πληκτρολογίου, παρά των μενού μάχης.

Αυτή η φόρμουλα βελτιώθηκε με το παιχνίδι δράσης περιπέτειας του 1986, The Legend of Zelda, το οποίο έθεσε το πρότυπο που χρησιμοποιήθηκε από πολλά επακόλουθα παιχνίδια. Αυτό περιλαμβάνει καινοτομίες όπως ο ανοικτός κόσμος, το μη γραμμικό σενάριο, την εξοικονόμηση μπαταρίας, και ένα κουμπί επίθεσης για χτύπημα με ξίφος. Το παιχνίδι ευθύνεται σε μεγάλο βαθμό για την μεγάλη έκρηξη των παιχνιδιών δράσης ρόλων που κυκλοφόρησαν τα τέλη της δεκαετίας του 1980 τόσο στην Ιαπωνία όσο και τη Βόρεια Αμερική. Η σειρά Legend of Zelda συνέχισε να ασκεί επιρροή τις επόμενες δεκαετίες στη μετατροπή των παιχνιδιών ρόλων από στατικά, σε παιχνίδια όπου οι μάχες διεξάγονται σε πραγματικό χρόνο.

Unity

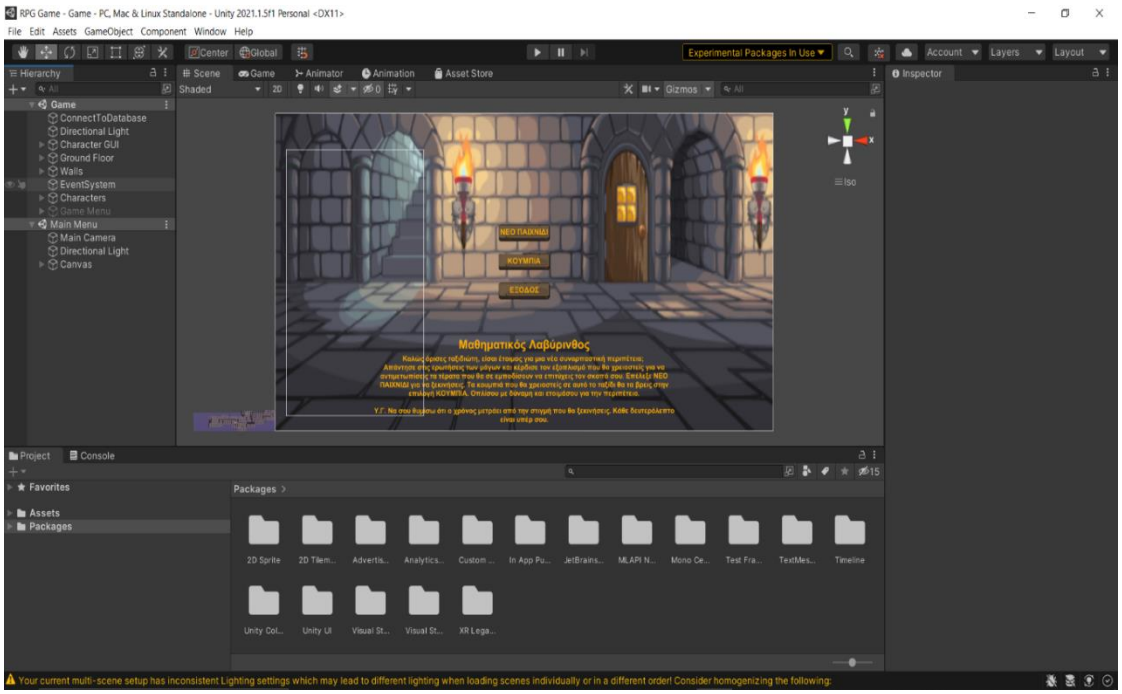
Στο κεφάλαιο αυτό θα γίνει μια περιληπτική περιγραφή της πλατφόρμας Unity 2021.1.5f1 εξηγώντας παράλληλα και το πως υλοποιήθηκε το παιχνίδι μας, ώστε να γίνουν κατανοητές οι δυνατότητες που παρέχει η πλατφόρμα στον χρήστη, καθώς και ο τρόπος ανάπτυξης ενός παιχνιδιού.

Με την ευκαιρία να αναφέρουμε ότι εκτός από το Unity, μια ακόμα μεγάλη πλατφόρμα δημιουργίας παιχνιδιών είναι το Unreal Engine. Για την παρούσα εργασία προτιμήθηκε το Unity καθώς η γλώσσα προγραμματισμού που χρησιμοποιεί είναι η C#, σε αντίθεση με την Unreal Engine που χρησιμοποιεί C++.

User Interface

Εισαγωγή στο UI του Unity

Ας ξεκινήσουμε με τα βασικά, δηλαδή το περιβάλλον στο οποίο ο χρήστης (προγραμματιστής) καλείται να χρησιμοποιήσει για τη δημιουργία του παιχνιδιού του. Ανοίγοντας το Unity ο χρήστης, και πιο συγκεκριμένα ανοίγοντας το παιχνίδι που περιγράψαμε παραπάνω, βλέπει την παρακάτω εικόνα.

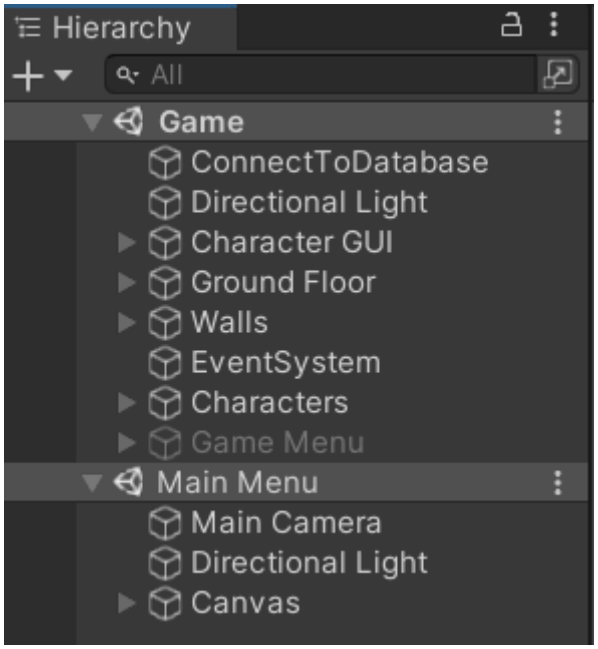


Εικόνα 1 - Unity - UI

Είναι λογικό η παραπάνω εικόνα να φαίνεται σύνθετη και δύσκολα κατανοήσιμη, όμως όλα θα εξηγηθούν στη συνέχεια.

Hierarchy Tab

Ας το πάρουμε λοιπόν από την αρχή. Στο αριστερό μέρος της εικόνας εμφανίζεται η καρτέλα **Hierarchy**.

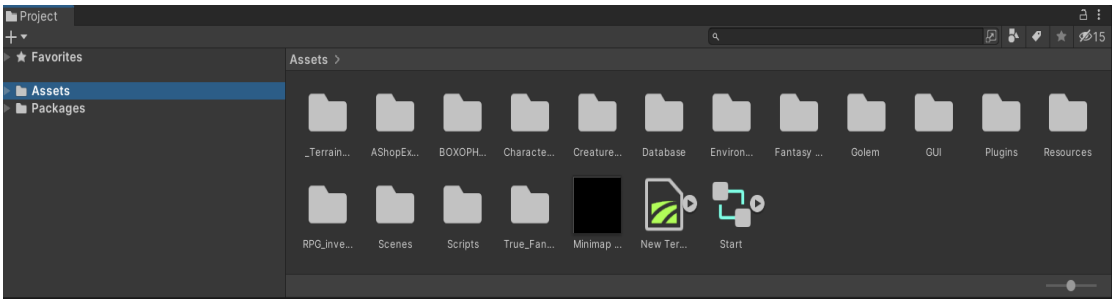


Εικόνα 2 - Unity - Hierarchy Tab

Στη συγκεκριμένη καρτέλα εμφανίζονται όλες οι σκηνές (**Scenes**) του παιχνιδιού ως βασική κατηγοριοποίηση (Game – Main Menu) και κάτω από κάθε σκηνή εμφανίζονται όλα τα αντικείμενα (**Objects**)- στη περίπτωση μας χωρισμένα ανά κατηγορία - που χρησιμοποιεί η κάθε μία.

Με λίγα λόγια στο συγκεκριμένο tab υπάρχουν όλα τα Game Objects που έχουν χρησιμοποιηθεί στην υλοποίηση του παιχνιδιού.

Project tab

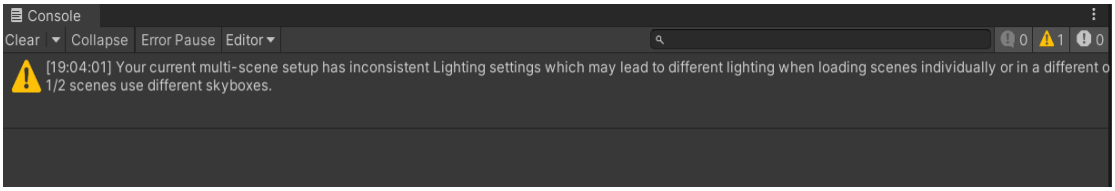


Εικόνα 3 - Unity - Project Tab

Στη καρτέλα αυτή όπως είναι εύκολα κατανοητό και από το όνομα της, υπάρχουν όλα τα Assets του παιχνιδιού. Με τον όρο Assets εννοούμε οτιδήποτε έχει χρησιμοποιηθεί για το στήσιμο του παιχνιδιού, όπως για παράδειγμα γραφικά, animations, animators, scripts κτλπ.

Μια χρήσιμη υποσημείωση είναι ότι ο φάκελος αυτός θα πρέπει να περιέχει μόνο τα απαραίτητα Assets για το παιχνίδι, γιατί κάθε τι περιττό αυξάνει το μέγεθος του παιχνιδιού.

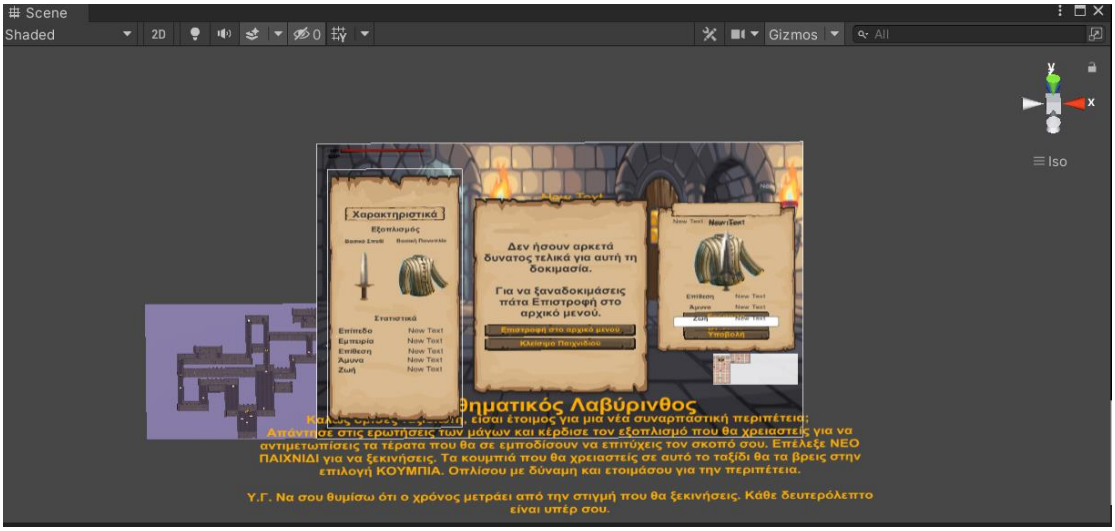
Console Tab



Εικόνα 4 - Unity - Console Tab

Τον όρο **Console** τον συναντάει πολύ συχνά ένα προγραμματιστής (σε κάθε IDE που χρησιμοποιεί υπάρχει σίγουρα Console) και είναι το βασικό εργαλείο για debugging. Εδώ εμφανίζονται όλα τα σφάλματα που αφορούν εσωτερικά την Unity και τον Compiler της, compile/syntax errors που αφορούν τα scripts τα οποία γράφουμε, warnings τα οποία μας ενημερώνουν για τυχόν ρυθμίσεις οι οποίες ενδέχεται να προκαλέσουν errors σε runtime ή μεταβλητές.

Scene Tab

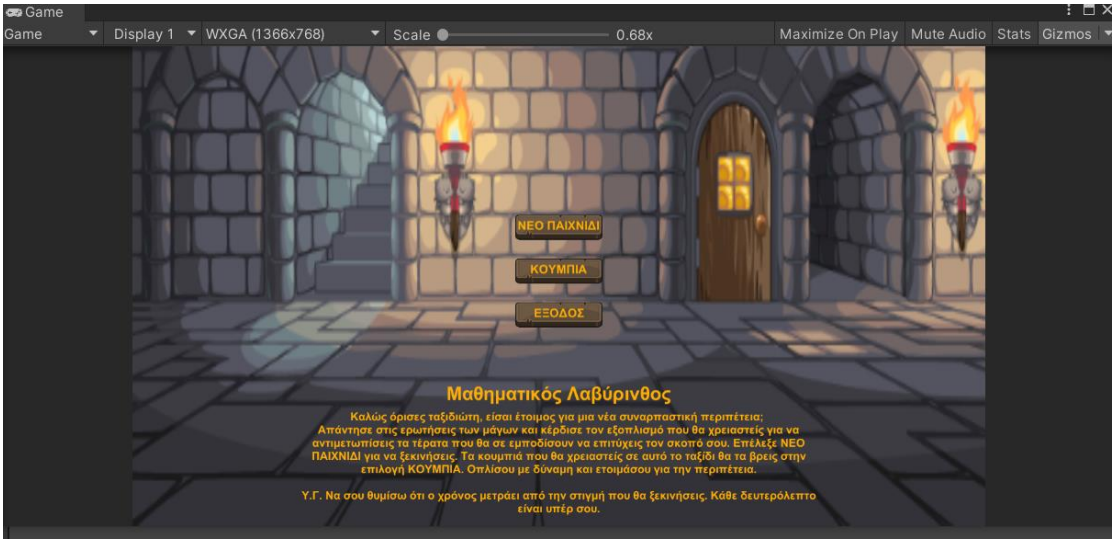


Εικόνα 5 - Unity - Scene Tab

Η καρτέλα αυτή χρησιμοποιείται για την σχεδίαση του κάθε Scene του παιχνιδιού, ή αλλιώς των γραφικών που περιέχονται σε κάθε «σκηνή» του παιχνιδιού. Χρησιμοποιώντας το ποντίκι ή το πληκτρολόγιο ο χρήστης έχει τη δυνατότητα να περιηγηθεί μέσα στη σκηνή του παιχνιδιού, να προσθέσει είτε αντικείμενα είτε UI Components(όπως κουμπιά, πλαίσια, πεδία κειμένου κτλπ), να τα επεξεργαστεί ή και να τα μετακινήσει όπως εκείνος επιθυμεί.

Μια σημαντική παρατήρηση για τη συγκεκριμένη καρτέλα είναι ότι εδώ ο χρήστης δεν μπορεί να δει πως παίζει το παιχνίδι, αλλά μόνο να το επεξεργαστεί.

Game Tab

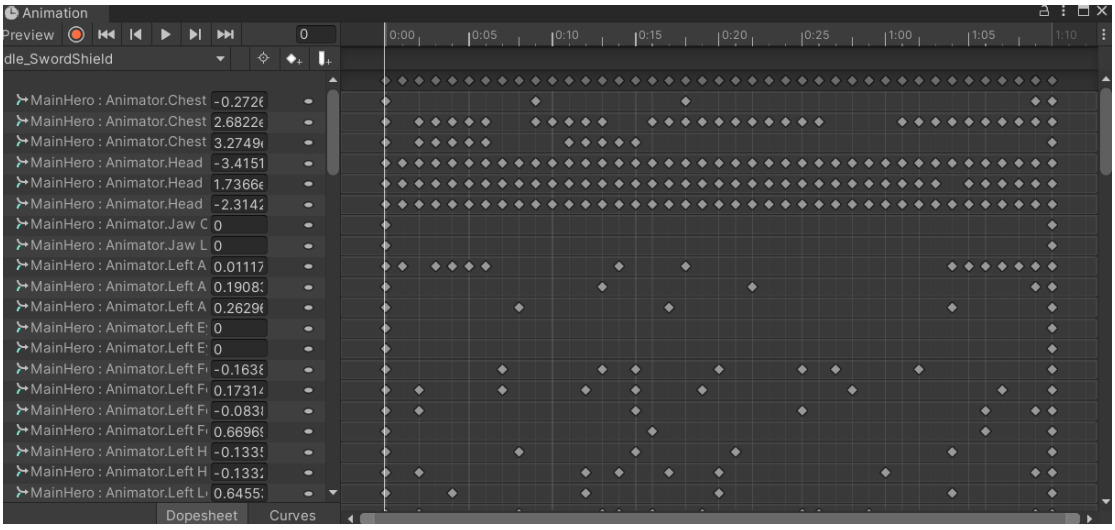


Εικόνα 6 - Unity - Game Tab

Αυτή η καρτέλα ουσιαστικά είναι το παιχνίδι μας. Εδώ ο χρήστης βλέπει το πως είναι πραγματικά το παιχνίδι που έχει φτιάξει. Μπορεί να δοκιμάσει όποια αντικείμενα και λειτουργίες έχει προσθέσει, να κινηθεί μέσα στο χάρτη, να περιηγηθεί στα αντίστοιχα menu που έχει σχεδιάσει ώστε να βρει τις βελτιώσεις που χρειάζονται και να ελέγξει ότι όλα λειτουργούν όπως επιθυμεί.

Αξίζει να σημειωθεί ότι δεν μπορούν να πραγματοποιηθούν αλλαγές στο παιχνίδι μέσα από αυτό το tab. Για να κάνει αλλαγές ο χρήστης, θα πρέπει να επιστρέψει στο Scene Tab.

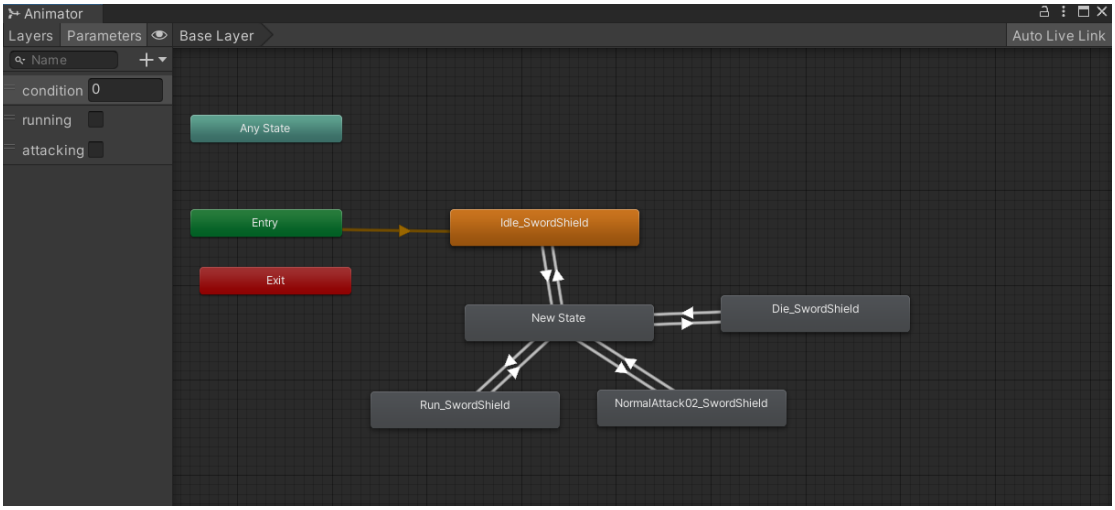
Animation Tab



Εικόνα 7 - Unity - Animation Tab

Το συγκεκριμένο Tab υπάρχει και χρησιμοποιείται για τα Objects τα οποία δρουν μέσα στο παιχνίδι. Εδώ μπορεί ο προγραμματιστής να επεξεργαστεί το πως θα αντιδρά κάθε αντικείμενο ανάλογα με την κατάσταση στην οποία βρίσκεται. Δηλαδή για παράδειγμα, αν ο κύριος χαρακτήρας του παιχνιδιού κινείται, εδώ μπορούμε να ορίσουμε τον τρόπο που θα κινείται, το πόσο γρήγορα θα γίνεται η κίνηση και τι διάρκεια θα έχει.

Animator Tab

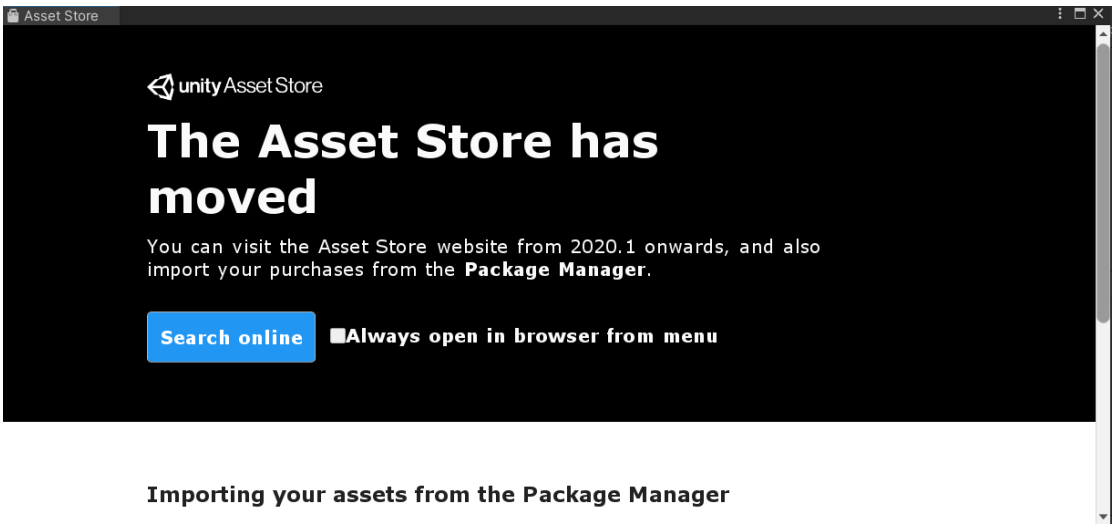


Εικόνα 8 - Unity - Animator Tab

Η καρτέλα αυτή μπορεί να θεωρηθεί και ως η επέκταση της καρτέλας Animations. Εφόσον ολοκληρωθεί η υλοποίηση των animations, επόμενο βήμα είναι η τοποθέτηση τους με τέτοιο τρόπο ώστε να υπάρχει σωστή αλληλεπίδραση του χρήστη με τον χαρακτήρα. Δηλαδή, πατώντας για παράδειγμα το κουμπί W στη δικιά μας περίπτωση, ο χαρακτήρας θα πρέπει να εκτελεί το animation της κίνησης. Αυτό επιτυγχάνεται κατά ένα μεγάλο ποσοστό μέσα από αυτή τη καρτέλα.

Μια σημαντική σημείωση εδώ είναι ότι βασικό ρόλο στον τρόπο λειτουργίας αυτής της καρτέλας παίζει και το κομμάτι του κώδικα που αναφέρεται σε κάθε δράση (κίνηση, επίθεση κτλπ).

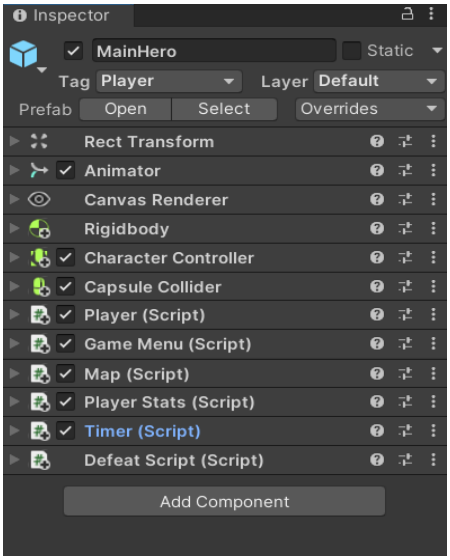
Asset Store



Εικόνα 9 - Unity - Asset Store

Αυτό το Tab μπορεί να θεωρηθεί και ως η πηγή. Εδώ μπορεί ο προγραμματιστής να βρει Assets (δωρεάν ή επί πληρωμή) για το παιχνίδι του.

Inspector Tab



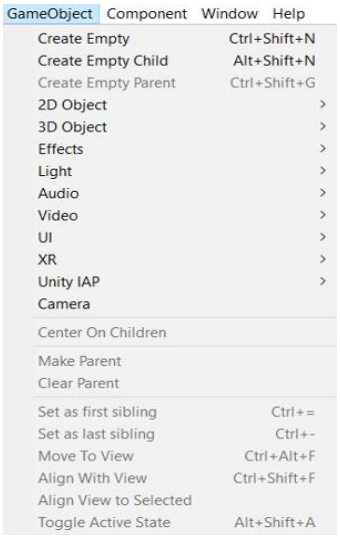
Εικόνα 10 - Unity - Inspector Tab

Ίσως από τα πιο σημαντικά παράθυρα του Unity. Ανάλογα με το Game Object που θα επιλέξουμε, ο Inspector μας δείχνει όλα τα συστατικά του (Components). Υπάρχει μια μεγάλη λίστα από Components που μπορεί να υπάρχουν μέσα σε ένα Game Object, κάποια από τα πιο βασικά που μπορούμε να δούμε και στην παραπάνω εικόνα είναι τα C# Scripts, ο Character Controller και ο Animator.

Game Objects

Η έννοια ενός Game Object (GO)

Ως μια γενική έννοια του τι είναι το Game Object, μπορούμε να ορίσουμε ότι είναι οτιδήποτε χρησιμοποιείται για την υλοποίηση των γραφικών του παιχνιδιού. Όπως είδαμε και παραπάνω όλα τα Game Objects εμφανίζονται στο Hierarchy Tab. Ένα GO μπορεί να περιέχει και άλλα ή να είναι αυτόνομο.



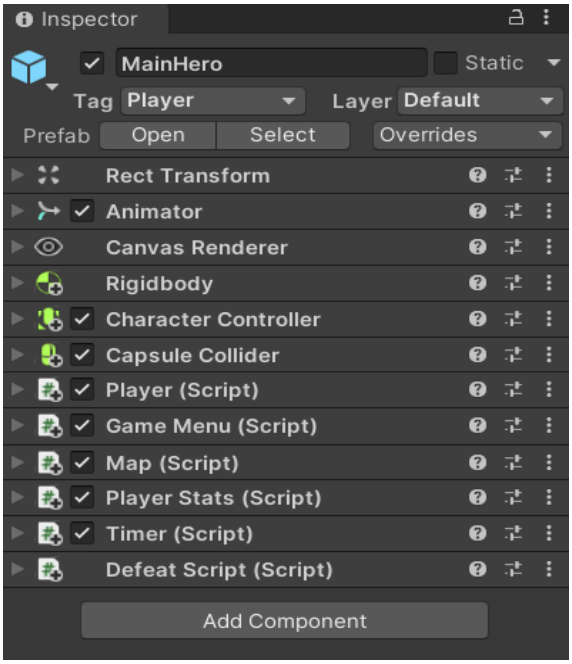
Εικόνα 11 - Unity - Game Object Options

Όπως βλέπουμε και στην εικόνα υπάρχουν πολλών ειδών Game Objects που μπορούμε να φτιάξουμε μέσα σε ένα παιχνίδι. Εύκολα διακρίνουμε ότι οι κατηγορίες που είναι χωρισμένα τα Game Objects ομολογούν και την χρησιμότητα του καθενός.

Ανάλυση ενός Game Object και τα Components

Όπως είδαμε και στην ενότητα του UI του Unity, όλα τα Components από τα οποία αποτελείται ένα GO εμφανίζονται στην καρτέλα Inspector και μαρτυρούν τη λειτουργία του.

Για να γίνει πιο κατανοητή η έννοια του Component ας αναλύσουμε ως παράδειγμα την καρτέλα Inspector του κύριου χαρακτήρα μας, η οποία είναι και η πιο σύνθετη.



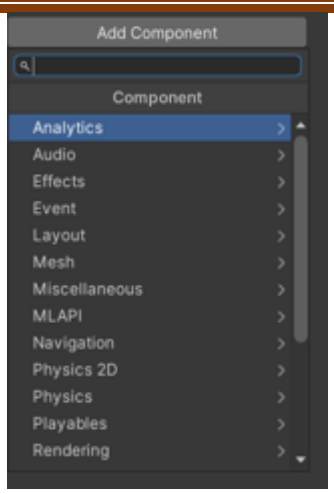
Εικόνα 12 - Unity - Game Object

Τα βασικά Components ενός GO όπως εμφανίζονται και παραπάνω είναι τα εξής:

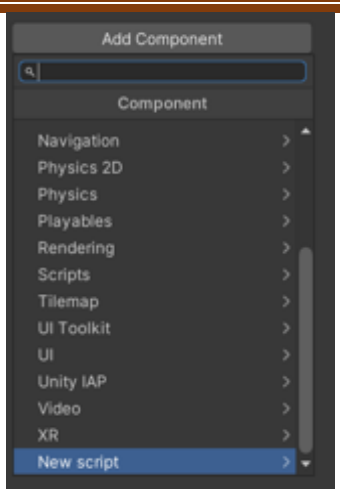
- **Όνομα :** Κάθε GO έχει ένα όνομα ώστε να ξεχωρίζουν μεταξύ τους. Καλό είναι ο προγραμματιστής να ορίζει εκείνος τα ονόματα ώστε να είναι πιο κατανοητή η σχεδίαση του παιχνιδιού και να είναι εύκολο στη συντήρηση και στην βελτίωσή του στο μέλλον.
- **Tag :** Με τον όρο Tag αναφερόμαστε στην ταμπέλα του κάθε GO, η οποία μας βοηθάει στην κατηγοριοποίηση των GO.
- **Layer :** Τα Layers χρησιμοποιούνται και αυτά για να ομαδοποιήσουν τα GO, όμως διαφέρουν από τα Tags στο γεγονός ότι το Layer χρησιμοποιείται κυρίως από τις κάμερες και αναφέρεται στο τι αυτές θα δείξουν.
- **Prefab:** Το Prefab δίνει τη δυνατότητα στον χρήστη να φτιάχνει ένα GO όπως εκείνος επιθυμεί και να το αποθηκεύει σαν μια δομή η οποία μπορεί να χρησιμοποιηθεί για GO τα οποία μοιάζουν μεταξύ τους.

Εκτός από τα παραπάνω, τα οποία είναι σταθερά για όλα τα GO, υπάρχουν και κάποια τα οποία μπορεί να τα προσθέσει ο προγραμματιστής για την σωστή και σταθερή λειτουργία αυτού.

Πατώντας το κουμπί «**Add Component**» όπως αυτό εμφανίζεται στην παραπάνω εικόνα, ο χρήστης μπορεί να επιλέξει από μία ποικιλία από Components ομαδοποιημένα ανάλογα με την λειτουργία του καθενός.



Εικόνα 13 - Unity - Game Object Components



Εικόνα 14 - Unity - Game Object Components

Γίνεται αντιληπτό εύκολα ότι ένα GO έχει άπειρες δυνατότητες και αυτό φαίνεται και από την ποικιλία των Components που μπορούν να του προστεθούν.

Συνεχίζοντας, από το παράδειγμα που θέσαμε (του κύριου χαρακτήρα μας) ήρθε η ώρα να εξηγήσουμε τα Components που έχουν χρησιμοποιηθεί.

- **React Transform :** Το React Transform χρησιμοποιείται για την διαχείριση και την αποθήκευση της θέσης, του μεγέθους, τη σταθερότητα ενός GO.
- **Animator Controller :** Εδώ προστίθεται ο Animator που έχει σχεδιαστεί στην αντίστοιχη καρτέλα, ώστε να μπορεί να χρησιμοποιηθεί από το GO.
- **Canvas Renderer :** Το συγκεκριμένο Component υπάρχει σε όλα τα GO και είναι αυτό που το καθιστά ως UI element.
- **Character Controller :** Χρησιμοποιείται για τον κύριο έλεγχο της κίνησης του χαρακτήρα πάνω στο περιβάλλον. Δεν επηρεάζεται ούτε επηρεάζει τα φυσικά χαρακτηριστικά του παιχνιδιού.
- **Rigidbody :** Έχει ακριβώς την ίδια λειτουργία με τον Character Controller, αφού και αυτό ελέγχει την κίνηση του χαρακτήρα. Η διαφορά τους είναι ότι ο έλεγχος της θέσης ενός αντικειμένου γίνεται μέσω προσομοίωσης φυσικής σε αυτή τη περίπτωση.
- **Capsule Collider (Colliders) :** Οι Colliders χρησιμοποιούνται για να ορίσουμε το χώρο που καταλαμβάνει ένα GO μέσα στο παιχνίδι. Συνδυάζονται με το Rigidbody ώστε να επιτύχουν τη προσομοίωση της φυσικής μέσα στο περιβάλλον του παιχνιδιού.
- **Script C# :** Τα Scripts είναι αυτά τα οποία δίνουν «ζωή» σε ένα GO. Μέσα από τα Scripts δημιουργείται η αλληλεπίδραση του χρήστη με τον χαρακτήρα, όπως θα δούμε και στη συνέχεια

Scripts

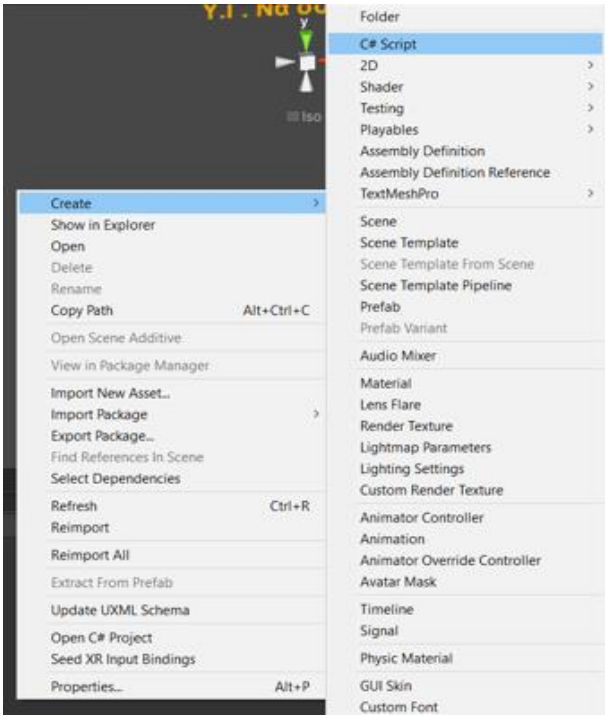
Δημιουργία ενός Script

Ας κάνουμε μια σύντομη αναδρομή στο τι είναι τεχνικά ένα Script. Το Script είναι ένα αρχείο .cs (csharp) το οποίο περιλαμβάνει κώδικα γραμμένο σε γλώσσα προγραμματισμού C#. Μέσα από αυτό το αρχείο μπορούμε να υλοποιήσουμε όλο το Game Play του παιχνιδιού μας. Ως Game Play μπορούμε να ορίσουμε τη κίνηση του παίχτη, τη λειτουργία ενός menu, την αυτόματη επίδραση ενός αντικείμενου που δεν χειρίζεται ο χρήστης μέσα στο παιχνίδι και γενικά ότι έχει να κάνει με τις λειτουργίες και τον τρόπο παιχνιδιού.

Μπορείς να φτιάξεις ένα αυτόνομο Script γενικής χρήσης που θα το προσθέτεις σε κάθε GO όπου είναι απαραίτητο, είτε να φτιάξεις Script συγκεκριμένα για ένα και μόνο GO.

Ας δούμε ένα παράδειγμα το οποίο θα αναλύσουμε στη συνέχεια για το πως φτιάχνουμε ένα script και τι περιέχει ένα καινούριο Script.

Για να φτιάξουμε ένα Script πάμε στα Assets του παιχνιδιού μας και κάνοντας δεξί κλικ θα δούμε την παρακάτω εικόνα.



Εικόνα 15 - Unity - Create Script

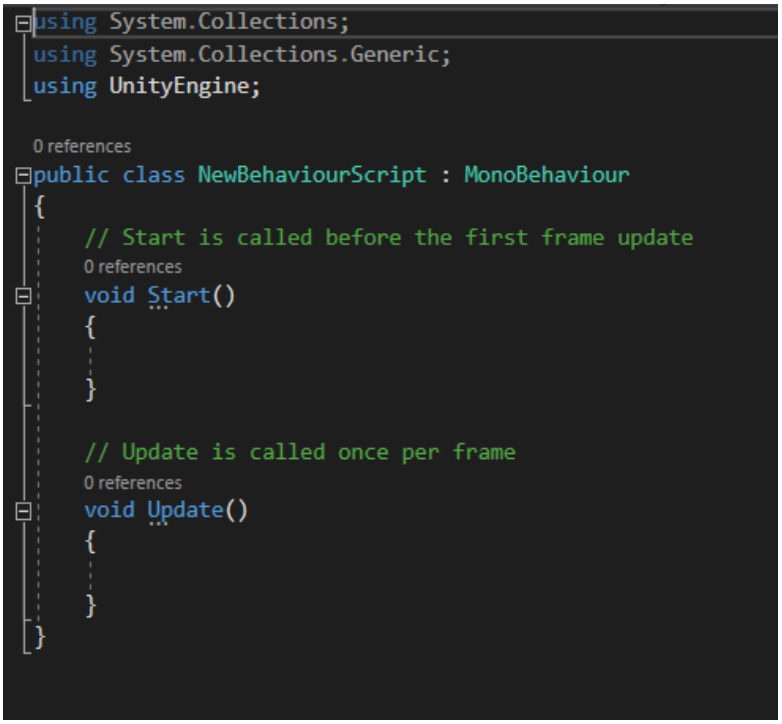
Κάνοντας Create-> C# Script φτιάχνεται ένα νέο asset (βάζοντας όνομα Example) το οποίο έχει την εξής εικόνα.

Εικόνα 16 - Unity - Script Icon

Ανοίγοντας αυτό Script, ξεκινάει το Visual Studio το οποίο το φορτώνει και σε αφήνει να το επεξεργαστείς.

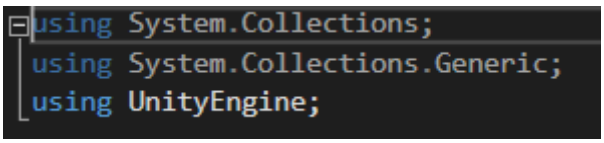
Ανάλυση ενός Script

Η πρώτη εικόνα που θα δει ο προγραμματιστής (είναι και αυτή που θα αναλύσουμε προς το παρόν) είναι η κάτωθι:



Εικόνα 17 - Unity - Script

Όπως βλέπουμε, στις πρώτες 3 γραμμές, φορτώνονται 3 βιβλιοθήκες (**System.Collections, System.Collections.Generic, Unity Engine**).



Εικόνα 18 - Unity - Script Default Libraries

Φυσικά ο προγραμματιστής μπορεί να προσθέσει ή και να αφαιρέσει όσες εκείνος επιθυμεί.

Μια βασική λεπτομέρεια που πρέπει να προσέξουμε είναι ο ορισμός της κλάσης **NewBehaviourScript** η οποία είναι και η κύρια κλάση του Script. Όπως μπορούμε να διακρίνουμε η κλάση που αναφέραμε «κληρονομεί» την κλάση **MonoBehaviour.** Αυτή είναι η βασική κλάση που όλα τα Unity Scripts κληρονομούν.

Η κλάση αυτή παρέχει αρκετές τιμές και μεθόδους που μπορεί να χρησιμοποιήσει ένα Script. Δύο από αυτές τις μεθόδους είναι και αυτές που εμφανίζονται στην εικόνα, δηλαδή η **Start()** και η **Update().**

- **Start()**

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

}
```

Εικόνα 19 - Unity - Script Start function

Όπως αναφέρεται και στην είναι στα σχόλια η μέθοδος αυτή καλείται μία φορά κατά την αρχικοποίηση του παιχνιδιού. Η μέθοδος αυτή είναι πολύ χρήσιμη όταν θέλουμε να αρχικοποιήσουμε μεταβλητές ή να ορίσουμε Components.

- **Update()**

```
// Update is called once per frame
0 references
void Update()
{

}
```

Εικόνα 20 - Unity - Script Update function

Η Update() όπως προδίδει και το ονομά της είναι η μέθοδος η οποία «τρέχει» συνέχεια και πραγματοποιεί όλες τις ανανεώσεις του παιχνιδιού. Καλείται 60 φορές το δευτερόλεπτο ή 60 frames το δευτερόλεπτο). Με λίγα λόγια είναι αυτή που πραγματοποιεί όλες τις κύριες ενέργειες του κώδικα. Για παράδειγμα, εκεί γράφεται ο κώδικας για την ανανέωση κάποιου σκορ, την κίνηση του χαρακτήρα, την αυτόματη κίνηση ανα second ενός αντιπάλου, την επίθεση κτλπ.

Φυσικά υπάρχουν πολλές ακόμα προκαθορισμένες μέθοδοι από την MonoBehaviour class, τις οποίες μπορεί κάποιος να βρει στο Documentation του Unity.

Αναλυτική Περιγραφή Παιχνιδιού

Σενάριο

Το παιχνίδι που θα περιγράψουμε στη συνέχεια είναι ένα διασκεδαστικό-εκπαιδευτικό Role Playing game το οποίο εξελίσσεται μέσα στον λαβύρινθο ενός κάστρου. Ο ιππότης μας (κύριος χαρακτήρας του παιχνιδιού) προσπαθεί να βρει την έξοδο από τον λαβύρινθο. Στη προσπάθειά του αυτή, θα πρέπει να χρησιμοποιήσει τις γνώσεις του ώστε να γίνει καλύτερος με τις ανταμοιβές που θα πάρει από τους μάγους του κάστρου διότι θα βρεθεί αντιμέτωπος με εχθρικά τέρατα που θέλουν να τον εμποδίσουν. Κάθε δευτερόλεπτο είναι πολύτιμο, καθώς ο χρόνος θα δείξει ποιος είναι πιο άξιος ιππότης.

Οντότητες

Ακολουθούν οι οντότητες που εμφανίζονται στο παιχνίδι βάση του ρόλου τους.

Κύριος Χαρακτήρας του παιχνιδιού

Εικόνα 21 - Οντότητες - Κύριος Χαρακτήρας

Μάγος του Κάστρου

Εικόνα 22 - Οντότητες - Μάγος του Κάστρου

Βασιλιάς του Κάστρου

Εικόνα 23 - Οντότητες - Βασιλιάς του Κάστρου

Τέρατα

Level 1

Εικόνα 24 - Οντότητες - Τέρας Επιπέδου 1

Level 2

Εικόνα 25 - Οντότητες - Τέρας Επιπέδου 2

Level 3

Εικόνα 26 - Οντότητες - Τέρας Επιπέδου 3

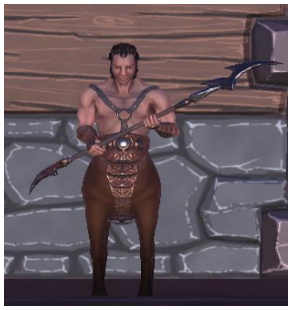
Level 4

Εικόνα 27 - Οντότητες - Τέρας Επιπέδου 4

Level 5

Εικόνα 28 - Οντότητες - Τέρας Επιπέδου 5

Level 6



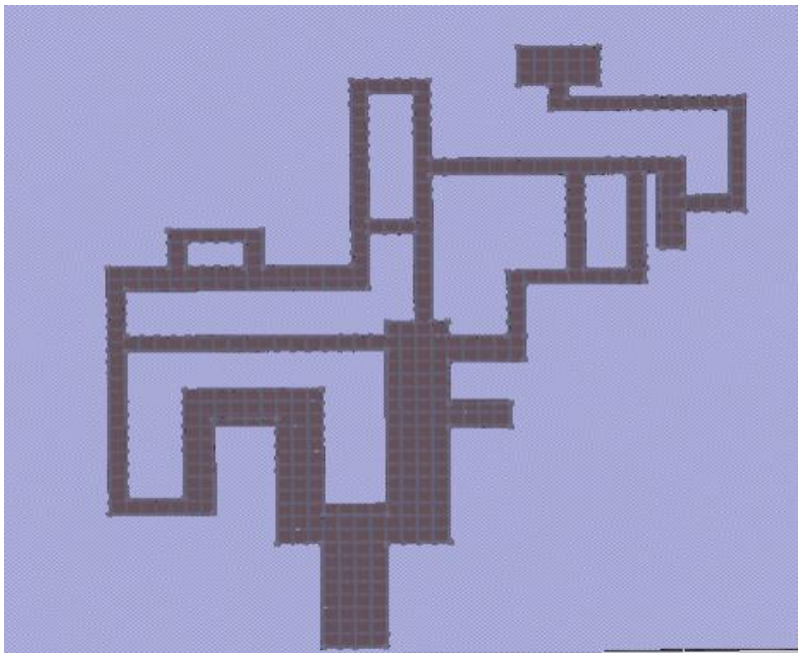
Εικόνα 29 - Οντότητες - Τέρας Επιπέδου 6

Κόσμος

Όπως αναφέραμε και νωρίτερα το παιχνίδι εξελίσσεται μέσα σε ένα λαβύρινθο, όπου ο κύριος χαρακτήρας μας προσπαθεί να βρει την έξοδο.

Στη συνέχεια εμφανίζεται από ψηλά ο λαβύρινθος-χάρτης του παιχνιδιού.

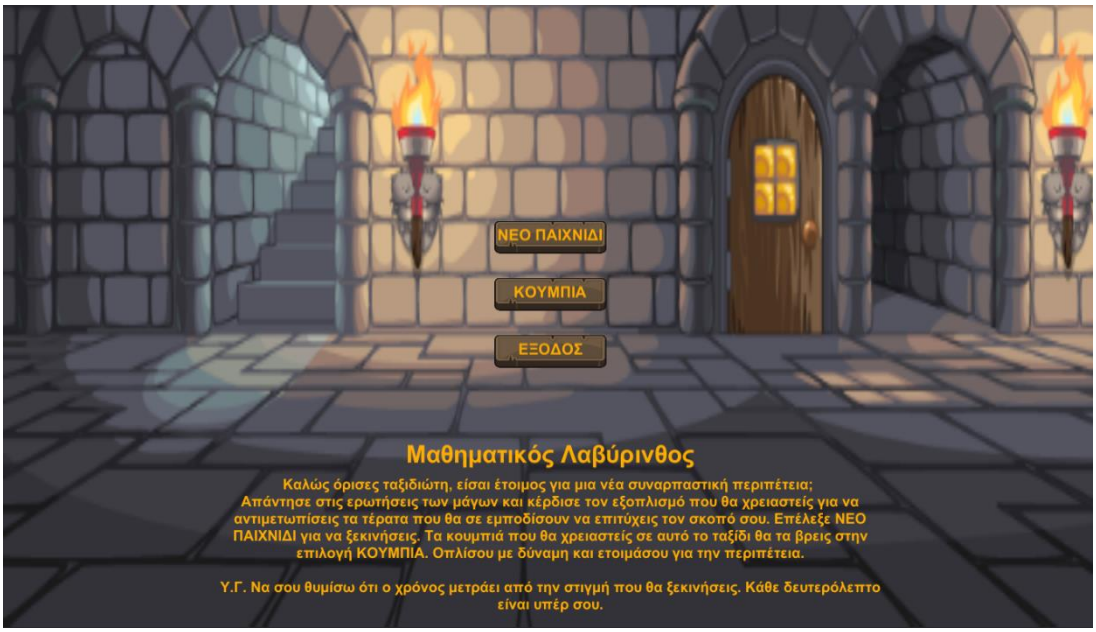
Λαβύρινθος - Χάρτης



Εικόνα 30 - Λαβύρινθος - Χάρτης

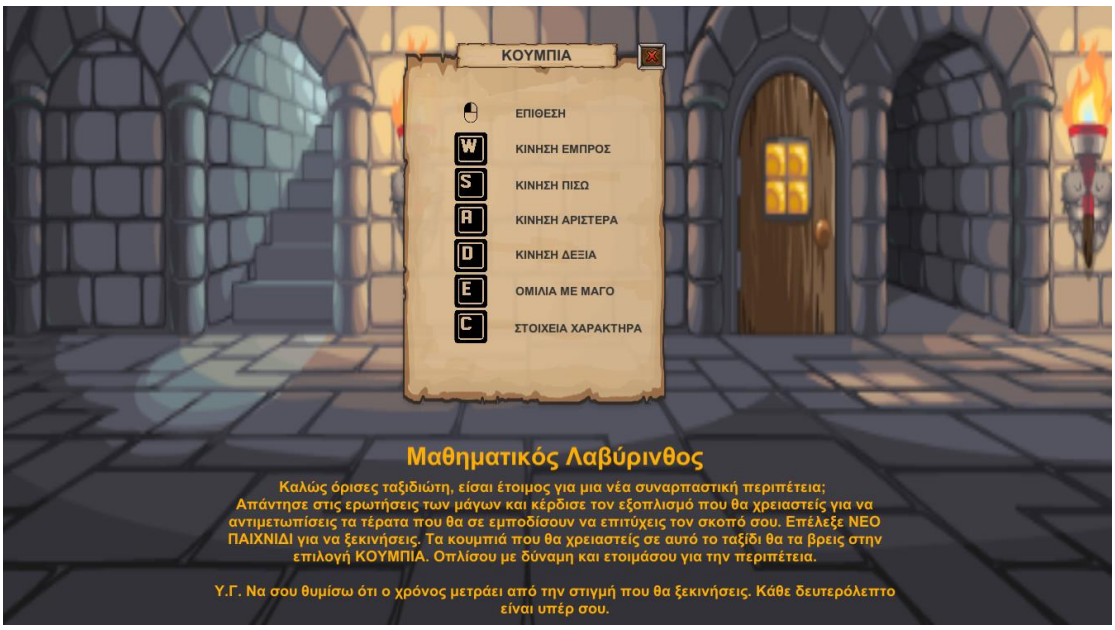
Ροή του Παιχνιδιού

1. Ξεκινώντας το παιχνίδι, ο χρήστης βλέπει το αρχικό menu, όπως εμφανίζεται στη συνέχεια.



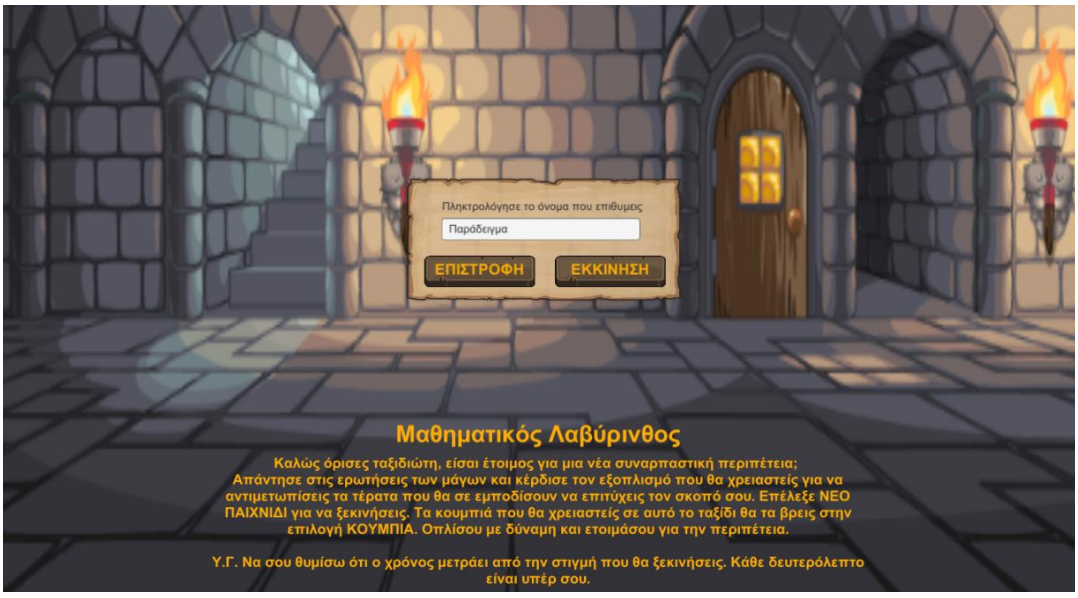
Εικόνα 31 - Ροή Παιχνιδιού - Κεντρικό Menu

α) Επιλογή «**ΚΟΥΜΠΙΑ**»: Με το πάτημα αυτού του κουμπιού ο χρήστης μπορεί να δει τα κουμπιά που θα του χρειαστούν κατά τη διάρκεια του παιχνιδιού.



Εικόνα 32 - Κεντρικό Menu - Κουμπιά

β) Επιλογή «**ΝΕΟ ΠΑΙΧΝΙΔΙ**» : Το συγκεκριμένο κουμπί δίνει την ευκαιρία στον χρήστη να επιλέξει το όνομα που θα έχει ο χαρακτήρας, και στη συνέχεια να ξεκινήσει το παιχνίδι.



Εικόνα 33 - Κεντρικό Menu - Νέο Παιχνίδι

2) Με την επιλογή του ονόματος και πατώντας το κουμπί «**ΕΚΚΙΝΗΣΗ**» ξεκινάει το παιχνίδι.

Ο κύριος χαρακτήρας μας ξεκινάει από την αρχή όντας Level 1 με τη ζωή του να είναι στο 100% και με το βασικό εξοπλισμό του.

Πάνω δεξιά στην οθόνη υπάρχει χρονόμετρο, το οποίο ξεκινάει με το που ανοίξει η οθόνη του παιχνιδιού.

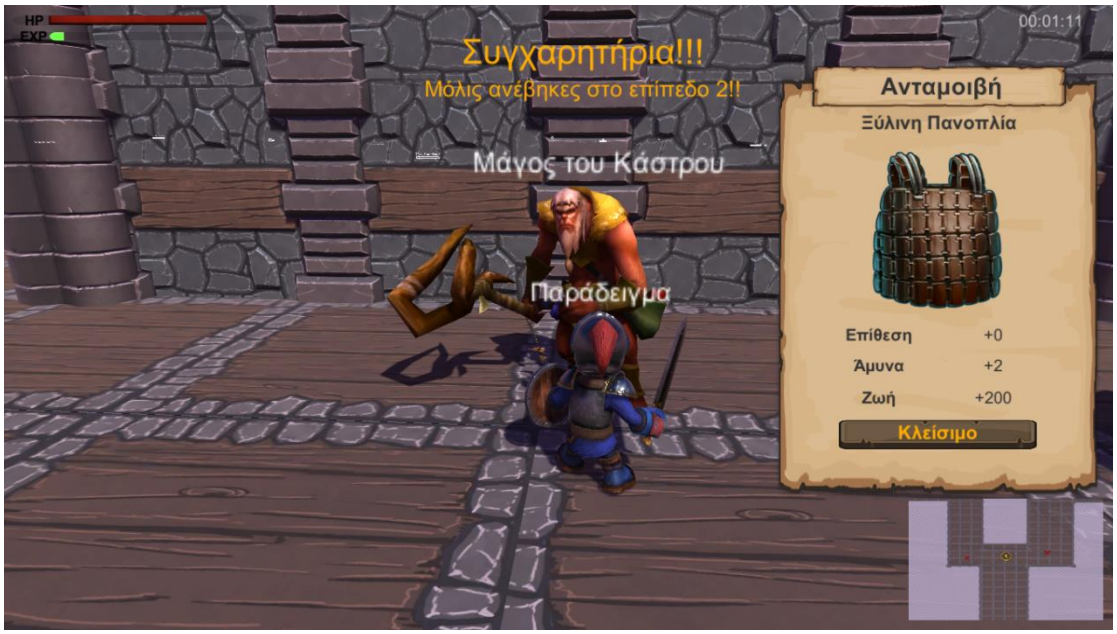
Ενώ κάτω δεξιά υπάρχει ο χάρτης στον οποίο εμφανίζονται με σύμβολα τα τέρατα και οι μάγοι μέσα στον λαβύρινθο.

Τέλος όπως αναφέραμε παραπάνω ο χρήστης πατώντας το κουμπί «C» μπορεί να δει τα χαρακτηριστικά του.

Εικόνα 34 - Παιχνίδι - Χαρακτηριστικά

3) Όπως είδαμε και στο υποκεφάλαιο των οντοτήτων, υπάρχει ο μάγος του κάστρου, ο οποίος δίνει στον κύριο χαρακτήρα καλύτερο εξοπλισμό και εμπειρία (experience) εφόσον αυτός απαντήσει σωστά στην ερώτηση Μαθηματικών που θα του δώσει.

Μέσα στον λαβύρινθο υπάρχουν 10 Μάγοι, κάθε μάγος έχει διαφορετική ερώτηση και διαφορετική ανταμοιβή. Όσο το παιχνίδι προχωράει τόσο το επίπεδο των ερωτήσεων δυσκολεύει.



Εικόνα 35 - Παιχνίδι - Ανταμοιβή

4) Βασικό ρόλο παίζουν και τα τέρατα μέσα στο παιχνίδι. Αυτά είναι που στην ουσία σου δείχνουν το δρόμο μέχρι το τέλος. Ξεκινώντας το παιχνίδι, όπως είδαμε και παραπάνω, ο κύριος χαρακτήρας βλέπει τα τέρατα Level 1 τα οποία πρέπει να αντιμετωπίσει.

Εικόνα 36 - Παιχνίδι - Μάχη

Ακολουθώντας την άνοδο των Level, ο χρήστης θα καταλήξει στα τέρατα Level 6 τα οποία βρίσκονται στο τελικό στάδιο του παιχνιδιού. Ουσιαστικά τα τέρατα είναι η πυξίδα του χρήστη.

5) Στο τελευταίο στάδιο του παιχνιδιού, όταν δηλαδή ο κύριος χαρακτήρας φτάσει στο τέλος του λαβυρίνθου, θα βρει τον βασιλιά του κάστρου ο οποίος θα παγώσει τον χρόνο, θα του δείξει τους καλύτερους παίκτες του παιχνιδιού βάσει χρόνου και το παιχνίδι θα ολοκληρωθεί.

Εικόνα 37 - Παιχνίδι - Τέλος

Εικόνα 38 - Παιχνίδι - Στατιστικά

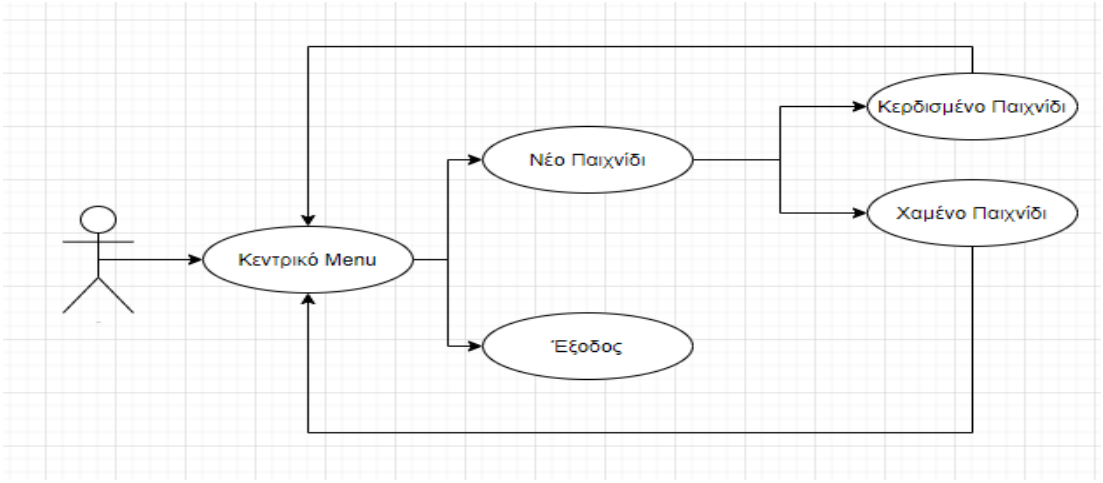
Σχεδίαση και ανάλυση του παιχνιδιού

Πρώτο βήμα για την υλοποίηση ενός παιχνιδιού είναι η ανάλυση των απαιτήσεων και ο σχεδιασμός αυτού, με σκοπό να υπάρχει μία πυξίδα κατά τη διάρκεια της υλοποίησης.

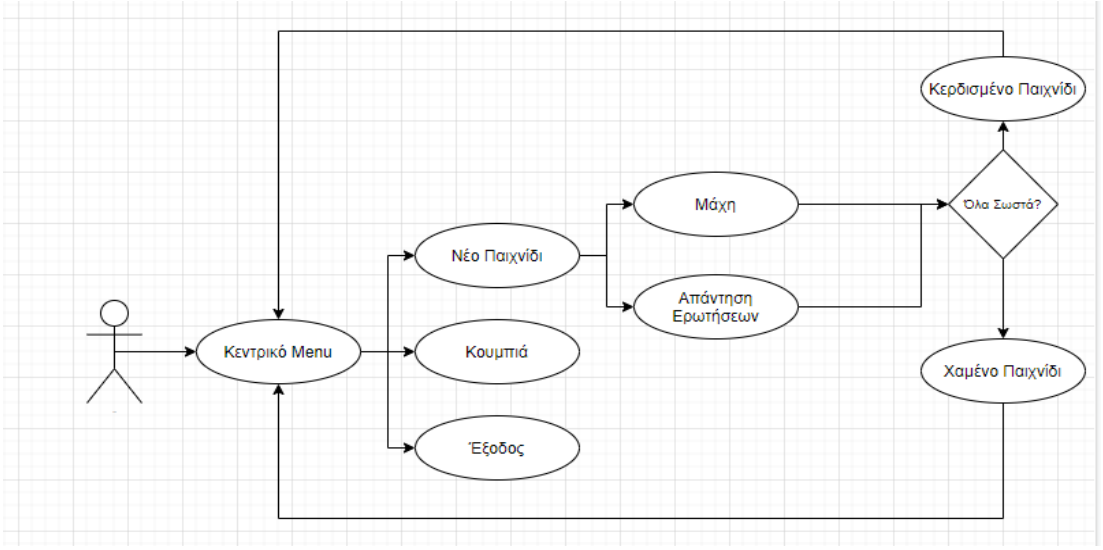
Αλληλεπίδραση χρήστη παιχνιδιού

Αρχικά, σημαντικό ρόλο στην κατεύθυνση στην οποία θα κινηθούμε για την υλοποίηση του παιχνιδιού παίζει η επαφή που έχει ο χρήστης με αυτό, ξεκινώντας από τις βασικές ενέργειες που θα μπορεί να κάνει μέσα στο παιχνίδι και συνεχίζοντας αναλύοντας αυτές.

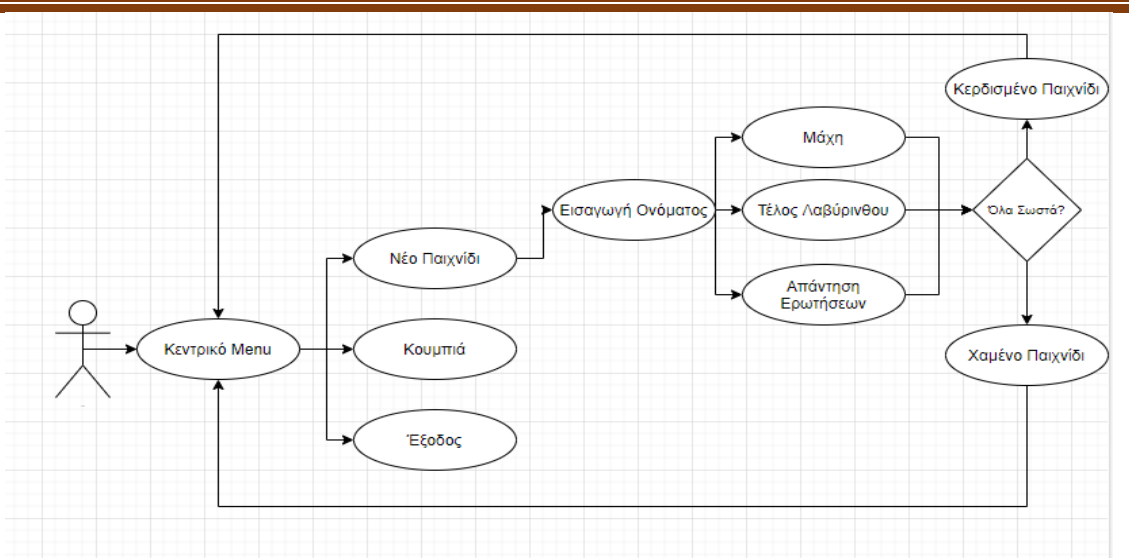
Παρακάτω εμφανίζεται σε μορφή διαγράμματος η παραπάνω αλληλεπίδραση που περιγράψαμε.



Εικόνα 39 - Σχεδίαση και Ανάλυση – Διάγραμμα Χρήστη 1



Εικόνα 40 - Σχεδίαση και Ανάλυση – Διάγραμμα Χρήστη 2



Εικόνα 41 - Σχεδίαση και Ανάλυση – Διάγραμμα Χρήστη 3

Οντότητες, Αντικείμενα και Λειτουργίες

Εφόσον έχουμε ολοκληρώσει τη γενική σχεδίαση του παιχνιδιού, δηλαδή την φιλοσοφία στην οποία θα βασιστούμε, σειρά παίρνει η σχεδίαση των λεπτομερειών. Σημαντικό ρόλο στο παιχνίδι (όπως είναι λογικό εφόσον σχεδιάζουμε παιχνίδι τύπου RPG) θα έχουν οι οντότητες και ο τρόπος λειτουργίας τους.

Στη συνέχεια βασιζόμενοι στη λειτουργία των οντοτήτων ακολουθεί η σωστή τοποθέτηση και η περιγραφή των αντικειμένων και των λειτουργιών που θα χρησιμοποιηθούν για την καλύτερη εμπειρία του χρήστη μέσα στο παιχνίδι.

Οντότητες

Εκτός από τον κύριο χαρακτήρα του παιχνιδιού, οι οντότητες που θα συναντήσουμε μέσα στο παιχνίδι – όπως αναφέραμε και πιο πάνω – χωρίζονται σε δύο κατηγορίες όπως τις συναντάμε στα περισσότερα Role Playing Games,

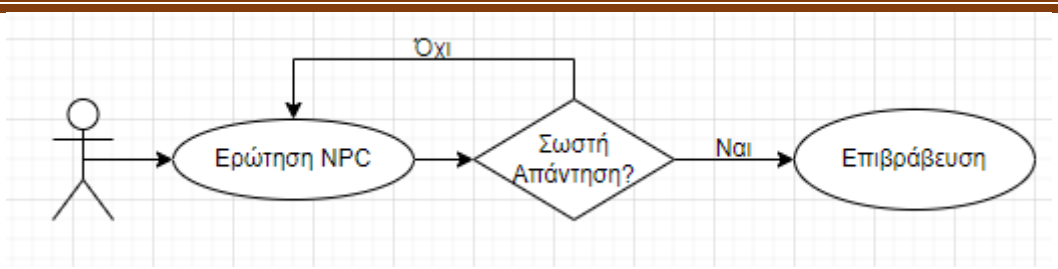
- NPC (Non-player character) ή αλλιώς βοηθητικοί χαρακτήρες
- Mobs, όρος ο οποίος χρησιμοποιείται για να περιγράψει τους εχθρούς του κύριου χαρακτήρα μέσα στο παιχνίδι

Ας ξεκινήσουμε από τη πιο σημαντική κατηγορία κατά τη ροή του παιχνιδιού η οποία είναι τα NPCs. Η σωστή λειτουργία του NPC κατά κύριο λόγο βασίζεται στον τρόπο με τον οποίο θα βοηθάνε τον χρήστη να πραγματοποιήσει τον σκοπό του μέσα στο παιχνίδι.

Στην περίπτωση του δικού μας παιχνιδιού, ένα NPC θα δίνει ένα πρόβλημα μαθηματικών 6ης δημοτικού στον χρήστη, που όταν ο χρήστης δώσει τη σωστή απάντηση θα παίρνει ως ανταμοιβή,

1. **Experience** το οποίο τον βοηθάει να ανέβει επίπεδο και να αντιμετωπίσει τέρατα μεγαλύτερου επιπέδου και
2. **Weapon/Armor** τα οποία τον κάνουν πιο δυνατό απέναντι στα mobs.

Ακολουθεί ένα διάγραμμα ώστε να γίνει πιο κατανοητή η λειτουργία των NPC.



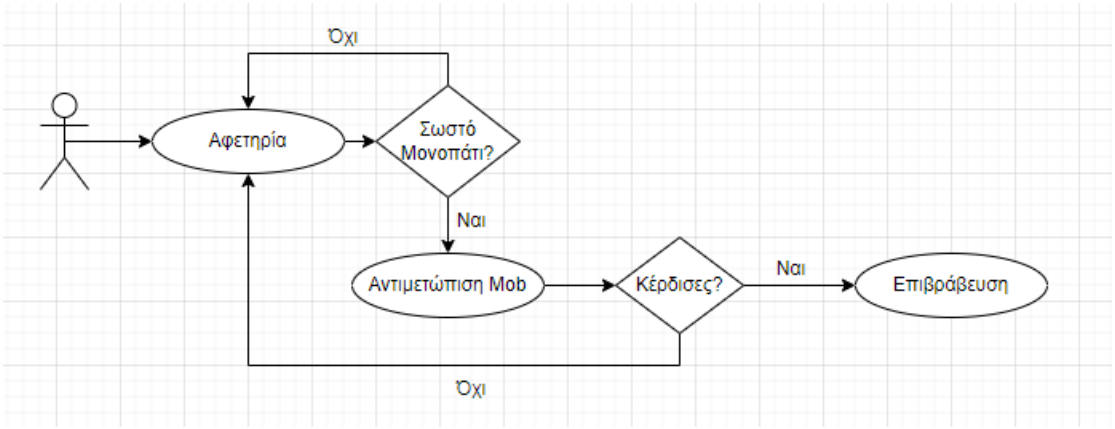
Εικόνα 42 - Σχεδίαση και Ανάλυση – Διάγραμμα NPC

Η δεύτερη κατηγορία όπως αναφέρθηκε παραπάνω είναι τα mobs. Σε γενικό πλαίσιο τα mobs αποτελούν τους «εχθρούς» του κύριου χαρακτήρα που προσπαθούν να τον δυσκολέψουν να επιτύχει το σκοπό του, που δεν είναι άλλος από το να κερδίσει το παιχνίδι.

Στη δική μας περίπτωση, τα mobs εκτός από τον ρόλο του «εχθρού» παίζουν και τον ρόλο του καθοδηγητή για τον παίκτη. Ακολουθώντας δηλαδή το επίπεδο των mobs, ο κύριος χαρακτήρας οδηγείται προς το τέλος του λαβυρίνθου, κάτι το οποίο όμως είναι και παγίδα, αφού αν βρεθεί αντιμέτωπος με μεγαλύτερο επιπέδου τέρατα δεν θα καταφέρεις να τα νικήσει και θα πρέπει να ξεκινήσει από την αρχή.

Σκοπός αυτής της ιδιαιτερότητας των mobs είναι να μάθει ο παίκτης να ακολουθεί το σωστό μονοπάτι προς τη νίκη και όχι το σύντομο, που θα τον φέρει αντιμέτωπο με αρκετές δυσκολίες που δεν θα μπορεί να αντιμετωπίσει.

Οι λειτουργίες των mob περιγράφονται διαγραμματικά παρακάτω.



Εικόνα 43 - Σχεδίαση και Ανάλυση – Διάγραμμα Τέρατος

Αντικείμενα και Λειτουργίες

Με τους όρους αντικείμενα και λειτουργίες θα περιγράψουμε οτιδήποτε έχει να κάνει με τις λεπτομέρειες του παιχνιδιού που το κάνουν ξεχωριστό. Χαρακτηριστικό παράδειγμα είναι οι επιβραβεύσεις του παίκτη κατόπιν ολοκλήρωσης μιας δοκιμασίας.

Ας ξεκινήσουμε λοιπόν από το βασικό σκέλος του παιχνιδιού, που είναι ο τόπος στον οποίοι διαδραματίζεται το παιχνίδι. Ως τόπος του παιχνιδιού επιλέχθηκε ένας λαβύρινθος ώστε το παιχνίδι να έχει αρχή, μέση και τέλος ώστε να είναι πιο κατανοητός ο σκοπός του παιχνιδιού και να είναι πιο εύκολη τυχόν επέκτασή του. Για παράδειγμα, μια πιθανή επέκταση θα μπορούσε να είναι μία πίστα με έναν λαβύρινθο για κάθε τάξη του δημοτικού, έτσι θα μπορούσε να φτιαχτεί ένα ολοκληρωμένο παιχνίδι για όλες τις τάξεις με παραπάνω βαθμό δυσκολίας.

Επειδή όμως ένας λαβύρινθος ποτέ δεν είναι εύκολος, πόσο μάλλον για παιδιά δημοτικού, προστέθηκαν κάποιες ιδιαιτερότητες σε αυτόν με σκοπό την πιο εύκολη επίλυσή του. Όπως αναφέρθηκε και παραπάνω, μια βασική πυξίδα για την ολοκλήρωση του λαβύρινθου είναι τα τέρατα που ανάλογα με το επίπεδό τους, παίζουν το ρόλο του καθοδηγητή για τον παίκτη.

Σε αυτή την λειτουργία θα έρθει να προστεθεί ο χάρτης. Ως χάρτη του παιχνιδιού, θα ορίσουμε ένα υπόμνημα στο κάτω μέρος της οθόνης ώστε ο παίκτης να ξέρει που βρίσκεται και να μπορεί να αναγνωρίσει τα NPCs και τους αντιπάλους του καθώς και να γνωρίζει τη θέση τους.

Εφόσον ολοκληρώσαμε το βασικό σκέλος της σχεδίασης που είναι ο τόπος του παιχνιδιού, ήρθε η ώρα να ασχοληθούμε με τα επιπλέον στοιχεία του που θα δώσουν «ζωή» στο παιχνίδι. Βασικές ερωτήσεις που πρέπει να απαντηθούν είναι

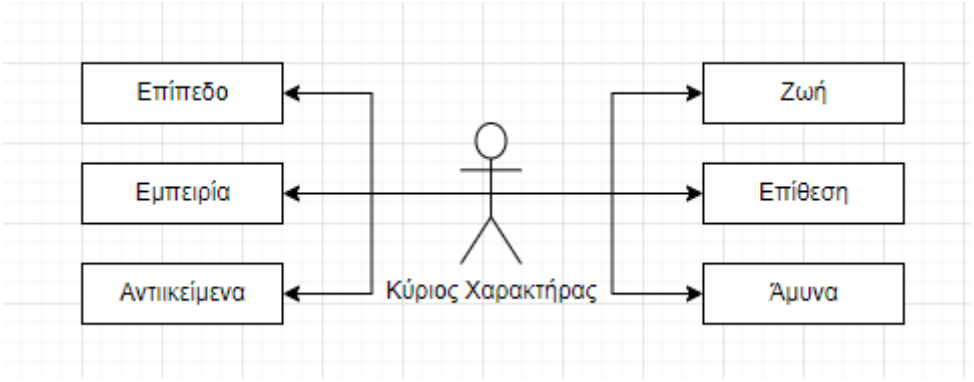
- πως θα παίρνει ανεβαίνει επίπεδο ο παίκτης
- τι αντικείμενα θα έχει ώστε να γίνεται πιο δυνατός
- τι ιδιότητες θα έχουν οι χαρακτήρες του παιχνιδιού
- πως θα μπορεί να αντιμετωπίσει ο παίκτης τα τέρατα
- πως θα ολοκληρώνεται το παιχνίδι

Για να απαντηθούν τα παραπάνω ερωτήματα θα πρέπει να σχεδιαστούν και να καταγραφούν τα χαρακτηριστικά του καθενός χαρακτήρα μέσα στο παιχνίδι.

Για την ανάλυση και την καταγραφή των στοιχείων αυτών θα χρησιμοποιηθούν διαγράμματα με τα οποία θα πετύχουμε την σαφέστερη κατανόηση τους.

Ας ξεκινήσουμε λοιπόν από τον κύριο χαρακτήρα του παιχνιδιού.

Ξεκινώντας το παιχνίδι ο χαρακτήρας θα πρέπει να έχει κάποια αρχικά χαρακτηριστικά τα οποία θα μεταβάλλονται με την πάροδο του παιχνιδιού.



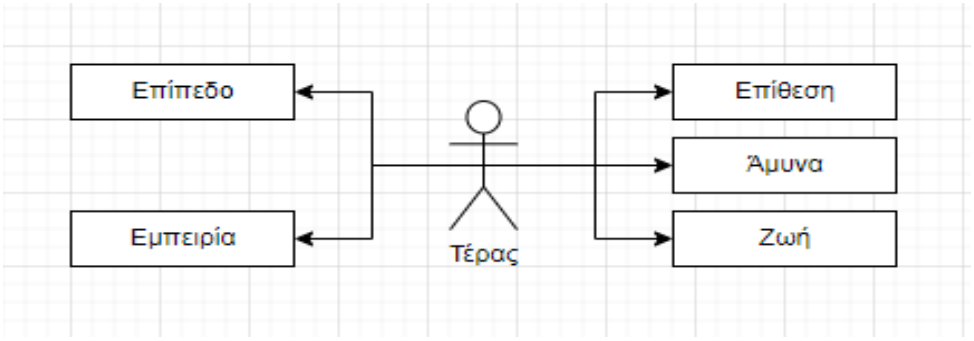
Εικόνα 44 - Σχεδίαση και Ανάλυση – Διάγραμμα Χαρακτηριστικών Παίκτη

Όπως φαίνεται στο διάγραμμα ο κύριος χαρακτήρας μας θα έχει 6 στοιχεία που θα τον χαρακτηρίζουν μέσα στο παιχνίδι. Ακολουθεί αναλυτικότερη περιγραφή αυτών των στοιχείων.

1. Επίπεδο : Το στοιχείο αυτό αναφέρεται στο επίπεδο του παίκτη. Ξεκινάει με την τιμή 1 και αυξάνεται όσο ο παίκτης προχωράει μέσα στο παιχνίδι και απαντάει ερωτήσεις.
2. Εμπειρία : Για να ανέβει επίπεδο ο παίκτης θα πρέπει να έχει την ανάλογη εμπειρία. Τέρατα και NPCs θα τον βοηθάνε να αποκτάει περισσότερο ώστε να ανεβαίνει και επίπεδο.
3. Αντικείμενα : Τα αντικείμενα του παίκτη είναι αυτά που θα του δίνουν έξτρα βοήθεια στην αντιμετώπιση των τεράτων. Κατά την εκκίνηση θα έχει τα αρχικά και όσο απαντάει σωστά τις ερωτήσεις θα παίρνει καλύτερα που θα του δίνουν παραπάνω ζωή, επίθεση και άμυνα.
4. Ζωή : Αποτελεί το χαρακτηριστικό που τον κρατάει ζωντανό μέσα στο παιχνίδι, όταν αδειάσει η ζωή του από τις επιθέσεις των τεράτων χάνει και ξεκινάει από την αρχή.
5. Επίθεση : Ως επίθεση ορίζουμε τη ζημιά (damage) που θα κάνει στα τέρατα. Επηρεάζεται από το επίπεδο και τα αντικείμενα που θα κατέχει κάθε στιγμή μέσα στο παιχνίδι.
6. Άμυνα : Με τον όρο άμυνα αναφερόμαστε στην αντοχή που θα έχει ο παίκτης όταν βρίσκεται αντιμέτωπος με τα τέρατα. Αντίστοιχα με την επίθεση, μεταβάλλεται ανάλογα με τα αντικείμενα και το επίπεδο.

Εφόσον ολοκληρώσαμε το σχεδιασμό του κύριου χαρακτήρα, σειρά έχουν τα τέρατα και τα NPCs. Τα χαρακτηριστικά τους μοιάζουν με αυτά του παίκτη αλλά ο τρόπος λειτουργίας τους είναι διαφορετικός.

Ας ξεκινήσουμε λοιπόν από τα τέρατα.



Εικόνα 45 - Σχεδίαση και Ανάλυση – Διάγραμμα Χαρακτηριστικών Τέρατος

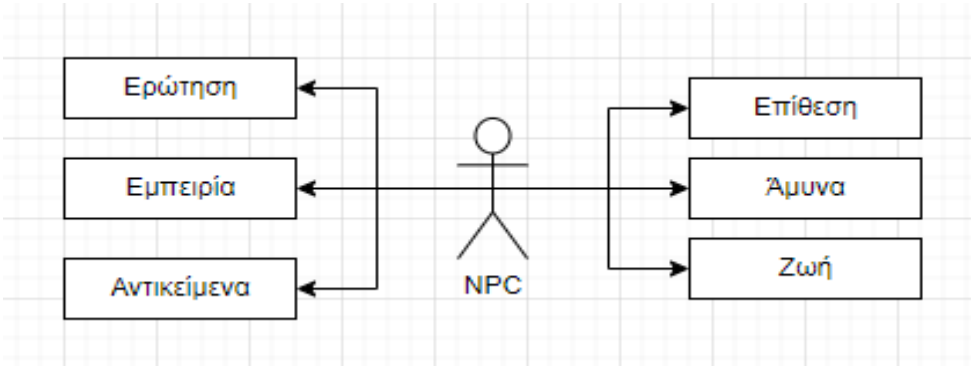
Όπως γίνεται αντιληπτό από το παραπάνω διάγραμμα τα στοιχεία των τεράτων είναι τα ίδια με αυτά του παίκτη εκτός μόνο από τα αντικείμενα.

Ας δούμε όμως τις διαφορές που προκύπτουν μεταξύ των χαρακτηριστικών των τεράτων και του παίκτη.

1. Επίπεδο : Το επίπεδο των τεράτων είναι σταθερό, δεν μεταβάλλεται μέσα στο παιχνίδι.
2. Εμπειρία : Ως εμπειρία ενός τέρατος ορίζεται το ποσό της εμπειρίας που θα δοθεί στον παίκτη όταν το τέρας σκοτωθεί.
3. Επίθεση : Η επίθεση του τέρατος είναι σταθερή ανάλογα με το επίπεδο του (το οποίο όπως αναφέραμε είναι και αυτό σταθερό).
4. Άμυνα : Η άμυνα του τέρατος είναι σταθερή ανάλογα με το επίπεδο του (το οποίο όπως αναφέραμε είναι και αυτό σταθερό).
5. Ζωή : Η ζωή του τέρατος είναι σταθερή ανάλογα με το επίπεδο του (το οποίο όπως αναφέραμε είναι και αυτό σταθερό).

Αρκετές διαφορές στο τρόπο λειτουργίας των χαρακτηριστικών θα δούμε στα NPC, που ο ρόλος τους μέσα στο παιχνίδι είναι καθαρά βοηθητικός.

Ακολουθεί η διαγραμματική σχεδίαση των στοιχείων ενός NPC καθώς και η λεπτομερής επεξήγηση αυτών και των διαφορών που έχουν με τα στοιχεία των άλλων χαρακτήρων.



Εικόνα 46 - Σχεδίαση και Ανάλυση – Διάγραμμα Χαρακτηριστικών NPC

Εκτός από το στοιχείο της ερώτησης, όλα τα υπόλοιπα μπορούν να καταταγούν σε μια πιο γενική κατηγορία που θα την ονομάσουμε ανταμοιβές.

Ας τα πάρουμε λοιπόν με τη σειρά,

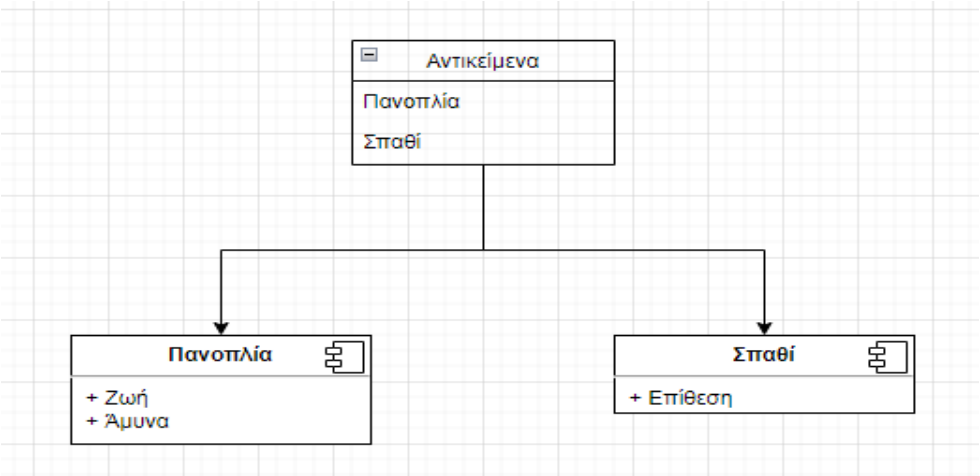
1. Ερώτηση : Αφορά αποκλειστικά το NPC και επηρεάζει και τα υπόλοιπα στοιχεία που το χαρακτηρίζουν. Ως ερώτηση λοιπόν ορίζεται το πρόβλημα το οποίο θα πρέπει να λύσει ο παίκτης για να πάρει τις ανταμοιβές του.
2. Εμπειρία : Απαντώντας σωστά στην ερώτηση ο παίκτης λαμβάνει αρκετή εμπειρία που θα τον ανεβάσει επίπεδο. Το ποσό της εμπειρίας που θα λάβει από μία ερώτηση είναι αρκετά μεγαλύτερο από αυτό που θα λάβει σκοτώνοντας ένα τέρας, γεγονός που κάνει τις ερωτήσεις πιο σημαντικές από τα τέρατα.
3. Αντικείμενα : Τα αντικείμενα που θα κερδίσει ο παίκτης από το NPC είναι και αυτά που θα τον κάνουν πιο δυνατό απέναντι σε μεγαλύτερου επιπέδου τέρατα. Ο μοναδικός τρόπος για να πάρει αυτά τα αντικείμενα ο παίκτης είναι να απαντήσει σωστά στην ερώτηση του NPC.

Τελευταίο αλλά εξίσου σημαντικό με τα υπόλοιπα στο σχεδιασμό του παιχνιδιού είναι τα αντικείμενα και τα χαρακτηριστικά τους. Όπως αναφέραμε πιο πάνω, κερδίζοντας αντικείμενα ένας παίκτης θα γίνεται πιο δυνατός απέναντι στα τέρατα. Τι εννοούμε όμως με αυτό;

Τα αντικείμενα που θα κερδίζει ο παίκτης χωρίζονται σε 2 κατηγορίες,

- Πανοπλία : Η πανοπλία θα δίνει στον παίκτη ένα επιπλέον πόσο ζωής και άμυνας ώστε να μην δέχεται πολύ ζημιά (damage) από τα τέρατα μεγαλύτερου επιπέδου.
- Σπαθί : Αντίστοιχα το σπαθί, θα δυναμώνει το στοιχείο της επίθεσης, ώστε να κάνει παραπάνω ζημιά (damage) στα τέρατα μεγαλύτερου επιπέδου που αντιμετωπίζει.

Ακολουθεί και η διαγραμματική αναπαράσταση των αντικειμένων.



Εικόνα 47 - Σχεδίαση και Ανάλυση – Διάγραμμα Αντικειμένων

Εφόσον ολοκληρώθηκε η ανάλυση και ο σχεδιασμός του παιχνιδιού, υπάρχει η συνταγή ώστε να ξεκινήσουμε να υλοποιούμε το παιχνίδι.

Στο επόμενο κεφάλαιο θα γίνει αναλυτική περιγραφή στην υλοποίηση του παιχνιδιού, και θα δούμε στην πράξη πως λειτουργούν όλα αυτά τα οποία σχεδιάσαμε.

Υλοποίηση του Παιχνιδιού

Υλοποίηση Βάσης Δεδομένων

Συνοψίζοντας τον σχεδιασμό που κάναμε, παρατηρήθηκε το γεγονός πως υπάρχει αρκετή πληροφορία μέσα στο παιχνίδι η οποία θα πρέπει να είναι αποθηκευμένη και υπάρχει εύκολη πρόσβαση σε αυτή ανά πάσα στιγμή. Για να το πετύχουμε αυτό, επιλέξαμε να χρησιμοποιήσουμε μια βάση δεδομένων SQLite ώστε να υπάρχει όλη η πληροφορία καταγεγραμμένη και να υπάρχει εύκολη δυνατότητα επέκτασης στο μέλλον.

Στη ΒΔ επιλέξαμε να κρατάμε δεδομένα τα οποία σχετίζονται με τα NPCs, τα Mobs και τα τελικά στατιστικά του κάθε παίκτη.

Για την τελική υλοποίηση της βάσης χρειάστηκε να γίνει καταγραφή της πληροφορίας που θα χρειαστούμε και σωστός σχεδιασμός όπως θα δούμε παρακάτω.

Ένα βασικό ερώτημα που πρέπει να απαντηθεί για τον σωστό σχεδιασμό της βάσης είναι τι πληροφορία θα κρατάμε σε αυτήν. Τα δεδομένα της βάσης θα πρέπει να είναι σταθερά και σωστά δομημένα ώστε να είναι πιο εύκολη η άντλησή τους.

Ας ξεκινήσουμε λοιπόν την καταγραφή, συνοψίζοντας τα δεδομένα που θα χρειαστούμε ώστε να καταλήξουμε στο σωστό σχήμα της ΒΔ.

Βασική πληροφορία την οποία θα πρέπει να έχουμε αποθηκευμένη είναι τα το όνομα του παίκτη καθώς και το χρόνο στον οποίο ολοκλήρωσε την αποστολή του. Για να το πετύχουμε αυτό χρειαζόμαστε έναν πίνακα στον οποίο θα τα αποθηκεύουμε και ένα View στο οποίο θα τα εμφανίζουμε, ώστε να μην γίνεται το query μέσα στο runtime του παιχνιδιού, αλλά στη βάση.

Όπως εμφανίζεται στη συνέχεια βλέπουμε ότι ο πίνακας έχει 3 στήλες, ένα ID το οποίο ξεχωρίζει τον κάθε παίκτη, το όνομα (name) που έβαλε στην αρχή του παιχνιδιού καθώς και το χρόνο (time) που χρειάστηκε για να ολοκληρώσει το παιχνίδι.

Πίνακας Stats

Name	Type	NN	PK	AI	U	Default
id	INTEGER	☐	☑	☑	☐	
name	TEXT	☐	☐	☐	☐	
time	TEXT	☐	☐	☐	☐	

Εικόνα 48 - ΒΔ - Πίνακας Stats

Ακολουθεί το Query για την δημιουργία του συγκεκριμένου πίνακα.

```
CREATE TABLE "Stats" (
    "id"    INTEGER PRIMARY KEY AUTOINCREMENT,
    "name"  TEXT,
    "time"  TEXT
);
```

Εικόνα 49 - ΒΔ - Script Πίνακα Stats

Για την άντληση της πληροφορίας αυτής, όπως τέθηκε παραπάνω σχεδιάστηκε ένα View το οποίο θα φέρνει τα στοιχεία του παραπάνω πίνακα ταξινομημένα κατά αύξουσα σειρά βάση του χρόνου (time) ώστε να εμφανίζεται πρώτο το όνομα του παίκτη που έκανα το λιγότερο χρόνο

```
CREATE VIEW "DisplayStats" AS
  select name,time
  from Stats
  where time is not null
  order by time;
```

Εικόνα 50 - ΒΔ - Script View Display Stats

Μια σημαντική λεπτομέρεια είναι ότι ο συγκεκριμένος πίνακας (**Stats**) είναι ο μοναδικό στον οποίο γίνεται Insert μέσα από το παιχνίδι.

Όπως αναφέρθηκε στο σχεδιασμό του παιχνιδιού και πιο συγκεκριμένα της οντότητας NPC, υπάρχουν ερωτήσεις και ανταμοιβές οι οποίες θα πρέπει να καταγραφούν, ώστε να είναι εύκολη η πρόσβαση σε αυτές και να μπορούμε να προσθέσουμε ανά πάσα στιγμή νέα δεδομένα.

Για αυτό επιλέξαμε να σχεδιάσουμε 2 πίνακες, οι οποίοι θα συνδέονται μεταξύ τους χρησιμοποιώντας ένα foreign key. Ας δούμε λοιπόν αυτούς τους πίνακες.

Πίνακας Quests

Εδώ θα καταγραφεί όλη η πληροφορία για τις ερωτήσεις που θα υπάρχουν μέσα στο παιχνίδι.

Name	Type	NN	PK	AI	U	Default
id	INTEGER	☐	☑	☑	☑	
name	TEXT	☐	☐	☐	☐	
quest	TEXT	☐	☐	☐	☐	
reward_id	INTEGER	☐	☐	☐	☐	
answer	TEXT	☐	☐	☐	☐	
exp	INTEGER	☐	☐	☐	☐	
active	INTEGER	☐	☐	☐	☐	

Εικόνα 51 - ΒΔ - Πίνακας Quests

```
CREATE TABLE "Quests" (
    "id"    INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
    "name"  TEXT,
    "quest" TEXT,
    "reward_id" INTEGER,
    "answer"    TEXT,
    "exp"   INTEGER,
    "active"    INTEGER
);
```

Εικόνα 52 - ΒΔ - Script Πίνακα Quests

Κάθε ερώτηση (Quest) λοιπόν αποτελείται από τα εξής στοιχεία,

- Id : Ο μοναδικός αριθμός κάθε ερώτησης
- Name : Το όνομα της ερώτησης
- Quest : Η ερώτηση που θα γίνεται προς το παίκτη.
- Reward_id : Οι ανταμοιβές που θα παίρνει από κάθε ερώτηση ο παίκτης και αποτελεί το foreign key για την σύνδεση με τον πίνακα Rewards.
- Answer : Η απάντηση της ερώτησης
- Exp : Η εμπειρία που θα παίρνει από κάθε ερώτηση
- Active : Η στήλη αυτή υπάρχει ώστε να γνωρίζουμε ποιες ερωτήσεις είναι ενεργές (Active=1) και ποιες έχουν απαντηθεί μέσα στο παιχνίδι (Active=0). Κάθε φορά που ξεκινάει το παιχνίδι γίνονται αυτόματα update όλες οι ερωτήσεις σε Active=1.

Πίνακας Rewards

Name	Type	NN	PK	AI	U	Default
id	INTEGER	☐	☑	☑	☑	
label	TEXT	☐	☐	☐	☐	
name	TEXT	☐	☐	☐	☐	
def	INTEGER	☐	☐	☐	☐	
atk	INTEGER	☐	☐	☐	☐	
health	INTEGER	☐	☐	☐	☐	

Εικόνα 53 - ΒΔ - Πίνακας Rewards

```
CREATE TABLE "Rewards" (
	"id"	INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
	"label"	TEXT,
	"name"	TEXT,
	"def"	INTEGER,
	"atk"	INTEGER,
	"health"	INTEGER
);
```

Εικόνα 54 - ΒΔ - Script Πίνακα Rewards

Όπως φαίνεται παραπάνω, μια ανταμοιβή (Reward) αποτελείται από,

- Id : Μοναδικός κωδικός κάθε ανταμοιβής και σύνδεση με τον πίνακα quest.
- Label : Η γενική ταμπέλα μιας ανταμοιβής
- Name : Το όνομα της ανταμοιβής που εμφανίζεται μέσα στο παιχνίδι
- Def : Το ποσό της άμυνας που δίνει κάθε ανταμοιβή στον παίκτη
- Atk : Το ποσό της επίθεσης που δίνει κάθε ανταμοιβή στον παίκτη
- Health : Το ποσό της ζωής που δίνει κάθε ανταμοιβή στον παίκτη

Οι πίνακες **Quests** και **Rewards** αποτελούν τα στοιχεία της οντότητας NPC.

Για την σωστή διαχείρισή και εμφάνιση τους σχεδιάστηκε ένα View με όνομα NPC το οποίο θα συγχωνεύει αυτούς τους 2 πίνακες και θα εμφανίζει τα στοιχεία που χρειαζόμαστε μέσα στο παιχνίδι.

```
CREATE VIEW NPC as
select
q.id as qid
,q.name as qname
,q.quest
,q.answer as qanswer
,q.exp
,r.label as rLabel
,r.name as rName
,r.atk
,r.def
,r.health
,q.active
from Quests as q
left join Rewards as r on r.id=q.reward_id;
```

Εικόνα 55 - ΒΔ - Script View NPC

Εφόσον έχουμε όλη την πληροφορία που χρειαζόμαστε οργανωμένη για τα NPC και τον κύριο χαρακτήρα, σειρά παίρνουν τα τέρατα.

Για τα τέρατα θα χρειαστούμε έναν πίνακα ώστε να κρατάμε τα χαρακτηριστικά τους και να τα αντλούμε όπου τα χρειαζόμαστε μέσα στο παιχνίδι. Για τον λόγο αυτό σχεδιάστηκε ο πίνακας MobStats όπου όπως θα δούμε παρακάτω περιλαμβάνει όλη τη πληροφορία.

Πίνακας MobStats

Name	Type	NN	PK	AI	U	Default
id	INTEGER	☐	☑	☑	☐	
level	INTEGER	☐	☐	☐	☐	
atk	INTEGER	☐	☐	☐	☐	
def	INTEGER	☐	☐	☐	☐	
health	INTEGER	☐	☐	☐	☐	
exp	INTEGER	☐	☐	☐	☐	

Εικόνα 56 - Πίνακας MobStats

```
CREATE TABLE "MobStats" (
    "id"     INTEGER PRIMARY KEY AUTOINCREMENT,
    "level" INTEGER,
    "atk"    INTEGER,
    "def"    INTEGER,
    "health"     INTEGER,
    "exp"    INTEGER
);
```

Εικόνα 57 - ΒΔ - Script Πίνακα MobStats

Όπως αναφέραμε και κατά το σχεδιασμό των χαρακτηριστικών της οντότητας των τεράτων, ένα mob αποτελείται από,

- Id : Μοναδικός κωδικός κάθε τέρατος
- Level : Το επίπεδο του τέρατος
- Atk : Το πόσο της επίθεσης κάθε τέρατος
- Def : Το πόσο της άμυνας κάθε τέρατος
- Health : Το πόσο της ζωής κάθε τέρατος
- Exp : Το ποσό της εμπειρίας που θα δίνει στον παίκτη όταν το σκοτώσει

Υλοποίηση παιχνιδιού σε Unity

Υλοποίηση περιφερειακών Script

Η Βάση Δεδομένων που θα χρησιμοποιηθεί στο παιχνίδι πλέον έχει ολοκληρωθεί. Επόμενο βήμα στην υλοποίηση του παιχνιδιού είναι αρχικά η εισαγωγή της ΒΔ μέσα στο παιχνίδι ώστε να μπορεί να χρησιμοποιηθεί ανά πάσα στιγμή.

Για να το πετύχουμε αυτό δημιουργήσαμε ένα C# Script το οποίο θα δούμε παρακάτω, καλείται μία φορά στην αρχή του παιχνιδιού και επιτυγχάνει τη σύνδεση με τη βάση.

```
using UnityEngine;
using System.Data;
using Mono.Data.Sqlite;
using System.IO;

Unity Script (2 asset references) | 8 references
public class ConnectToDatabase : MonoBehaviour
{
    private static IDbConnection dbcon;

    10 references
    public static IDbConnection Dbcon { get => dbcon; set => dbcon = value; }

    // Use this for initialization
    Unity Message | 0 references
    void Awake()
    {
        string connection = "URI=file:" + Application.dataPath + "/Database/RPG_Game.db";
        Dbcon = new SqliteConnection(connection);
        Dbcon.Open();
    }

}
```

Εικόνα 58 - Υλοποίηση - Script ConnectToDatabase

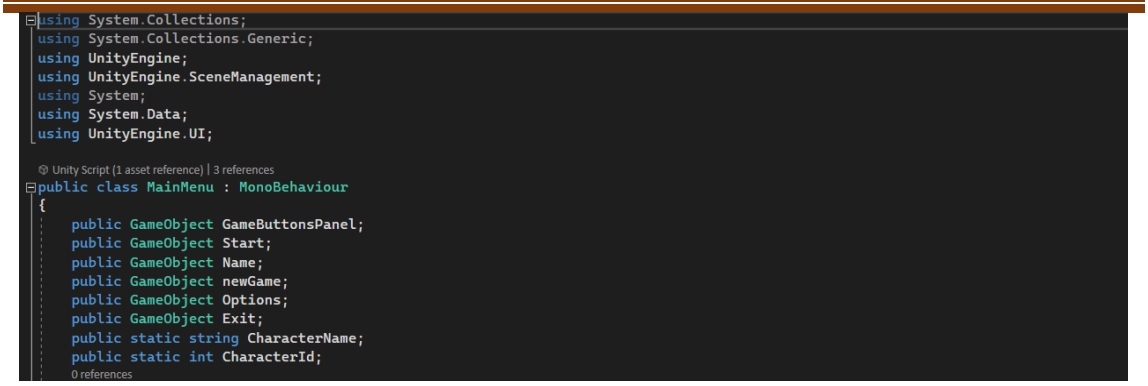
Όπως φαίνεται στο Script, η σύνδεση της βάσης γίνεται μέσα στη function Awake, η οποία καλείται όταν φορτωθεί το πρώτο scene του παιχνιδιού. Έχει δομηθεί έτσι ώστε να καλείται μόνο μια φορά, και όχι κάθε φορά που τη χρειαζόμαστε, και με αυτό επιτυγχάνεται λιγότερος φόρτος στο παιχνίδι κατά το runtime.

Όταν ολοκληρωθεί η σύνδεση με τη βάση, είναι διαθέσιμη για κάθε περίπτωση που θα την χρειαστούμε.

Ξεκινώντας λοιπόν το παιχνίδι, πρώτη εικόνα που βλέπει ο χρήστης είναι η εικόνα του Main Menu, η οποία του δίνει τη δυνατότητα να ξεκινήσει το παιχνίδι, να δει τα κουμπιά που χρειάζεται μέσα στο παιχνίδι καθώς και να επιλέξει την έξοδο από αυτή.

Για να επιτευχθούν οι παραπάνω ενέργειες, δημιουργήθηκε ένα Script που αφορά μόνο τη συγκεκριμένη οθόνη.

Αρχικά ορίστηκαν οι μεταβλητές που θα χρειαστούν, και στις συνέχεια οι functions που θα πραγματοποιούν τις ενέργειες που έχουμε ορίσει.



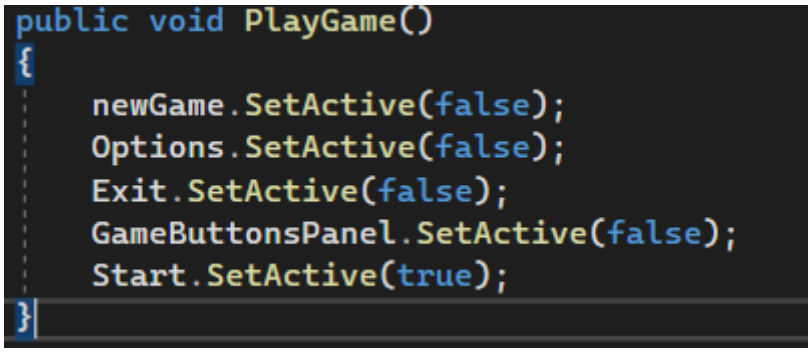
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using System;
using System.Data;
using UnityEngine.UI;

Unity Script (1 asset reference) | 3 references
public class MainMenu : MonoBehaviour
{
    public GameObject GameButtonsPanel;
    public GameObject Start;
    public GameObject Name;
    public GameObject newGame;
    public GameObject Options;
    public GameObject Exit;
    public static string CharacterName;
    public static int CharacterId;
    0 references
```

Εικόνα 59 - Υλοποίηση - Script MainMenu -Μεταβλητές

Όπως φαίνεται χαρακτηριστικά στην παραπάνω εικόνα, έχουν οριστεί τα GameObjects που θα χρειαστούμε για το Main Menu καθώς και 2 μεταβλητές που αφορούν τον χαρακτήρα ώστε να χρησιμοποιηθούν για την έναρξη του παιχνιδιού.

Σειρά έχουν οι functions. Οι functions που έχουν φτιαχτεί στο συγκεκριμένο script, χρησιμοποιούνται από τα κουμπιά της αρχικής οθόνης κατά το OnClick() ώστε να γίνονται δυναμικά οι ενέργειες που απαιτούνται.

Πατώντας δηλαδή το κουμπί «Νέο Παιχνίδι» καλείται η function PlayGame() η οποία όπως φαίνεται παρακάτω θέτει ως active(true) και active(false) τα ανάλογα components για την σωστή λειτουργία του παιχνιδιού.



```
public void PlayGame()
{
    newGame.SetActive(false);
    Options.SetActive(false);
    Exit.SetActive(false);
    GameButtonsPanel.SetActive(false);
    Start.SetActive(true);
}
```

Εικόνα 60 - Υλοποίηση - Script MainMenu - PlayGame()

Για να ξεκινήσει το παιχνίδι όπως περιγράψαμε και σε προηγούμενα κεφάλαια, ο χρήστης πρέπει να δώσει το όνομα του χαρακτήρα και να πατήσει «**Εκκίνηση**».

Για το σκοπό αυτό, δημιουργήθηκε η **function StartGame()** η οποία εκτελείται όταν πατηθεί το κουμπί «Εκκίνηση» και εκτός από την καταγραφή του παίκτη στη βάση δεδομένων, ορίζει όλα τα Quests ως active=1 ώστε να είναι διαθέσιμα για το παιχνίδι και τέλος εκκινεί τη Scene “Game”, η οποία είναι η βασική Scene του παιχνιδιού.

```
public void StartGame()
{
    name = Name.GetComponent<Text>().text;
    using (IDbCommand cmnd_read = ConnectToDatabase.Dbcon.CreateCommand())
    {
        IDataReader reader;
        string query = "insert into Stats (name) values ('" + name + "')";
        cmnd_read.CommandText = query;
        reader = cmnd_read.ExecuteReader();
    }
    using (IDbCommand update = ConnectToDatabase.Dbcon.CreateCommand())
    {

        string sqlQuery1 = "Update Quests set active=1";
        update.CommandText = sqlQuery1;
        update.ExecuteScalar();

    }
    using (IDbCommand cmnd_read = ConnectToDatabase.Dbcon.CreateCommand())
    {
        IDataReader reader;
        string query = "select id,name from Stats order by id desc LIMIT 1;";
        cmnd_read.CommandText = query;
        reader = cmnd_read.ExecuteReader();
        while (reader.Read())
        {
            CharacterId = reader.GetInt32(0);
            CharacterName = reader.GetString(1);
        }
        reader.Close();
    }
    SceneManager.LoadScene("Game");

}
```

Εικόνα 61 - Υλοποίηση - Script MainMenu - StartGame()

Οι άλλες 3 functions του script MainMenu, αφορούν τα κουμπιά «Κουμπιά» και «Έξοδος». Οι ενέργειες που πραγματοποιούν είναι η εμφάνιση των κατάλληλων components για τη σωστή λειτουργία του παιχνιδιού και ενεργοποιούνται κατά το πάτημα των ανάλογων κουμπιών.

```
0 references
public void CloseButtons()
{
    newGame.SetActive(true);
    Options.SetActive(true);
    Exit.SetActive(true);
    GameButtonsPanel.SetActive(false);
    Start.SetActive(false);
}
0 references
public void GameButtons()
{
    newGame.SetActive(false);
    Options.SetActive(false);
    Exit.SetActive(false);
    GameButtonsPanel.SetActive(true);
}
0 references
public void QuitGame()
{
    Application.Quit();
}
```

Εικόνα 62 - Υλοποίηση - Script MainMenu - Υπόλοιπες μέθοδοι

Ολοκληρώνοντας λοιπόν με την πρώτη οθόνη και κύριο μενού του παιχνιδιού, μεταφερόμαστε στο επόμενο Scene το οποίο είναι το παιχνίδι μας. Με το που ξεκινήσει το παιχνίδι, αρχίζει να μετράει ο χρόνος, ο οποίος παγώνει και καταγράφεται στη βάση όταν ο παίχτης μιλήσει και στον τελευταίο NPC.

Για να επιτευχθεί αυτό, υλοποιήθηκε ένα script που αφορά καθαρά τον χρόνο του παιχνιδιού και ονομάζεται Timer. Στο πρώτο κομμάτι αυτού του script έγινε ο ορισμός των μεταβλητών που θα χρειαστούμε καθώς και η αρχικοποίηση του χρόνου.

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

Unity Script (1 asset reference) | 1 reference
public class Timer : MonoBehaviour
{
    public Text timerText;
    private float startTime;
    private bool finished = false;
    public static string finishedTime;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        startTime = Time.time;
    }
```

Εικόνα 63 - Υλοποίηση - Script Timer - Start()

Η συνάρτηση Start() χρησιμοποιείται ώστε να κατά την εκκίνηση να οριστεί το startTime του παιχνιδιού, στο οποίο θα βασιστούμε για να μετράμε τον χρόνο που κάνει.

Στη συνέχεια, με την χρήση της function Update(), υπολογίζεται και καταγράφεται στο Text Component timerText η διάρκεια του παιχνιδιού.

```
void Update()
{
    if (finished)
    {
        return;
    }
    float t = Time.time - startTime;
    string hours = ((int)t / 3600).ToString("00");
    string minutes = ((int) t / 60).ToString("00");
    string seconds = (t % 60).ToString("00");

    //timerText.text = t.ToString();
    timerText.text = hours+ ":"+ minutes + ":" + seconds;
}
```

Εικόνα 64 - Υλοποίηση - Script Timer - Update()

Κατά την κλήση της συνάρτησης Update(), αρχικά ελέγχεται μια Boolean μεταβλητή η οποία έχει οριστεί για τον ορισμό του τέλους του παιχνιδιού. Αν είναι false, όπως έχει οριστεί και στην αρχή, μετράει τον χρόνο που είναι ενεργό το παιχνίδι, τον μετατρέπει σε μορφή ΩΩ:ΛΛ:ΔΔ και τον εμφανίζει στο component timerText που εμφανίζεται πάνω δεξιά στην οθόνη του παίκτη.

Για να σταματήσει ο χρόνος, έχει φτιαχτεί η συνάρτηση Finish() η οποία καλείται όταν ο παίχτης μιλήσει στον τελευταίο NPC.

```
public void Finish()
{
    finished = true;
    finishedTime = timerText.text;
    timerText.color = Color.red;
    using (IDbCommand update = ConnectToDatabase.Dbcon.CreateCommand())
    {
        string sqlQuery1 = "Update Stats set time='" + timerText.text + "' where id=" + MainMenu.CharacterId;
        update.CommandText = sqlQuery1;
        update.ExecuteScalar();
    }

}
```

Εικόνα 65 - Υλοποίηση - Script Timer - Finish()

Η συνάρτηση αυτή μετατρέπει την μεταβλητή finished σε true, και καταγράφει στη ΒΔ το id του παίχτη καθώς και τον χρόνο που έκανε, ώστε στα αποτελέσματα να του εμφανίσει τη θέση στην οποία βρίσκεται.

Πριν ασχοληθούμε με τα κύρια Scripts του παιχνιδιού που αφορούν τον παίκτη, τα mobs και τα NPCs, θα ήταν χρήσιμο να περιγράψουμε και τα υπόλοιπα scripts που αφορούν τα περιφερειακά κομμάτια του παιχνιδιού. Τέτοια scripts είναι το script του Game Menu, του Minimap καθώς και του damage που εμφανίζεται κατά την μάχη με τα τέρατα.

Κατά τη διάρκεια του παιχνιδιού ο παίχτης έχει δυνατότητα να «παγώσει» το χρόνο και το παιχνίδι. Πατώντας το κουμπί ESC, εμφανίζεται το menu του παιχνιδιού με τις επιλογές «Συνέχεια», «Κουμπιά» και «Έξοδος». Το menu αυτό δίνει τη δυνατότητα στον παίχτη να κάνει Pause το παιχνίδι.

Για να το πετύχουμε αυτό υλοποιήσαμε το script GameMenu.

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

Unity Script (1 asset reference) | 0 references
public class GameMenu : MonoBehaviour
{
    public GameObject pauseMenuUI;
    public GameObject GameButtonsPanel;
    public static bool GameisPaused = false;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        pauseMenuUI.SetActive(false);
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameisPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }
```

Εικόνα 66 - Υλοποίηση - Script GameMenu - Μεταβλητές και Start(),Update()

Στο script αυτό όπως φαίνεται και στην παραπάνω εικόνα, αρχικά θέτουμε το game menu ως active=false, και εφόσον ο χρήστης πατήσει το κουμπί ESC, ελέγχει αν το παιχνίδι είναι Paused ή όχι και καλεί την ανάλογη συνάρτηση.

```
public void Resume()
{
    pauseMenuUI.SetActive(false);
    Time.timeScale = 1f;
    GameisPaused = false;
}
1 reference
void Pause()
{
    pauseMenuUI.SetActive(true);
    Time.timeScale = 0f;
    GameisPaused = true;
}
```

Εικόνα 67 - Υλοποίηση - Script GameMenu - Resume(),Pause()

Όπως αναφέραμε και παραπάνω, στο game menu υπάρχουν και τα κουμπιά «Κουμπιά» και «Έξοδος». Κατά το πάτημα του κουμπιού «Κουμπιά» εμφανίζεται ο κατάλογος των επιλογών των κουμπιών του παιχνιδιού ενώ με το κουμπί έξοδος, το παιχνίδι επιστρέφει στην αρχική οθόνη.

```
0 references
public void GameButtons()
{
    GameButtonsPanel.SetActive(true);
}

0 references
public void CloseGameButtons()
{
    GameButtonsPanel.SetActive(false);
}
0 references
public void Quit()
{
    SceneManager.LoadScene("Main Menu");
}
```

Εικόνα 68 - Υλοποίηση - Script GameMenu – Buttons Fuctions

Το επόμενο Script σχετίζεται με τη εμφάνιση του κύριου χαρακτήρα μέσα στον χάρτη και ονομάζεται Minimap.

Όπως φαίνεται και παρακάτω, αρχικά παίρνουμε την νέα θέση του παίχτη κάθε στιγμή του παιχνιδιού και τη μετατρέπουμε με βάση τον άξονα y ώστε να εμφανίζεται κάθετα στον χάρτη.

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

Unity Script (1 asset reference) | 0 references
public class Minimap : MonoBehaviour
{
    public Transform player;
    Unity Message | 0 references
    void LateUpdate()
    {
        Vector3 newPosition = player.position;
        newPosition.y = transform.position.y;
        transform.position = newPosition;

        //transform.rotation = Quaternion.Euler(90f, player.eulerAngles.y, 0f);
    }
}
```

Εικόνα 69 - Υλοποίηση - Script Minimap

Παραπάνω έγινε αναφορά σε ένα ακόμα περιφερειακό script το οποίο παρουσιάζεται παρακάτω και χρησιμοποιείται για την εμφάνιση του damage που κάνει ο παίκτης στα τέρατα και ονομάζεται PopUpDmg.

Η λειτουργία του βασίζεται στην τυχαία εμφάνιση (με τη χρήση της συνάρτησης Random) της θέσης του damage μπροστά στα τέρατα, κάτι το οποίο λειτουργεί σαν εφέ μέσα στο παιχνίδι. Λειτουργικά δεν επηρεάζει το παιχνίδι.

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

Unity Script (1 asset reference) | 0 references
public class PopUpDmg : MonoBehaviour
{
    // Start is called before the first frame update

    public float DestroyTime;
    public Vector3 Offset = new Vector3(0, 2, 0);
    public Vector3 Randomize = new Vector3(0.5f, 0, 0);
    Transform target;
    Unity Message | 0 references
    void Start()
    {
        target = GameObject.FindGameObjectWithTag("Player").transform;
        transform.localPosition += Offset;
        transform.localPosition += new Vector3(Random.Range(-Randomize.x, Randomize.x), Random.Range(-Randomize.y, Randomize.y), Random.Range(-Randomize.z, Randomize.z));
        transform.rotation = target.rotation;
        Destroy(gameObject, DestroyTime);
    }
```

Εικόνα 70 - Υλοποίηση - Script PopUpDmg

Εφόσον ολοκληρώσαμε την περιγραφή κάποιων περιφερειακών script, ήρθε η ώρα να δούμε πιο αναλυτικά τον τρόπο με τον οποίο ο χαρακτήρας δρα μέσα στο παιχνίδι. Τα 3 βασικά scripts του παιχνιδιού, αφορούν τον κύριο χαρακτήρα, τα NPC και τα Mobs, εφόσον υπάρχει αλληλεπίδραση μεταξύ τους, καλύτερο είναι να τα περιγράψουμε μαζί και όχι ως 3 ξεχωριστά.

Ας ξεκινήσουμε το script Player, όπου σε αυτό υπάρχουν όλες οι λειτουργίες του παίκτη. Αρχικά, όπως και σε κάθε script υπάρχει η δήλωση των μεταβλητών που θα χρησιμοποιηθούν οι οποίες είτε είναι public, ώστε να μπορούμε να τις περάσουμε μέσα από το περιβάλλον του Unity (κυρίως για τα γραφικά) είτε private και αφορούν καθαρά το script.

```
Unity Script (1 asset reference) | 9 references
public class Player : MonoBehaviour
{
    //Variables
    float speed=4;
    float rot =0f;
    float rotSpeed = 80;
    float gravity = 8;


    Vector3 moveDir = new Vector3(0, 0, 0);
    CharacterController controller;
    public Animator anim;
    public Image HPBar;
    public Image ExpBar;
    float experience;

    public GameObject HP;
    public GameObject MP;

    private List<Transform> enemiesInRange = new List<Transform>();
    public List<string> map = new List<string>();
    public bool canAttack;
    public float attackDamage;
    public float def;

    public int health;
    public int exp;
    public float currentExp;
    public float currentHealth;
    float damageTaken;
    public int level;
```

Εικόνα 71 - Υλοποίηση - Script Player - Μεταβλητές

```
public int level;

public GameObject LevelUp;
public Text CongratsMessage;
public Text LevelMessage;

public GameObject CharacterName;
int id;
string name;

static int rewardAtk;
static int rewardDef;
static int rewardHealth;

public Sprite EquipWeapon;
public Sprite EquipArmor;

public GameObject DefeatPanel;
```

Εικόνα 72 - Υλοποίηση - Script Player - Μεταβλητές

Επόμενο βήμα είναι η αρχικοποίηση των μεταβλητών που επιτυγχάνεται κατά την εκκίνηση του παιχνιδιού, είτε στην function Start() είτε στη function Awake(). Επίσης θα χρησιμοποιήσουμε την function Update() η οποία ανανεώνεται σε κάθε frame του παιχνιδιού για τις ενέργειες του παίχτη μέσα στο παιχνίδι.

```
void Start()
{
    controller = GetComponent<CharacterController>();
    anim = GetComponent<Animator>();
    currentHealth = health;
    LevelUp.SetActive(false);
    name = MainMenu.CharacterName;
    id = MainMenu.CharacterId;
    CharacterName.GetComponent<TextMesh>().text = name;
    DefeatPanel.SetActive(false);
}
Unity Message | 0 references
void Update()
{
    Movement();
    GetInput();
}

Unity Message | 0 references
private void Awake()
{
    HP.SetActive(true);
    MP.SetActive(true);

}
```

Εικόνα 73 - Υλοποίηση - Script Player - Start(),Update(),Awake()

Η διαφορά μεταξύ της function Start() και Awake() είναι ότι η Start() καλείται στην αρχή του παιχνιδιού ενώ η Awake() όταν φορτωθεί το script το οποίο γίνεται όταν ξεκινήσει το Scene του παιχνιδιού.

Υλοποίηση της κίνησης του χαρακτήρα

Όπως φαίνεται και στην εικόνα στη function Update() καλούνται άλλες δύο συναρτήσεις, η Movement() και η GetInput() τις οποίες θα περιγράψουμε στη συνέχεια.

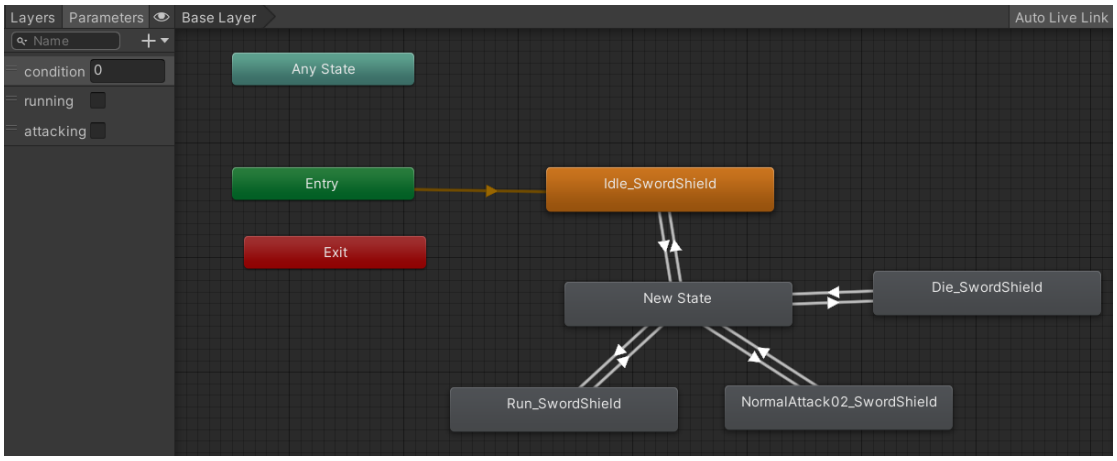
```
void Movement()
{
    if (controller.isGrounded)
    {
        if (Input.GetKey(KeyCode.W))
        {
            if (anim.GetBool("attacking") == true)
            {
                return;
            }
            else if (anim.GetBool("attacking") == false)
            {
                anim.SetBool("running", true);
                anim.SetInteger("condition", 1);
                moveDir = new Vector3(0, 0, 1);
                moveDir *= speed;
                moveDir = transform.TransformDirection(moveDir);
            }

        }
        if (Input.GetKeyUp(KeyCode.W))
        {
            anim.SetBool("running", false);
            anim.SetInteger("condition", 0);
            moveDir = new Vector3(0, 0, 0);

        }
```

Εικόνα 74 - Υλοποίηση - Script Player - Movement() 1

```
        if (Input.GetKey(KeyCode.S))
        {
            if (anim.GetBool("attacking") == true)
            {
                return;
            }
            else if (anim.GetBool("attacking") == false)
            {
                anim.SetBool("running", true);
                anim.SetInteger("condition", 1);
                moveDir = new Vector3(0, 0, -1);
                moveDir *= speed;
                moveDir = transform.TransformDirection(moveDir);
            }

        }
        if (Input.GetKeyUp(KeyCode.S))
        {
            anim.SetBool("running", false);
            anim.SetInteger("condition", 0);
            moveDir = new Vector3(0, 0, 0);

        }
    }
    rot += Input.GetAxis("Horizontal") * rotSpeed * Time.deltaTime;
    transform.eulerAngles = new Vector3(0, rot, 0);
    moveDir.y -= gravity * Time.deltaTime;
    controller.Move(moveDir * Time.deltaTime);
```

Εικόνα 75 - Υλοποίηση - Script Player - Movement() 2

Για την κίνηση του παίχτη όπως βλέπουμε χρησιμοποιούμε δύο μεταβλητές Boolean (running, attacking) και 1 μεταβλητή integer (condition) οι οποίες εξυπηρετούν στην σωστή λειτουργία του Animator του χαρακτήρα που εμφανίζεται παρακάτω.



Εικόνα 76 - Υλοποίηση - Animator Player

Το Animator χρησιμοποιείται για την μίξη και την σωστή λειτουργία των animations. Ας πάρουμε για παράδειγμα την κίνηση του παίκτη, όταν ο χρήστης πατήσει το W, μέσα από τον κώδικα το παιχνίδι ξέρει ότι ο παίκτης θα κινηθεί στον άξονα Z προς τα μπροστά, όμως για να γνωρίζει με ποιο animation θα κινηθεί θέτουμε τη μεταβλητή condition=1 ώστε να χρησιμοποιήσει το

animation Run_SwordShield όπως το έχουμε μέσα από τα βελάκια που εμφανίζονται στην εικόνα. Αντίστοιχα με condition=2 το παιχνίδι θα φορτώσει το animation της επίθεσης NormalAttack02_SwordShield.

Η επόμενη συνάρτηση που βρίσκεται μέσα στην function Update() είναι η GetInput(). Η συγκεκριμένη συνάρτηση είναι η πρώτη στην αλυσίδα συναρτήσεων που καλύπτουν τη μάχη με τα τέρατα όπως θα δούμε στη συνέχεια.

Υλοποίηση της μάχης

Όπως αναφέρθηκε παραπάνω η πρώτη συνάρτηση για την υλοποίηση της μάχης του παίκτη με τα τέρατα είναι η GetInput() η οποία βρίσκεται στο script Player().

```
void GetInput()
{
    if (controller.isGrounded)
    {
        if (Input.GetMouseButtonDown(0)&&!EventSystem.current.IsPointerOverGameObject())
        {
            attacking();
        }

    }
}
```

Εικόνα 77 - Υλοποίηση - Script Player - GetInput()

Η συνάρτηση αυτή χρησιμοποιείται ώστε να αναγνωρίζει το παιχνίδι πότε ο χρήστης πατάει το αριστερό κουμπί στο ποντίκι και να υλοποιεί τις αντίστοιχες ενέργειες, στη περίπτωση μας να καλεί την function attacking().

```
void attacking()
{
    anim.SetInteger("condition", 2);
    StartCoroutine(attackRoutine());
    StartCoroutine(attackCooldown());
}
```

Εικόνα 78 - Υλοποίηση - Script Player - Attacking()

Η συγκεκριμένη συνάρτηση αποτελεί την έναρξη της μάχης, αρχικά θέτει τη μεταβλητή condition=2 στον animator όπως είδαμε παραπάνω, και στη συνέχεια καλεί δύο νέες επαναλαμβανόμενες συναρτήσεις, τις attackRoutine και attackCooldown() τις οποίες θα αναλύσουμε παρακάτω.

```
IEnumerator attackRoutine()
{

    anim.SetBool("attacking", true);
    yield return new WaitForSeconds(0.1f);
    anim.SetInteger("condition", 0);
    GetEnemiesInRange();
    foreach (Transform enemy in enemiesInRange)
    {
        EnemyController ec = enemy.GetComponent<EnemyController>();
        if (ec == null) continue;
        ec.GetHit(attackDamage);
    }
    yield return new WaitForSeconds(0.65f);
    anim.SetBool("attacking", false);
}
```

Εικόνα 79 - Υλοποίηση - Script Player - AttackRoutine()

Η attackRoutine() που εμφανίζεται παραπάνω είναι ο κινητήριος μοχλός στην υλοποίηση της μάχης. Αρχικά κάνει true την μεταβλητή attacking του animator, και στη συνέχεια ψάχνει να βρει αντιπάλους στο Range της επίθεσης που έχουμε ορίσει (μέσω της function GetEnemiesInRange()) και στη συνέχεια πραγματοποιεί την επίθεση. Η συνάρτηση GetHit() βρίσκεται στο script του αντιπάλου (EnemyController) και σηματοδοτεί τη ζημιά που δέχεται ο αντίπαλος από τον παίκτη. Τελειώνοντας η μεταβλητή attacking γυρνάει πάλι σε false. Το yield statement που εμφανίζεται στην παραπάνω συνάρτηση χρησιμοποιείται μέσα στις επαναλαμβανόμενες μεθόδους ώστε να γίνονται paused σε όποιο σημείο και για όσο χρόνο επιθυμούμε.

Η επόμενη μέθοδος είναι η attackCooldown() η οποία χρησιμοποιείται ώστε να υπάρχει χρόνος «φόρτωσης» για κάθε επίθεση που κάνει ο παίκτης.

```
IEnumerator attackCooldown()
{
    canAttack = false;
    yield return new WaitForSeconds(1 / 2);
    canAttack = true;
}
```

Εικόνα 80 - Υλοποίηση - Script Player - AttackCooldown()

Οι επόμενες μέθοδοι στην αλυσίδα της μάχης είναι όπως είδαμε και παραπάνω η GetEnemiesInRange() και η GetHit() που βρίσκεται στο script EnemyController που θα περιγράψουμε στη συνέχεια.

```
void GetEnemiesInRange()
{
    enemiesInRange.Clear();
    foreach (Collider c in Physics.OverlapSphere((transform.position + transform.forward * 0.5f), 0.5f)) {
        if (c.gameObject.CompareTag("Enemy"))
        {
            enemiesInRange.Add(c.transform);
            EnemyController ec = c.transform.GetComponent<EnemyController>();
            //ec.GetHit(attackDamage);
        }
    }
}
```

Εικόνα 81 - Υλοποίηση - Script Player - GetEnemiesInRange()

Αρχικά, κάθε φορά που καλείται η συγκεκριμένη μέθοδος καθαρίζουμε τη λίστα με τους αντιπάλους που βρίσκονται μέσα στον χώρο που μπορεί να επιτεθεί ο παίκτης. Στη συνέχεια ψάχνουμε τους αντιπάλους που βρίσκονται μέσα σε αυτό τον χώρο και γεμίζουμε τη λίστα ώστε να ξέρει ο χαρακτήρας μας σε ποιον θα επιτεθεί.

Με αφορμή τώρα τη συνάρτηση GetHit() που αναφέραμε παραπάνω, είναι ευκαιρία το πάρουμε από την αρχή και να περιγράψουμε τη συμπεριφορά των τεράτων μέσα στο παιχνίδι και στη συνέχεια να δούμε και τη συγκεκριμένη μέθοδο.

Όπως και στα άλλα scripts έτσι και στο EnemyController, αρχικά ορίζουμε τις μεταβλητές και στη συνέχεια γίνεται η αρχικοποίηση τους μέσα στη function Start().

```
public class EnemyController : MonoBehaviour
{
    public Animator anim;
    public Image HPBar;
    public GameObject DmgPopUp;
    public bool dead;
    public int level;
    float health;
    float currentHealth;
    float exp;
    float def;
    float atkDmg;
    float damageTaken;

    float lookRadius = 10f;
    Transform target;
    Player p;

    public GameObject levelText;
```

Εικόνα 82 - Υλοποίηση - Script EnemyController - Μεταβλητές

```
void Start()
{
    levelText.GetComponent<TextMesh>().text = "Τέρας Επιπέδου " + level;
    IDbConnection connection = ConnectToDatabase.Dbcon;
    using (IDbCommand cmnd_read = connection.CreateCommand())
    {
        IDataReader reader;
        string query = "SELECT atk,def,health,exp from MobStats where level=" + level;
        cmnd_read.CommandText = query;
        reader = cmnd_read.ExecuteReader();
        while (reader.Read())
        {
            atkDmg = reader.GetInt32(0);
            def = reader.GetInt32(1);
            health = reader.GetInt32(2);
            exp = reader.GetInt32(3);
        }
        //reader.Close();
    }

    currentHealth = health;
    target =GameObject.FindGameObjectWithTag("Player").transform;
    p = target.GetComponent<Player>();

}
```

Εικόνα 83 - Υλοποίηση - Script EnemyController - Start()

Όπως φαίνεται παραπάνω, αρκεί η μεταβλητή level να περαστεί μέσα στο script από το Unity ώστε το κάθε τέρας να έχει τις ανάλογες ικανότητες. Για να το πετύχουμε αυτό χρησιμοποιούμε ένα query στη ΒΔ μας, ώστε να ανακτήσουμε όλες τις απαραίτητες πληροφορίες που χρειαζόμαστε για το κάθε τέρας. Αυτό μας βοηθάει να ομαδοποιήσουμε τα τέρατα μέσα στο παιχνίδι ανά επίπεδο και να μπορούμε να προσθέσουμε ή να επεξεργαστούμε τα δεδομένα των τεράτων χωρίς να αλλάξουμε τίποτα μέσα στον κώδικά μας.

Εκτός από τα δεδομένα που χρειάζεται το κάθε τέρας για τη σωστή λειτουργία του, υπάρχει και η συμπεριφορά του μέσα στο παιχνίδι. Η συμπεριφορά των τεράτων είναι η ίδια για όλα και η αρχή της γίνεται από την μέθοδο Update().

```
void Update()
{
    if (currentHealth <= 0)
    {
        Die();
        return;
    }
    else
    {
        FollowAndAttack();
    }
}
```

Εικόνα 84 - Υλοποίηση - Script EnemyController - Update()

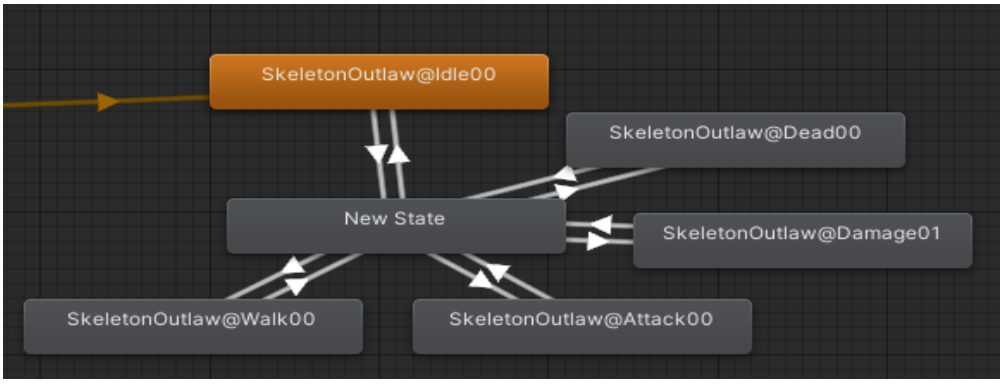
Σε κάθε ανανέωση των frames ελέγχεται αν ένα τέρας έχει ζωή ίση ή μικρότερη του μηδενός ώστε να πραγματοποιηθούν οι ανάλογες ενέργειες.

Αν η ζωή του τέρατος είναι μικρότερη ή ίση του μηδενός (δηλαδή αν έχει πεθάνει) τότε καλείται η μέθοδος Die().

```
void Die()
{
    dead = true;
    anim.SetInteger("condition", 4);
    GameObject.Destroy(this.gameObject,5);
}
```

Εικόνα 85 - Υλοποίηση - Script EnemyController - Die()

Εδώ όπως βλέπουμε θέτουμε τη μεταβλητή dead=true ώστε να ξέρουμε ότι ένα τέρας είναι νεκρό. Στη συνέχεια βάζουμε την τιμή 4 στη μεταβλητή condition του animator του τέρατος που εμφανίζεται παρακάτω και στο τέλος μετά από 5 δευτερόλεπτα αφότου πέθανε, το τέρας εξαφανίζεται από το παιχνίδι.



Εικόνα 86 - Υλοποίηση - Animator EnemyController

Το παράδειγμα του παραπάνω animator αναφέρεται στα τέρατα level 1. Οι τιμές της μεταβλητής condition είναι ίδιες για όλα τα τέρατα,

- Walk : condition=1,
- Attack : condition=2,
- Damage : condition=3,
- Dead : condition=4,

η διαφορά είναι στα animations του κάθε level τέρατος.

Ας επιστρέψουμε τώρα στην μέθοδο Update() και στη περίπτωση που ένα τέρας έχει ζωή μεγαλύτερη του μηδενός. Τότε καλείται η μέθοδος FollowAndAttack().

Η συγκεκριμένη μέθοδος αποτελεί τον πυρήνα της συμπεριφοράς ενός τέρατος, στην αρχή ορίζονται η απόσταση μεταξύ του τέρατος και του χαρακτήρα και η κατεύθυνση που θα πάρει το τέρας όταν βρεθεί ο παίκτης μέσα στα όρια του χώρου που έχουμε θέσει. Αν βρεθεί ο παίκτης στο συγκεκριμένο χώρο, τότε το condition του animator γίνεται 1 (Walk) και το τέρας πηγαίνει προς τον παίκτη. Όταν βρεθεί κοντά στον παίκτη, και ο παίκτης είναι ζωντανός τότε το condition γίνεται 2 (Attack) και ξεκινάει την επίθεση, αλλιώς το condition γίνεται 0 και το τέρας μπαίνει σε κατάσταση idle.

```
void FollowAndAttack()
{

    float distance = Vector3.Distance(target.position, transform.position);
    Vector3 direction = target.position - transform.position;
    float angle = Vector3.Angle(direction, this.transform.forward);
    if (distance <= lookRadius && angle < 45)
    {

        this.transform.rotation = Quaternion.Slerp(this.transform.rotation, Quaternion.LookRotation(direction), Time.deltaTime * 5f);

        if (distance > 3)
        {
            anim.SetInteger("condition", 1);
            this.transform.Translate(0, 0, 0.025f);
        }
        else
        {
            if (target.GetComponent<Player>().anim.GetInteger("condition") == 3)
            {
                anim.SetInteger("condition", 0);
            }
            else
            {
                anim.SetInteger("condition", 2);

            }
        }
    }
    else
    {
        anim.SetInteger("condition", 0);

    }

}
```

Εικόνα 87 - Υλοποίηση - Script EnemyController - FollowAndAttack()

Όταν το τέρας ξεκινάει την επίθεση (condition=2) τότε καλείται η μέθοδος attack() η οποία για καλύτερη αναπαράσταση των γραφικών καλείται μέσα από το animation της επίθεσης του τέρατος και όχι μέσα από τον κώδικα.

```
public void attack() //Animation event
{
    p.GetHit(atkDmg);
}
```

Εικόνα 88 - Υλοποίηση - Script EnemyController - Attack()

Όπως βλέπουμε και εδώ καλείται η μέθοδος GetHit() όμως μέσα από το script του Player. Ευκαιρία να δούμε και να συγκρίνουμε τις δύο μεθόδους GetHit() των διαφορετικών script. Το κοινό τους σημείο είναι ότι και οι δύο χρησιμοποιούνται για την ζημιά που λαμβάνει η κάθε οντότητα, είτε είναι τέρας είτε ο χαρακτήρας μας.

Ας δούμε λοιπόν τη συνάρτηση GetHit() από το script του παίχτη (Player), η οποία καλείται από τη παραπάνω μέθοδο attack().

```
public void GetHit(float damage)
{
    if (currentHealth <= 0)
    {
        Die();
        return;
    }
    else {
        damageTaken = damage - def;
        currentHealth -= damageTaken;
        HPBar.fillAmount -= damageTaken / health;
    }
}
```

Εικόνα 89 - Υλοποίηση - Script Player - GetHit()

Η μέθοδος αυτή δέχεται ως παράμετρο το damage που κάνει τέρας στον παίκτη. Ελέγχεται αν ο παίκτης είναι ζωντανός, αν ναι τότε κάνοντας κάποιους υπολογισμούς αφαιρείται η τελική ζημιά που δέχεται ο παίκτης από τη ζωή του και αυτό αναπαρίσταται και στο γραφικό της ζωής του (HPBar). Αν είναι νεκρός ο παίκτης τότε καλείται η function Die().

Αντίστοιχα, η function GetHit() από το script EnemyController, δέχεται ως παράμετρο το damage που κάνει ο παίκτης στο τέρας. Αν το τέρας είναι νεκρό τότε δεν γίνεται τίποτα. Αν δεν είναι όμως, η μεταβλητή condition του animator γίνεται 3 (Damage) και υπολογίζεται η τελική ζημιά (damage) που γίνεται στο τέρας από τον παίκτη.

```
public void GetHit(float damage)
{
    if (dead) return;
    anim.SetInteger("condition", 3);
    damageTaken = damage-def;
    currentHealth -= damageTaken;
    HPBar.fillAmount -= damageTaken / health;


    ShowDmgPopUp();
    StartCoroutine(RecoverFromHit());

}
```

Εικόνα 90 - Υλοποίηση - Script EnemyController - GetHit()

Στη συνέχεια καλούνται 2 επιπλέον συναρτήσεις οι οποίες εξυπηρετούν την καλύτερη αναπαράσταση του γραφικού περιβάλλοντος της μάχης.

Η μέθοδος ShowDmgPopUp() εμφανίζει σε τυχαίο σημείο κάθε φορά (όπως είδαμε και παραπάνω) τη ζημιά που δέχεται το τέρας από τον παίκτη.

```
1 reference
void ShowDmgPopUp()
{
    var go = Instantiate(DmgPopUp, transform.position, Quaternion.identity, transform);
    go.GetComponent<TextMesh>().text = damageTaken.ToString();

}
```

Εικόνα 91 - Υλοποίηση - Script EnemyController - ShowDmgPopUp()

Η επαναλαμβανόμενη μέθοδος RecoverFromHit() επιστρέφει το τέρας σε κατάσταση idle μέχρι το επόμενο χτύπημα του παίκτη.

```
IEnumerator RecoverFromHit()
{
        yield return new WaitForSeconds(0.1f);
        anim.SetInteger("condition", 0);

}
```

Εικόνα 92 - Υλοποίηση - Script EnemyController - RecoverFromHit()

Για την ολοκλήρωση της μάχης απομένει μόνο η μέθοδος Die() από το script του Player, η οποία καλείται όπως είδαμε όταν η ζωή του παίκτη είναι μικρότερη ή ίση του μηδενός.

Η μέθοδος αυτή, όπως φαίνεται παρακάτω, αρχικά μετατρέπει την τιμή του condition=3 ώστε να αναπαρασταθεί με το αντίστοιχο animation ο θάνατος του παίκτη, και στη συνέχεια καλείται η μέθοδος waitForDeathMessages() η οποία εμφανίζει το panel που ο παίκτης μπορεί να κάνει «Επανεκκίνηση» του παιχνιδιού ή «Έξοδος». Μέχρι ο παίκτης να πατήσει μία από τις δύο επιλογές το παιχνίδι βρίσκεται σε κατάσταση Paused.

```
void Die()
{
    anim.SetInteger("condition", 3);
    StartCoroutine(waitForDeathMessages());
}
```

Εικόνα 93 - Υλοποίηση - Script Player - Die()

```
IEnumerator waitForDeathMessages()
{
    yield return new WaitForSeconds(3);
    DefeatPanel.SetActive(true);
    Time.timeScale = 0f;
}
```

Εικόνα 94 - Υλοποίηση - Script Player - WaitForDeathMessages()

Ολοκληρώνοντας λοιπόν τη μάχη, αν ο παίκτης νικήσει το τέρας, τότε λαμβάνει εμπειρία ως ανταμοιβή. Αυτή η κατάσταση υλοποιείται μέσα από τη function DropLoot() η οποία καλείται μέσα από το animation του θανάτου του τέρατος και καλεί τη μέθοδο GetExp() από το script του Player.

```
public void DropLoot()
{
    p.GetExp(exp);
}
```

Εικόνα 95 - Υλοποίηση - Script EnemyController - DropLoot()

Η μέθοδος GetExp() δέχεται ως παράμετρο την εμπειρία που λαμβάνει ο παίκτης ως ανταμοιβή σκοτώνοντας ένα τέρας (ή ολοκληρώνοντας μια ερώτηση από τον NPC όπως θα δούμε παρακάτω), την προσθέτει στην υπάρχουσα εμπειρία (και σε δεδομένα και σε γραφικά) και στη συνέχεια καλείται η συνάρτηση checkCurrentXp().

```
public void GetExp(float xp)
{
    currentExp += xp;
    ExpBar.fillAmount += xp / exp;
    checkCurrentXP();
}
```

Εικόνα 96 - Υλοποίηση - Script Player - GetExp()

Σκοπός της checkCurrentXp() είναι ο έλεγχος του πότε ο παίκτης ανεβαίνει επίπεδο μέσα από την εμπειρία που μαζεύει ώστε να γίνονται οι απαραίτητες ενέργειες. Αρχικά γίνεται ο έλεγχος για το αν ο παίκτης ανέβηκε επίπεδο, αν ναι τότε του εμφανίζει ένα μήνυμα και στη συνέχεια του προσθέτει τις ανταμοιβές που έλαβε από αυτή την άνοδο. Τέλος, καλείται η μέθοδος checkHP() η οποία αποτελεί τον έλεγχο για τη σωστή λειτουργία της ζωής του παίκτη (δεδομένα και γραφικά).

```
void checkCurrentXP()
{
    while (currentExp>=exp)
    {
        LevelUp.SetActive(true);
        CongratsMessage.text = "Συγχαρητήρια!!!";
        currentExp -= exp;
        ExpBar.fillAmount = currentExp/exp ;
        level += 1;
        exp += 100;
        attackDamage += 1;
        def += 1;
        health += 100;
        checkHP();
        LevelMessage.text = "Μόλις ανέβηκες στο επίπεδο " + level.ToString()+"!!";
        StartCoroutine(waitForMessages());
    }
    return;
}
```

Εικόνα 97 - Υλοποίηση - Script Player - CheckCurrentXP()

```
void checkHP()
{
    currentHealth += 100;
    if (currentHealth>=health)
    {
        HPBar.fillAmount = 1;
        currentHealth = health;
    }
    else
    {
        HPBar.fillAmount = currentHealth / health;
    }

}
```

Εικόνα 98 - Υλοποίηση - Script Player - CheckHP()

Υλοποίηση αλληλεπίδρασης με NPC

Εφόσον ολοκληρώσαμε την υλοποίηση της κίνησης του παίχτη και της μάχης, σειρά έχει η υλοποίηση της αλληλεπίδρασης που υπάρχει μεταξύ του παίκτη και του NPC.

Αρχικά, ας ξεκινήσουμε από το script που αφορά το NPC το οποίο δεν το έχουμε αναφέρει μέχρι τώρα και στη συνέχεια θα δούμε και τις υπόλοιπες functions του script Player που σχετίζονται με τον NPC.

Ξεκινώντας λοιπόν, όπως και στα υπόλοιπα script, γίνεται ο ορισμός των μεταβλητών που θα χρησιμοποιηθούν καθώς και η αρχικοποίηση τους.

```
public class NPCController : MonoBehaviour
{

    //public string[] dialogues;
    //public int dialogueIndex=0;
    float lookRadius = 5f;
    Transform target;

    public GameObject DialogueBox;
    public GameObject Text;

    public GameObject npc;
    //public string npcTag;
    static GameObject npcLast;


    //Quest quest;
    public GameObject Title; //Quest Name Text
    public GameObject q; //Quest Button
    public GameObject qt; //Quest Text
    //public GameObject bi; //Buy Items Button
    public GameObject cl; //Close Button
    public GameObject answer; //input field answer
    public GameObject answerText;
    public GameObject submit;//Submit Button
    public GameObject conclusion; //Text to show if the answer is correct

    public GameObject RewardTitle;
    public GameObject RewardSword;
    public GameObject RewardArmor;
    public GameObject RewardStats;

    public Image RewardSwordImage;
    public Image RewardArmorImage;
```

Εικόνα 99 - Υλοποίηση - Script NPCController - Μεταβλητές 1

```
public Image RewardSwordImage;
public Image RewardArmorImage;
public Text RewardText;
public Text RewardAtkAmnt;
public Text RewardDefAmnt;
public Text RewardHealthAmnt;

public Sprite Armor1;
public Sprite Armor2;
public Sprite Armor3;
public Sprite Armor4;
public Sprite Armor5;

public Sprite Weapon1;
public Sprite Weapon2;
public Sprite Weapon3;
public Sprite Weapon4;
public Sprite Weapon5;

string SelectedSword;
string SelectedArmor;
string SelectedSwordLabel;
string SelectedArmorLabel;

public bool isForArmor;
int qid;
string qname;
string quest;
string qanswer;
int reward_id;
string rname; // reward name
string rdescr; //reward descr
int exp,atk, def, health;
string rLabel, rName;

public bool isFinal;
```

Εικόνα 100 - Υλοποίηση - Script NPCController - Μεταβλητές 2

Όπως εύκολα παρατηρείται οι μεταβλητές που έχουν οριστεί σε αυτό το script είναι πολλές καθώς η πληροφορία και οι ενέργειες που πραγματοποιεί ένα NPC αποτελούν ένα μεγαλό μέρος του παιχνιδιού. Επίσης, κάτι ακόμα αξιοσημείωτο είναι ότι οι πιο πολλες μεταβλητές είναι public και αφορούν GameObjects, αυτό συμβαίνει γιατί πρέπει να περαστεί το γραφικό μέρος του παχνιδιού μέσα στον κώδικα ώστε να χρησιμοποιηθεί όπου είναι απαραίτητο.

Επόμενο βήμα είναι η function Start() η οποία χρησιμοποιείται ώστε ένα NPC να αναγνωρίσει το target του που είναι ο παίκτης και να οριστεί ότι το panel του διαλογού με τον παίκτη δεν θα είναι active (active=false) ώσπου να το ενεργοποιήσει ο παίκτης.

```
void Start()
{
    target = GameObject.FindGameObjectWithTag("Player").transform;
    DialogueBox.SetActive(false);

}
```

Εικόνα 101 - Υλοποίηση - Script NPCController - Start()

Για να ενεργοποιηθεί τώρα το panel διαλόγου θα πρέπει ο παίκτης να βρίσκεται σε συγκεκριμένη απόσταση και σε γωνία 45 μοιρών από τον NPC και να πατήσει το κουμπί «E». Για να επιτευχθεί η συγκεκριμένη ενέργεια, χρησιμοποιήθηκε η function Update() η οποία όπως έχουμε δει επαναλαμβάνεται συνέχεια μέσα στο παιχνίδι. Οπότε όταν ο παίκτης είναι κοντά στον NPC και πατήσει το «E» τότε πραγματοποιούνται οι ενέργειες που περιγράφει η function Update().

```
private void Update()
{
    float distance = Vector3.Distance(target.position, transform.position);
    Vector3 direction = target.position - transform.position;
    float angle = Vector3.Angle(direction, this.transform.forward);
    if (distance <= lookRadius && angle < 45)
    {
        this.transform.rotation = Quaternion.Slerp(this.transform.rotation, Quaternion.LookRotation(direction), 0.1f);
        if (Input.GetKeyDown(KeyCode.E))
        {
            ShowDialogue();
        }


    }
}
```

Εικόνα 102 - Υλοποίηση - Script NPCController - Update()

Εφόσον καλύπτονται τα κριτήρια απόστασης, γωνίας και πατήματος σωστού κουμπιού (E), τότε καλείται η μέθοδος ShowDialogue() η οποία φέρνει στην οθόνη τα απαραίτητα γραφικά (panel, buttons, texts κτλπ), και ορίζει τις ενέργειες στα κουμπιά ώστε στο κλικ του καθενός να ενεργοποιηθεί η κατάλληλη μέθοδος (βλ. onClick.AddListener()).

```
public void ShowDialogue()
{
    npcLast = GameObject.FindGameObjectWithTag(npc.name);

    DialogueBox.SetActive(true);
    answer.SetActive(false);
    submit.SetActive(false);
    conclusion.SetActive(false);
    q.SetActive(true);
    qt.SetActive(false);
    cl.SetActive(true);
    Text.SetActive(true);
    Title.SetActive(true);
    RewardTitle.SetActive(false);
    RewardArmor.SetActive(false);
    RewardSword.SetActive(false);
    RewardStats.SetActive(false);
    Title.GetComponent<Text>().text = "Μάγος του Κάστρου";
    //bi.GetComponentInChildren<Text>().text = "BUY ITEMS";
    Text.GetComponent<Text>().text = "Καλωσήρθες Ταξιδιώτη, βρίσκομαι εδώ για να σε βοηθήσω. " +
        "Το ταξίδι σου δεν χρειάζεται άγχος, " +
        "χρησιμοποίησε το μυαλό σου και το τέλος είναι κοντά. Καλή επιτυχία!";

    q.GetComponent<Button>().onClick.AddListener(Quest);

    cl.GetComponent<Button>().onClick.AddListener(Close);

}
```

Εικόνα 103 - Υλοποίηση - Script NPCController - ShowDialogue()

Ξεκινώντας από την πιο απλή μέθοδος, αν ο παίκτης πατήσει «Έξοδος» τότε καλείται η function Close() η οποία κλείνει το παράθυρο, θέτοντάς το πάλι ως active=false.

```
public void Close()
{
    DialogueBox.SetActive(false);
}
```

Εικόνα 104 - Υλοποίηση - Script NPCController - Close()

Το άλλο κουμπί που υπάρχει στην πρώτη οθόνη του NPC είναι το «Ερώτηση». Πατώντας αυτό το κουμπί ο παίκτης τότε καλείται η μέθοδος Quest() η οποία ενεργοποιεί μια αλυσίδα μεθόδων που χρησιμοποιούνται ώστε να ολοκληρωθεί με επιτυχία η αλληλεπίδραση μεταξύ παίκτη και NPC.

Επειδή τώρα οι επόμενες μέθοδοι είναι μεγάλες, καλό είναι να τις δούμε σε κομμάτια ώστε να είναι πιο εύκολες στην κατανόηση. Αρχικά ξεκινώντας η μέθοδος Quest() καθορίζει ποια γραφικά θα εμφανιστούν και ποια όχι.

```
void Quest()
{
    q.SetActive(false);
    cl.SetActive(false);
    qt.SetActive(true);
    Text.SetActive(false);
    RewardTitle.SetActive(false);
    RewardArmor.SetActive(false);
    RewardSword.SetActive(false);
    RewardStats.SetActive(false);
```

Εικόνα 105 - Υλοποίηση - Script NPCController - Quest() Components

Επόμενο βήμα είναι η άντληση των ενεργών (active=1) ερωτήσεων από τη βάση η οποία πραγματοποιείται ελέγχοντας αν η ανταμοιβή που θα δώσει το συγκεκριμένο NPC αφορά Armor ή Weapon. Ολοκληρώνοντας την άντληση της πληροφορίας από τη ΒΔ, γίνεται η εισαγωγή αυτής στις αντίστοιχες μεταβλητές που θα χρησιμοποιηθούν από το script.

```
using (IDbCommand cmnd_read = ConnectToDatabase.Dbcon.CreateCommand())
{
    IDataReader reader;
    string query;
    if (isForArmor)
    {
        query= "SELECT qid,qname,quest,qanswer,exp,rLabel,rName,atk,def,health FROM NPC where active=1 and qid <= 5 LIMIT 1";
    }
    else
    {
        query = "SELECT qid,qname,quest,qanswer,exp,rLabel,rName,atk,def,health FROM NPC where active=1 and qid > 5 LIMIT 1";
    }
    cmnd_read.CommandText = query;
    reader = cmnd_read.ExecuteReader();
    while (reader.Read())
    {
        qid = reader.GetInt32(0);
        qname = reader.GetString(1);
        quest = reader.GetString(2);
        qanswer = reader.GetString(3);
        exp = reader.GetInt32(4);
        rLabel = reader.GetString(5);
        rName = reader.GetString(6);
        atk = reader.GetInt32(7);
        def = reader.GetInt32(8);
        health = reader.GetInt32(9);

    }
}
```

Εικόνα 106 - Υλοποίηση - Script NPCController - Quest() Select

Στο τελευταίο κομμάτι της function Quest() γεμίζουν τα αντίστοιχα πεδία του panel με την πληροφορία και ορίζεται στο κουμπί «Υποβολή» η μέθοδος SubmitQuest() η οποία ελέγχει αν ο παίκτης απάντησε σωστά στην ερώτηση.

```
qt.GetComponent<Text>().text = quest;
Title.GetComponent<Text>().text = qname;

answer.SetActive(true);
submit.SetActive(true);
conclusion.SetActive(true);
submit.GetComponent<Button>().onClick.AddListener(SubmitQuest);
```

Εικόνα 107- Υλοποίηση - Script NPCController - Quest() Final Part

Επόμενη και τελευταία μέθοδος στο script NPCController, είναι η μέθοδος SubmitQuest() η οποία καλείται όταν πατηθεί το κουμπί «Υποβολή». Αρχικά στη συγκεκριμένη μέθοδο, ελέγχεται αν η απάντηση που έδωσε ο παίκτης είναι σωστή. Αν είναι σωστή τότε η ερώτηση στη ΒΔ γίνεται active=0 και γίνεται η κατάλληλη επεξεργασία στα γραφικά για το ποια θα εμφανιστούν.

```
public void SubmitQuest()
{
    if (qanswer == answerText.GetComponent<Text>().text)
    {
        IDbCommand update = ConnectToDatabase.Dbcon.CreateCommand();

        string sqlQuery = "Update Quests set active=0 where id=" + qid;
        update.CommandText = sqlQuery;
        update.ExecuteScalar();

        answer.SetActive(false);
        submit.SetActive(false);
        conclusion.SetActive(false);
        q.SetActive(false);
        qt.SetActive(false);
        cl.SetActive(true);
        Text.SetActive(false);
        Title.SetActive(true);
        RewardTitle.SetActive(true);
```

Εικόνα 108 - Υλοποίηση - Script NPCController - SubmitQuest() Initial Part

Στη συνέχεια ελέγχεται η ανταμοιβή της ερώτησης, αν αφορά Armor ή Weapon ώστε να δοθεί η αντίστοιχη στον παίκτη, με βάση την πληροφορία που έχουμε στη ΒΔ για κάθε NPC.

```
if (isForArmor)
{
    RewardArmor.SetActive(true);
    switch (rLabel)
    {
        case "Armor1":
            RewardArmorImage.sprite = Armor1;
            SelectedArmor = "Armor1";
            SelectedArmorLabel = rName;
            break;
        case "Armor2":
            RewardArmorImage.sprite = Armor2;
            SelectedArmor = "Armor2";
            SelectedArmorLabel = rName;
            break;
        case "Armor3":
            RewardArmorImage.sprite = Armor3;
            SelectedArmor = "Armor3";
            SelectedArmorLabel = rName;
            break;
        case "Armor4":
            RewardArmorImage.sprite = Armor4;
            SelectedArmor = "Armor4";
            SelectedArmorLabel = rName;
            break;
        case "Armor5":
            RewardArmorImage.sprite = Armor5;
            SelectedArmor = "Armor5";
            SelectedArmorLabel = rName;
            break;
    }
}
```

Εικόνα 109 - Υλοποίηση - Script NPCController - SubmitQuest() Rewards

```
else
{
    RewardSword.SetActive(true);
    switch (rLabel)
    {
        case "Weapon1":
            RewardArmorImage.sprite = Weapon1;
            SelectedSword = "Weapon1";
            SelectedSwordLabel = rName;
            break;
        case "Weapon2":
            RewardArmorImage.sprite = Weapon2;
            SelectedSword = "Weapon2";
            SelectedSwordLabel = rName;
            break;
        case "Weapon3":
            RewardArmorImage.sprite = Weapon3;
            SelectedSword = "Weapon3";
            SelectedSwordLabel = rName;
            break;
        case "Weapon4":
            RewardArmorImage.sprite = Weapon4;
            SelectedSword = "Weapon4";
            SelectedSwordLabel = rName;
            break;
        case "Weapon5":
            RewardArmorImage.sprite = Weapon5;
            SelectedSword = "Weapon5";
            SelectedSwordLabel = rName;
            break;
    }
}
```

Εικόνα 110 - Υλοποίηση - Script NPCController - SubmitQuest() Rewards

Εφόσον ολοκληρωθεί ο έλεγχος της ανταμοιβής που θα δοθεί, τότε εμφανίζεται το παράθυρο με τις ανταμοιβές που έλαβε ο παίκτης από την ερώτηση και στη συνέχεια λαμβάνει τα bonus εμπειρίας και στατιστικών καλώντας δύο ακόμα μεθόδους από το script του παίκτη την GetExp() και την getRewardStats(). Τέλος μετά από 6 δευτερόλεπτα το NPC καταστρέφεται.

```
RewardStats.SetActive(true);
RewardText.text = rName;
RewardAtkAmnt.text = "+"+atk.ToString();
RewardDefAmnt.text = "+"+def.ToString();
RewardHealthAmnt.text = "+"+health.ToString();
Title.GetComponent<Text>().text = "Ανταμοιβή";
cl.GetComponent<Button>().onClick.AddListener(Close);
Player p = target.GetComponent<Player>();
p.GetExp(exp);
p.getRewardStats(atk, def, health,SelectedSword,SelectedArmor,SelectedSwordLabel,SelectedArmorLabel);
StartCoroutine(waitForRewards());
Destroy(npcLast, 6);
```

Εικόνα 111 - Υλοποίηση - Script NPCController - SubmitQuest() Give Rewards

Αν τώρα η απάντηση δεν είναι σωστή εμφανίζεται στον παίκτη ένα ενημερωτικό μήνυμα.

```
else
{
    conclusion.SetActive(true);
    conclusion.GetComponent<Text>().text = "Η απάντηση είναι λάθος!";
}
```

Εικόνα 112 - Υλοποίηση - Script NPCController - SubmitQuest() Λάθος Απάντηση

Για να ολοκληρωθεί τώρα η αλληλεπίδραση μεταξύ NPC και παίκτη, απομένει η μέθοδος getRewardStats(), καθώς η μέθοδος GetExp() είναι η ίδια που χρησιμοποιείται και κατά το τέλος της μάχης και την περιγράψαμε παραπάνω.

Όσον αναφορά τώρα τη μέθοδο getRewardStats(), είναι η μέθοδος η δέχεται ως παραμέτρους τα στατιστικά που λαμβάνει ο παίκτης ως ανταμοιβή από την ερώτηση και τα προσθέτει στα ήδη υπάρχοντα. Επίσης, καλείται η συνάρτηση SetEquipment() από το script PlayerStats όπως θα δούμε παρακάτω.

```
public void getRewardStats(int atk, int RewardDef, int RewardHealth, string Sword, string Armor,string SwordLabel,string ArmorLabel)
{
    rewardAtk = atk;
    rewardDef = RewardDef;
    rewardHealth = RewardHealth;
    health += rewardHealth;
    currentHealth += rewardHealth;
    attackDamage += rewardAtk;
    def += rewardDef;

    PlayerStats.SetEquipment(Sword,Armor,SwordLabel,ArmorLabel);
}
```

Εικόνα 113 - Υλοποίηση - Script NPCController - GetRewardStats()

Η μέθοδος SetEquipment(), χρησιμοποιείται ώστε να ξέρει το παιχνίδι ανά πάσα στιγμή τι αντικείμενα έχει ο παίκτης ώστε να εμφανίζονται σωστά όταν ο παίκτης θέλει να τα δει.

```
public static void SetEquipment(string RewardSword,string RewardArmor,string WeaponLabel,string ArmorLabel)
{
    Weapon = RewardSword;
    Armor = RewardArmor;
    WeaponName = WeaponLabel;
    ArmorName = ArmorLabel;
}
```

Εικόνα 114 - Υλοποίηση - Script PlayerStats - SetEquipment()

Όπως αναφέραμε η μέθοδος SetEquipment() ανήκει στο script PlayerStats, το οποίο δημιουργήθηκε ώστε να μπορεί ο παίκτης να γνωρίζει τα στατιστικά του καθώς και τα αντικείμενα που έχει πατώντας το κουμπί «C».

Αρχικά, για το συγκεκριμένο script ορίσαμε τις μεταβλητές, οι οποίες κατά κύριο λόγο αναφέρονται σε γραφικά αντικείμενα που μεταφέρονται μέσα από το Unity ώστε να εμφανίζονται σωστά όταν αυτό ζητηθεί. Για να το πετύχουμε αυτό, κατά την έναρξη του παιχνιδιού (μέθοδος Start()) η μεταβλητή p ορίζεται ως την τιμή του script Player ώστε να μπορεί να παίρνει τιμές από το συγκεκριμένο script όπως θα δούμε στη συνέχεια.

```
public class PlayerStats : MonoBehaviour
{
    public GameObject Stats;
    public Text SwordLabel;
    public Text ArmorLabel;
    public Image SwordImage;
    public Image ArmorImage;
    public Text LevelAmount;
    public Text ExpAmount;
    public Text AtkAmount;
    public Text DefAmount;
    public Text HealthAmount;

    public Sprite Armor1;
    public Sprite Armor2;
    public Sprite Armor3;
    public Sprite Armor4;
    public Sprite Armor5;

    public Sprite Weapon1;
    public Sprite Weapon2;
    public Sprite Weapon3;
    public Sprite Weapon4;
    public Sprite Weapon5;

    static bool isOpen = false;
    static string Weapon;
    static string Armor;
    static string WeaponName;
    static string ArmorName;
    Transform target;
    Player p;
```

Εικόνα 115 - Υλοποίηση - Script PlayerStats - Μεταβλητές

```
private void Start()
{
    target = GameObject.FindGameObjectWithTag("Player").transform;
    p = target.GetComponent<Player>();
    Stats.SetActive(false);

}
```

Εικόνα 116 - Υλοποίηση - Script PlayerStats - Start()

Για να είναι διαθέσιμη η προβολή των στατιστικών ανά πάσα στιγμή, αρκεί να οριστεί μέσα στη μέθοδο Update(). Βασική δουλειά της συγκεκριμένης μεθόδου είναι σε κάθε στιγμή μέσα στο παιχνίδι να γνωρίζει τα στατιστικά του παίκτη, οπότε λαμβάνουμε τις τιμές από το script Player και τις ορίζουμε στα αντίστοιχα πεδία των χαρακτηριστικών.

```
void Update()
{
    LevelAmount.text = p.level.ToString();
    ExpAmount.text = p.currentExp.ToString() + " / " + p.exp.ToString();
    AtkAmount.text = p.attackDamage.ToString();
    DefAmount.text = p.def.ToString();
    HealthAmount.text = p.currentHealth.ToString() + " / " + p.health.ToString();
```

Εικόνα 117 - Υλοποίηση - Script PlayerStats - Update() Μεταβλητές Χαρακτηριστικών

Επόμενο βήμα, εφόσον ολοκληρώθηκε ο ορισμός των τιμών είναι η εμφάνισή τους. Για το λόγο αυτό γίνεται ένας συνεχής έλεγχος για το αν ο παίκτης έχει πατήσει το κουμπί «C», όταν πατηθεί το συγκεκριμένο κουμπί τότε ελέγχεται αν το panel των στατιστικών είναι ανοιχτό, αν ναι τότε το panel κλείνει, αλλιώς ανοίγει.

```
if (Input.GetKeyDown(KeyCode.C))
{
    if (isOpen)
    {
        isOpen = false;
        Stats.SetActive(false);
    }
    else
    {

        isOpen = true;
        Stats.SetActive(true);
        if (Weapon == null && Armor==null)
        {
            return;
        }
```

Εικόνα 118 - Υλοποίηση - Script PlayerStats - Update() Εμφάνιση Χαρακτηριστικών

Για να ολοκληρωθεί η σωστή εμφάνιση των χαρακτηριστικών του παίκτη, πρέπει να γνωρίζουμε και τα αντικείμενα του εκείνη την στιγμή. Βασικό ρόλο σε αυτό παίζει η συνάρτηση SetEquipment() την οποία είδαμε παραπάνω. Εφόσον μέσω της μεθόδου αυτής ο κώδικας γνωρίζει τις ονομασίες των αντικειμένων, αρκεί να τις συσχετίσει με τα αντικείμενα ώστε να εμφανιστούν σωστά στην οθόνη. Για να το πετύχουμε αυτό χρησιμοποιήσαμε 2 συναρτήσεις switch, μία για Armor και μία για Weapon.

```
switch (Weapon)
{
    case "Weapon1":
        SwordImage.sprite = Weapon1;
        SwordLabel.text = WeaponName;
        break;
    case "Weapon2":
        SwordImage.sprite = Weapon2;
        SwordLabel.text = WeaponName;
        break;
    case "Weapon3":
        SwordImage.sprite = Weapon3;
        SwordLabel.text = WeaponName;
        break;
    case "Weapon4":
        SwordImage.sprite = Weapon4;
        SwordLabel.text = WeaponName;
        break;
    case "Weapon5":
        SwordImage.sprite = Weapon5;
        SwordLabel.text = WeaponName;
        break;
}
```

Εικόνα 119 - Υλοποίηση - Script PlayerStats - Update() Εμφάνιση Αντικειμένων

```
switch (Armor)
{
    case "Armor1":
        ArmorImage.sprite = Armor1;
        ArmorLabel.text = ArmorName;
        break;
    case "Armor2":
        ArmorImage.sprite = Armor2;
        ArmorLabel.text = ArmorName;
        break;
    case "Armor3":
        ArmorImage.sprite = Armor3;
        ArmorLabel.text = ArmorName;
        break;
    case "Armor4":
        ArmorImage.sprite = Armor4;
        ArmorLabel.text = ArmorName;
        break;
    case "Armor5":
        ArmorImage.sprite = Armor5;
        ArmorLabel.text = ArmorName;
        break;
}
```

Εικόνα 120 - Υλοποίηση - Script PlayerStats - Update() Εμφάνιση Αντικειμένων

Υλοποίηση της ολοκλήρωσης του παιχνιδιού

Για να ολοκληρωθεί το παιχνίδι, όπως αναφέραμε και κατά τον σχεδιασμό αυτού, ο παίκτης πρέπει να μιλήσει στον τελευταίο NPC. Για να γίνεται πιο εύκολη η κατανόηση του κώδικα καθώς και η επεξεργασία του, η συγκεκριμένη λειτουργία υλοποιήθηκε σε ένα ξεχωριστό script το FinalNPCController. Οι ενέργειες που πραγματοποιούνται σε αυτό είναι παρεμφερής με αυτές του απλού NPC, με μόνη διαφορά ότι δεν γίνεται ερώτηση στον παίκτη, αλλά του εμφανίζεται ο χρόνος που έκανε καθώς και ο χρόνος των πέντε πιο γρήγορων παικτών. Όπως και στο script NPCController έτσι και εδώ, ορίζονται οι μεταβλητές, «αναγνωρίζεται» το αντικείμενο παίκτης και στη συνέχεια καθορίζεται το πότε θα ενεργοποιηθούν οι ενέργειες του τελικού NPC.

Η μόνη διαφορά με το script του NPC είναι η function Finish() η οποία καλείται όταν ο παίκτης πατήσει το κουμπί ολοκλήρωση. Η ενέργειες που πραγματοποιεί η συγκεκριμένη μέθοδος είναι η άντληση των στατιστικών από τη ΒΔ και η εμφάνιση τους στο τελικό panel.

```
public void Finish()
{
    FinalDialogueBox.SetActive(false);
    FinalStats.SetActive(true);
    Text.text = "Ο χρόνος σου στο παιχνίδι ήταν: " + Timer.finishedTime+ "\nΠαρακάτω εμφανίζονται οι κορυφαίοι 5 παίκτες.";
    using (IDbCommand cmnd_read = ConnectToDatabase.Dbcon.CreateCommand())
    {
        IDataReader reader;
        string query = "select name,time from DisplayStats LIMIT 5;";
        cmnd_read.CommandText = query;
        reader = cmnd_read.ExecuteReader();
        int i = 0;
        while (reader.Read())
        {
            Names.Add(reader.GetString(0));
            Times.Add(reader.GetString(1));
        }
        reader.Close();
        Name1.text = Names[0];
        Name2.text = Names[1];
        Name3.text = Names[2];
        Name4.text = Names[3];
        Name5.text = Names[4];

        Time1.text = Times[0];
        Time2.text = Times[1];
        Time3.text = Times[2];
        Time4.text = Times[3];
        Time5.text = Times[4];
        Time.timeScale = 0f;
    }
}
```

Εικόνα 121- Υλοποίηση - Script FinalNPCController - Finish()

Σε αυτό το σημείο ολοκληρώνεται και το παιχνίδι, εφόσον ο παίκτης δει τους 5 καλύτερους χρόνους και πατώντας το κουμπί έξοδος, το παιχνίδι επιστρέφει στην αρχική οθόνη και δίνει τη δυνατότητα στον παίκτη να ξεκινήσει από την αρχή.

Συμπεράσματα

Φτάνοντας λοιπόν στο τέλος και ολοκληρώνοντας την συγκεκριμένη εργασία, προέκυψαν αρκετά συμπεράσματα όσον αναφορά την σχεδίαση και την υλοποίηση ενός παιχνιδιού. Αξίζει να σημειωθεί ότι η συγκεκριμένη εργασία είναι η πρώτη μου επαφή με τον κόσμο του Game Development.

Αρχικά, η πλατφόρμα Unity που χρησιμοποιήθηκε για την συγκεκριμένη εργασία, είναι αρκετά εύχρηστη και εύκολη στην εκμάθηση της, καθώς υπάρχουν αρκετά tutorials και επεξηγήσεις στο διαδίκτυο. Επίσης, οι δυνατότητες που σου παρέχει η πλατφόρμα μέσα από το UI της είναι πολλές, πράγμα που δίνει την ευκαιρία στον χρήστη της να δημιουργήσει αρκετές λεπτομέρειες για να κάνει το παιχνίδι πιο ενδιαφέρον.

Το γεγονός ότι η συγκεκριμένη πλατφόρμα χρησιμοποιεί για την υλοποίηση των Scripts μια γλώσσα προγραμματισμού (C#) η οποία είναι ευρέως γνωστή και με αρκετά tutorials στο διαδίκτυο είναι ένα μεγάλο συν για κάποιον ο οποίος ήδη τη γνωρίζει και θέλει να ασχοληθεί με την ανάπτυξη παιχνιδιών. Προσωπικά, αυτός ήταν ο λόγος για τον οποίο επέλεξα την πλατφόρμα Unity.

Ένα ακόμα μεγάλο πλεονέκτημα, το οποίο με βοήθησε αρκετά είναι ότι υπάρχει μεγάλη ποικιλία από ολοκληρωμένα assets δωρεάν στο Asset Store της πλατφόρμας, ώστε να επιλέξει ο χρήστης ποιο καλύπτει τις ανάγκες του, τα οποία είναι εύκολα στη χρήση τους, ακόμα και για ανθρώπους που δεν γνωρίζουν και δεν έχουν ασχοληθεί καθόλου με γραφικά πριν την πρώτη τους επαφή με τη Unity.

Ως μειονέκτημα θα μπορούσα να θέσω το γεγονός ότι η Unity είναι πολλή απαιτητική όσον αναφορά τους πόρους που χρειάζεται από το μηχάνημα στο οποίο τρέχει. Καλή κάρτα γραφικών, αρκετή μνήμη ram και καλό επεξεργαστή, είναι μερικά από τα στοιχεία που χρειάζεται η συγκεκριμένη πλατφόρμα για να τρέξει σωστά σε έναν υπολογιστή, πράγμα που αναγκάζει τον χρήστη να ξοδέψει αρκετά χρήματα ώστε να εκμεταλλευτεί όλα τα πλεονεκτήματα της.

Ολοκληρώνοντας λοιπόν, θα ήθελα να αναφέρω την προσωπική μου άποψη πάνω στην ανάπτυξη των παιχνιδιών. Όντας ένας άνθρωπος ο οποίος ασχολείται επαγγελματικά πλέον με τον προγραμματισμό τα τελευταία 3 χρόνια, η σωστή ανάπτυξη ενός παιχνιδιού αποτελεί μια αρκετά χρονοβόρα αλλά ενδιαφέρουσα διαδικασία, η οποία απαιτεί την απασχόληση αρκετών ανθρώπων διαφορετικών ειδικοτήτων ώστε το αποτέλεσμα να καλύπτει τις ανάγκες του μεγαλύτερου ποσοστού των gamers. Επίσης, χρειάζονται γνώσεις και από τομείς εκτός της πληροφορικής, όπως πχ μαθηματικών, ώστε να στηθεί σωστά το παιχνίδι ειδικά αν μιλάμε για παιχνίδι 3D.

Δυστυχώς στην Ελλάδα, δεν έχει δοθεί βάση στον συγκεκριμένο τομέα, με αποτέλεσμα να μην υπάρχει η γνώση, η εμπειρία και η τεχνογνωσία που απαιτείται για την σωστή υλοποίηση ενός παιχνιδιού, κάτι το οποίο δεν δίνει ευκαιρίες σε ανθρώπους που θέλουν να ασχοληθούν με το συγκεκριμένο αντικείμενο.

Βιβλιογραφία – Ιστότοποι

- Ιστορία των Video Games
  https://el.wikipedia.org/wiki/%CE%99%CF%83%CF%84%CE%BF%CF%81%CE%AF%CE%B1_%CF%84%CF%89%CE%BD_%CE%B2%CE%B9%CE%BD%CF%84%CE%B5%CE%BF%CF%80%CE%B1%CE%B9%CF%87%CE%BD%CE%B9%CE%B4%CE%B9%CF%8E%CE%BD

- Άρθρο για την εξέλιξη των Video Games
  https://smassingculture.gr/i-exelixi-tou-psifiakou-pechnidiou/

- RPG Video Game :
  https://el.wikipedia.org/wiki/%CE%92%CE%B9%CE%BD%CF%84%CE%B5%CE%BF%CF%80%CE%B1%CE%B9%CF%87%CE%BD%CE%AF%CE%B4%CE%B9_%CF%81%CF%8C%CE%BB%CF%89%CE%BD

- Unity User Manual
  https://docs.unity3d.com/Manual/index.html

- Unity Scripting API
  https://docs.unity3d.com/ScriptReference/index.html

- Unity Asset Store
  https://assetstore.unity.com/

- How to make an RPG in Unity Tutorial
  https://www.youtube.com/watch?v=nu5nyrB9U_o&list=PLPV2KyIb3jR4KLGCCAciWQ5qHudKtYeP7&ab_channel=Brackeys

- Let's make an RPG Tutorial (Basic Following Tutorial)
  https://www.youtube.com/watch?v=CGBQHvYovWc&list=PLTm4FjoXO7nfEe8vx3Jjn-dGN4YZberSS&ab_channel=FireBrainGames%28Creagines%29

- Menus in Unity
  https://www.youtube.com/watch?v=zc8ac_qUXQY&list=PLPV2KyIb3jR4JsOygkHOd2q0CFoslwZOZ

- DB Browser for SQLite Wiki
  https://github.com/sqlitebrowser/sqlitebrowser/wiki

- Vahé Karamian - Building An Rpg With Unity 2018 - Second Edition [EPUB]
  https://vdoc.pub/documents/building-an-rpg-with-unity-2018-second-edition-5sunk7p7te80