ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Σχολή Μηχανικών
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πρόγραμμα Σπουδών – Μηχανικών Πληροφορικής Τ.Ε.

Πτυχιακή Εργασία

Σχεδιασμός και υλοποίηση ενός Turn-Based RPG σε Unity 3D

ΜΠΑΛΛΑ ΧΡΙΣΤΙΝΑ - ΑΜ 4771

Επιβλέπων Καθηγητής: Παχουλάκης Ιωάννης

ΗΡΑΚΛΕΙΟ 2023

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον υπεύθυνο καθηγητή, κ. Παχουλάκη Ιωάννη, για την υποστήριξη που μου παρείχε, καθ' όλη τη διάρκεια εκπόνησης της παρούσας πτυχιακής εργασίας. Επίσης ένα μεγάλο ευχαριστώ, οφείλω στην οικογένεια μου που με στήριξε και με στηρίζει σε κάθε ακαδημαϊκό μου βήμα.

Περίληψη

Η παρούσα πτυχιακή εργασία, παρουσιάζει τον σχεδιασμό και την υλοποίηση ενός Turn-Based RPG, χρησιμοποιώντας τη Unity3D Game Engine.

Η διαδικασία σχεδιασμού θα επικεντρωθεί, στη δημιουργία ενός ισορροπημένου και ελκυστικού συστήματος μάχης, διατηρώντας παράλληλα της αίσθηση της ιστορίας του φανταστικού. Ο σχεδιασμός περιλαμβάνει βασικά δομικά στοιχεία, όπως η εξέλιξη των χαρακτήρων κατά τη διάρκεια των μαχών, όσον αφορά τα στατιστικά τους, η εχθρική τεχνητή νοημοσύνη και η διεπαφή του χρήστη. Στοιχεία τα οποία συνδυάζονται με τέτοιο τρόπο, ώστε να βγει ένα άρτιο αποτέλεσμα.

Η διαδικασία υλοποίησης θα αναλυθεί σε τέσσερα μέρη, χαρακτήρες και τα χαρακτηριστικά τους, σχεδιασμός του παιχνιδιού, υλοποίηση κώδικα, καθώς και βελτιστοποίηση της απόδοσης. Για τους χαρακτήρες θα αναλυθούν οι ενέργειες που μπορούν να κάνουν κατά τη διάρκεια του παιχνιδιού καθώς και το εκάστοτε τρισδιάστατο μοντέλο τους. Για τον σχεδιασμό, θα περιγραφούν αναλυτικά όλες οι διαδικασίες, από τη δημιουργία της διεπαφής χρήστη, μέχρι και το πως έχει δημιουργηθεί η κάθε σκηνή από το μηδέν. Όσον αφορά την υλοποίηση κώδικα, περιγράφονται τα σημεία εκείνα, τα οποία αποτελούν σημαντικά προγραμματιστικά μέρη, με χρήση φωτογραφιών του κώδικα, μέσα στον οποίον υπάρχουν αναλυτικά σχόλια. Τέλος το σύστημα ελέγχεται και βελτιστοποιείται, χρησιμοποιώντας διάφορες μετρήσεις απόδοσης της Unity στον Profiler, όπως τα FPS και η χρήση της μνήμης. Τα αποτελέσματα από αυτού του ελέγχου παρουσιάζονται μαζί με τις επιπτώσεις τους για το σύστημα. Τέλος, παρουσιάζονται οι δυσκολίες που αντιμετωπίστηκαν, καθώς και πιθανές μελλοντικές βελτιώσεις και επεκτάσεις.

Το τελικό αποτέλεσμα της παρούσας πτυχιακής εργασίας, θα είναι ένα πλήρως λειτουργικό παιχνίδι, όπου θα παρέχει στον χρήστη μία ελκυστική και ταυτόχρονα καθηλωτική εμπειρία.

Abstract

This thesis, presents the design and implementation of a Turn-Based RPG, using the Unity 3D Game Engine.

The design process, will focus on creating a balanced and engaging battle system, while maintaining the feeling of the fictional story in a fantasy world. The design includes basic structural blocks, such as , character progression during battles, in terms of their stats, enemy Artificial Intelligence and user interface. Elements which are combined in such a way, to have an excellent result come out.

The implementation process will be broken down into four parts, characters and their features, game design, code implementation and performance optimization. For the characters, the actions they can perform, during the game, will be analyzed , as well as their 3D model. For the design, all the processes will be described in detail, from the creation of the user interface, to how each scene is created from scratch. Regarding the code implementation, the important programming parts, are described, using snippets of the code, in which there are detailed comments. Then, the system is tested and optimized, using various Unity performance metrics in the Profiler tool, such as FPS and CPU usage. The results, from this audit, are presented, along with their implications for the system. Finally, the difficulties encountered along the way, they are presented , as well as the possible future improvements and extensions.

The result of this thesis, will be a fully functional game, which will provide the user, an attractive and at the same time, immersive experience.

Περιεχόμενα

Λίστα Εικόνων

---- ΕΙΚΟΝΕΣ ΠΑΡΑΡΤΗΜΑΤΟΣ ----

Λίστα Πινάκων

1. Εισαγωγή

1.1 Περίληψη του Παιχνιδιού

Το 3D Turn-Based Battle “Corrupted Lands”, είναι ένα παιχνίδι στρατηγικής και φαντασίας, το οποίο μπορεί να παιχτεί σε κινητά. Ο χρήστης θα παίρνει τη σειρά του, και με γνώμονα την στρατηγική, θα σχεδιάζει τις κινήσεις της ομάδας του, με σκοπό την ήττα της αντίπαλης ομάδας, που στην προκειμένη περίπτωση θα αντιπροσωπεύεται από τον υπολογιστή.

Το παιχνίδι διαδραματίζεται στη μεσαιωνική εποχή με στοιχεία ενός μαγικού κόσμου. Τα λεγόμενα, πλέον, “Corrupted Lands”, είναι ένα μακρινό βασίλειο, το οποίο κατατροπώθηκε πριν πολλά χρόνια από τέρατα του αρχαίου κόσμου. Οι τρεις ήρωες, χρησιμοποιώντας την πολεμική τους ικανότητα, καθώς και τη μαγεία που έχουν μέσα τους, προσπαθούν να λυτρώσουν τις τρεις εναπομείναντες περιοχές, ώστε να επαναφέρουν το βασίλειο στη δόξα του.

1.2 Κίνητρο για τη Διεξαγωγή της Εργασίας

Το κίνητρο για τη διεξαγωγή της παρούσας πτυχιακής εργασίας, προέρχεται από την αγάπη που έχω για τα παιχνίδια και ιδιαιτέρως για το γραφιστικό κομμάτι τους. Κάθε φορά που παίζω ένα παιχνίδι, πάντα θαυμάζω το περιβάλλον και την έμφαση στη λεπτομέρεια κάθε σκηνής. Παράλληλα αποτέλεσε λόγο και η θέληση να δημιουργήσω ένα παιχνίδι από την αρχή, έχοντας μια ιδέα και συνδυάζοντάς την με το δημιουργικό κομμάτι του χαρακτήρα μου, ώστε να βγει ένα άρτιο αποτέλεσμα.

1.3 Σκοπός και Στόχος της Εργασίας

Ο σκοπός της παρούσας πτυχιακής εργασίας, είναι ο σχεδιασμός και η υλοποίηση ενός 3D Turn-Based παιχνίδι ρόλων, σε γλώσσα προγραμματισμού C#, χρησιμοποιώντας την παιχνιδομηχανή της Unity. Επίσης για τη δημιουργία των Animations, χρησιμοποιήθηκε το Mixamo. Το παιχνίδι φτιάχτηκε εξ ολοκλήρου από την αρχή, εστιάζοντας περισσότερο στο γραφιστικό-σχεδιαστικό κομμάτι του, το οποίο θα αναλυθεί παρακάτω.

Στόχος του παιχνιδιού, είναι η επιτυχής ολοκλήρωση όλων των επιπέδων για την τελική νίκη. Σε κάθε επίπεδο η ομάδα του χρήστη, θα έρχεται αντιμέτωπη με διάφορους εχθρούς. Ο κάθε παίκτης της, έχει διαθέσιμες κάποιες ενέργειες, καθ’ όλη τη διάρκεια της μάχης, τις οποίες μπορεί να χρησιμοποιεί όταν έρθει η σειρά του. Οι ενέργειες αυτές μπορεί να έχουν να κάνουν είτε με επιθέσεις προς τους αντιπάλους-εχθρούς, είτε με τη θεράπευση του ίδιου του παίκτη ή συμπαίκτη του. Παρόμοιες ενέργειες έχουν και οι εχθροί, οι οποίοι παίρνουν τη σειρά τους μόλις δράσει και ο τελευταίος παίκτης της ομάδας του χρήστη. Βασικός στόχος λοιπόν, είναι να χρησιμοποιήσει ο χρήστης την στρατηγική του σωστά, επιλέγοντας τις σωστές κινήσεις της ομάδας του, ώστε να επέλθει νίκη.

2. Έννοιες και Τεχνολογίες

2.1 Τι είναι η Ανάπτυξη Παιχνιδιών (Game Development)

Το Game Development, είναι η διαδικασία δημιουργίας παιχνιδιών σε ποικίλες πλατφόρμες. Ξεκινάει από την ιδέα και καταλήγει στην κυκλοφορία ενός παιχνιδιού, περνώντας τα στάδια του σχεδιασμού, προγραμματισμού, δοκιμής και παραγωγής του. Η διαδικασία αυτή μπορεί να διαρκέσει από λίγες εβδομάδες, έως και αρκετά χρόνια, γεγονός που εξαρτάται από το είδος του παιχνιδιού, την πολυπλοκότητα του καθώς και από το πόσα άτομα απαρτίζουν την ομάδα ανάπτυξης.[1]

Οι ομάδες ανάπτυξης παιχνιδιών, αποτελούνται συνήθως, από προγραμματιστές, σχεδιαστές, συγγραφείς και καλλιτέχνες. Το μέγεθος και η σύνθεση μιας τέτοιας ομάδας, θα ποικίλλει ανάλογα με το εκάστοτε παιχνίδι. Αξίζει να σημειωθεί πως αρκετά παιχνίδια έχουν αναπτυχθεί από ένα άτομο ή πολύ μικρές ομάδες, τους λεγόμενους Indie Developers. Οι Indie Developers, στην ουσία, είναι αυτοχρηματοδοτούμενα άτομα και δεν έχουν την υποστήριξη ενός εκδότη ή μιας εταιρείας. Συνήθως διαθέτουν τα παιχνίδια τους σε πλατφόρμες όπως το Play/Apple Store, Steam και το Itch.io.

2.2 Τι είναι η Σχεδίαση Παιχνιδιών (Game Design)

Το Game Design, είναι η διαδικασία σχεδιασμού παιχνιδιών, όπου περιλαμβάνει την ανάπτυξη μίας ιδέας, τη δημιουργία λεπτομερών προδιαγραφών για τους μηχανισμούς, τον οπτικό σχεδιασμό, τον ηχητικό σχεδιασμό, καθώς και τον προγραμματισμό τους. Ο σχεδιαστής (Game Designer), είναι ο υπεύθυνος, για την συνολική ισορροπία και τη ροή του παιχνιδιού.

Ο σκοπός των Game Designers, είναι να δημιουργούν διασκεδαστικές, διαδραστικές και ελκυστικές εμπειρίες για τους χρήστες. Πρέπει να είναι σε θέση να αναλύουν την απόδοση του παιχνιδιού και να κάνουν προσαρμογές, όπου χρειάζεται, για να κρατούν τους χρήστες αφοσιωμένους και να διασφαλίζουν ότι το παιχνίδι εκπληρώνει τις προσδοκίες και τους στόχους τους.

2.2.1 Βασικές Αρχές Σχεδίασης Παιχνιδιών

Όταν πρόκειται για σχεδίαση παιχνιδιών, είναι πολύ δύσκολο να οριστούν κανόνες και οδηγίες που να ισχύουν για όλα τα παιχνίδια, διότι τα παιχνίδια είναι διαφορετικά μεταξύ τους. Χρειάζεται μία βάση λοιπόν, οπότε παρακάτω περιγράφονται 7 βασικές αρχές για τη σχεδίαση παιχνιδιών. [5]

- Στόχος. Πρέπει να προσδιοριστεί ποιος είναι ο στόχος του παιχνιδιού, ώστε να ξέρει και ο χρήστης που θέλει να φτάσει.
- Προκλήσεις. Πολύ σημαντικό είναι το επίπεδο δυσκολίας του παιχνιδιού, μιας και θα κρατήσει τον χρήστη αφοσιωμένο και παρακινημένο στο να συνεχίσει να παίζει.
- Πρόοδος. Ο χρήστης πρέπει να έχει μία αίσθηση προόδου, επιτρέποντας του να ξεκλειδώσει δεξιότητες, αντικείμενα και επίπεδα, όσο προχωράει στο παιχνίδι.
- Προσβασιμότητα. Το παιχνίδι πρέπει να διατηρείται κατανοητό και απλό, ούτως ώστε οι χρήστες να μπορούν να το απολαύσουν ανεξαρτήτως εμπειρίας.

- Ιστορία. Συνίσταται να υπάρχει και μία ιστορία πίσω από το παιχνίδι, όπου μέσω μιας αφήγησης, δίνεται ένα βασικό πλαίσιο για τις ενέργειες του χρήστη.
- Διασκέδαση. Πρέπει το παιχνίδι να παραμένει ευχάριστο, διασκεδαστικό αλλά ταυτόχρονα με μία αίσθηση πρόκλησης.
- Διάρκεια. Σημαντικό είναι το παιχνίδι να έχει αρκετό περιεχόμενο και αξία επανάληψης, ώστε ο παίκτης να παραμείνει αφοσιωμένος για μεγάλο χρονικό διάστημα.

Λαμβάνοντας υπόψη τα παραπάνω, η σχεδίαση των παιχνιδιών είναι ένας αρκετά απαιτητικός τομέας, όπου η διατήρηση της ισορροπίας είναι βασική προϋπόθεση για την επιτυχημένη δημιουργία τους.

2.3 Τι είναι Turn-Based RPG

Τα Turn-Based Role Playing Games, είναι ένας τύπος παιχνιδιού ρόλων, όπου οι παίκτες εναλλάσσονται, όταν αλληλοεπιδρούν με το παιχνίδι. Περιλαμβάνει χαρακτήρες που ελέγχει ο χρήστης και τις ενέργειες τους με άλλους χαρακτήρες του παιχνιδιού. Συχνά διαδραματίζονται σε κόσμους επιστημονικής φαντασίας και περιέχουν μάχες και συγκρούσεις που πρέπει να επιλύσει ο χρήστης. Στόχος είναι η ολοκλήρωση μιας αποστολής ή το να φτάσουν οι χαρακτήρες σε ένα ορισμένο επίπεδο (level).

Μερικά παραδείγματα Turn-Based RPGs, είναι τα Final Fantasy, Dragon Quest, Darkest Dungeon, Pokémon και πολλά ακόμη.

Εικόνα 1: Final Fantasy IV – Σύστημα Μάχης

Εικόνα 2: Dragon Quest – Σύστημα Μάχης

Εικόνα 3: Darkest Dungeon – Σύστημα Μάχης

Εικόνα 4: Pokémon, Let's Go Pikachu – Σύστημα Μάχης

2.4 Τι είναι μια Παιχνιδομηχανή (Game Engine)

Ένα Game Engine, είναι μία πλατφόρμα λογισμικού, η οποία παρέχει τα εργαλεία και την υποδομή που χρειάζεται για τη δημιουργία βιντεοπαιχνιδιών. Αποτελούνται συνήθως από μια σειρά διαφορετικών στοιχείων, όπως το rendering, η γλώσσα σεναρίων, η μηχανή φυσική, η τεχνητή νοημοσύνη, η μηχανή ήχου, τα κινούμενα σχέδια, και η δικτύωση. [1]

Το Rendering, πρόκειται για τη διαδικασία δημιουργίας μιας δισδιάστατης (2D) ή τρισδιάστατης (3D) εικόνας. Χρησιμοποιείται για την παροχή γραφικών, όπως φωτισμό και υφές.

Η Γλώσσα Σεναρίων (Scripting Language), είναι μία γλώσσα προγραμματισμού που χρησιμοποιείται για τη δημιουργία της λογικής του παιχνιδιού, τον έλεγχο των αντικειμένων του και την αλληλεπίδραση με το περιβάλλον του παιχνιδιού. Μερικά παραδείγματα Scripting Languages, είναι οι C#, C++, JavaScript, Python και άλλες.

Η Μηχανή Φυσικής (Physics Engine), είναι ο κώδικας που στην ουσία προσομοιώνει τις φυσικές ιδιότητες του περιβάλλοντος του παιχνιδιού, όπως η βαρύτητα και η σύγκρουση. Αυτό επιτρέπει στα αντικείμενα να αλληλοεπιδρούν μεταξύ τους με ρεαλιστικούς τρόπους.

Η Τεχνητή Νοημοσύνη (Artificial Intelligence – AI), είναι ο κώδικας που επιτρέπει σε χαρακτήρες που δεν είναι παίκτες (Non Player Characters – NPCs), να συμπεριφέρονται με αξιόπιστους τρόπους. Το AI χρησιμοποιείται για να κάνει τα NPCs, να αντιδρούν στις ενέργειες του παίκτη με βάσει την λογική που τους έχει δοθεί προγραμματιστικά, και έτσι να λαμβάνουν τις ανάλογες αποφάσεις.

Η Μηχανή Ήχου (Sound Engine), χειρίζεται τον ήχο και περιλαμβάνει τα ηχητικά εφέ και τη μουσική του παιχνιδιού.

Τα Κινούμενα σχέδια (Animations), είναι η διαδικασία δημιουργίας κίνησης σε ένα παιχνίδι. Χρησιμοποιείται για να ζωντανέψει χαρακτήρες, να δημιουργήσει ειδικά εφέ και σε ένα γενικό πλαίσιο, για να προσθέσει ρεαλισμό τον κόσμο του παιχνιδιού.

Η Δικτύωση (Networking), είναι ο κώδικας που επιτρέπει στους χρήστες να συνδέονται και να παίζουν μεταξύ τους μέσω δικτύου. Χρησιμοποιείται κυρίως σε παιχνίδια για πολλούς παίκτες (Multiplayer).

Υπάρχουν πολλές διαφορετικές Παιχνιδομηχανές διαθέσιμες σήμερα, η καθεμία με τις δικές της δυνατότητες. Από τις πιο δημοφιλείς, όμως είναι η Unity, η οποία χρησιμοποιήθηκε για τον σχεδιασμό και την υλοποίηση της παρούσας πτυχιακής εργασίας.

Εικόνα 5: Unity Logo

2.5 Τι είναι η Παιχνιδομηχανή Unity

Η Unity είναι μια Παιχνιδομηχανή που δημιουργήθηκε από την Unity Technologies και υποστηρίζει περισσότερες από 27 πλατφόρμες από την κυκλοφορία της το 2005. Χρησιμοποιείται για τη δημιουργία βιντεοπαιχνιδιών, προσομοιώσεων και άλλου τρισδιάστατου και δισδιάστατου διαδραστικού περιεχομένου, σε υπολογιστές, κινητές συσκευές, κονσόλες και άλλες προσθήκες. Αξίζει να σημειωθεί πως η Unity, χρησιμοποιείται επίσης σε κλάδους όπως η αρχιτεκτονική, ο κινηματογράφος, η αυτοκινητοβιομηχανία και η μηχανική. [2]

Η Unity, είναι γραμμένη σε γλώσσες C# και C++, και περιλαμβάνει μια ολοκληρωμένη σειρά εργαλείων για την ανάπτυξη ενός παιχνιδιού, κάποια από τα οποία εξηγήθηκαν παραπάνω, όπως το AI, και το Physics Engine. Έννοιες και μερικά ακόμη εργαλεία, τα οποία χρησιμοποιήθηκαν στην παρούσα πτυχιακή εργασία, εξηγούνται ενδελεχή παρακάτω.

2.5.1 Εργαλεία της Unity που χρησιμοποιήθηκαν

Το Unity Asset Store, είναι μία ηλεκτρονική αγορά, για μοντέλα, υπηρεσίες και εργαλεία, που σχετίζονται με τη Unity. Παρέχει στους προγραμματιστές, πρόσβαση σε μία τεράστια ποικιλία που περιλαμβάνει 2D και 3D μοντέλα, υφές (textures), ήχους, οπτικά εφέ (VFX), scripts και πολλούς άλλους τύπους περιεχομένου. Από το Unity Asset Store, αγοράστηκαν τα 3D πακέτα για τους χαρακτήρες και τις σκηνές και τα 2D πακέτα για το Graphic User Interface του παιχνιδιού της παρούσας πτυχιακής εργασίας. Επίσης οι ήχοι για τα Sound Effects και τη μουσική. Οπτικά εφέ για τις ενέργειες των παικτών (επιθέσεις, θεράπευση) και τέλος υφές και Skyboxes για τις ανάγκες της κάθε σκηνής. [3]

Ο Profiler, είναι ένα πάρα πολύ σημαντικό και ισχυρό εργαλείο που επιτρέπει στους προγραμματιστές να κατανοήσουν καλύτερα την απόδοση του παιχνιδιού τους. Μπορεί να χρησιμοποιηθεί για τον εντοπισμό διαφόρων προβλημάτων όπως διαρροές μνήμης, σημεία συμφόρησης στον κώδικα, και ανεπαρκής βελτιστοποίηση. Επίσης ο Profiler, βοηθάει στην συγκριτική αξιολόγηση διαφορετικών εκδόσεων ενός παιχνιδιού και για τη σύγκριση της απόδοσης μεταξύ διαφορετικών πλατφορμών και συσκευών.

Το ProBuilder, είναι ένα πρόσθετο εργαλείο της Unity, το οποίο επιτρέπει στους χρήστες να δημιουργήσουν εύκολα και γρήγορα, 3D επίπεδα, απευθείας μέσα στο περιβάλλον της Unity. Συνδυάζει μια οπτική διεπαφή (Visual Interface) με εργαλεία επεξεργασίας για τον χειρισμό αντικειμένων, συμπεριλαμβανομένου του χρωματισμού κορυφών (Vertex Coloring) και σπάσιμο αντικειμένων (Object Snapping). Πολύ σημαντικό εργαλείο του ProBuilder, είναι και το UV Mapping, το οποίο παρέχει έναν τρόπο χαρτογράφησης των συντεταγμένων ενός τρισδιάστατου αντικειμένου για τη δημιουργία χαρτών υφής και φωτός.

Το Particle System, είναι ένα χαρακτηριστικό της Unity, που χρησιμοποιείται για τη δημιουργία εφέ, όπως φωτιά, νερό, καπνό και άλλα. Οι προγραμματιστές μπορούν να χειραγωγήσουν τις παραμέτρους που δίνει το σύστημα αυτό, όπως το μέγεθος των σωματιδίων, το χρώμα, η ποσότητα και η ταχύτητα τους, για να δημιουργήσουν τα κατάλληλα οπτικά εφέ που χρειάζονται στην εκάστοτε περίπτωση.

Στην παρούσα πτυχιακή εργασία, θα επεξηγηθεί στα επόμενα κεφάλαια η χρήση των παραπάνω εργαλείων και τα αποτελέσματά τους.

2.5.2 Game Objects

Τα Game Objects είναι τα πιο σημαντικά δομικά στοιχεία της Unity. Αντιπροσωπεύουν όλα τα στοιχεία που συνθέτουν ένα παιχνίδι, όπως χαρακτήρες, αντικείμενα, σκηνικά, κάμερες, φώτα, ήχους και πολλά άλλα. Κάθε Game Object έχει στοιχεία (Components) που χρησιμοποιούνται για να ορίσουν την εμφάνιση, την συμπεριφορά και τις ιδιότητες του αντικειμένου. Αυτά τα στοιχεία μπορούν να προστεθούν, να αφαιρεθούν και να επεξεργαστούν ανάλογα με το είδος τους. Στην ουσία όλα τα αντικείμενα στο παιχνίδι είναι Game Objects. [3]

2.5.3 Prefabs

Τα Prefabs, είναι επαναχρησιμοποιούμενα Game Objects, που αποτελούνται από στοιχεία όπως μοντέλα, αποσπάσματα ήχου, scripts και άλλα, και μπορούν να χρησιμοποιηθούν για τη γρήγορη δημιουργία πολλών αντιγράφων, μιας και θα μοιράζονται τις ίδιες ιδιότητες των εκάστοτε προτύπων. [3]

2.5.4 Scriptable Objects

Ένα Scriptable Object, είναι ένας τύπος στοιχείου της Unity, που επιτρέπει την αποθήκευση δεδομένων και την πολύ εύκολη τροποποίηση τους από τους προγραμματιστές. Μπορούν να αποθηκευτούν και να επαναχρησιμοποιηθούν, χωρίς να απαιτείται παρέμβαση ξανά από scripts – κώδικα. Η διαδικασία αυτή έχει κάνει τα Scriptable Objects, τον πιο διαδεδομένο και γρήγορο τρόπο αποθήκευσης δεδομένων και επεξεργασίας των τιμών τους. [3]

2.5.5 Μοτίβα Προγραμματισμού (Programming Patterns)

Programming Patterns ονομάζονται οι λύσεις σε κοινά προβλήματα που αφορούν την ανάπτυξη κώδικα. Τα Patterns αυτά είναι επαναχρησιμοποιήσιμα, αλλά δεν είναι συγκεκριμένοι αλγόριθμοι. Παρέχουν μια γενική προσέγγιση για την επίλυση ενός προβλήματος καθώς επίσης κάνουν τον κώδικα πιο ανεξάρτητο και αρθρωτό (modular). Είναι πολύ σημαντικό οι κώδικες, είτε αφορούν παιχνίδια, είτε άλλες εφαρμογές, να χωρίζονται σε διαφορετικά κομμάτια τα οποία να μπορούν να χρησιμοποιηθούν και αργότερα στην ανάπτυξη ενός άλλου λογισμικού. Τέτοια Programming Patterns παρουσιάζονται παρακάτω. [4]

Το Singleton Pattern, διασφαλίζει ότι δημιουργείται μόνο ένα αντικείμενο μίας συγκεκριμένης κλάσης, παρέχοντας ένα μόνο σημείο πρόσβασης σε αυτό.

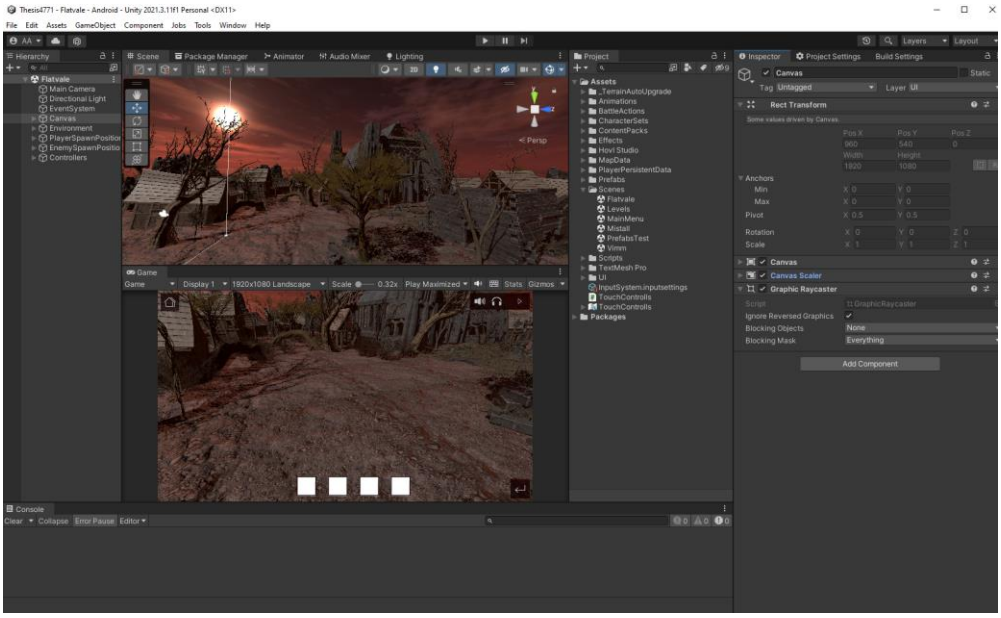
Το Factory Pattern, παρέχει έναν οργανωμένο και δομημένο τρόπο δημιουργίας αντικειμένων, χωρίς τον ακριβή προσδιορισμό τύπου του αντικειμένου.

Το Observer Pattern, επιτρέπει στα αντικείμενα να μπορούν να εγγραφούν για να ειδοποιηθούν όταν συμβαίνει ένα γεγονός.

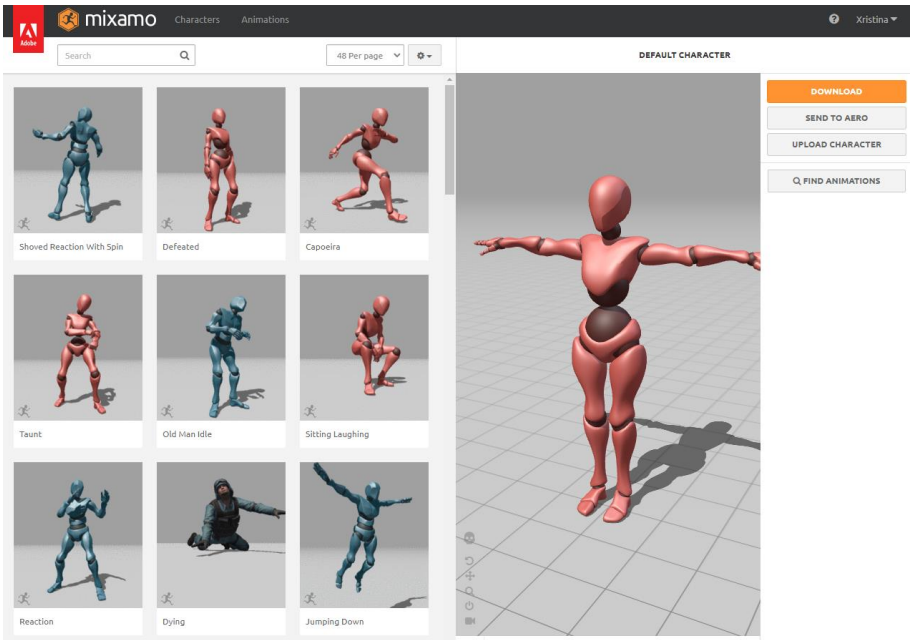
Το Model-View-Controller (MVC) Pattern, χρησιμοποιείται όταν υπάρχει ανάγκη διαχωρισμού των δεδομένων και της λογικής μιας εφαρμογής από τη διεπαφή χρήστη της.

2.6 Mixamo

Το Mixamo, είναι μια ηλεκτρονική υπηρεσία 3D κίνησης χαρακτήρων, η οποία παρέχει στους χρήστες έναν γρήγορο και εύκολο τρόπο δημιουργίας animation υψηλής ποιότητας. Με το Mixamo, οι χρήστες μπορούν να προσαρμόσουν τους χαρακτήρες, να τους ζωντανέψουν με μία βιβλιοθήκη κινήσεων και να τους εξάγουν για δική τους χρήση. Η υπηρεσία αυτή χρησιμοποιήθηκε με τρόπο που θα εξηγηθεί παρακάτω για χαρακτήρες, οι οποίοι δεν είχαν τα απαιτούμενα animations που χρειάζονταν για το παιχνίδι. [6]



Εικόνα 6: Περιβάλλον της Unity



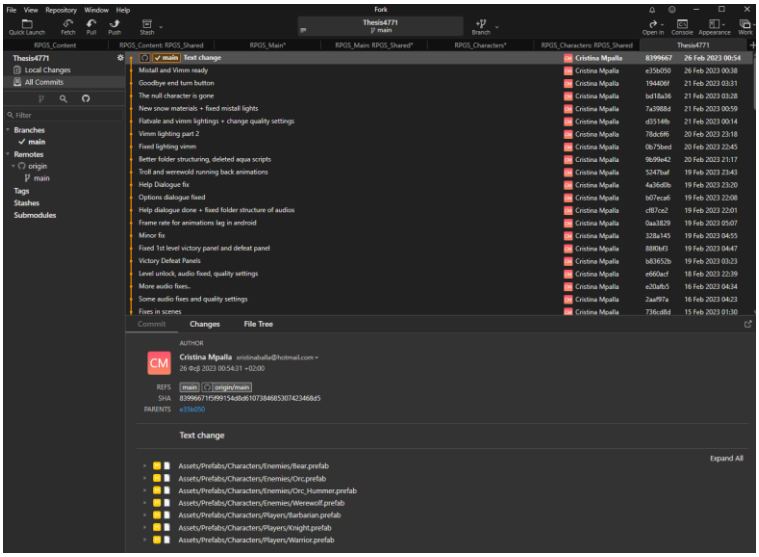
Εικόνα 7: Περιβάλλον του Mixamo

3. Μεθοδολογία Υλοποίησης

3.1 Ροή Εργασίας (Workflow)

Το σημαντικότερο που χρειάζεται η δημιουργία ενός παιχνιδιού, ούτως ώστε να υλοποιηθεί σωστά και μέσα στα απαιτούμενα χρονικά πλαίσια, είναι μια καλή οργάνωση από μέρους των δημιουργών. Ιδιαιτέρως αν ο δημιουργός του είναι ένα άτομο, όπου πρέπει να αναλάβει το κομμάτι του Design αλλά και του Development, τότε η οργάνωση, είναι το Άλφα και το Ωμέγα.

Για την οργάνωση της παρούσας πτυχιακής εργασίας, χρησιμοποιήθηκε το πρόγραμμα Fork. Είναι ένα δωρεάν και ανοιχτού κώδικα πρόγραμμα Git Client. Προσφέρει μια διεπαφή για τους χρήστες, όπου μπορούν να διαχειριστούν εύκολα τα Git Repositories πηγαίου κώδικα. Αυτά τα Repositories, συμπεριλαμβάνουν όλες τις αλλαγές που έγιναν στον κώδικα, στάδια και δεσμεύσεις αλλαγών (Stage and Commit Changes), διακλαδώσεις για διαχωρισμό των σταδίων και άλλα πολλά χαρακτηριστικά. Με απλά λόγια, επιτρέπει στους χρήστες να οργανώνουν τον κώδικα και τη δουλειά τους, που έχουν συνδέσει μέσω GitHub, και να δουλεύουν πιο ασφαλείς, εννοώντας πως αν συμβεί κάποιο λάθος, μπορεί πολύ εύκολα να διαγραφούν οι αλλαγές που το προκάλεσαν χωρίς κάποια επιρροή στο βασικό πρότζεκτ. Αξίζει να σημειωθεί ότι το Fork, χρησιμοποιείται περισσότερο μεταξύ ομάδων που δουλεύουν μαζί, ώστε να συνδέεται το κάθε μέρος της δουλειάς τους σε ένα, εύκολα, οργανωμένα και γρήγορα.



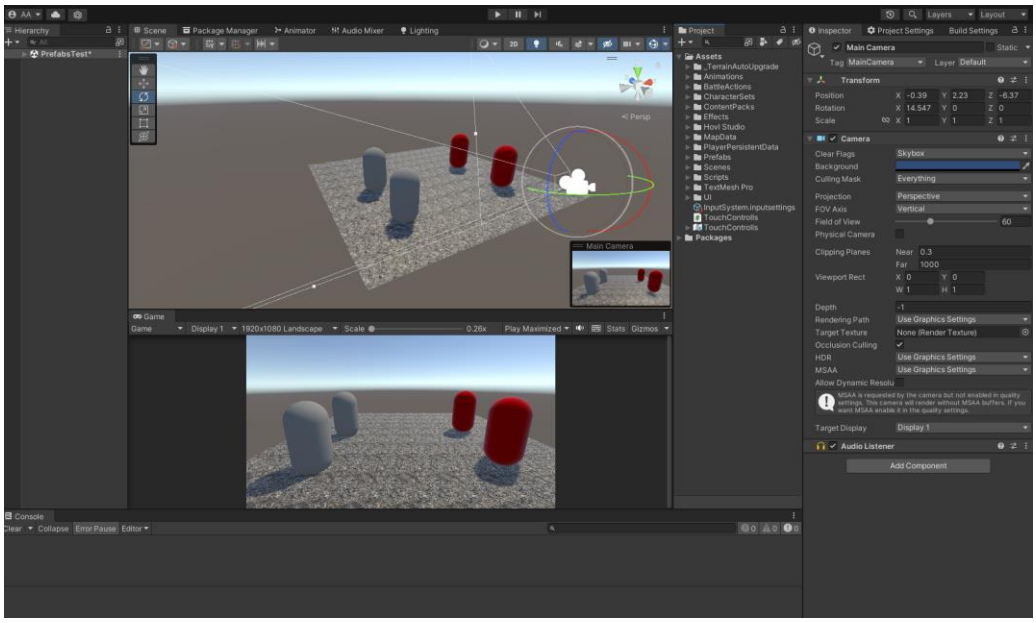
Εικόνα 8: Περιβάλλον του Fork

Για την ανάπτυξη της εργασίας υπήρχαν αρκετά στάδια, ούτως ώστε να ολοκληρωθεί. Στάδια όπως, η καταγραφή της ιδέας, το prototype παιχνίδι, η δημιουργία του βασικού παιχνιδιού, το γραφικό περιβάλλον χρήστη (Graphic User Interface – GUI), το τι μοντέλα χρειάζονταν για τις σκηνές, ήχοι και επιπλέον εφέ, και τέλος η σύνδεση του prototype παιχνιδιού στο βασικό, μαζί με τον εντοπισμό σφαλμάτων και τη βελτιστοποίηση.

Αρχικό στάδιο, όπως και σε όλα τα παιχνίδια, είναι η καταγραφή της ιδέας. Εγώ κατέγραψα την ιδέα μου για το παιχνίδι σε ένα χαρτί, όπου και σιγά σιγά έγραφα τους κανόνες και το σύστημα μάχης. Μόλις είχα όλο το gameplay στα χέρια μου, σκέφτηκα μία

βασική ιστορία για να συνδέεται με το στυλ του παιχνιδιού, από πού ξεκινάει ο παίκτης, πού πρέπει να τελειώσει και γιατί, ούτως ώστε να έχει ένα κίνητρο να συνεχίσει.

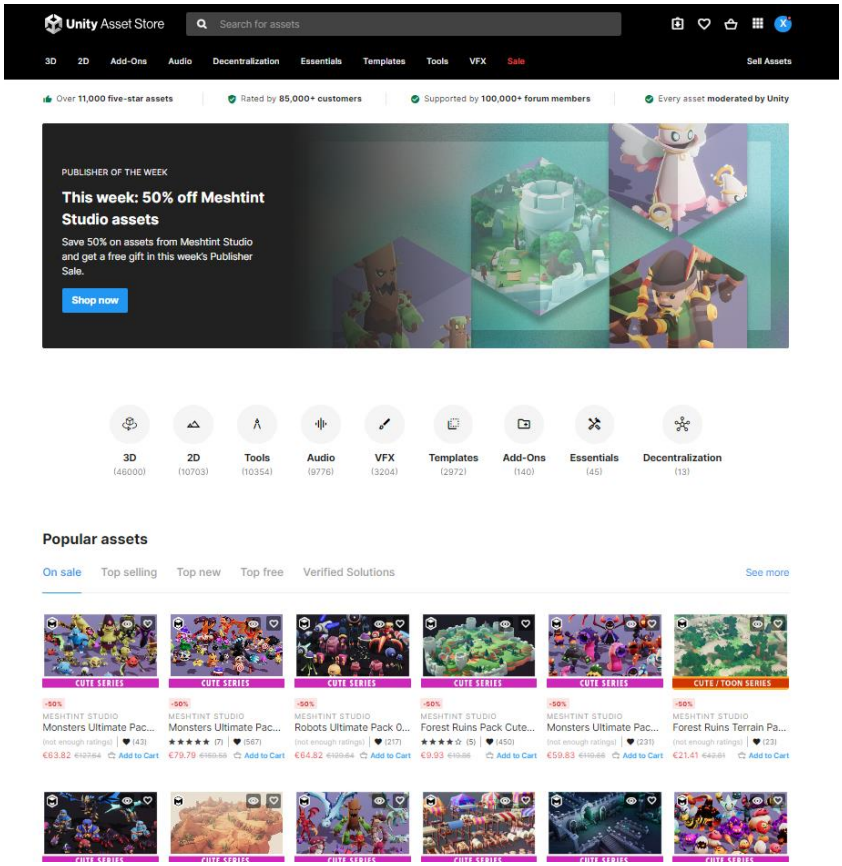
Επειδή το παιχνίδι αυτό, είναι και το πρώτο που φτιάχνω εξ ολοκλήρου από την αρχή, δημιούργησα ένα prototype παιχνίδι. Ένα prototype, είναι στην ουσία το παιχνίδι στην πολύ απλή έκδοση του, το οποίο περιέχει μόνο τους βασικούς μηχανισμούς, χωρίς χαρακτήρες και εφέ. Εκεί χτίστηκε το μεγαλύτερο προγραμματιστικό κομμάτι της εργασίας. Ο λόγος δημιουργίας του, ήταν ο εντοπισμός λαθών στον βασικό κώδικα του παιχνιδιού, χωρίς να επενδυθεί χρόνος σε σκηνές, μοντέλα, φωτισμούς και άλλα γραφικά στοιχεία. Συνεπώς, όταν οι σκηνές, οι χαρακτήρες και το γραφικό περιβάλλον ήταν έτοιμα, τότε έγινε η ένωση του prototype στο βασικό πρότζεκτ.



Εικόνα 9: Το Prototype Πρότζεκτ

Στη συνέχεια, προχώρησα στη δημιουργία του βασικού πρότζεκτ, ούτως ώστε να φτιαχτεί η διεπαφή του χρήστη, συμπεριλαμβανομένων των επιπέδων του παιχνιδιού. Αφού σχεδιάστηκε η ροή του μενού, η οποία θα αναλυθεί παρακάτω, σιγά σιγά δημιουργήθηκαν και τα επίπεδα του παιχνιδιού. Σε αυτό το σημείο ανέτρεξα στην ηλεκτρονική αγορά της Unity για τις αγορές των απαιτούμενων μοντέλων και υπηρεσιών.

Από το Unity Asset Store, προμηθεύτηκα όλα τα 3D πακέτα που χρειάστηκαν για τη δημιουργία των σκηνών, τους 3D χαρακτήρες για το παιχνίδι και τα 2D στοιχεία για τη διεπαφή του χρήστη. Επίσης κάποια Particle Systems για τα εφέ, ήχους συστήματος και μουσική και τέλος Skyboxes και υφές, όπου υπήρχε έλλειψη. Αναλυτικά τα μοντέλα που αγοράστηκαν βρίσκονται στη βιβλιογραφική σημείωση στο τέλος της αναφοράς.



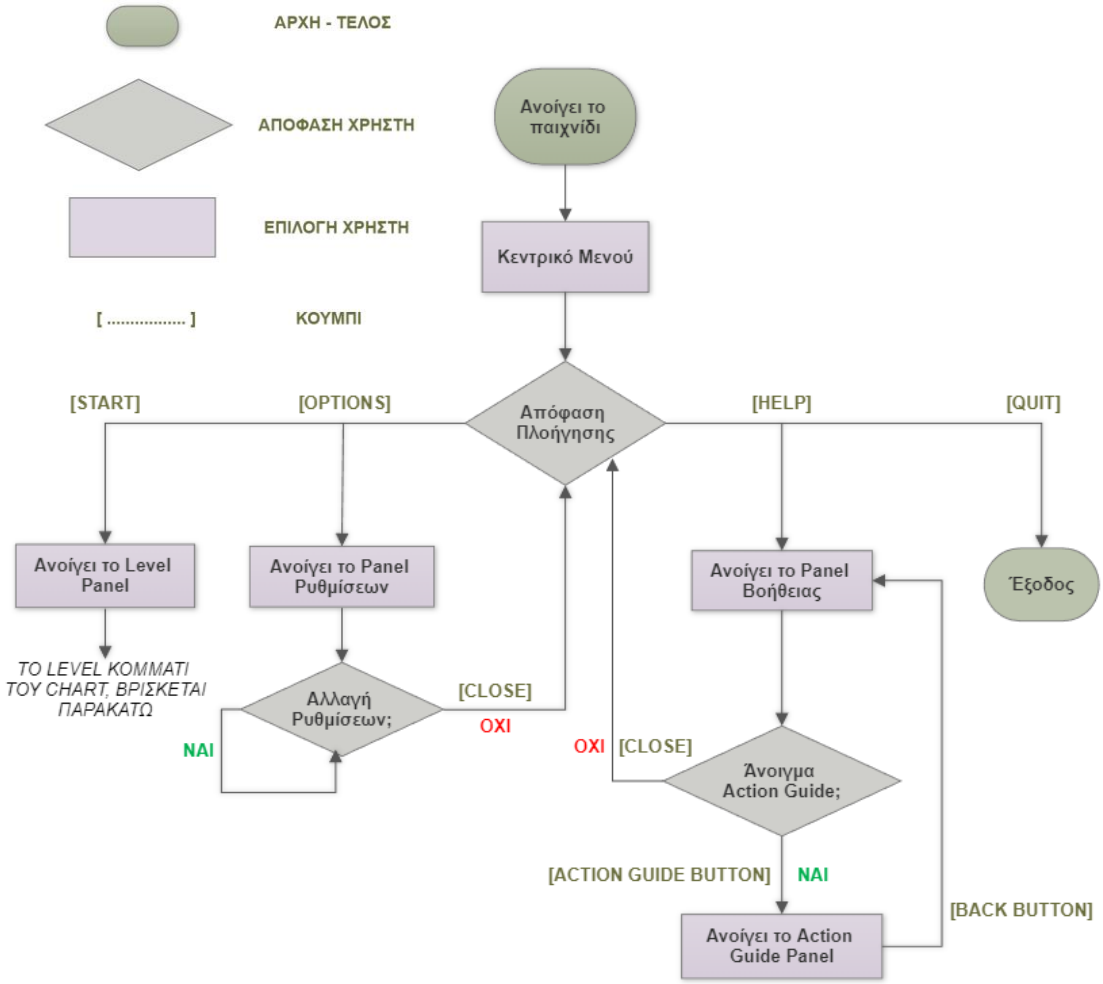
Εικόνα 10: Η ηλεκτρονική αγορά Unity Asset Store

Πλέον με τις τελικές σκηνές έτοιμες, και αφού έχω καταλήξει στο ποια μοντέλα αναλογούν στους χαρακτήρες του παιχνιδιού, καθώς και το τελικό γραφικό περιβάλλον για τη διεπαφή χρήστη, ήρθε η στιγμή για να συνδεθεί το prototype πρότζεκτ με το βασικό της τελικής εργασίας. Η μεταφορά όλων των scripts και των μηχανισμών, απαιτούσε ιδιαίτερη προσοχή και λεπτομέρεια, ούτως ώστε να μη χαθεί κανένα αρχείο.

Τέλος για την ολοκλήρωση της εργασίας, πραγματοποιήθηκε μια διαδικασία αποσφαλμάτωσης και βελτιστοποίησης των μηχανισμών του παιχνιδιού, ούτως ώστε να διαθέτει τη μεγαλύτερη δυνατή απόδοση. Η διαδικασία αυτή θα αναλυθεί στα παρακάτω κεφάλαια.

3.2 Διαγράμματα Ροής Διεπαφής

Η παρούσα πτυχιακή εργασία, βασίζεται σε σενάρια που μπορεί να ακολουθήσει ο χρήστης, για τα οποία έχουν χρησιμοποιηθεί διαγράμματα ροής. Τα παρακάτω διαγράμματα απεικονίζουν το πώς ο χρήστης μπορεί να αλληλοεπιδράσει με τις διεπαφές και τις ακολουθίες του παιχνιδιού κατά τη διάρκεια πλοήγησης του. Τα διαγράμματα αυτά δημιουργήθηκαν με τη χρήση του ηλεκτρονικού εργαλείου SmartDraw [10].



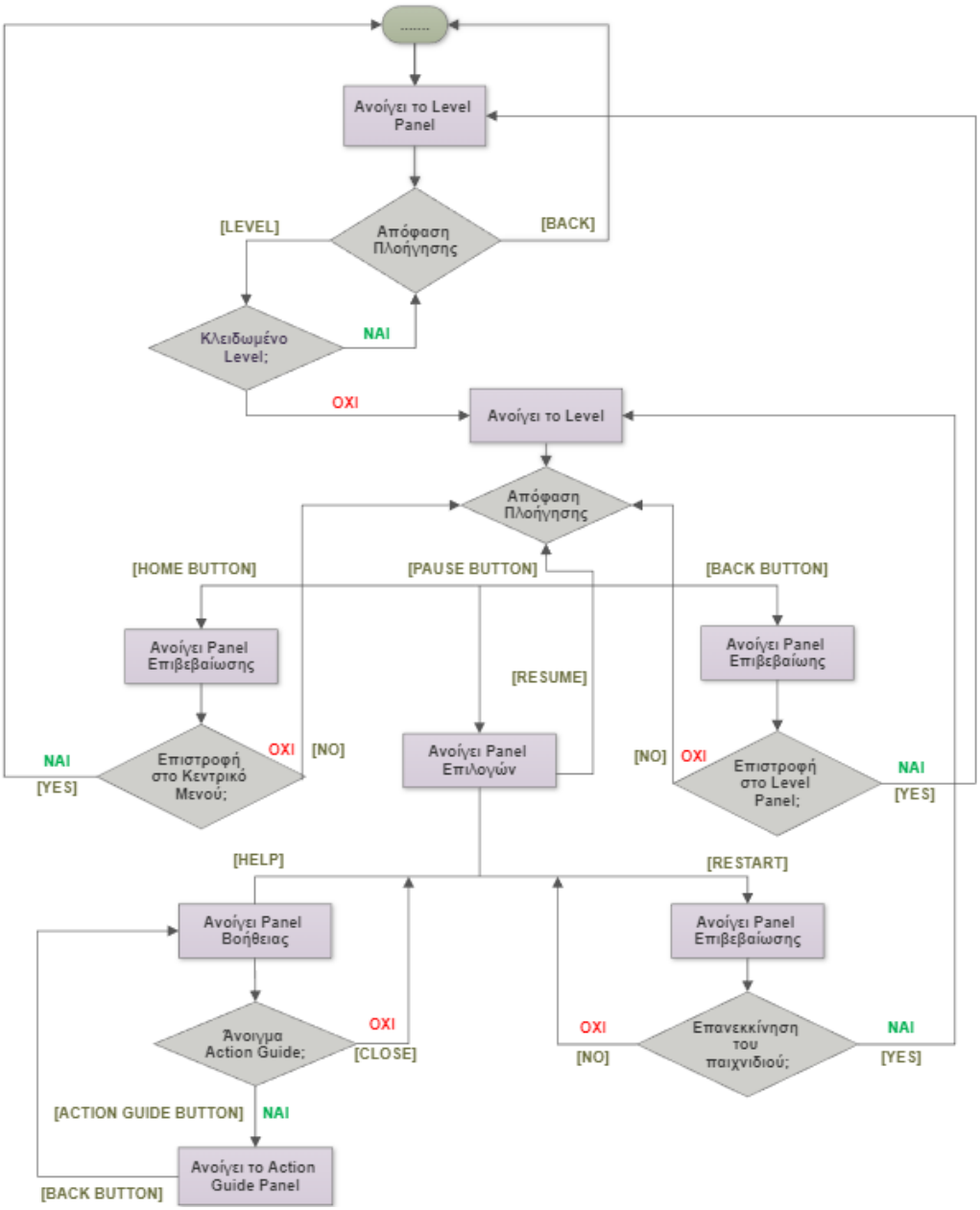
Εικόνα 11: Διάγραμμα Ροής για την πλοήγηση του χρήστη στη διεπαφή (Μέρος Α)

Ο χρήστης ανοίγει το παιχνίδι, και μόνο την πρώτη φορά της εκκίνησης, εμφανίζεται η εισαγωγική σκηνή για την αφήγηση της ιστορίας. Μόλις τελειώσει η εισαγωγή λοιπόν, ανοίγει το κεντρικό μενού. Στο κεντρικό μενού, ο χρήστης μπορεί να πλοηγηθεί στο παιχνίδι με τέσσερα κουμπιά, την εκκίνηση (START), τις επιλογές (OPTIONS), την βοήθεια (HELP) και την έξοδο (QUIT).

Η εκκίνηση θα αναλυθεί παρακάτω. Όσον αφορά το κουμπί επιλογών, θα ανοίξει στον χρήστη ένα panel, στο οποίο, αν επιθυμεί, μπορεί να αλλάξει την ποιότητα του παιχνιδιού, καθώς και την ένταση των ήχων συστήματος και της μουσικής. Τέλος, με το κουμπί κλείσιμο (CLOSE), μπορεί να βρεθεί ξανά στο κεντρικό μενού.

Με το κουμπί της βοήθειας, ανοίγει ένα panel με κανόνες και tips για την στρατηγική που μπορεί ο χρήστης να ακολουθήσει στο παιχνίδι. Επιπλέον εκεί, βρίσκεται και ένα κουμπί-οδηγός, που εμφανίζει ποιες κινήσεις έχει διαθέσιμες, ο κάθε χαρακτήρας της ομάδας του, καθώς επίσης και την περιγραφή τους. Όπως και παραπάνω μπορεί να πλοηγηθεί ξανά στο κεντρικό μενού με το κουμπί κλείσιμο (CLOSE) ή πίσω (BACK), ανάλογα ποιο panel είναι ανοιχτό σε εκείνο το σημείο.

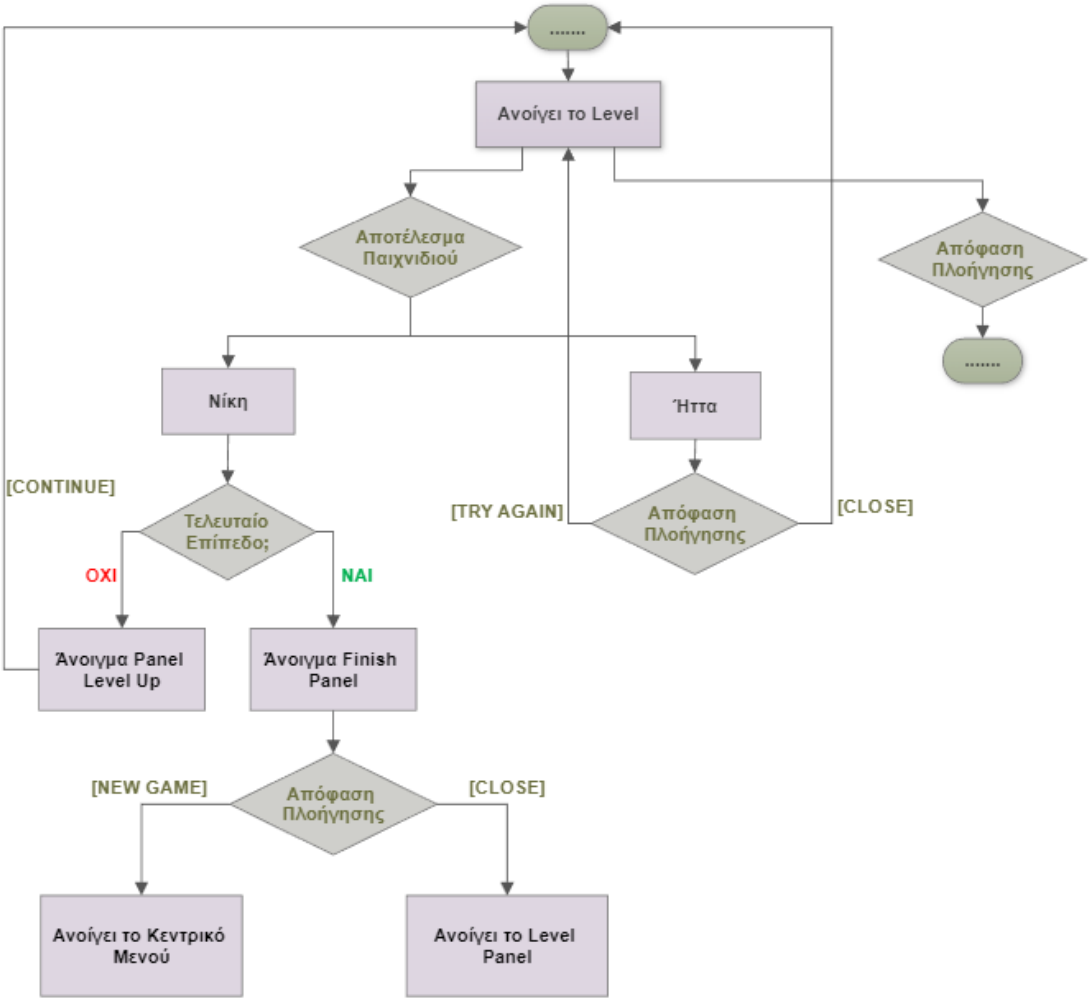
Τέλος για το κεντρικό μενού υπάρχει και η έξοδος (QUIT), για την οποία ερωτάται ο χρήστης αν είναι σίγουρος ή όχι, για το κλείσιμο του παιχνιδιού.



Εικόνα 12: Διάγραμμα Ροής για την πλοήγηση του χρήστη στη διεπαφή (Μέρος Β)

Όταν ο χρήστης πατήσει το κουμπί εκκίνησης, θα του εμφανιστεί ένα panel, με τα διαθέσιμα επίπεδα που μπορεί να παίξει. Από προεπιλογή είναι ξεκλειδωμένο μόνο το πρώτο επίπεδο, και όταν το ολοκληρώσει με επιτυχία ξεκλειδώνεται το επόμενο. Αυτό γίνεται μέχρι να φτάσει στο τελικό. Αφού, λοιπόν, ο χρήστης ανοίξει ένα επίπεδο, τα διαθέσιμα User Interface κουμπιά που είναι διαθέσιμα, είναι το κουμπί για το κεντρικό μενού (HOME BUTTON), το κουμπί πίσω για τα επίπεδα (BACK BUTTON) και το κουμπί της παύσης (PAUSE).

Αν ο χρήστης επιλέξει είτε να πάει πίσω στα επίπεδα, είτε στο κεντρικό μενού, εμφανίζεται ένα ανάλογο panel επιβεβαίωσης για την κάθε περίπτωση. Εάν όμως πατήσει την παύση, τότε έχει τρεις επιλογές πλοήγησης. Μπορεί, να πατήσει την βοήθεια (HELP), που αναφέραμε παραπάνω, η οποία βρίσκεται σαν διαθέσιμο κουμπί και εκτός κεντρικού μενού. Μπορεί να κάνει επαναφορά (RESTART), και να αρχίσει το παιχνίδι από την αρχή, και επίσης να συνεχίσει το παιχνίδι από εκεί που βρισκόταν με το κουμπί (RESUME).



Εικόνα 13: Διάγραμμα Ροής για την πλοήγηση του χρήστη στη διεπαφή (Μέρος Γ)

Επιπλέον panels, εμφανίζονται στον χρήστη και ανάλογα με το αποτέλεσμα του κάθε επιπέδου. Αν υπάρξει νίκη, αλλά το επίπεδο δεν είναι το τελευταίο, τότε εμφανίζεται ένα panel με τα ανανεωμένα στατιστικά των χαρακτήρων της ομάδας του, καθώς και ένα κουμπί συνέχειας (CONTINUE), που θα τον οδηγήσει πίσω στα επίπεδα. Αν το επίπεδο είναι το τελευταίο, όταν ο χρήστης νικήσει, θα του εμφανιστούν συγχαρητήρια λόγια, μαζί με το κουμπί κλείσιμο (CLOSE) και το κουμπί για νέο παιχνίδι, όπου όλα τα δεδομένα και τα στατιστικά διαγράφονται και αρχίζουν από την αρχή, με κλειδωμένα τα level. Από την άλλη αν υπάρξει ήττα, ανεξαρτήτως επιπέδου, εμφανίζεται το panel της ήττας με το κουμπί για επαναπροσπάθεια (TRY AGAIN), όπου το επίπεδο φορτώνεται ξανά και το κουμπί κλείσιμο (CLOSE), όπου γυρνάει τον χρήστη πίσω στα επίπεδα.

4. Game Design

4.1 Διεπαφή Χρήστη (User Interface – UI)

Για τη δημιουργία της διεπαφής του χρήστη (UI), χρησιμοποιήθηκε ο Unity Editor. Ο συνδυασμός των εργαλείων και των χαρακτηριστικών του, όπως κουμπιά, εικόνες, καμβάδες, κείμενα και πολλά άλλα βοήθησαν στο να φτιαχτούν όλες οι σκηνές και τα μενού, σωστά και οργανωμένα. Η εμφάνιση και η αισθητική παίζουν πολύ σημαντικό ρόλο για το πόσο ελκυστικό μπορεί να γίνει ένα παιχνίδι. Παρακάτω θα αναλυθεί διεξοδικά η διαδικασία σχεδίασης όλων των στοιχείων διεπαφής μία προς μία.

4.1.1 Σκηνή Εισαγωγής

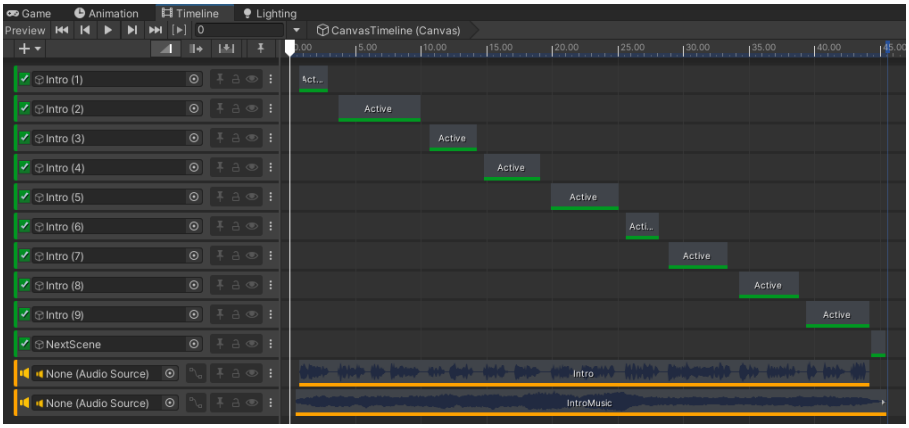
Μια εισαγωγική σκηνή, παρέχει έναν εξαιρετικό τρόπο για να μυηθεί ο χρήστης στο παιχνίδι. Δημιουργεί την ιστορία και την αισθητική του παιχνιδιού και βοηθά τους χρήστες να κατανοήσουν τους στόχους. Δίνει, επίσης, μία γεύση για τα γραφικά, τους ήχους και την ατμόσφαιρα του παιχνιδιού και κατά συνέπεια δημιουργεί προσμονή για τα επερχόμενα επίπεδα και το περιεχόμενο.

Για το παιχνίδι της παρούσας πτυχιακής, η σκηνή εισαγωγής, περιλαμβάνει απλά στοιχεία όπως, η περιγραφή της ιστορίας μέσω μιας αφήγησης κειμένου με υπότιτλους, συνοδευόμενη από μουσική υπόκρουση που ταιριάζει στην αισθητική.

Εικόνα 15: Σκηνής Εισαγωγής (Στιγμιότυπο 1)

Εικόνα 16: Σκηνής Εισαγωγής (Στιγμιότυπο 2)

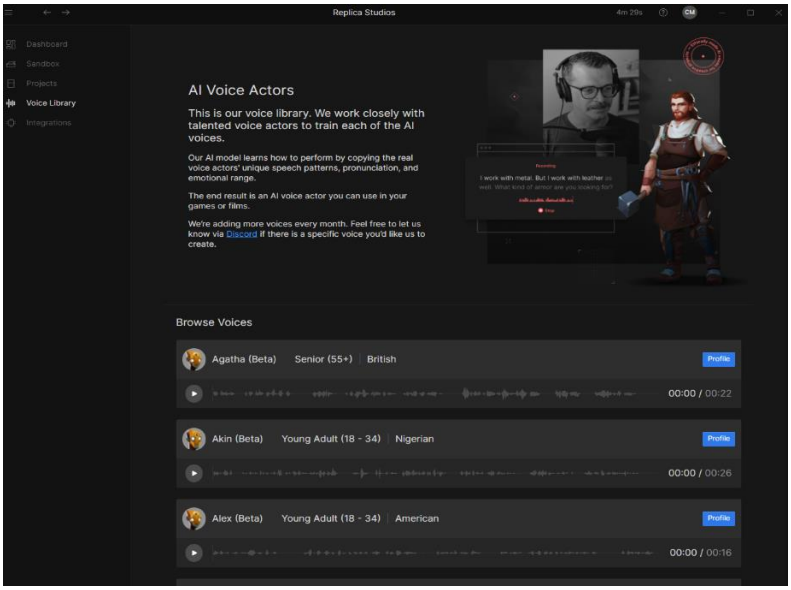
Για την διαδοχική εμφάνιση των υποτίτλων ανάλογα με την αφήγηση, χρησιμοποιήθηκε το εργαλείο Timeline του Unity Editor. Το Timeline, είναι ένα εργαλείο δημιουργίας κινηματογραφικού περιεχομένου και ακολουθιών παιχνιδιών, το οποίο βάζει σε μία χρονική γραμμή game objects, εφέ, ήχους, animations και πολλά άλλα. [7]



Εικόνα 17: Το Timeline της Σκηνής Εισαγωγής

Στα δεξιά παρατηρούνται τα κομμάτια των υποτίτλων, η επόμενη σκηνή που θα φορτωθεί με το πέρας της εισαγωγικής, καθώς και η αφήγηση μαζί με την μουσική. Στο χρονοδιάγραμμα κεντρικά, φαίνεται πως οι υπότιτλοι (Actives), έχουν τοποθετηθεί βάσει της αφήγησης (Intro) και της μουσικής (IntroMusic). Η διαδικασία αυτή ήταν αρκετά απαιτητική, αφού έπρεπε να μην υπάρχουν καθυστερήσεις και να ταιριάξει ο σωστός υπότιτλος στο αντίστοιχο αφηγηματικό κομμάτι, βλέποντας τα δευτερόλεπτα στο πάνω μέρος.

Όσον αφορά την αφήγηση, χρησιμοποιήθηκε το πρόγραμμα Replica. Είναι μία πλατφόρμα, όπου με τη βοήθεια της τεχνητής νοημοσύνης, επιτρέπει την παραγωγή φυσικών ήχων και αφηγήσεων. Στην ουσία έχει μία λίστα από ηθοποιούς φωνής, και ακούγοντας ένα δείγμα αφήγησης τους, επιλέγεις τον κατάλληλο για να αφηγηθεί το κείμενο που θα γράψεις. Μπορείς να διαλέξεις το είδος και το στυλ της αφήγησης, για ένα πιο ρεαλιστικό αποτέλεσμα, αναλόγως την ατμόσφαιρα που θέλεις να προσφέρεις.

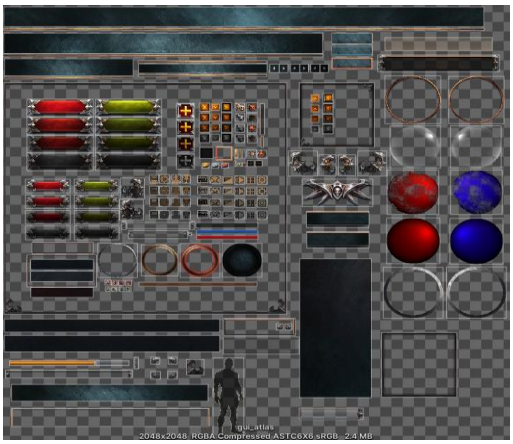


Εικόνα 18: Replica – Κείμενο σε Ομιλία

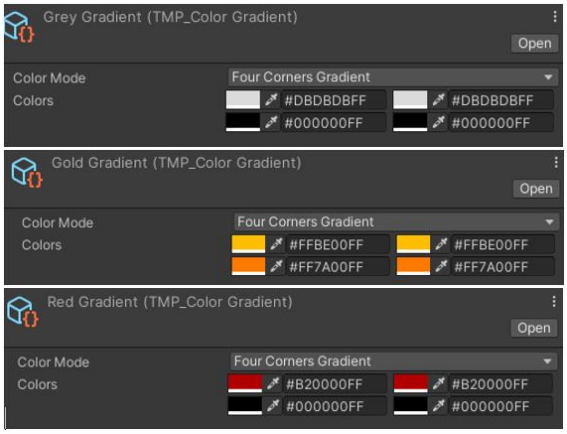
4.1.2 Σχεδίαση των Μενού

Για την σχεδίαση των μενού του παιχνιδιού, χρησιμοποιήθηκε ένα πακέτο της Unity [14], για τη διεπαφή, που περιλαμβάνει εικόνες για κουμπιά, backgrounds, input fields, περιγράμματα και άλλα. Επίσης, εικόνες για τα levels, βγήκαν μετά τη δημιουργία του κάθε επιπέδου, με σκοπό ο χρήστης να έχει μία ιδέα του πως είναι η ατμόσφαιρα της μάχης που πάει να ξεκινήσει. Επιπλέον εικονίδια για το μενού βοήθειας που αφορούν τις κινήσεις του εκάστοτε παίκτη, μπήκαν από άλλο πακέτο της Unity [21], που περιέχει μόνο τέτοια εικονίδια φαντασίας (βλ. Εικόνα 22).

Για την γραμματοσειρά, δημιουργήθηκαν Color Gradients by 4 Corners, ώστε να δώσουν μεγαλύτερη έμφαση στο κείμενο, καθώς και για να ενισχύσουν την ατμόσφαιρα του παιχνιδιού. Στην ουσία, είναι μια σταδιακή ανάμειξη μεταξύ τεσσάρων χρωμάτων, σε οπτικά διακριτές ενότητες. Πιο συγκεκριμένα το γκρι color gradient, χρησιμοποιήθηκε για την κεντρική γραμματοσειρά του παιχνιδιού (βλ. Εικόνα 20), το χρυσό gradient, για τα στατιστικά του Level Up και του panel νίκης (βλ. Εικόνα 34), ενώ το κόκκινο-μαύρο για το panel ήττας (βλ. Εικόνα 37).



Εικόνα 19: Dark Fantasy GUI



Εικόνα 20: Color Gradients των κειμένων

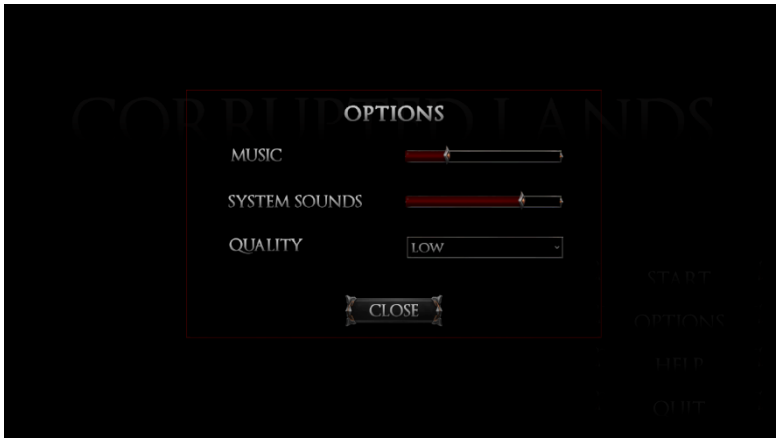
Παρακάτω φαίνονται τα panel όλων των μενού του παιχνιδιού, ξεκινώντας από το κεντρικό μενού, και περνώντας σταδιακά από όλα τα κουμπιά και τα panel που έρχονται αφού πατηθούν.

Εικόνα 21: Κεντρικό Μενού

Εικόνα 22: Μενού Βοήθειας – Κανόνες και Tips

Εικόνα 23: Μενού Βοήθειας – Οδηγίες Κινήσεων

Εικόνα 24: Επιβεβαίωση Εξόδου

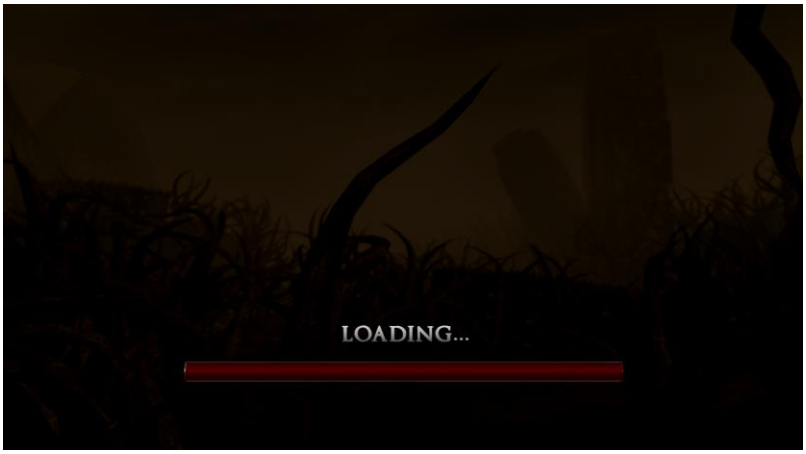


Εικόνα 25: Επιλογές Παιχνιδιού

Εμβαθύνοντας στην υλοποιήση των μενού, υπήρχε αρκετή επεξεργασία για το τελικό αποτέλεσμα που φαίνεται παραπάνω. Αρχικά να επισημάνουμε πως τα παραπάνω panel, εφάπτονται πάνω στο κεντρικό μενού, και εμφανίζονται με το πάτημα του ανάλογου κουμπιού. Το κάθε panel, έγινε με συνδυασμό δύο εικόνων. Η βασική, είναι μία μαύρη εικόνα για το background, με ενεργοποιημένη τη διαφάνεια σε ένα ελάχιστο ποσοστό, τόσο ώστε να φαίνεται αχνά στο φόντο το κεντρικό μενού. Η δεύτερη εικόνα έχει να κάνει με το εσωτερικό panel που κρατά τις πληροφορίες για κάθε μενού, είτε αυτό είναι κουμπιά, είτε απλό κείμενο.

Εκτός των κειμένων, άλλα στοιχεία που συντέλεσαν στη σχεδίαση αυτή είναι κουμπιά, dropdowns για την επιλογή ποιότητας του παιχνιδιού, και sliders για την ένταση της μουσικής και των ήχων συστήματος. Εδώ αξίζει να σημειωθεί ότι οι επιλογές στο μενού επιλογών αποθηκεύονται καθ'οόλη τη διάρκεια του παιχνιδιού.

Η μετάβαση στα επόμενα μενού και επίπεδα γίνεται με τη χρήση fade in και fade out animation, το οποίο θα εξηγηθεί εκτενέστερα παρακάτω. Για την μετάβαση αυτή επίσης χρησιμοποιείται μια ασύγχρονη τεχνική της φόρτωσης της σκηνής (Loading), η οποία βοηθάει στη μείωση χρόνου αναμονής των χρηστών μεταξύ των σκηνών. Η ασύγχρονη λειτουργία αυτή, στην ουσία, είναι μια διαδικασία της οποίας η εκτέλεση μπορεί να προχωρήσει ανεξάρτητα ή στο παρασκήνιο. Σε συνδυασμό με το fade in και fade out animation, η τεχνική φόρτωσης φαίνεται παρακάτω, στην εξήγηση των scripts (βλ. Εικόνα 72).



Εικόνα 26: Panel Φόρτωσης Σκηνής

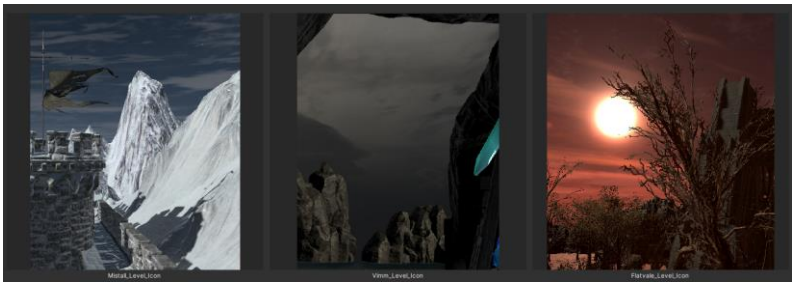
4.1.3 Σχεδίαση του Gameplay UI

Όταν φορτωθεί η επόμενη σκηνή μετά το πάτημα του κουμπιού εκκίνησης, εμφανίζεται στο χρήστη το μενού των επιπέδων. Εδώ παρατηρείται πως μόνο το πρώτο επίπεδο είναι διαθέσιμο, με τα υπόλοιπα δύο να είναι κλειδωμένα. Ξεκλειδώνονται, μόνο αν το προηγούμενο επίπεδο τους κερδηθεί. Για να παραμείνουν κλειδωμένα, τα κουμπιά του εκάστοτε επιπέδου είναι απενεργοποιημένα, καθώς και στην εικόνα τους βρίσκεται μία επιπλέον εικόνα κλειδαριάς. Επίσης κάτω δεξιά υπάρχει ένα κουμπί, όπου με το πάτημα του ο χρήστης μπορεί να πάει πίσω στο κεντρικό μενού.

Εικόνα 27: Μενού Επιπέδων

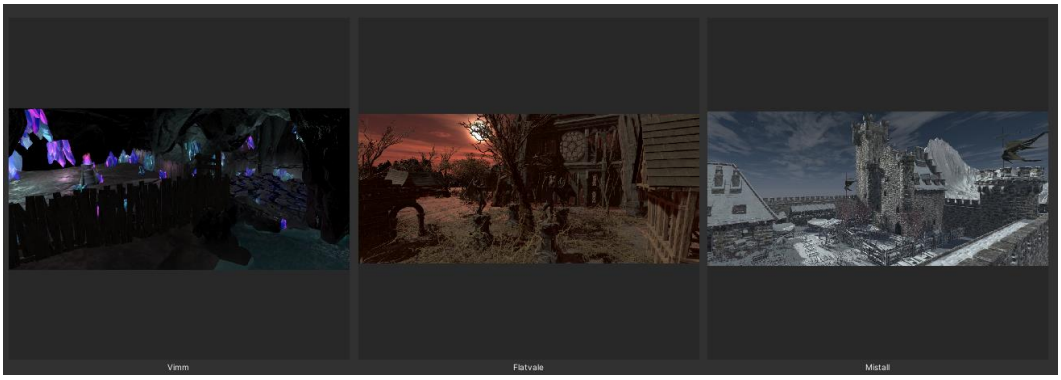
Για τις εικόνες του κάθε επιπέδου, τραβήχτηκαν στιγμιότυπα μέσα από το παιχνίδι, από διαφορετικές γωνίες των σκηνών. Αυτή η διαδικασία πραγματοποιήθηκε, περισσότερο με σκοπό την βελτίωση της αισθητικής, καθώς και για την προετοιμασία του χρήστη για το τι θα συναντήσει στην κάθε περιοχή.

Οι ονομασίες Flatvale, Mistall και Vimm, επρόκειτο για τις τρεις εναπομείναντες περιοχές, στην ιστορία του παιχνιδιού, που έμειναν για την απελευθέρωση του βασιλείου από τα τέρατα.



Εικόνα 28: Στιγμιότυπα για το Μενού Επιπέδων

Επιπλέον στιγμιότυπα, τραβήχτηκαν με διαφορετικό μέγεθος εικόνας, ώστε να χρησιμοποιηθούν στη φόρτωση του εκάστοτε επιπέδου. Τα αποτελέσματα φαίνονται στις παρακάτω εικόνες.



Εικόνα 29: Στιγμιότυπα για τη Φόρτωση των Επιπέδων

Εικόνες 30,31,32: Panel Φόρτωσης Επιπέδων Flatvale, Mistall, Vimm

Συνεχίζοντας με τη διεπαφή του χρήστη κατά τη διάρκεια της μάχης, υπάρχει ένα σταθερό UI, σε όλα τα επίπεδα. Για την περιγραφή της διεπαφής εντός του gameplay, χρησιμοποιείται ως παράδειγμα το πρώτο επίπεδο του παιχνιδιού. Η σχεδίαση του περιβάλλοντος των τριών επιπέδων, αναλύεται διεξοδικά παρακάτω.

Εικόνα 33: Διεπαφή του Gameplay

Βάσει της αρίθμησης που βλέπουμε στην Εικόνα 33, Υπάρχουν συνολικά πέντε κουμπιά για την πλοήγηση του χρήστη και ρυθμίσεις συστήματος. Αλλά, τρία κουμπιά βλέπουμε στο κάτω και κεντρικό κομμάτι της σκηνής.[23] Αυτά τα κουμπιά, υποδηλώνουν τις κινήσεις του κάθε παίκτη και αλλάζουν, σε συνολικό αριθμό και εμφάνιση, αναλόγως τον χαρακτήρα της ομάδας του χρήστη, που είναι η σειρά του εκείνη τη στιγμή να παίξει.

Όσον αφορά τα υπόλοιπα κουμπιά πλοήγησης.

1. Κουμπί Εξόδου, για την πλοήγηση πίσω στο κεντρικό μενού, αφού πρώτα εμφανιστεί και το panel επιβεβαίωσης (βλ. Εικόνα 34).
2. Κουμπί Σίγασης, για τους ήχους συστήματος.
3. Κουμπί Σίγασης, για την μουσική.
4. Κουμπί Παύσης, όπου με το πάτημα του, το παιχνίδι σταματά προσωρινά και εμφανίζεται ένα panel με επιλογές πλοήγησης-κουμπιά (βλ. Εικόνα 35). Οι επιλογές είναι το άνοιγμα του μενού βοήθειας (βλ. Εικόνα 22,23), η επανεκκίνηση του παιχνιδιού με επιβεβαίωση από τον χρήστη (βλ. Εικόνα 36) και τέλος η συνέχιση του παιχνιδιού όπου απλώς εξαφανίζεται το panel των επιλογών αυτών.
5. Το τελευταίο κουμπί είναι αυτό της εξόδου από το παιχνίδι, πίσω στο μενού των επιπέδων. Ξανά με το πάτημα του κουμπιού, εμφανίζεται panel επιβεβαίωσης, σε περίπτωση που ο χρήστης το πάτησε χωρίς να το θέλει (βλ. Εικόνα 37).

Όλα τα panel που αναφέρθηκαν, δημιουργήθηκαν με τον ίδιο τρόπο που περιεγράφηκε στην προηγούμενη υποενότητα. Δύο εικόνες με λίγη διαφάνεια στο χρώμα για το εφέ του fade, ώστε να φαίνεται ακόμα πίσω η σκηνή.

Αξιοσημείωτο είναι, ότι η ροή της πλοήγησης των μενού, παίζει πολύ σημαντικό ρόλο σε ένα παιχνίδι, μιας και ο χρήστης θα πρέπει να μπορεί να ρυθμίζει και να σταματά με οποιοδήποτε τρόπο τη ροή του παιχνιδιού, όποτε εκείνος επιθυμεί.

Εικόνα 34: Panel Επιστροφής στο Κεντρικό Μενού

Εικόνα 35: Panel Προσωρινής Παύσης Παιχνιδιού

Εικόνα 36: Panel Επανεκκίνησης Επιπέδου

Εικόνα 37: Panel Εξόδου από το Επίπεδο

Από τα τελευταία panels που βλέπουμε κατά τη διάρκεια του παιχνιδιού, είναι εκείνα του αποτελέσματος κάθε επιπέδου, με λίγα λόγια τα panel νίκης και ήττας. Αυτά λοιπόν, έχουν δημιουργηθεί, με συνδυασμό κειμένου, εφέ, εικόνων, και κουμπιών.

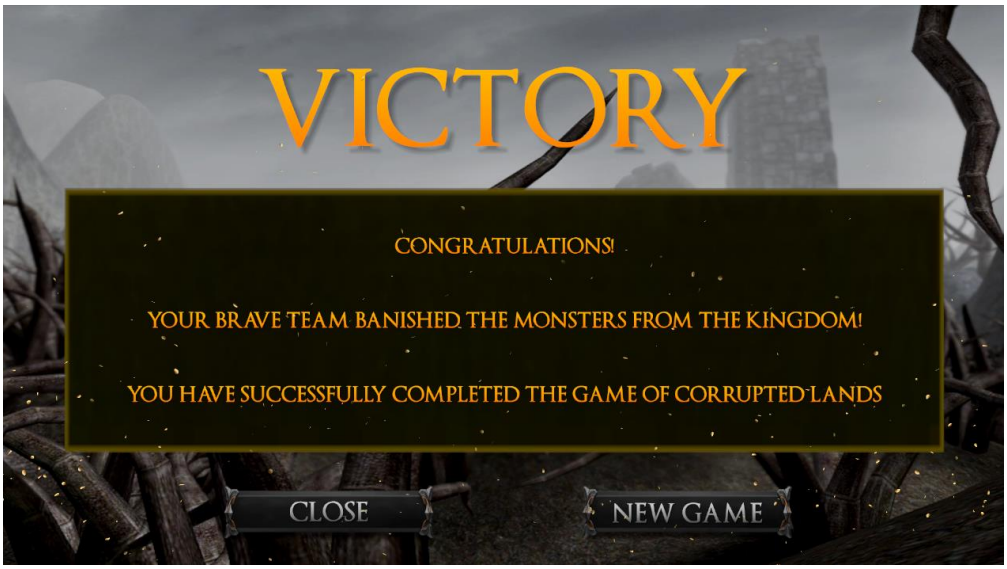
Το panel νίκης, στα δύο πρώτα επίπεδα (βλ. Εικόνα 38), εμφανίζει τα ανανεωμένα στατιστικά των χαρακτήρων της ομάδας, ανάλογα τις κινήσεις τους στο παιχνίδι. Πιο συγκεκριμένα, εμφανίζει το όνομα, την εικόνα του, τις κινήσεις – ενέργειες που διαθέτουν, μαζί με τα συνολικά Health Points (HP), τα συνολικά Experience Points (XP), το Level Up κείμενο και τα XP που κερδήθηκαν κατά τη διάρκεια της μάχης. Το τελευταίο εμφανίζεται μηδέν, στην περίπτωση που ο χαρακτήρας δεν επέζησε κατά τη διάρκεια της μάχης. Θα δούμε και στην επόμενη υποενότητα, πώς το κάθε είδος ενέργειας του κάθε χαρακτήρα, του δίνει Experience Points, και πώς γίνεται το level up.

Επιπλέον το κουμπί Continue, αποθηκεύει όλα τα ανανεωμένα στατιστικά, για την μεταφορά τους στο επόμενο επίπεδο, και επιστρέφει στον χρήστη πίσω στο Μενού Επιπέδων.

Εικόνα 38: Panel Νίκης Επιπέδων

Από την άλλη, το panel νίκης του τελευταίου επιπέδου διαφέρει από τα προηγούμενα επίπεδα. Αυτό γίνεται, μιας και ο χρήστης έχει τελειώσει όλο το παιχνίδι και έτσι του εμφανίζεται συγχαρητήριο μήνυμα. Αξίζει να παρατηρήσουμε πως το φόντο του panel αυτού είναι αρκετά διαφορετικό σε χρώμα από τα υπόλοιπα (βλ. Εικόνα 38). Αυτή είναι μία λεπτομέρεια, που προστέθηκε για να αναδείξει πως το παιχνίδι σιγά σιγά τελειώνει και με βάση την ιστορία του, πως σταδιακά απελευθερώνονται οι τελευταίες περιοχές του βασιλείου.

Επιπροσθέτως υπάρχουν δύο διαφορετικά κουμπιά σε αυτή την περίπτωση. Το κουμπί Close, πηγαίνει τον χρήστη πίσω στο Μενού Επιπέδων, ενώ το κουμπί New Game, διαγράφει όλα τα δεδομένα που αποθηκεύτηκαν κατά τη διάρκεια του παιχνιδιού και πλοηγεί τον χρήστη πίσω στο Κεντρικό Μενού. Σαν δεδομένα που αποθηκεύτηκαν, εννοούνται τα στατιστικά των παικτών, τα επίπεδα που είχαν ξεκλειδωθεί, καθώς και η Σκηνή Εισαγωγής (βλ. Εικόνα 15,16) μπορεί να προβληθεί ξανά.



Εικόνα 39: Panel Νίκης όλου του Παιχνιδιού

Τέλος, το panel ήττας, εμφανίζεται σε όλα τα επίπεδα, αν δεν επέλθει νίκη. Αυτό είναι το μόνο panel, που χρησιμοποιείται το κόκκινο Color Gradient by 4 (βλ. Εικόνα 20), για να υποδηλώσει την κατάλληλη ατμόσφαιρα, μιας και οι χαρακτήρες δεν επέζησαν. Εδώ βλέπουμε το κουμπί Try Again, το οποίο δεν αποθηκεύει στατιστικά για τους χαρακτήρες και ξεκινά το παιχνίδι ξανά για τον χρήστη. Τέλος το κουμπί Close, επαναφέρει τον χρήστη στο Μενού Επιπέδων.

Εικόνα 40: Panel Ήττας Επιπέδων

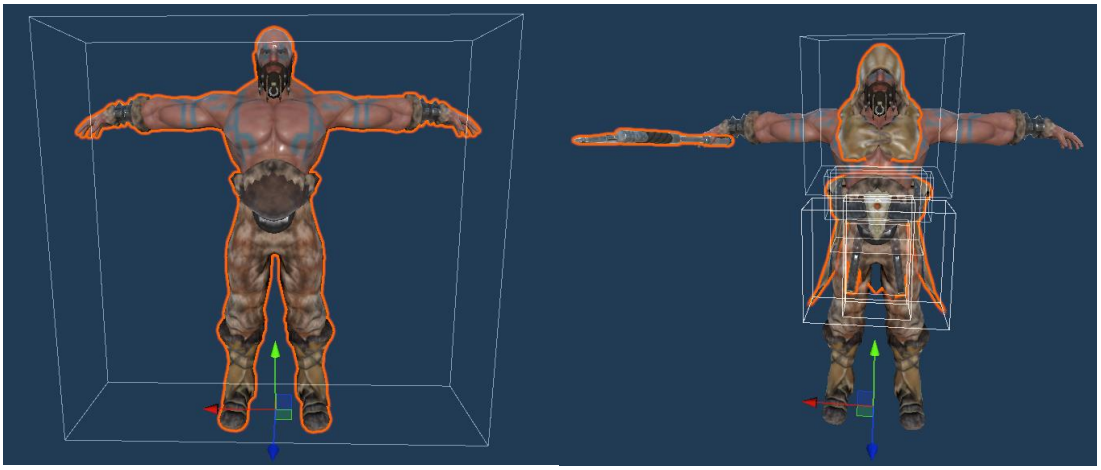
Για την παραπάνω υλοποίηση της ροής όλης της διεπαφής του χρήστη, χρειάστηκε εκτεταμένος έλεγχος και σχεδιαγράμματα, ώστε να υπάρχει μία λογική σύνδεση όλων των στοιχείων του UI. Αυτή είναι και η υποχρέωση ενός Game Designer, η σχεδίαση του παιχνιδιού με τον πιο βέλτιστο τρόπο, στοχεύοντας στην καλύτερη εμπειρία του χρήστη.

4.2 Χαρακτήρες

Οι χαρακτήρες του παιχνιδιού, εισάχθηκαν από ένα πακέτο της Unity [18], το οποίο περιλαμβάνει τα 3D μοντέλα, με materials και υφές, καθώς και τα animations που διαθέτει το κάθε ένα.

4.2.1 Εμφάνιση και Χαρακτηριστικά

Το κάθε τρισδιάστατο μοντέλο χαρακτήρα, μπορεί να προσαρμοστεί με διαφορετικά ρούχα και αξεσουάρ, που να ταιριάζει στην αισθητική του παιχνιδιού. Έχοντας την κατάλληλη συλλογή διαθέσιμη μέσα στο πακέτο, η εμφάνιση του κάθε χαρακτήρα περιέχει χαρακτηριστικά μάχης, όπως όπλα, πανοπλίες και ρουχισμό, τα οποία είναι τοποθετημένα με τέτοιο τρόπο πάνω στο κάθε μοντέλο ούτως ώστε κατά τη διάρκεια του animation, να υπάρχει ένα λογικό αποτέλεσμα, με την κίνηση των ρούχων και των όπλων να ακολουθεί την κίνηση του παίκτη (βλ. Εικόνα 41).



Εικόνα 41: Παράδειγμα – Αρχικό μοντέλο κεντρικού χαρακτήρα vs. τελικό

Με την προσθήκη του ρουχισμού, πλέον ο χαρακτήρας ταιριάζει περισσότερο στην αισθητική του φανταστικού κόσμου. Η ίδια διαδικασία έγινε για όλους τους χαρακτήρες του παιχνιδιού, κεντρικούς και εχθρικούς, με τα αποτελέσματα να φαίνονται παρακάτω (βλ. Εικόνα 42,43).

Εικόνα 42: Κεντρικοί Χαρακτήρες

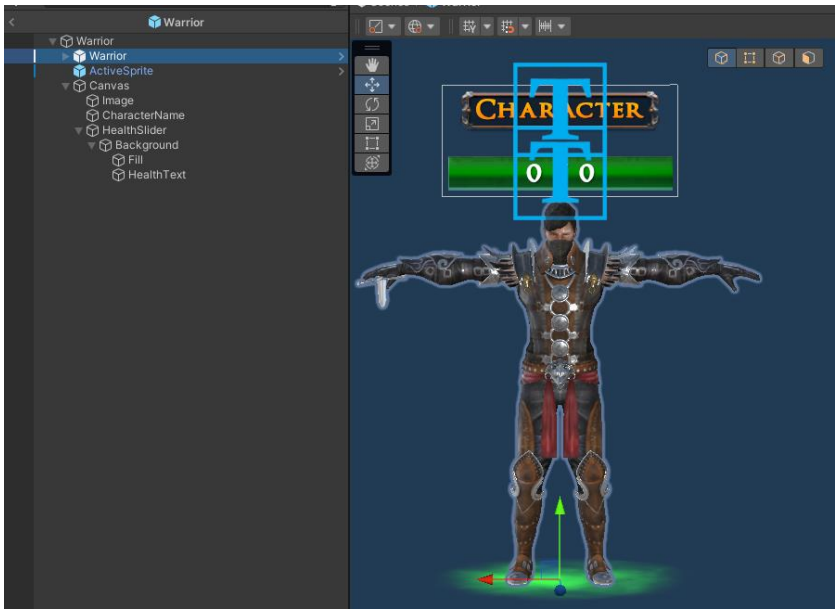
Εικόνα 43: Εχθρικοί Χαρακτήρες

4.2.2 UI Χαρακτήρων

Η διεπαφή χρήστη για τους χαρακτήρες, περιλαμβάνει έναν καμβά ο οποίος είναι τύπου world space, μιας και οι καμβάδες είναι προκαθορισμένοι στην επιλογή Screen Space. Ένας world space καμβάς, επηρεάζεται από τον φωτισμό και την κάμερα, πράγμα το οποίο χρειαζόμαστε, διότι θέλουμε ο καμβάς να είναι προσκολλημένος στον χαρακτήρα καθ' όλη τη διάρκεια του παιχνιδιού.

Ο καμβάς, περιέχει το όνομα του κάθε χαρακτήρα, καθώς και ένα πλαίσιο γύρω του, το οποίο θα εμφανίζεται όταν έχει έρθει η σειρά του. Επίσης, μια μπάρα υγείας που αντιπροσωπεύει την κατάσταση υγείας του χαρακτήρα (health points), τα οποία μειώνονται σε περίπτωση που δέχεται επίθεση και αυξάνονται σε περίπτωση που θεραπεύεται.

Άλλο ένα στοιχείο διεπαφής, που διακρίνουμε παρακάτω στην Εικόνα 44, είναι ένα πράσινο επίπεδο στη βάση του χαρακτήρα. Το επίπεδο αυτό, είναι πάντα απενεργοποιημένο, και εμφανίζεται μόνο όταν ο χαρακτήρας επιλεγεί για την χρήση κάποιας ενέργειας από τους κεντρικούς χαρακτήρες.



Εικόνα 44: Στοιχεία UI Χαρακτήρα

Ένα ακόμη στοιχείο διεπαφής, είναι το panel με τις ενέργειες, που έχουν διαθέσιμες οι κεντρικοί χαρακτήρες κατά τη διάρκεια της μάχης. Το panel αυτό, εμφανίζεται στο κάτω μέρος της κάθε σκηνής (βλ. Εικόνα 33) και επιτρέπει στον χρήστη, να επιλέξει μεταξύ διαφορετικών ενεργειών μάχης (battle actions), όπως επίθεση, θεράπευση και χρήση μαγικών ικανοτήτων. Αντιπροσωπεύονται από εικόνες, που διευκολύνουν τον χρήστη να αναγνωρίσει εύκολα το κάθε είδος, σε συνδυασμό με το μενού βοήθειας, που περιγράφει την κάθε ενέργεια ξεχωριστά. Αξίζει να σημειωθεί πως τα battle actions, διαφέρουν από χαρακτήρα σε χαρακτήρα, όσον αφορά τον συνολικό αριθμό, το είδος και τα χαρακτηριστικά.

4.2.3 Animations Χαρακτήρων

Η κίνηση που αποδίδεται στους χαρακτήρες, ή αλλιώς το Animation, παίζει πολύ σημαντικό ρόλο για ένα παιχνίδι, μιας και είναι ένα από τα βασικά στοιχεία που δίνουν έναν τόνο ροής και την αίσθηση ζωής στο περιβάλλον. Βοηθούν στο να γίνει το παιχνίδι πιο ελκυστικό οπτικά, δημιουργώντας μία συναρπαστική εμπειρία στον χρήστη.

Υπήρχαν πολλά animations διαθέσιμα στο πακέτο χαρακτήρων που χρησιμοποιήθηκε, όμως οκτώ ήταν αυτά που χρειάζονταν για το παιχνίδι, τα οποία είναι τα εξής.

- Idle, που είναι το προκαθορισμένο animation, όταν ο χαρακτήρας δεν ενεργεί με το περιβάλλον και απλώς στέκεται με μία μικρή κίνηση, συνήθως αναπνοή.
- Run, το animation που ενεργοποιείται όταν επιλεχθεί η ενέργεια επίθεσης σε κοντινή απόσταση, οπότε ο χαρακτήρας τρέχει προς το στόχο του.
- Run Back, για την επιστροφή του χαρακτήρα στη θέση του μετά από μία επίθεση κοντινής απόστασης.
- Melee Attack, το animation που ο χαρακτήρας τραυματίζει τον αντίπαλο σε κοντινή απόσταση.
- Range Attack, για την επίθεση σε μακρινή εμβέλεια, όπου ο χαρακτήρας παραμένει στη θέση του και εκτελεί ένα ξόρκι.
- Got Hit, που είναι το animation που δείχνει τον χαρακτήρα να τραυματίζεται.
- Healing, το animation για τη θεράπευση των χαρακτήρων.
- Και το Die, που είναι το τελικό animation αφού όταν ενεργοποιείται, ο χαρακτήρας εμφανίζεται να πέφτει στο πάτωμα νεκρός.

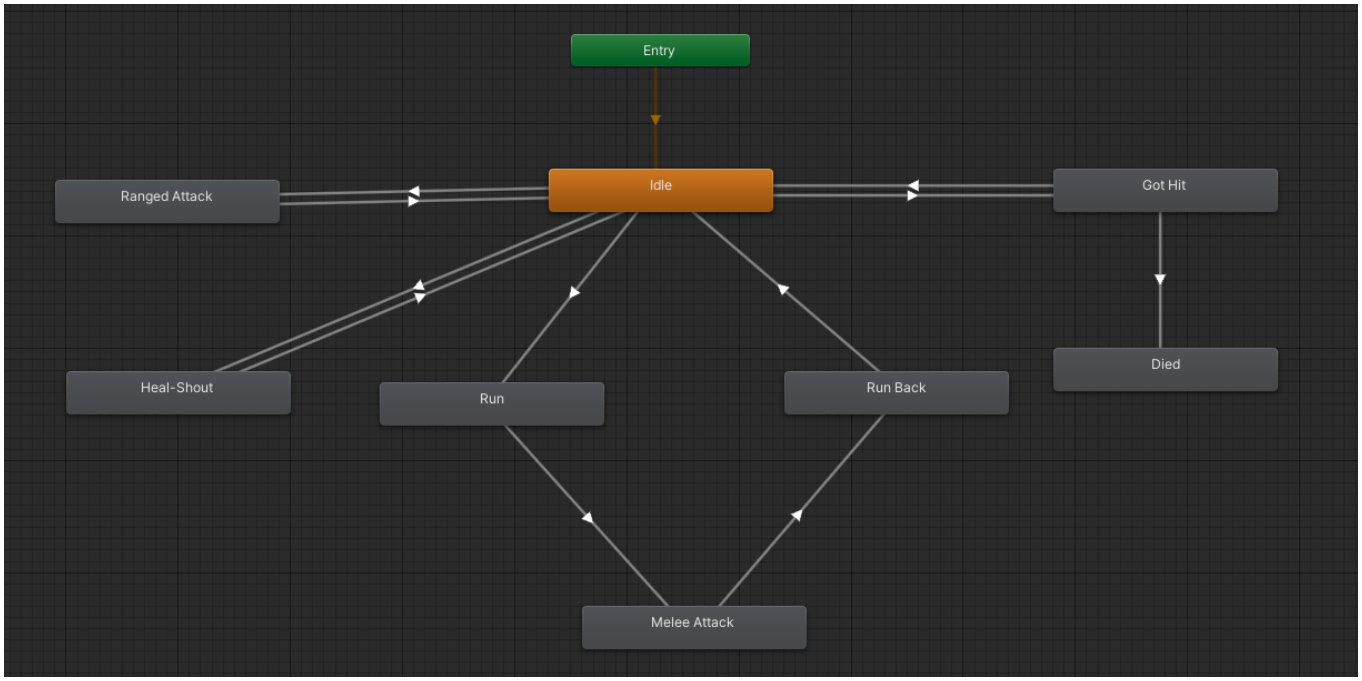
Εδώ αξίζει να σημειωθεί πως κάποιοι από τους χαρακτήρες, δεν είχαν όλα τα animations που χρειάζονταν για τη μάχη, συγκεκριμένα το Run Back. Για να λύσω αυτό το πρόβλημα χρησιμοποίησα το πρόγραμμα Mixamo (βλ. Κεφ. 2.6).

Αρχικά λοιπόν, περιηγήθηκα στο Mixamo, ώστε να βρεθεί το κατάλληλο animation για το Run Back. Ύστερα, τοποθετήθηκε το Mesh του χαρακτήρα, στο περιβάλλον αναπαράστασης animation μέσα στο πρόγραμμα, και μετά από λίγα λεπτά επεξεργασίας του μοντέλου, κατάφερε να γίνει η σύνδεση των αρθρώσεων του χαρακτήρα πάνω στο νέο animation.

Έχοντας λοιπόν, όλα τα animations για τους χαρακτήρες, έπρεπε να γίνει η σύνδεση μεταξύ τους, ώστε να εμφανίζονται οι κατάλληλες κινήσεις την κατάλληλη στιγμή, με τη βοήθεια του Animator Controller.

Ο Animator Controller, είναι ένα πολύ χρήσιμο και ισχυρό εργαλείο, που επιτρέπει τον έλεγχο της συμπεριφοράς ενός χαρακτήρα ή αντικειμένου στο παιχνίδι. Αποθηκεύει τη λογική που τίθεται από τον προγραμματιστή, τα animations, την ανταπόκριση στα δοθείσα δεδομένα και την αλληλεπίδραση με άλλα animated μοντέλα. Με τη χρήστη μεταβλητών και transitions μεταξύ των animations, ο προγραμματιστής να μπορεί, εύκολα και γρήγορα, να ελέγξει έναν ή και πολλούς χαρακτήρες μέσω ενός μόνο Animator Controller.

Με αυτόν τον τρόπο, για όλους τους χαρακτήρες χρησιμοποιήθηκε ο ίδιος τύπος Controller, μιας και όλοι είχαν τις ίδιες περιπτώσεις κινήσεων. Όπως φαίνεται στην Εικόνα 45, τα κουτάκια αντιπροσωπεύουν τα οκτώ διαφορετικά animation-καταστάσεις, ενώ τα βελάκια υποδεικνύουν τα transitions, τις μεταβάσεις δηλαδή, από κατάσταση σε κατάσταση.



Εικόνα 45: Ο Animator Controller των Χαρακτήρων

4.3 Δημιουργία Περιβάλλοντος Παιχνιδιού (Game Environment)

Η δημιουργία ενός περιβάλλοντος παιχνιδιού, είναι μία δύσκολη και χρονοβόρα διαδικασία, που απαιτεί σωστό σχεδιασμό, οργάνωση και σκέψη. Οι game designers, πρέπει να συνδυάσουν την ιστορία, τους χαρακτήρες, τα αντικείμενα και το φυσικό περιβάλλον, όπου αλληλοεπιδρούν οι χρήστες, κατά τη διάρκεια του παιχνιδιού. Ο σχεδιασμός, προϋποθέτει προσοχή και αλληλουχία των παραπάνω στοιχείων, για ένα εντυπωσιακό αποτέλεσμα.

4.3.1 Ιστορία του Παιχνιδιού

Το πρώτο βήμα στο σχεδιασμό του περιβάλλοντος, είναι η δημιουργία μιας ιστορίας. Η ιστορία, πρέπει να παρέχει, στον χρήστη, πληροφορίες για τον κόσμο, τους χαρακτήρες και τα κίνητρά τους. Αυτή, είναι το θεμέλιο του περιβάλλοντος του παιχνιδιού, καθώς και αποτελεί την βάση για την παραγωγή του φυσικού κόσμου.

Στην παρούσα πτυχιακή εργασία, η εξιστόρηση, γίνεται στην σκηνή εισαγωγής, η οποία εμφανίζεται μόνο με το πρώτο άνοιγμα του παιχνιδιού (βλ. Κεφ. 4.1.1). Εμβαθύνοντας στην ιστορία, αναφέρεται σε ένα βασίλειο του φανταστικού κόσμου, το οποίο κατακτήθηκε από τέρατα και έκαναν τους κατοίκους να φύγουν μακριά. Μετά από χρόνια μία ομάδα πολεμιστών, εξεγέρθηκαν εναντίων των τεράτων, πολεμώντας για την ανάκτηση των Διεφθαρμένων Περιοχών (“Corrupted Lands”). Έχοντας αυτή την ιστορία, δημιουργήθηκαν τα τρία επίπεδα που αντιπροσωπεύουν τις τρεις εναπομείναντες περιοχές του βασιλείου.

Το πρώτο επίπεδο με όνομα Flatvale, αντιπροσωπεύει ένα από τα χωριά του βασιλείου, Τα κατεστραμμένα σπίτια και η κατεστραμμένη εκκλησία στην πλατεία, καθώς και τα

διάσπαρτα διαλυμένα αντικείμενα σε συνδυασμό με πολλά νεκρά στοιχεία φύσης (δέντρα, θάμνους, κορμούς), προσθέτουν έναν απόκοσμο τόνο και την αίσθηση της ερείπωσης.

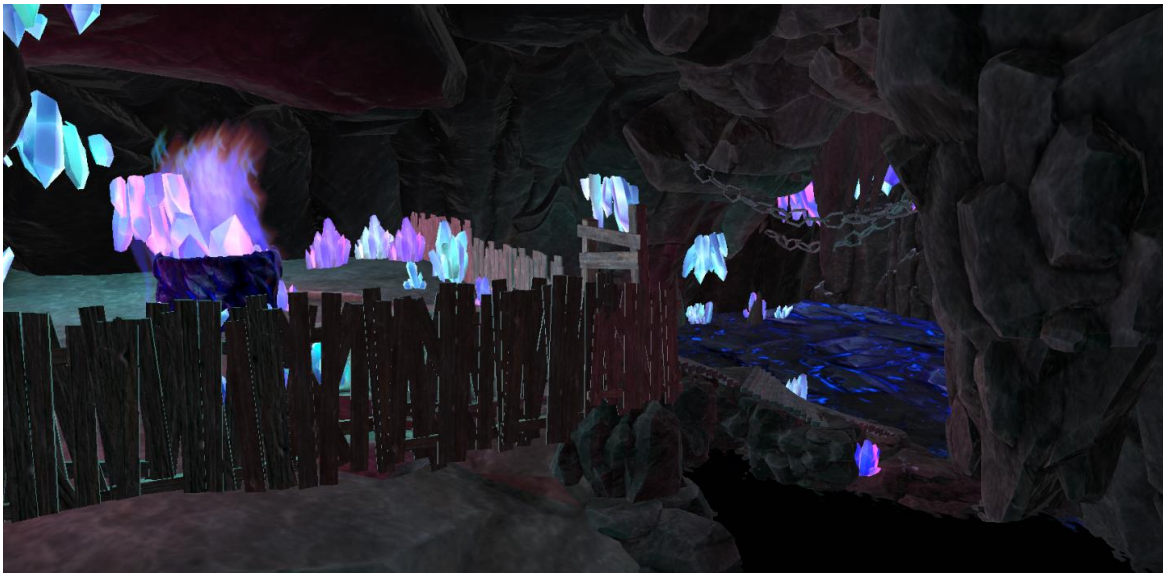
Εικόνα 46: 1ο Επίπεδο - Flatvale

Το δεύτερο επίπεδο, Mistall, αντικατοπτρίζει το κάστρο του βασιλείου, το οποίο βρίσκεται σε μεγάλο υψόμετρο, ανάμεσα σε χιονισμένα βουνά. Οι τρεις πύργοι με την κεντρική αυλή, καθώς και καλύβες με ξεχασμένα, από τους κατοίκους, αντικείμενα, έχουν τοποθετηθεί με τέτοιο τρόπο, ώστε το περιβάλλον να μοιάζει εγκαταλελειμμένο.

Εικόνα 47: 2ο Επίπεδο - Mistall

Το τελευταίο επίπεδο, Vimm, αναπαριστά την τελευταία μάχη, στη σπηλιά των τεράτων. Αφού κατάφερε να κερδηθεί το προηγούμενο επίπεδο, μεταφερόμαστε στην πηγή από όπου προήλθαν τα τέρατα. Η μάχη εξελίσσεται σε κλειστό χώρο, για την

προσθήκη της διαφορετικότητας από τις άλλες δύο περιοχές. Στοιχεία μαγείας με χρήση κρυστάλλων και VFX, σε συνδυασμό με ένα απόκοσμο βραχώδες περιβάλλον, συνθέτουν μια εχθρική ατμόσφαιρα, που αντιπροσωπεύει συνήθως ένα τελικό επίπεδο.



Εικόνα 48: 2ο Επίπεδο - Mistall

4.3.2 Ζητήματα Σχεδιασμού Περιβάλλοντος

Κατά τη δημιουργία του περιβάλλοντος του παιχνιδιού. Πρέπει να ληφθούν υπόψη μερικά σημαντικά στοιχεία σχεδιασμού, τα οποία απαντώνται άμεσα. Αυτά είναι το στυλ του παιχνιδιού, το μέγεθος του χάρτη για το περιβάλλον και το πόσο χώρο καταλαμβάνουν οι χαρακτήρες και τα αντικείμενα, βάσει αυτού.

Το στυλ που επιλέχθηκε για τον σχεδιασμό του “Corrupted Lands”, είναι ενός φανταστικού κόσμου, με την επιλογή γραφικών τύπου AA. Τα γραφικά AA, είναι μια τεχνική anti-aliasing, που χρησιμοποιείται για τη δημιουργία πιο ομαλών ακμών και πιο ρεαλιστικών εικόνων παιχνιδιού.

Το μέγεθος του χάρτη περιβάλλοντος, δεν χρειάζονταν να είναι αρκετά μεγάλο, μιας και το είδος του παιχνιδιού δεν είναι εξερεύνησης αλλά στρατηγικής. Με την κάμερα λοιπόν, σταθερή σε ένα σημείο, η τοποθέτηση των αντικειμένων, έγινε με βάση το οπτικό της πεδίο, έχοντας ως αποτέλεσμα έναν μικρό σχετικά χάρτη. Οι χαρακτήρες τοποθετούνται σε ένα χώρο 15 x 15μ., μπροστά στην κάμερα, με τον υπόλοιπο χώρο της σκηνής να είναι γύρω στα 100 μέτρα, τον οποίο καταλαμβάνουν τα υπόλοιπα αντικείμενα που βρίσκονται στον φόντο.

Εικόνα 49 : Κάτοψη 1ου Επιπέδου

Εικόνα 50 : Κάτοψη 2ου Επιπέδου

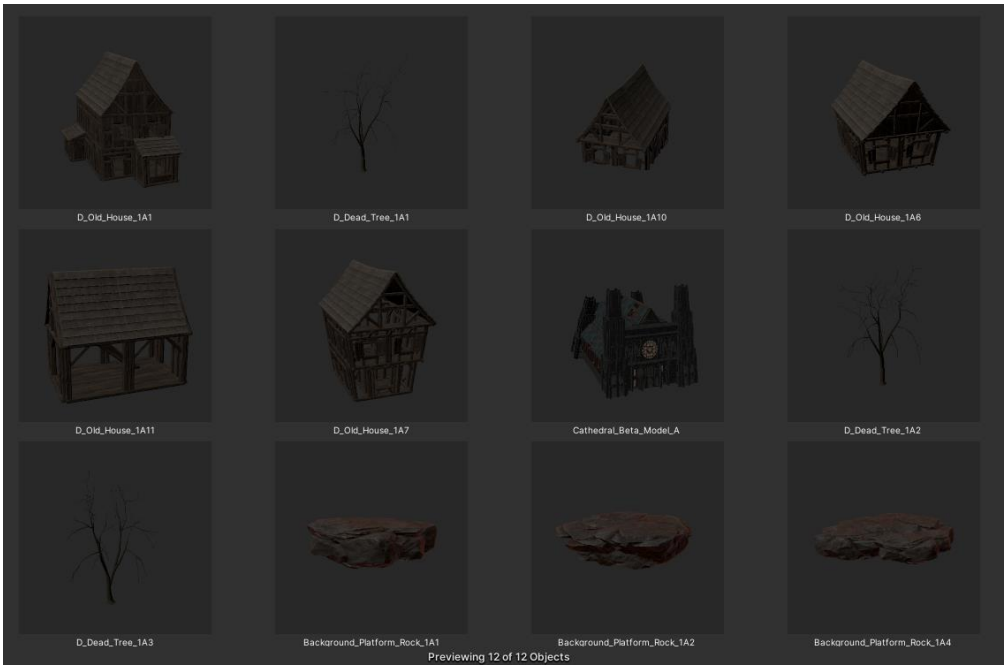
Εικόνα 51 : Κάτοψη 3ου Επιπέδου (Εξωτερικό κομμάτι σπηλιάς)

4.3.3 Στοιχεία και Prefabs που χρησιμοποιήθηκαν

Η δημιουργία του φυσικού περιβάλλοντος του κάθε επιπέδου, περιλαμβάνει στοιχεία, όπως 3D μοντέλα, έδαφος [20] και skyboxes [19]. Η τοποθέτηση των αντικειμένων, πρέπει να γίνει με τέτοιο τρόπο, ώστε να έχει νόημα με το σχεδιασμό και την ιστορία του παιχνιδιού, αλλά και ταυτόχρονα να ενισχύει την εμπειρία του παίκτη. Τα στοιχεία αυτά, αγοράστηκαν από το Unity Asset Store και αναφέρονται αναλυτικά παρακάτω.

4.3.3.1 Prefabs Πρώτου Επιπέδου

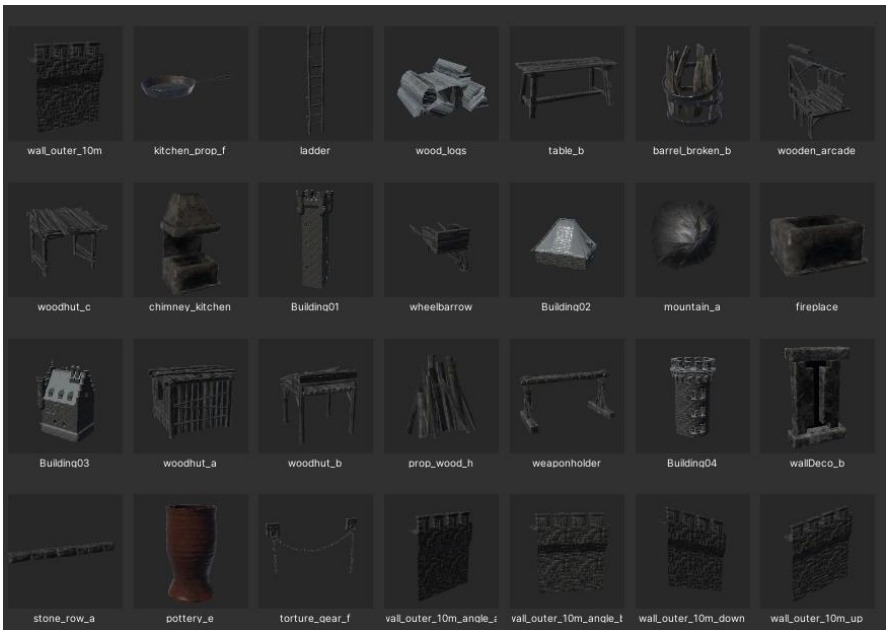
Για το φυσικό περιβάλλον του 1ου επιπέδου χρησιμοποιήθηκαν prefabs από δύο διαφορετικά πακέτα του Asset Store [16] [17]. Περιέχει σπίτια, δέντρα, βράχους και πολλά άλλα. Μερικά από αυτά που χρησιμοποιήθηκαν φαίνονται στην Εικόνα 52.



Εικόνα 52: Μερικά από τα Prefabs για τη σκηνή Flatvale

4.3.3.2 Prefabs Δεύτερου Επιπέδου

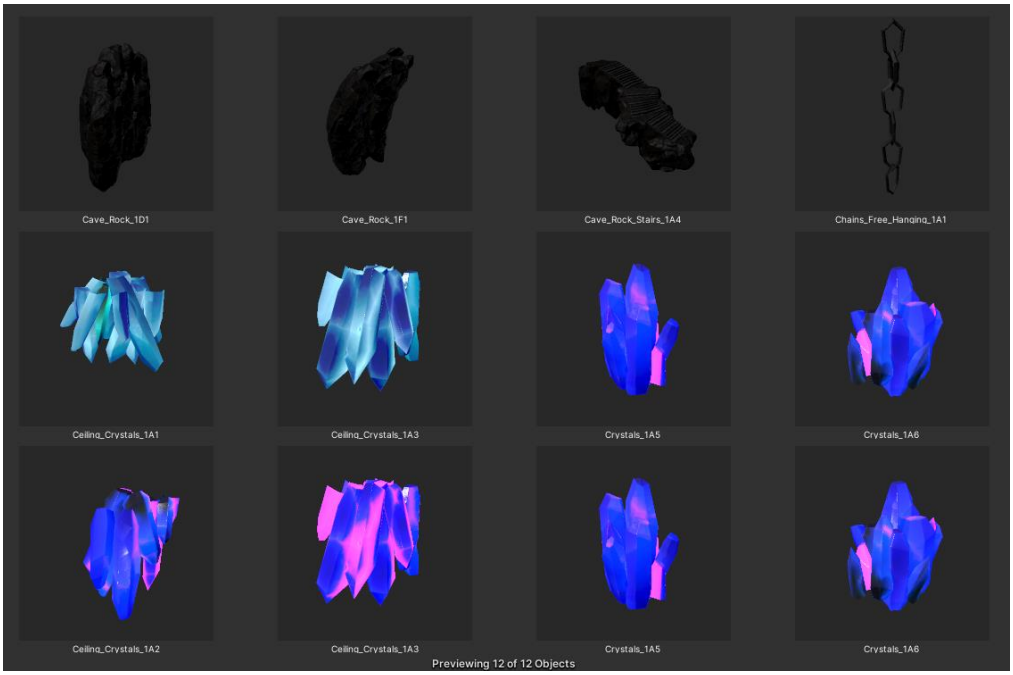
Για τη δημιουργία του 2ου επιπέδου, ένα πακέτο ήταν αυτό που χρησιμοποιήθηκε από το Asset Store [15]. Περιέχει ένα ολόκληρο κάστρο με πύργους και κτήρια, μεσαιωνικά αντικείμενα εποχής, όπως πέτρινους φούρνους, τραπέζια και καλύβες, καθώς και βουνά για το φόντο. Μερικά από αυτά φαίνονται στην Εικόνα 53.



Εικόνα 53: Μερικά από τα Prefabs για τη σκηνή Mistall

4.3.3.3 Prefabs Τρίτου Επιπέδου

Τέλος, για το τελευταίο επίπεδο, χρησιμοποιήθηκαν τρία πακέτα από το Asset Store [17] [22] [28]. Περιλαμβάνουν βραχώδεις επιφάνειες και γκρεμούς, κρυστάλλους, αλυσίδες, καθώς και 2 materials. Το ένα material αναπαριστά το νερό που υπάρχει στη σκηνή, ενώ το δεύτερο, το έδαφος της σπηλιάς. Ενδεικτικά φαίνονται μερικά στην Εικόνα 54 παρακάτω.



Εικόνα 54: Μερικά από τα Prefabs για τη σκηνή Vimm

4.4 Φωτισμός

Το σύστημα φωτισμού της Unity, επιτρέπει τη δημιουργία ρεαλιστικών και δυναμικών εφέ φωτός. Περιλαμβάνει τη χρήση πηγών φωτισμού, όπως Point Lights, Spot Lights, Directional Lights and Ambient Lights. Το σύστημα αυτό, επιτρέπει στον προγραμματιστή, να ελέγχει την ένταση του κάθε φωτός, την εμβέλεια τους, το χρώμα και πολλά άλλα. Γενικότερα τον τρόπο με τον οποίο αλληλοεπιδρούν τα φώτα της σκηνής, με μοντέλα και αντικείμενα που βρίσκονται μέσα σε αυτήν.

Υπάρχουν δύο επιλογές φωτισμού, real-time lighting και baked lighting. Ο real-time, δηλαδή φωτισμός σε πραγματικό χρόνο, είναι αυτός που προκαθορίζει η Unity, και σημαίνει τον υπολογισμό του φωτός κάθε καρέ. Από την άλλη πλευρά, ο baked φωτισμός, υπολογίζεται από τον προγραμματιστή, πριν καν ο χρήστης παίξει το παιχνίδι. Στην ουσία κάνει bake τα textures, δημιουργώντας lightmaps, τα οποία είναι χάρτες φωτός που εφάπτονται πάνω από όλα τα αντικείμενα της σκηνής, αποθηκεύοντας έτσι από πριν, πως αλληλοεπιδρά το κάθε αντικείμενο με αυτό το είδος φωτισμού.

Ο φωτισμός που επιλέχθηκε για το παιχνίδι, είναι ο baked, μιας και η απόδοση του παιχνιδιού, είναι πολύ καλύτερη σε σχέση με τον φωτισμό σε πραγματικό χρόνο. Περισσότερες λεπτομέρειες για την απόδοση με βάση τον φωτισμό, περιγράφονται παρακάτω (βλ. Κεφ. 5.2).[3]

Για τις σκηνές της πτυχιακής εργασίας, τα είδη πηγών φωτός που τοποθετήθηκαν, είναι τα Directional Lights και τα Point Lights. Directional Lights, ή αλλιώς φώτα κατευθύνσεως, αντιπροσωπεύουν μια μακρινή πηγή φωτός, όπως ο ήλιος. Παράγουν, δηλαδή, παράλληλες ακτίνες, που ταξιδεύουν προς μία κατεύθυνση και χρησιμοποιούνται για την προσομοίωση φυσικού φωτισμού. Σε αντίθεση με τα Point Lights, τα οποία εκπέμπουν φως από ένα μεν σημείο, αλλά προς όλες τις κατευθύνσεις. Χρησιμοποιούνται περισσότερο, για την προσομοίωση φωτός από λάμπες. [3]

4.5 Οπτικά Εφέ (Visual Effects -VFX)

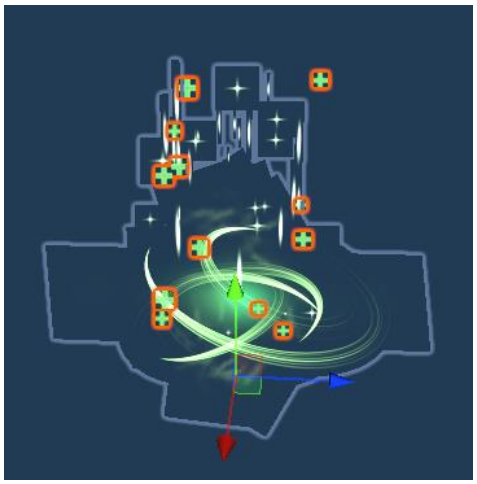
Τα Visual Effects, είναι ειδικά εφέ, όπως σωματίδια και trails, για την αναπαράσταση ρεαλιστικών εκρήξεων, φωτιές, νερό, ομίχλη, καπνούς, μαγικά ξόρκια και πολλά ακόμη. Χρησιμοποιούνται στα περισσότερα παιχνίδια πλέον, μιας και δημιουργούν ένα καθηλωτικό περιβάλλον, το οποίο βελτιώνει την αισθητική και οπτική πλευρά του παιχνιδιού. Επίσης ενισχύουν την αντίληψη του χώρου και του χρόνου, και μπορούν να δημιουργήσουν στιγμές που εμβαθύνουν στο συναίσθημα του χρήστη.

Για παράδειγμα, αν σε ένα παιχνίδι συμβεί μια έκρηξη, και ο χώρος καλυφθεί με φωτιά, ο χρήστης θα αγχωθεί, θέλοντας να φύγει από εκείνο το σημείο, αλλά παράλληλα θα εκθαμβωθεί παρατηρώντας τον τρόπο με τον οποίο αναπαρίστανται τέτοιου είδους σκηνικά από VFX. Οπότε όσο καλύτερη είναι η ποιότητα τους από πλευρά γραφικών, τόσο πιο ελκυστικό είναι και το παιχνίδι στον χρήστη.

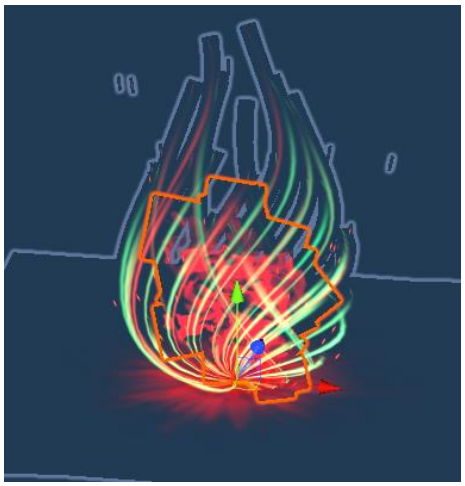
Για το “Corrupted Lands”, χρησιμοποιήθηκαν VFX, από δύο πακέτα της Unity Asset Store [24] [25], για τις ενέργειες μάχης, καθώς και άλλο ένα, για τη διεπαφή του χρήστη στη νίκη του παιχνιδιού και στο κεντρικό μενού [13].

Τα VFX, για τις ενέργειες μάχης, αναπαριστούν τις επιθέσεις από απόσταση, δηλαδή τα μαγικά ξόρκια, που κάνουν οι χαρακτήρες μέσα στη σκηνή, καθώς και τη θεράπευση τους. Μερικά παραδείγματα των εφέ που χρησιμοποιήθηκαν, φαίνονται παρακάτω (βλ. Εικόνες 55,56,57,58).

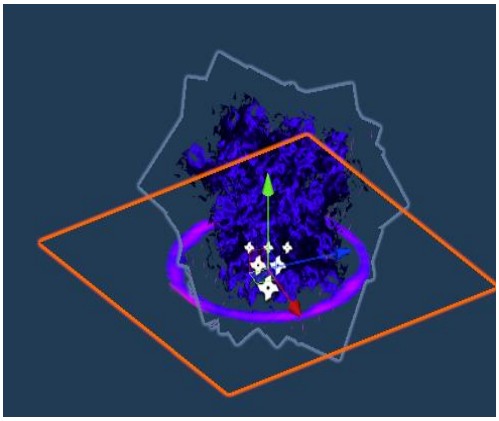
Τα εφέ για την νίκη του παιχνιδιού και το κεντρικό μενού, εφαρμόστηκαν πάνω στον εκάστοτε καμβά που κρατούσε αυτά τα δύο στοιχεία διεπαφής. Είναι τοποθετημένα μπροστά από τους καμβάδες, μιας και αν βρίσκονταν πίσω, δεν θα φαινόντουσαν στην κάμερα κατά τη διάρκεια του παιχνιδιού. (βλ. Εικόνα 21, 39).



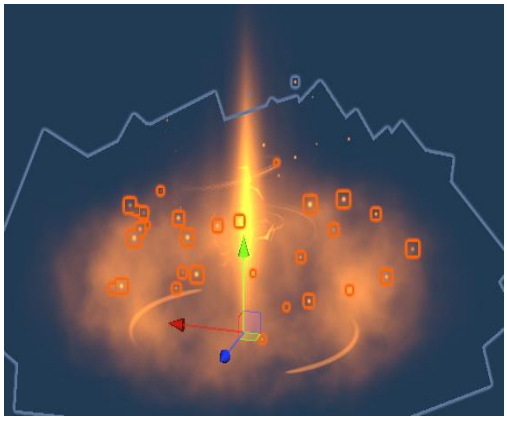
Εικόνα 55: Εφέ Θεράπευσης



Εικόνα 56: Εφέ Μεμονωμένου Χαρακτήρα



Εικόνα 57: Εφέ Μεμονωμένου Χαρακτήρα



Εικόνα 58: Εφέ Μεμονωμένου Χαρακτήρα

4.6 Μουσική και Ηχητικά Εφέ (Sound Effects – SFX)

Η χρήση της μουσικής και των ηχητικών εφέ, είναι σημαντικοί παράγοντες για τη δημιουργία ενός παιχνιδιού, αφού βοηθούν στην διατήρηση της κατάλληλης ατμόσφαιρας και παρέχουν σημαντικές ενδείξεις στον χρήστη όσο παίζει.

Η μουσική, χρησιμοποιείται για να δώσει τον τόνο του παιχνιδιού, αλλά και για να δημιουργήσει την κατάλληλη συναισθηματική ατμόσφαιρα, είτε αυτή είναι αγωνίας, είτε ενθουσιασμού. Βοηθά στην αίσθηση προόδου στο παιχνίδι, όπως για παράδειγμα στην νίκη ενός επιπέδου, όπου η μουσική θα είναι ηρωική, σε σχέση με την ήττα του επιπέδου, όπου η μουσική γίνεται δραματική. Ακριβώς αυτό έχω καταφέρει και στο Corrupted Lands.

Για την μουσική χρησιμοποίησα ένα πακέτο του Unity Asset Store [27], το οποίο περιλαμβάνει διαφόρων ειδών μουσικές ενός φανταστικού κόσμου. Για κάθε σκηνή χρησιμοποιείται και διαφορετική μουσική, ενώ για την περιήγηση στο κεντρικό μενού και στα επίπεδα έγινε χρήση του ίδιου μουσικού κομματιού, μιας και αποτελεί το main score του “Corrupted Lands”. Αξίζει να σημειωθεί πως, για την νίκη και την ήττα του κάθε επιπέδου, ακούγονται διαφορετικές μουσικές, οι οποίες παίζουν πάνω από το κομμάτι της εκάστοτε σκηνής.

Τα ηχητικά εφέ από την άλλη, χρησιμοποιούνται συχνά για να δώσουν feedback στον χρήστη, σχετικά με τις ενέργειές του, όπως όταν χτυπάει έναν εχθρό, όταν θεραπεύεται, όταν παίρνει ένα αντικείμενο. Βοηθούν στην αίσθηση της έντασης και ενθουσιασμού του χρήστη, με αποτέλεσμα τη βύθιση του στον κόσμο του παιχνιδιού.

Για το “Corrupted Lands”, χρησιμοποιήθηκε ένα πακέτο από το Unity Asset Store [26]. Τέτοια ηχητικά εφέ χρησιμοποιήθηκαν για τα είδη ενεργειών μάχης κατά τη διάρκεια του παιχνιδιού, για τέσσερις διαφορετικές περιπτώσεις. Την επίθεση σε κοντινή απόσταση, σε μακρινή απόσταση, την θεράπευση, και τον θάνατο των χαρακτήρων. Επίσης, τρεις ήχοι χρησιμοποιήθηκαν για τη διεπαφή του χρήστη μετά το πάτημα διαφόρων κουμπιών. Ένας για το πάτημα βασικών κουμπιών του κεντρικού μενού και των πάνελ που εμφανίζονταν (όπως Start, Help, Quit, Options). Ένας για τα διαθέσιμα κουμπιά-εργαλεία που έχει στη διάθεση του ο χρήστης (Κουμπί πίσω, Σίγαση ήχων), καθώς και ένας ήχος για την επιλογή επιπέδου.

Εν κατακλείδι, η χρήση μουσικής και sound effects, προσελκύουν τους χρήστες, ώστε να μείνουν αφοσιωμένοι, δίνοντας τους έναν λόγο να παίζουν το παιχνίδι για μεγαλύτερη χρονική διάρκεια, ωφελώντας έτσι και τον δημιουργό.

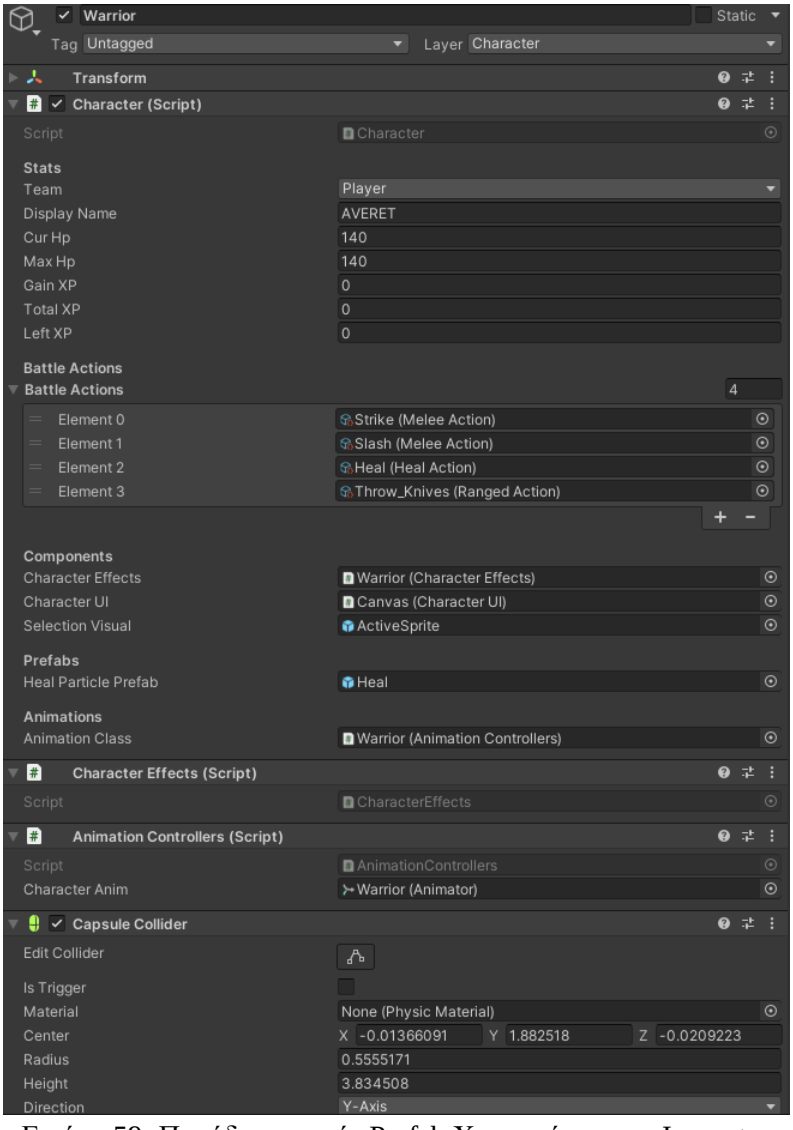
5. Game Development

5.1 Υλοποίηση Κώδικα

5.1.1 Character

5.1.1.1 Character Script

Ένας χαρακτήρας, είναι μία οντότητα που μπορεί να προκαλέσει ζημιά (Damage Attack), να θεραπεύσει (Heal) και να καταστραφεί (Die). Ελέγχεται είτε από τον παίκτη (Player), είτε από το εχθρικό AI (Enemy). Κάθε μάχη αποτελείται από τον παίκτη και τον εχθρό, όπου διαλέγουν ενέργειες μάχης (Battle Actions). Όλοι αυτοί οι χαρακτήρες λοιπόν, χρησιμοποιούν το ίδιο script, το οποίο μας επιτρέπει να προσθέσουμε ή να αφαιρέσουμε λειτουργίες πολύ εύκολα για όλους τους χαρακτήρες. Στην ουσία είναι από τα πιο σημαντικά κομμάτια κώδικα της εργασίας. (βλ. Παράρτημα 1)



Εικόνα 59: Παράδειγμα ενός Prefab Χαρακτήρα στον Inspector

5.1.1.2 Character Data Script

Για τα δεδομένα των χαρακτήρων που αποθηκεύονται και μεταφέρονται από σκηνή σε σκηνή, δημιουργήθηκαν τρία μικρά scripts τύπου Scriptable Object. Το CharacterSet.cs, επιτρέπει την αποθήκευση σετ εχθρών για κάθε μάχη. Το PlayerPersistentData.cs, όπου είναι τα δεδομένα των παικτών και μεταφέρονται μεταξύ των μαχών που εξελίσσονται και το PlayerPersistentCharacter.cs, το οποίο κρατά τις πληροφορίες που χρειάζεται να είναι αποθηκευμένες. Αυτά τα τρία scripts είναι και η βάση του level up των παικτών, που θα δούμε αναλυτικά παρακάτω.

```
using UnityEngine;

[CreateAssetMenu(fileName = "Character Set", menuName = "New Character Set")]
Cristina Mpalla *
public class CharacterSet : ScriptableObject
{
    //the enemy characters that our players will encounter at each level
    public GameObject[] characters;
}
```

Εικόνα 60: CharacterSet.cs

```
using UnityEngine;

[CreateAssetMenu(fileName = "PlayerPersistentData", menuName = "New Player Persistent Data")]
Cristina Mpalla *
public class PlayerPersistentData : ScriptableObject { //the scriptable which will store the persistent data for all players characters

    public PlayerPersistentCharacter[] characters;
    Cristina Mpalla *
    public void OnValidate() {
        ResetCharacters();
    }

    Cristina Mpalla *
    public void ResetCharacters() {
        for (int i = 0; i < characters.Length; i++) {
            //we use try get component to be safe in case some of the prefabs dont have the character component
            if (characters[i].characterPrefab.TryGetComponent<Character>(out var character)) {
                characters[i].health= characters[i].characterPrefab.GetComponent<Character>().maxHp;
                characters[i].totalXP= characters[i].characterPrefab.GetComponent<Character>().totalXP;
                characters[i].leftXP= characters[i].characterPrefab.GetComponent<Character>().leftXP;

            }
        }
    }
}
```

Εικόνα 61: PlayerPersistentData.cs

```
using UnityEngine;

[System.Serializable]
Cristina Mpalla *
public class PlayerPersistentCharacter //the class for the players characters data
{
    public GameObject characterPrefab;
    public int health; //leveled up health of the characters
    public int totalXP; //the total xp amount
    public int leftXP; //the xp that didn't give a level up and are left to be added
}
```

Εικόνα 62: PlayerPersistentCharacter.cs

5.1.1.3 Character UI Script

Το CharacterUI.cs, περιέχει το όνομα του χαρακτήρα και τα health points του. Στατιστικά, τα οποία εμπερικλείονται σε έναν διάφανο καμβά, όπου περιστρέφεται κατά τη διάρκεια της μάχης, ώστε πάντα να κοιτάει την κάμερα, ακόμα και αν ο χαρακτήρας είναι στο πλάι. Επίσης φαίνεται και ποιος χαρακτήρας έχει έρθει η σειρά του, με ένα πλαίσιο που εμφανίζεται γύρω από το όνομα του κάθε φορά.

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
 Cristina Mpalla
public class CharacterUI : MonoBehaviour {

    public TMP_Text characterNameText;

    public Image healthSlider;
    public Image healthFill;
    public Image turnVisual;

    public TMP_Text healthText;
     Cristina Mpalla
    private void Update() {
        //the stats canvas above the character, faces the camera on every frame
        transform.forward = transform.position - Camera.main.transform.position;
    }

    //the visual feedback we take when its a characters turn
     Cristina Mpalla
    public void ToggleTurnVisual(bool toggle) {
        turnVisual.gameObject.SetActive(toggle);
    }

     Cristina Mpalla
    public void SetCharacterNameText(string characterName) {
        characterNameText.text = characterName;
    }

    //the health bar updates with the fill amount
     Cristina Mpalla
    public void UpdateHealthBar(int curHp, int maxHp) {

        if (curHp < 0)
            healthText.text = $"{0} / {maxHp}";
        else
            healthText.text = $"{curHp} / {maxHp}";
        healthFill.fillAmount = (float)curHp / (float)maxHp;
    }

     Cristina Mpalla
    public void DeactivateUI() {
        characterNameText.enabled = false;
        healthFill.enabled = false;
        turnVisual.enabled = false;
        healthSlider.enabled = false;
    }
}
```

Εικόνα 63: CharacterUI.cs

5.1.2 Battle Actions

Τα Battle Actions, είναι όλες οι ενέργειες που μπορούν οι χαρακτήρες να εκτελέσουν. Είτε αυτό σημαίνει ενέργεια στον εαυτό τους, είτε σε άλλους, συμπαίκτες και εχθρούς. Τέτοιες ενέργειες είναι, επιθέσεις σε κοντινή απόσταση (Melee Attacks), επιθέσεις με εμβέλεια (Ranged Attacks), θεράπευση (Heal), αλλά και εφέ (Effects) που στην ουσία προέρχονται από Ranged Attacks.

Το script BattleAction.cs, είναι η βάση και ορίζει την κοινή λειτουργία που θα έχουν τα διαφορετικά αυτά, είδη ενεργειών. Αξίζει να σημειωθεί πως η κλάση αυτή, μιας και είναι η βάση όλων, είναι αφηρημένη (Abstract). Αυτό σημαίνει πώς μπορεί να κληρονομηθεί από άλλες κλάσεις και έτσι να δημιουργηθούν περιπτώσεις (Instances), των τεσσάρων μη αφηρημένων κλάσεων που ακολουθούν. Με αυτό τον τρόπο, το παιχνίδι μπορεί πολύ εύκολα να γίνει expandable, όσον αφορά τις ενέργειες των παικτών. Γενικά η χρήση των Scriptable Objects, όπως έχει αναφερθεί παραπάνω, κάνει τα παιχνίδια περισσότερο modular.

```
using UnityEngine;

Cristina Mpalla
public abstract class BattleAction : ScriptableObject
{
    //the sprite of each players action that will be appeared on each button
    public Sprite imageSprite;

    //define the abstract class. making this abstract means that every extending class
    //must implement it in its own way
    Cristina Mpalla
    public abstract void Cast(Character caster, Character target);
}
```

Εικόνα 64: BattleAction.cs

5.1.2.1 Melee Action Script

Οι επιθέσεις τύπου σώμα με σώμα (Melee Attack), χρησιμοποιούνται σε μάχες από κοντά και περιλαμβάνουν χτυπήματα, όπως μπουνιές, μαχαιρώματα και άλλες φυσικές μεθόδους επίθεσης σε έναν αντίπαλο.

```
using UnityEngine;

[CreateAssetMenu(fileName = "Melee Action", menuName = "Battle Actions/Melee Action")]
public class MeleeAction : BattleAction //melee actions are the ones that are close-quarters damage dealt to a character
{
    public int minMeleeDamage;//how much damage the action will have
    public int maxMeleeDamage;

    //when the caster has moved to the target and invoked the arriveCallback, the OnDamageTargetCallback will be called
    public override void Cast(Character caster, Character target) {
        caster.MoveToTarget(target, OnDamageTargetCallback);
    }

    void OnDamageTargetCallback(Character target)
    {
        int meleeDamage = Random.Range(minMeleeDamage, maxMeleeDamage);
        target.TakeDamage(meleeDamage);
    }
```

Εικόνα 66: MeleeAction.cs

5.1.2.2 Heal Action Script

Η θεράπευση (Heal), είναι μία ενέργεια που πραγματοποιείται για την αποκατάσταση/βελτίωση της υγείας σε ένα στόχο, άτομο ή αντικείμενο.

```
using UnityEngine;

[CreateAssetMenu(fileName = "Heal Action", menuName = "Battle Actions/Heal Action")]
Cristina Mpalla
public class HealAction : BattleAction //an action to heal yourself or a teammate
{
    public int minHealAmount;
    public int maxHealAmount;
    Cristina Mpalla
    public override void Cast(Character caster, Character target)
    {
        int healAmount = Random.Range(minHealAmount, maxHealAmount);
        target.Heal(healAmount);
    }
}
```

Εικόνα 65: HealAction.cs

5.1.2.3 Ranged Action Script

Μια επίθεση εμβέλειας (Ranged Attack), είναι η επίθεση, όπου ο χαρακτήρας χρησιμοποιεί κάποιο όπλο ή ρίχνει κάποιο ξόρκι από απόσταση προς τον στόχο του. Τέτοιες επιθέσεις είναι ένας πυροβολισμός, ρίψη βέλων, ρίψη μαγικού και άλλα πολλά.

```
using UnityEngine;

[CreateAssetMenu(fileName = "Ranged Action", menuName = "Battle Actions/Ranged Action")]
Cristina Mpalla
public class RangedAction : BattleAction //is an action from away like shooting an arrow
{
    public GameObject projectilePrefab;

    //if there is no caster set, return from that function as we need a caster to set the initial position
    Cristina Mpalla
    public override void Cast(Character caster, Character target) {
        if (caster == null)
            return;

        //instantiate the projectile at the middle of the caster (its hands for example)
        //and then initialize the projectile with the target character
        GameObject projectile = Instantiate(projectilePrefab, position: caster.transform.position + new Vector3( x: 0,  y: 2f,  z: 0), Quaternion.identity);
        projectile.GetComponent<Projectile>().Initialize(target);
    }
}
```

Εικόνα 67: RangedAction.cs

5.1.2.3 Effect Action Script

Η ενέργεια ενός εφέ, στην ουσία λειτουργεί όπως το Ranged Action, με μερικές διαφορές. Είναι η ρίψη αποκλειστικά ενός μαγικού ξορκιού προς τον στόχο, αλλά το ξόρκι παραμένει και απενεργοποιείται μετά από ένα συγκεκριμένο χρονικό διάστημα το οποίο θέτεται στο script CharacterEffects.cs. Το script αυτό, διαχειρίζεται όλα τα εφέ του κάθε χαρακτήρα.

```
using UnityEngine;

[CreateAssetMenu(fileName = "Effect Action", menuName = "Battle Actions/Effect Action")]
Cristina Mpalla *
public class EffectAction : BattleAction //an action that will affect a character for some seconds
{
    public Effect effectToCast;
    public bool canEffectSelf;
    public bool canEffectTeam;
    public bool canEffectEnemy;

    Cristina Mpalla
    public override void Cast(Character caster, Character target) {
        target.characterEffects.AddNewEffect(effectToCast);
    }
}
```

Εικόνα 68: EffectAction.cs

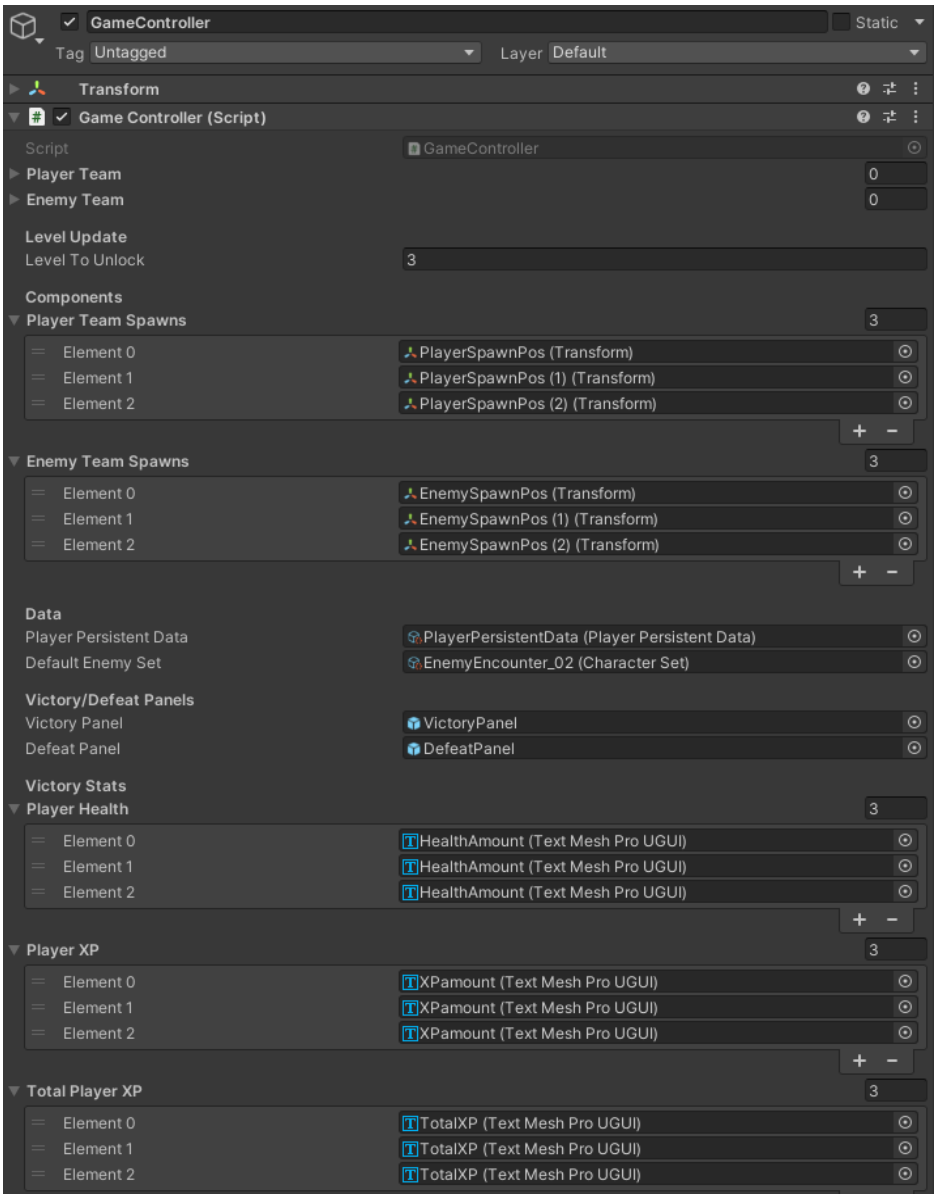
```
using System.Collections.Generic;
using UnityEngine;

Cristina Mpalla
public class CharacterEffects : MonoBehaviour //this will handle all the effects on the characters
{
    private List<EffectInstance> curEffects = new List<EffectInstance>(); //to keep track of the current effects
    private Character character; //the target character for the effects
    private int particleLifetime = 10;

    Cristina Mpalla
    private void Awake() {
        character = GetComponent<Character>();
    }

    //this function will add a new effect to the list of the effects
    //if the effect has the active prefab then instantiate it and save the reference in the effect instance var
    Cristina Mpalla
    public void AddNewEffect(Effect effect) {
        EffectInstance effectInstance = new EffectInstance(effect);

        if (effect.activePrefab != null)
            effectInstance.curActiveGameObject = Instantiate(effect.activePrefab, transform);

        curEffects.Add(effectInstance);
        ApplyEffect(effectInstance);
    }

    //this function will apply a single effect to the character
    Cristina Mpalla
    void ApplyEffect(EffectInstance effectIns) {

        //if the effect is damage type, make the character take damage
        if (effectIns.effect as DamageEffect) {

            var damageEffectInstance = (DamageEffect)effectIns.effect;
            int damageAmount = Random.Range(damageEffectInstance.minDamage, damageEffectInstance.maxDamage);

            if (damageEffectInstance != null)
                character.TakeDamage(damageAmount);
        }
        RemoveEffect(effectIns);
    }

    //this function will remove an effect from the list of the effects and also destroy the game object
    Cristina Mpalla
    void RemoveEffect(EffectInstance effectIns) {
        if(effectIns.curActiveGameObject != null)
            Destroy(effectIns.curActiveGameObject, particleLifetime);

        curEffects.Remove(effectIns);
    }
```

Εικόνα 69: CharacterEffects.cs

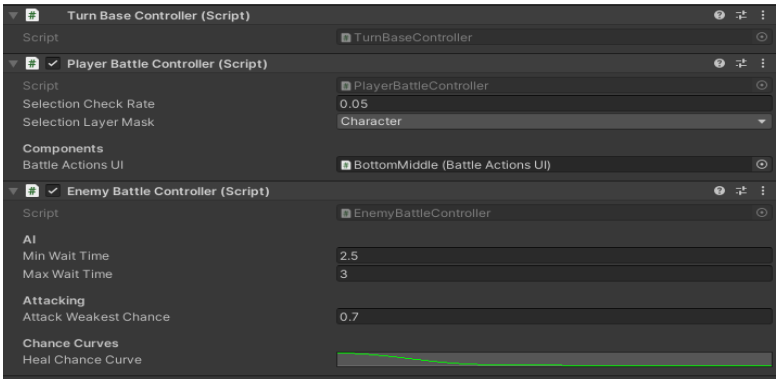
5.1.3 Controllers

5.1.3.1 Game Controller Script

Το script του Game Controller, είναι από τα πιο σημαντικά και θεμελιώδη κομμάτια κώδικα του παιχνιδιού, μιας και είναι υπεύθυνο για να παρακολουθεί την κατάσταση του κάθε χαρακτήρα και της μάχης. Επίσης κρατά την πληροφορία του level up για τους χαρακτήρες αλλά και για το επόμενο επίπεδο που πρόκειται να ξεκλειδωθεί. Επιπλέον περιλαμβάνει τα επόμενα τρία controllers για τους κεντρικούς χαρακτήρες, τους εχθρικούς, καθώς και το controller για την turn-based λογική. Αξίζει να σημειωθεί πως, το script αυτό, είναι μία Singleton κλάση, που σημαίνει ότι υπάρχει μία μόνο οντότητα του, διαθέσιμη από οπουδήποτε σε ολόκληρο το πρότζεκτ, γεγονός πολύ χρήσιμο, μιας και το Game Controller αναφέρεται σε αρκετά διαφορετικά κομμάτια κώδικα.(βλ. Παράρτημα 2)



Εικόνα 70: Game Controller Inspector



Εικόνα 70: Game Controller Inspector (Συνέχεια)

5.1.3.2 Turn-Base Controller Script

Το Turn-Base Controller script, είναι υπεύθυνο για να παρακολουθεί ποιας ομάδας έχει έρθει η σειρά, και πιο εσωτερικά, ποιος χαρακτήρας της κάθε ομάδας έχει σειρά. Είναι προκαθορισμένο η σειρά της ομάδας του χρήστη να ξεκινάει πρώτη και στη συνέχεια εναλλάσσονται οι σειρές που παίρνει η κάθε ομάδα, με τη βοήθεια ελέγχου για το ποια ομάδα είναι ενεργή τη δεδομένη στιγμή. Η κλάση αυτή είναι Singleton, που καθιστά την μοναδική της οντότητα ανιχνεύσιμη από όλα τα αρχεία του πρότζεκτ. (βλ. Παράρτημα 3)

5.1.3.3 Player Battle Controller Script

Αυτό το script, αφορά αποκλειστικά το σύστημα μάχης των κεντρικών χαρακτήρων. Αφού έχουμε βεβαιωθεί πως ο χαρακτήρας που είναι ενεργός δεν είναι εχθρικός, ο χρήστης μπορεί να επιλέξει την ενέργεια του.

Η διαδικασία είναι η εξής, με το που έρθει η σειρά του κάθε χαρακτήρα, εμφανίζεται το panel με τις διαθέσιμες ενέργειες που έχει να κάνει. Η κάθε ενέργεια έχει ένα flag για τον στόχο που προορίζεται. Για παράδειγμα ο ενεργός χαρακτήρας, μπορεί να επιλέξει τη θεράπευση για τον εαυτό του ή κάποιον από την ομάδα του, όμως όχι για τους εχθρούς. Οι επιθέσεις είναι αυτές που γίνονται αποκλειστικά και μόνο για εχθρικούς χαρακτήρες. Με το που επιλεχθεί η ενέργεια, αλλά και ο στόχος στον οποίο προορίζεται, ο χαρακτήρας εκτελεί την κίνηση και το παιχνίδι προχωρά στον επόμενο. (βλ. Παράρτημα 4)

5.1.3.4 Enemy Battle Controller Script

Οι εχθροί ελέγχονται από το AI, οπότε όλες οι αποφάσεις που παίρνουν γίνονται στη στιγμή. Για να φαίνεται πιο ρεαλιστικό το σύστημα μάχης τους, προστέθηκαν καθυστερήσεις για τις αποφάσεις που παίρνουν. Οι ενέργειες των εχθρών βρίσκονται σε μία λίστα εντός του κώδικα, κρυφά από τον χρήστη, σε αντίθεση με το panel που εμφανίζεται στους κεντρικούς χαρακτήρες.

Όσον αφορά τις επιθετικές κινήσεις, οι στόχοι των εχθρών διακρίνονται σε δύο κατηγορίες, τους κεντρικούς χαρακτήρες με τα λιγότερα health points, αλλά και τυχαίους κεντρικούς χαρακτήρες. Για τον τελικό στόχο, έχει τεθεί μία μεταβλητή, όπου κατά ένα

μεγάλο ποσοστό ο εχθρός επιλέγει να επιτεθεί στον κεντρικό χαρακτήρα με τη λιγότερη υγεία, ενώ το υπόλοιπο ποσοστό κάνει τον εχθρό να επιλέξει να επιτεθεί σε έναν τυχαίο χαρακτήρα.

Όσον αφορά την θεράπευση, η διαδικασία είναι η ίδια όπως και στους κεντρικούς χαρακτήρες, με τη μόνη διαφορά ότι γίνεται στον ίδιο ή σε κάποιον από την ομάδα του βάσει ενός Curve. Τα Curves, είναι εργαλεία της Unity, που επιτρέπουν στους προγραμματιστές να ορίσουν μία σειρά από τιμές για τον έλεγχο της συμπεριφοράς ενός αντικειμένου με την πάροδο του χρόνου. Έτσι και σε αυτή την περίπτωση το Heal Curve, βρίσκει την πιθανότητα θεράπευσης για τα health points οποιουδήποτε χαρακτήρα στην ομάδα του εχθρού.(βλ. Παράρτημα 5)

5.1.4 Σημαντικά UI Scripts

5.1.4.1 Battle Action UI Script

Το script αυτό, αντικατοπτρίζει τις ενέργειες που έχουν οι χαρακτήρες κατά τη διάρκεια του παιχνιδιού. Χρησιμοποιείται μόνο για τις ενέργειες του χρήστη, μιας και οι ενέργειες του εχθρού δεν εμφανίζονται στη διεπαφή. Ανάλογα, λοιπόν τον χαρακτήρα, που είναι η σειρά του να κάνει κάποια όσο διαρκεί η μάχη, εμφανίζεται ο ανάλογος αριθμός κινήσεων που έχει διαθέσιμες, με τη μορφή κουμπιών. Με το πάτημα τους, τα κουμπιά εκείνου του χαρακτήρα απενεργοποιούνται και εμφανίζονται οι ενέργειες-κουμπιά του επόμενου, σε σειρά, κεντρικού χαρακτήρα.

```
using UnityEngine;
Cristina Mpalla *
public class BattleActionsUI : MonoBehaviour {
    public GameObject panel;
    public BattleActionButton[] buttons;

    //subscribe to the onNewTurn event
    Cristina Mpalla
    void OnEnable() {
        TurnBaseController.Instance.onNewTurn += OnNewTurn;
    }

    //unsubscribe from that event
    Cristina Mpalla
    void OnDisable() {
        TurnBaseController.Instance.onNewTurn -= OnNewTurn;
    }

    //this will listen to the onNewTurn event, if it is the players turn the battle actions will be displayed, otherwise disable it when its enemy's turn
    Cristina Mpalla
    void OnNewTurn() {
        if (TurnBaseController.Instance.GetCurrentTurnCharacter().team == Character.Team.Player){
            panel.SetActive(true);
            DisplayBattleActions(TurnBaseController.Instance.GetCurrentTurnCharacter());
        }
        else
            DisableBattleActions();
    }

    //this function will display the battle actions of a character, it iterates over the buttons and enable as many of them as the character has as battle actions
    Cristina Mpalla
    public void DisplayBattleActions(Character character) {
        panel.SetActive(true);

        for (int i = 0; i < buttons.Length; i++) {
            if (i < character.battleActions.Length) {
                buttons[i].gameObject.SetActive(true);
                buttons[i].SetBattleAction(character.battleActions[i]);
            }
            else
                buttons[i].gameObject.SetActive(false);
        }
    }

    //this function will disable the battle actions panel
    Cristina Mpalla
    public void DisableBattleActions() {
        panel.SetActive(false);
    }
}
```

Εικόνα 71: BattleActionsUI.cs

5.1.4.2 Loading Scenes Script

Το Loading Scenes script, εφαρμόζεται για τη μετάβαση από σκηνή σε σκηνή με μία μπάρα φόρτωσης, η οποία γεμίζει με τη χρήση ασύγχρονης διαδικασίας μέσα σε IEnumerator, μέσω της μαθηματικής συνάρτησης Mathf.Clamp01. Η συνάρτηση αυτή φροντίζει η επιστρεφόμενη τιμή της να μην βγαίνει εκτός των ορίων 0 και 1. Επίσης, με την φόρτωση της κάθε σκηνής, ενεργοποιείται το animation Fade In και Fade Out, αν ανοίγει ή κλείνει η σκηνή αντίστοιχα. Έτσι, οι μεταβάσεις δεν φαίνονται γρήγορες και απότομες και παρέχεται στον χρήστη το αίσθημα προόδου.[12]

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

public class LoadingScenes : MonoBehaviour
{
    public GameObject loadingScreen;
    public GameObject currentScene;

    public Image loadingBarFill;
    public Image black;

    public Animator anim;

    private string fadeTrigger = "Fade";

    public void LoadScene(int sceneId) {
        StartCoroutine( routine: LoadSceneAsync(sceneId));
    }

    //this enumerator fades in and out when opening new scenes and also
    //it has the async operation for the loading process
    IEnumerator LoadSceneAsync(int sceneId) {

        anim.SetBool( name: fadeTrigger,  value: true);
        yield return new WaitUntil(() => black.color.a == 1);

        AsyncOperation operation = SceneManager.LoadSceneAsync(sceneId);
        loadingScreen.SetActive(true);
        currentScene.SetActive(false);

        while (!operation.isDone) {
            float progressValue = Mathf.Clamp01(operation.progress / 0.9f);
            loadingBarFill.fillAmount = progressValue;

            yield return null;
        }
    }
}
```

Εικόνα 72: LoadingScenes.cs

5.1.4.3 Level Selector Script

Το Level Selector script, καλείται κάθε φορά στο τέλος μίας μάχης. Αν έχει επέλθει νίκη, τότε το επόμενο επίπεδο, από αυτό που βρίσκεται ο παίκτης εκείνη τη στιγμή, ξεκλειδώνεται, με το αντίστοιχο κουμπί να ενεργοποιείται και να εξαφανίζεται το εικονίδιο του λουκέτου. Η πληροφορία αποθηκεύεται σε μία μεταβλητή η οποία διατηρείται καθ' όλη τη διάρκεια του παιχνιδιού. Διαγράφεται μόνο όταν ο χρήστης επιλέξει να ξεκινήσει από την αρχή όλα τα επίπεδα. [11]

```
using UnityEngine;
using UnityEngine.UI;

public class LevelSelector : MonoBehaviour
{
    public Image[] lockedSprite;
    public Button[] levelButtons;

    private string lvlReached = "levelReached";

    //whenever this is called, the next level of the current is unlocked,
    //with the buttons interractable again
    private void Start() {
        int levelReached = PlayerPrefs.GetInt( key: lvlReached, defaultValue: 1);

        for (int i = 0; i < levelButtons.Length; i++) {
            if (i + 1 > levelReached) {
                levelButtons[i].interactable = false;
                lockedSprite[i].enabled = true;
            }
            else
                lockedSprite[i].enabled = false;
        }
    }
}
```

Εικόνα 73: LevelSelector.cs

5.1.5 Audio Script

Η λειτουργία του Audio Script, είναι στην ουσία να σώζει την πληροφορία που παίρνει από τα δύο sliders για την ένταση των ήχων του παιχνιδιού (βλ. Εικόνα 25). Η πληροφορία αυτή, παραμένει αποθηκευμένη ακόμα και μετά την έξοδο από το παιχνίδι. Διαγράφεται και επαναφέρεται στις προκαθορισμένες τιμές, όταν ο χρήστης επιλέξει νέο παιχνίδι. [9] (βλ. Παράρτημα 6)

5.2 Βελτιστοποίηση Απόδοσης (Performance Optimization)

Η βελτιστοποίηση της απόδοσης γενικά στα παιχνίδια είναι ιδιαίτερα σημαντική για πολλούς λόγους. Όλα τα παιχνίδια, πρέπει να τρέχουν ομαλά, ανεξαρτήτως hardware, ειδικά σε υλικά χαμηλής ποιότητας και κινητές συσκευές. Η κακή απόδοση, οδηγεί τις πλείστες φορές σε κακή εμπειρία χρήστη, γεγονός που επηρεάζει αρνητικά το παιχνίδι και την ομάδα δημιουργίας του. Έχοντας ως παράδειγμα τη σκηνή από το δεύτερο επίπεδο, θα αναλυθεί η διαδικασία που πραγματοποιήθηκε για να εκτελείται πιο ομαλά και αποτελεσματικά η ροή του παιχνιδιού.

Παρακάτω (βλ. Εικόνα 73), φαίνονται τα στατιστικά του παιχνιδιού την ώρα που τρέχει. Ο πίνακας αυτός της Unity Editor, παρέχει πληροφορίες σχετικά με την απόδοση του παιχνιδιού, ανάλογα τη σκηνή που τρέχει εκείνη την ώρα. Εμφανίζει τα καρέ ανά δευτερόλεπτο (Frames Per Second – FPS), τον αριθμό Draw Calls, τον αριθμό των Batches, και των αριθμό των τριγώνων (Tris). Πάνω σε αυτά τα στοιχεία θα επικεντρωθούμε για να βελτιωθεί η απόδοση του παιχνιδιού. Αξίζει να σημειωθεί, πως επειδή υπάρχουν τρεις σκηνές επιπέδων, η διαδικασία αυτή επαναλήφθηκε τρεις φορές για να βελτιστοποιηθεί όλο το παιχνίδι ομοιόμορφα με τον ίδιο τρόπο. Περιληπτικά, για να κατανοηθούν τα παραπάνω στοιχεία, θα αναφερθούμε στη λειτουργία και ορολογία τους. [8]

Τα FPS, είναι μία μέτρηση της ποιότητας των γραφικών σε ένα παιχνίδι σε καρέ ανά δευτερόλεπτο. Όσο υψηλά είναι τα FPS, τόσο ομαλό είναι το παιχνίδι.

Τα Draw Calls, είναι ο αριθμός των εντολών της κάρτας γραφικών, που του λένε να σχεδιάσει ένα αντικείμενο σε μια 3D σκηνή. Είναι σημαντικό λοιπόν, οι εντολές αυτές να ελαχιστοποιηθούν όσο το δυνατόν περισσότερο.

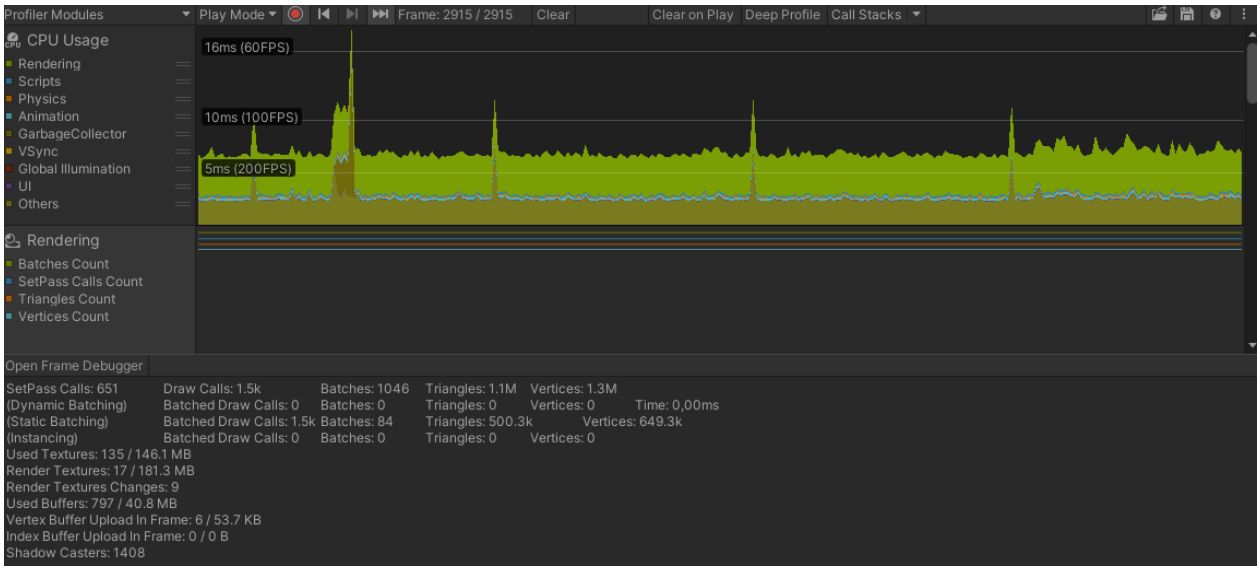
Τα Batches, εμφανίζουν τον αριθμό των Draw Calls ανά καρέ, καθώς και τα Game Objects με τα materials τους που αποδίδονται.

Τα Tris, ή αλλιώς τρίγωνα, είναι το βασικό δομικό στοιχείο των 3D μοντέλων στη Unity. Η τιμή αυτή, υποδεικνύει τον συνολικό αριθμό τους με βάση τα μοντέλα που υπάρχουν εκείνη την στιγμή στη σκηνή.

Εικόνα 74: Στατιστικά Παιχνιδιού, ενδεικτικά στο 2ο Επίπεδο

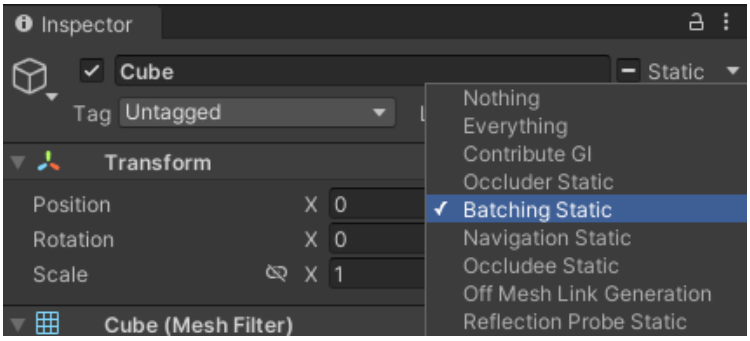
Συμπεραίνουμε πως ο πίνακας στατιστικών είναι χρήσιμος για τη βελτιστοποίηση της απόδοσης του παιχνιδιού, μιας και βοηθάει στον εντοπισμό περιοχών που χρειάζονται βελτίωση. Όμως το βασικό εργαλείο της Unity, σε συνδυασμό με τον πίνακα αυτό, είναι ο Profiler, αφού μας δείχνει ακριβώς τι είναι αυτό που δημιουργεί το πρόβλημα στην απόδοση.

Πιο συγκεκριμένα, αυτό που δίνουμε περισσότερη προσοχή είναι στη χρήση της CPU, όπως βλέπουμε στο αριστερό κομμάτι του Profiler (βλ. Εικόνα 75). Εκεί, μπορούμε να δούμε, μέσω του chart, ακριβώς πόσος χρόνος ξοδεύτηκε σε κάθε είδος task, Scripts, Rendering, Physics, Animation, GarbageCollector, VSync, Global Illumination, UI και άλλα. Βλέποντας το chart, παρατηρούμε πως το Rendering είναι αυτό που χαμηλώνει την ποιότητα της απόδοσης. Οπότε στο αριστερό κομμάτι υπάρχει ένα profiler module αποκλειστικά για το Rendering. Με το πάτημα εμφανίζεται ο Frame Debugger στο κάτω μέρος, ο οποίος μας παρουσιάζει, πολύ πιο αναλυτικά από τον πίνακα στατιστικών που είδαμε παραπάνω, το που υπάρχει πρόβλημα. Εδώ θα επικεντρωθούμε καθ' όλη τη διάρκεια του optimization, για πιο στοχευμένη βελτιστοποίηση και έτσι καλύτερα αποτελέσματα.



Εικόνα 75: Profiler Window

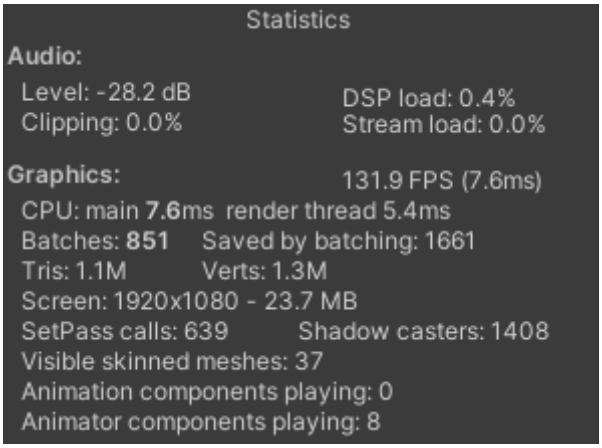
Η πρώτη κίνηση για το Optimization, είναι η επιλογή Static Batching σε αντικείμενα στη σκηνή. Τα αντικείμενα αυτά, όμως, πρέπει να παραμένουν σταθερά στη σκηνή και να μην έχουν κάποια κίνηση.



Εικόνα 76: Επιλογή Static Batching

Επιπλέον, ένας πολύ σημαντικός παράγοντας που συμβάλει στην απόδοση είναι ο φωτισμός (βλ. Κεφ. 4.4). Υπάρχουν δύο ήδη φωτισμών, Real-Time Lighting και Baked Lighting. Επέλεξα τον Baked Lighting για όλες τις σκηνές του παιχνιδιού, μιας και χρησιμοποιείται καλύτερα με στατικά αντικείμενα.

Οπότε αυτό το βήμα, σε συνδυασμό με το προηγούμενο του Static Batching μας δίνει το αποτέλεσμα παρακάτω (βλ. Εικόνα 77). Όπως βλέπουμε τα FPS, διπλασιάστηκαν, ενώ τα batches μειώθηκαν πάρα πολύ.



Εικόνα 77: Στατιστικά Παιχνιδιού (Μετά από το πρώτο Optimization)

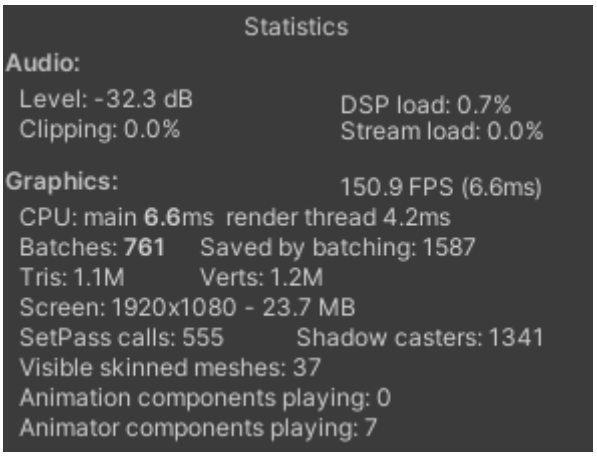
Συνεχίζοντας το optimization του παιχνιδιού, για τα shaders των materials, επιλέχθηκαν τα mobile shaders, αντί των standard που ήταν προκαθορισμένα. Τα mobile shaders, παρέχουν καλύτερη απόδοση στις κινητές συσκευές, προσφέρουν χαρακτηριστικά όπως καλύτερο φωτισμό και καλύτερο batching, όπως βλέπουμε στην Εικόνα 78 παρακάτω. Όσον αφορά τα materials, μια ακόμη σημαντική βελτιστοποίηση έγινε από την πλευρά των υφών (textures).

Τα textures, χρησιμοποιούνται για να δώσουν στα αντικείμενα μία οπτική εμφάνιση. Εφαρμόζονται σε materials, για μεγαλύτερο ρεαλισμό και λεπτομέρεια. Υπάρχουν διαφορετικά ήδη 3D textures. Τα Base Color, περιέχουν το βασικό χρώμα του αντικειμένου, τα Normal Map, που περιλαμβάνουν τις λεπτομέρειες της κάθε επιφάνειας του αντικειμένου, όπως εξογκώματα, γρατσουνιές και αυλακώματα. Επίσης, τα Specular και Metallic Textures, που παρέχουν έναν τρόπο ελέγχου της αντανάκλασης μιας επιφάνειας ορίζοντας τις μεταλλικές τιμές της. Με λίγα λόγια πόσο μεταλλική είναι αυτή η επιφάνεια.

Αυτό που εφαρμόστηκε, με βάση τα textures, είναι η μείωση των μεγεθών όλων των textures αρχείων, μιας και στη Unity, αποθηκεύονται ως εικόνες με προκαθορισμένο μέγεθος περίπου στα 2 με 4 Giga Bytes. Σε μικρά αντικείμενα, μείωσα τα αντίστοιχα textures σε 512 Mega Bytes, για τα Base Colors και Normal Maps, ενώ για τα Specular και Metallic Textures, μείωσα το μέγεθος στα 256 Mega Bytes. Από την άλλη πλευρά, σε μεγαλύτερα αντικείμενα το μέγεθος μειώθηκε στα 1024 Mega Bytes για τα Base Color textures και Normals, και για τα Specular και Metallic στα 256 Mega Bytes, ανεξαρτήτως μεγέθους.

Μειώνοντας λοιπόν κατ' αυτόν τον τρόπο τα textures, καταφέραμε καλύτερο optimization, όπως φαίνεται στην Εικόνα 78, και τα γραφικά φαίνεται να μην έχουν αλλάξει. Τα μικρότερα σε μέγεθος, textures, απαιτούν λιγότερη μνήμη, βοηθούν στο

χρόνο φόρτωσης, βελτιώνουν τα FPS, αλλά και τη χρήση μνήμης. Επιπλέον βοηθά στη μείωση του συνολικού μεγέθους που έχει το αρχείο του παιχνιδιού, εξοικονομώντας κόστος αποθήκευσης.



Εικόνα 78: Στατιστικά Παιχνιδιού (Μετά από το δεύτερο Optimization)

Επιπλέον optimization έγινε στη σκηνή και στα αντικείμενα που ήδη βρισκόταν σε αυτή. Πολλά από αυτά είχαν game objects που στην ουσία δεν χρειαζόταν και ήταν περιττά, όπως LODs και game objects με μόνο στοιχείο ένα collider.

Τα LOD (Level of Detail) game objects, είναι 3D αντικείμενα που χρησιμοποιούνται για τη βελτίωση της απόδοσης, μειώνοντας την ποσότητα της λεπτομέρειας σε μία σκηνή. Χρησιμοποιούνται για τη μείωση της ποσότητας πολυγώνων και textures, για μακρινά αντικείμενα, επιτρέποντας πιο λεπτομερή αντικείμενα πιο κοντά στην κάμερα. Τέτοιου είδους game objects, χρησιμοποιούνται σε παιχνίδια, στα οποία η κάμερα έχει κάποια κίνηση στο χώρο και ανάλογα την απόσταση, κάνει render το αντίστοιχο LOD. Στην περίπτωση του παρόντος παιχνιδιού όμως, από την στιγμή που η κάμερα είναι απολύτως σταθερή, δεν χρειάζονται τέτοια αντικείμενα. Αρκεί μόνο ένα, επιλέγοντας πάντα το κατάλληλο, βάσει του πόσο κοντά στην κάμερα βρίσκεται.

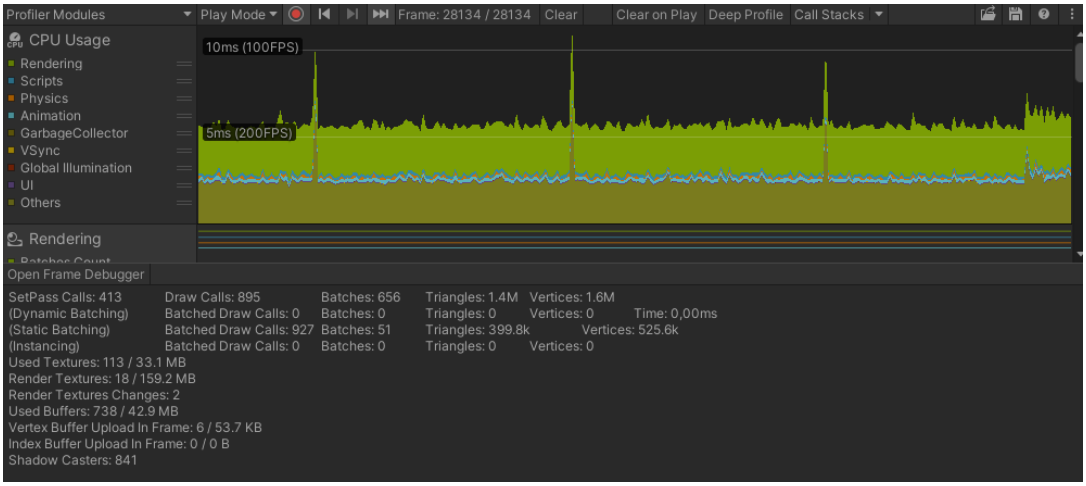
Συμπληρωματικά, game objects που είχαν collider, ως το μοναδικό στοιχείο τους, διεγράφησαν επίσης, μιας και δεν χρειαζόμαστε colliders για το παιχνίδι στα αντικείμενα. Είναι και ο λόγος που τα αντικείμενα είναι static. Δεν αλληλοεπιδρούν με τους κινούμενους χαρακτήρες κατά τη διάρκεια της μάχης, απλά βρίσκονται στον φόντο, ως μέρος της σχεδίασης του εκάστοτε επιπέδου.

Τελευταίο και σημαντικό βήμα του optimization ήταν και ο συνδυασμός των meshes.Ο συνδυασμός αυτός μπορεί να γίνει, μόνο μεταξύ αντικειμένων που μοιράζονται το ίδιο material και mesh. Με το βήμα αυτό, μειώνεται δραστικά ο αριθμός των draw calls, και έτσι βελτιώνεται η απόδοση. Στην ουσία ένα συνδυασμένο mesh, μεταφράζεται ως ένα draw call στην κάρτα γραφικών, γεγονός που εκτινάσσει το optimization.

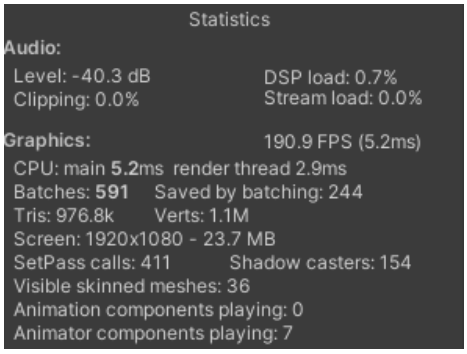
Για να κάνω αυτή τη διαδικασία, χρησιμοποίησα ένα εργαλείο της Unity Asset Store [29], όπου, πολύ απλά επιλέγουμε τα αντικείμενα που θέλουμε να συνδυάσουμε και με το πάτημα ενός κουμπιού έχουμε πλέον ένα μεμονωμένο αντικείμενο, απαρτισμένο από μικρότερα ίδια. Ένα έξτρα βήμα σε αυτό το σημείο, ήταν

η εξαγωγή του νέου, πλέον, mesh, εκτός της σκηνής, ώστε να μην γίνεται baked σε πραγματικό χρόνο, με αποτέλεσμα να αργεί η φόρτωση.

Έχοντας ολοκληρώσει το optimization του παιχνιδιού, έχουμε τα παρακάτω αποτελέσματα στον Profiler και στον πίνακα στατιστικών.



Εικόνα 79: Profiler Window (Optimized)



Εικόνα 80: Στατιστικά Παιχνιδιού (Optimized)

Τοποθετώντας τα αρχικά αποτελέσματα σε σχέση με τα τελικά, έχουμε τον εξής πίνακα σύγκρισης, πριν και μετά το optimization.

Απόδοση	FPS	Batches	Draw Calls	Tris
Πριν	68.8	1046	1.5 χιλ.	1.1 εκ.
Μετά	190.9	591	895	976.8 χιλ.

Πίνακας 1: Στατιστικά βελτίωσης της απόδοσης

Συμπερασματικά, η βελτιστοποίηση της απόδοσης ενός παιχνιδιού, διαφέρει από παιχνίδι σε παιχνίδι. Σημαντικό στοιχείο είναι η χρήση του Profiler, μιας και σε βοηθά να κατανοήσεις την πηγή του προβλήματος, κάνοντας στοχευμένες κινήσεις για την επίλυσή του. Επίσης σημαντικό είναι και η κατανόηση του στόχου για την απόδοση του παιχνιδιού. Για το δικό μου ήταν γύρω στα 200 FPS για τον υπολογιστεί, το οποίο αντισταθμίζεται με 40 FPS, στο κινητό, γεγονός που επετεύχθη.

6. Επίλογος

6.1 Συμπεράσματα

Μετά το πέρας της παρούσας πτυχιακής εργασίας, κατανοήθηκε εις βάθος η διαδικασία δημιουργίας ενός παιχνιδιού από το μηδέν. Ο σχεδιασμός και η υλοποίηση του "Corrupted Lands", έδειξε ότι η παιχνιδομηχανή της Unity είναι μια εξαιρετική πλατφόρμα για την ανάπτυξη παιχνιδιών όλων των ειδών, είτε είναι για κινητά, είτε για υπολογιστές. Η Unity παρέχει ισχυρά εργαλεία και σε συνδυασμό με την εύκολη σε χρήση διεπαφή, την καθιστά πρώτη στις προτιμήσεις των game developers και designers. Μέσω της εφαρμογής σωστού προγραμματισμού και σχεδιαστικών στοιχείων, όπως η εξέλιξη χαρακτήρων, το εχθρικό AI και η οργανωμένη διεπαφή χρήστη, μπορεί να δημιουργηθεί μια μοναδική και ολοκληρωμένη εμπειρία παιχνιδιού. Επιπλέον η χρήση στοιχείων και εργαλείων από το Unity Asset Store, βοηθά στην ανάπτυξη ενός παιχνιδιού, με ελκυστικό περιβάλλον, γραφικά, μουσική και εφέ, που καθηλώνουν τον χρήστη.

6.2 Δυσκολίες

Το "Corrupted Lands", είναι το πρώτο παιχνίδι που φτιάχνω εξ ολοκλήρου από την αρχή. Παρ' όλα αυτά το μεγαλύτερο πρόβλημα, που αντιμετώπισα εν τέλει, ήταν αυτό της βελτιστοποίησης της απόδοσης. Είναι ένα δύσκολο κομμάτι, το οποίο απαιτεί ιδιαίτερη προσοχή, καθ' όλη τη διάρκεια δημιουργίας του παιχνιδιού. Η πολυπλοκότητα του, προέρχεται από τους πολλούς παράγοντες και τις μεταβλητές που πρέπει να ληφθούν υπόψη. Αυτό μπορεί να περιλαμβάνει περιορισμούς hardware, τον αριθμό των στοιχείων που περιλαμβάνονται στις σκηνές και πολλά άλλα. Η διαδικασία της βελτιστοποίησης είναι μία εξαιρετικά χρονοβόρα διαδικασία και απαιτεί βαθιά κατανόηση της Unity, προκειμένου να επιτευχθούν τα επιθυμητά αποτελέσματα. Μετά, λοιπόν, από αρκετή έρευνα και διάβασμα, η χρήση του εργαλείου Profiler, ήταν και η λύση του προβλήματος, έχοντας ως αποτέλεσμα την ομαλή λειτουργία του παιχνιδιού και κατά συνέπεια μια ευχάριστη εμπειρία για τον χρήστη.

6.3 Μελλοντικές Βελτιώσεις και Επεκτάσεις

Υπάρχουν αρκετές βελτιώσεις που μπορούν να γίνουν μελλοντικά, ώστε να μπορεί το παιχνίδι να φτάσει σε κατάσταση παραγωγής.

Για την βελτίωση, η προσθήκη νέων επιπέδων και χαρακτήρων, για περισσότερες μάχες, είναι σίγουρα το πρώτο βήμα που θα έκανα. Αυτό θα κάνει το παιχνίδι πιο ενδιαφέρον, διασκεδαστικό και με διάρκεια. Τυχόν βελτιώσεις σε ηχητικά και οπτικά εφέ θα ήταν κάτι επιθυμητό, μιας και δεν υπάρχει ποικιλία στην παρούσα εργασία. Άλλη μία σημαντική βελτίωση είναι η μάχη αυτή καθ' αυτήν. Περισσότερες ενέργειες των χαρακτήρων, δεξιότητες, καθώς και ένα σύστημα διαλόγου για την επιπλέον διασκέδαση του χρήστη. Επιπλέον μια βελτίωση που θα έκανα, είναι μία που δεν υπάρχει στο παιχνίδι τώρα. Ένα σύστημα timer, για να προσθέσει μία γεύση έξτρα αγωνίας, όπου με την λήξη του timer, να επέλθει ήττα στην ομάδα του χρήστη.

Μπορεί να υπάρξουν περισσότερες βελτιώσεις στο παιχνίδι, όμως οι παραπάνω είναι σίγουρα οι πρώτες, που θα έκανα στο μέλλον.

Βιβλιογραφία και Πηγές

Για τη συγγραφή της πτυχιακής εργασίας:

[1] Jing-Wei Liu, Chia-Yu Ho, Jamie Y.T. Chang, Jacob Chia-An Tsai, *The role of Sprint planning and feedback in game development projects: Implications for game quality*, Journal of Systems and Software (2019)
[2] Unity (Game Engine), Wikipedia: https://en.wikipedia.org/wiki/Unity_(game_engine)
[3] Unity Documentation: https://docs.unity3d.com/Manual/index.html
[4] Patryk Galach (2019), *Design Patterns in Unity*.
https://www.patrykgalach.com/2019/05/06/design-patterns-in-unity/
[5] VOiD1 Gaming (2020), *10 Basic Principles of Game Design*.
https://www.void1gaming.com/post/10-basic-principles-of-game-design
[6] Mixamo, Wikipedia: https://en.wikipedia.org/wiki/Mixamo
[7] Cinemachine and Timeline Editor for Unity 2D Game Development, GameDev Academy: https://gamedevacademy.org/cinemachine-and-timeline-editor-for-unity-2d-game-development/
[8] Unity Documentation: https://docs.unity3d.com/Manual/RenderingStatistics.html
[9] Stack Overflow: https://stackoverflow.com/questions/37773729/making-volume-slider-in-unity
[10] Smart Draw: https://cloud.smartdraw.com/
[11] Stack Overflow: https://stackoverflow.com/questions/66795450/unity-load-unlock-levels-and-check-if-there-are-next-levels-existing
[12] Loading261: https://loading261.rssing.com/chan-29474081/latest.php

Για τη σχεδίαση του παιχνιδιού:

[13] Unity Particle Pack, Unity Technologies. Unity Asset Store:
https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-particle-pack-127325
[14] Dark Fantasy GUI / UI Kit, HONETi. Unity Asset Store:
https://assetstore.unity.com/packages/2d/gui/dark-fantasy-gui-ui-kit-68828
[15] The Big Castle Kit, Triplebrick. Unity Asset Store:
https://assetstore.unity.com/packages/3d/environments/historic/the-big-castle-kit-75818
[16] Alien Fantasy Forest, Manufactura K4. Unity Asset Store:
https://assetstore.unity.com/packages/3d/environments/alien-fantasy-forest-150615
[17] Aquarius Fantasy Series: Dark Elves, Aquarius Max. Unity Asset Store:
https://assetstore.unity.com/packages/3d/environments/fantasy/aquarius-fantasy-series-dark-elves-237879
[18] RPG Fantasy (Pack), Maksim Bugrimov. Unity Asset Store:
https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/rpg-fantasy-pack-47749
[19] Skybox Series Free, Avionx. Unity Asset Store:
https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633

[20] Textures Part 1, ALP8310. Unity Asset Store: *https://assetstore.unity.com/packages/2d/textures-materials/textures-part-1-122421*
[21] Minimal Icon Pack, nappin. Unity Asset Store: *https://assetstore.unity.com/packages/2d/gui/minimal-icon-pack-customizable-550-197164*
[22] AQUAS Lite, Dogmatic. Unity Asset Store: *https://assetstore.unity.com/packages/vfx/shaders/aquas-lite-built-in-render-pipeline-53519#description*
[23] 2000 Fantasy Icons, PONETI. Unity Asset Store: *https://assetstore.unity.com/packages/2d/gui/icons/2000-fantasy-icons-145876*
[24] RPG VFX Bundle, Hovl Studio. Unity Asset Store: *https://assetstore.unity.com/packages/vfx/particles/spells/rpg-vfx-bundle-133704*
[25] 100 Special Skills Effects Pack, GAPH. Unity Asset Store: *https://assetstore.unity.com/packages/vfx/particles/spells/100-special-skills-effects-pack-171146*
[26] Fantasy Game Sound Effects, Epic Sounds and FX. Unity Asset Store: *https://assetstore.unity.com/packages/audio/sound-fx/fantasy-game-sound-effects-94186*
[27] Colossal Game Music Collection, The Indie Devs Nation. Unity Asset Store: *https://assetstore.unity.com/packages/audio/music/orchestral/colossal-game-music-collection-88190*
[28] Cave Material, Vergil190202. Sketchfab: *https://sketchfab.com/3d-models/cave-material-88ca7e565a9c4f48a5306f4cc85e57fb*
[29] Easy Combine Mesh, ALlyerEdon, Unity Asset Store: https://assetstore.unity.com/packages/tools/utilities/easy-mesh-combine-tool-211915

Παραρτήματα

Παράρτημα 1: Character Script

```
using System.Collections;
using UnityEngine;
using UnityEngine.Events;

Cristina Mpalla *
public class Character : MonoBehaviour
{
    Cristina Mpalla
    public enum Team {
        Player,
        Enemy
    }

    [Header("Stats")]
    public Team team;
    public string displayName;
    public int curHp;
    public int maxHp;
    public int gainXP;
    public int totalXP;
    public int leftXP;

    [Header("Battle Actions")]
    public BattleAction[] battleActions;
    private BattleActionsUI battleActionsUI;

    [Header("Components")]
    public CharacterEffects characterEffects;
    public CharacterUI characterUI;
    public GameObject selectionVisual;

    [Header("Prefabs")]
    public GameObject healParticlePrefab;

    [Header("Animations")]
    public AnimationControllers animationClass;
    private Shake shake;

    private Vector3 standingPosition;

    private GamePlaySounds gamePlaySounds;

    public static UnityAction<Character> onCharacterDeath;

    //string based variables
    private string screenShakeTag= "ScreenShake";
    private string battleSoundsTag = "BattleSounds";
    private string dieAnimTrigger = "Died";
    private string walkingAnimTrigger = "Walking";
    private string meleeAnimTrigger = "MeleeAttack";
    private string walkBackAnimTrigger = "WalkingBack";
    private string gotHitAnimTrigger = "GotHit";
```

Εικόνα 81: Character.cs (Μέρος Α)

```
//this function is called when the object is enabled
//+= we add that function to the list of the listeners of that event
 Cristina Mpalla
void OnEnable() {
    TurnBaseController.Instance.onNewTurn += OnNewTurn;
}

//this function is called when the object is disabled or destroyed
//-= we remove that function from the list of the listeners of that event
 Cristina Mpalla
void OnDisable() {
    TurnBaseController.Instance.onNewTurn -= OnNewTurn;
}

//enable the visual if it is the turn of the calling character now, or not if, its not their turn
 Cristina Mpalla
void OnNewTurn() {
    characterUI.ToggleTurnVisual(TurnBaseController.Instance.GetCurrentTurnCharacter() == this);
}

 Cristina Mpalla *
private void Start() {

    //shake is a small animation move of the camera when a character is taking damage
    shake = GameObject.FindGameObjectWithTag(screenShakeTag).GetComponent<Shake>();

    //gameplay sounds are the ones that are played when a character is attacking,healing,dying
    gamePlaySounds = GameObject.FindGameObjectWithTag(battleSoundsTag).GetComponent<GamePlaySounds>();

    //where is the position of our characters, for the movement
    standingPosition = transform.position;

    characterUI.SetCharacterNameText(displayName);
    characterUI.UpdateHealthBar(curHp,maxHp);

}

//if the character is null set the target to the current character as the target can be null only when we cast ourselves
//then cast the battle action with the current character as the caster and the target as the target
 Cristina Mpalla *
public void CastBattleAction(BattleAction battleAction, Character target = null) {
    if (target == null)
        target = this;

    battleAction.Cast( caster: this,target);
    gainXP += 10; //the character gains 10xp for an attack
    if ( (bool) battleAction as EffectAction || (bool) battleAction as RangedAction) {
        animationClass.RangeAttackAnimation();
        gamePlaySounds.PlayRangeSound();
        gainXP += 5; //the character gains extra 5xp if the attack is ranged or effect
    }
    if (battleAction as HealAction)
        gainXP += 8; //the character gains 8xp if he's healing
}
```

Εικόνα 82: Character.cs (Μέρος Β)

```
//this function is called when we take damage to decrease our health bar stats accordingly
//shake the camera when someone is taking damage and trigger the animation
 Cristina Mpalla *
public void TakeDamage(int damage) {

    shake.CamShake();
    animationClass.GotHit();
    curHp -= damage;

    characterUI.UpdateHealthBar(curHp,maxHp);

    if (curHp <= 0)
        Die();
}

//this function is called when the character is healed, increase the hp with the given amount,
//play the appropriate sound and trigger the heal animation
 Cristina Mpalla
public void Heal(int amount) {

    gamePlaySounds.PlayHealSound();

    animationClass.HealingAnimation();
    curHp += amount;

    if (curHp > maxHp)
        curHp = maxHp;

    characterUI.UpdateHealthBar(curHp, maxHp);
    Instantiate(healParticlePrefab, transform);
}

//when a character dies, the ui is being deactivated, proper sound is played
//the die animation is triggered and then the gameobject is destroyed
 Cristina Mpalla *
void Die() {
    characterUI.DeactivateUI();
    animationClass.characterAnim.SetTrigger(dieAnimTrigger);
    gamePlaySounds.PlayDeadSound();
    onCharacterDeath.Invoke( arg0: this);
    Destroy(gameObject, t: 0.8f);
}
```

Εικόνα 83: Character.cs (Μέρος Γ)

```
    //this function moves the character towards the target, invokes the arrive callback and then moves the character
    //back to the standing position
     Cristina Mpalla *
    public void MoveToTarget(Character target, UnityAction<Character> arriveCallback) {

        StartCoroutine( routine: MeleeAttackAnimation());

        IEnumerator MeleeAttackAnimation() {

            //to make the character look at the opponent
            Quaternion current = transform.rotation;
            transform.LookAt(target.transform);

            while (Vector3.Distance( a: transform.position,  b: target.transform.position) > 3f) {

                animationClass.WalkingAnimation();
                transform.position = Vector3.MoveTowards( current: transform.position,  target: target.transform.position,  maxDistanceDelta: 10*Time.deltaTime);
                yield return null;
            }

            //reset trigger functions, reset the value of the trigger parameter
            animationClass.characterAnim.ResetTrigger(walkingAnimTrigger);

            animationClass.MeleeAttackAnimation();

            yield return new WaitForSeconds(animationClass.characterAnim.GetCurrentAnimatorStateInfo( layerIndex: 0).length);
            gamePlaySounds.PlayMeleeSound(); //after the animation ends, play the melee attack sound

            arriveCallback?.Invoke(target);

            yield return new WaitForSeconds(animationClass.characterAnim.GetCurrentAnimatorStateInfo( layerIndex: 0).normalizedTime + .2f);
            animationClass.characterAnim.ResetTrigger(meleeAnimTrigger);

            while (transform.position != standingPosition) {

                animationClass.WalkingBackAnimation();
                transform.position = Vector3.MoveTowards( current: transform.position,  target: standingPosition,  maxDistanceDelta: 8 * Time.deltaTime);
                yield return null;
            }

            animationClass.characterAnim.ResetTrigger(walkBackAnimTrigger);
            animationClass.characterAnim.ResetTrigger(gotHitAnimTrigger);
            animationClass.IdleAnimation();

            transform.eulerAngles = current.eulerAngles; //when the attack is finished, the character returns to default position
        }
    }

    //the visual at the base of each selected character
     Cristina Mpalla
    public void ToggleSelectionVisual(bool toggle) {
        selectionVisual.SetActive(toggle);
    }
}
```

Εικόνα 84: Character.cs (Μέρος Δ)

Παράρτημα 2: Game Controller Script

```
using System.Collections.Generic;
using UnityEngine;
using TMPro;
Cristina Mpalla
public class GameController : MonoBehaviour
{
    public static CharacterSet curEnemySet;

    public Character[] playerTeam;
    public Character[] enemyTeam;

    private List<Character> allCharacters = new List<Character>();

    [Header("Level Update")] public int levelToUnlock = 2;

    [Header("Components")]
    //references of the spawn positions for the characters
    public Transform[] playerTeamSpawns;
    public Transform[] enemyTeamSpawns;

    [Header("Data")]
    public PlayerPersistentData playerPersistentData;
    public CharacterSet defaultEnemySet;

    [Header("Victory/Defeat Panels")]
    public GameObject victoryPanel;
    public GameObject defeatPanel;

    [Header("Victory Stats")]
    public TMP_Text[] playerHealth;
    public TMP_Text[] playerXP;
    public TMP_Text[] totalPlayerXP;
    public TMP_Text[] levelUP;

    //integer variables in script
    private int levelUpVar = 100;
    private int levelUpCondition = 70;

    //the game controller is now a singleton class, that means that only one instance exists
    //and that instance is available from anywhere in the project
    public static GameController Instance;
```

Εικόνα 85: GameController.cs (Μέρος Α)

```
Cristina Mpalla
void OnEnable () {
    Character.onCharacterDeath += OnCharacterKilled;
}
Cristina Mpalla
void OnDisable () {
    Character.onCharacterDeath -= OnCharacterKilled;
}

Cristina Mpalla
private void Awake() {
    if (Instance != null && Instance != this)  //check if the instance is already set in awake
        Destroy(gameObject);
    else
        Instance = this;
}

Cristina Mpalla
void Start() {
    if(curEnemySet == null)
        CreateCharacters(playerPersistentData, defaultEnemySet);
    else
        CreateCharacters(playerPersistentData, curEnemySet);

    TurnBaseController.Instance.Begin(); //the turn base controller will be called to define which team strikes first
}
```

Εικόνα 86: GameController.cs (Μέρος Β)

```
//this function will spawn and organize all of the characters on the scene
Cristina Mpalla
void CreateCharacters(PlayerPersistentData playerData, CharacterSet enemyTeamSet) {

    //set the player team and enemy team to the new character array of the same length as the array in the brackets
    playerTeam = new Character[playerData.characters.Length];
    enemyTeam = new Character[enemyTeamSet.characters.Length];

    //we need this var because some characters may be dead so we must iterate over the spawn positions
    //so we wont have empty gaps on the players side on the scene
    int playerSpawnIndex = 0;

    for (int i = 0; i < playerData.characters.Length; i++) {
        Character character = CreateCharacter(playerData.characters[i].characterPrefab, playerTeamSpawns[playerSpawnIndex]);

        character.curHp = playerData.characters[i].health;
        character.maxHp = playerData.characters[i].health;

        playerTeam[i] = character;
        playerSpawnIndex++;
    }

    for (int i = 0; i < enemyTeamSet.characters.Length; i++) {
        Character character = CreateCharacter(enemyTeamSet.characters[i], enemyTeamSpawns[i]);
        enemyTeam[i] = character;
    }

    //both player team and enemy team arrays are added to allCharacters list
    allCharacters.AddRange(playerTeam);
    allCharacters.AddRange(enemyTeam);
}
```

Εικόνα 87: GameController.cs (Μέρος Γ)

```
//this function will be called for every character we want to create
Cristina Mpalla
Character CreateCharacter(GameObject characterPrefab, Transform spawnPos) {

    //instantiate the requested character prefab on the desired position and rotation
    GameObject obj = Instantiate(characterPrefab, spawnPos.position, spawnPos.rotation);

    //get the character component and return the reference to the spawned character
    return obj.GetComponent<Character>();
}

//this function removes the killed character from the all characters list,
//iterates through the all characters list and check the characters team
Cristina Mpalla
void OnCharacterKilled(Character character) {

    allCharacters.Remove(character);

    int playersRemaining = 0;
    int enemiesRemaining = 0;

    for (int i = 0; i < allCharacters.Count; i++)
        if (allCharacters[i].team == Character.Team.Player)
            playersRemaining++;
        else
            enemiesRemaining++;

    if (enemiesRemaining == 0)
        Invoke(nameof(PlayerTeamWins), time: 2.5f);
    else if (playersRemaining == 0)
        Invoke(nameof(EnemyTeamWins), time: 2.5f);
}
```

Εικόνα 88: GameController.cs (Μέρος Δ)

```
//this function updates the stats of the players xp,hp if they won
//and unlocks the next level available, also triggers the victory panel
//which is the ui for the result of the battle
Cristina Mpalla
void PlayerTeamWins() {

    UpdatePlayerPersistentData();
    PlayerPrefs.SetInt("levelReached", levelToUnlock);

    victoryPanel.gameObject.SetActive(true);
}

//this function reveals the defeat panel if the players lost the battle
Cristina Mpalla
void EnemyTeamWins() {

    defeatPanel.gameObject.SetActive(true);
    if (defeatPanel.activeSelf)
        defeatPanel.GetComponent<AudioSource>().Play();
}
```

Εικόνα 89: GameController.cs (Μέρος Ε)

```
//this function updates the persistent data of the character, total xp, gained xp, left xp and total health points
//basically here we make the level up of the character according to how many xp the player gained throughout the battle
//and in combination with the left xp from a previous battle, we set a limit and each time the limit is crossed then the level up occurs
 Cristina Mpalla
void UpdatePlayerPersistentData()
{
    for (int i = 0; i < playerTeam.Length; i++) {

        if (playerTeam[i] != null) {
            levelUP[i].text = " ";
            playerPersistentData.characters[i].leftXP += playerTeam[i].leftXP;

            playerXP[i].text = playerTeam[i].gainXP.ToString(); //each xp gained throughout the current battle

            playerTeam[i].leftXP = playerTeam[i].gainXP + playerPersistentData.characters[i].leftXP; //xp left  (didnt give level up) from previous battle

            while (playerTeam[i].leftXP > 0) {

                if (playerTeam[i].leftXP >= levelUpCondition){
                    //levelUpCondition is the limit 70 xp, whenever crossed, level up occurs
                    playerTeam[i].maxHp +=
                        levelUpVar; //levelUpVar, how many hp the character will gain for the level up (+100hp)
                    playerTeam[i].leftXP -=
                        levelUpCondition; //to check again if 2 level ups where gained, we subtract the levelUpCondition
                    levelUP[i].text = "LEVEL UP";
                }else
                    break;
            }

            //save the variables above to the player persistent data of the character and also display the variables on victory panel
            playerPersistentData.characters[i].health = playerTeam[i].maxHp;
            playerHealth[i].text = playerPersistentData.characters[i].health.ToString();

            playerPersistentData.characters[i].leftXP = playerTeam[i].leftXP;

            playerTeam[i].totalXP = playerTeam[i].gainXP + playerPersistentData.characters[i].totalXP;
            playerPersistentData.characters[i].totalXP = playerTeam[i].totalXP;

            totalPlayerXP[i].text = playerPersistentData.characters[i].totalXP.ToString();
        }
        else {
            playerHealth[i].text = playerPersistentData.characters[i].health.ToString();
            totalPlayerXP[i].text = playerPersistentData.characters[i].totalXP.ToString();

        }
    }
}
```

Εικόνα 90: GameController.cs (Μέρος ΣΤ)

Παράρτημα 3: Turn Base Controller Script

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

Cristina Mpalla *
public class TurnBaseController : MonoBehaviour
{
    private List<Character> turnOrder = new List<Character>();
    private int curTurnOrderIndex;
    private Character curTurnCharacter;

    //the event keyword makes the onNew Turn event invokable only from this class which is defined in
    //this is basically a safety measure to make sure that no other script calls the onNewTurn event
    Cristina Mpalla
    public event UnityAction onNewTurn;

    //Singleton class
    public static TurnBaseController Instance;
    Cristina Mpalla
    void Awake() {
        if (Instance != null && Instance != this)
            Destroy(gameObject);
        else
            Instance = this;
    }

    Cristina Mpalla
    public void Begin() {
        GenerateTurnOrder(Character.Team.Player); //this means that the player team will start first by default
        NewTurn(turnOrder[0]);
    }

    //this function will generate which team strikes first
    Cristina Mpalla
    void GenerateTurnOrder(Character.Team startingTeam) {
        if (startingTeam == Character.Team.Player) {
            turnOrder.AddRange(GameController.Instance.playerTeam);
            turnOrder.AddRange(GameController.Instance.enemyTeam);
        }
        else if (startingTeam == Character.Team.Enemy) {
            turnOrder.AddRange(GameController.Instance.enemyTeam);
            turnOrder.AddRange(GameController.Instance.playerTeam);
        }
    }
```

Εικόνα 91: TurnBaseController.cs (Μέρος Α)

```
Cristina Mpalla *
void NewTurn(Character character) {
    curTurnCharacter = character;

    if (curTurnCharacter.gameObject == null) {
        EndTurn();
        return;
    }
    onNewTurn?.Invoke(); //this is a null check, basically a safety precaution because if the event is called falsely
                         //we will get an exception and that will stop the NewTurn function
}

Cristina Mpalla
public void EndTurn(){
    curTurnOrderIndex++; //change to the next character
    if (curTurnOrderIndex == turnOrder.Count)
        curTurnOrderIndex = 0;

    //as some of the entries in the turn order might be null, the curTurnOrderIndex must be increased until we find a
    //not null character
    while (turnOrder[curTurnOrderIndex] == null) {
        curTurnOrderIndex++;
        if (curTurnOrderIndex == turnOrder.Count)
            curTurnOrderIndex = 0;
    }

    NewTurn(turnOrder[curTurnOrderIndex]);
}

//this function is called to know whose turn it is now
Cristina Mpalla
public Character GetCurrentTurnCharacter() {
    return curTurnCharacter;
}
```

Εικόνα 92: TurnBaseController.cs (Μέρος Β)

Παράρτημα 4: Player Battle Controller Script

```
using UnityEngine;
using UnityEngine.InputSystem;
using Random = UnityEngine.Random;

Cristina Mpalla *
public class PlayerBattleController : MonoBehaviour
{
    public float selectionCheckRate = 0.02f;  //to setup how often the selection is being checked
    private float lastSelectionCheckTime; //to cache the last time we have checked the selection
    public LayerMask selectionLayerMask; //to set the physics layers we are going to check for selection

    private BattleAction curSelectionBattleAction; //to cache which action we have selected
    private Character curSelectedCharacter; //to keep track of the character we tapped

    private bool isActive;
    private bool canSelectSelf;
    private bool canSelectTeam;
    private bool canSelectEnemy;

    [Header("Components")]
    public BattleActionsUI battleActionsUI;

    //classic singleton
    public static PlayerBattleController Instance;
    Cristina Mpalla
    void Awake() {
        if (Instance != null && Instance != this)
            Destroy(gameObject);
        else
            Instance = this;
    }

    Cristina Mpalla
    void OnEnable() {
        TurnBaseController.Instance.onNewTurn += OnNewTurn;
    }

    Cristina Mpalla
    void OnDisable() {
        TurnBaseController.Instance.onNewTurn -= OnNewTurn;
    }
```

Εικόνα 93: PlayerBattleController.cs (Μέρος Α)

```
//here we check whose turn it is
 Cristina Mpalla
void OnNewTurn() {

    if (TurnBaseController.Instance.GetCurrentTurnCharacter().gameObject == null) {
        EndTurn();
        return;
    }

    if (TurnBaseController.Instance.GetCurrentTurnCharacter().team == Character.Team.Player) {
        EnablePlayerBattle();
        battleActionsUI.DisplayBattleActions(TurnBaseController.Instance.GetCurrentTurnCharacter());
    }
    else
        DisablePlayerBattle();
}

//to change the status of the player battle controller
 Cristina Mpalla
void EnablePlayerBattle() {
    curSelectedCharacter = null;
    curSelectionBattleAction = null;
    isActive = true;
}

//to change the status of the player battle controller
 Cristina Mpalla
void DisablePlayerBattle() {
    isActive = false;
}
 Cristina Mpalla
private void Update() {
    //only run update if battle is enabled
    if (!isActive || curSelectionBattleAction == null)
        return;

    //if enough time has passed from the last selection check time then update this var and call the selection check
    if (Time.time - lastSelectionCheckTime > selectionCheckRate) {
        lastSelectionCheckTime = Time.time;
        SelectionCheck();
    }

    if ((Input.touchPressureSupported || Mouse.current.leftButton.isPressed) && curSelectedCharacter != null)
        CastBattleAction();
}
```

Εικόνα 94: PlayerBattleController.cs (Μέρος Β)

```
//this function is called to quickly check which character we tapped
Cristina Mpalla *
void SelectionCheck() {

    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;

    //if the raycast has hit something we will check what it hit, otherwise we will un select the currently selected character
    if (Physics.Raycast(ray, out hit, maxDistance: 999, (int) selectionLayerMask)) {

        Character character = hit.collider.GetComponent<Character>();

        //if the hit character is the same as the previously one then do nothing
        if (curSelectedCharacter != null && curSelectedCharacter == character)
            return;

        //if the current action allows us to select ourselves and the hit character is us...
        //else if the current action is allowed to be used on player team and the hit char is from player team...
        //else if the current action is allowed to be used on the enemy team and the hit char is from enemy team...
        if (canSelectSelf && character == TurnBaseController.Instance.GetCurrentTurnCharacter()) {
            SelectCharacter(character);
            return;
        }else if (canSelectTeam && character.team == Character.Team.Player) {
            SelectCharacter(character);
            return;
        }else if (canSelectEnemy && character.team == Character.Team.Enemy) {
            SelectCharacter(character);
            return;
        }
    }
    Invoke(nameof(UnSelectCharacter), time: 1); //we call the unselect character function with 1 sec delay to have the visibility
                                                //of green turn visual
}

//this function is called once we click on the selected character so it gets the current character
//from the turn base controller and make it cast an action
Cristina Mpalla *
void CastBattleAction() {

    curSelectedCharacter.ToggleSelectionVisual(true);
    TurnBaseController.Instance.GetCurrentTurnCharacter().CastBattleAction(curSelectionBattleAction, curSelectedCharacter);
    curSelectionBattleAction = null;

    UnSelectCharacter();
    DisablePlayerBattle();
    battleActionsUI.DisableBattleActions();

    Invoke(nameof(EndTurn), time: Random.Range(1.5f,2.2f)); //we call the next turn delay function with delay
}
```

Εικόνα 95: PlayerBattleController.cs (Μέρος Γ)

```
//this function makes the next character take turn
 Cristina Mpalla
void EndTurn() {
    TurnBaseController.Instance.EndTurn();
}

//unselect the current character in case someone was selected before
//and cache the character to the cur selected character var
 Cristina Mpalla
void SelectCharacter(Character character) {

    UnSelectCharacter();
    curSelectedCharacter = character;
    curSelectedCharacter.ToggleSelectionVisual(true);
}

//return from the function if no character is selected, disable the selection visual and se the current char to null
 Cristina Mpalla
void UnSelectCharacter() {
    if(curSelectedCharacter == null)
         return;

    curSelectedCharacter.ToggleSelectionVisual(false);
    curSelectedCharacter = null;
}

//this function caches the passed battle action and sets the flags accordingly
 Cristina Mpalla *
public void SetCurrentBattleAction(BattleAction battleAction) {

    curSelectionBattleAction = battleAction;

    canSelectSelf = false;
    canSelectTeam = false;
    canSelectEnemy = false;

    if ((bool)battleAction as MeleeAction || (bool)battleAction as RangedAction)
        canSelectEnemy = true;
    else if (battleAction as HealAction) {
        canSelectSelf = true;
        canSelectTeam = true;
    }
    else if (battleAction as EffectAction) {
        canSelectSelf = ((EffectAction)battleAction).canEffectSelf;
        canSelectTeam = ((EffectAction)battleAction).canEffectTeam;
        canSelectEnemy = ((EffectAction)battleAction).canEffectEnemy;
    }
}
```

Εικόνα 96: PlayerBattleController.cs (Μέρος Δ)

Παράρτημα 5: Enemy Battle Controller Script

```
using UnityEngine;
using System.Linq;
using Random = UnityEngine.Random;
Cristina Mpalla *
public class EnemyBattleController : MonoBehaviour
{
    [Header("AI")]
    public float minWaitTime = 1f;
    public float maxWaitTime = 2f;

    [Header("Attacking")]
    //to set up the chance of attacking the weakest opponent
    public float attackWeakestChance = 0.85f;

    [Header("Chance Curves")]
    //to set up the probability of healing
    public AnimationCurve healChanceCurve;

    private Character curEnemy; //to keep track of the current enemy
    Cristina Mpalla
    void OnEnable() {
        TurnBaseController.Instance.onNewTurn += OnNewTurn;
    }
    Cristina Mpalla
    void OnDisable() {
        TurnBaseController.Instance.onNewTurn -= OnNewTurn;
    }

    //here we check if the current turn is the enemy's turn. cache the current character to the appropriate var
    //and invoke the main function of the script
    Cristina Mpalla
    void OnNewTurn() {
        if (TurnBaseController.Instance.GetCurrentTurnCharacter().team == Character.Team.Enemy) {
            curEnemy = TurnBaseController.Instance.GetCurrentTurnCharacter();
            Invoke(nameof(DecideBattleAction), time: Random.Range(minWaitTime,maxWaitTime));
        }
    }
```

Εικόνα 97: EnemyBattleController.cs (Μέρος Α)

```
//the main function of the scripts which picks the battle action for the current enemy
Cristina Mpalla
void DecideBattleAction() {

    //if the current enemy has a healing battle action
    //check if the weakest enemy partner requires healing and the cast the heal
    if (HasBattleActionOfType(typeof(HealAction))) {
        Character weakestEnemy = GetWeakestCharacter(Character.Team.Enemy);
        if (Random.value < healChanceCurve.Evaluate( time: GetHealthPercentage(weakestEnemy))) {
            CastBattleAction(GetHealBattleAction(), weakestEnemy);
            return;
        }
    }

    //find a target character, depending on how the chance will roll attack the weakest one or a random one
    Character playerToDamage;

    if (Random.value < attackWeakestChance)
        playerToDamage = GetWeakestCharacter(Character.Team.Player);
    else
        playerToDamage = GetRandomCharacter(Character.Team.Player);

    //if the target character is not null then if we have the appropriate type of damage action, cast it to the target
    if (playerToDamage != null)
        if(HasBattleActionOfType(typeof(MeleeAction)) || HasBattleActionOfType(typeof(RangedAction))) {
            CastBattleAction(GetDamageBattleAction(), playerToDamage);
            return;
        }
    //end the turn of the current character after a certain amount of time which bounds we set in the inspector
    Invoke(nameof(EndTurn), time: Random.Range(minWaitTime,maxWaitTime));
}

//this function casts the battle action towards the target character
Cristina Mpalla
void CastBattleAction(BattleAction battleAction, Character target) {
    if (curEnemy == null) {
        EndTurn();
        return;
    }

    curEnemy.CastBattleAction(battleAction, target);
    Invoke(nameof(EndTurn), time: Random.Range(minWaitTime,maxWaitTime));
}

//to call once the enemy has finished their turn
Cristina Mpalla
void EndTurn() {
    TurnBaseController.Instance.EndTurn();
}
```

Εικόνα 98: EnemyBattleController.cs (Μέρος Β)

```
//this should return the characters current hp divided by the characters max hp
Cristina Mpalla
float GetHealthPercentage (Character character) {
    return (float)character.curHp / (float)character.maxHp;
}

//this checks if our current character has the battle action of the specified type
//so we iterate over the battle actions of the current enemy
Cristina Mpalla
bool HasBattleActionOfType (System.Type type) {
    foreach (BattleAction ba in curEnemy.battleActions)
        if (ba.GetType() == type)
            return true;
    return false;
}

//this function will return one of the damage battle actions the current character has.
//so we create an array of the damage actions and select randomly one of them
Cristina Mpalla *
BattleAction GetDamageBattleAction (){
    BattleAction[] ba = curEnemy.battleActions.Where(x :BattleAction => x.GetType() == typeof(MeleeAction) || x.GetType() == typeof(RangedAction)).ToArray();

    if (ba == null || ba.Length == 0)
        return null;
    return ba[Random.Range(0, ba.Length)];
}

//this function will return one of the heal battle actions the current character has.
//so we create an array of the healing actions and select randomly one of them
Cristina Mpalla
BattleAction GetHealBattleAction () {
    BattleAction[] ba = curEnemy.battleActions.Where(x :BattleAction => x.GetType() == typeof(HealAction)).ToArray();

    if (ba == null || ba.Length == 0)
        return null;
    return ba[Random.Range(0, ba.Length)];
}

//this function will return one of the effect battle actions the current character has.
//so we create an array of the effect actions and select randomly one of them
Cristina Mpalla
BattleAction GetEffectBattleAction () {
    BattleAction[] ba = curEnemy.battleActions.Where(x :BattleAction => x.GetType() == typeof(EffectAction)).ToArray();

    if (ba == null || ba.Length == 0)
        return null;
    return ba[Random.Range(0, ba.Length)];
}
```

Εικόνα 99: EnemyBattleController.cs (Μέρος Γ)

```
//this function finds the weakest opponent
Cristina Mpalla
Character GetWeakestCharacter (Character.Team team) {
    int weakestHp = 999; //by default to be init with a number bigger than any possible health amount, so we could be able to pick at least one character in any way
    int weakestIndex = 0; //to keep track of the index of the weakest character in the array

    //we're checking if the requested team is the player team to get the playerTeam array,otherwise, we get the enemyTeam array
    Character[] characters = team == Character.Team.Player
        ? GameController.Instance.playerTeam
        : GameController.Instance.enemyTeam;

    for (int i = 0; i < characters.Length; i++) {
        if (characters[i] == null)
            continue;

        if (characters[i].curHp < weakestHp) {
            weakestHp = characters[i].curHp;
            weakestIndex = i;
        }
    }

    return characters[weakestIndex];
}

//this function return some random opponent, depending on the requested team
Cristina Mpalla
Character GetRandomCharacter (Character.Team team) {
    Character[] characters = null;

    if (team == Character.Team.Player)
        characters = GameController.Instance.playerTeam.Where(x Character => x != null).ToArray();
    else if(team == Character.Team.Enemy)
        characters = GameController.Instance.enemyTeam.Where(x Character => x != null).ToArray();

    return characters[Random.Range(0, characters.Length)];
}
```

Εικόνα 100: EnemyBattleController.cs (Μέρος Δ)

Παράρτημα 6: Audio Controller Script

```
using UnityEngine;
using UnityEngine.UI;
Cristina Mpalla
public class AudioController : MonoBehaviour
{
    private static readonly string FirstPlay = "FirstPlay"; //player prefs
    private static readonly string MusicPref = "MusicPref";
    private static readonly string SoundsPref = "SoundsPref";

    private int firstPlayInt;

    public Slider musicSlider;
    public Slider systemSoundsSlider;

    private float musicFloat;
    private float soundsFloat;

    public AudioSource musicAmbience;
    public AudioSource[] systemSoundsAudio;
```

Εικόνα 101: AudioController.cs (Μέρος Α)

```
    public void Start() {
        firstPlayInt = PlayerPrefs.GetInt( key: FirstPlay);

        if (firstPlayInt == 0) {
            musicFloat = .25f;
            soundsFloat = .75f;

            musicSlider.value = musicFloat;
            systemSoundsSlider.value = soundsFloat;

            PlayerPrefs.SetFloat(MusicPref, musicFloat);
            PlayerPrefs.SetFloat(SoundsPref, soundsFloat);

            PlayerPrefs.SetInt(FirstPlay, -1);
        }
        else {
            musicFloat = PlayerPrefs.GetFloat( key: MusicPref);
            musicSlider.value = musicFloat;

            soundsFloat = PlayerPrefs.GetFloat( key: SoundsPref);
            systemSoundsSlider.value = soundsFloat;
        }
    }

    Cristina Mpalla
    public void SaveSoundSettings() {
        PlayerPrefs.SetFloat(MusicPref, musicSlider.value);
        PlayerPrefs.SetFloat(SoundsPref, systemSoundsSlider.value);
    }

    Cristina Mpalla
    public void OnApplicationFocus(bool inFocus) {
        if (!inFocus)
            SaveSoundSettings();
    }

    Cristina Mpalla
    public void UpdateSound() {
        musicAmbience.volume = musicSlider.value;

        for (int i = 0; i < systemSoundsAudio.Length; i++)
            systemSoundsAudio[i].volume = systemSoundsSlider.value;
    }
}
```

Εικόνα 102: AudioController.cs (Μέρος Β)