ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τίτλος Πτυχιακής Εργασίας	Ελληνικά : Ένα 2D Unity Platformer παιχνίδι με θέμα τις περιπέτειες του Οδυσσέα Αγγλικά : A 2D Unity Platformer Game concerning Odysseus Adventures
Ονοματεπώνυμο Φοιτητή	Εμμανουήλ Ρούτσης
Πατρώνυμο	Βασίλειος
Αριθμός Μητρώου	Π19237
Επιβλέπων Καθηγητής	Θεμιστοκλής Παναγιωτόπουλος

Ημερομηνία Παράδοσης **Σεπτέμβριος 2024**

Copyright ©

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν αποκλειστικά τον συγγραφέα και δεν αντιπροσωπεύουν τις επίσημες θέσεις του Πανεπιστημίου Πειραιώς.

Ως συγγραφέας της παρούσας εργασίας δηλώνω πως η παρούσα εργασία δεν αποτελεί προϊόν λογοκλοπής και δεν περιέχει υλικό από μη αναφερόμενες πηγές.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, Θεμιστοκλή Παναγιωτόπουλο, για την καθοδήγησή του, τις πολύτιμες συμβουλές του και την υποστήριξή του σε όλη τη διάρκεια της πτυχιακής μου εργασίας.

Επίσης, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στην οικογένειά μου, που στάθηκε δίπλα μου σε κάθε βήμα αυτής της πορείας, προσφέροντάς μου την ψυχική και ηθική στήριξη που χρειαζόμουν για να φτάσω σε αυτό το σημείο.

Ιδιαίτερες ευχαριστίες οφείλω στους φίλους μου, που με υποστήριξαν αδιάκοπα και με ενθάρρυναν καθ' όλη τη διάρκεια αυτής της δύσκολης και απαιτητικής διαδικασίας. Χωρίς την αφοσίωση και τη συνεχή υποστήριξή τους, δεν θα μπορούσα να φτάσω στην ολοκλήρωση αυτής της εργασίας.

Περίληψη

Η παρούσα πτυχιακή εργασία επικεντρώνεται στην ανάπτυξη ενός 2D βιντεοπαιχνιδιού με τίτλο "Οι περιπέτειες του Οδυσσέα", το οποίο εμπνέεται από την ελληνική μυθολογία και συγκεκριμένα τις περιπέτειες του Οδυσσέα. Σκοπός του παιχνιδιού είναι να εκπαιδεύσει τους παίκτες στην ιστορία του Οδυσσέα με έναν σύγχρονο και διασκεδαστικό τρόπο, μέσα από την αλληλεπίδραση με χαρακτήρες και την εξερεύνηση του παιχνιδιού.

Η υλοποίηση του παιχνιδιού έγινε χρησιμοποιώντας τη μηχανή ανάπτυξης Unity και προγραμματιστική γλώσσα C#. Το παιχνίδι διαθέτει διάφορους μηχανισμούς όπως μάχες με εχθρούς, γρίφους, και εξερεύνηση διαφόρων επιπέδων, προσφέροντας μια ποικιλία προκλήσεων στον παίκτη.

Η πρωτοτυπία του παιχνιδιού έγκειται στο γεγονός ότι αντλεί έμπνευση από την ελληνική μυθολογία, δίνοντας την ευκαιρία στους παίκτες να γνωρίσουν τα κατορθώματα του Οδυσσέα μέσα από δια δραστικές εμπειρίες.

Το project βασίστηκε σε γραφικά και ήχους που αντλήθηκαν από διάφορες ανοιχτές πλατφόρμες όπως το Unity Asset Store, Itch.io FreeMusic. Μέσω αυτής της εργασίας, ο συγγραφέας απέκτησε σημαντικές δεξιότητες στην ανάπτυξη βιντεοπαιχνιδιών και κατανόησε σε βάθος τις σύγχρονες τεχνικές ανάπτυξης παιχνιδιών.

Λέξεις Κλειδιά:

Οδυσσέας, ελληνική μυθολογία, 2D Action Platformer, Unity, ανάπτυξη παιχνιδιών

ABSTRACT

This abstract focuses on the development of a 2D video game titled "The Adventures of Odysseus," inspired by Greek mythology, specifically the adventures of Odysseus. The goal of the game is to educate players about Odysseus' story in a modern and entertaining way, through character interaction and game exploration.

The game was developed using the Unity game engine and the C# programming language. It features various mechanics, such as enemy battles, puzzles, and exploration across multiple levels, offering a variety of challenges to the player.

The originality of the game lies in its inspiration from Greek mythology, giving players the opportunity to experience Odysseus' achievements through interactive gameplay. The project utilized graphics and sounds sourced from various open platforms such as the Unity Asset Store, Itch.io, and FreeMusic. Through this work, the author gained significant skills in video game development and gained a deep understanding of modern game development techniques.

Keywords:

Odysseus, Greek mythology, 2D Action Platformer, Unity, game development.

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

Τα βιντεοπαιχνίδια πλατφόρμας 2D έχουν γίνει μια πολύ δημοφιλής κατηγορία τα τελευταία χρόνια, με τίτλους όπως το Hollow Knight και το Blasphemous να προσφέρουν καθηλωτικές εμπειρίες σε παίκτες παγκοσμίως. Σε αυτό το πλαίσιο, το παρόν project, "Οι περιπέτειες του Οδυσσέα", έρχεται να αναδείξει έναν σημαντικό ήρωα της ελληνικής μυθολογίας, τον Οδυσσέα, μέσα από ένα εκπαιδευτικό και διασκεδαστικό βιντεοπαιχνίδι.

Το παιχνίδι τοποθετεί τον παίκτη στον ρόλο του Οδυσσέα, καθώς προσπαθεί να επιστρέψει στην πατρίδα του, την Ιθάκη, αντιμετωπίζοντας προκλήσεις και εχθρούς που βασίζονται σε κλασικούς μύθους. Η επιλογή αυτού του θέματος βασίζεται στην επιθυμία να αναδειχθούν οι ιστορίες της ελληνικής μυθολογίας με έναν διαδραστικό και σύγχρονο τρόπο, καθιστώντας την εκπαίδευση πάνω σε αυτές τις ιστορίες πιο ελκυστική για το ευρύ κοινό.

Η ανάπτυξη του παιχνιδιού πραγματοποιήθηκε με τη χρήση της Unity, μιας από τις πιο δημοφιλείς πλατφόρμες ανάπτυξης παιχνιδιών, και της C#, που χρησιμοποιήθηκε για τη δημιουργία των scripts που διαχειρίζονται τις διάφορες μηχανικές του παιχνιδιού.

Η εργασία αυτή έχει ως κύριο στόχο την παρουσίαση των μηχανισμών, των εργαλείων και των μεθόδων που χρησιμοποιήθηκαν για την ανάπτυξη του παιχνιδιού, καθώς και την αξιολόγηση της επίδρασης που έχει στην κατανόηση της ελληνικής μυθολογίας από τους παίκτες.

Το πρόβλημα και πιθανές προσεγγίσεις

Ένα από τα βασικά προβλήματα που αντιμετώπισα κατά τη διάρκεια της ανάπτυξης του παιχνιδιού μου ήταν η έλλειψη προηγούμενης εμπειρίας με τη μηχανή Unity. Ενώ είχα επιλέξει αυτό το είδος παιχνιδιού για την πτυχιακή μου εργασία λόγω του ενδιαφέροντός μου, η μετάβαση από το μηδέν σε ένα πλήρως λειτουργικό παιχνίδι αποδείχθηκε ιδιαίτερα απαιτητική.

Ένα από τα μεγαλύτερα εμπόδια που αντιμετώπισα ήταν η εκμάθηση του τρόπου λειτουργίας της Unity και συγκεκριμένα η διαχείριση των animation των χαρακτήρων μου. Η δημιουργία και διαχείριση των animations, ειδικά μέσω του Animator και του συστήματος parameters και transitions, αποτέλεσε μια πρόκληση. Η δυσκολία βρισκόταν στη διασύνδεση των διαφόρων animations, ώστε να λειτουργούν αρμονικά μεταξύ τους χωρίς προβλήματα.

Η λύση στο πρόβλημα αυτό ήρθε μέσα από πολλές ώρες προσπάθειας και πρακτικής. Αφιέρωσα αρκετό χρόνο στην εξάσκηση και στην παρακολούθηση διαδικτυακών μαθημάτων και βίντεο στο YouTube, όπου άλλοι δημιουργοί παρουσίαζαν τις δικές τους προσεγγίσεις και μεθόδους ανάπτυξης παιχνιδιών στην Unity. Μέσα από αυτή τη διαδικασία, κατάφερα να κατανοήσω καλύτερα τη λειτουργία της πλατφόρμας και να ξεπεράσω τα αρχικά εμπόδια που αντιμετώπιζα.

Ένα άλλο σημαντικό πρόβλημα που με απασχόλησε ήταν το πώς θα μπορούσα να "σχεδιάσω" το παιχνίδι, δεδομένης της τεράστιας κλίμακας και της πολυπλοκότητας της ιστορίας του Οδυσσέα. Αυτό μου δημιούργησε μεγάλο άγχος, καθώς η ιστορία περιλαμβάνει πολλές πτυχές και λεπτομέρειες, και ήταν δύσκολο να φανταστώ πώς θα μπορούσα να αποτυπώσω την πλήρη ιστορία μέσα σε ένα παιχνίδι. Ως αποτέλεσμα, αποφάσισα να εστιάσω σε ένα αρκετά μεγάλο μέρος της ιστορίας, χωρίς να προσπαθήσω να καλύψω το σύνολό της, διατηρώντας έτσι μια ισορροπία ανάμεσα στην πολυπλοκότητα και τη διαχείριση του project.

Προδιαγραφές – Ανάλυση Απαιτήσεων

και Ανάλυση της Εφαρμογής

Το παιχνίδι εκτελέστηκε σε έναν υπολογιστή με τα εξής χαρακτηριστικά:

Λειτουργικό Σύστημα: Microsoft Windows 11 Pro - 64 bit

Επεξεργαστής: AMD Ryzen 5 5600X 6-Core Processor, 3.7 GHz (6 Cores, 12 Logical Processors)

Μνήμη RAM: 16.0 GB

Κάρτα Μητρικής: Gigabyte B550 AORUS ELITE V2

Συνολική Χωρητικότητα Δίσκου: 1TB (SSD)

Έκδοση Unity: 2022.3.22f1

Πρέπει να σημειωθεί ότι το παιχνίδι δεν είναι απαιτητικό ως προς το hardware. Επομένως, ακόμη και ένας υπολογιστής με παλαιότερα ή χαμηλότερα χαρακτηριστικά μπορεί να εκτελέσει την εφαρμογή χωρίς την εμφάνιση τεχνικών προβλημάτων.

Περιγραφή της Εφαρμογής

Η εφαρμογή περιλαμβάνει δύο επίπεδα (game levels), τα οποία ο παίκτης πρέπει να ολοκληρώσει για να τερματίσει το παιχνίδι. Κατά την πορεία του, ο παίκτης συναντά διάφορους εχθρούς και τέρατα που πρέπει να αντιμετωπίσει. Η κεντρική αποστολή του παίκτη στο πρώτο επίπεδο είναι να βρει τον δρόμο για τον Κάτω Κόσμο, όπου βρίσκεται ένα κλειδί που απαιτείται για να ξεκλειδώσει την πόρτα που οδηγεί στο δεύτερο επίπεδο.

Πρώτο Επίπεδο

Στο πρώτο επίπεδο, ο παίκτης καλείται να εξερευνήσει διάφορες περιοχές και να πολεμήσει εχθρούς για να βρει το κλειδί στον Κάτω Κόσμο. Αυτό το κλειδί είναι απαραίτητο για να ξεκλειδώσει την πόρτα που θα του επιτρέψει να προχωρήσει στο δεύτερο επίπεδο. Τα τέρατα που συναντά ο παίκτης αποτελούν σοβαρές προκλήσεις, καθώς απαιτούν στρατηγική και γρήγορη αντίδραση για να νικηθούν.

Δεύτερο Επίπεδο

Φτάνοντας στο δεύτερο επίπεδο, ο παίκτης εισέρχεται στη σπηλιά του μυθικού τέρατος, Κύκλωπα. Η αναμέτρηση με τον Κύκλωπα είναι το αποκορύφωμα του παιχνιδιού, καθώς πρόκειται για το κεντρικό Boss Fight. Ο Κύκλωπας διαθέτει τρομερή δύναμη και η αντιμετώπισή του απαιτεί από τον παίκτη άριστο έλεγχο του χαρακτήρα και των δεξιοτήτων του. Ο Οδυσσέας πρέπει να χρησιμοποιήσει την εξυπνάδα του και να εκμεταλλευτεί τις αδυναμίες του Κύκλωπα για να τον νικήσει.

Πριν φτάσει στην τελική μάχη με τον Κύκλωπα, ο παίκτης θα χρειαστεί να ξαναδιασχίσει επικίνδυνα μονοπάτια, όπου θα αντιμετωπίσει μοχθηρά τέρατα και εχθρούς, παρόμοιους με εκείνους του πρώτου επιπέδου. Ωστόσο, υπάρχει ένας επιπλέον εχθρός, ο οποίος θα δυσκολέψει σημαντικά τον παίκτη αν αποφασίσει να τον αντιμετωπίσει. Σε αυτήν την περίπτωση, μπορεί να είναι σοφότερο να προσπαθήσει να τον αποφύγει.

Σύστημα Θανάτου και Επαναφοράς

Σε περίπτωση που ο παίκτης ηττηθεί, εμφανίζεται ένα Death Panel με επιλογές για να:

- Ξεκινήσει ένα νέο παιχνίδι από το επίπεδο στο οποίο πέθανε (π.χ., αν πέθανε στο πρώτο επίπεδο, θα ξαναρχίσει από την αρχή του πρώτου επιπέδου· αν πέθανε στο δεύτερο επίπεδο, θα ξεκινήσει από την αρχή του δεύτερου επιπέδου).
- Επιστρέψει στο αρχικό μενού.
- Βγει από το παιχνίδι.

Η Ιστορία του Οδυσσέα και η Προσαρμογή της στο Παιχνίδι

Η ιστορία του Οδυσσέα, όπως περιγράφεται στην Οδύσσεια του Ομήρου, είναι μια από τις πιο γνωστές περιπέτειες της ελληνικής μυθολογίας. Ο Οδυσσέας, βασιλιάς της Ιθάκης, ξεκινά το ταξίδι του για να επιστρέψει στην πατρίδα του μετά τον δεκαετή Τρωικό Πόλεμο. Ωστόσο, η επιστροφή του δεν ήταν απλή· συνάντησε πληθώρα εμποδίων και προκλήσεων που τον καθυστέρησαν για άλλα δέκα χρόνια. Από τις περιπέτειες με τους Λωτοφάγους και τον Κύκλωπα Πολύφημο μέχρι τη μάχη με τη Σκύλλα και την παραμονή του στο νησί της Καλυψούς, ο Οδυσσέας έπρεπε να επιδείξει αντοχή, πονηριά και θάρρος για να φτάσει πίσω στην Ιθάκη και να αποκαταστήσει την τάξη στο βασίλειό του.

Η ιστορία του Οδυσσέα περιλαμβάνει έναν τεράστιο αριθμό περιπετειών, που όμως είναι δύσκολο να ενσωματωθούν πλήρως σε ένα βιντεοπαιχνίδι. Για τον λόγο αυτό, στην ανάπτυξη του παιχνιδιού "Οι περιπέτειες του Οδυσσέα", εστιάσαμε στις πιο χαρακτηριστικές και σημαντικές στιγμές του ταξιδιού του. Οι περιπέτειες που επιλέχθηκαν δεν ακολουθούν αυστηρά την ιστορική ακρίβεια της Οδύσσειας, αλλά προσαρμόστηκαν ώστε να ταιριάζουν στην ανάγκη για δυναμικό gameplay και να προσφέρουν στον παίκτη μια ικανοποιητική εμπειρία.

Η Προσαρμογή της Ιστορίας στο Παιχνίδι

Το Νησί της Κίρκης και ο Κάτω Κόσμος

Μία από τις πιο σημαντικές περιπέτειες του Οδυσσέα είναι η επίσκεψή του στο νησί της Κίρκης. Στο παιχνίδι, όπως και στην αρχική ιστορία, οι σύντροφοι του Οδυσσέα μετατρέπονται σε γουρούνια από τη μάγισσα Κίρκη. Για να μεταφέρουμε αυτό το γεγονός στο gameplay, έχουμε ενσωματώσει healing items που απεικονίζουν τα πρόσωπα γουρουνιών. Αυτά τα αντικείμενα λειτουργούν ως σύμβολα των συντρόφων του Οδυσσέα, οι οποίοι πλέον δεν μπορούν να τον βοηθήσουν και έχουν μετατραπεί σε μια πηγή που του παρέχει μόνο αναγκαία επιβίωση.

Στο ίδιο μέρος του παιχνιδιού, ο Οδυσσέας κατεβαίνει στον Κάτω Κόσμο, αναζητώντας το κλειδί που χρειάζεται για να προχωρήσει στο επόμενο στάδιο. Αυτή η κατάβαση υποδηλώνει την ανάγκη του Οδυσσέα να αντιμετωπίσει τους νεκρούς και τις παλιές του αμαρτίες προτού μπορέσει να συνεχίσει το ταξίδι του. Ο παίκτης πρέπει να βρει το κλειδί στον Κάτω Κόσμο για να επιστρέψει και να ανοίξει την πύλη προς την επόμενη περιπέτεια.

Η Σπηλιά του Κύκλωπα

Στη συνέχεια, ο Οδυσσέας κατευθύνεται στη σπηλιά του Κύκλωπα Πολύφημου. Παρόλο που στην πραγματική ιστορία αυτή η περιπέτεια δεν αποτελεί το τελικό εμπόδιο του Οδυσσέα, στο παιχνίδι μας ο Κύκλωπας έχει υλοποιηθεί ως το τελικό εμπόδιο πριν από την ολοκλήρωση της αποστολής. Ο Πολύφημος είναι ένας τρομερός εχθρός, και η νίκη επί αυτού αποτελεί την κορύφωση του παιχνιδιού.

Ο Κύκλωπας είναι η τελευταία μεγάλη πρόκληση του Οδυσσέα στο παιχνίδι και απαιτεί από τον παίκτη να χρησιμοποιήσει όλες τις δεξιότητες που έχει μάθει μέχρι τώρα για να τον νικήσει. Μόλις ο παίκτης καταφέρει να εξουδετερώσει τον Κύκλωπα, το παιχνίδι μεταβαίνει στη τελική σκηνή, η οποία δείχνει τον Οδυσσέα να επιστρέφει στην πατρίδα του.

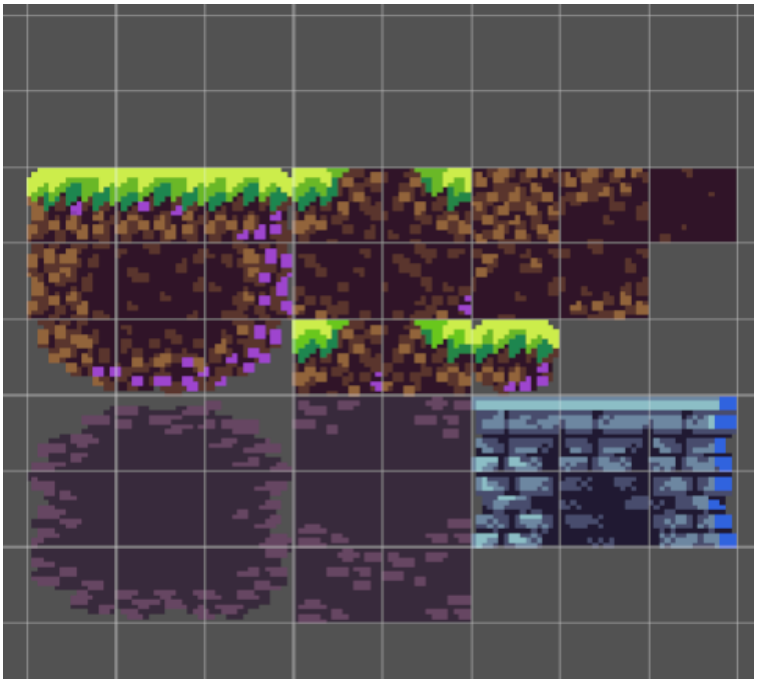
Το Τέλος του Παιχνιδιού

Αφού ο Οδυσσέας καταφέρει να νικήσει τον Κύκλωπα, το παιχνίδι ολοκληρώνεται με μια τελική σκηνή που δείχνει μια σειρά από εικόνες μαζί με τους τίτλους τέλους. Αυτή η σκηνή συμβολίζει την επιθυμία του Οδυσσέα να επιστρέψει στην Ιθάκη μετά από πολλά χρόνια περιπέτειας. Η ιστορία του Οδυσσέα είναι μια ιστορία επιμονής και θάρρους, και το παιχνίδι επιλέγει να τελειώσει με αυτήν την αίσθηση της νίκης και της επιστροφής στην πατρίδα.

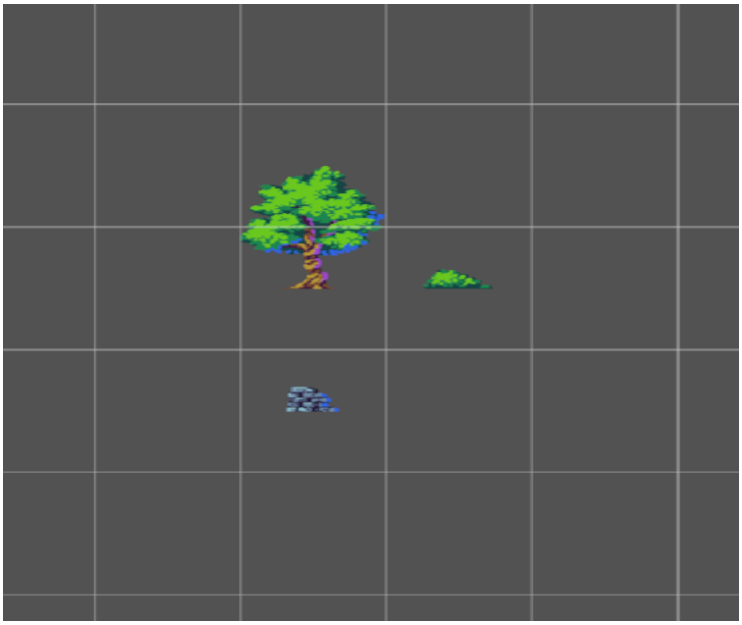
Σχεδίαση & Εκτέλεση

Η γραφική σχεδίαση του παιχνιδιού πραγματοποιήθηκαν με τη χρήση μερικών Tileset. Παρακάτω παρατίθενται εικόνες από τις γραφικές παλέτες της εφαρμογής.

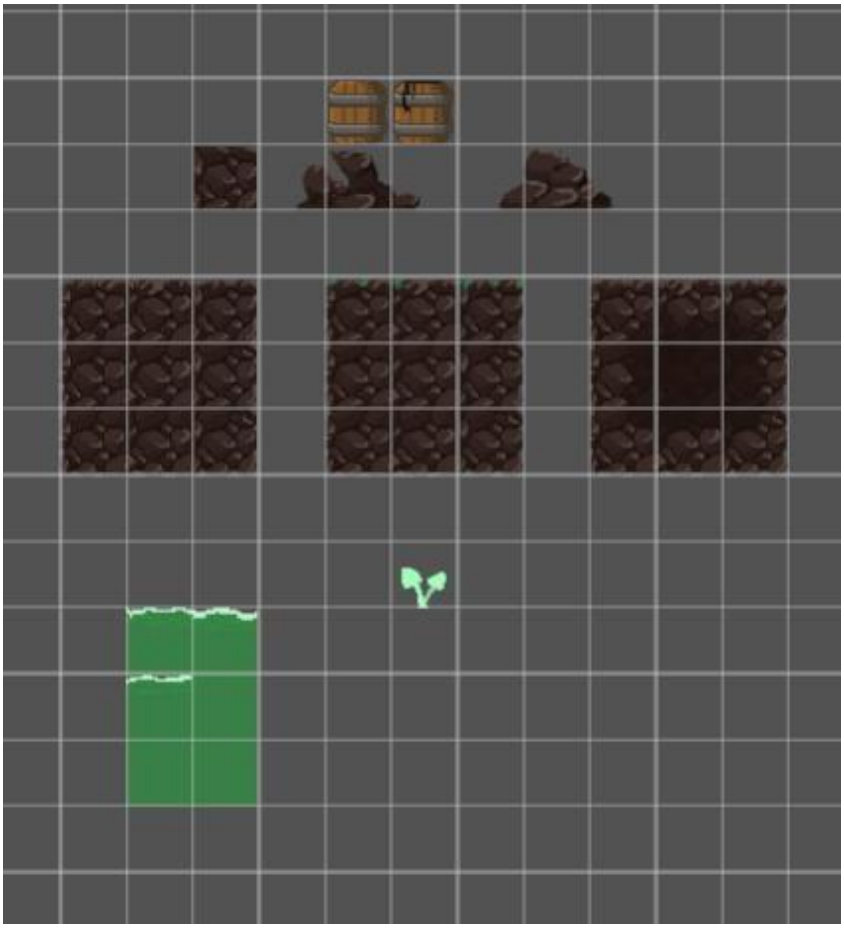
Γραφική Παλέτα 1: ForestTileset



Γραφική Παλέτα 2: ForestDecors



Γραφική Παλέτα 3: CaveTileset



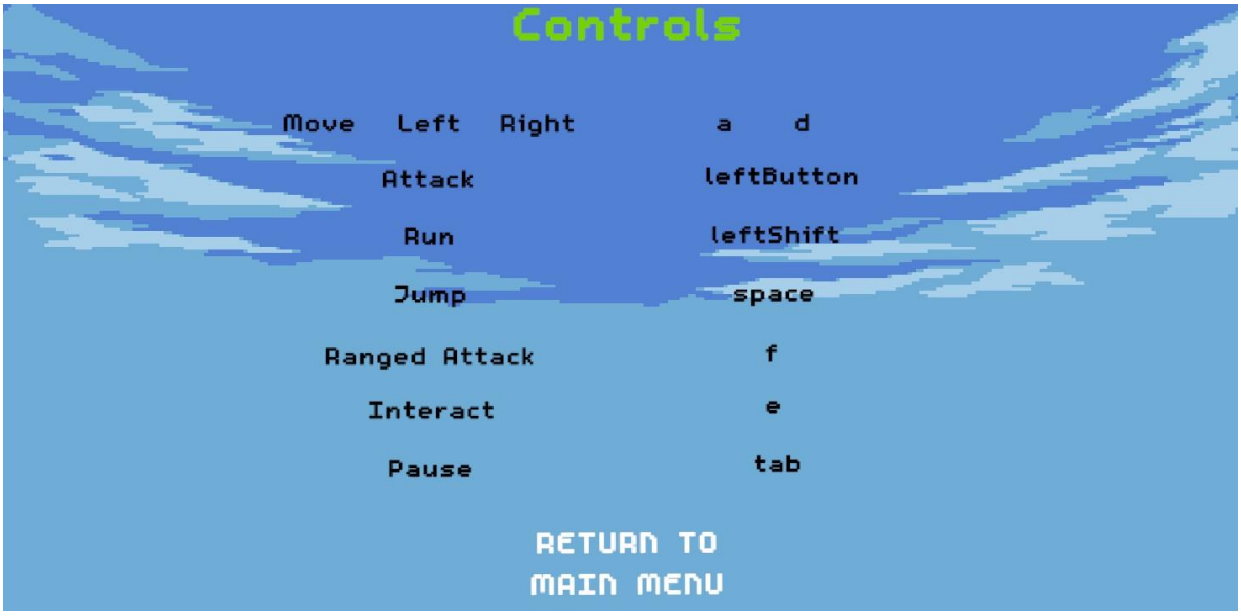
Αρχικές Οθόνες εκκίνησης της εφαρμογής

Ο χρήστης ύστερα από τη προβολή των αρχικών εικόνων της εφαρμογής μεταβαίνει στο μενού έναρξης. Με βάση το παρακάτω μενού ο παίκτης έχει τις εξής επιλογές :

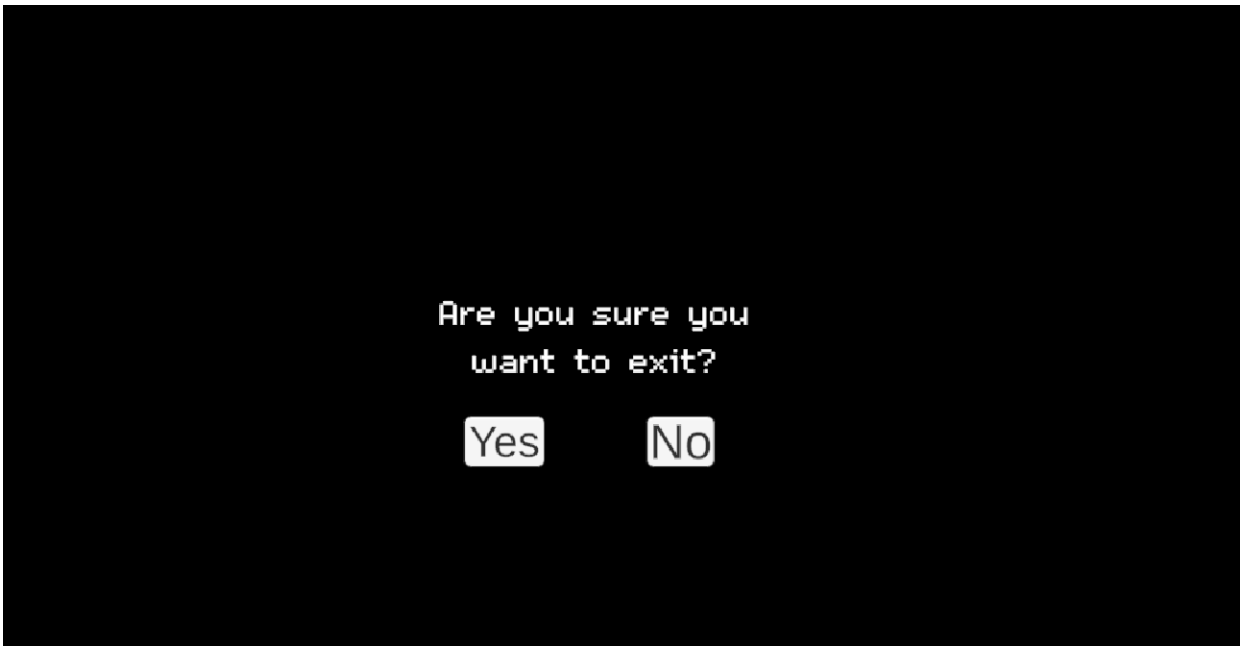
- Να εκκινήσει το παιχνίδι πατώντας το κουμπί 'Start Game'
- Να ανοίξει το μενού επιλογών πατώντας το κουμπί 'Controls'
- Να κλείσει την εφαρμογή πατώντας το κουμπί 'Exit Game'

Στην κάτω δεξιά γωνία της οθόνης, υπάρχει μια μπάρα ρύθμισης της έντασης της μουσικής. Αυτή η μπάρα επιτρέπει στον παίκτη να προσαρμόσει την ένταση του ήχου του παιχνιδιού σύμφωνα με τις προτιμήσεις του. Ο παίκτης μπορεί εύκολα να αυξήσει ή να μειώσει την ένταση, εξασφαλίζοντας ότι η μουσική του παιχνιδιού δεν θα είναι ούτε πολύ δυνατή ούτε πολύ χαμηλή, ανάλογα με τις προσωπικές του προτιμήσεις. Η δυνατότητα αυτή συμβάλλει στη βελτίωση της εμπειρίας παιχνιδιού, καθώς ο παίκτης έχει πλήρη έλεγχο στον ήχο από την αρχή του παιχνιδιού.

Σε περίπτωση που κάνει κλικ στο μενού επιλογών (Controls), τότε θα εμφανιστεί στο παρακάτω καμβά που θα βλέπει όλα τα Controls Που μπορεί να χρησιμοποιήσει ο παίκτης κατά τη διάρκεια του παιχνιδιού, ώστε με αυτό το τρόπο να μπει προετοιμασμένος για να ξεκινήσει το παιχνίδι και εννοείται πως υπάρχει κουμπί επιστροφής στο αρχικό μενού.



Σε περίπτωση που κάνει κλικ στο μενού επιλογών (Exit Game), τότε θα εμφανιστεί στο παρακάτω καμβά βρίσκεται μια ερώτηση επιβεβαίωσης σχετικά με τον αν ο παίκτης όντως θέλει να βγει από το παιχνίδι ή όχι, σε περίπτωση που επιλέξει το Yes θα βγει από το παιχνίδι και θα κλείσει η εφαρμογή αλλά σε περίπτωση που επιλέξει το No θα εμφανιστεί πάλι πίσω στο αρχικό μενού



Σε περίπτωση που ο παίκτης επιλέξει το Start Game, θα μεταφερθεί σε σημείο όπου θα πρέπει να εισάγει ένα όνομα (μπορεί να περιέχει αριθμούς, γράμματα ή οποιοδήποτε άλλο σύμβολο επιθυμεί) και να πατήσει το κουμπί Confirm ώστε να ξεκινήσει το παιχνίδι. Αν δεν συμπληρώσει κάτι στο πεδίο που φαίνεται στην εικόνα, τότε θα του εμφανιστεί κατάλληλο μήνυμα που θα τον προτρέπει να εισάγει ένα όνομα χρήστη. Φυσικά, υπάρχει και κουμπί επιστροφής στο αρχικό μενού.

Μόλις ο παίκτης εισάγει το όνομα χρήστη και πατήσει Confirm, το παιχνίδι θα τον κατευθύνει στο loadingScreen και μόλις αυτό ολοκληρωθεί τότε θα ξεκινήσει το Intro Story, όπου θα εξηγείται το πώς ξεκίνησε το ταξίδι του Οδυσσέα για την επιστροφή του στην Ιθάκη και πώς βρέθηκε σε αυτό το συγκεκριμένο νησί. Όπως έχουμε ήδη αναφέρει στην ιστορία, η πρώτη πίστα διαδραματίζεται στο νησί της Κίρκης. Στο εισαγωγικό κείμενο, περιγράφεται τι θα αντιμετωπίσει ο παίκτης στο νησί αυτό και ποιο είναι το μοναδικό πράγμα που μπορεί να τον βοηθήσει όσον αφορά την τροφή του. Μετά από αυτή την εισαγωγή, το παιχνίδι ξεκινά.

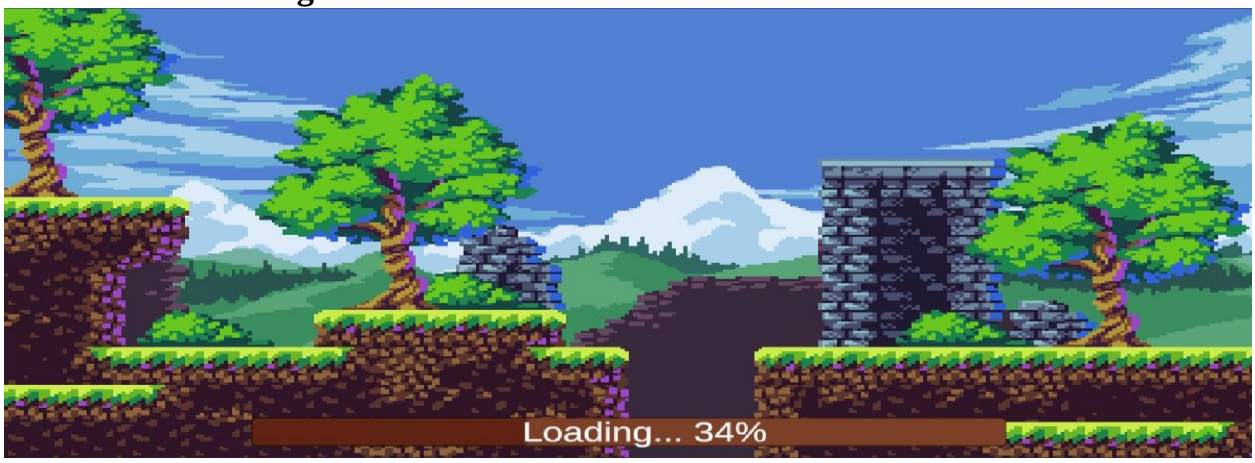
Όταν ολοκληρωθεί το εισαγωγικό στόρι, το παιχνίδι θα ξεκινήσει, και η αγωνία θα κορυφωθεί καθώς ο Οδυσσέας θα προσπαθήσει να επιστρέψει στην πατρίδα του και στους αγαπημένους του ανθρώπους. Αυτό θα γίνει μόνο αν καταφέρει να αντιμετωπίσει αρκετούς από τους εχθρούς που θα βρει στον δρόμο του, όπως τα Goblins, τους Skeletons, τους Knights και τα Flying Eyes.

Καθώς πολεμάει αυτούς τους εχθρούς, ο Οδυσσέας θα πρέπει ταυτόχρονα να αναζητά το πέρασμα για την επόμενη πίστα. Κατά τη διάρκεια του παιχνιδιού, δίνονται οδηγίες για το πού πρέπει να πάει, οι οποίες είναι ενσωματωμένες στο περιβάλλον του χάρτη, είτε με εικόνες είτε με χτισμένα Tilesets και Backgrounds που καθοδηγούν τον παίκτη.

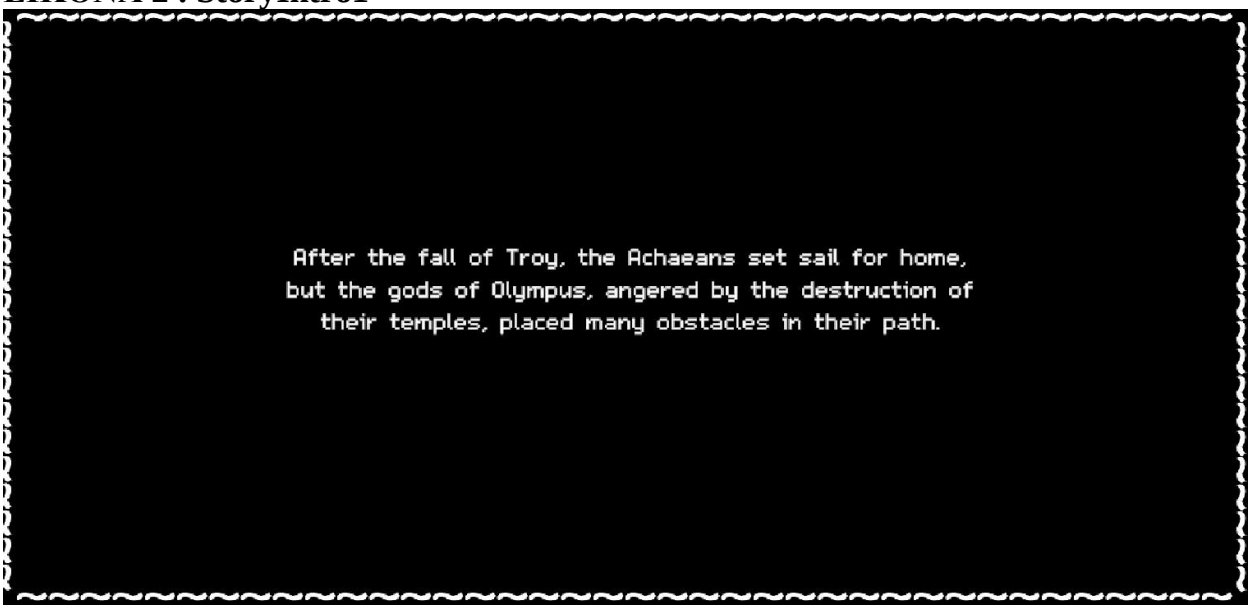
Προχωρώντας στο παιχνίδι, ο Οδυσσέας θα βρει μια πύλη, η οποία όμως θα είναι κλειδωμένη. Πατώντας το κουμπί Interact, θα εμφανιστεί ένα Hint, που θα τον βοηθήσει να καταλάβει τι πρέπει να κάνει. Ο μόνος τρόπος για να ανοίξει την πύλη είναι να κατέβει στον κάτω κόσμο, τον κόσμο του Άδη, για να βρει το κλειδί που ζητά η πύλη.

Εκεί, όμως, τα πράγματα θα είναι πολύ πιο δύσκολα, καθώς θα συναντήσει περισσότερα τέρατα και εχθρούς, μιας και βρίσκεται πλέον στον κόσμο του Άδη. Αν καταφέρει να βρει το κλειδί που αναζητά και επιστρέψει στο σημείο όπου βρίσκεται η πύλη, θα μπορέσει να την ξεκλειδώσει και να περάσει στη δεύτερη πίστα.

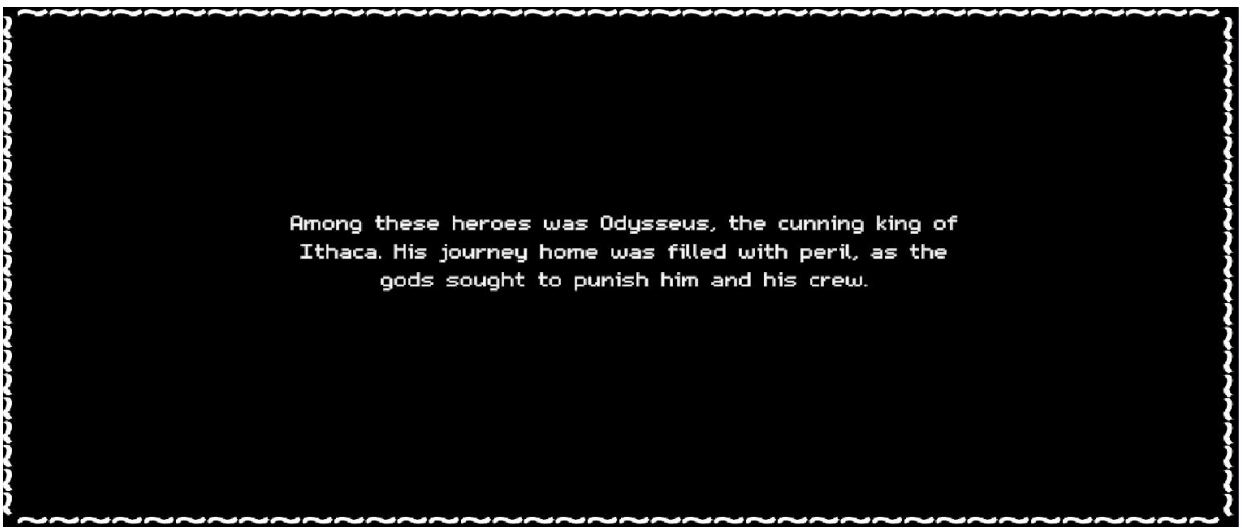
ΕΙΚΟΝΑ 1 : LoadingScreen1



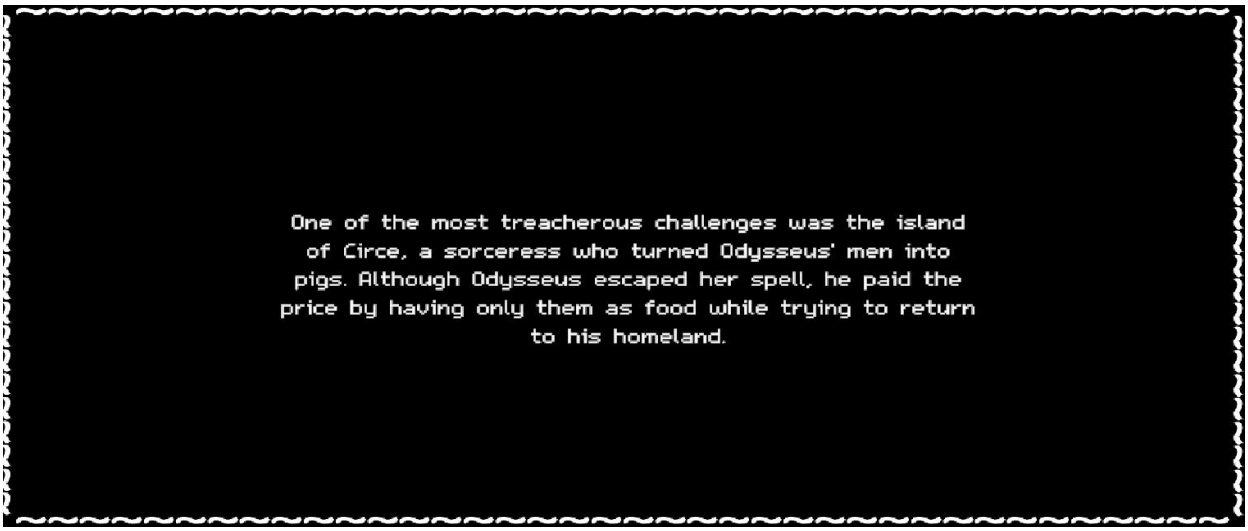
ΕΙΚΟΝΑ 2 : StoryIntro1



ΕΙΚΟΝΑ 3 : StoryIntro2



ΕΙΚΟΝΑ 4 : StoryIntro3



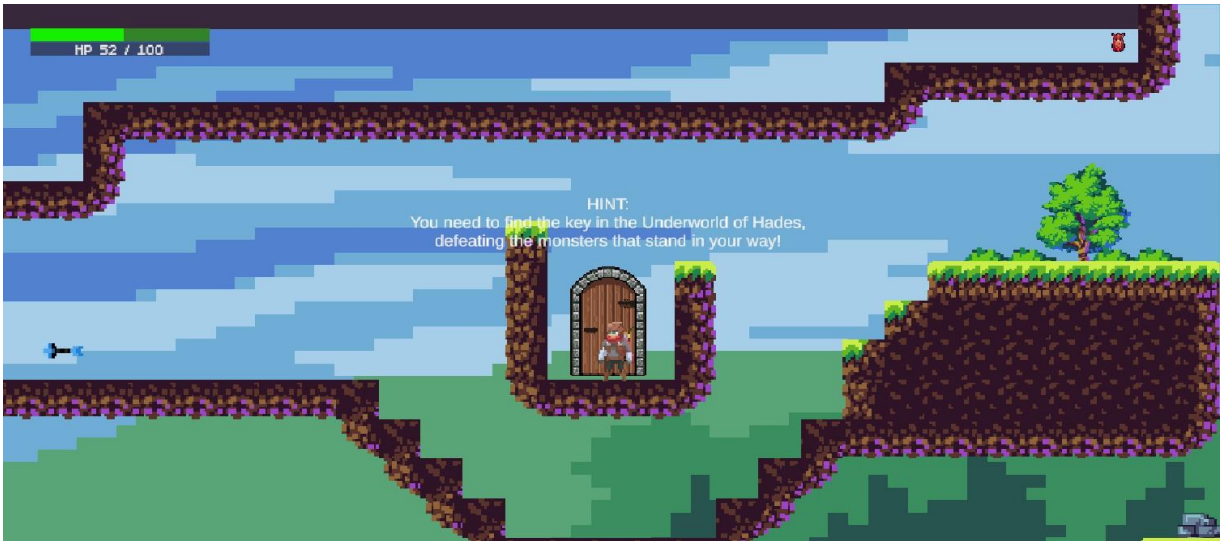
ΕΙΚΟΝΑ 5 : Δεξιότητα αναπήδησης

ΕΙΚΟΝΑ 6 : Δεξιότητα επίθεσης

ΕΙΚΟΝΑ 7 : Δεξιότητα εναέριας επίθεσης

ΕΙΚΟΝΑ 8 : Δεξιότητα επίθεσης με τόξο

ΕΙΚΟΝΑ 9 : Hint after interact with the door



EIKONA 10 : Knight enemy

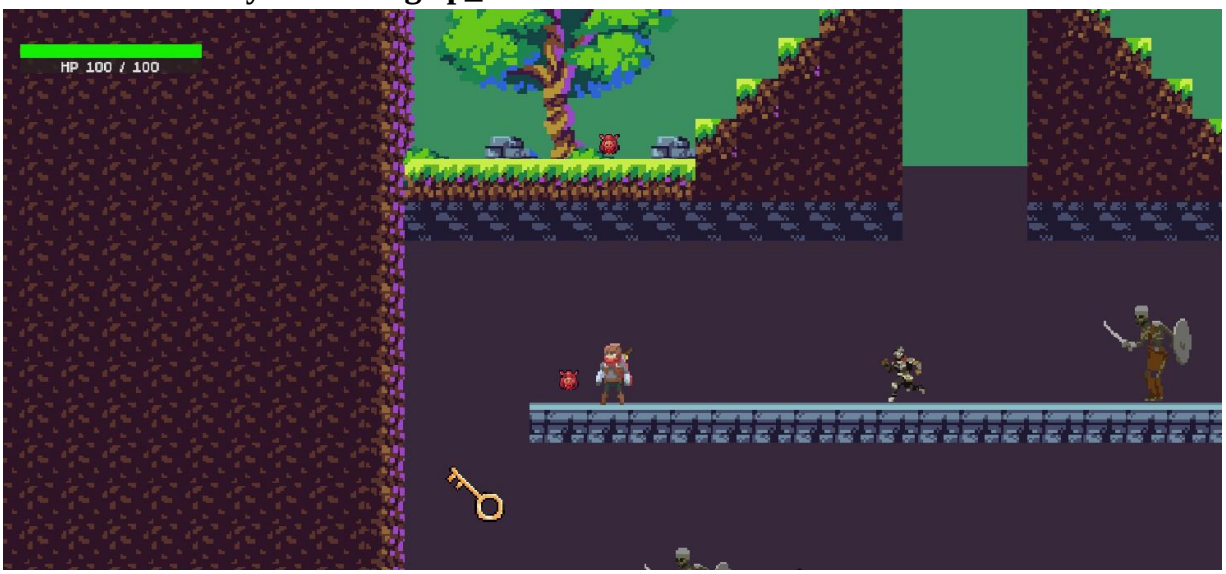
EIKONA 11 : FlyingEye enemy

EIKONA 12 : Skeleton enemy

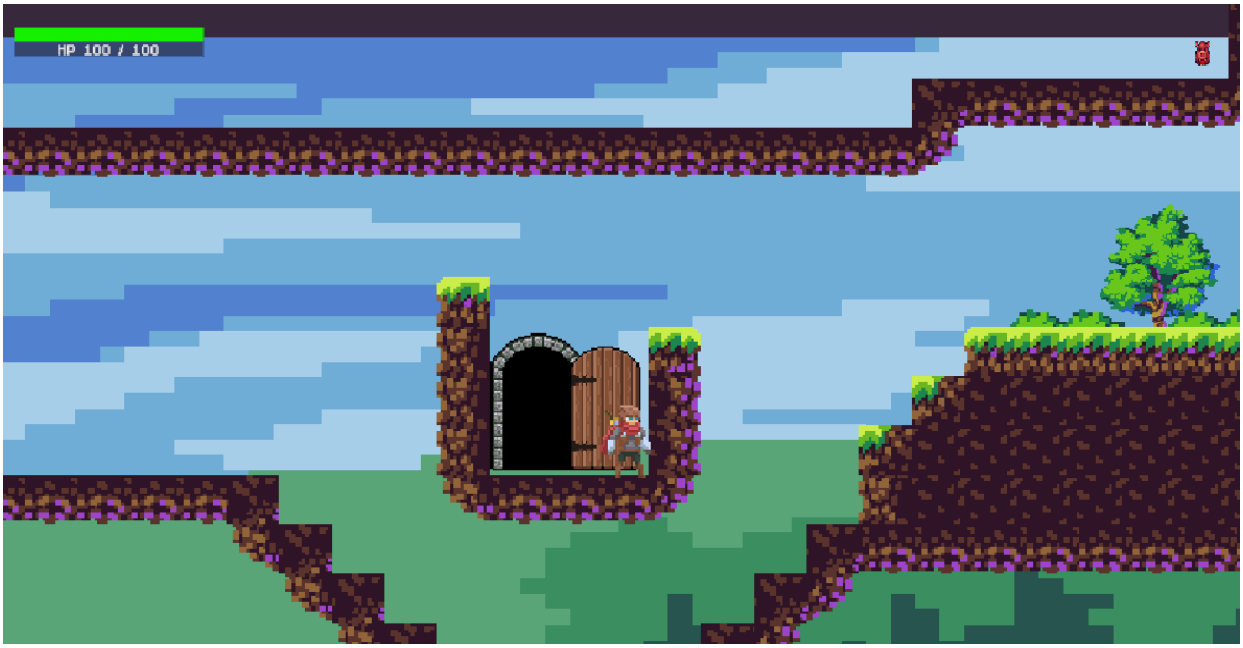
EIKONA 13 : MapHint1

EIKONA 14 : MapHint2

EIKONA 15: Key & HealingUp_Item



ΕΙΚΟΝΑ 15: OpenDoor



Μετά την ολοκλήρωση της πρώτης πίστας και τη χρήση του κλειδιού για να ανοίξει την πύλη, ο Οδυσσέας περνά στη δεύτερη πίστα. Το παιχνίδι τότε κατευθύνει τον παίκτη στο δεύτερο loading screen, και μόλις αυτό ολοκληρωθεί, ξεκινά το δεύτερο Story, το οποίο αφηγείται το επόμενο κομμάτι του ταξιδιού του Οδυσσέα.

Σύμφωνα με το StoryCave, ο Οδυσσέας, έχοντας ξεφύγει από το νησί της Κίρκης, συνεχίζει το ταξίδι του μόνος, φτάνοντας σε ένα μυστηριώδες νησί, γεμάτο σκιές και μύθους. Σε αυτό το νησί, κατοικεί ο τρομερός Κύκλωπας, γιος του Ποσειδώνα, ο οποίος καταβροχθίζει όποιον τολμήσει να πλησιάσει.

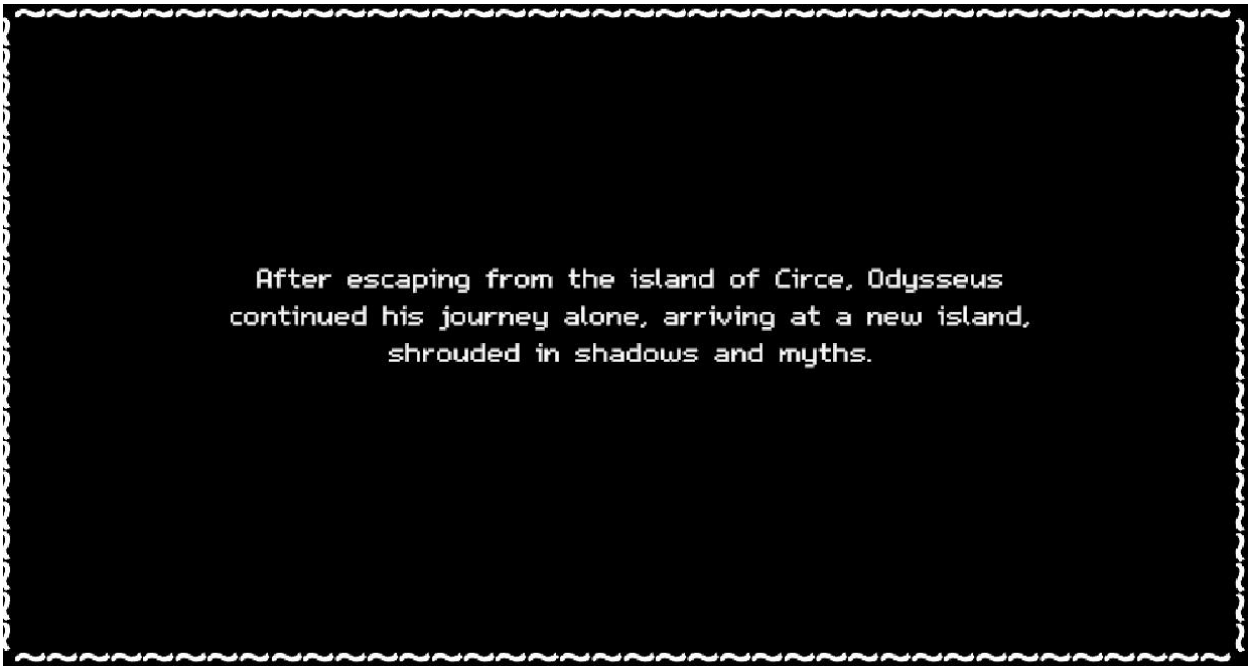
Καθώς πλησιάζει στη σπηλιά του Κύκλωπα, η ένταση αυξάνεται, καθώς ο Οδυσσέας γνωρίζει ότι η επόμενη μάχη θα είναι η πιο δύσκολη και επικίνδυνη που έχει αντιμετωπίσει μέχρι τώρα. Ξέρει ότι μόνο με τη φρόνηση και την εξυπνάδα του θα μπορέσει να νικήσει αυτό το τερατώδες πλάσμα και να βρει τον δρόμο της επιστροφής για την Ιθάκη.

Αυτό το εισαγωγικό κείμενο προετοιμάζει τον παίκτη για τη δεύτερη πίστα, που διαδραματίζεται στη σπηλιά του Κύκλωπα, όπου ο Οδυσσέας θα πρέπει να χρησιμοποιήσει όλες του τις δυνάμεις και στρατηγικές για να επιβιώσει και να νικήσει τον πανίσχυρο Κύκλωπα.

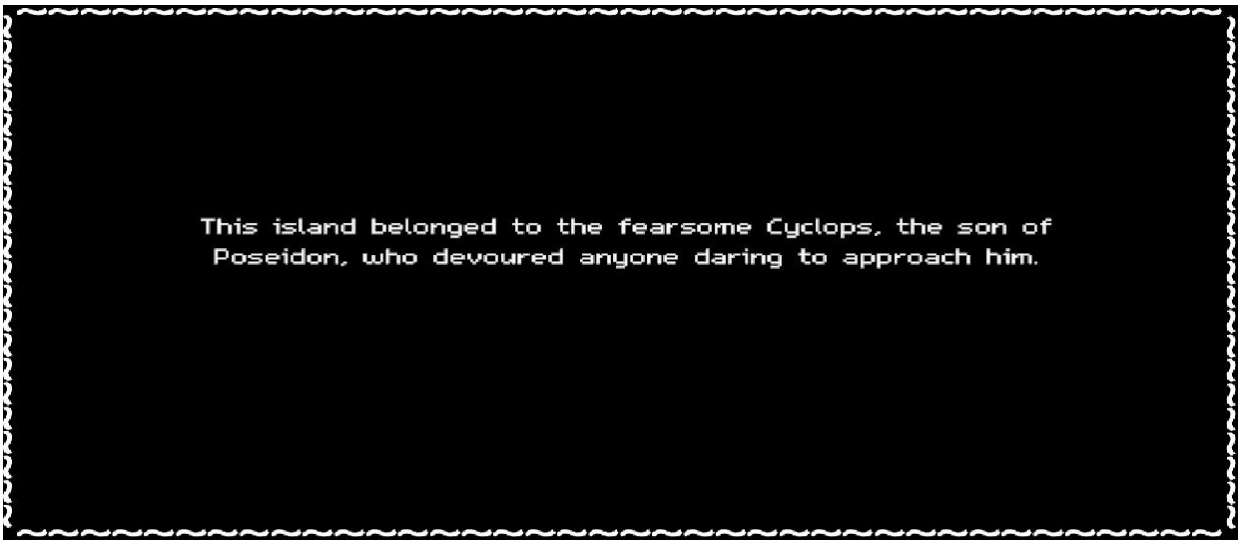
Αυτό το σημείο της σχεδίασης ενσωματώνει τις εικόνες και τα backgrounds με ενσωματωμένα hints και οδηγίες για το που πρέπει να πάει ο παίκτης, καθιστώντας τη ροή του παιχνιδιού καθαρή και συνεκτική, επιτρέποντας στον παίκτη να προχωρήσει με στόχο τη νίκη.

EIKONA 16 : LoadingScreen2

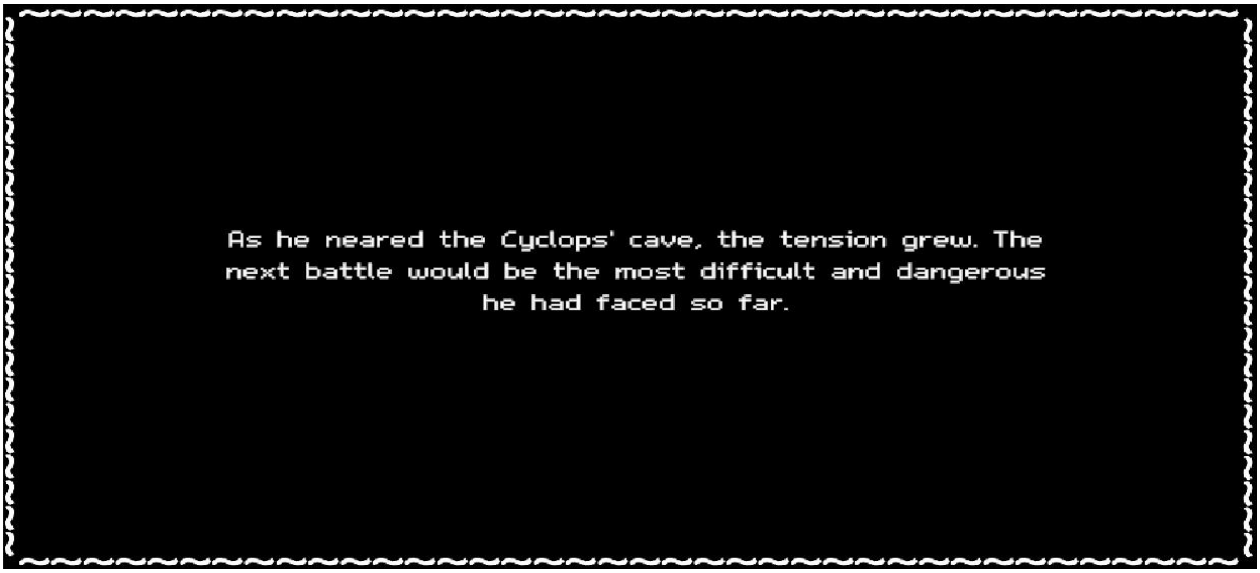
EIKONA 17 : StoryCave1



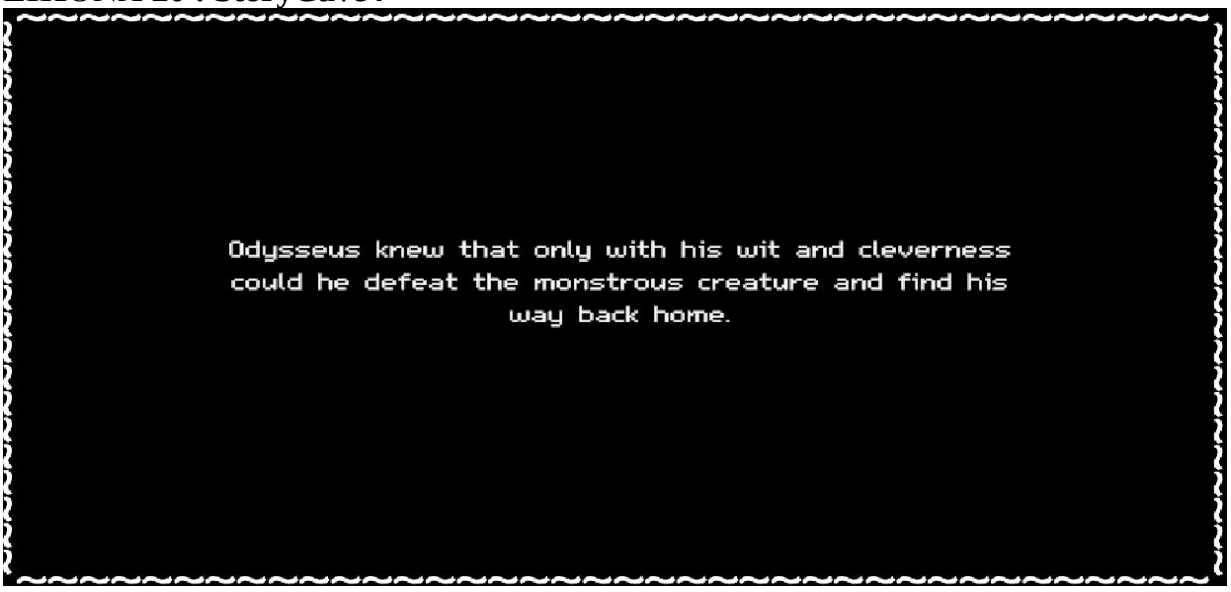
EIKONA 18 : StoryCave2



EIKONA 19 : StoryCave3



EIKONA 20 : StoryCave4



EIKONA 21 : CaveStart

EIKONA 22 : Goblin enemy

EIKONA 23 : MapHint

EIKONA 24 : BossFight

ΕΙΚΟΝΑ 25 : CyclopsIsDead

Μόλις ο παίκτης μπει στην περιοχή του Boss Fight Zone, δεν υπάρχει επιστροφή. Ο Οδυσσέας έχει πλέον παγιδευτεί, και ο μόνος τρόπος για να συνεχίσει το ταξίδι του είναι να αντιμετωπίσει τον τρομερό Κύκλωπα. Ο Κύκλωπας είναι ένας πανίσχυρος αντίπαλος, γιγαντιαίος και αδυσώπητος, γιος του Ποσειδώνα, γνωστός για τη βίαιη φύση του και τη σκληρότητά του. Η μάχη με αυτόν τον εχθρό θα είναι η πιο δύσκολη δοκιμασία του Οδυσσέα μέχρι στιγμής. Μόνο με εξυπνάδα και στρατηγική θα μπορέσει να βγει νικητής.

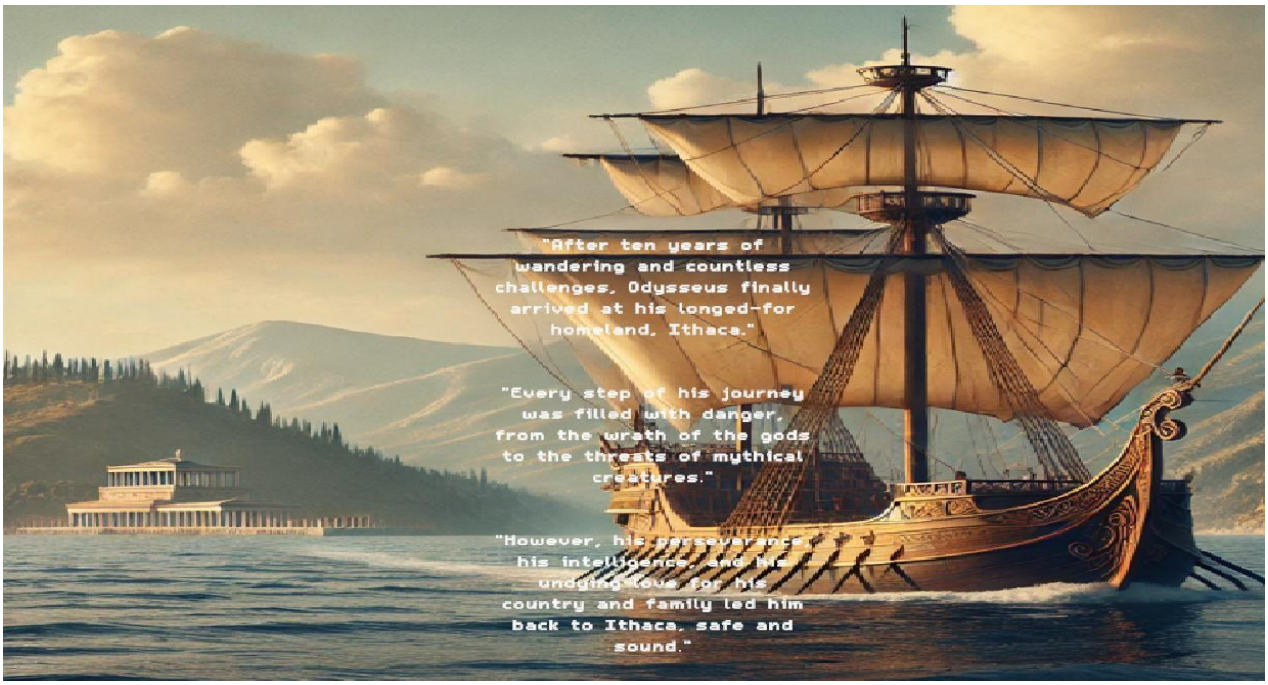
Εφόσον καταφέρει να νικήσει τον Κύκλωπα, ξεκινά ένα cutscene, όπου το παιχνίδι επιβραδύνει για να δημιουργήσει την απαραίτητη ένταση και τον δραματικό τόνο. Στα επόμενα δευτερόλεπτα, η σκηνή μεταβαίνει αργά στη τελευταία σκηνή του παιχνιδιού, το EndStory. Εκεί, παρακολουθούμε τους "τίτλους τέλους" να κυλούν, συνοδευόμενοι από εικόνες που αλλάζουν σταδιακά, παρουσιάζοντας τις τελευταίες στιγμές του Οδυσσέα καθώς επιστρέφει στην Ιθάκη.

Στην τελευταία εικόνα, βλέπουμε τον Οδυσσέα με την οικογένειά του, μαζεμένους όλοι μαζί στο σπίτι τους γύρω από τη φωτιά. Ο Οδυσσέας, γεμάτος εμπειρίες και αναμνήσεις από τις περιπέτειες του ταξιδιού του, αφηγείται στους αγαπημένους του όσα έζησε, ενώ η εικόνα συμβολίζει την τελική του νίκη, όχι μόνο απέναντι στους εχθρούς του, αλλά και απέναντι στις αντιξοότητες που αντιμετώπισε για να επιστρέψει στο σπίτι του.

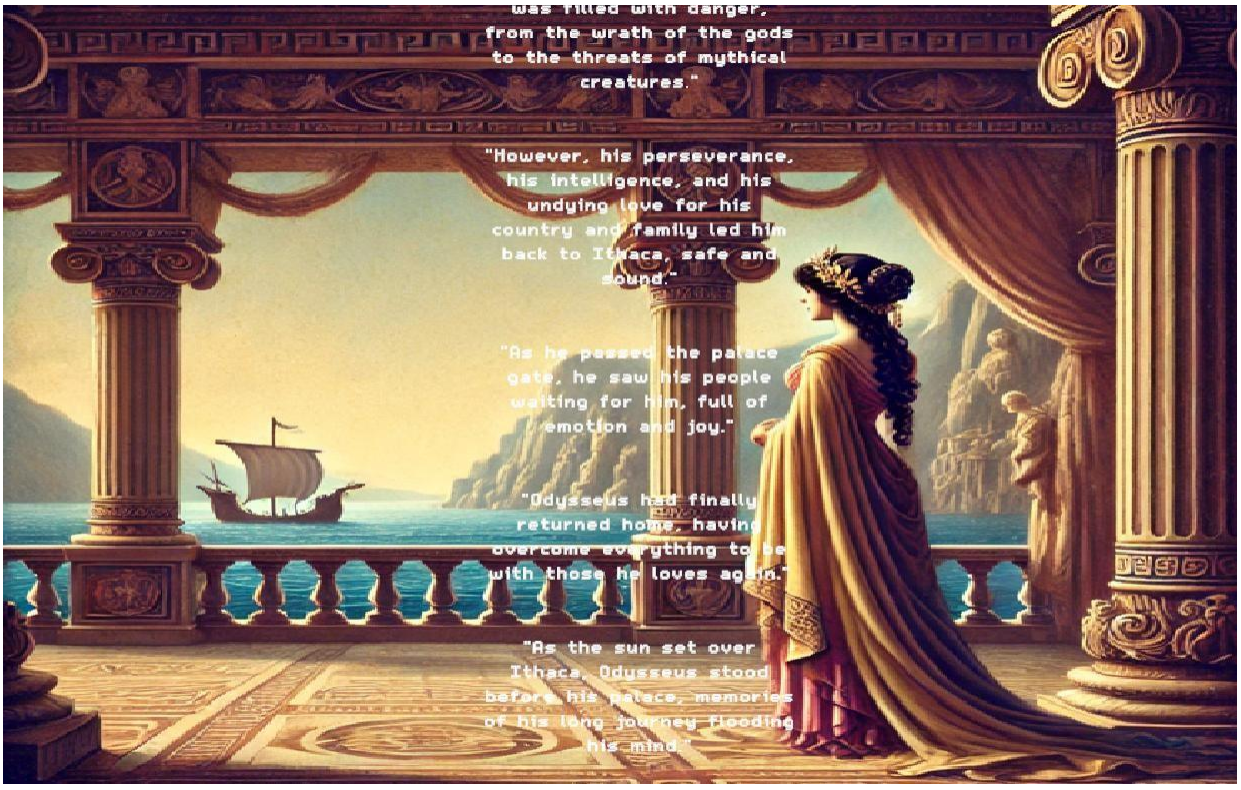
Κατά τη διάρκεια αυτής της σκηνής, η εικόνα της οικογένειας μένει σταθερή στο παρασκήνιο, ενώ οι τίτλοι τέλους κυλούν αργά στην οθόνη, παρουσιάζοντας όλους τους συντελεστές του παιχνιδιού.

Μόλις ολοκληρωθεί το animation των credits, το παιχνίδι τελειώνει, φέρνοντας το main menu στην τελική οθόνη, αφήνοντας μια αίσθηση επιτυχίας και ολοκλήρωσης της περιπέτειας του Οδυσσέα. Έτσι, το The Adventures of Odysseus φτάνει στο τέλος του, με την ιστορία του ήρωα να έχει ολοκληρωθεί επιτυχώς.

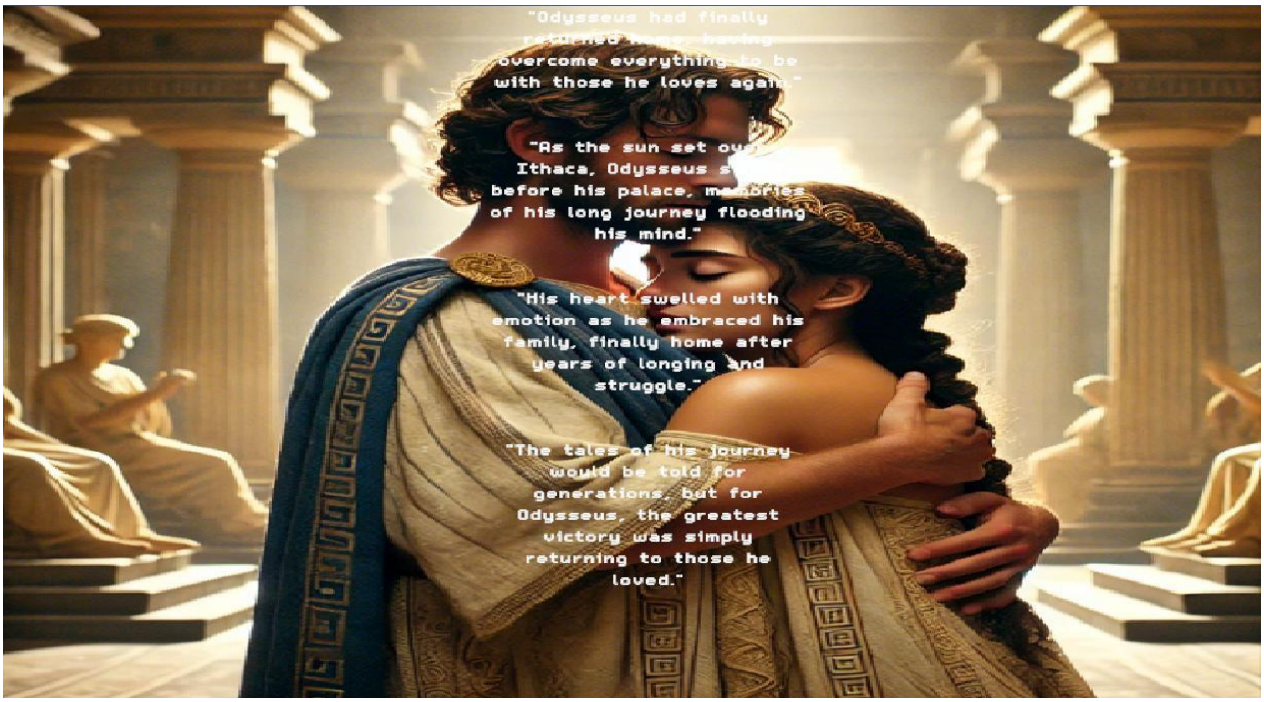
ΕΙΚΟΝΑ 26 : ReturnToIthaca1



ΕΙΚΟΝΑ 27 : ReturnToIthaca2

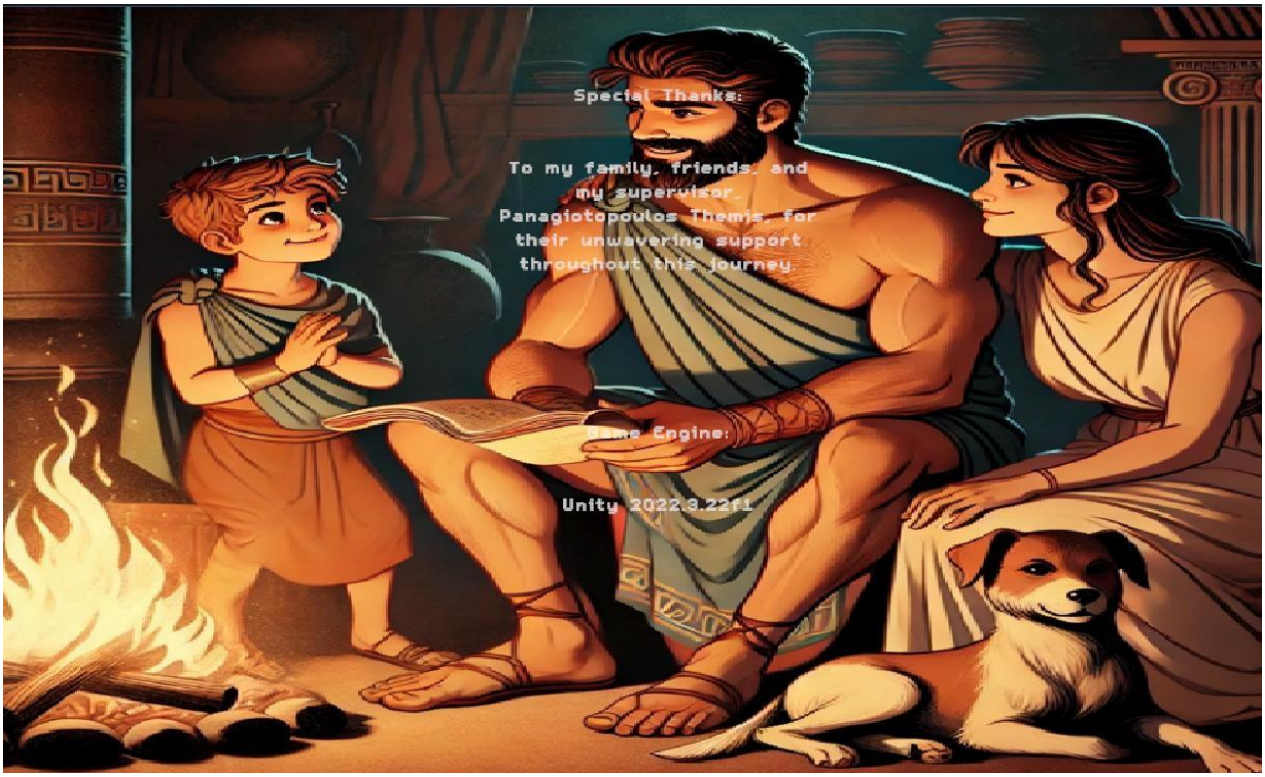


EIKONA 28 : ReturnToIthaca3

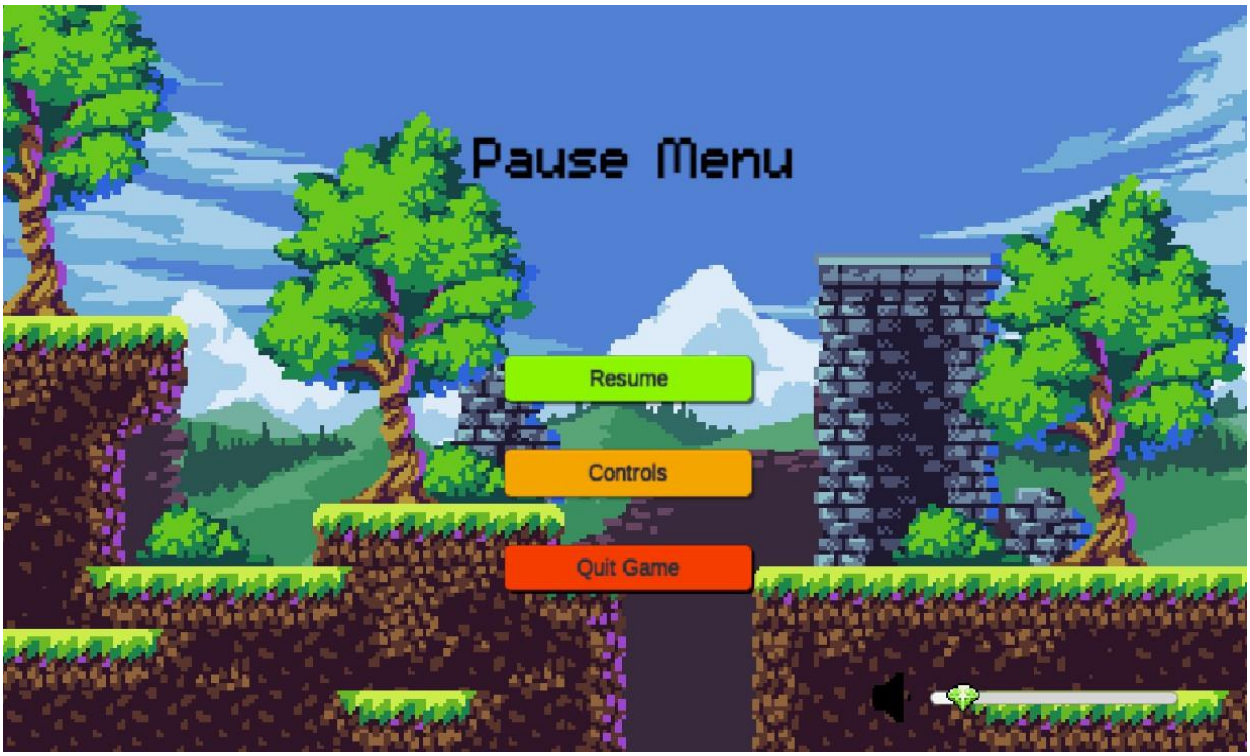


EIKONA 29 : ReturnToIthaca4

EIKONA 30 : ReturnToIthaca5



EIKONA 31: PauseMenu



Στο Pause Menu, όταν επιλέξουμε την επιλογή Controls, υπάρχει μια σημαντική διαφορά σε σχέση με την αντίστοιχη επιλογή στο Main Menu. Αυτή τη φορά, ο παίκτης έχει τη δυνατότητα να αλλάξει τα πλήκτρα και να τα προσαρμόσει όπως εκείνος επιθυμεί, με την προϋπόθεση ότι δεν μπορεί να επιλέξει κάποιο πλήκτρο που χρησιμοποιείται ήδη από άλλη ενέργεια. Αν προσπαθήσει να το κάνει, το παιχνίδι δεν θα του το επιτρέψει.

Επιπλέον, υπάρχει το κουμπί Reset Controls, το οποίο επαναφέρει τα πλήκτρα στις προκαθορισμένες ρυθμίσεις του παιχνιδιού, προσφέροντας στον παίκτη τη δυνατότητα να επιστρέψει στις αρχικές επιλογές αν χρειαστεί.

Επίσης, όπως και στο Main Menu, υπάρχει μια μπάρα στην κάτω δεξιά γωνία, η οποία επιτρέπει στον παίκτη να ρυθμίσει την ένταση της μουσικής του παιχνιδιού. Αυτό δίνει τη δυνατότητα στον παίκτη να αυξήσει ή να μειώσει την ένταση ανάλογα με τις προτιμήσεις του, εξασφαλίζοντας την καλύτερη δυνατή εμπειρία κατά τη διάρκεια του παιχνιδιού.

ΕΙΚΟΝΑ 32 : DeathPanel

ΥΛΟΠΟΙΗΣΗ

ΠΙΝΑΚΑΣ ΜΕ ΤΑ SCRIPTS

RebindButton.cs	Διαχειρίζεται την αλλαγή των πλήκτρων/κουμπιών εισόδου από τον παίκτη, επιτρέποντας την αποθήκευση και ενημέρωση των αλλαγών.
ResetBindings.cs	Επαναφέρει όλες τις ρυθμίσεις πλήκτρων/κουμπιών στην αρχική τους κατάσταση, διαγράφοντας οποιεσδήποτε αλλαγές.
BossFightZone.cs	Διαχειρίζεται τη ζώνη μάχης του boss, ενεργοποιεί την μάχη, σταματάει τη μουσική και ελέγχει τους περιορισμούς του χώρου.
CyclopsController.cs	Ελέγχει τη συμπεριφορά του Κυκλώπα, περιλαμβάνοντας την κίνηση, τις επιθέσεις και τη λογική σύγκρουσης με τον παίκτη.
CyclopsHealthBarController.cs	Διαχειρίζεται τη μπάρα υγείας του Κυκλώπα, ενημερώνοντάς την καθώς δέχεται ζημιά και εμφανίζει το κείμενο HP.
TouchingDirectionsCyclops.cs	Ανιχνεύει την επαφή του Κυκλώπα με το έδαφος, τους τοίχους και το ταβάνι για να βελτιώσει την κίνηση και τις αλληλεπιδράσεις.
CharacterEvents.cs	Δημιουργεί Unity events που ενεργοποιούνται όταν ο χαρακτήρας δέχεται ζημιά ή θεραπεύεται, παρέχοντας έτσι μεγαλύτερη ευελιξία.
LoadingScreenManager1.cs	Διαχειρίζεται την εμφάνιση οθονών φόρτωσης κατά τη μετάβαση μεταξύ σκηνών, παρέχοντας ομαλότερη εμπειρία στον παίκτη (έκδοση 1).
LoadingScreenManager2.cs	Μια εναλλακτική υλοποίηση οθονών φόρτωσης με διαφορετικές ρυθμίσεις και animations για τη φόρτωση των σκηνών (έκδοση 2).
MainMenuController.cs	Διαχειρίζεται το κεντρικό μενού, επιτρέποντας στον παίκτη να ξεκινήσει το παιχνίδι, να προσαρμόσει τις ρυθμίσεις ή να βγει.
PauseMenuController.cs	Επιτρέπει την παύση του παιχνιδιού, την πρόσβαση στο μενού ρυθμίσεων και τη διαχείριση της έντασης μουσικής.

MusicPlayer.cs	Ελέγχει την αναπαραγωγή της μουσικής στο παρασκήνιο, παρέχοντας επιλογές παύσης, συνέχισης και ρύθμισης της έντασης.
FadeRemoveBehaviour.cs	Προσθέτει εφέ σταδιακής εξαφάνισης (fade out) σε ένα αντικείμενο πριν αυτό αφαιρεθεί από τη σκηνή.
PlayOneShotBehaviour.cs	Παίζει ένα ηχητικό κλιπ κατά τη διάρκεια της αλλαγής των animations, βοηθώντας στην ενίσχυση της εμπειρίας του παίκτη.
SetBoolBehaviour.cs	Ελέγχει τις boolean παραμέτρους στο Animator, ενεργοποιώντας διαφορετικά animations με βάση την κατάσταση του χαρακτήρα.
SetFloatBehaviour.cs	Ελέγχει τις float παραμέτρους στο Animator, επιτρέποντας πιο ομαλές μεταβάσεις μεταξύ animations.
EndStoryManager.cs	Διαχειρίζεται το τέλος της ιστορίας, κυλώντας κείμενα στην οθόνη και αλλάζοντας εικόνες, μέχρι να επιστρέψει στο κύριο μενού.
StoryManagerCave.cs	Παρουσιάζει τις σελίδες της ιστορίας καθώς ο παίκτης προχωρά, με δυνατότητα παράκαμψης και μετάβασης στην επόμενη σκηνή.
StoryManagerIntro.cs	Εμφανίζει την εισαγωγή της ιστορίας του παιχνιδιού, με δυνατότητα παράκαμψης και μετάβασης στην κύρια σκηνή παιχνιδιού.
AnimationStrings.cs	Περιέχει σταθερές συμβολοσειρές για τις παραμέτρους του Animator, μειώνοντας λάθη κατά την αναφορά τους σε διάφορα scripts.
Attack.cs	Χειρίζεται τις επιθέσεις του χαρακτήρα και εφαρμόζει ανάκρουση στους εχθρούς, ενώ παρέχει και διαχείριση ζημιάς.
CameraFollow.cs	Η κάμερα ακολουθεί τον παίκτη, διατηρώντας μια σταθερή απόσταση (offset), βελτιώνοντας την οπτική εμπειρία.
Damageable.cs	Διαχειρίζεται την υγεία των χαρακτήρων, την εφαρμογή ζημιάς και την κατάσταση ατρωσίας, ενώ ενεργοποιεί animations θανάτου.
DetectionZone.cs	Ανιχνεύει τα αντικείμενα που εισέρχονται σε μια ζώνη ανίχνευσης,

	χρησιμοποιώντας τα για να ενεργοποιήσει συγκεκριμένες ενέργειες.
FlyingEye.cs	Ελέγχει την κίνηση ενός ιπτάμενου εχθρού και τον καθοδηγεί μέσα από προκαθορισμένα σημεία (waypoints) για επιθέσεις.
Goblin.cs	Ελέγχει τον εχθρό Goblin, την ανίχνευση του παίκτη, την κατεύθυνση της κίνησης και τις επιθέσεις του.
Skeleton.cs	Ελέγχει τον εχθρό Skeleton, επιτρέποντάς του να επιτίθεται, να ανιχνεύει τον παίκτη και να κινείται στη σκηνή.
Knight.cs	Ελέγχει τον εχθρό Knight, συμπεριλαμβανομένων των επιθέσεων, της ανίχνευσης του παίκτη και της κίνησης σε συγκεκριμένες κατευθύνσεις.
ParallaxEffect.cs	Δημιουργεί το εφέ parallax στα φόντα, προσθέτοντας βάθος στην οπτική εμπειρία του παίκτη καθώς κινείται στη σκηνή.
PlayerController.cs	Ο κύριος έλεγχος του παίκτη, διαχειρίζεται την κίνηση, τις επιθέσεις και τις αλληλεπιδράσεις με το περιβάλλον.
Projectile.cs	Δημιουργεί και διαχειρίζεται βλήματα, επιτρέποντας την πρόκληση ζημιάς σε εχθρούς και την εφαρμογή ανάκρουσης.
ProjectileLauncher.cs	Εκτοξεύει βλήματα από ένα καθορισμένο σημείο εκκίνησης, υπολογίζοντας τη διεύθυνση και την κίνηση τους.
TouchingDirections.cs	Ανιχνεύει αν ο χαρακτήρας είναι στο έδαφος, σε τοίχο ή ταβάνι, για να ρυθμίσει κατάλληλα τις κινήσεις και τα animations του.
HealthText.cs	Εμφανίζει κείμενο που δείχνει την ποσότητα θεραπείας ή ζημιάς που λαμβάνει ο χαρακτήρας, με κίνηση και fade-out.
HealthPickup.cs	Παρέχει στον παίκτη αποκατάσταση υγείας όταν συλλεχθεί, συνοδευόμενο από ηχητικά εφέ.
KeyPickup.cs	Ελέγχει τη συλλογή κλειδιών από τον παίκτη, τα οποία χρησιμοποιούνται για το ξεκλείδωμα περιοχών στο παιχνίδι.

UIManager.cs	Διαχειρίζεται την εμφάνιση γραφικών στοιχείων (UI), όπως κείμενα ζημιάς/θεραπείας, και επιτρέπει την έξοδο από το παιχνίδι.
HealthBar.cs	Ελέγχει τη μπάρα υγείας του παίκτη, ενημερώνοντάς την δυναμικά όταν η υγεία αυξάνεται ή μειώνεται.

ΣΗΜΑΝΤΙΚΑ SCRIPTS

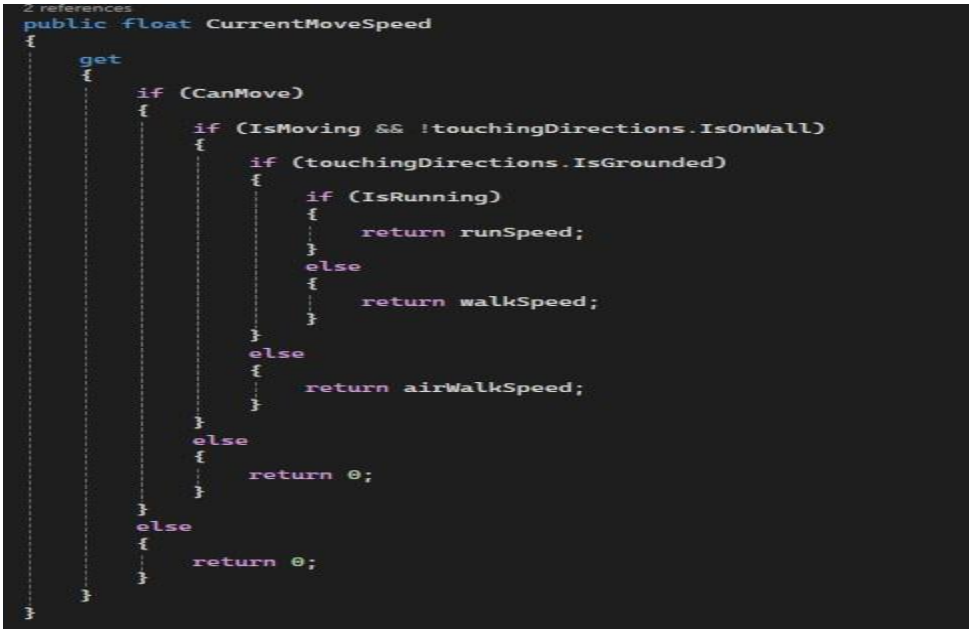
PlayerController.cs

Στην αρχή του κώδικα, έχουμε τη δήλωση όλων των μεταβλητών και των components που θα χρησιμοποιηθούν για τον έλεγχο του παίκτη:

```
public float walkSpeed = 5f;
public float runSpeed = 8f;
public float airWalkSpeed = 3f;
public float jumpImpulse = 10f;

public GameObject gameOverPanel;
public float deathPanelDelay = 2f;
public bool isPlayerDead = false;
public string lastSceneName;
public Volume postProcessingVolume;
public DepthOfField depthOfField;

public bool moveLeft;
public bool moveRight;
```

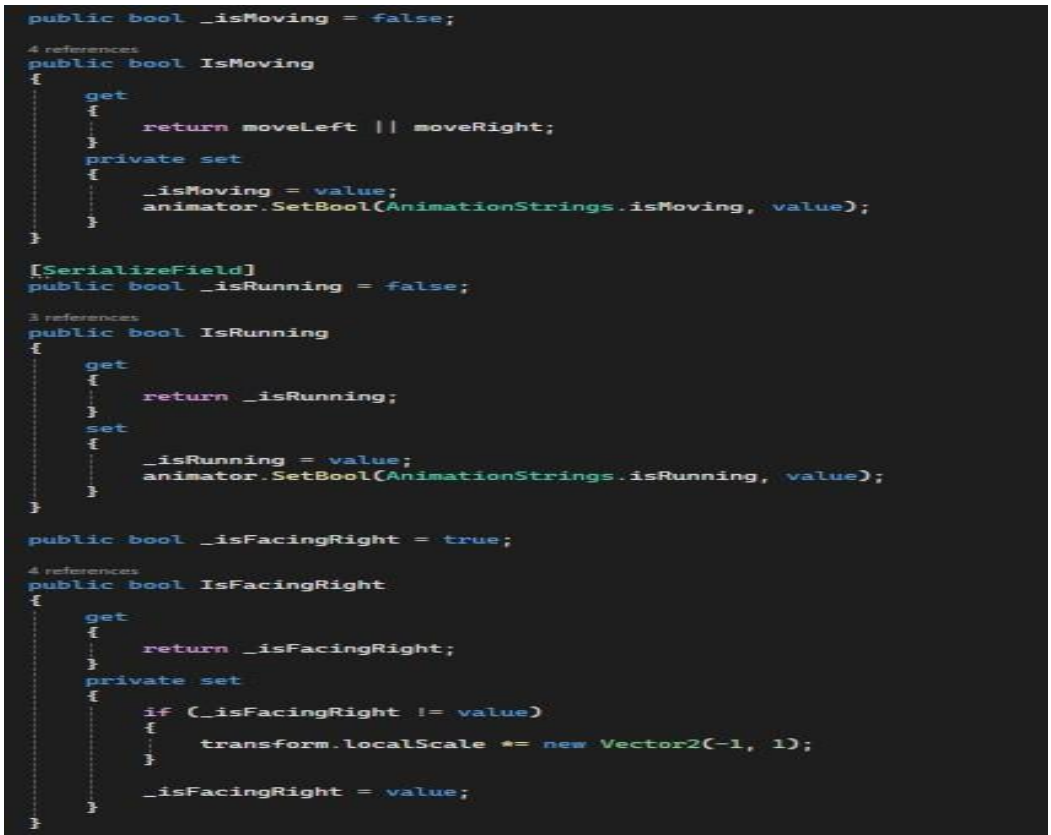
- walkSpeed, runSpeed, airWalkSpeed: Ορίζουν την ταχύτητα του παίκτη κατά την κίνηση στο έδαφος (περπάτημα/τρέξιμο) και στον αέρα.
- jumpImpulse: Η δύναμη του άλματος του παίκτη.
- gameOverPanel: Αναφορά στο πάνελ που εμφανίζεται όταν ο παίκτης πεθαίνει.
- deathPanelDelay: Ορίζει το χρονικό διάστημα καθυστέρησης πριν εμφανιστεί το gameOverPanel.
- postProcessingVolume: Χρησιμοποιείται για να ελέγξουμε τα οπτικά εφέ μετά την επεξεργασία, όπως το βάθος πεδίου (DepthOfField).
- moveLeft, moveRight: Λογικές τιμές που δείχνουν αν ο παίκτης κινείται αριστερά ή δεξιά.

Ακολουθούν οι ιδιότητες και οι μέθοδοι που ελέγχουν την κίνηση του παίκτη



```
2 references
public float CurrentMoveSpeed
{
    get
    {
        if (CanMove)
        {
            if (IsMoving && !touchingDirections.IsOnWall)
            {
                if (touchingDirections.IsGrounded)
                {
                    if (IsRunning)
                    {
                        return runSpeed;
                    }
                    else
                    {
                        return walkSpeed;
                    }
                }
                else
                {
                    return airWalkSpeed;
                }
            }
            else
            {
                return 0;
            }
        }
        else
        {
            return 0;
        }
    }
}
```

- CurrentMoveSpeed: Επιστρέφει την ταχύτητα με την οποία κινείται ο παίκτης. Ελέγχεται αν ο παίκτης μπορεί να κινηθεί (CanMove), αν είναι σε κίνηση (IsMoving), και αν είναι στο έδαφος ή στον αέρα. Αν είναι στο έδαφος, χρησιμοποιούμε είτε την ταχύτητα περπατήματος (walkSpeed) είτε την ταχύτητα τρεξίματος (runSpeed). Αν είναι στον αέρα, χρησιμοποιούμε την ταχύτητα πτήσης (airWalkSpeed).



```
public bool _isMoving = false;

4 references
public bool IsMoving
{
    get
    {
        return moveLeft || moveRight;
    }
    private set
    {
        _isMoving = value;
        animator.SetBool(AnimationStrings.isMoving, value);
    }
}

[SerializeField]
public bool _isRunning = false;

3 references
public bool IsRunning
{
    get
    {
        return _isRunning;
    }
    set
    {
        _isRunning = value;
        animator.SetBool(AnimationStrings.isRunning, value);
    }
}

public bool _isFacingRight = true;

4 references
public bool IsFacingRight
{
    get
    {
        return _isFacingRight;
    }
    private set
    {
        if (_isFacingRight != value)
        {
            transform.localScale *= new Vector2(-1, 1);
        }

        _isFacingRight = value;
    }
}
```

Οι μεταβλητές και μέθοδοι αυτές ελέγχουν αν ο παίκτης κινείται και τρέχει:

- IsMoving: Επιστρέφει true αν ο παίκτης κινείται αριστερά ή δεξιά (χρησιμοποιώντας τις μεταβλητές moveLeft και moveRight).
- IsRunning: Χρησιμοποιείται για να καθορίσει αν ο παίκτης τρέχει. Η τιμή της ενημερώνει το animation του παίκτη.

Αν ο παίκτης αλλάξει κατεύθυνση, δηλαδή από αριστερά σε δεξιά ή το αντίστροφο, τότε αναστρέφουμε την κλίμακα στον άξονα X:

- IsFacingRight: Ελέγχει αν ο παίκτης κοιτάζει προς τα δεξιά. Αν η κατεύθυνση αλλάξει, η κλίμακα του παίκτη αντιστρέφεται ώστε να κοιτάζει προς την αντίθετη πλευρά.

Στο **Awake()** αρχικοποιούμε όλα τα components που απαιτούνται για τον έλεγχο του παίκτη:

- Rigidbody2D: Χρησιμοποιείται για τη φυσική κίνηση του παίκτη στο παιχνίδι.
- Animator: Χειρίζεται τα animations, όπως η κίνηση, το άλμα, και ο θάνατος.
- TouchingDirections: Ελέγχει αν ο παίκτης βρίσκεται στο έδαφος, σε τοίχο ή στην οροφή.
- Damageable: Διαχειρίζεται την υγεία του παίκτη.

```
public void FixedUpdate()
{
    if (!damageable.LockVelocity)
    {
        float moveX = 0;
        if (moveLeft)
        {
            moveX = -1;
        }
        else if (moveRight)
        {
            moveX = 1;
        }

        // Έλεγχος αν ο παίκτης ακουμπάει τον τοίχο και δεν είναι στο έδαφος
        if (touchingDirections.IsOnWall && !touchingDirections.IsGrounded)
        {
            // Διατηρείται η ταχύτητα προς τα κάτω λόγω βαρύτητας
            rb.velocity = new Vector2(rb.velocity.x, Mathf.Max(rb.velocity.y, -5f));
        }
        else
        {
            rb.velocity = new Vector2(moveX * CurrentMoveSpeed, rb.velocity.y);
        }

        IsMoving = moveX != 0;
        SetFacingDirection(moveX);
    }

    animator.SetFloat(AnimationStrings.yVelocity, rb.velocity.y);
}
```

Στη συνεχεια στο FixedUpdate(), η φυσική κίνηση του παίκτη ενημερώνεται βάσει των εισόδων του χρήστη:

- Ελέγχουμε αν ο παίκτης κινείται αριστερά ή δεξιά και προσαρμόζουμε την ταχύτητα (velocity) του Rigidbody2D ανάλογα με τη CurrentMoveSpeed.
- Αν ο παίκτης ακουμπάει τοίχο και δεν είναι στο έδαφος, διατηρείται μια σταθερή πτώση για να αποφεύγεται η πολύ γρήγορη πτώση.

```
0 references
public void OnJump(InputAction.CallbackContext context)
{
    if (context.started && touchingDirections.IsGrounded && CanMove && !PauseMenuController.isPaused)
    {
        animator.SetTrigger(AnimationStrings.jumpTrigger);
        rb.velocity = new Vector2(rb.velocity.x, jumpImpulse);
    }
}

0 references
public void OnAttack(InputAction.CallbackContext context)
{
    if (context.started && !PauseMenuController.isPaused)
    {
        moveLeft = false;
        moveRight = false;
        IsMoving = false;
        rb.velocity = new Vector2(0, rb.velocity.y);

        animator.SetTrigger(AnimationStrings.attackTrigger);
    }
}
```

Ο παίκτης μπορεί να πηδήξει και να επιτεθεί με βάση τις εισόδους του χρήστη:

- Στο OnJump(), αν ο παίκτης είναι στο έδαφος και δεν έχει παγώσει το παιχνίδι, καλείται η ενέργεια άλματος, προσδίδοντας μια ώθηση στον άξονα Y.
- Στο OnAttack(), σταματάει η κίνηση και καλείται το trigger επίθεσης.

```
2 references
public void OnPause(InputAction.CallbackContext context)
{
    if (context.started && pauseMenuController != null)
    {
        var playerInput = GetComponent<PlayerInput>();
        playerInput.DeactivateInput();

        pauseMenuController.OnPause(context);
    }
}
```

Ο παίκτης μπορεί να θέσει το παιχνίδι σε παύση (pause) και να το επαναφέρει μέσω του συστήματος εισόδων:

- OnPause(): Όταν ο παίκτης πατήσει το κουμπί για την παύση (που έχει οριστεί μέσω του InputAction), απενεργοποιούνται οι εισόδοι του παίκτη και ενεργοποιείται το pause menu μέσω του pauseMenuController. Αυτό διακόπτει την εξέλιξη του παιχνιδιού μέχρι να βγει από την παύση.

```
1 reference
public IEnumerator HandleDeathTransition()
{
    yield return new WaitForSeconds(deathPanelDelay);

    Time.timeScale = 0f;

    if (backgroundMusic != null)
    {
        backgroundMusic.ignoreListenerPause = true;
    }

    gameOverPanel.SetActive(true);

    float blurDuration = 1.5f;
    float elapsedTime = 0f;

    if (depthOfField != null)
    {
        depthOfField.active = true;
        depthOfField.gaussianStart.Override(5f);
        depthOfField.gaussianEnd.Override(5f);

        while (elapsedTime < blurDuration)
        {
            float t = elapsedTime / blurDuration;
            depthOfField.gaussianStart.Override(Mathf.Lerp(5f, 1f, t));
            depthOfField.gaussianEnd.Override(Mathf.Lerp(30f, 10f, t));

            elapsedTime += Time.unscaledDeltaTime;
            yield return null;
        }

        depthOfField.gaussianStart.Override(1f);
        depthOfField.gaussianEnd.Override(10f);
    }
}
```

Η HandleDeathTransition() είναι υπεύθυνη για το transition που ακολουθεί τον θάνατο του παίκτη:

- HandleDeathTransition(): Αυτή η συνάρτηση χρησιμοποιεί ένα IEnumerator για να καθυστερήσει την εμφάνιση του gameOverPanel (του πάνελ της οθόνης θανάτου) για ένα καθορισμένο διάστημα (deathPanelDelay). Μετά από την καθυστέρηση, το παιχνίδι παγώνει με Time.timeScale = 0f και ενεργοποιείται το gameOverPanel. Επιπλέον, αν υπάρχει ενεργό το εφέ βάθους πεδίου (Depth of Field), γίνεται μια σταδιακή εστίαση και θόλωση (blur effect) στην οθόνη κατά τη διάρκεια του transition.

```
public void OnInteract(InputAction.CallbackContext context)
{
    if (context.started)
    {
        if (isPlayerInRangeOfDoor && doorAnimator != null)
        {
            if (HasKey)
            {
                if (!isDoorOpen)
                {
                    doorAnimator.SetTrigger("OpenDoor");
                    isDoorOpen = true;

                    // Παίξε τον ήχο ανοίγματος της πόρτας
                    if (doorAnimator.TryGetComponent<AudioSource>(out var audioSource))
                    {
                        audioSource.Play();
                    }
                    else
                    {
                        Debug.LogError("AudioSource is missing on the door object.");
                    }
                }
                else
                {
                    SceneManager.LoadScene("LoadingScreenScene2");
                }
            }
            else
            {
                ShowHintMessage();
            }
        }
    }
}
```

Ο παίκτης μπορεί να αλληλεπιδράσει με πόρτες στο παιχνίδι. Η OnInteract() ελέγχει αυτήν τη διαδραστικότητα:

- OnInteract(): Αυτή η συνάρτηση ελέγχει αν ο παίκτης βρίσκεται κοντά σε μια πόρτα (isPlayerInRangeOfDoor). Αν έχει το κλειδί (HasKey) και η πόρτα δεν είναι ήδη ανοιχτή, η πόρτα ανοίγει και αναπαράγεται ένας ήχος. Αν η πόρτα είναι ανοιχτή, τότε μεταφερόμαστε στη σκηνή "LoadingScreenScene2". Αν δεν υπάρχει το κλειδί, εμφανίζεται ένα μήνυμα υπόδειξης στον παίκτη για το πού να το βρει.

```
public void ShowHintMessage()
{
    // Δημιουργία και τοποθέτηση του μηνύματος προγραμματιστικά
    GameObject hintObject = new("HintMessage");
    hintObject.transform.position = doorAnimator.transform.position + new Vector3(0, 2.5f, 0); // Τοποθέτηση πάνω από την πόρτα

    // Προσθήκη TextMeshPro συστατικού
    TextMeshPro textMesh = hintObject.AddComponent<TextMeshPro>();
    textMesh.text = "HINT: \nYou need to find the key in the Underworld of Hades,\ndefeating the monsters that stand in your way!";
    textMesh.fontSize = 4; // Ρύθμισε το μέγεθος όπως χρειάζεται
    textMesh.alignment = TextAlignmentOptions.Center;

    // Χρώμα κειμένου
    textMesh.color = Color.white;

    // Προσθήκη outline στο κείμενο για καλύτερη ορατότητα
    var outline = textMesh.gameObject.AddComponent<Outline>();
    outline.effectColor = Color.black; // Μαύρο περίγραμμα για καλύτερη αντίθεση
    outline.effectDistance = new Vector2(1f, -1f); // Προσαρμογή της απόστασης του περιγράμματος

    // Καταστροφή του αντικειμένου μετά από 3 δευτερόλεπτα
    Destroy(hintObject, 4f);
}
```

Όταν ο παίκτης δεν έχει το κλειδί για την πόρτα, εμφανίζεται ένα μήνυμα υπόδειξης στο παιχνίδι:

- ShowHintMessage(): Δημιουργεί ένα μήνυμα υπόδειξης προγραμματιστικά χρησιμοποιώντας TextMeshPro. Το μήνυμα τοποθετείται πάνω από την πόρτα και παραμένει ορατό για 4 δευτερόλεπτα πριν καταστραφεί.

```
public void PlayerDeath()
{
    if (!isPlayerDead)
    {
        isPlayerDead = true;

        lastSceneName = SceneManager.GetActiveScene().name;

        if (backgroundMusic != null)
        {
            backgroundMusic.pitch = 0.5f;
        }

        if (deathMusic != null)
        {
            backgroundMusic.Stop();
            deathMusic.Play();
        }

        StartCoroutine(HandleDeathTransition());
    }
}
```

Όταν η υγεία του παίκτη φτάσει στο μηδέν, καλείται η διαδικασία θανάτου. Η μέθοδος PlayerDeath() είναι υπεύθυνη για τη διαχείριση του θανάτου του παίκτη:

- PlayerDeath(): Ελέγχει αν ο παίκτης είναι ήδη νεκρός. Αν δεν είναι, αλλάζει την τιμή του isPlayerDead σε true και αποθηκεύει το όνομα της τελευταίας σκηνής στην οποία βρισκόταν ο παίκτης, ώστε να επανεκκινηθεί από αυτή. Στη συνέχεια,

η μουσική φόντου επιβραδύνεται και σταματάει, παίζοντας τη μουσική του θανάτου (αν υπάρχει). Τέλος, καλείται η HandleDeathTransition(), η οποία διαχειρίζεται το transition στην οθόνη θανάτου.

```
public void NewGame()
{
    Time.timeScale = 1f;

    if (!string.IsNullOrEmpty(lastSceneName))
    {
        SceneManager.LoadScene(lastSceneName);
    }
    else
    {
        SceneManager.LoadScene("DefaultScene");
    }

    if (depthOfField != null)
    {
        depthOfField.active = false;
    }
}

0 references
public void ReturnToMainMenu()
{
    Time.timeScale = 1f;
    SceneManager.LoadScene("MainMenu");

    if (depthOfField != null)
    {
        depthOfField.active = false;
    }
}

0 references
public void QuitGame()
{
    Application.Quit();
}
```

Η NewGame() μέθοδος χρησιμοποιείται για να ξεκινήσει ξανά το παιχνίδι. Τα βήματα που ακολουθούνται είναι τα εξής:

- Time.timeScale = 1f;: Η ταχύτητα του χρόνου επανέρχεται στο κανονικό (1f), καθώς πιθανόν να είχε γίνει παύση (Time.timeScale = 0) προηγουμένως.
- if (!string.IsNullOrEmpty(lastSceneName)): Ελέγχει αν υπάρχει αποθηκευμένο το όνομα της τελευταίας σκηνής στην οποία βρισκόταν ο παίκτης (μέσω της lastSceneName μεταβλητής). Αν υπάρχει, τότε φορτώνει αυτή τη σκηνή, επιτρέποντας στον παίκτη να ξεκινήσει ξανά από εκεί που σταμάτησε.
- else: Αν δεν υπάρχει σκηνή αποθηκευμένη στη lastSceneName, φορτώνεται η προεπιλεγμένη σκηνή με το όνομα "DefaultScene".

- depthOfField.active = false;: Απενεργοποιεί το εφέ βάθους πεδίου (Depth of Field) αν αυτό είναι ενεργοποιημένο. Το Depth of Field χρησιμοποιείται για να θολώνει το υπόβαθρο, πιθανόν κατά τη διάρκεια του θανάτου του παίκτη. Με την επανεκκίνηση του παιχνιδιού, αυτό το εφέ απενεργοποιείται για να έχουμε καθαρή εικόνα.

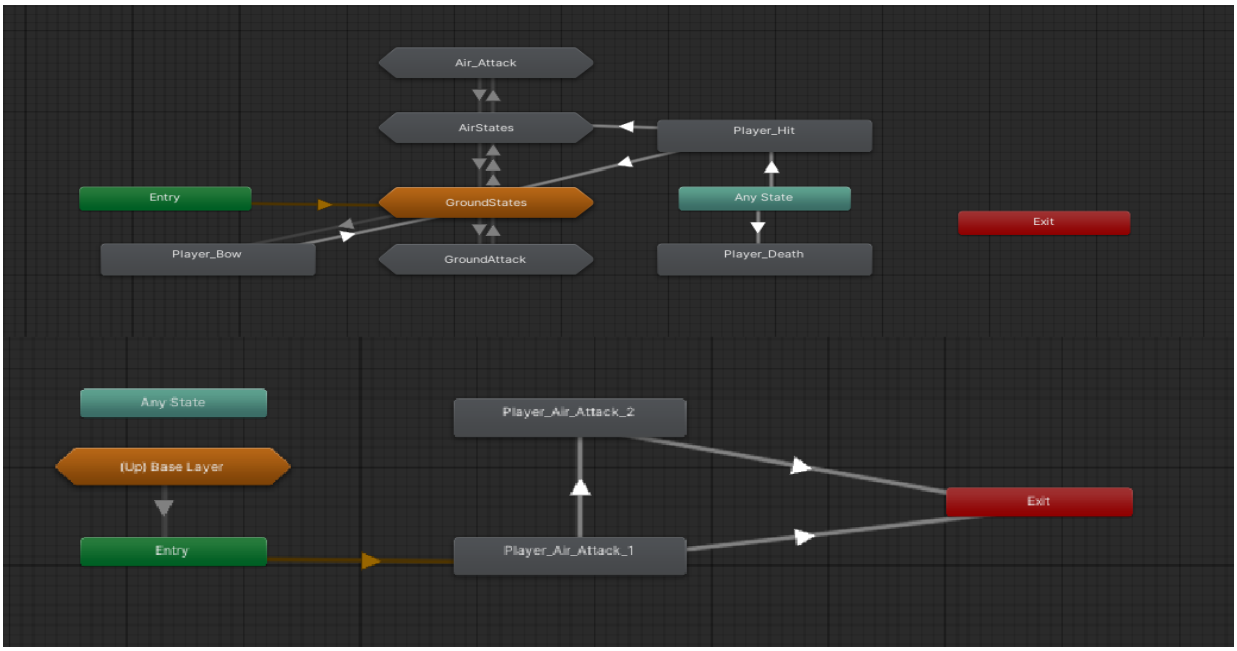
Η ReturnToMainMenu() χρησιμοποιείται για την επιστροφή στο κύριο μενού του παιχνιδιού. Ακολουθεί τα εξής βήματα:

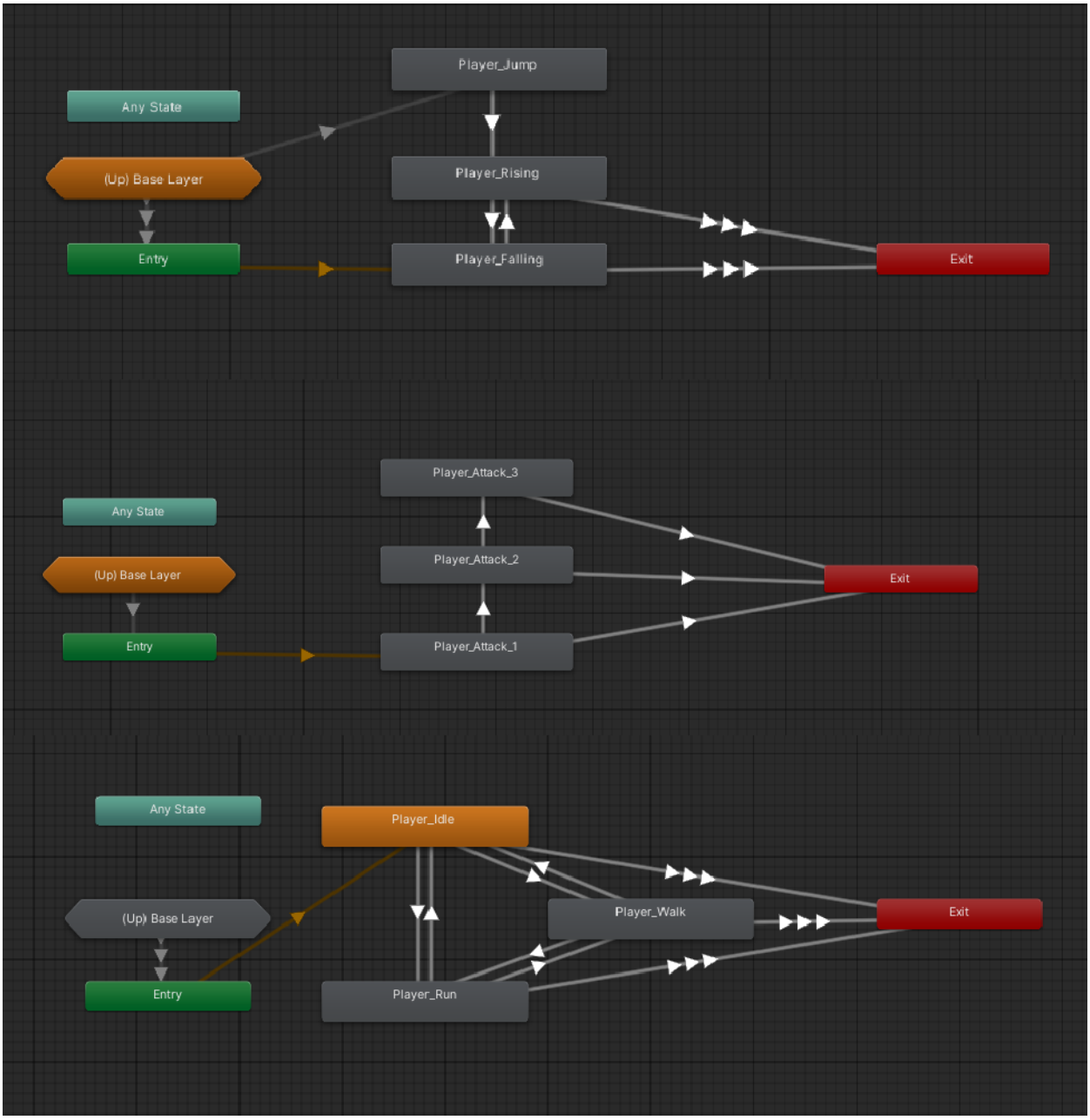
- Time.timeScale = 1f;: Όπως και στη μέθοδο NewGame(), η ταχύτητα του χρόνου επανέρχεται στο κανονικό επίπεδο.
- SceneManager.LoadScene("MainMenu");: Φορτώνει τη σκηνή του κύριου μενού του παιχνιδιού (MainMenu). Ο παίκτης επιστρέφει στο αρχικό σημείο από όπου ξεκίνησε το παιχνίδι.
- depthOfField.active = false;: Όπως και στην προηγούμενη μέθοδο, αν το εφέ Depth of Field είναι ενεργό, αυτό απενεργοποιείται για να εξασφαλιστεί ότι το μενού θα είναι καθαρό και ευκρινές.

Η QuitGame() είναι η πιο απλή από τις τρεις μεθόδους, και έχει ως σκοπό να κλείσει το παιχνίδι.

- Application.Quit();: Χρησιμοποιείται για να κλείσει την εφαρμογή. Όταν ο παίκτης επιλέξει να βγει από το παιχνίδι, το Application.Quit() τερματίζει την εκτέλεση του προγράμματος. Σε περίπτωση που το παιχνίδι βρίσκεται σε περιβάλλον ανάπτυξης (όπως το Unity Editor), αυτό δεν θα κλείσει το πρόγραμμα, αλλά σε κανονικό build θα τερματίσει την εφαρμογή.

Εικόνες από το Animator του Player :




Player_Jump
Any State
(Up) Base Layer
Player_Rising
Entry
Player_Falling
Exit
Player_Attack_3
Any State
Player_Attack_2
(Up) Base Layer
Exit
Entry
Player_Attack_1
Any State
Player_Idle
(Up) Base Layer
Player_Walk
Exit
Entry
Player_Run

Goblin/Knight/Skeleotn.cs

```
public float walkAcceleration = 50f;
public float maxSpeed = 3f;
public float walkStopRate = 0.05f;
public DetectionZone attackZone;
public DetectionZone cliffDetectionZone;
```

Στην αρχή του script έχουμε το attribute [RequireComponent] το οποίο διασφαλίζει ότι το αντικείμενο Goblin θα περιέχει τα απαραίτητα components για τη λειτουργία του, συγκεκριμένα τα Rigidbody2D, TouchingDirections, και Damageable.

- Rigidbody2D: Χρησιμοποιείται για τη φυσική κίνηση του εχθρού.
- TouchingDirections: Ελέγχει αν ο εχθρός ακουμπάει το έδαφος, τοίχους, ή οροφές.
- Damageable: Διαχειρίζεται τη ζωή και την υγεία του εχθρού.

Οι μεταβλητές walkAcceleration, maxSpeed και walkStopRate χρησιμοποιούνται για να καθορίσουν την ταχύτητα κίνησης και την επιτάχυνση του Goblin.

Οι DetectionZones είναι δύο σημαντικές ζώνες ανίχνευσης που καθορίζουν τη συμπεριφορά του εχθρού:

- attackZone: Ζώνη ανίχνευσης που καθορίζει αν ο παίκτης είναι εντός εμβέλειας για να δεχθεί επίθεση.
- cliffDetectionZone: Ζώνη ανίχνευσης γκρεμών ή άκρων, ώστε το Goblin να αλλάξει κατεύθυνση όταν πλησιάζει έναν γκρεμό.

Διαχείριση κατεύθυνσης κίνησης (WalkableDirection)

```
public enum WalkableDirection { Right, Left }

private WalkableDirection _walkDirection;
private Vector2 walkDirectionVector = Vector2.right;
```

- Το Goblin μπορεί να κινηθεί είτε προς τα δεξιά είτε προς τα αριστερά. Αυτό καθορίζεται από την enum WalkableDirection, η οποία μπορεί να πάρει τις τιμές Right ή Left.
- Η μεταβλητή _walkDirection καθορίζει την τρέχουσα κατεύθυνση του Goblin και η walkDirectionVector είναι ένα Vector2 που υποδεικνύει την κατεύθυνση κίνησης στον άξονα X.

Μεταβολή κατεύθυνσης (FlipDirection)

```
public WalkableDirection WalkDirection
{
    get { return _walkDirection; }
    set
    {
        if (_walkDirection != value)
        {
            gameObject.transform.localScale = new Vector2(gameObject.transform.localScale.x * -1, gameObject.transform.localScale.y);

            if (value == WalkableDirection.Right)
            {
                walkDirectionVector = Vector2.right;
            }
            else if (value == WalkableDirection.Left)
            {
                walkDirectionVector = Vector2.left;
            }
        }
        _walkDirection = value;
    }
}
```

- Αν το Goblin αλλάξει κατεύθυνση (π.χ., από δεξιά προς αριστερά), η localScale του αντικειμένου αλλάζει. Αυτό σημαίνει ότι το sprite του Goblin αναστρέφεται, ώστε να "κοιτάζει" προς τη νέα κατεύθυνση.
- Η walkDirectionVector ενημερώνεται αναλόγως, καθορίζοντας αν το Goblin θα κινηθεί προς τα δεξιά (Vector2.right) ή αριστερά (Vector2.left).

```
public bool _hasTarget = false;

1 reference
public bool HasTarget
{
    get
    {
        return _hasTarget;
    }
    private set
    {
        _hasTarget = value;
        animator.SetBool(AnimationStrings.hasTarget, value);
    }
}

1 reference
public bool CanMove
{
    get
    {
        return animator.GetBool(AnimationStrings.canMove);
    }
}

2 references
public float AttackCooldown
{
    get
    {
        return animator.GetFloat(AnimationStrings.attackCooldown);
    }
    private set
    {
        animator.SetFloat(AnimationStrings.attackCooldown, Mathf.Max(value, 0));
    }
}
```

Εύρεση στόχου (HasTarget)

- Η ιδιότητα HasTarget χρησιμοποιείται για να ελέγξει αν το Goblin έχει στόχο (παίκτη) εντός της ζώνης επίθεσης.
- Αν βρεθεί στόχος, τότε ενημερώνεται το Animator για να ξεκινήσει την επίθεση μέσω του AnimationStrings.hasTarget.

CanMove: Έλεγχος κίνησης

- Το Goblin μπορεί να κινηθεί μόνο όταν η τιμή CanMove είναι true, η οποία ρυθμίζεται από το Animator.

AttackCooldown: Χρονόμετρο επιθέσεων

- Αυτή η ιδιότητα χρησιμοποιείται για να ελέγχει το χρονικό διάστημα που μεσολαβεί μεταξύ των επιθέσεων του Goblin. Κατά τη διάρκεια του Cooldown, το Goblin δεν μπορεί να επιτεθεί ξανά μέχρι να λήξει αυτό το χρονικό διάστημα.

```
Unity Message | 0 references
private void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    touchingDirections = GetComponent<TouchingDirections>();
    animator = GetComponent<Animator>();
    damageable = GetComponent<Damageable>();
}

Unity Message | 0 references
void Update()
{
    // Checking if there's any collider in the attackZone
    HasTarget = attackZone.detectedColliders.Count > 0 && attackZone.detectedColliders[0].CompareTag("Player");

    if (AttackCooldown > 0)
    {
        AttackCooldown -= Time.deltaTime;
    }
}

Unity Message | 0 references
private void FixedUpdate()
{
    if (touchingDirections.IsGrounded && touchingDirections.IsOnWall)
    {
        FlipDirection();
    }

    if (!damageable.LockVelocity)
    {
        if (CanMove && touchingDirections.IsGrounded)
        {
            rb.velocity = new Vector2(
                Mathf.Clamp(rb.velocity.x + (walkAcceleration * walkDirectionVector.x * Time.fixedDeltaTime), -maxSpeed, maxSpeed),
                rb.velocity.y);
        }
        else
        {
            rb.velocity = new Vector2(Mathf.Lerp(rb.velocity.x, 0, walkStopRate), rb.velocity.y);
        }
    }
}
```

Awake(): Αρχικοποίηση components

- Στο Awake() αρχικοποιούνται όλα τα απαραίτητα components:
    - Rigidbody2D: Χρησιμοποιείται για τη φυσική κίνηση του Goblin.

- TouchingDirections: Ελέγχει αν το Goblin βρίσκεται στο έδαφος, σε τοίχο ή σε γκρεμό.
- Animator: Χρησιμοποιείται για να αλλάζει τα animations του Goblin.
- Damageable: Διαχειρίζεται την υγεία του Goblin.

Update() και FixedUpdate()

- Στο Update(), το Goblin ελέγχει αν υπάρχει κάποιος στόχος εντός της ζώνης επίθεσης (attackZone).
- Αν βρεθεί στόχος με το tag "Player", το HasTarget γίνεται true και το Goblin προετοιμάζεται για επίθεση.
- Παράλληλα, το χρονόμετρο της επίθεσης (AttackCooldown) μειώνεται όσο περνάει ο χρόνος.
- Στο FixedUpdate(), το Goblin ελέγχει αν ακουμπάει τοίχο και, αν ναι, αλλάζει κατεύθυνση χρησιμοποιώντας τη FlipDirection().
- Όσο το Goblin μπορεί να κινηθεί και είναι στο έδαφος, η ταχύτητά του αυξάνεται σταδιακά χρησιμοποιώντας την walkAcceleration μέχρι να φτάσει τη μέγιστη ταχύτητα (maxSpeed).
- Όταν το Goblin σταματά να κινείται, η ταχύτητα του μειώνεται σταδιακά μέσω της Lerp μέχρι να σταματήσει εντελώς.

```
2 references
private void FlipDirection()
{
    if (WalkDirection == WalkableDirection.Right)
    {
        WalkDirection = WalkableDirection.Left;
    }
    else if (WalkDirection == WalkableDirection.Left)
    {
        WalkDirection = WalkableDirection.Right;
    }
    else
    {
    }
}

0 references
public void OnHit(int damage, Vector2 knockback)
{
    rb.velocity = new Vector2(knockback.x, rb.velocity.y + knockback.y);
}

0 references
public void OnCliffDetected()
{
    if (touchingDirections.IsGrounded)
    {
        FlipDirection();
    }
}
```

FlipDirection

- Η συνάρτηση FlipDirection() αντιστρέφει την κατεύθυνση του Goblin όταν βρει εμπόδιο, όπως τοίχο ή γκρεμό.

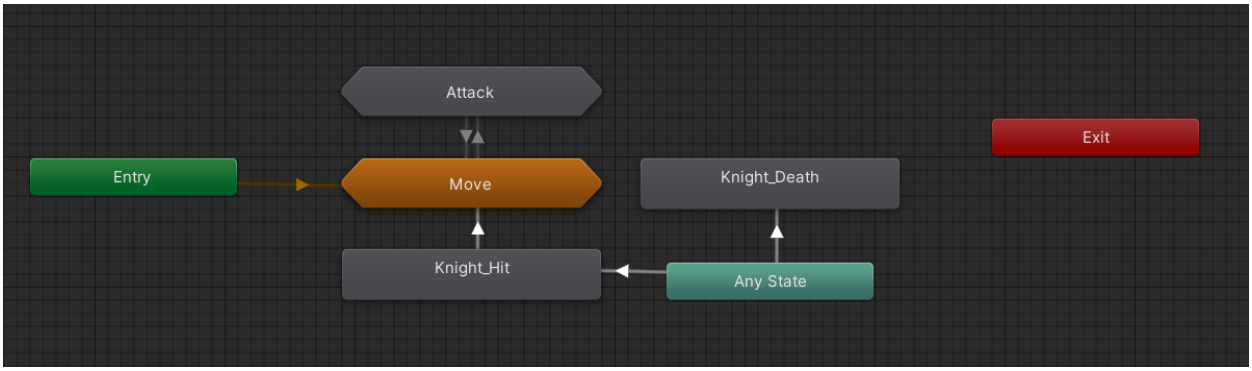
OnHit

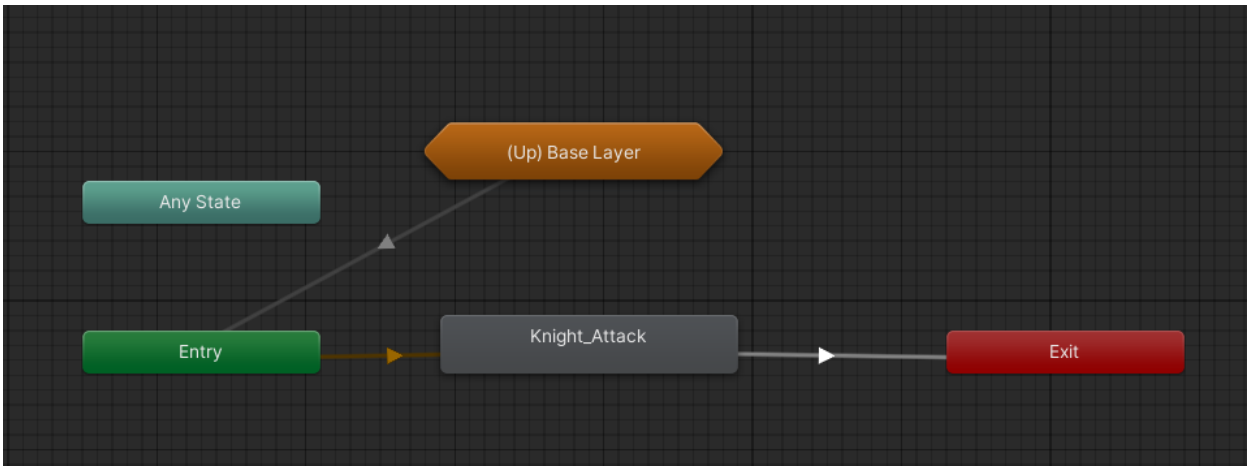
- Όταν το Goblin δεχθεί χτύπημα, η συνάρτηση OnHit εφαρμόζει knockback στην κίνηση του, μεταβάλλοντας την ταχύτητά του ανάλογα με τις συντεταγμένες του χτυπήματος.

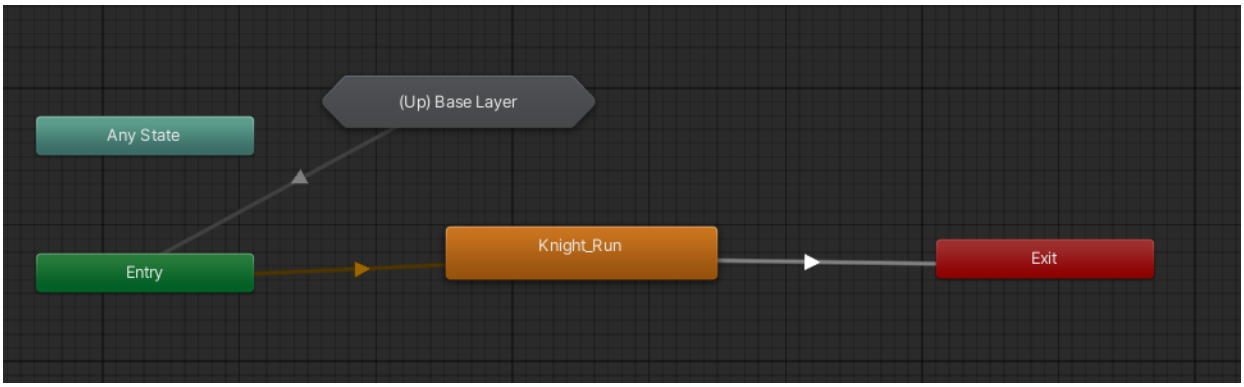
OnCliffDetected

- Αυτή η συνάρτηση καλείται όταν το Goblin εντοπίσει γκρεμό (μέσω της cliffDetectionZone) και αλλάζει κατεύθυνση για να μην πέσει.

Εικόνες από το Animator του Knight(ισχύουν τα ίδια και για τα Goblin /Skeleton) :






(Up) Base Layer
Any State
Entry
Knight_Run
Exit

CyclopsController.cs

Το CyclopsController είναι το script που ελέγχει το μεγάλο boss του παιχνιδιού, τον Κύκλωπα.

```
public float walkAcceleration = 50f;
public float maxSpeed = 3f;
public float walkStopRate = 0.05f;
public DetectionZone attackZone;
public DetectionZone cliffDetectionZone;
public Camera mainCamera; // Κύρια κάμερα
public float zoomDuration = 2f; // Διάρκεια του zoom-in
public float slowMotionScale = 0.4f; // Η τιμή για το slow motion (π.χ. 30% της κανονικής ταχύτητας)
```

- Όπως και στα άλλα scripts, χρησιμοποιούμε το attribute [RequireComponent] για να βεβαιωθούμε ότι ο Κύκλωπας θα έχει τα απαραίτητα components:
    - Rigidbody2D: Χρησιμοποιείται για την φυσική κίνηση του Κύκλωπα.
    - TouchingDirectionsCyclops: Ανιχνεύει αν ο Κύκλωπας ακουμπά το έδαφος ή έναν τοίχο.
    - Damageable: Διαχειρίζεται την υγεία του Κύκλωπα και τις αλληλεπιδράσεις όταν δέχεται χτυπήματα.
- Η ταχύτητα και η κίνηση του Κύκλωπα ελέγχονται μέσω των μεταβλητών walkAcceleration, maxSpeed, και walkStopRate.
    - walkAcceleration: Η επιτάχυνση του Κύκλωπα καθώς αρχίζει να περπατά.
    - maxSpeed: Η μέγιστη ταχύτητα που μπορεί να φτάσει ο Κύκλωπας.
    - walkStopRate: Ο ρυθμός με τον οποίο ο Κύκλωπας σταματά όταν δεν κινείται πλέον.
- Οι DetectionZones περιλαμβάνουν δύο ζώνες:
    - attackZone: Η ζώνη μέσα στην οποία ο Κύκλωπας μπορεί να ανιχνεύσει τον παίκτη και να επιτεθεί.
    - cliffDetectionZone: Η ζώνη που ανιχνεύει αν ο Κύκλωπας πλησιάζει έναν γκρεμό για να αλλάξει κατεύθυνση.
- Η mainCamera χρησιμοποιείται για να εφαρμοστεί ένα οπτικό εφέ zoom κατά τη διάρκεια της ήττας του Κύκλωπα, σε συνδυασμό με ένα slow-motion effect.

```
10 references
public enum WalkableDirection { Right, Left }

private WalkableDirection _walkDirection;
private Vector2 walkDirectionVector = Vector2.right;
```

Κατεύθυνση κίνησης και αρχικοποίηση μεταβλητών

- Η κατεύθυνση κίνησης του Κύκλωπα ορίζεται μέσω του enum WalkableDirection, το οποίο μπορεί να είναι είτε Right είτε Left.
- Η μεταβλητή walkDirectionVector αντιπροσωπεύει την κατεύθυνση κίνησης στον άξονα X. Αν το WalkDirection είναι Right, το walkDirectionVector είναι Vector2.right. Αν είναι Left, τότε Vector2.left.

```
6 references
public WalkableDirection WalkDirection
{
    get { return _walkDirection; }
    set
    {
        if (_walkDirection != value)
        {
            Debug.Log("Changing walk direction.");
            transform.localScale = new Vector2(transform.localScale.x * -1, transform.localScale.y);

            if (value == WalkableDirection.Right)
            {
                walkDirectionVector = Vector2.right;
                Debug.Log("Cyclops walking right.");
            }
            else if (value == WalkableDirection.Left)
            {
                walkDirectionVector = Vector2.left;
                Debug.Log("Cyclops walking left.");
            }
        }
        _walkDirection = value;
    }
}
```

Αλλαγή κατεύθυνσης (WalkDirection)

- Αν αλλάξει η κατεύθυνση κίνησης του Κύκλωπα (π.χ. από Right σε Left), η κλίμακα στον άξονα X αναστρέφεται, με αποτέλεσμα να γυρίζει το sprite του ώστε να κοιτάζει την αντίθετη κατεύθυνση.
- Χρησιμοποιούμε το Debug.Log() για να παρακολουθήσουμε στο Console την αλλαγή κατεύθυνσης του Κύκλωπα.

```
public bool _hasTarget = false;
1 reference
public bool HasTarget
{
    get { return _hasTarget; }
    private set
    {
        _hasTarget = value;
        animator.SetBool(AnimationStrings.hasTarget, value);
        Debug.Log("Cyclops has target: " + value);
    }
}

1 reference
public bool CanMove
{
    get
    {
        bool canMove = animator.GetBool(AnimationStrings.canMove);
        Debug.Log("Cyclops can move: " + canMove);
        return canMove;
    }
}

3 references
public float AttackCooldown
{
    get { return animator.GetFloat(AnimationStrings.attackCooldown); }
    private set
    {
        animator.SetFloat(AnimationStrings.attackCooldown, Mathf.Max(value, 0));
        Debug.Log("Attack cooldown set to: " + value);
    }
}
```

Έλεγχος στόχου (HasTarget)

- Το HasTarget χρησιμοποιείται για να ελέγχει αν υπάρχει στόχος (ο παίκτης) εντός της attackZone. Αν εντοπιστεί στόχος, τότε η τιμή του HasTarget γίνεται true και το animation αλλάζει σε επίθεση.
- Η μεταβλητή _hasTarget ενημερώνεται μέσω του Animator, ο οποίος θέτει το animation string hasTarget ανάλογα με το αν υπάρχει στόχος ή όχι.

CanMove: Έλεγχος αν μπορεί να κινηθεί

- Το CanMove είναι μια boolean ιδιότητα που ελέγχει αν ο Κύκλωπας μπορεί να κινηθεί, με βάση την κατάσταση του Animator.
- Αν ο Κύκλωπας δεν μπορεί να κινηθεί, η ταχύτητά του μηδενίζεται.

Cooldown επιθέσεων (AttackCooldown)

- Το AttackCooldown ελέγχει τον χρόνο που πρέπει να περάσει μέχρι να μπορέσει ο Κύκλωπας να επιτεθεί ξανά.

- Κάθε φορά που πραγματοποιείται επίθεση, το Cooldown μειώνεται, διασφαλίζοντας ότι ο Κύκλωπας δεν θα επιτεθεί υπερβολικά γρήγορα.

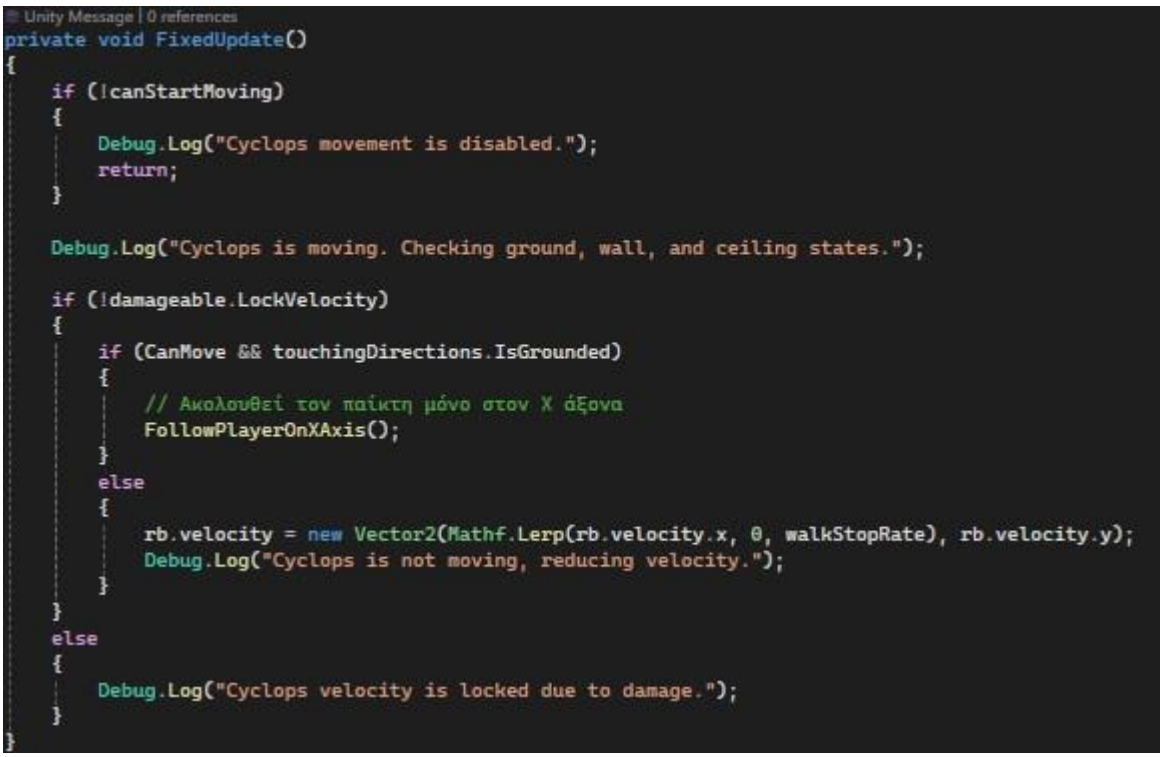
```
Unity Message | 0 references
private void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    touchingDirections = GetComponent<TouchingDirectionsCyclops>();
    animator = GetComponent<Animator>();
    damageable = GetComponent<Damageable>();
    Debug.Log("Cyclops initialized.");
}

Unity Message | 0 references
private void Update()
{
    if (!canStartMoving)
    {
        Debug.Log("Cyclops cannot start moving yet.");
        return;
    }

    HasTarget = attackZone.detectedColliders.Count > 0;
    Debug.Log("Cyclops detected " + attackZone.detectedColliders.Count + " colliders in attack zone.");

    if (AttackCooldown > 0)
    {
        AttackCooldown -= Time.deltaTime;
        Debug.Log("Cyclops attack cooldown: " + AttackCooldown);
    }
}
```

Στη μέθοδο Awake(), αρχικοποιούμε τα απαραίτητα components του Κύκλωπα:

- Rigidbody2D: Για την κίνηση με φυσική.
- TouchingDirectionsCyclops: Για την ανίχνευση επαφών με το έδαφος ή άλλες επιφάνειες.
- Animator: Για τα animations.
- Damageable: Για τη διαχείριση της υγείας του και τις αλληλεπιδράσεις με χτυπήματα.

Update(): Ενημέρωση κατάστασης και στόχου

- Στο Update(), ο Κύκλωπας ελέγχει αν μπορεί να ξεκινήσει την κίνηση και αν υπάρχουν στόχοι εντός της attackZone.
- Επίσης, το AttackCooldown μειώνεται σε κάθε frame για να ελέγχει πότε μπορεί να γίνει η επόμενη επίθεση.



```
Unity Message | 0 references
private void FixedUpdate()
{
    if (!canStartMoving)
    {
        Debug.Log("Cyclops movement is disabled.");
        return;
    }

    Debug.Log("Cyclops is moving. Checking ground, wall, and ceiling states.");

    if (!damageable.LockVelocity)
    {
        if (CanMove && touchingDirections.IsGrounded)
        {
            // Ακολουθεί τον παίκτη μόνο στον X άξονα
            FollowPlayerOnXAxis();
        }
        else
        {
            rb.velocity = new Vector2(Mathf.Lerp(rb.velocity.x, 0, walkStopRate), rb.velocity.y);
            Debug.Log("Cyclops is not moving, reducing velocity.");
        }
    }
    else
    {
        Debug.Log("Cyclops velocity is locked due to damage.");
    }
}
```

FixedUpdate(): Κίνηση και αλληλεπιδράσεις

- Η FixedUpdate() διαχειρίζεται την κίνηση του Κύκλωπα με βάση τη φυσική και τον αν μπορεί να κινηθεί.
- Εφαρμόζει κίνηση προς τον παίκτη στον X άξονα μέσω της FollowPlayerOnXAxis().

FollowPlayerOnXAxis(): Ακολουθεί τον παίκτη

- Ο Κύκλωπας ακολουθεί τον παίκτη μόνο στον άξονα X, ανάλογα με τη θέση του παίκτη σε σχέση με αυτόν.

```
0 references
public void OnDeath()
{
    // Ξεκινάμε την ομαλή μετάβαση με slow motion και zoom
    StartCoroutine(SmoothTransition());
}

1 reference
private IEnumerator SmoothTransition()
{
    // Ξεκινάμε το zoom-in
    float startZoom = mainCamera.orthographicSize; // Αρχικό μέγεθος της κάμερας
    float targetZoom = startZoom * 0.8f; // Στόχος zoom-in (π.χ. 20% πιο κοντά)

    float elapsedTime = 0f;

    // Εφαρμόζουμε το slow motion
    Time.timeScale = slowMotionScale;

    while (elapsedTime < zoomDuration)
    {
        // Ζουμάρουμε την κάμερα
        elapsedTime += Time.unscaledDeltaTime; // Χρησιμοποιούμε unscaled για να μη μας επηρεάζει το slow motion
        mainCamera.orthographicSize = Mathf.Lerp(startZoom, targetZoom, elapsedTime / zoomDuration);

        yield return null;
    }

    // Περιμένουμε λίγο αφού ολοκληρωθεί το zoom (3.5 δευτερόλεπτα σε πραγματικό χρόνο)
    yield return new WaitForSecondsRealtime(3.5f);

    // Επαναφορά του χρόνου στο κανονικό
    Time.timeScale = 1f;

    // Μετάβαση στην επόμενη σκηνή
    SceneManager.LoadScene("ReturnToIthaca");
}
```

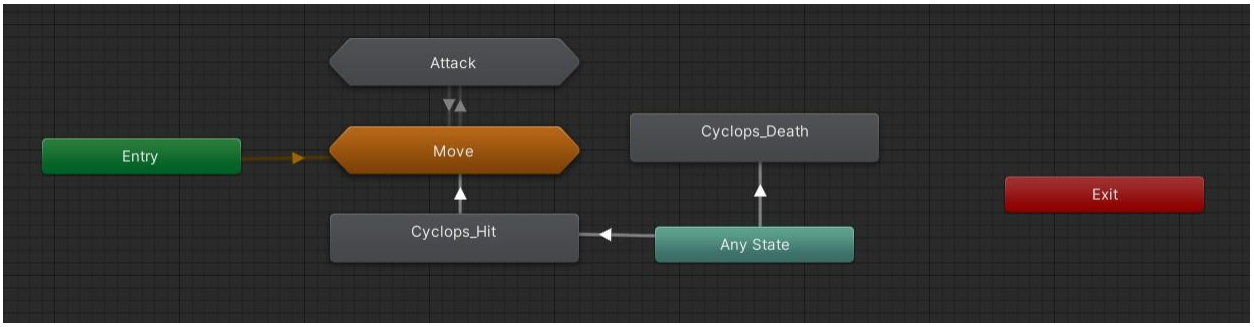
OnDeath(): Διαχείριση θανάτου και εφέ

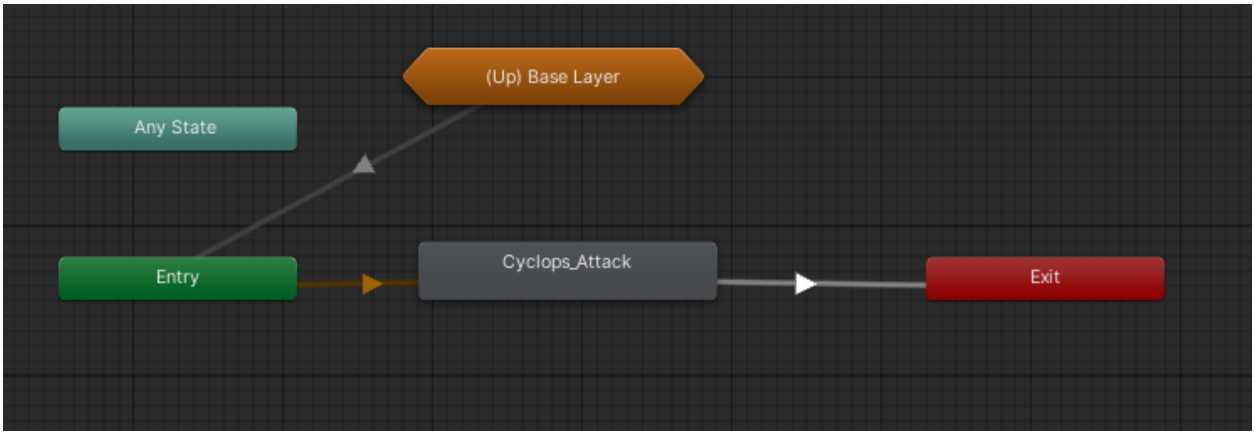
- Η OnDeath() καλείται όταν ο Κύκλωπας πεθαίνει, ενεργοποιώντας μια σταδιακή μετάβαση (zoom-in και slow-motion effect).

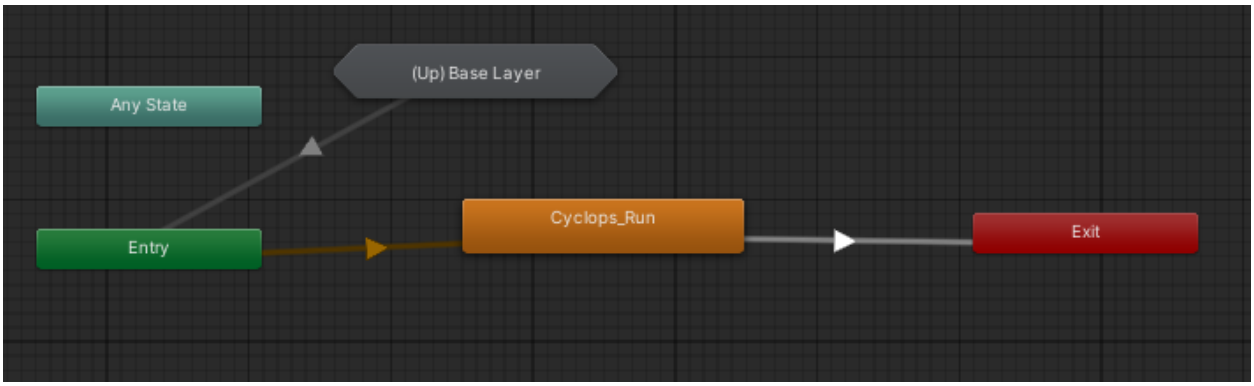
SmoothTransition(): Zoom-in και slow-motion effect

- Η SmoothTransition() εφαρμόζει ένα zoom-in στην κάμερα και ένα slow-motion εφέ κατά τη διάρκεια του θανάτου του Κύκλωπα. Μετά από 3.5 δευτερόλεπτα, το παιχνίδι επιστρέφει στην κανονική ταχύτητα και γίνεται μετάβαση στην επόμενη σκηνή.

Εικόνες από το Animator του Knight(ισχύουν τα ίδια και για τα Goblin /Skeleton) :

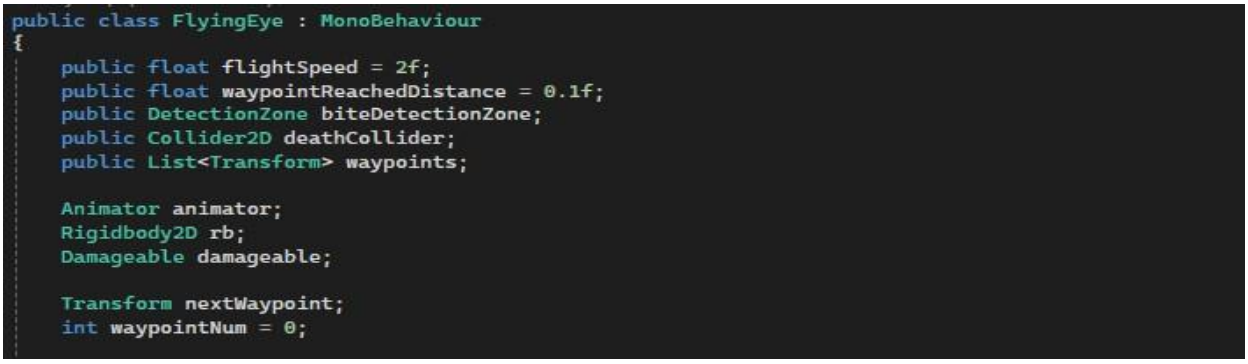






FlyingEye.cs

Το πρώτο κομμάτι του κώδικα περιέχει τις δηλώσεις των μεταβλητών που θα χρησιμοποιηθούν για τον έλεγχο της κίνησης, της συμπεριφοράς του FlyingEye, και της αλληλεπίδρασης με τον παίκτη:



```
public class FlyingEye : MonoBehaviour
{
    public float flightSpeed = 2f;
    public float waypointReachedDistance = 0.1f;
    public DetectionZone biteDetectionZone;
    public Collider2D deathCollider;
    public List<Transform> waypoints;

    Animator animator;
    Rigidbody2D rb;
    Damageable damageable;

    Transform nextWaypoint;
    int waypointNum = 0;
```

- flightSpeed: Ορίζει την ταχύτητα πτήσης του Flying Eye.
- waypointReachedDistance: Η απόσταση στην οποία θεωρούμε ότι το Flying Eye έχει φτάσει σε ένα waypoint (σημείο στόχου).
- biteDetectionZone: Χρησιμοποιείται για να ανιχνεύσει αν ο παίκτης βρίσκεται σε εμβέλεια για επίθεση.
- deathCollider: Ο collider που ενεργοποιείται όταν το Flying Eye πεθάνει, επιτρέποντας πιθανές επιπτώσεις με το περιβάλλον.
- waypoints: Μια λίστα από Transform αντικείμενα που αντιπροσωπεύουν τα σημεία στα οποία θα κινηθεί το Flying Eye.

Ορίζουμε κάποια βασικά components που θα χρησιμοποιηθούν για να χειριστούμε την κίνηση, τα animations, και την αλληλεπίδραση με την υγεία:

- Animator: Χειρίζεται τα animations του Flying Eye, όπως την κίνηση και τον θάνατο.
- Rigidbody2D: Χρησιμοποιείται για να κινηθεί το Flying Eye στο 2D περιβάλλον του παιχνιδιού.
- Damageable: Το component που διαχειρίζεται τη ζωή του Flying Eye, δηλαδή το πόση ζημιά μπορεί να δεχτεί πριν πεθάνει.

```
public bool HasTarget
{
    get { return _hasTarget; }
    private set
    {
        _hasTarget = value;
        animator.SetBool(AnimationStrings.hasTarget, value);
    }
}

1 reference
public bool CanMove
{
    get
    {
        return animator.GetBool(AnimationStrings.canMove);
    }
}
```

Εδώ βλέπουμε την κατάσταση του Flying Eye σχετικά με το αν έχει στόχο και αν μπορεί να κινηθεί:

- HasTarget: Αν το Flying Eye εντοπίσει τον παίκτη στο detection zone (bite zone), η μεταβλητή αυτή γίνεται true, και το αντίστοιχο animation (που ελέγχεται μέσω του Animator) ξεκινά.
- CanMove: Ελέγχει αν το Flying Eye μπορεί να κινηθεί, κάτι που εξαρτάται από την τρέχουσα κατάσταση του animator. Αν το Flying Eye βρίσκεται σε κατάσταση επίθεσης ή θανάτου, αυτό είναι false.

```
private void Awake()
{
    animator = GetComponent<Animator>();
    rb = GetComponent<Rigidbody2D>();
    damageable = GetComponent<Damageable>();
}

Unity Message | 0 references
private void Start()
{
    nextWaypoint = waypoints[waypointNum];
}

Unity Message | 0 references
void Update()
{
    HasTarget = biteDetectionZone.detectedColliders.Count > 0;
}

Unity Message | 0 references
private void FixedUpdate()
{
    if (damageable.IsAlive)
    {
        if (CanMove)
        {
            Flight();
        }
        else
        {
            rb.velocity = Vector3.zero;
        }
    }
}
```

Αρχικοποίηση των βασικών component και ορισμός του πρώτου waypoint.

- Στο Awake(), αρχικοποιούμε τα βασικά components (Animator, Rigidbody2D, Damageable).
- Στο Start(), ορίζεται το πρώτο waypoint στο οποίο θα κατευθυνθεί το Flying Eye.

Οι δύο βασικές μέθοδοι για τον έλεγχο της κατάστασης και της κίνησης του Flying Eye.

- Στο Update(), ελέγχεται αν ο παίκτης βρίσκεται στη ζώνη επίθεσης μέσω του biteDetectionZone. Αν υπάρχουν colliders στη ζώνη, τότε το HasTarget γίνεται true.

- Στο FixedUpdate(), γίνεται ο έλεγχος της κίνησης. Αν το CanMove είναι true (δηλαδή, αν το Flying Eye δεν είναι σε κατάσταση επίθεσης ή θανάτου), καλείται η συνάρτηση Flight() για να κινηθεί προς το επόμενο waypoint. Αν όχι, το Rigidbody2D σταματάει (δηλαδή η ταχύτητα γίνεται μηδέν).

```
private void Flight()
{
    Vector2 directionToWaypoint = (nextWaypoint.position - transform.position).normalized;
    float distance = Vector2.Distance(nextWaypoint.position, transform.position);

    rb.velocity = directionToWaypoint * flightSpeed;
    UpdateDirection();

    if (distance <= waypointReachedDistance)
    {
        waypointNum++;
        if (waypointNum >= waypoints.Count)
        {
            waypointNum = 0;
        }

        nextWaypoint = waypoints[waypointNum];
    }
}
```

Αυτή είναι η συνάρτηση που κινεί το Flying Eye από το ένα waypoint στο άλλο.

- directionToWaypoint: Υπολογίζει την κατεύθυνση προς το επόμενο waypoint.
- rb.velocity: Ορίζει την ταχύτητα του Flying Eye βάσει της κατεύθυνσης και της ταχύτητας πτήσης.
- Αν η απόσταση μέχρι το waypoint είναι μικρότερη από την waypointReachedDistance, τότε το Flying Eye προχωράει στο επόμενο waypoint.
- Το Flying Eye επιστρέφει στο πρώτο waypoint όταν φτάσει στο τελευταίο.

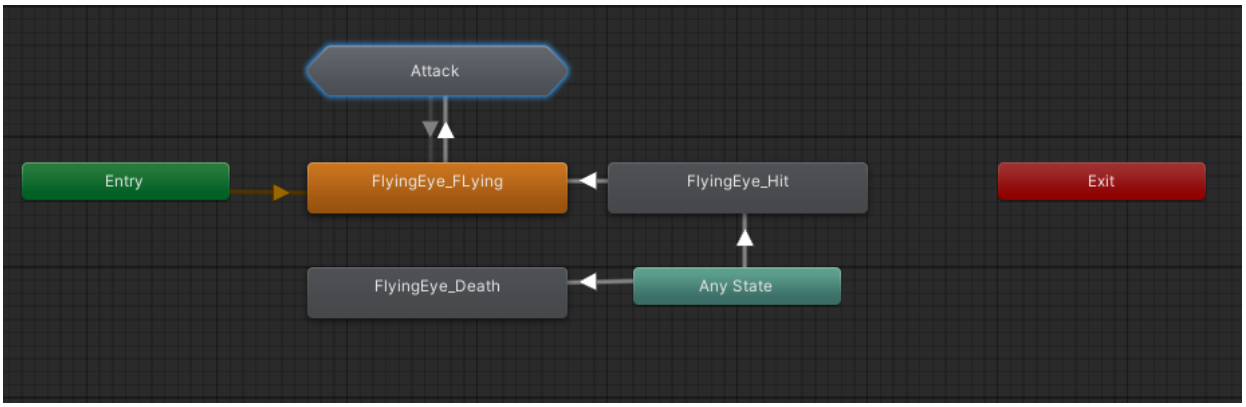
```
1 reference
private void UpdateDirection()
{
    Vector3 locScale = transform.localScale;

    if (transform.localScale.x > 0)
    {
        if (rb.velocity.x < 0)
        {
            transform.localScale = new Vector3(-1 * locScale.x, locScale.y, locScale.z);
        }
    }
    else
    {
        if (rb.velocity.x > 0)
        {
            transform.localScale = new Vector3(-1 * locScale.x, locScale.y, locScale.z);
        }
    }
}

0 references
public void OnDeath()
{
    rb.gravityScale = 2f;
    rb.velocity = new Vector2(0, rb.velocity.y);
    deathCollider.enabled = true;
}
```

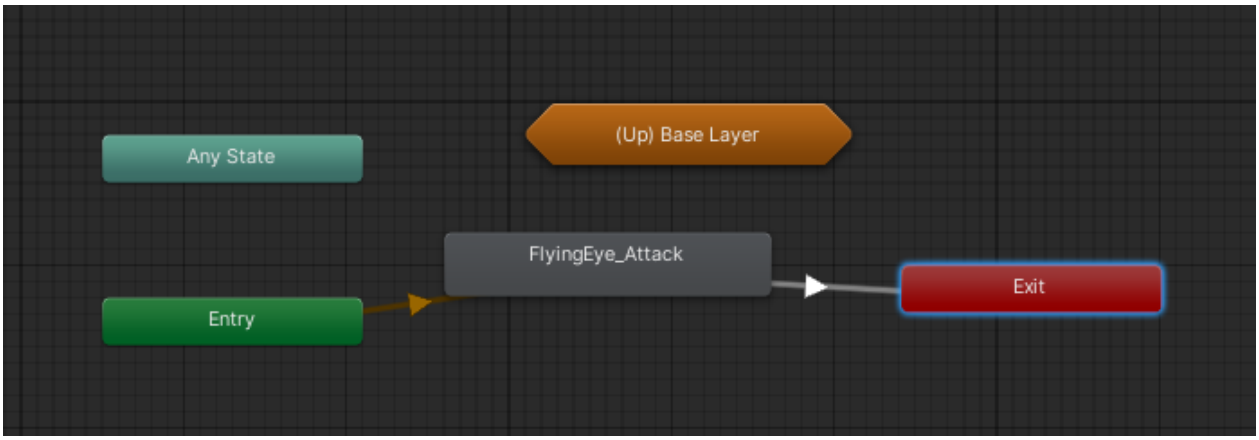
Αλλάζει την κατεύθυνση του Flying Eye ώστε να κοιτάζει προς τη σωστή κατεύθυνση ανάλογα με το αν κινείται δεξιά ή αριστερά.

Η συνάρτηση που καλείται όταν το Flying Eye πεθάνει. Ρυθμίζει τη βαρύτητα και την ταχύτητα για την πτώση του σώματος μετά τον θάνατο.

- Αυξάνει τη βαρύτητα για να πέσει το σώμα του Flying Eye στο έδαφος.
- Ενεργοποιεί τον deathCollider, ο οποίος μπορεί να αλληλεπιδράσει με το περιβάλλον (π.χ., παγίδες ή άλλες επιπτώσεις).

Εικόνες από το Animator του Flying Eye :





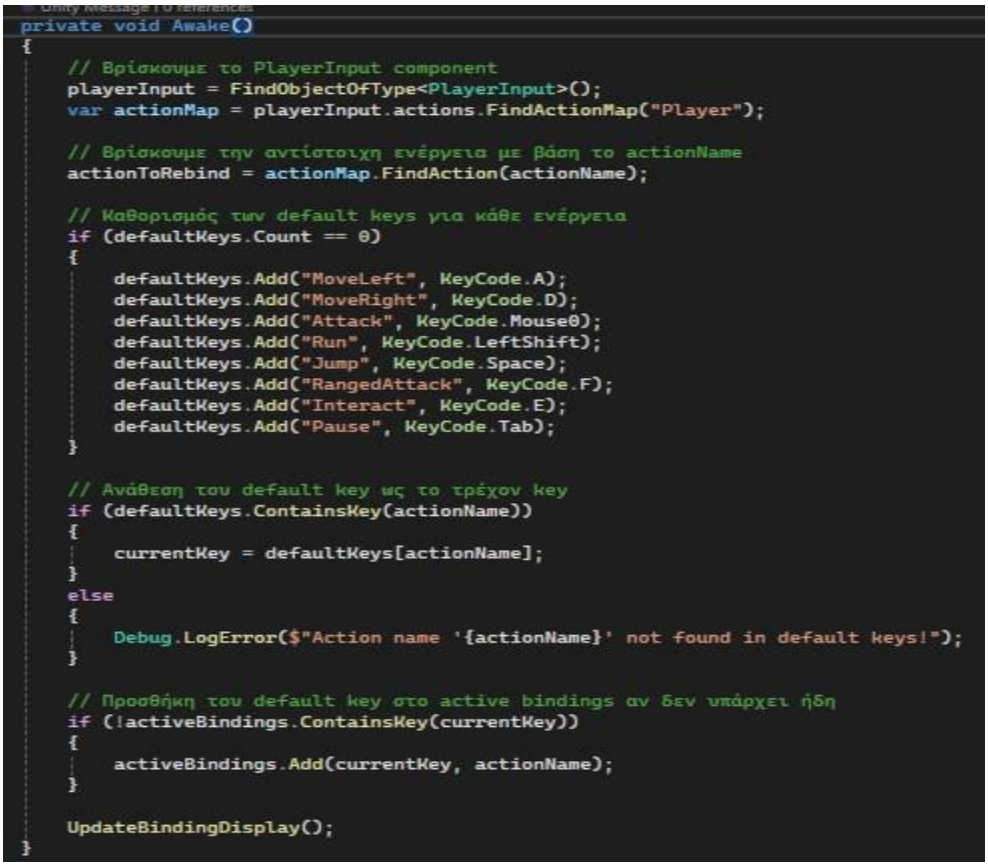
RebindButton.cs και ResetBindings.cs

Στην αρχή του κώδικα, έχουμε τις δηλώσεις των βασικών μεταβλητών:

```
public class RebindButton : MonoBehaviour
{
    public string actionName; // Μοναδικό όνομα για κάθε ενέργεια
    public TextMeshProUGUI bindingDisplayName;

    private KeyCode currentKey;
    private static Dictionary<string, KeyCode> defaultKeys = new Dictionary<string, KeyCode>();
    private static Dictionary<KeyCode, string> activeBindings = new Dictionary<KeyCode, string>();
```

- actionName: Μοναδικό όνομα που αντιστοιχεί σε κάθε ενέργεια του παίκτη (π.χ., "MoveLeft", "Jump"). Χρησιμοποιείται για να προσδιορίσουμε ποια ενέργεια θέλουμε να αναθέσουμε σε ένα κουμπί.
- bindingDisplayName: Το TextMeshProUGUI πεδίο που εμφανίζει το όνομα του κουμπιού στην οθόνη του παίκτη.
- currentKey: Αποθηκεύει το τρέχον πλήκτρο που είναι ανατεθειμένο στην ενέργεια αυτή.
- defaultKeys: Ένα λεξικό που αποθηκεύει τις προεπιλεγμένες τιμές κουμπιών για κάθε ενέργεια (π.χ., "MoveLeft" -> KeyCode.A).
- activeBindings: Ένα λεξικό που αποθηκεύει τις τρέχουσες αναθέσεις κουμπιών (π.χ., KeyCode.A -> "MoveLeft").

Στην Awake(), αρχικοποιούνται τα παρακάτω:



```
private void Awake()
{
    // Βρίσκουμε το PlayerInput component
    playerInput = FindObjectOfType<PlayerInput>();
    var actionMap = playerInput.actions.FindActionMap("Player");

    // Βρίσκουμε την αντίστοιχη ενέργεια με βάση το actionName
    actionToRebind = actionMap.FindAction(actionName);

    // Καθορισμός των default keys για κάθε ενέργεια
    if (defaultKeys.Count == 0)
    {
        defaultKeys.Add("MoveLeft", KeyCode.A);
        defaultKeys.Add("MoveRight", KeyCode.D);
        defaultKeys.Add("Attack", KeyCode.Mouse0);
        defaultKeys.Add("Run", KeyCode.LeftShift);
        defaultKeys.Add("Jump", KeyCode.Space);
        defaultKeys.Add("RangedAttack", KeyCode.F);
        defaultKeys.Add("Interact", KeyCode.E);
        defaultKeys.Add("Pause", KeyCode.Tab);
    }

    // Ανάθεση του default key ως το τρέχον key
    if (defaultKeys.ContainsKey(actionName))
    {
        currentKey = defaultKeys[actionName];
    }
    else
    {
        Debug.LogError($"Action name '{actionName}' not found in default keys!");
    }

    // Προσθήκη του default key στο active bindings αν δεν υπάρχει ήδη
    if (!activeBindings.ContainsKey(currentKey))
    {
        activeBindings.Add(currentKey, actionName);
    }

    UpdateBindingDisplay();
}
```

- playerInput: Βρίσκεται το PlayerInput component που χρησιμοποιείται για τη διαχείριση των ενεργειών του παίκτη.
- actionMap: Βρίσκεται ο χάρτης ενεργειών που περιλαμβάνει όλες τις ενέργειες του παίκτη.
- actionToRebind: Βρίσκεται η συγκεκριμένη ενέργεια που αντιστοιχεί στο actionName, δηλαδή η ενέργεια που θέλουμε να αλλάξουμε.

Ορίζονται οι προεπιλεγμένες τιμές κουμπιών για τις ενέργειες του παίκτη. Αν το defaultKeys είναι κενό, προσθέτουμε τις τιμές:

Π.χ. "MoveLeft" -> KeyCode.A σημαίνει ότι το πλήκτρο A αντιστοιχεί στην κίνηση προς τα αριστερά.

Επίσης ορίζεται το προεπιλεγμένο πλήκτρο που αντιστοιχεί στο actionName. Αν υπάρχει προεπιλεγμένο πλήκτρο στο defaultKeys για την ενέργεια, το αποθηκεύουμε στο currentKey. Αν όχι, εμφανίζεται ένα σφάλμα στο console.

Τέλος, αν το currentKey δεν έχει ήδη προστεθεί στο activeBindings, προστίθεται και ενημερώνεται η εμφάνιση του κουμπιού στον παίκτη μέσω της μεθόδου UpdateBindingDisplay().

Η StartRebinding() ξεκινάει τη διαδικασία αλλαγής κουμπιού. Ο παίκτης καλείται να πατήσει το νέο πλήκτρο, ενώ η ένδειξη στον παίκτη ενημερώνεται (το bindingDisplayName γίνεται "...").

```
0 references
public void StartRebinding()
{
    Debug.Log($"Starting rebinding for key: {currentKey}");
    bindingDisplayName.text = "...";
    StartCoroutine(WaitForKeyPress());
}
```

Η WaitForKeyPress() είναι ένα IEnumerator που περιμένει τον παίκτη να πατήσει κάποιο πλήκτρο. Κάθε καρέ, ελέγχουμε αν έχει πατηθεί κάποιο πλήκτρο. Όταν πατηθεί, το αποθηκεύουμε ως newKey και βγαίνουμε από τη λούπα.

```
private System.Collections.IEnumerator WaitForKeyPress()
{
    bool keyAssigned = false;
    KeyCode newKey = KeyCode.None;

    while (!keyAssigned)
    {
        foreach (KeyCode keyCode in System.Enum.GetValues(typeof(KeyCode)))
        {
            if (Input.GetKeyDown(keyCode))
            {
                newKey = keyCode;
                keyAssigned = true;
                break;
            }
        }
        yield return null;
    }

    // Έλεγχος αν το νέο κουμπί χρησιμοποιείται ήδη
    if (activeBindings.ContainsKey(newKey))
    {
        Debug.Log($"The key {newKey} is already in use by {activeBindings[newKey]}.");
        UpdateBindingDisplay(); // Επαναφορά στο προηγούμενο binding
    }
    else
    {
        // Αφαίρεση του προηγούμενου binding από το dictionary
        activeBindings.Remove(currentKey);

        // Ενημέρωση του binding στο σύστημά σου
        UpdateInputSystemBinding(newKey);

        // Ενημέρωση του binding
        currentKey = newKey;

        // Προσθήκη του νέου binding στο dictionary
        activeBindings.Add(currentKey, actionName);

        UpdateBindingDisplay();
    }
}
```

Αν το νέο πλήκτρο είναι ήδη σε χρήση (υπάρχει στο activeBindings), ενημερώνουμε τον παίκτη και επαναφέρουμε το προηγούμενο binding. Αν δεν είναι σε χρήση, αφαιρούμε το

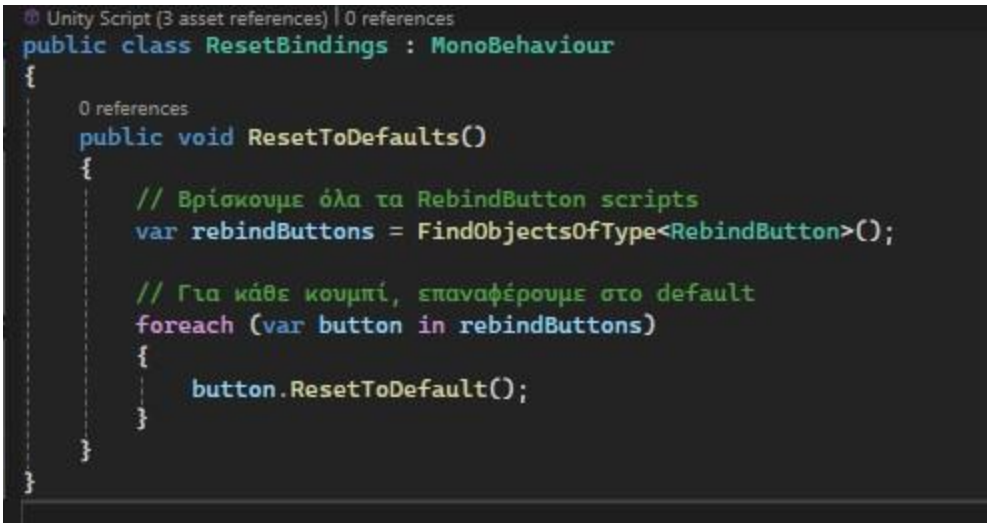
προηγούμενο binding, ενημερώνουμε το σύστημα για τη νέα αντιστοίχιση με τη μέθοδο UpdateInputSystemBinding(), και τελικά ενημερώνουμε το activeBindings και το currentKey.

```
2 references
private void UpdateInputSystemBinding(KeyCode newKey)
{
    if (actionToRebind != null)
    {
        string newBindingPath = "";

        // Έλεγχος αν το νέο πλήκτρο είναι του ποντικιού
        if (newKey == KeyCode.Mouse0)
        {
            newBindingPath = "<Mouse>/leftButton"; // Αριστερό κλικ
        }
        else if (newKey == KeyCode.Mouse1)
        {
            newBindingPath = "<Mouse>/rightButton"; // Δεξί κλικ
        }
        else
        {
            newBindingPath = $"<Keyboard>/{newKey}";
        }

        // Εφαρμογή του νέου binding στο Input System
        actionToRebind.ApplyBindingOverride(new InputBinding { overridePath = newBindingPath });
        Debug.Log($"Updated Input System binding for {actionName} to {newBindingPath}");
    }
}
```

Αυτή η μέθοδος ενημερώνει το Unity Input System για το νέο πλήκτρο. Ανάλογα με το νέο πλήκτρο, δημιουργείται ένα νέο binding path (για το ποντίκι ή το πληκτρολόγιο), και αυτό εφαρμόζεται μέσω της ApplyBindingOverride().

Η μέθοδος ResetToDefault() επαναφέρει το binding στην προεπιλεγμένη του τιμή, χρησιμοποιώντας το defaultKeys. Αφαιρεί το τρέχον binding από το activeBindings, ενημερώνει το Input System, και προσθέτει την προεπιλεγμένη τιμή στο activeBindings.

```
public void ResetToDefault()
{
    // Αφαίρεση του τρέχοντος binding από το dictionary
    if (activeBindings.ContainsKey(currentKey))
    {
        activeBindings.Remove(currentKey);
    }

    // Επαναφορά στο default key
    if (defaultKeys.ContainsKey(actionName))
    {
        currentKey = defaultKeys[actionName];
    }

    // Ενημέρωση του συστήματος με το default key
    UpdateInputSystemBinding(currentKey);

    // Προσθήκη του default key στο dictionary
    if (!activeBindings.ContainsKey(currentKey))
    {
        activeBindings.Add(currentKey, actionName);
    }

    UpdateBindingDisplay();
}
```



Αυτό το script είναι απλό και επαναφέρει όλα τα bindings στις προεπιλεγμένες τιμές. Η μέθοδος ResetToDefaults() βρίσκει όλα τα RebindButton scripts στο παιχνίδι και καλεί για το καθένα τη μέθοδο ResetToDefault(), επαναφέροντας έτσι όλες τις ενέργειες στις προεπιλεγμένες τιμές.

Συνοψίζοντας:

- Το **RebindButton**.cs επιτρέπει την αλλαγή κουμπιών μέσω αναμονής για την εισαγωγή του παίκτη και ελέγχει αν το νέο πλήκτρο είναι ήδη δεσμευμένο. Ενημερώνει το Unity Input System για τη νέα ρύθμιση.
- Το **ResetBindings**.cs επαναφέρει όλα τα bindings στις προεπιλεγμένες τιμές τους.

Αυτή η δομή προσφέρει ευελιξία στους παίκτες να αλλάξουν τις ρυθμίσεις ελέγχου όπως επιθυμούν, ενώ διατηρείται η δυνατότητα επαναφοράς στις αρχικές τιμές αν χρειαστεί.

Αξιολόγηση Παιχνιδιού

Η αξιολόγηση του παιχνιδιού έγινε μέσα από τη δοκιμή του από τρεις άτομα, οι οποίοι έπαιξαν το παιχνίδι και έδωσαν ανατροφοδότηση για την εμπειρία τους. Παρακάτω παρουσιάζονται οι κριτικές τους, οι οποίες αναδεικνύουν τα θετικά στοιχεία του παιχνιδιού, καθώς και μία πιο αναλυτική κριτική που παρέχει ένα σημαντικό σημείο προς σκέψη.

Κριτική 1

«Το παιχνίδι προσφέρει μια πολύ ωραία εμπειρία και καταφέρνει να συνδυάσει τη μυθολογία με διασκεδαστικό gameplay. Οι μηχανισμοί μάχης είναι απλοί αλλά αποτελεσματικοί, και η πρόκληση με το τελικό boss, τον Κύκλωπα, προσφέρει ένα ικανοποιητικό επίπεδο δυσκολίας. Η πλοκή είναι ενδιαφέρουσα και δίνει στον παίκτη μια καλή αίσθηση του ταξιδιού του Οδυσσέα. Συνολικά, πρόκειται για ένα διασκεδαστικό παιχνίδι που καταφέρνει να κρατήσει τον παίκτη εντός της ιστορίας.»

Κριτική 2

«Είναι εμφανές ότι το παιχνίδι έχει σχεδιαστεί με προσοχή και φροντίδα στη λεπτομέρεια. Η απεικόνιση των σημαντικών γεγονότων της Οδύσσειας γίνεται με τρόπο που καθιστά το παιχνίδι ελκυστικό, τόσο για όσους γνωρίζουν την ιστορία του Οδυσσέα όσο και για νέους παίκτες. Η μάχη με τον Κύκλωπα ήταν το αγαπημένο μου κομμάτι, καθώς η δυσκολία ήταν ικανοποιητική και οι εχθροί είχαν ποικιλία. Το παιχνίδι δίνει επίσης μια καλή εικόνα της προσωπικότητας του Οδυσσέα και της πορείας του.»

Κριτική 3

«Αυτό που παρατήρησα παίζοντας το παιχνίδι, και κατάλαβα μετά από μερικές φορές που έχασα, είναι ότι δεν χρειάζεται να αντιμετωπίσεις όλους τους εχθρούς που βρίσκονται στον δρόμο σου. Αρκεί απλώς να τους προσπεράσεις όταν μπορείς, ακριβώς όπως έκανε και ο Οδυσσέας με αρκετά από τα εμπόδια που βρέθηκαν στον δρόμο του. Αυτή η παρατήρηση μου έδειξε πόσο καλοσχεδιασμένο είναι το παιχνίδι και πόσο πιστό είναι στην ιστορία που διηγείται. Το παιχνίδι είναι επίσης μια εξαιρετική ευκαιρία να μάθει κάποιος βασικά πράγματα για την ιστορία του Οδυσσέα και τις περιπέτειές του.»

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Η ανάπτυξη του παιχνιδιού "Οι περιπέτειες του Οδυσσέα" αποτέλεσε μια σημαντική πρόκληση και ταυτόχρονα μια ιδιαίτερα δημιουργική διαδικασία. Μέσα από αυτή την εργασία, κατάφερα να συνδυάσω τις γνώσεις που απέκτησα κατά τη διάρκεια των σπουδών μου με την προσωπική μου αγάπη για τη μυθολογία και τα βιντεοπαιχνίδια. Η υλοποίηση του παιχνιδιού προσέφερε την ευκαιρία να εξερευνήσω σε βάθος την πλατφόρμα της Unity, την ανάπτυξη 2D παιχνιδιών, και την εφαρμογή μηχανισμών μάχης, αφήγησης και περιπέτειας.

Τα κυριότερα συμπεράσματα που προέκυψαν από τη διαδικασία ανάπτυξης αφορούν τη σημασία της σωστής διαχείρισης του προγραμματισμού και της οργάνωσης του έργου. Η έλλειψη εμπειρίας με τη Unity στην αρχή της εργασίας αποτέλεσε ένα από τα μεγαλύτερα εμπόδια, όμως, με την πάροδο του χρόνου και μέσα από τη συνεχή πρακτική, κατάφερα να ξεπεράσω τις αρχικές δυσκολίες και να ολοκληρώσω ένα παιχνίδι που να ανταποκρίνεται στις αρχικές προσδοκίες μου.

1. **Προσθήκη περισσότερων επιπέδων:** Η τρέχουσα έκδοση του παιχνιδιού εστιάζει σε μερικές από τις πιο σημαντικές περιπέτειες του Οδυσσέα. Μια μελλοντική επέκταση θα μπορούσε να περιλαμβάνει περισσότερα επίπεδα και προκλήσεις, ώστε να καλυφθούν περισσότερα κομμάτια της ιστορίας του Οδυσσέα.

2. **Βελτίωση της AI των εχθρών:** Αν και η τεχνητή νοημοσύνη των εχθρών λειτουργεί ικανοποιητικά, υπάρχει περιθώριο για βελτίωση, με σκοπό να προσφέρουν μεγαλύτερη πρόκληση στον παίκτη και πιο ρεαλιστικές αντιδράσεις.

3. **Εμπλουτισμός της αφήγησης:** Η ιστορία του Οδυσσέα μπορεί να αναπτυχθεί περισσότερο, με προσθήκη διαλόγων, περισσότερων cutscenes και αφηγηματικών στοιχείων που θα δώσουν στο παιχνίδι μεγαλύτερο βάθος και θα επιτρέψουν στον παίκτη να συνδεθεί περισσότερο με τους χαρακτήρες.

4. **Βελτιστοποίηση του συστήματος μάχης:** Μια πιο εκλεπτυσμένη μηχανική μάχης, με περισσότερες κινήσεις και ειδικές επιθέσεις, θα μπορούσε να προσφέρει μια πιο δυναμική και ενδιαφέρουσα εμπειρία στον παίκτη.

5. **Δημιουργία συστήματος αναβαθμίσεων:** Ένα σύστημα που θα επιτρέπει στον παίκτη να αναβαθμίζει τον χαρακτήρα του με βάση την πρόοδό του στο παιχνίδι θα μπορούσε να προσθέσει μια νέα διάσταση στο gameplay, ενθαρρύνοντας τον παίκτη να εξερευνά και να επιτυγχάνει νέους στόχους.

6. **Προσθήκη συστήματος αποθήκευσης και φόρτωσης παιχνιδιού:** Στα πλαίσια της μελλοντικής ανάπτυξης, σχεδιάζεται η υλοποίηση ενός πλήρους συστήματος save/load που θα επιτρέπει στους παίκτες να αποθηκεύουν την πρόοδό τους και να επιστρέφουν

αργότερα στο σημείο που άφησαν το παιχνίδι. Αυτό το χαρακτηριστικό θα βελτιώσει σημαντικά την εμπειρία του παίκτη, επιτρέποντας μεγαλύτερη ελευθερία και ευελιξία, καθώς ο χρήστης δεν θα χρειάζεται να ξεκινά το παιχνίδι από την αρχή σε κάθε νέα συνεδρία.

Συνολικά, η ανάπτυξη του παιχνιδιού ήταν μια δημιουργική και διαφωτιστική εμπειρία. Αν και υπάρχουν δυνατότητες βελτίωσης, το παιχνίδι επιτυγχάνει τον στόχο του: να παρουσιάσει με διασκεδαστικό και διδακτικό τρόπο ορισμένες από τις περιπέτειες του Οδυσσέα, ενώ ταυτόχρονα προσφέρει στους παίκτες μια προκλητική και ευχάριστη εμπειρία.

Βιβλιογραφία – Υπερσύνδεσμοι

Στην ανάπτυξη του παιχνιδιού "Οι περιπέτειες του Οδυσσέα", χρησιμοποιήθηκαν διάφορα assets και πόροι από τις παρακάτω πηγές:

- **Itch.io:** Η κύρια πηγή για τα περισσότερα assets, συμπεριλαμβανομένων γραφικών, animations, και μοντέλων χαρακτήρων.
https://itch.io/

- **FreeMusic:** Οι ηχητικές επενδύσεις του παιχνιδιού προήλθαν από το FreeMusic, ενώ οι ήχοι τροποποιήθηκαν περαιτέρω από εμένα μέσω του προγράμματος Audacity για να ταιριάζουν καλύτερα στο παιχνίδι.
https://freesound.org/

- **ChrisTutorialsYT (YouTube):** Το κανάλι του ChrisTutorialsYT στο YouTube αποτέλεσε βασική πηγή γνώσεων και resources. Από αυτό το κανάλι προήλθαν τα περισσότερα characters, sounds, και γενικότερα τα assets που χρησιμοποιήθηκαν στο παιχνίδι.
https://www.youtube.com/@ChrisTutorialsYT