



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Ελληνικά: Ανάπτυξη 2D παιχνιδιού πλατφόρμας σε μεσαιωνικό περιβάλλον: «Μεσαιωνικές Περιπέτειες» Αγγλικά: Development of a 2D platform game in a medieval environment: "Medieval Adventures"
Ονοματεπώνυμο Φοιτητή	Πούλαδος Κωνσταντίνος
Πατρώνυμο	Στυλιανός
Αριθμός Μητρώου	Π22146
Επιβλέπων	Ευθύμιος Αλέπης, Καθηγητής

Ημερομηνία Παράδοσης: Σεπτέμβριος 2026

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 τον καθηγητή μου, κύριο Ευθύμιο Αλέπη και τον βοηθό του Μαργαρίτη Καμήτσιο για την βοήθεια τους στην εκπόνηση αυτής της εργασίας.

Θα ήθελα επίσης να ευχαριστήσω όλους τους αγαπημένους μου ανθρώπους που με στηρίζουν πάντα και με αγαπάνε, και ειδικά αυτούς που έπαιξαν το παιχνίδι και βρήκαν λάθη που δεν είχα εντοπίσει εγώ. Σας ευχαριστώ και σας αγαπάω πολύ.

Περίληψη

Η παρούσα πτυχιακή εργασία με τίτλο: Ανάπτυξη 2D παιχνιδιού πλατφόρμας σε μεσαιωνικό περιβάλλον: «Μεσαιωνικές Περιπέτειες» είναι ένα 2D side-scroller platformer παιχνίδι υλοποιημένο χρησιμοποιώντας την μηχανή γραφικών Unity και τη γλώσσα C#. Το παιχνίδι αποτελείται από πέντε (5) επίπεδα τα οποία ξεκλειδώνει σταδιακά ο παίκτης. Έχει κάποιους βασικούς μηχανισμούς ευρέως διαδεδομένους στα ηλεκτρονικά παιχνίδια κατά την ιστορία, σε μια αρχική τους μορφή. Για παράδειγμα, υπάρχει σύστημα μάχης και εχθροί που μπορεί να χτυπήσει ο παίκτης, καθώς και ένας πιο δύσκολος εχθρός στο τέλος κάθε επιπέδου (boss fight). Επιπλέον, υπάρχουν κάποιες αναβαθμίσεις (upgrades) που μπορεί να αποκτήσει ο παίκτης στην περιπέτειά του, και άλλοι μηχανισμοί που θα αναλυθούν στη συνέχεια, όπως η μουσική. Η εργασία είναι εμπνευσμένη από παιχνίδια που έχω παίξει και είναι ορόσημο στη βιομηχανία και ταυτόχρονα αποτελούν έμπνευση για νέους ανερχόμενους game developers.

Λέξεις κλειδιά: Unity, 2D, C#, platformer

Abstract

This thesis titled: Development of a 2D platform game in a medieval environment: “Medieval Adventures” is a 2D side-scroller platformer game developed using the Unity graphics engine and the programming language C#. The game contains five (5) levels which the player unlocks gradually. It has some basic mechanics widely used in videogames throughout history, in a basic form. For instance, there is a combat system and enemies that the player can attack, in addition to a more difficult enemy in the end of each level (boss fight). Moreover, there are upgrades the player can obtain in their adventure, and other mechanisms that will be analysed, like the music. The project is inspired by videogames I have played and are landmarks in the industry, and are also inspiring for young upcoming developers.

Key words: Unity, 2D, C#, platformer

Πίνακας Περιεχομένων

Copyright ©.....	1
Ευχαριστίες.....	2
Περίληψη.....	3
Abstract.....	3
Πίνακας Περιεχομένων.....	4
1. Εισαγωγή.....	7
1.1 Δομή εργασίας.....	9
2. Μεθοδολογία.....	10
2.1 Σκοπός.....	10
2.2 Ανάλυση.....	10
2.3 Σχεδίαση.....	10
2.4 Υλοποίηση.....	10
3. Σχεδιασμός Παιχνιδιού.....	11
3.1 Απαιτήσεις.....	11
3.2 Σενάριο.....	11
3.3 Gameplay - Walkthrough.....	12
3.4 Σκηνές.....	13
3.4.1 StartingMenu.....	13
3.4.2 LevelSelection.....	14
3.4.3 Level1.....	14
3.4.4 Level2.....	15
3.4.5 Level3.....	15
3.4.6 Level4.....	16
3.4.7 Level5.....	16
3.4.8 House.....	17
3.4.9 House2.....	17
3.5 UI.....	18
3.5.1 Βασικό UI.....	18
3.5.2 Αρχικό Μενού.....	20
3.5.3 Μενού Επιλογής Πίστας.....	23
3.5.4 Μενού Παύσης.....	24
3.5.5 Μενού Ήττας.....	25
4. Ανάλυση και Αρχιτεκτονική.....	26
4.1 Βασικά συστήματα.....	26
4.1.1 Παίκτης.....	26
4.1.2 Έδαφος - Δομή πίστας.....	30
4.1.3 Checkpoint.....	31
4.1.4 Μουσική - Ήχοι.....	32

4.1.5 Chests – Upgrades.....	33
4.1.6 Εχθροί.....	35
4.1.7 Final Bosses.....	36
4.2 Ροή δεδομένων.....	38
4.3 UML διαγράμματα.....	39
4.3.1 Use Case Diagram.....	39
4.3.2 Flow Diagram.....	40
5. Ανάλυση Scripts.....	40
5.1 ActivateUIDialogue.....	40
5.2 Bandit.....	41
5.3 DeathPatrolling & Patrolling.....	42
5.3.1 DeathPatrolling.....	42
5.3.2 Patrolling.....	42
5.4 EnterBossArena.....	43
5.5 EnterBuilding.....	43
5.6 FollowingCameraTarget.....	43
5.7 HealthBar.....	44
5.8 HeroKnight.....	44
5.9 LampCheckpoint.....	45
5.10 LoadLevels.....	46
5.11 LoseMenu.....	46
5.12 MainMenu.....	47
5.13 PauseMenu.....	47
5.14 SceneSpawnPoint.....	49
5.15 SelectLevel.....	49
6. Προβλήματα που προέκυψαν.....	49
7. Συμπεράσματα και μελλοντική επέκταση.....	50
7.1 Συμπεράσματα.....	50
7.2 Μελλοντική επέκταση.....	50
8. Ορολογία.....	51
9. Βιβλιογραφία.....	51
10. Παραρτήματα.....	53
10.1 Παράρτημα.....	53
10.2 Παράρτημα.....	54
10.3 Παράρτημα.....	57
10.4 Παράρτημα.....	60
10.5 Παράρτημα.....	62
10.6 Παράρτημα.....	63
10.7 Παράρτημα.....	65
10.8 Παράρτημα.....	66
10.9 Παράρτημα.....	67

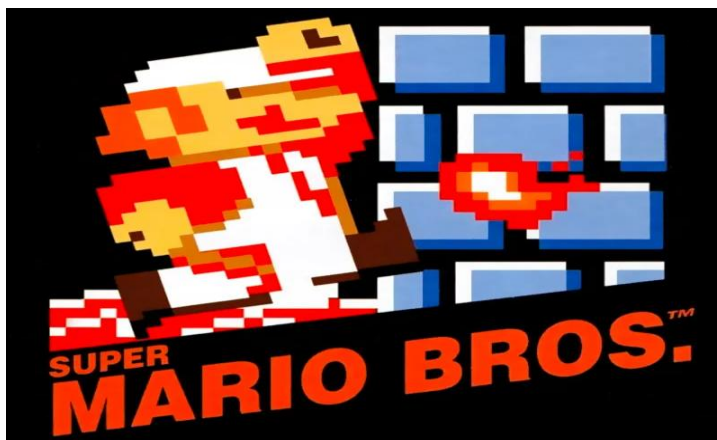
10.10 Παράρτημα.....	79
10.11 Παράρτημα.....	80
10.12 Παράρτημα.....	82
10.13 Παράρτημα.....	82
10.14 Παράρτημα.....	86
10.15 Παράρτημα.....	87
10.16 Παράρτημα.....	88

1. Εισαγωγή

Τα ηλεκτρονικά παιχνίδια (videogames) έχουν μια πολύ πλούσια ιστορία. Ξεκίνησαν από κάτι πολύ απλό όπως το να χτυπάει μια μπάλα σε μια πλευρά της οθόνης και να πηγαίνει στην άλλη πλευρά, σε κάτι πολύπλοκο που προσφέρει τεράστια χαρά σε πλήθος κόσμου. Τα ηλεκτρονικά παιχνίδια είναι πλέον ένα τεράστιο μέσο ψυχαγωγίας, στα οποία ο καταναλωτής δεν είναι παθητικός, και απευθύνονται σε άτομα κάθε ηλικίας που ο καθένας μπορεί να παίξει όποιο παιχνίδι προτιμάει. Παιχνίδι που φτιάχτηκε από μεγάλη εταιρεία με τεράστια οικονομική δύναμη και χιλιάδες εργαζόμενους, παιχνίδι που φτιάχτηκε από ομάδα δέκα ατόμων με ένα όνειρο και περιορισμένες δυνατότητες, παιχνίδι που επιδιώκει να πει μια πολύπλοκη ιστορία, παιχνίδι με κάρτες κλπ. Το νόημα είναι ότι υπάρχει ένα παιχνίδι για τον καθένα σε αυτή τη τεράστια βιομηχανία. Και ότι πέρα από ευτυχία, ένα παιχνίδι μπορεί να παρέχει συντροφιά σε κάποιον σε δύσκολες στιγμές.

Εγώ ξεκίνησα να ασχολούμαι με τα βιντεοπαιχνίδια από τα τέλη του δημοτικού και κέρδισαν το ενδιαφέρον μου ακόμα περισσότερο στο γυμνάσιο. Εκεί λάτρεψα διάφορα είδη παιχνιδιών αλλά ιδιαίτερα αυτά που έχουν να πουν μια καλή ιστορία. Έως και σήμερα τα παιχνίδια παραμένουν ένα μεγάλο πάθος στη ζωή μου και κάτι που λατρεύω να κάνω. Μου έχουν προσφέρει αμέτρητες στιγμές ευτυχίας, θυμού, συγκίνησης, μελαγχολίας. Πολλά παιχνίδια αποτελούν και πηγή έμπνευσης για εμένα αλλά σε αυτή την εργασία εμπνεύστηκα κυρίως από δυο. Δυο παιχνίδια που είναι τόσο παρόμοια μεταξύ τους αλλά και ταυτόχρονα διαφορετικά.

Το πρώτο παιχνίδι είναι γενικά όλη η σειρά των παιχνιδιών Super Mario Brothers από την Nintendo. Ίσως το πιο ιστορικό παιχνίδι και πιθανόν ο πιο αναγνωρίσιμος χαρακτήρας της βιομηχανίας, ο Mario, ακόμα και από άτομα που δεν ασχολούνται με τα παιχνίδια. Το παιχνίδι αυτό είναι φαινομενικά απλό. Έχει πίστες όπου ο παίκτης ξεκινάει από τα αριστερά και πάει προς τα δεξιά, αποφεύγοντας τυχόν εμπόδια και εχθρούς, μέχρι να φτάσει στο τέλος της πίστας. Σκοπός του είναι να νικήσει όλες τις πίστες για να σώσει την πριγκίπισσα. Η εταιρεία που έφτιαξε το παιχνίδι, η Nintendo, είναι μια ιστορική επιχείρηση με πολύ φανατικό κοινό και μερικά από τα πιο θρυλικά παιχνίδια στην ιστορία των βιντεοπαιχνιδιών. Όπως φαίνεται και από τις παρακάτω εικόνες, το παιχνίδι έχει εξελιχθεί πάρα πολύ σχετικά με το πρώτο στη σειρά, από τα γραφικά μέχρι τους πιο πολύπλοκους μηχανισμούς. Η ουσία όμως παραμένει η ίδια και αποτελεί ακόμα ένα παιχνίδι όπου και ένας αρχάριος μπορεί να καταλάβει εύκολα τι χρειάζεται να κάνει.



1.1 Το πρώτο Super Mario Bros, στο Nintendo Entertainment System



1.2 Το πιο πρόσφατο Super Mario Bros, στο Nintendo Switch 1 & 2

Το άλλο παιχνίδι που αποτέλεσε έμπνευση για την εργασία ήταν το Hollow Knight, της Team Cherry. Σε αντίθεση με το Super Mario όπου φτιάχτηκε από μια τεράστια επιχείρηση, το Hollow Knight δημιουργήθηκε κυρίως από μια ομάδα τριών ατόμων οι οποίοι έβαλαν αυτό το εγχείρημα στο Kickstarter για οικονομική ενίσχυση. Έχει γίνει ένα από τα πιο αγαπημένα παιχνίδια αυτού του είδους για τους πλούσιους μηχανισμούς του, την μοναδική του ατμόσφαιρα αλλά και για την ιστορία της δημιουργίας του. Πέρα από το γεγονός ότι είναι εκπληκτικό σαν παιχνίδι, είχε και τεράστια οικονομική επιτυχία κάτι που ώθησε την ομάδα να φτιάξει τον διάδοχό του, το Hollow Knight: Silksong που κυκλοφόρησε πρόσφατα και ήταν ένα από τα πιο πολυαναμενόμενα παιχνίδια.



1.3 Hollow Knight, Team Cherry

Όπως αναφέρθηκε παραπάνω, τα παιχνίδια στη βασική τους ιδέα μοιάζουν αρκετά. Είναι και τα δυο 2D platformer όπου ο παίκτης πρέπει να νικήσει τους εχθρούς χρησιμοποιώντας διάφορους μηχανισμούς. Διάφερουν όμως στην ατμόσφαιρα, στην πρόκληση, στην ιστορία και σε άλλους τομείς. Χωρίς όμως το Super Mario, το Hollow Knight δεν θα υπήρχε όπως και άλλα χιλιάδες παιχνίδια για τα οποία αποτέλεσε και αποτελεί ακόμα έμπνευση. Όπως και για την παρούσα εργασία, σε διάφορους τομείς.

1.1 Δομή εργασίας

Στο κεφάλαιο 2 παρουσιάζεται η μεθοδολογία που ακολούθησα για την ανάπτυξη του παιχνιδιού. Ποιος είναι ο σκοπός του και περιληπτικά οι φάσεις της ανάλυσης, της σχεδίασης και της υλοποίησης. Στο κεφάλαιο 3 παρουσιάζεται αναλυτικά ο σχεδιασμός του παιχνιδιού. Ποιες ήταν οι απαιτήσεις του, το σενάριο και gameplay. Επιπλέον, παρουσιάζονται ξεχωριστά όλες οι σκηνές και όλα τα τμήματα του UI. Το κεφάλαιο 4 είναι για την ανάλυση και την αρχιτεκτονική του παιχνιδιού. Ποια είναι τα βασικά του συστήματα και υποενότητες για την ροή δεδομένων και UML διαγράμματα. Στο κεφάλαιο 5 αναλύονται τα scripts του παιχνιδιού και ολόκληρος ο κώδικας παρουσιάζεται στα παραρτήματα. Στο κεφάλαιο 6 αναφέρονται τα προβλήματα που προέκυψαν. Τέλος στο κεφάλαιο 7 αναγράφονται τα συμπεράσματα και η πιθανή μελλοντική επέκταση του παιχνιδιού.

2. Μεθοδολογία

2.1 Σκοπός

Ο σκοπός του παιχνιδιού είναι να καταφέρει ο παίκτης να νικήσει όλες τις πίστες. Για να τα καταφέρει πρέπει να περάσει από τα εμπόδια που υπάρχουν στο περιβάλλον, από εχθρούς και από έναν πιο δυνατό εχθρό στο τέλος κάθε πίστας. Επιλέχθηκε ένα μεσαιωνικό τοπίο για το παιχνίδι με στοιχεία φαντασίας, χωρίς να δίνεται απαραίτητα έμφαση στον ρεαλισμό.

2.2 Ανάλυση

Στο παιχνίδι ο παίκτης έχει ένα σπαθί και μια ασπίδα στα οποία θα πρέπει να βασιστεί κατά τη διάρκεια της περιπέτειάς του για να φτάσει έναν άγνωστο στόχο. Στη πορεία θα μπορεί να επιλέξει και κάποιες «αναβαθμίσεις» που θα τον βοηθήσουν να νικήσει τους εχθρούς ευκολότερα. Πέρα από τους εχθρούς που μπορούν να τον βλάψουν ο παίκτης πρέπει να αντιμετωπίσει και το ίδιο το περιβάλλον καθώς υπάρχουν παγίδες και άμα πέσει από το έδαφος θα χάσει ζωή. Επιπλέον, κάθε πίστα έχει ένα checkpoint. Αν το φτάσει ο παίκτης και χάσει μια ζωή μετά από εκεί, θα κάνει spawn στο checkpoint και όχι στην αρχή της πίστας.

2.3 Σχεδίαση

Το πρώτο βήμα ήταν η εύρεση των assets , από το Unity Asset Store, που να ταίριαζαν στο είδος παιχνιδιού που ήθελα να φτιάξω. Έχουν χρησιμοποιηθεί assets για τον παίκτη, για όλα τα γραφικά για μουσική και ήχο. Στη συνέχεια ξεκίνησα να υλοποιώ τον παίκτη όπως τον ήθελα για το παιχνίδι, το σημαντικότερο μέρος κάθε παιχνιδιού. Έπειτα, σχεδιάστηκε το αρχικό μενού, το μενού για την παύση καθώς και το μενού για την επιλογή της πίστας. Μετά άρχισε η σχεδίαση της κάθε πίστας ξεχωριστά. Αρχικά το έδαφος και η γενική δομή της πίστας και στη συνέχεια η θέση του checkpoint και των εχθρών, η μουσική και οι ήχοι, το μέρος που παίκτης μπορεί να πάρει upgrade, οι εχθροί, το final boss κάθε πίστας και τέλος το UI (User Interface – Διεπαφή Χρήστη).

2.4 Υλοποίηση

Υπάρχουν διάφορες μηχανές γραφικών για την ανάπτυξη παιχνιδιών. Κάποιες από τις πιο γνωστές είναι η Unreal Engine, η Unity και η Godot. Για αυτή την εργασία επιλέχθηκε η Unity επειδή είμαι εξοικειωμένος με αυτή, τρέχει χωρίς πολύ μεγάλη επιβάρυνση στον υπολογιστή μου (για την ανάπτυξη δισδιάστατων παιχνιδιών) και τέλος για τον κώδικα χρησιμοποιεί την γλώσσα C#, με την οποία είμαι εξοικειωμένος. Για την ανάπτυξη παιχνιδιών στο Unity χρησιμοποιούνται assets. Τα assets μπορεί να είναι γραφικά, ήχοι, εφέ, ακόμα και κομμάτια κώδικα. Σχεδόν όλα τα assets του παιχνιδιού είναι από το Unity Asset Store και είναι όλα δωρεάν. Το παιχνίδι αποτελείται

από σκηνές (scenes) μέσα στις οποίες υπάρχουν τα assets, κώδικας και ότι άλλο χρειάζεται για να λειτουργήσει σωστά το παιχνίδι.

3. Σχεδιασμός Παιχνιδιού

3.1 Απαιτήσεις

Οι λειτουργικές απαιτήσεις του παιχνιδιού είναι απλές. Ο παίκτης πρέπει να κινείται ελεύθερα στον χώρο. Δηλαδή, να μπορεί να προχωράει δεξιά και αριστερά, να πηδάει και να κάνει επίθεση. Επιπλέον, να υπάρχει αρχικό μενού και μενού παύσης και γενικότερα ένα πλήρως λειτουργικό και εύχριστο UI για να έχει μια ευχάριστη εμπειρία ο παίκτης. Επίσης, θα πρέπει να αποθηκεύονται στοιχεία όπως η πρόοδος του παίκτη και οι επιλογές του, για παράδειγμα ποιες πίστες έχει ξεκλειδώσει. Τέλος, οι ενέργειες του χαρακτήρα να έχουν αποτέλεσμα στον κόσμο. Στην περίπτωση που πέσει στο κενό να χάσει μια ζωή, αν χτυπήσει έναν εχθρό να μειώσει την ζωή του.

Οι μη λειτουργικές απαιτήσεις είναι: να τρέχει σταθερά το παιχνίδι. Ακόμη, να είναι συμβατό με Windows και τέλος να είναι απλά και κατανοητά τα controls του παιχνιδιού και να μην μπερδεύουν τον παίκτη.

3.2 Σενάριο

Το παιχνίδι δεν είναι story driven. Δηλαδή δεν παρουσιάζει κάποια ιστορία στον παίκτη ούτε εξελίσσεται κάποια πλοκή. Δίνει στον παίκτη την ελευθερία να ερμηνεύσει τον κόσμο όπως θέλει χωρίς να υπάρχει σωστό ή λάθος. Ωστόσο, στη φάση του σχεδιασμού του παιχνιδιού είχα μια ιδέα για την ιστορία στην οποία θα μπορούσα να βασιστώ για τον σχεδιασμό των επιπέδων.

Ο πρωταγωνιστής βρίσκεται σε ένα μέρος όπου είναι ο τελευταίος ζωντανός άνθρωπος. Για κάποιο λόγο έχουν απεινώσει όλοι και οι ψυχές μερικών έχουν μετατραπεί στους απλούς εχθρούς του παιχνιδιού. Δεν κάνουν προσπάθεια όμως να επιτεθούν στον πρωταγωνιστή, απλά μετακινούνται άσκοπα στον χώρο περιμένοντας να τους σκοτώσει ο παίκτης για να μπορέσει να απελευθερωθεί και η ψυχή τους. Αντιθέτως, τα bosses κάθε πίστας είναι εχθροί που έχει δημιουργήσει το final boss, η ενσάρκωση του Θανάτου, για να εμποδίσει τον πρωταγωνιστή να τον φτάσει. Στο τέταρτο επίπεδο δημιουργεί μια αντιγραφή του παίκτη σε μια τελευταία απόπειρα να τον σκοτώσει. Στην τελευταία πίστα ο πρωταγωνιστής έρχεται αντιμέτωπος επιτέλους με τον Θάνατο. Αν καταφέρει να τον νικήσει, θα έχει απελευθερώσει όλες τις ψυχές του κόσμου του και αφού είναι ο τελευταίος άνθρωπος, δεν έχει κάτι άλλο να κάνει και πεθαίνει και αυτός.

3.3 Gameplay - Walkthrough

Ξεκινώντας το παιχνίδι, ο παίκτης μεταφέρεται στο αρχικό μενού. Εκεί μπορεί να δει τα controls, να επιλέξει το επίπεδο δυσκολίας που επιθυμεί και όταν είναι έτοιμος να ξεκινήσει το παιχνίδι. Η δυσκολία σε αυτό το παιχνίδι επηρεάζει τη ζωή των εχθρών. Στη Normal δυσκολία θα έχουν τη προεπιλεγμένη ζωή τους. Στην Easy θα έχουν λιγότερη και στην Hard περισσότερη. Στη συνέχεια μεταφέρεται στο μενού επιλογής πίστας όπου θα δει ότι η μόνη διαθέσιμη πίστα είναι η πρώτη.

Ξεκινώντας την πρώτη πίστα, ο παίκτης θα δει μια πίστα με πρασινάδα και δέντρα. Θα παρατηρήσει κάποια spikes που πρέπει να προσπεράσει διότι αν τα ακουμπήσει θα μειωθεί η ζωή του. Οπότε χρησιμοποιώντας το space bar μπορεί να κάνει άλμα και να τα αποφύγει. Θα έρθει και αντιμέτωπος με τον πρώτο εχθρό τον οποίο μπορεί είτε να αποφύγει ή να σκοτώσει με το σπαθί του και το αριστερό κλικ στο ποντίκι. Αφού περάσει το checkpoint θα δει ένα σπίτι. Μπαίνοντας μέσα θα αντικρίσει δυο σεντούκια από τα οποία μπορεί να διαλέξει μόνο ένα. Το ένα θα του αυξήσει τη ζωή μόνιμα και το άλλο θα μειώσει τη ζημιά που δέχεται από όλους τους εχθρούς. Μόλις βγει από το σπίτι, θα πρέπει να αντιμετωπίσει το πρώτο boss, τον Blue Bandit. Νικώντας τον, πρέπει να προσευχηθεί στο άγαλμα και έτσι επιστρέφει στο μενού επιλογής πίστας όπου και θα δει ότι έχει ξεκλειδωθεί και το δεύτερο επίπεδο.

Προχωρώντας στη δεύτερη πίστα, θα παρατηρήσει ότι το τοπίο είναι παρόμοιο αλλά τώρα έχει πέσει ο ήλιος και δεν έχει τόσο φως. Αφού ξεπεράσει όλα τα εμπόδια, spikes, εχθρούς και τα κενά στο έδαφος, θα φτάσει στο δεύτερο boss στο νεκροταφείο όπου και θα δει τον Haunting King. Πρέπει να το νικήσει και να κάνει ό,τι έκανε και στην πρώτη πίστα και θα ξεκλειδωθεί το τρίτο επίπεδο.

Στην τρίτη πίστα ο παίκτης βρίσκεται σε ένα πιο ορεινό μέρος της περιοχής και το τοπίο έχει αλλάξει. Ο ουρανός είναι μωβ, το έδαφος έχει ξεραθεί και αποτελείται από πέτρες και νεκροκεφαλές και τα κενά που πρέπει να προσπεράσει είναι μεγαλύτερα. Για να προσπεράσει την πίστα θα πρέπει να ξεπεράσει το τρίτο boss, την υπεύθυνη για τα κρανία στο έδαφος, την Skullen Witch. Αν κυριαρχήσει στη μάχη, θα ξεκλειδώσει το επίπεδο τέσσερα.

Στο τέταρτο επίπεδο, ο παίκτης πλησιάζει στο τέλος. Έχει βραδιάσει εντελώς και το μόνο φως είναι από τα αστέρια, τα οποία δίνουν ένα κοκκινωπό χρώμα στο έδαφος και σε όλα τα αντικείμενα της σκηνής. Θα κληθεί να αντιμετωπίσει τη μεγαλύτερή του πρόκληση μέχρι στιγμής σχετικά με τα άλματα που θα χρειαστεί να κάνει για να ολοκληρώσει την πίστα. Πριν το boss όμως, θα έχει μια ευκαιρία να πάρει άλλο ένα upgrade μπαίνοντας στο δεύτερο σπίτι. Μπορεί πάλι να επιλέξει ένα από τα δυο σεντούκια όπου το ένα θα του δώσει επιπλέον ζωή και το άλλο θα τον κάνει να δέχεται λιγότερη ζημιά από τους εχθρούς. Αν καταφέρει ο παίκτης και φτάσει στο τέλος θα έρθει αντιμέτωπος με έναν άγνωστο εχθρό. Έναν που γνωρίζει την κάθε του κίνηση και θα δυσκολέψει αρκετά την πορεία του.

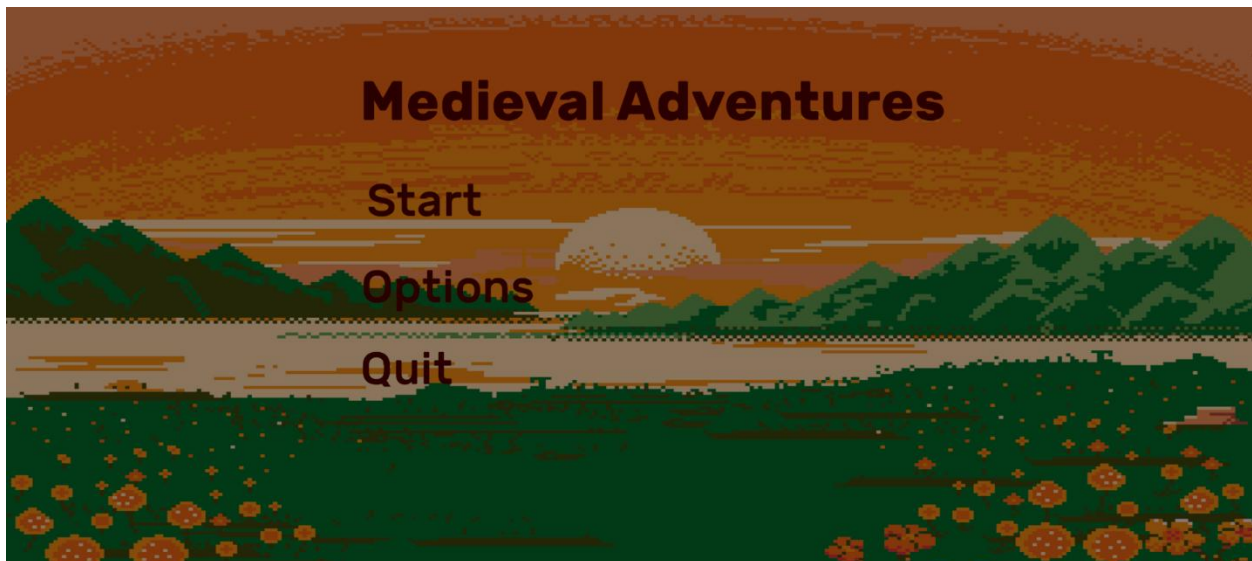
Το πέμπτο επίπεδο είναι το τελευταίο αυτής της περιπέτειας του πρωταγωνιστή. Ο ουρανός φαίνεται πολύ δυσοίωνος όπως και οι εχθροί. Επιπλέον, από το έδαφος προεξέχουν τεράστια σπαθιά που ο παίκτης δεν γνωρίζει γιατί είναι εκεί. Αυτό το επίπεδο είναι το μεγαλύτερο σε έκταση αλλά και το πιο δύσκολο στη μετακίνηση του παίκτη καθώς πρέπει να πραγματοποιήσει άλματα με μεγάλη ακρίβεια αλλιώς θα πέσει στο κενό. Όταν ο παίκτης τα προσπεράσει όλα αυτά, θα έρθει αντιμέτωπος με τον δυσκολότερο εχθρό. Τον ίδιο τον Θάνατο. Αν βγει νικητής από τη μάχη θα έχει

ολοκληρώσει το παιχνίδι και θα μπορεί να διαλέξει οποιοδήποτε επίπεδο από τα πέντε για να παίξει ξανά.

3.4 Σκηνές

Στο Medieval Adventures υπάρχουν εννιά σκηνές. Οι σκηνές StartingMenu που είναι το αρχικό μενού του παιχνιδιού και η LevelSelection όπου ο παίκτης επιλέγει πίστα. Οι πίστες του παιχνιδιού είναι οι Level1, Level2, Level3, Level4, Level5. Οι House, House2 είναι για τα σπίτια όπου ο παίκτης μπορεί να πάρει τα upgrades. Παρακάτω παρουσιάζονται οι σκηνές αναλυτικά.

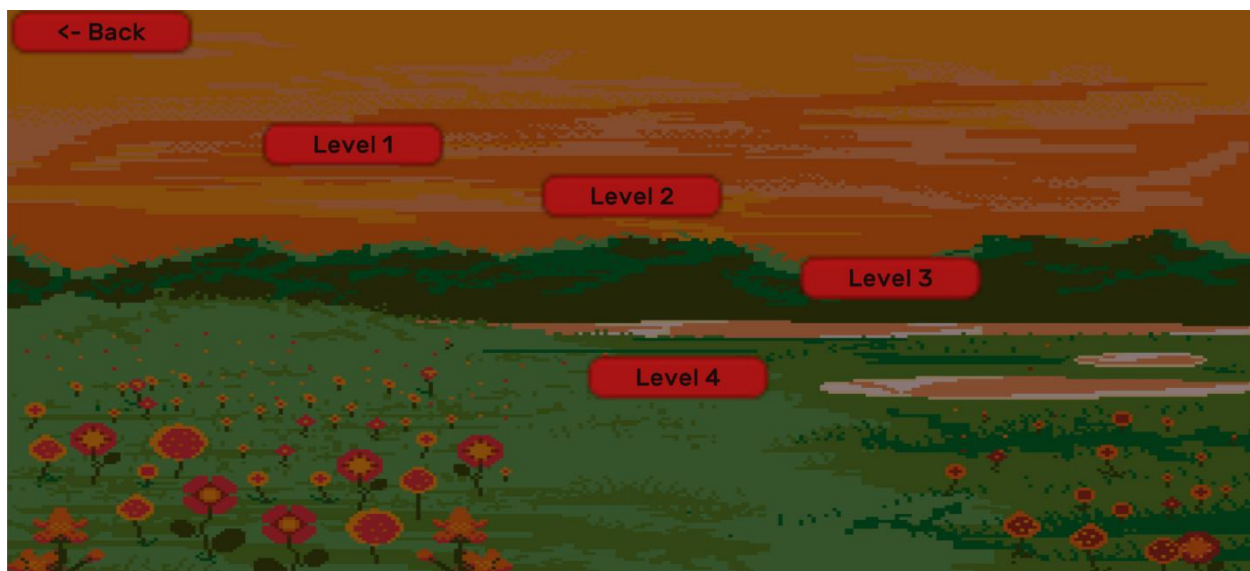
3.4.1 StartingMenu



3.1 Σκηνή Αρχικού Μενού

Αυτή είναι η πρώτη σκηνή κάθε φορά που ανοίγει ο παίκτης το παιχνίδι. Έχει επιλογή να ξεκινήσει το παιχνίδι, να ανοίξει το μενού των ρυθμίσεων και να κλείσει την εφαρμογή.

3.4.2 LevelSelection



3.2 Σκηνή Επιλογής Πίστας

Αυτή η σκηνή εμφανίζει τις πίστες που έχει ξεκλειδώσει και μπορεί να παίξει ο παίκτης. Με το κουμπί «Back» πάνω αριστερά οδηγείται στο αρχικό μενού.

3.4.3 Level1



3.3 Επίπεδο 1

Η σκηνή αυτή είναι το πρώτο επίπεδο του παιχνιδιού. Είναι σχετικά μικρή και απλή για να μπορέσει ο παίκτης να προσαρμοστεί χωρίς ιδιαίτερη δυσκολία. Σε αυτή την πίστα είναι το πρώτο από τα δυο σπίτια όπου ο παίκτης μπορεί να διαλέξει ανάμεσα σε δυο upgrades.

3.4.4 Level2



3.4 Επίπεδο 2

Στην δεύτερη πίστα ο παίκτης θα δυσκολευθεί λίγο περισσότερο να προσπεράσει τα εμπόδια που υπάρχουν. Ένας τρόπος με τον οποίο έχει αυξηθεί η δυσκολία είναι ότι τα άλματα που χρειάζεται να πραγματοποιήσει απαιτούν μεγαλύτερη ακρίβεια.

3.4.5 Level3



3.5 Επίπεδο 3

Αυτή η σκηνή αποτελεί την τρίτη πίστα του παιχνιδιού και είναι και αυτή δυσκολότερη συγκριτικά με τις προηγούμενες της.

3.4.6 Level4



3.6 Επίπεδο 4

Η τέταρτη πίστα είναι μέχρι τώρα η πιο απωθητική για τον παίκτη, τόσο στην πρόκληση όσο και στον σχεδιασμό. Σε αυτή την πίστα βρίσκεται το δεύτερο και τελευταίο σπίτι που ο παίκτης μπορεί να επιλέξει ανάμεσα σε δυο upgrades.

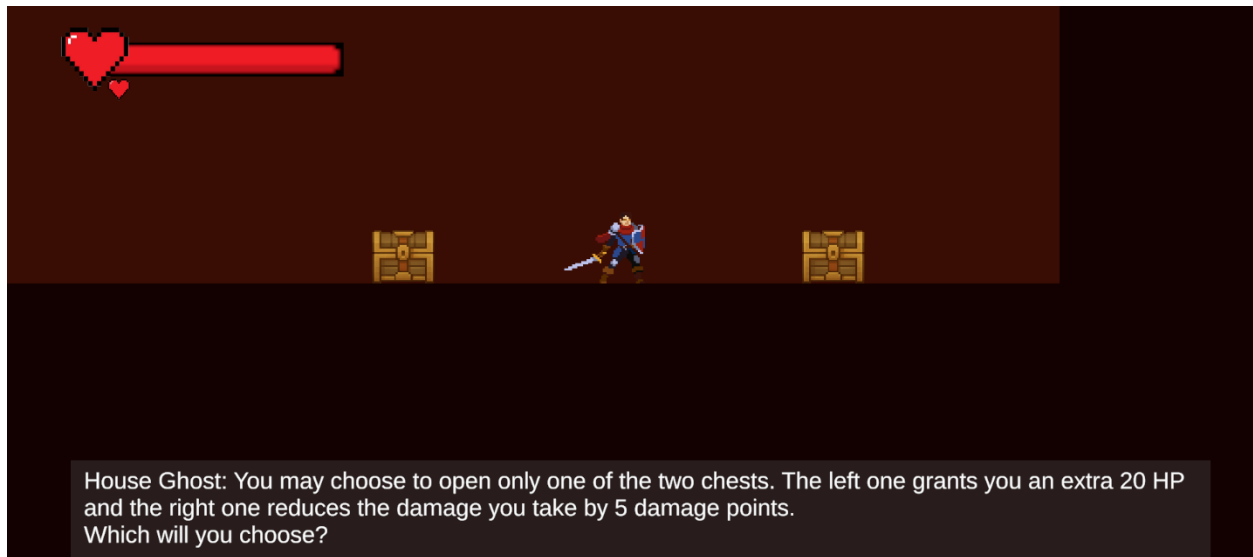
3.4.7 Level5



3.7 Επίπεδο 5

Η πέμπτη και τελευταία πίστα είναι η μεγαλύτερη σε έκταση και η πιο δύσκολη. Έχει περισσότερους εχθρούς, δύσκολο platforming και το πιο ισχυρό final boss στο παιχνίδι. Είναι επίσης η πιο σκοτεινή σε σχεδιασμό, μαζί με το Επίπεδο 4.

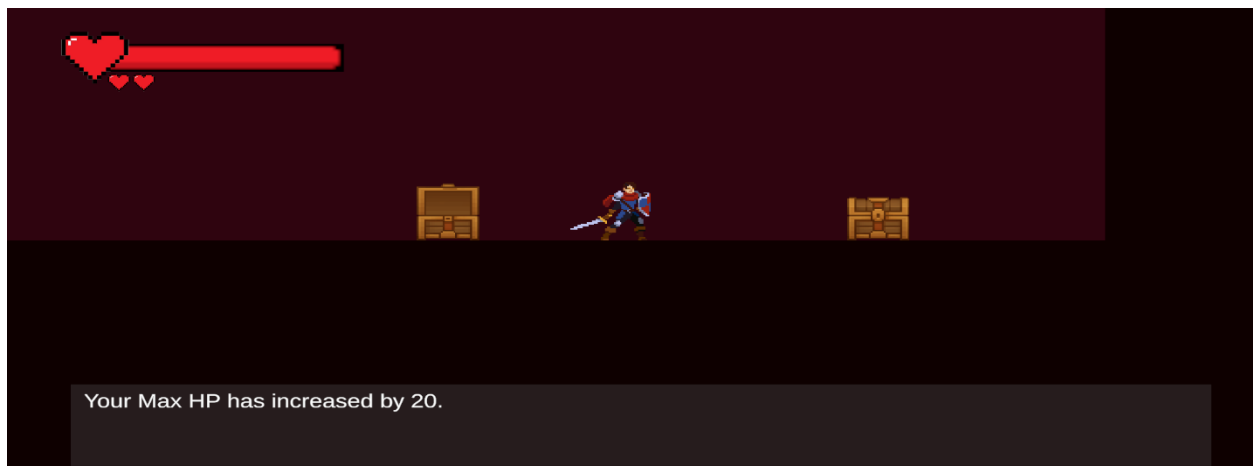
3.4.8 House



3.8 Σπίτι

Το πρώτο από τα δυο σπίτια που μπορεί να πάρει το πρώτο του upgrade ο παίκτης, βρίσκεται στο Επίπεδο 1. Μπορεί να διαλέξει μόνο ένα από τα δυο σεντούκια να ανοίξει και μόλις βγει από το σπίτι δεν μπορεί να επιστρέψει.

3.4.9 House2



3.9 Σπίτι 2

Το δεύτερο σπίτι βρίσκεται στο Επίπεδο 4. Όπως και στο πρώτο σπίτι, ο παίκτης μπορεί να επιλέξει μόνο ένα από τα δυο σεντούκια και αν βγει δεν μπορεί να μπει μέσα ξανά.

3.5 UI

3.5.1 Βασικό UI

Το User Interface είναι ένα από τα σημαντικότερα τμήματα ενός παιχνιδιού. Είναι ο τρόπος με τον οποίον ο παίκτης αλληλεπιδράει με το παιχνίδι και μπορεί να ενημερωθεί μέσω αυτού. Εκτός από τα τέσσερα μενού που παρουσιάστηκαν παραπάνω, το UI έχει και άλλα στοιχεία. Το σημαντικότερο για τον παίκτη είναι η μπάρα ζωής που βρίσκεται στην πάνω αριστερή γωνία της οθόνης. Εκεί φαίνεται η ζωή του παίκτη καθώς και πόσες ζωές του απομένουν.



3.10 Γεμάτη μπάρα με 2 ζωές



3.11 Μπάρα αφού έχει μειωθεί η ζωή του παίκτη και μόνο με 1 ζωή

Εκτός από την ζωή του παίκτη χρησιμοποιώντας μπάρα φαίνεται και η ζωή του final boss σε κάθε επίπεδο. Επιπροσθέτως, εμφανίζεται ανάλογο κείμενο όταν ο παίκτης θέλει να μπει και να βγει στο σπίτι, όταν θέλει να ανοίξει κάποιο σεντούκι και όταν νικήσει το boss και θέλει να αλλάξει επίπεδο.



3.12 Μπάρα ζωής του Final Boss



3.13 Είσοδος στο σπίτι



3.14 Έξοδος από το σπίτι

House Ghost: You may choose to open only one of the two chests. The left one grants you an extra 20 HP and the right one reduces the damage you take by 5 damage points. Which will you choose?

3.15 Φωνή του φαντάσματος στο σπίτι



3.15 Αλλαγή επιπέδου στο άγαλμα

3.5.2 Αρχικό Μενού



3.17 Αρχικό μενού

Στην εικόνα 3.17 φαίνεται το πρώτο πράγμα που βλέπει ο παίκτης μόλις ξεκινήσει το παιχνίδι. Είναι ένα απλό μενού με μια εικόνα από πίσω, κάποιες επιλογές και μουσική. Πατώντας το Start ξεκινάει το παιχνίδι, αν ο παίκτης δεν έχει επιλέξει ήδη δυσκολία θα του εμφανιστεί αυτόματα το μενού επιλογής δυσκολίας. Στη συνέχεια θα μεταβιβαστεί στο μενού επιλογής πίστας. Πατώντας το Options ανοίγει το μενού με τις ρυθμίσεις. Τέλος, πατώντας το Quit κλείνει την εφαρμογή.



3.18 Μενού ρυθμίσεων

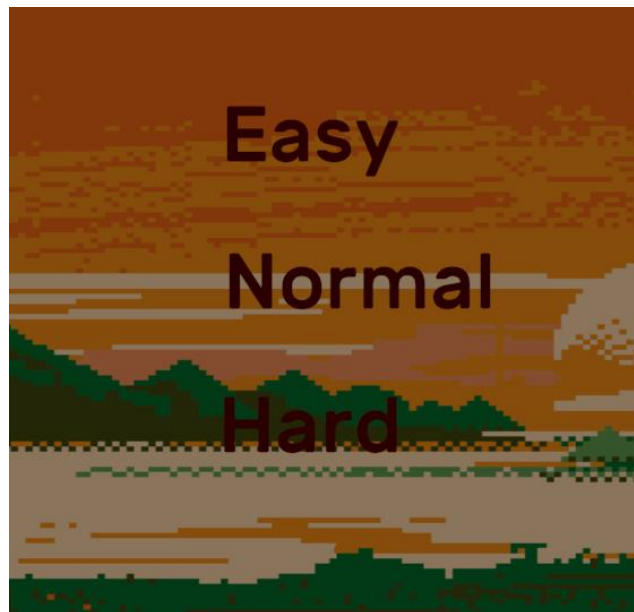
Στην εικόνα 3.18 φαίνεται το μενού ρυθμίσεων. Αναλυτικά:

- Controls: επιλέγοντας αυτό θα εμφανιστούν τα controls με τα οποία ο παίκτης ελέγχει τον χαρακτήρα
- Difficulty: επιλέγοντας αυτό θα εμφανισθούν τρεις επιλογές δυσκολίας: Easy, Medium, Hard

- Progress: επιλέγοντας αυτό ο παίκτης μπορεί αν το επιθυμεί, να διαγράψει την πρόοδο που έχει κάνει στο παιχνίδι



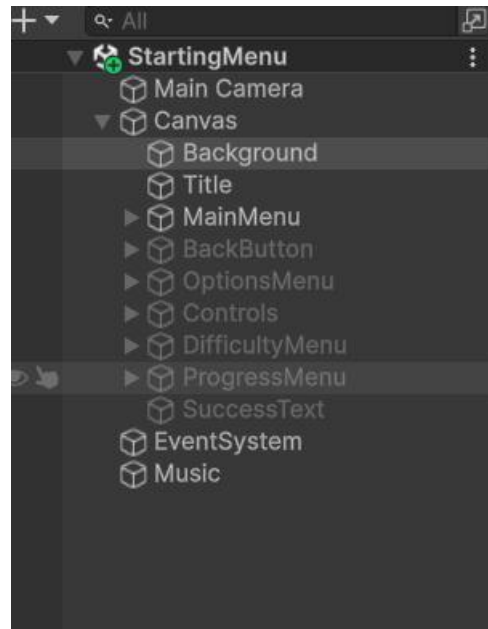
3.19 Controls



3.20 Difficulty



3.21 Progress



3.22 StartingMenu στο Editor

Στην εικόνα 3.22 φαίνονται τα στοιχεία από τα οποία αποτελείται η σκηνή του αρχικού μενού. Την κάμερα που απεικονίζει ουσιαστικά τι φαίνεται στην οθόνη, το Canvas που περιέχει όλα τα UI στοιχεία, το EventSystem που χειρίζεται όλες τις ενέργειες του UI και ένα Audio Source (Music).

Πέρα από την εικόνα και το απλό κείμενο, τα κουμπιά αποτελούνται από το κείμενο τους, το Button element που μπορεί να πατήσει ο παίκτης και από κομμάτια κώδικα που θα παρουσιαστούν στη συνέχεια. Για κάθε αλλαγή που μπορεί να κάνει ο παίκτης, π.χ. να αλλάξει τη δυσκολία, εμφανίζεται κείμενο στον παίκτη που του επιβεβαιώνει ότι η ενέργειά του ολοκληρώθηκε επιτυχώς.

3.5.3 Μενού Επιλογής Πίστας



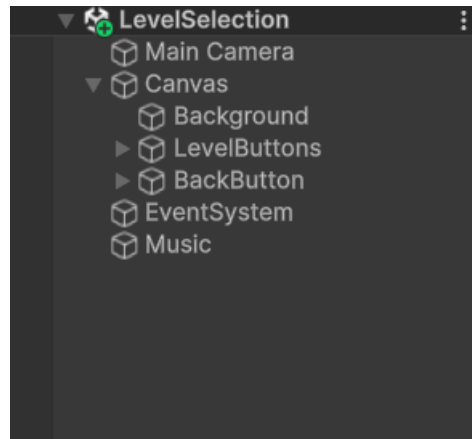
3.23 Μενού επιλογής πίστας

Στην εικόνα 3.23 φαίνεται το μενού επιλογής πίστας, αυτό εμφανίζεται στον παίκτη αν πατήσει Start στο αρχικό μενού και όταν νικήσει κάποια πίστα. Με το κουμπί Back πηγαίνει στο αρχικό μενού και με το κουμπί του επιπέδου πηγαίνει στο αντίστοιχο επίπεδο. Υπάρχουν σύνολο πέντε επίπεδα. Το κάθε επίπεδο (εκτός του επιπέδου 1) ξεκλειδώνεται μόνο όταν ο παίκτης ολοκληρώσει με επιτυχία το προηγούμενο επίπεδο.



3.24 Μενού επιλογής πίστας (2)

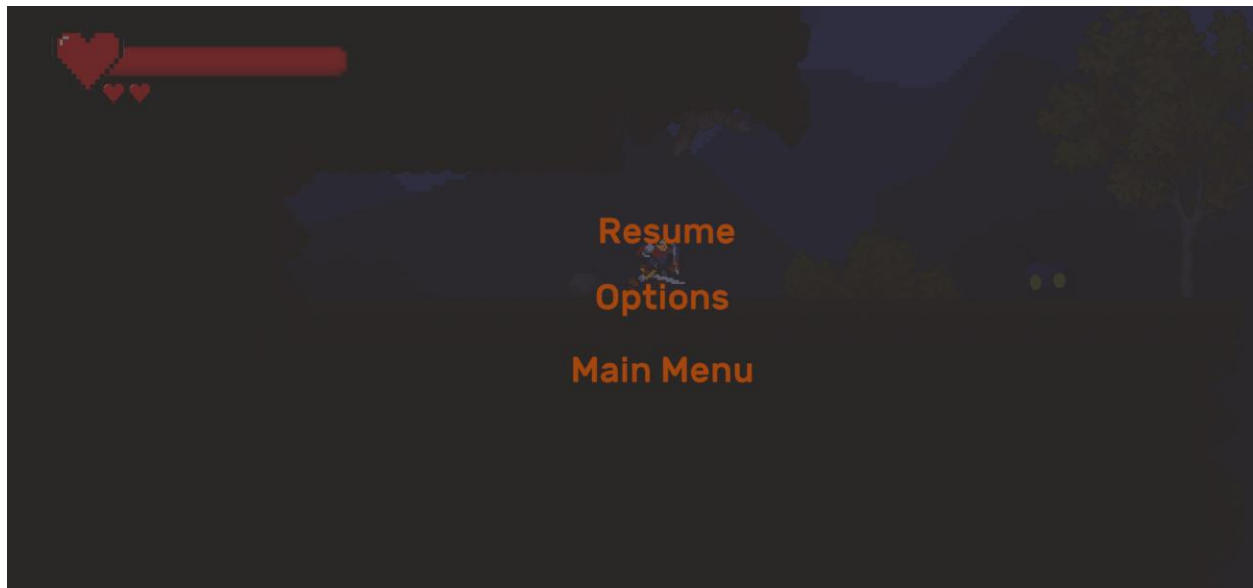
Για παράδειγμα, στην εικόνα 3.24 φαίνεται το μενού όταν έχει ξεκλειδωθεί και η δεύτερη πίστα αφού ο παίκτης έχει νικήσει την πρώτη.



3.25 LevelSelection στο Editor

Στην εικόνα 3.25 φαίνεται η δομή της σκηνής στο editor η οποία ακολουθεί την ίδια λογική με αυτή που χρησιμοποιήθηκε για την υλοποίηση του αρχικού μενού.

3.5.4 Μενού Παύσης



3.26 Μενού παύσης

Στην εικόνα 3.26 φαίνεται το μενού για την παύση. Όταν ο παίκτης είναι σε μια πίστα πατώντας το πλήκτρο Escape (Esc) στο πληκτρολόγιο, κάνει παύση το παιχνίδι και εμφανίζεται αυτό το

μενού. Παγώνει τον χρόνο στο παιχνίδι ακινητοποιώντας τα πάντα, σταματώντας τη μουσική και βάζει και ένα γκριζο φίλτρο στην οθόνη πίσω από τα κουμπιά. Οι επιλογές στο μενού είναι:

- Resume: συνεχίζει το παιχνίδι
- Options: εμφανίζει επιλογή να δει ο παίκτης τα controls
- Main Menu: οδηγείται στο αρχικό μενού, βγαίνοντας από την πίστα

Ακόμη, όταν ο παίκτης πατήσει Esc και ανοίξει το μενού παύσης, αν το ξαναπατήσει το παιχνίδι θα συνεχιστεί. Είναι δηλαδή σαν να πατάει το κουμπί Resume.

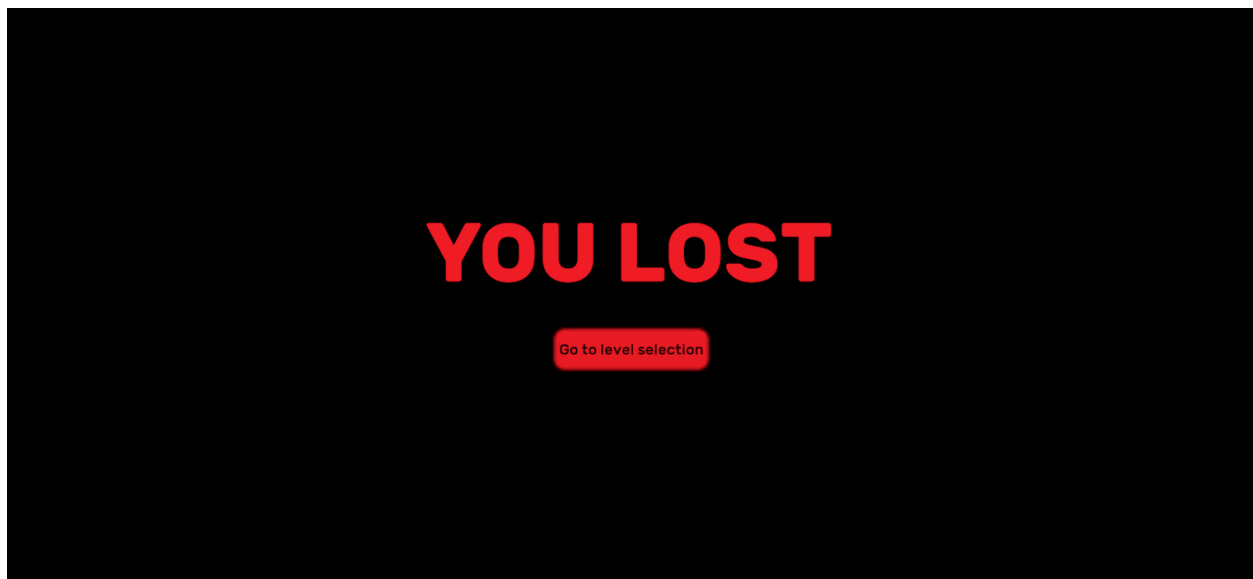


3.27 Controls



3.28 Controls (2)

3.5.5 Μενού Ήττας



3.29 Μενού Ήττας

Αυτό το μενού εμφανίζεται όταν σε μια πίστα ο παίκτης χάσει και τις δυο του ζωές. Η οθόνη γίνεται μαύρη και εμφανίζεται κείμενο που αναγράφει: YOU LOST (Έχασες), και ένα κουμπί που οδηγεί τον παίκτη στο μενού επιλογής πίστας.

4. Ανάλυση και Αρχιτεκτονική

4.1 Βασικά συστήματα

4.1.1 Παίκτης

Σε κάθε παιχνίδι το σημαντικότερο μέρος του είναι ο παίκτης. Να κινείται ομαλά στον χώρο χωρίς να επηρεάζεται αρνητικά η εμπειρία του χρήστη. Ο παίκτης είναι το μέσο με το οποίο ο χρήστης θα εξερευνήσει τον κόσμο άρα πρέπει να λειτουργεί όσο καλύτερα γίνεται. Για αυτό πριν γίνει οτιδήποτε άλλο έβαλα στην σκηνή τον παίκτη και ξεκίνησα να κάνω αλλαγές για να πετύχω το επιθυμητό αποτέλεσμα. Ήθελα ο παίκτης να μπορεί να κινηθεί δεξιά και αριστερά στον χώρο και να έχει την δυνατότητα να κάνει άλμα. Επιπλέον, να μπορεί να επιτίθεται στους εχθρούς με κάποιο όπλο. Ψάχνοντας στο Unity Asset Store βρήκα ακριβώς αυτό που έψαχνα, το asset: HeroKnight.



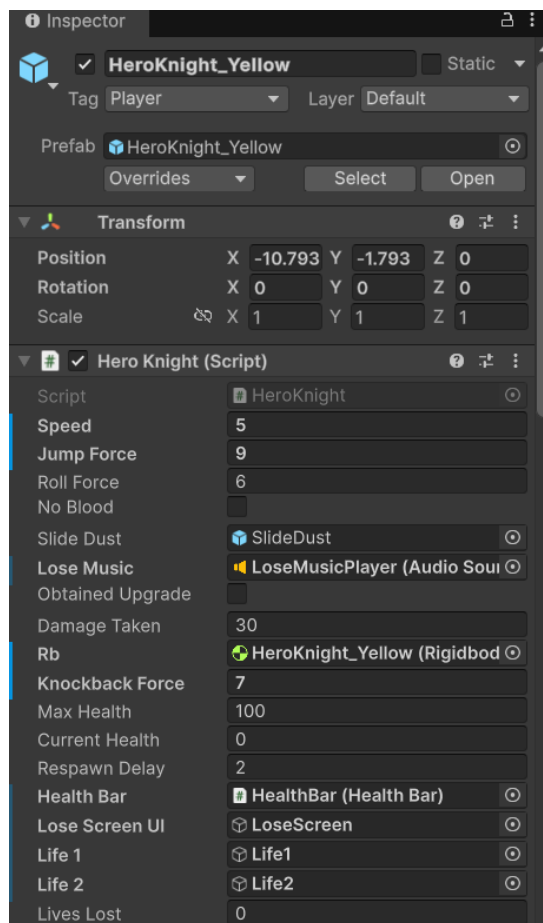
4.1 Ο παίκτης HeroKnight

Το asset που επιλέχθηκε είχε ήδη ολοκληρωμένα sprites, colliders, animations και κώδικα κίνησης, ο οποίος τροποποιήθηκε για να ικανοποιηθούν οι ανάγκες του παιχνιδιού. Είχε επίσης ασπίδα η οποία ενσωματώθηκε και αυτή στο παιχνίδι.



4.2 Τα υποτμήματα του παίκτη

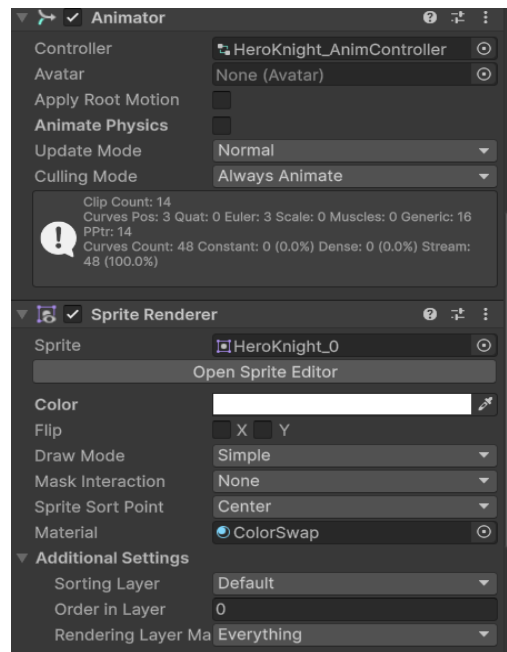
Όπως φαίνεται στην παραπάνω εικόνα «μέσα» στον παίκτη υπάρχει ένα SwordHitbox και ShieldHitbox τα οποία είναι colliders και ενεργοποιούνται όταν ο παίκτης κάνει επίθεση με το σπαθί και όταν σηκώνει την ασπίδα αντίστοιχα. Επίσης υπάρχουν αισθητήρες για το έδαφος, για να εντοπίζει αν ο παίκτης είναι πάνω στο έδαφος ή αν πέφτει, και για τους τοίχους για να ενεργοποιείται το wall slide animation.



4.3 Το Inspector για τον παίκτη: script

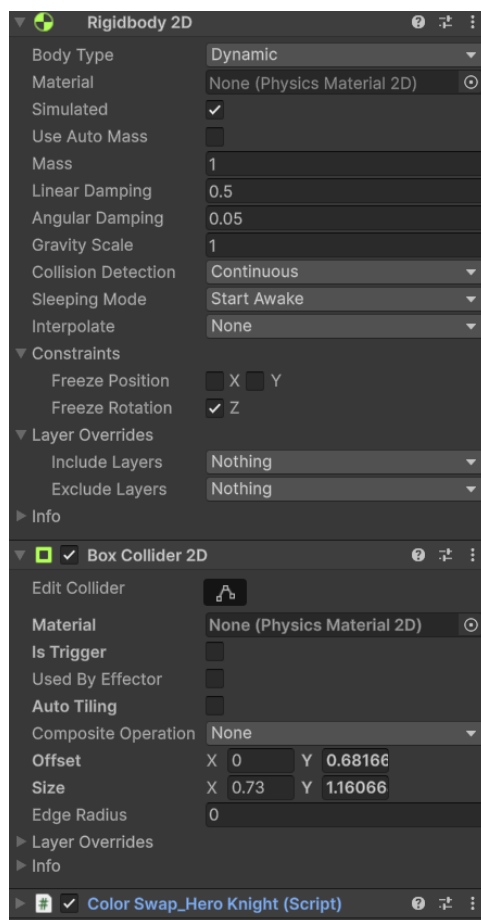
Στην εικόνα 4.3 είναι το inspector του παίκτη και φαίνεται το script του που θα αναλυθεί παρακάτω. Φαίνεται επίσης ότι ο παίκτης έχει tag: Player που χρησιμοποιείται κυρίως για να ελέγχεται η είσοδός του σε colliders. Στο inspector παρατηρούμε τα διάφορα πεδία του script που μπορεί να αλλάξει ο χρήστης μέσω του Unity. Συγκεκριμένα υπάρχουν πεδία:

- Για την ταχύτητα με την οποία κινείται ο παίκτης
- Για τη δύναμη του άλματος
- Για τη δύναμη του roll
- Checkbox σε περίπτωση που δεν επιθυμούμε να φαίνεται αίμα
- Particle effect για το χώμα
- Μουσική για όταν χάνει ο παίκτης
- Checkbox για το αν έχει πάρει ήδη upgrade
- Πόσο damage παίρνει από τους εχθρούς
- Το rigid body 2d
- Πόσο μακριά πετάγεται όταν δέχεται damage
- Η μέγιστη ζωή του
- Η τρέχουσα ζωή του η οποία γίνεται όσο είναι η μέγιστη μόλις ξεκινήσει η πίστα
- Η καθυστέρηση για το respawn
- Το script για τη μπάρα ζωής
- Το UI για την οθόνη ήττας και για την κάθε ζωή ξεχωριστά
- Counter με πόσες ζωές έχει χάσει



4.4 Το inspector για τον παίκτη: animator, sprite renderer

Στην εικόνα 4.4 βλέπουμε στο inspector το animator, που χειρίζεται την αλλαγή των sprite για κάθε κίνηση (π.χ. άλμα) και το sprite renderer που εμφανίζει και ελέγχει τα sprite (π.χ. αλλαγή χρώματος του sprite).



4.5 Το inspector για τον παίκτη: Rigidbody2D, Box Collider 2D, ColorSwap

Στην εικόνα 4.5 φαίνεται στο inspector του παίκτη το Rigidbody2D, που δίνει υπόσταση στον παίκτη, τον μετατρέπει ουσιαστικά σε κάτι με βαρύτητα και όχι απλά μια εικόνα. Η περιστροφή στον άξονα Z έχει επιλεγεί να είναι «παγωμένη» γιατί υπήρχε ένα πρόβλημα όταν ο εχθρός περπατούσε προς τον παίκτη και ο παίκτης προς τον εχθρό και συγκρούονταν, μπορεί να έπεφτε ο παίκτης. Επιπλέον, υπάρχει το Box Collider 2D το οποίο χειρίζεται τις συγκρούσεις του παίκτη στο παιχνίδι και συνδυαστικά με το Rigidbody2D του δίνουν μια ολοκληρωμένη φυσική υπόσταση. Τέλος, υπάρχει script για αλλαγή χρώματος του παίκτη αν υπάρχει αυτή η επιθυμία.

Για τον παίκτη χρησιμοποιήθηκε επίσης το DontDestroyOnLoad (θα αναλυθεί παρακάτω) για να είναι ο ίδιος παίκτης κατά τη διάρκεια του παιχνιδιού. Να συγκρατεί δηλαδή upgrades, ζωή και ότι άλλο είναι απαραίτητο. Για να ακολουθεί η κάμερα τον παίκτη στον χώρο και να αλλάζει κατεύθυνση με εκείνον στον κέντρο χρησιμοποιήθηκε Cinemachine Brain και Cinemachine Camera που είναι έτοιμα εργαλεία που παρέχει το Unity.

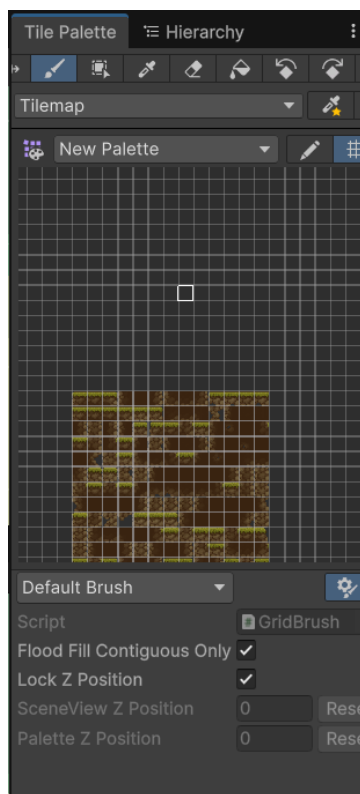
4.1.2 Έδαφος – Δομή πίστας

Το έδαφος σε κάθε πίστα είναι ένα Tilemap σε ένα grid. Είναι κλασσικός τρόπος δημιουργίας επιπέδου σε τέτοιου είδους παιχνίδια. Χωρίζει ουσιαστικά το παιχνίδι σε τετραγωνάκια στα οποία μπορούν να τοποθετηθούν tiles κατά την δημιουργία της πίστας.



4.6 Grid & Tilemap στο Editor

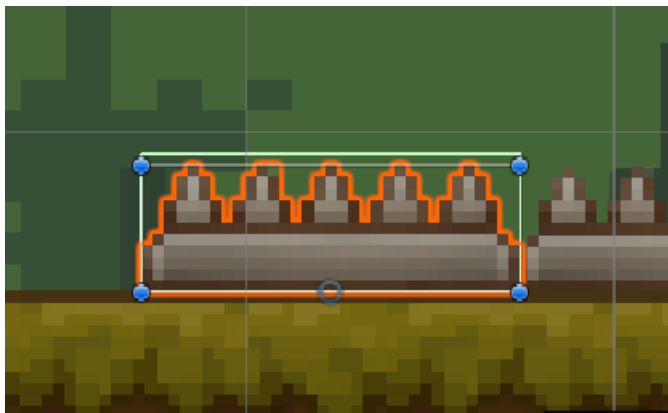
Έχουν χρησιμοποιηθεί έτοιμα assets για τα tiles από το Unity Asset Store και έχουν τοποθετηθεί με τέτοιο τρόπο ώστε ο παίκτης να χρειάζεται να κάνει και άλματα που στη συνέχεια γίνονται όλο και δυσκολότερα.



4.7 Tile Palette

Στην εικόνα 4.7 βλέπουμε το παράθυρο που χρησιμοποιείται για την δημιουργία του tilemap. Ο δημιουργός μπορεί να επιλέξει όποιο tile επιθυμεί και στην συνέχεια να «βάψει» την πίστα όπως θέλει. Το tilemap έχει και Colliders και Rigidbody για να λειτουργεί σωστά και να μπορούν υπόλοιπα αντικείμενα όπως ο παίκτης να στέκονται πάνω του χωρίς να περνάνε μέσα του και να πέφτουν.

Στην πίστα ως εμπόδια, πέρα από τα κενά που πρέπει να πηδήξει ο παίκτης, υπάρχουν και spikes (καρφιά) ως παγίδες. Αν τα ακουμπήσει ο παίκτης θα χάσει ζωή.



4.8 Spikes

Επιπλέον, σε κάθε πίστα υπάρχουν fall detectors. Δηλαδή αόρατα Colliders που ενεργοποιούνται αν ο παίκτης πέσει από την πίστα. Όταν φτάσει στο collider χάνει μια ζωή και κάνει respawn πίσω στην πίστα.

Γενικά η δομή που ακολουθεί η κάθε πίστα είναι η εξής. Ο παίκτης ξεκινάει από τα αριστερά και πάει προς τα δεξιά. Στην πορεία θα πρέπει να προσπεράσει εχθρούς και παγίδες στην πίστα. Το checkpoint βρίσκεται περίπου στη μέση κάθε πίστας. Σε δυο από τις πέντε πίστες υπάρχει σπίτι στο οποίο μπορεί να μπει για να διαλέξει upgrade. Υπάρχουν μερικά chest σε κάθε πίστα που του ανανεώνουν ένα ποσό της ζωής του. Τέλος, κάθε πίστα έχει ένα final boss το οποίο ο παίκτης πρέπει να νικήσει για να ξεκλειδώσει το επόμενο επίπεδο.

4.1.3 Checkpoint

Σε κάθε πίστα υπάρχει ένα checkpoint περίπου στη μέση. Όταν ο παίκτης φτάσει το checkpoint, γεμίζει η ζωή του, γίνεται πράσινος για ένα πολύ μικρό χρονικό διάστημα και παίζει ένα sound effect. Αν χάσει μια ζωή από αυτό το σημείο και μετά, κάνει respawn εκεί και όχι στην αρχή της πίστας.

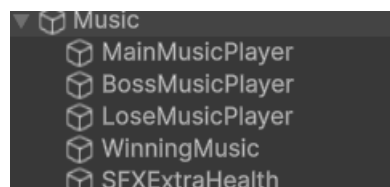


4.9 To Checkpoint Lamp

Όπως φαίνεται στην εικόνα 4.9 το checkpoint έχει την μορφή μιας λάμπας. Μόλις ο παίκτης φτάσει τη λάμπα, θα ανάψει. Εκτός από το sprite, η λάμπα αποτελείται και από effects που ενεργοποιούνται μόλις ο παίκτης τη φτάσει, από Collider και από script που διαχειρίζεται όλες τις λειτουργίες της.

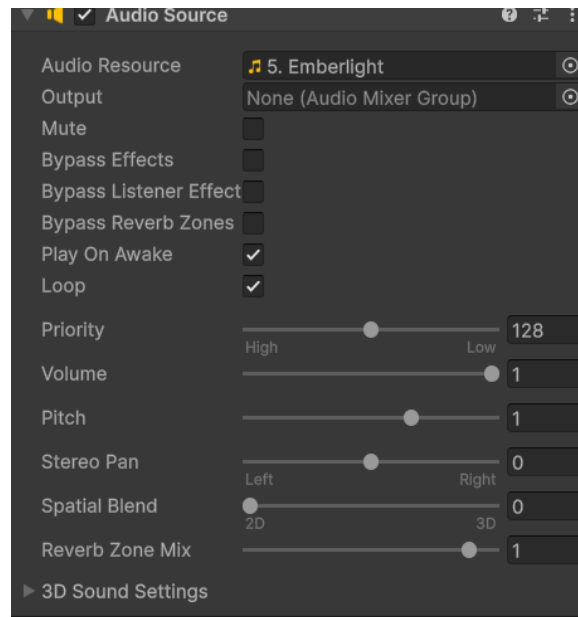
4.1.4 Μουσική – Ήχοι

Η μουσική και οι ήχοι αναπαράγονται από Audio Sources. Κάθε πίστα έχει δική της ξεχωριστή μουσική. Επίσης υπάρχει μουσική για όταν χάνει ο παίκτης, για όταν νικήσει μια πίστα, ξεχωριστή μουσική για κάθε Boss Fight και ηχητικά εφέ για όταν παίρνει ζωή και όταν ανοίγει chests (σεντούκια).



4.10 Μουσικές επιπέδων

Για παράδειγμα, στην εικόνα 4.10 φαίνονται οι μουσικές για τα πέντε επίπεδα. Ελέγχεται πότε θα αρχίσουν να παίζουν ή πότε θα σταματήσουν μέσω κώδικα. Στην εικόνα 4.11 αναπαρίσταται το Audio Source στο Editor και οι διάφορες ρυθμίσεις του.

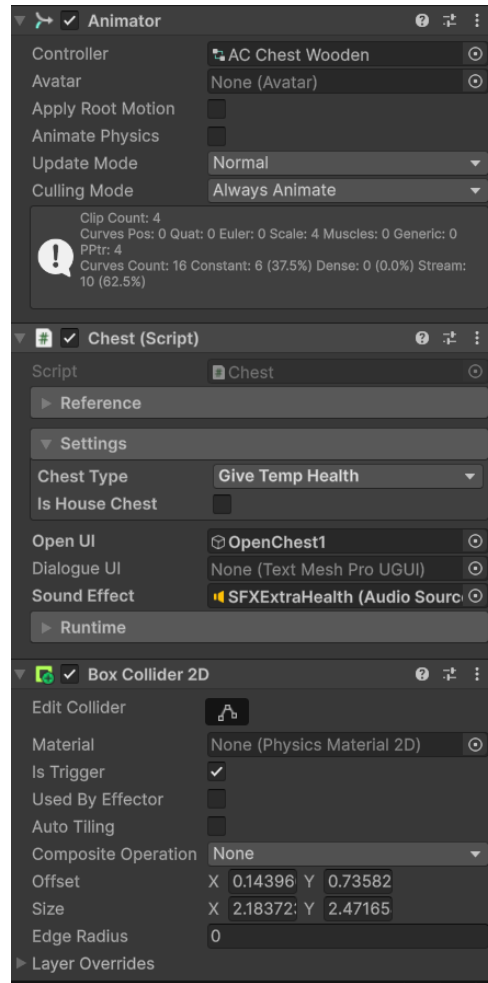


4.11 Ρυθμίσεις Audio Source

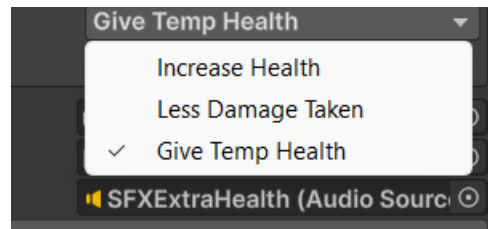
4.1.5 Chests – Upgrades

Στο παιχνίδι υπάρχουν δυο είδη chest: αυτά που είναι εντός και αυτά που είναι εκτός σπιτιού. Αυτά που είναι έξω από το σπίτι, δηλαδή σκόρπια στην πίστα, δίνουν απλά στον παίκτη προσωρινή ζωή αν έχει χάσει ένα ποσό. Σε κάθε σπίτι υπάρχουν δυο chests: το ένα δίνει upgrade στον παίκτη αυξάνοντας τη μέγιστη ζωή του, και το άλλο μειώνει τη ζωή που χάνει όταν δέχεται χτύπημα από εχθρούς ή από παγίδες.

Όταν ανοίγει ο παίκτης το chest παίζει το animation του και ένα sound effect και ο παίκτης προσωρινά αλλάζει χρώμα ως ένδειξη στον χρήστη ότι πήρε το «δώρο».



4.12 Τα τμήματα του Chest στο Editor

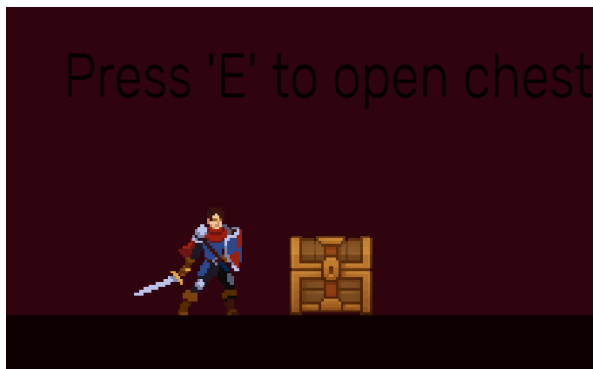


4.13 Επιλογές είδους Chest

Όπως φαίνεται στις εικόνες 4.12, 4.13 το chest πέρα από Collider και Animator, έχει και κώδικα με τον οποίο μπορεί να επιλεχθεί αν είναι chest σπιτιού, τι δίνει στον παίκτη, να επιλεχθεί ο ήχος και UI.



4.14 Κλειστό Chest στο σπίτι



4.15 Κλειστό Chest με τον παίκτη πιο κοντά



4.16 Ανοικτό Chest στο σπίτι

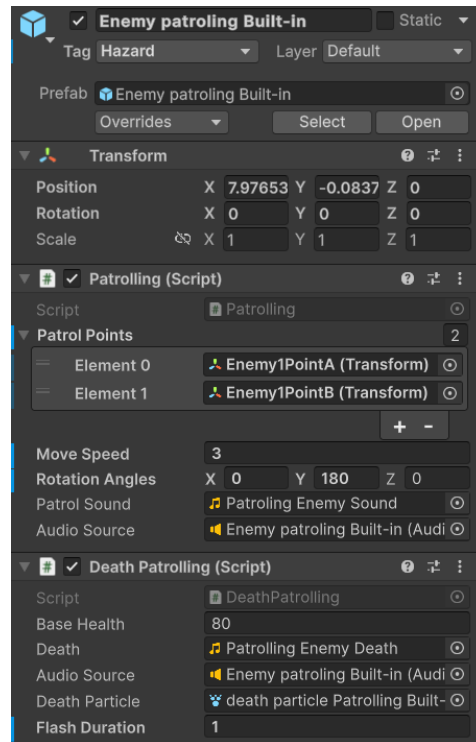


4.17 Κλειστό Chest στην πίστα

Στις εικόνες 4.14, 4.16 φαίνεται το chest πριν και αφού το ανοίξει ο παίκτης, στις εικόνες 4.15, 4.17 φαίνεται το prompt που εμφανίζεται όταν ο παίκτης πλησιάσει το chest.

4.1.6 Εχθροί

Σε κάθε επίπεδο ο παίκτης θα έχει να αντιμετωπήσει κάποιους εχθρούς. Μπορεί να επιλέξει να τους αγνοήσει ή να τους επιτεθεί. Όπως φαίνεται στην εικόνα 4.18, εκτός από τα colliders και άλλα στοιχεία από τα οποία αποτελείται ο εχθρός, φαίνονται τα δυο script του καθώς και το tag του, που είναι Hazard, το ίδιο με τα spikes. Οι εχθροί δεν έχουν σκοπό να επιτεθούν στον παίκτη. Απλά μετακινούνται από ένα σημείο στο άλλο και ο παίκτης χάνει ζωή αν τους ακουμπήσει. Στο editor ο χρήστης μπορεί να θέσει τα σημεία στα οποία θα κινείται ο εχθρός, την ταχύτητά του και τον ήχο που κάνει όταν πεθαίνει. Επιπλέον, την αρχική ζωή του και τη διάρκεια των εφέ για όταν πεθαίνει. Τα σημεία των εχθρών είναι απλά empty objects μέσα στην πίστα.



4.18 Enemy στο Editor

Όταν ο εχθρός δέχεται χτύπημα από τον παίκτη θα γίνει προσωρινά κόκκινος ως ένδειξη ότι ο παίκτης πράγματι τον πέτυχε. Όταν καταφέρει να τον σκοτώσει θα παίξει ένα ηχητικό εφέ και θα εμφανισθούν και particles όπως φαίνεται στην εικόνα 4.20.



4.19 Απλός εχθρός



4.20 Τα particles αφού πεθάνει ο εχθρός

4.1.7 Final Bosses

Υπάρχουν πέντε final bosses, ένα για κάθε επίπεδο. Μόλις φτάσει ο παίκτης στην «αρένα», εμφανίζεται στο UI το όνομα του boss και η μπάρα ζωής του και αλλάζει και η μουσική. Δεν μπορεί να βγει από εκεί μέχρι να νικήσει το boss ή να χάσει. Αν νικήσει, ενεργοποιείται η νικητήρια

μουσική και ο παίκτης πάει και «προσεύχεται» σε ένα άγαλμα και έτσι μεταβιβάζεται στο μενού επιλογής πίστας όπου θα δει ότι έχει ξεκλειδώσει το επόμενο επίπεδο.



4.21 Άγαλμα

Σκοπός των bosses είναι να καταφέρουν να σκοτώσουν τον παίκτη. Κυνηγάνε τον παίκτη μέχρι να φθάσουν όσο κοντά του χρειάζεται για να τον χτυπήσουν. Όπως φαίνεται στην εικόνα 4.22 στο editor ο χρήστης μπορεί να προσαρμόσει την ταχύτητα του boss και ποιον στόχο θα κυνηγάει (στην περίπτωση αυτή, τον παίκτη). Ακόμη, το εύρος εντοπισμού και το εύρος επίθεσης, την μέγιστη ζωή του, και την τρέχουσα η οποία γίνεται όσο η μέγιστη μόλις ξεκινήσει η πίστα.



4.22 Boss: Script & Animator στο Editor



4.23 Level 1 Boss



4.24 Level 2 Boss



4.25 Level 3 Boss



4.26 Level 4 Boss



4.27 Level 5 Boss

4.2 Ροή Δεδομένων

Η ροή δεδομένων είναι ίσως το πιο σημαντικό κομμάτι ενός παιχνιδιού, τουλάχιστον σε λειτουργικό επίπεδο. Με τον όρο ροή δεδομένων εννοούμε το «μονοπάτι» που ακολουθούν τα δεδομένα για να εμφανιστεί στο παιχνίδι το αποτέλεσμα μιας ενέργειας. Για την διευκόλυνση της παρουσίασης της ροής δεδομένων, θα χρησιμοποιηθεί ως παράδειγμα η επίθεση του παίκτη. Δηλαδή, πώς από το πάτημα ενός πλήκτρου από τον χρήστη, βλέπει το αποτέλεσμα της πράξης του στο UI του παιχνιδιού.

Αρχικά, για να επιτεθεί ο παίκτης, ο χρήστης πατάει το αριστερό κλικ του ποντικιού. Στη συνέχεια μέσω του script του παίκτη ελέγχεται αν ο χρήστης πάτησε αυτό το πλήκτρο. Αφού πατήθηκε, στέλνει σήμα στο animator και παίζει το animation της επίθεσης. Σε ένα frame του animation ενεργοποιείται το collider του σπαθιού, το SwordHitbox το οποίο έχει το tag: Weapon. Το collider του όπλου δεν είναι μονίμως ενεργοποιημένο για να αποφευχθεί η ακούσια ζημιά στον εχθρό. Δηλαδή, να δεχθεί ζημιά από τον παίκτη χωρίς αυτός να έχει δώσει εντολή για επίθεση. Αν το hitbox του όπλου «μπει» στο hitbox του εχθρού, μέσω του κώδικα των εχθρών ελέγχεται αν το αντικείμενο που «μπήκε» έχει το tag: Weapon. Αν όχι, δεν γίνεται τίποτα, αν ναι, μειώνεται η ζωή του εχθρού. Για την μείωση της ζωής του εχθρού και άρα για το πως θα εμφανησθεί στο UI για να καταλάβει ο παίκτης ότι όντως χτύπησε τον εχθρό, υπάρχουν δυο περιπτώσεις. Η μια είναι ο εχθρός να είναι ένας από τους απλούς εχθρούς στην πίστα. Τότε μόλις δεχθεί το χτύπημα γίνεται προσωρινά κόκκινος και έτσι ο παίκτης επιβεβαιώνει ότι τον πέτυχε. Αν με το χτύπημα του

τελειώσει η ζωή, παίζει το animation του θανάτου του και ένα ηχητικό εφέ. Η άλλη περίπτωση είναι όταν εχθρός είναι ένα από τα πέντε bosses του παιχνιδιού. Σε αυτή την περίπτωση μόλις δεχθεί ζημιά θα μειωθεί η μπάρα ζωής του στο UI. Αν το χτύπημα που δεχθεί του μειώσει εντελώς την ζωή τότε παίζει το animation του θανάτου του, νικητήρια μουσική και εξαφανίζεται η μπάρα ζωής του.

4.3 UML διαγράμματα

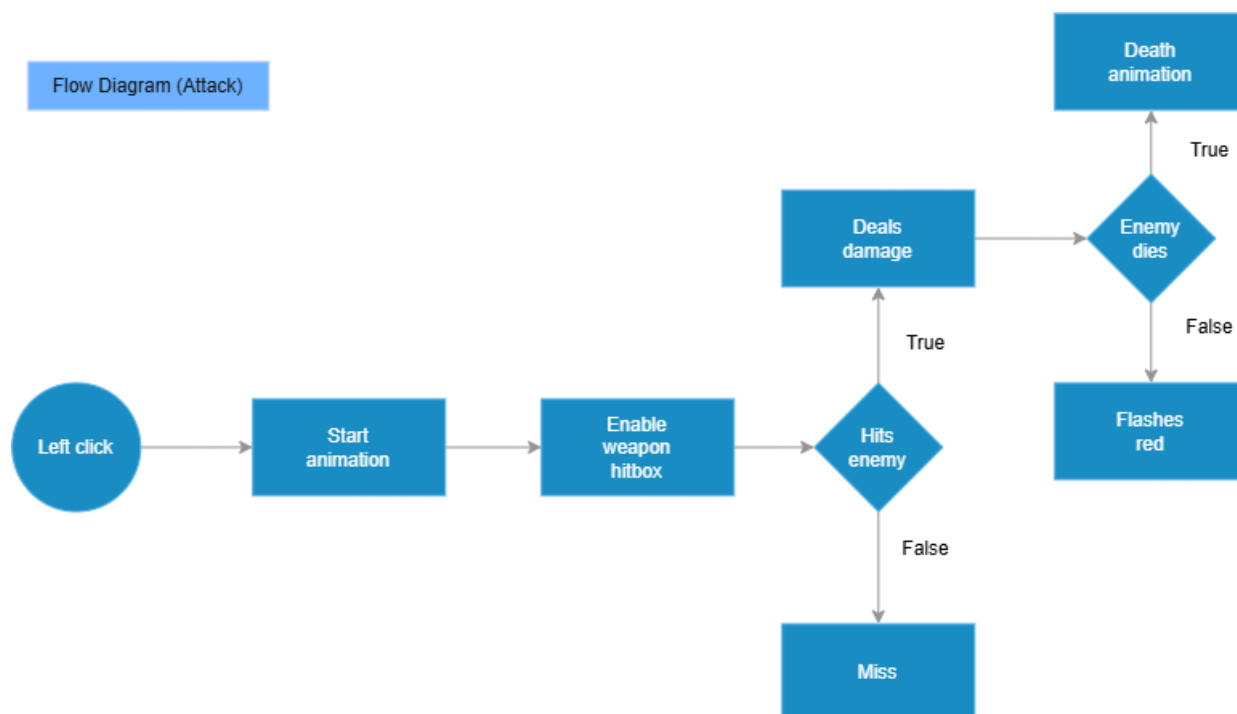
4.3.1 Use Case Diagram



Διάγραμμα 4.1 Use Case Diagram (draw.io)

Στο Use Case Diagram 4.1, παρουσιάζονται οι ενέργειες που μπορεί να κάνει ο χρήστης. Μόλις ανοίξει την εφαρμογή μπορεί να ξεκινήσει το παιχνίδι, να διαλέξει δυσκολία και να επιλέξει και επίπεδο. Μέσα στις πίστες μπορεί να κάνει παύση το παιχνίδι και αν το επιθυμεί, να βγει και να κλείσει την εφαρμογή.

4.3.2 Flow Diagram



Διάγραμμα 4.2 Flow Diagram – Attack (draw.io)

Το διάγραμμα 4.2 είναι ένα flow diagram για την επίθεση του παίκτη στους απλούς εχθρούς μέσα στην πίστα. Δηλαδή το παράδειγμα της ροής δεδομένων που παρουσιάστηκε παραπάνω αλλά μόνο για τους απλούς εχθρούς, όχι για τα bosses. Ο παίκτης ξεκινάει την επίθεση πατώντας το αριστερό κλικ στο ποντίκι του, ενεργοποιώντας έτσι το attack animation στην πορεία του οποίου ενεργοποιείται το hitbox του σπαθιού. Στη συνέχεια αν ο παίκτης δεν χτυπήσει τον εχθρό δεν γίνεται τίποτα, αν τον χτυπήσει του προκαλεί ζημιά. Αν ως αποτέλεσμα της ζημιάς αυτής, ο εχθρός πεθάνει, τότε παίζει το death animation του. Αν όχι, γίνεται στιγμιαία κόκκινος.

5. Ανάλυση Scripts

5.1 ActivateUIDialogue

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.1.

Αυτό το κομμάτι κώδικα χρησιμοποιείται για να εμφανιστεί ο διάλογος όταν ο παίκτης βρίσκεται μέσα στα σπίτια. Στο `OnTriggerEnter2D` ελέγχει αν μπαίνει στο collider κάποιο αντικείμενο με tag: `Player`, άρα ο παίκτης, και εμφανίζει τον διάλογο. Στο `OnTriggerExit2D` αντίστοιχα, με την ίδια διαδικασία εξαφανίζει τον διάλογο.

5.2 Bandit

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.2.

Αυτός ο κώδικας είναι για όλα τα boss fights. Αναλυτικά:

- **Start:** κάνει assign τις τιμές στις μεταβλητές, προσαρμόζει τη ζωή του boss ανάλογα με την επιλεγμένη δυσκολία, χρησιμοποιώντας `PlayerPrefs`.
- **Update:** χρησιμοποιώντας if statements, αλλάζει το animation του boss και καλούνται οι κατάλληλες μέθοδοι.
- **OnTriggerEnter2D:** ελέγχει αν «μπει» στο collider του boss, το αντικείμενο με tag «`Weapon`», το οποίο είναι το σπαθί του παίκτη, αν ναι καλεί τη μέθοδο `TakeDamage`.
- **TakeDamage:** αν το boss δεν μπορεί να δεχθεί ζημιά, ή είναι ήδη νεκρό, τότε κάνει return, καλεί το `DamageCooldown` για να μην δέχεται ζημιά συνεχώς το boss, μειώνει την ζωή του και προσαρμόζει κατάλληλα τα UI στοιχεία όπως μπάρα ζωής και αν πεθαίνει να εξαφανιστεί η μπάρα, να αλλάξει η μουσική και να παίξει το animation θανάτου. Αν η ζωή του δεν είναι ≤ 0 , δηλαδή δεν είναι νεκρό, παίζει το animation «`hurt`» που παίζει όταν δέχεται χτύπημα.
- **DamageCooldown:** αλλάζει την τιμή της μεταβλητής για το αν μπορεί να δεχθεί ζημιά μετά από ένα χρονικό διάστημα.
- **ChasePlayer:** στην Update ελέγχεται αν ο παίκτης είναι σε απόσταση που μπορεί να τον κυνηγήσει το boss, αν ναι καλείται αυτή η μέθοδος. Παίζει το «`run`» animation και αλλάζει και η κατεύθυνση του sprite ανάλογα με το που βρίσκεται ο παίκτης.
- **Attack:** στην Update ελέγχεται αν ο παίκτης είναι σε απόσταση που μπορεί να του επιτεθεί το boss. Παίζει το «`attack`» animation και μετά καλείται το `AttackCooldown`.
- **AttackCooldown:** υπάρχει για να αποτρέπει το boss από το να επιτίθεται ασταμάτητα.
- **StopMusic:** σταματάει τη μουσική του boss fight όταν πεθαίνει το boss.
- **MusicDelay:** παίζει η νικητήρια μουσική μετά από ένα χρονικό διάστημα όταν ο παίκτης νικήσει το boss.

5.3 DeathPatrolling & Patrolling

Αυτά τα δυο scripts είναι για τους απλούς εχθρούς του παιχνιδιού.

5.3.1 DeathPatrolling

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.3.

- **Awake:** αλλάζει τη ζωή του εχθρού ανάλογα με την επιλεγμένη δυσκολία.
- **OnTriggerEnter2D:** αν ο εχθρός είναι ήδη νεκρός κάνει return. Αν «μπει» στο collider ένα αντικείμενο με το tag «Weapon» τότε καλείται η μέθοδος TakeDamage.
- **OnTriggerExit2D:** αν «βγει» το αντικείμενο με tag «Weapon» τότε ο εχθρός μπορεί να δεχθεί ξανά ζημιά.
- **OnTriggerStay2D:** ίδιο με το OnTriggerEnter2D.
- **CreateDeathParticle:** εμφανίζει τα εφέ όταν πεθαίνει ο εχθρός.
- **TakeDamage:** αν δεν μπορεί να δεχθεί ζημιά κάνει return. Αλλιώς δέχεται ζημιά και καλείται η μέθοδος FlashRed. Αν η ζωή του εχθρού είναι ≤ 0 τότε καλείται η μέθοδος Die.
- **Die:** παίζει ο ήχος του θανάτου, εμφανίζονται τα εφέ μέσω της CreateDeathParticle, καταστρέφει το αντικείμενο (destroy).
- **FlashRed:** Αλλάζει το χρώμα του sprite του εχθρού για ένα πολύ μικρό χρονικό διάστημα ως ένδειξη στον παίκτη ότι κατάφερε και χτύπησε τον εχθρό.

5.3.2 Patrolling

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.4.

- **Start:** αρχικοποιεί τον animator και τα στοιχεία μεταξύ των οποίων θα μετακινείται κάθε εχθρός.
- **PlayPatrolSound:** παίζει ήχο αν ο εχθρός είναι ζωντανός.
- **FixedUpdate:** αν ο εχθρός είναι ζωντανός και υπάρχουν τα patrol points, καλεί τη μέθοδο Patrol.

- Patrol: ελέγχει αν λειτουργούν σωστά τα patrol points, αν είναι αρκετά κοντά τους ο εχθρός, αλλάζει το sprite ανάλογα με την κατεύθυνση προς την οποία πηγαίνει.

5.4 EnterBossArena

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.5.

- Start: απενεργοποιεί τη μουσική του boss όταν αρχίζει το επίπεδο.
- OnTriggerEnter2D: αν ο παίκτης «μπει» στο collider της αρένας μάχης, τότε απενεργοποιείται η βασική μουσική της πίστας, καλείται η μέθοδος MusicDelay, ενεργοποιείται το UI για την ζωή και το όνομα του boss.
- MusicDelay: αν ο παίκτης δεν έχει περάσει το collider, δεν είναι δηλαδή δεξιά από αυτό, κάνει return null. Αν όχι, ενεργοποιείται η μουσική του boss και το collider γίνεται στερεό για να μην μπορεί να φύγει ο παίκτης από την αρένα.

5.5 EnterBuilding

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.6.

- OnTriggerEnter2D: ελέγχει αν «μπει» ο παίκτης στο collider. Αν ναι, ελέγχει σε ποιο σπίτι θα μπει, ενεργοποιεί το UI για το ποιο πλήκτρο πρέπει να πατήσει ο παίκτης για να μπει στο σπίτι.
- OnTriggerExit2D: απενεργοποιεί το UI της ένδειξης πλήκτρου για να μπει στο σπίτι όταν ο παίκτης «βγει» από το collider.
- Update: αν ο παίκτης έχει μπει στο collider και πατήσει το πλήκτρο «Ε», τότε καλείται η μέθοδος EnterExitHouse.
- EnterExitHouse: αποθηκεύει σε PlayerPrefs το όνομα της προηγούμενης σκηνής και ανάλογα με το αν είναι η σκηνή Level1 ή Level4, φορτώνει τη σκηνή House ή House2 αντίστοιχα. Και αντίστροφα.

5.6 FollowingCameraTarget

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.7.

- Start: η κάμερα εντοπίζει και ακολουθεί τον παίκτη σε κάθε σκηνή.

5.7 HealthBar

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.8.

- Awake: αν είναι η μπάρα του παίκτη, τότε Instance = this.
- Start: αν είναι η μπάρα του παίκτη και υπάρχει Instance του παίκτη τότε αρχικοποιεί τις τιμές ζωής στην μπάρα.
- Update: ενημερώνει στην μπάρα την τρέχουσα τιμή ζωής.
- SetMaxHealth: ενημερώνει στην μπάρα την μέγιστη τιμή ζωής.
- SetHealth: ενημερώνει στην μπάρα την τρέχων τιμή ζωής.

5.8 HeroKnight

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.9.

- Awake: εντοπίζει τον παίκτη και κάνει Destroy τον παίκτη στην κάθε πίστα και συγκρατεί μόνο τον βασικό παίκτη που έχει πάρει και τυχόν upgrades χρησιμοποιώντας το DontDestroyOnLoad.
- Start: με τα PlayerPrefs αρχικοποιεί τη ζωή του και τη ζημιά που δέχεται από τους εχθρούς. Επιπλέον, δίνει στις μεταβλητές την αρχική τους τιμή.
- OnLevelWasLoaded: αν η σκηνή που φορτώνεται ξεκινάει με «Level» (εκτός από την LevelSelection), είναι δηλαδή μια από τα πέντε επίπεδα, εμφανίζει σωστά τη μπάρα ζωής, επαναφέρει τις ζωές του παίκτη, ο παίκτης κάνει spawn στην αρχή της πίστας, εντοπίζει το collider για την αρένα του boss fight, εντοπίζει το μενού και τη μουσική ήττας. Αρχικοποιεί επίσης το Animator, Rigidbody2D.
- Update: αν ο παίκτης είναι νεκρός, κάνει return. Αλλιώς, χειρίζεται τις ζωές του παίκτη, το χρονόμετρο που είναι υπεύθυνο για τις επιθέσεις, τα animations και γενικά την κίνηση.
- AE_SlideDust: για όταν κάνει slide στους τοίχους.
- OnTriggerEnter2D: αν μπει στο collider με tag: FallDetector, καλείται η μέθοδος HandleFall, αν το tag είναι checkpoint, γίνεται αυτό το respawn σημείο και «γεμίζει» η ζωή του, αν είναι Boss1, αν ο παίκτης δεν είναι νεκρός παίζει το animation που δέχεται ζημιά,

πετάγεται λίγο πίσω και καλείται η μέθοδος TakeDamage, αν είναι BossTrigger, το respawn σημείο είναι πλέον μέσα στην αρένα.

- OnCollisionEnter2D: αν ο παίκτης δεν είναι νεκρός και το tag του αντικειμένου είναι Hazard, δέχεται ζημιά, παίζει το ανάλογο animation, πετάγεται πίσω και καλείται η μέθοδος TakeDamage.
- TakeDamage: αν ο παίκτης δεν μπορεί να δεχτεί ζημιά ή είναι νεκρός κάνει return. Αλλιώς, καλείται η μέθοδος DamageCooldown και στη συνέχεια, αν ο παίκτης κάνει block με την ασπίδα του δέχεται τη μισή ζημιά από αυτή που θα δεχόταν κανονικά. Αλλάζει η μπάρα ζωής για να αντιπροσωπεύει την τρέχουσα ζωή του. Αν η ζωή του είναι ≤ 0 τότε, καλείται η μέθοδος LoseLife, και παίζει το animation του θανάτου, αν είναι > 0 παίζει το animation που δέχεται ζημιά.
- DamageCooldown: η μέθοδος αυτή υπάρχει έτσι ώστε ο παίκτης να μη δέχεται μονίμως ζημιά από κάθε χτύπημα, δηλαδή όσο βρίσκεται στο αντίστοιχο collider.
- LoseLife: αν ο παίκτης έχει και τις δυο ζωές του, χάνει μια ζωή και η καρδιά που την αντιπροσωπεύει στο UI κάτω από τη μπάρα ζωής εξαφανίζεται. Αν έχει χάσει ήδη μια ζωή, τότε απενεργοποιείται και η δεύτερη καρδιά στο UI, προσαρμόζονται τα PlayerPrefs έτσι ώστε αν έχει φθάσει το checkpoint σε μια πίστα όταν την προσπαθήσει ξανά, να κάνει spawn στην αρχή, εμφανίζεται το μενού ήττας και παίζει και ανάλογη μουσική.
- RespawnAfterDelay: αν χάσει την πρώτη του ζωή να κάνει respawn στην πίστα μετά από κάποια δευτερόλεπτα και όχι αμέσως. Αφού κάνει respawn επαναφέρονται όλα στις αρχικές τους τιμές (ζωή, μπάρα ζωής, animator rebind).
- HandleFall: αν πέσει ο παίκτης από την πίστα χάνει μια ζωή. Αν έχει χάσει ήδη μια ζωή και πέσει και τη δεύτερη, ενεργοποιείται η οθόνη και μουσική ήττας, αν όχι κάνει respawn.
- StopMusic: σταματάει όλους τους ήχους.
- MusicDelay: ενεργοποιεί τη μουσική ήττας μετά από ένα δευτερόλεπτο αφού χάσει ο παίκτης.
- ActivateLoseScreen: εμφανίζει την οθόνη ήττας με το κουμπί που οδηγεί στην επιλογή πίστας, όταν χάσει ο παίκτης.
- ResetPlayerOnRestart: κάνει reset όλα τα στατιστικά του παίκτη.

5.9 LampCheckpoint

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.10.

Αυτό το script είναι για τα checkpoints που υπάρχουν στις πίστες. Όταν ο παίκτης φτάσει ένα checkpoint, αν έχει χάσει ζωή, θα γεμίσει πάλι, θα γίνει δηλαδή το μέγιστο. Αν όμως έχει χάσει μία από τις δυο ζωές του, δεν θα την πάρει πίσω.

- Start: «σβήνει» τη λάμπα.
- OnTriggerEnter2D: αν έχει ήδη ενεργοποιηθεί το checkpoint, κάνει return. Αλλιώς, ελέγχει αν το αντικείμενο που μπαίνει στο collider είναι ο παίκτης, αν ναι, καλεί τις μεθόδους FlashPlayerGreen και Activate και αποθηκεύεται στα PlayerPrefs ότι έφθασε ο παίκτης στο checkpoint.
- Activate: η λάμπα «ανάβει» ως ένδειξη ότι ο παίκτης έφθασε στο checkpoint και παίζει ένα sound effect.
- FlashPlayerGreen: αν υπάρχει ο παίκτης και το sprite του, το κάνει πράσινο για ένα μικρό χρονικό διάστημα ως ένδειξη ότι πήρε ζωή και έφθασε στο checkpoint.

5.10 LoadLevels

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.11.

Ο κώδικας αυτός είναι στο άγαλμα που υπάρχει μετά από κάθε boss fight και μέσω αυτού ο παίκτης πηγαίνει στο μενού επιλογής πίστας.

- Update: αν ο παίκτης είναι κοντά στο άγαλμα και πατήσει το πλήκτρο «Ε», τότε με τη χρήση των PlayerPrefs αποθηκεύεται το όνομα της τρέχουσας σκηνής και ανάλογα με ποια είναι, ξεκλειδώνεται η ανάλογη πίστα και στη συνέχεια φορτώνει τη σκηνή LevelSelection. Για παράδειγμα αν ο παίκτης νικήσει το Level3, θα ξεκλειδωθεί το Level4 και θα εμφανιστεί στο μενού.
- OnTriggerEnter2D: αν το boss δεν υπάρχει ή έχει μηδέν ζωή, ελέγχει αν στο collider του αγάλματος μπαίνει ο παίκτης και αν ναι, εμφανίζει το UI που αναγράφει ποιο πλήκτρο χρειάζεται να πατήσει ο παίκτης και η μεταβλητή playerNearby γίνεται true.
- OnTriggerExit2D: αν βγει ο παίκτης, απενεργοποιεί το UI και η μεταβλητή playerNearby γίνεται false.

5.11 LoseMenu

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.12.

Αυτό το κομμάτι κώδικα είναι για το μενού που εμφανίζεται όταν χάνει οριστικά ο παίκτης σε κάποιο επίπεδο.

- GoToLevelSelection: αν πατηθεί το κουμπί στο UI menu, οδηγεί τον παίκτη στη σκηνή LevelSelection.

5.12 MainMenu

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.13

Το script αυτό είναι για το αρχικό μενού.

- StartGame: αυτή η μέθοδος είναι στο κουμπί Start. Αν ο παίκτης δεν έχει επιλέξει ήδη δυσκολία, του εμφανίζει το αντίστοιχο μενού, αφού επιλέξει πηγαίνει αυτόματα στο μενού επιλογής πίστας.
- Options: είναι στο κουμπί Options και εμφανίζει το μενού με τα controls, difficulty, progress.
- QuitGame: στο κουμπί Quit, κλείνει την εφαρμογή.
- Back: είναι στο κουμπί Back και ανάλογα με ποια UI στοιχεία είναι ενεργά, επιστρέφει στο προηγούμενο μενού. Για παράδειγμα, αν ο παίκτης βρίσκεται στο μενού Controls, αν πατήσει Back, θα επιστρέψει στο μενού Options.
- OpenControls: για το κουμπί Controls, εμφανίζει τα controls (τι κάνει κάθε πλήκτρο που χρησιμοποιείται στο παιχνίδι).
- OpenDifficulty: ανοίγει το μενού επιλογής δυσκολίας.
- OpenProgress: ανοίγει το μενού επιλογής διαγραφής προόδου.
- SetDifficultyEasy, SetDifficultyNormal, SetDifficultyHard: ανάλογα με ποια δυσκολία επιλέξει ο παίκτης, αποθηκεύεται στα PlayerPrefs.
- ResetProgress: διαγράφει όλη την πρόοδο του παίκτη στο παιχνίδι.

5.13 PauseMenu

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.14.

Ο κώδικας αυτός είναι για το μενού παύσης.

- Update: αν πατηθεί το πλήκτρο «Escape» (Esc) τότε εμφανίζεται το μενού παύσης. Αν πατηθεί ενώ είναι ήδη ενεργό τότε το παιχνίδι συνεχίζεται.

- Resume: συνεχίζει το παιχνίδι, επαναφέροντάς το στην κανονική του ταχύτητα, συνεχίζοντας τη μουσική και ενεργοποιώντας την κάμερα.
- Pause: σταματάει τη μουσική, την κάμερα και την ταχύτητα του παιχνιδιού.
- Options: εμφανίζει το κουμπί για τα controls.
- Controls: αν πατηθεί το κουμπί στο μενού εμφανίζει τα controls.
- Back: λειτουργεί όπως και στο MainMenu.
- GoToMainMenu: επιστρέφει στο αρχικό μενού και κάνει reset τα PlayerPrefs σχετικά με το checkpoint έτσι ώστε αν επιλέξει πάλι ο παίκτης την πίστα να ξεκινήσει από την αρχή και όχι από το checkpoint.

5.14 SceneSpawnPoint

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.15.

Το σημείο που κάνει spawn ο παίκτης στα Επίπεδα 1 και 4 όταν επιστρέφει από τα στίπια 1 και 2 αντίστοιχα και αντίστροφα.

- Start: συγκρατεί το όνομα της προηγούμενης σκηνής και κάνει spawn τον παίκτη ακριβώς έξω από το στίπι ή μέσα στην είσοδο του σπιτιού ανάλογα με τη σκηνή από την οποία έρχεται.

5.15 SelectLevel

Ολόκληρο το script παρουσιάζεται στο Παράρτημα 10.16.

Το script για την επιλογή πίστας.

- Start: ανάλογα με πόσες πίστες έχει ξεκλειδώσει ο παίκτης, εμφανίζει τις διαθέσιμες επιλογές στην οθόνη χρησιμοποιώντας PlayerPrefs.
- LoadLevel1-5: πάει στη σκηνή του επιπέδου.
- BackToStartMenu: επιστρέφει στο αρχικό μενού.

6. Προβλήματα που προέκυψαν

Όπως σε κάθε παιχνίδι, έτσι και κατά την ανάπτυξη αυτού, προέκυψαν διάφορα προβλήματα και εμπόδια. Αρχικά, το βασικότερο ήταν αν θα μπορούσα να βρω τα assets που ήθελα και χρειάζομαι για το παιχνίδι. Αυτό τελικά δεν ήταν τόσο σπουδαίο πρόβλημα διότι υπάρχει πληθώρα assets για κάθε είδος και τιμή στο Unity Asset Store. Όλα τα assets που χρησιμοποιήθηκαν ήταν δωρεάν. Στη συνέχεια έπρεπε να αποφασίσω τη δομή του παιχνιδιού όπως για παράδειγμα, πόσα επίπεδα θα υπάρχουν. Αυτά ήταν κάποια από τα προβλήματα σε ό,τι αφορά τον σχεδιασμό.

Στο κομμάτι του κώδικα υπήρχαν πολλές ιδέες που δεν γνώριζα τον τρόπο με τον οποίον θα τις υλοποιήσω. Σε αυτό χρησίμευσε πολύ το YouTube και «tips» που μπορεί να έβρισκα από άλλες ιστοσελίδες (Unity Documentation). Ένα από τα βασικότερα προβλήματα ήταν πώς θα αποθηκεύεται η πρόοδος και οι επιλογές του παίκτη στο παιχνίδι. Κατέληξα να χρησιμοποιώ τα PlayerPrefs που είναι μια κλάση στο Unity που αποθηκεύει δεδομένα σχετικά με τον παίκτη. Μερικά από τα τμήματα του παιχνιδιού που χρησιμοποιήθηκαν τα PlayerPrefs ήταν: η αποθήκευση επιλογών upgrade του παίκτη, αποθήκευση της προόδου του, αποθήκευση της επιλεγμένης δυσκολίας. Ακόμα, μια από τις μεγαλύτερες προκλήσεις κατά την ανάπτυξη ήταν πώς ο παίκτης θα συγκρατούσε τα δεδομένα του σε κάθε πίστα. Δηλαδή, να είναι το ίδιο αντικείμενο παίκτη κάθε φορά και όχι ξεχωριστό. Γι' αυτό σε μερικά κομμάτια κώδικα χρησιμοποιείται το Instance του αντικειμένου του παίκτη. Επιπλέον, υπήρχε πρόβλημα με την κίνηση του παίκτη καθώς «κολλούσε» σε μερικά σημεία στο έδαφος. Για να λυθεί αυτό το πρόβλημα προσαρμόστηκαν τα colliders του tilemap σε κάθε σκηνή. Ένα άλλο πρόβλημα ήταν ότι αν ο παίκτης προχωρούσε προς τους εχθρούς και εκείνοι προς αυτόν, άρχιζαν να πηγαίνουν προς τα πάνω και να προχωράνε στον αέρα. Για την επίλυση αυτού του προβλήματος, στα constraints του Rigidbody2D κάθε εχθρού, επιλέχθηκε το Freeze Position για τον άξονα Υ. Έτσι ό,τι και να γίνει, κινούνται μόνο στον άξονα Χ. Επιπροσθέτως, σε μερικά boss fights, αν ο εχθρός χτυπούσε τον παίκτη υπήρχε πιθανότητα να τον πετάξει εκτός πίστας, χωρίς να έχει τρόπο να επιστρέψει. Αυτό διορθώθηκε βάζοντας επιπλέον tiles στον χάρτη.

Εννοείται, πως υπήρχαν αρκετά περισσότερα προβλήματα που ξεπεράστηκαν που δεν αναφέρθηκαν παραπάνω διότι κρίθηκε ότι δεν ήταν αρκετά σημαντικά. Δηλαδή, κάποιες μικρές μετατροπές στον κώδικα όπως η ταχύτητα των εχθρών, η απόσταση εντοπισμού των boss για τον παίκτη κ.α. Ενδέχεται να υπάρχουν και άλλα bugs στο παιχνίδι που δεν εντοπίστηκαν.

7. Συμπεράσματα και μελλοντική επέκταση

7.1 Συμπεράσματα

Προέκυψαν διάφορα συμπεράσματα από την παρούσα πτυχιακή εργασία. Αρχικά, η ανάπτυξη μέχρι και ενός φαινομενικά απλού παιχνιδιού, είναι πολύπλοκη. Ξεκινώντας από την εύρεση των assets και την ανάλυση και υλοποίηση κάθε τομέα του παιχνιδιού, όπως: μουσική, ηχητικά εφέ, κώδικα, αλληλεπίδραση με τον χρήστη, κλπ. Πολύ μεγάλη πρόκληση επίσης ήταν η ανάπτυξη του κώδικα και η σύνδεση των scripts μεταξύ τους, έτσι ώστε να λειτουργούν όλα συνδυαστικά χωρίς κάποιο να προκαλεί προβλήματα σε κάποιο άλλο. Επίσης, ένα ακόμη συμπέρασμα είναι πως κάποια τμήματα του παιχνιδιού που μπορεί να μην παρατηρήσει κάποιος, κάνουν αισθητή διαφορά. Για εμένα στο Medieval Adventures αυτό ήταν τα ηχητικά εφέ, όπου πριν τα προσθέσω δεν ένιωθα ότι έλειπε κάτι αλλά όταν εφαρμόστηκαν και μετά δεν μπορούσα να φανταστώ το παιχνίδι δίχως αυτά.

Εν κατακλείδι, η ανάπτυξη ενός παιχνιδιού απαιτεί πολύ προεργασία και έμφαση στην κάθε λεπτομέρεια για να επιτευχθεί ένα ικανοποιητικό αποτέλεσμα. Χρειάζεται να γραφεί η ιστορία - αν υπάρχει - να αποφασισθεί η δομή του παιχνιδιού, το gameplay και όλα τα άλλα κομμάτια που αποτελούν ένα παιχνίδι. Τέλος, η ανάπτυξη ενός παιχνιδιού είναι κυρίως «trial and error», δηλαδή ο developer να μπορεί να προσαρμοστεί και να επιλύσει τα προβλήματα που προκύπτουν μέχρι να ολοκληρώσει το παιχνίδι.

7.2 Μελλοντική επέκταση

Το παιχνίδι είναι ολοκληρωμένο στα πλαίσια της εργασίας. Έχει δηλαδή, αρχή και τέλος, μενού, μουσική και ηχητικά εφέ και γενικά όλα τα κομμάτια από τα οποία αποτελείται ένα 2D platformer παιχνίδι. Αυτό δεν σημαίνει όμως ότι δεν μπορεί να βελτιωθεί αλλά και να επεκταθεί. Μια από τις σημαντικότερες επεκτάσεις θα μπορούσε να είναι η προσθήκη επιπλέον επιπέδων και ίσως ο διαχωρισμός τους σε ξεχωριστές περιοχές του κόσμου. Έτσι η κάθε περιοχή θα ήταν περισσότερο ιδιαίτερη στον σχεδιασμό και αναγνωρίσιμη. Επίσης, η μεγαλύτερη ποικιλία στους εχθρούς και προσθήκη άλλων NPCs (Non Playable Characters) θα έκανε το παιχνίδι ακόμα πιο ολοκληρωμένο. Τέλος, τα boss fights θα μπορούσαν μελλοντικά να έχουν διαφορετικούς μηχανισμούς το ένα από το άλλο και διαφορετικούς τρόπους να τα προσπεράσει ο παίκτης. Για παράδειγμα το ένα boss να ανακτά ζωή όταν δέχεται χτύπημα ο παίκτης ή να έχει και δεύτερη φάση το boss fight. Ένα παιχνίδι έχει αμέτρητους τρόπους με τους οποίους μπορεί να βελτιώνεται συνεχώς. Γι' αυτό είναι σημαντικό ο developer να μπορεί να σταματήσει την ανάπτυξη όταν είναι ικανοποιημένος γιατί αλλιώς το παιχνίδι δεν θα ολοκληρωθεί ποτέ.

8. Ορολογία

Αγγλικά	Ελληνικά
Animation	Κίνηση
Animator	Ελεγκτής κίνησης στο Unity
Boss / Boss fight	Τελευταία μάχη σε ένα επίπεδο
Checkpoint	Σημείο ελέγχου σε ένα επίπεδο
Collider	Περιοχή σύγκρουσης στα αντικείμενα στο Unity
Cotrols	Πλήκτρα χειρισμού
Difficulty	Βαθμός δυσκολίας
Instance	Στιγμιότυπο (αντικειμένου)
Level	Επίπεδο (πίστα)
Options	Ρυθμίσεις
Platformer	Παιχνίδι πλατφόρμας
Player	Παίκτης
Progress	Πρόοδος
Rebind	Επανασύνδεση (σε κώδικα)
Reset	Επαναφορά
Respawn	Επανεμφάνιση (π.χ. παίκτη)
Script	Κομμάτι κώδικα
Slide	Ολίσθηση
Sound effects	Ηχητικά εφέ
Spawn / Spawn point	Σημείο εμφάνισης
Sprite	Δισδιάστατο γραφικό (εικόνα)
Start / Pause / Resume	Έναρξη / Παύση / Συνέχεια
Tag	Ετικέτα (για αντικείμενα στο Unity)
UI	Διεπαφή χρήστη
Upgrades	Αναβαθμίσεις

9. Βιβλιογραφία

Unity Asset Store:

- [FREE Casual Game SFX Pack](#)
- [8-bit Fantasy & Adventure Music](#)
- [Free 2D Platformer - Skull Garden Tilesets Asset Pack](#)
- [Pixel Skies DEMO Background pack](#)
- [Free 6 Dark Fantasy Boss Battle Tracks](#)
- [Free 10 Medieval Ambient Fantasy Tracks Music Pack](#)

- Medieval King Pack 2
- Bandits - Pixel Art
- Hero Knight 2
- Evil Wizard 3
- Evil Wizard 2
- Hero Knight - Pixel Art
- 2D Pixel Art Platformer | Biome - Plains
- Free 2D Enemy Patrol AI – Animated Platformer Patrol System with VFX & Sound
- 2D Pixel Art Platformer | Biome - American Forest
- 2D TileSet Platformer
- Pixel Art Platformer - Village Props
- Medieval Music Pack Vol. 2
- GothicVania Town

Βιβλιογραφική βάση:

1. Kamitsios, M., Papadimitriou, S., Chrysafiadi, K., & Virvou, M. (2018). Incorporating quizzes into a 3D educational game calling web services. In EdMedia

+ Innovate Learning 2018 (pp. 1170-1179). Association for the Advancement of Computing in Education (AACE).

2. Kamitsios, M., Chrysafiadi, K., Virvou, M., & Sakkopoulos, E. (2018). A Stereotype User Model for an Educational Game: Overcome the Difficulties in

Game Playing and Focus on the Educational Goal. 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA).

IEEE. <https://doi.org/10.1109/IISA.2018.8633655>

3. Papadimitriou, S., Chrysafiadi, K., Kamitsios, M., & Virvou, M. (2018). FSP Creator: A Novel Web Service API Creator of Fuzzy Students Progress Profile.

2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA), 1-7. IEEE. <https://doi.org/10.1109/IISA.2018.8633662>

4. Papadimitriou, S., Kamitsios, M., Chrysafiadi, K., & Virvou, M. (2021). Learn-and-play personalised reasoning from point-and-click to virtual reality mobile

educational games. Intelligent Decision Technologies, 15(2), 321-332. <https://doi.org/10.3233/IDT-200204>

5. Chrysafiadi, K., Kamitsios, M., & Virvou, M. (2023). Fuzzy-based dynamic difficulty adjustment of an educational 3D-game. Multimedia Tools and

Applications, 82, 27525-27549. <https://doi.org/10.1007/s11042-023-14515-w>

6. Chrysafiadi, K., Papadimitriou, S., Kamitsios, M., & Virvou, M. (2026; first online 2025). User Experience in Serious Games: Adaptivity in the Plot and

Environment of the Game. In M. Virvou et al. (Eds.), Artificial Intelligence-Empowered Software Engineering 2024, Lecture Notes in Artificial Intelligence

and Soft Computing, Vol. 52, pp. 215-230. Springer. https://doi.org/10.1007/978-3-031-98410-5_13

Άλλες πηγές:

- [Super Mario Bros - εικόνα](#)
- [Super Mario Bros Wonder - εικόνα](#)
- [Super Mario Bros - Wikipedia](#)
- [Hollow Knight - εικόνα](#)
- [Hollow Knight - Wikipedia](#)
- [Εικόνες μενού: Images by pikisuperstar on Magnific](#)
- [YouTube: Brackeys – Health Bar](#)
- [Unity Documentation](#)

10. Παραρτήματα

10.1 Παράρτημα

```
using UnityEngine;

public class ActivateUIDialogue : MonoBehaviour
{
    [SerializeField] private GameObject enterUI;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        // activates when the object with the "Player" tag enters
        if (collision.CompareTag("Player"))
        {
            enterUI.SetActive(true);
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        // deactivates when the object with the "Player" tag enters
        if (collision.CompareTag("Player"))
        {
            enterUI.SetActive(false);
        }
    }
}
```

10.2 Παράρτημα

```

using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.Windows;

public class Bandit : MonoBehaviour {

    [SerializeField] float m_speed = 4.0f;

    [SerializeField] Transform player;
    [SerializeField] float detectionRange = 8f;
    [SerializeField] float attackRange = 1.5f;

    private Animator m_animator;
    private Rigidbody2D m_body2d;
    private Sensor_Bandit m_groundSensor;
    private bool m_grounded = false;
    private bool m_combatIdle = false;
    private bool m_isDead = false;

    private bool isAttacking = false;
    private Vector3 originalSize;

    public int maxHealth = 400;
    public int currentHealth;

    public HealthBar healthBar;
    private bool isDead = false;
    private bool canTakeDamage = true;

    [SerializeField] private GameObject enemyHealthUI;
    [SerializeField] private AudioSource winMusic;

    // Use this for initialization
    void Start () {
        originalSize = transform.localScale;
        m_animator = GetComponent<Animator>();
        m_body2d = GetComponent<Rigidbody2D>();
        m_groundSensor = transform.Find("GroundSensor").GetComponent<Sensor_Ban-
dit>();

        // changes boss' health based on the difficulty saved in the player prefer-
ences
        int difficulty = PlayerPrefs.GetInt("Difficulty");

        if (difficulty == 0)
        {
            maxHealth -= 100;
        }
        else if (difficulty == 2)
        {
            maxHealth += 100;
        }

        // updates health
        currentHealth = maxHealth;
    }

```

```
healthBar.SetMaxHealth(maxHealth);

if (HeroKnight.Instance != null)
{
    player = HeroKnight.Instance.transform;
}
}

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

    // if boss is already dead, return
    if (isDead) return;

    if (player == null && HeroKnight.Instance != null)
    {
        player = HeroKnight.Instance.transform;
    }

    if (player == null) return;

    float distance = Vector2.Distance(transform.position, player.position);

    //Check if character just landed on the ground
    if (!m_grounding && m_groundSensor.State()) {
        m_grounding = true;
        m_animator.SetBool("Grounded", m_grounding);
    }

    //Check if character just started falling
    if(m_grounding && !m_groundSensor.State()) {
        m_grounding = false;
        m_animator.SetBool("Grounded", m_grounding);
    }

    // Chase player
    if (distance <= detectionRange && distance > attackRange)
    {
        ChasePlayer();
    }

    // Attack player
    else if (distance <= attackRange)
    {
        Attack();
    }

    // Idle
    else
    {
        m_body2d.linearVelocity = new Vector2(0, m_body2d.linearVelocity.y);
        m_animator.SetInteger("AnimState", 0);
    }
}

private void OnTriggerEnter2D(Collider2D collision)
{

```

```
// if the object entering is tagged with "Weapon" (player's weapon), call
the TakeDamage method
if (collision.gameObject.CompareTag("Weapon"))
{
    TakeDamage(40);
}

void TakeDamage(int damage)
{
    // if boss can't take damage or is already dead, return
    if (!canTakeDamage || isDead) return;

    StartCoroutine(DamageCooldown());

    // reduces health
    currentHealth -= damage;

    // adjusts healthbar
    healthBar.SetHealth(currentHealth);

    // if health is 0, the boss dies, plays the correct animation and the object
    is destroyed
    if (currentHealth <= 0)
    {
        isDead = true;
        m_body2d.linearVelocity = Vector2.zero;
        m_animator.SetTrigger("Death");
        Destroy(gameObject, 2f);

        // disables enemy healthbar and stops boss music
        enemyHealthUI.SetActive(false);
        StopMusic();
        StartCoroutine(MusicDelay());
    }
    // if health is not 0, plays the hurt animation
    else
    {
        m_animator.SetTrigger("Hurt");
    }
}

// exists so that the boss doesn't constantly take damage
IEnumerator DamageCooldown()
{
    canTakeDamage = false;
    yield return new WaitForSeconds(0.5f);
    canTakeDamage = true;
}

// chases the player when detected
private void ChasePlayer()
{
    Vector2 direction = (player.position - transform.position).normalized;

    m_body2d.linearVelocity = new Vector2(direction.x * m_speed, m_body2d.linearVelocity.y);
}
```

```

        // Swap direction of sprite depending on walk direction
        if (direction.x > 0)
            transform.localScale = new Vector3(-originalSize.x, originalSize.y,
originalSize.z);
        else
            transform.localScale = new Vector3(originalSize.x, originalSize.y, orig-
inalSize.z);

        // Run
        m_animator.SetInteger("AnimState", 2);
    }

    // attacks the player when within attack range
    private void Attack()
    {
        m_body2d.linearVelocity = Vector2.zero;

        // if not already attacking, attack
        if (!isAttacking)
        {
            isAttacking = true;
            m_animator.SetTrigger("Attack");
            StartCoroutine(AttackCooldown());
        }
    }

    // exists so that the boss doesn't attack constantly
    IEnumerator AttackCooldown()
    {
        yield return new WaitForSeconds(1.5f);
        isAttacking = false;
    }

    // stops music when the boss dies
    public void StopMusic()
    {
        AudioSource[] allAudioSources = FindObjectsByType<AudioSource>(FindOb-
jectsSortMode.None);

        foreach (AudioSource source in allAudioSources)
        {
            source.Stop();
        }
    }

    // plays victory music after the boss is defeated
    private IEnumerator MusicDelay()
    {
        yield return new WaitForSeconds(1);
        winMusic.Play();
    }
}

```

10.3 Παράρτημα

```

using System.Collections;
using UnityEngine;

```

```
namespace EnemyPatrolling
{
    public class DeathPatrolling : MonoBehaviour
    {
        private bool isDead = false;
        private bool canTakeDamage = true;

        [SerializeField] int baseHealth = 80;
        private int currentHealth;

        [SerializeField] private AudioClip Death;
        [SerializeField] private AudioSource audioSource;
        [SerializeField] private ParticleSystem DeathParticle;
        [SerializeField] private float flashDuration = 0.1f;

        private Collider2D col;
        private SpriteRenderer sprite;
        private Color originalColor;

        // called when the script instance is being loaded
        private void Awake()
        {
            // changes the base enemy's health based on the difficulty chosen by the
            user
            int difficulty = PlayerPrefs.GetInt("Difficulty", 1);

            currentHealth = baseHealth;

            if (difficulty == 0)
            {
                currentHealth = baseHealth - 40;
            }
            else if (difficulty == 2)
            {
                currentHealth = baseHealth + 40;
            }

            // Get references to the collider and sprite renderer
            col = GetComponent<Collider2D>();
            sprite = GetComponent<SpriteRenderer>();
            originalColor = sprite.color;
        }

        private void OnTriggerEnter2D(Collider2D collision)
        {
            // if enemy already dead, returns
            if (isDead) return;

            // if not dead and the object entering is tagged with "Weapon" (the
            player's sword), calls the TakeDamage method to deal damage
            if (collision.CompareTag("Weapon"))
            {
                TakeDamage(20);
            }
        }

        private void OnTriggerExit2D(Collider2D collision)
```

```
{
    /* when the weapon exits, the enemy can take damage again
     * this is done to prevent the enemy from taking multiple hits
     * during on instance of the weapon entering its collider
     */

    if (collision.CompareTag("Weapon"))
    {
        canTakeDamage = true;
    }
}

private void OnTriggerStay2D(Collider2D collision)
{
    // if enemy already dead, returns
    if (isDead) return;

    // if not dead and the object entering is tagged with "Weapon" (the
    player's sword), calls the TakeDamage method to deal damage
    if (collision.CompareTag("Weapon") && canTakeDamage)
    {
        TakeDamage(20);
    }
}

private void CreateDeathParticle()
{
    // creates the death particles when enemy dies
    if (DeathParticle != null)
    {
        Instantiate(DeathParticle, transform.position, transform.rotation).Play();
    }
}

// is called when the enemy takes damage
void TakeDamage(int damage)
{
    if (!canTakeDamage) return;

    /* if they can take damage (depending on the value of the bool above)
     * their health is decreased and they flash red to indicate that they
     have been hit
     * if their health reaches 0, the method Die is called
     */

    canTakeDamage = false;
    baseHealth -= damage;
    StartCoroutine(FlashRed());

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

// is called when enemy health reaches 0
void Die()
{
}
```

```

        isDead = true;

        // Play death sound
        if (audioSource != null && Death != null)
        {
            audioSource.PlayOneShot(Death);
        }

        // Spawn particles
        CreateDeathParticle();

        // Hide visual and disable collision
        if (col != null) col.enabled = false;
        if (sprite != null) sprite.enabled = false;

        // Destroy after 0.5 seconds
        Destroy(gameObject, 0.5f);
    }

    // changes the color of the sprite to red for some seconds to indicate that
    // they've taken damage
    IEnumerator FlashRed()
    {
        sprite.color = Color.red;

        yield return new WaitForSeconds(flashDuration);

        sprite.color = originalColor;
    }
}

```

10.4 Παράρτημα

```

using UnityEngine;

namespace EnemyPatrolling
{
    public class Patrolling : MonoBehaviour
    {
        // sprite to rotate
        private SpriteRenderer sprite;

        // Array of waypoints the enemy will patrol between
        public Transform[] patrolPoints;

        // Speed at which the enemy moves between patrol points
        public float moveSpeed = 3.0f;

        // Rotation to apply to the enemy when it reaches a patrol point
        public Vector3 rotationAngles = new Vector3(0f, 180f, 0f);

        // Keeps track of the index of the current target patrol point
        private int currentPointIndex = 0;
    }
}

```

```
// The transform of the current patrol target
private Transform currentTarget;

// Indicates whether the enemy is dead; prevents movement and sounds if true
private bool isDead = false;

// Reference to the Animator component for controlling animations
private Animator anim;

// The sound that plays while patrolling
[SerializeField] private AudioClip PatrolSound;

// The audio source component used to play the patrol sound
[SerializeField] private AudioSource audioSource;

// Called once when the game starts
private void Start()
{
    // Ensure there are patrol points set up
    if (patrolPoints.Length > 0)
    {
        // Set the initial patrol target and position
        currentTarget = patrolPoints[currentPointIndex];
        transform.position = currentTarget.position + new Vector3(1, 0.5f,
0);

        // Start playing the patrol sound every 4 seconds
        // InvokeRepeating("PlayPatrolSound", 0f, 10f);

        // Get the Animator component attached to this GameObject
        anim = GetComponent<Animator>();
        sprite = GetComponent<SpriteRenderer>();
    }
}

// Plays a patrol sound if the enemy is alive
private void PlayPatrolSound()
{
    if (!isDead)
    {
        audioSource.PlayOneShot(PatrolSound);
    }
}

// Called at a fixed time interval, suitable for physics-based movement
private void FixedUpdate()
{
    // Only patrol if the enemy is alive and has patrol points
    if (!isDead && patrolPoints.Length > 0)
    {
        Patrol();
    }
}

// Handles moving the enemy towards the current patrol point
private void Patrol()
{
    // Calculate normalized direction vector towards the target
```

```

        Vector3 moveDirectionVector = (currentTarget.position - transform.position).normalized;

        // flipping sprite when changing direction
        if (moveDirectionVector.x > 0)
        {
            sprite.flipX = false;    // right
        }
        else if (moveDirectionVector.x < 0)
        {
            sprite.flipX = true;     // left
        }

        // Move the enemy towards the target point at the defined speed
        transform.position += moveDirectionVector * moveSpeed * Time.deltaTime;

        // Check if the enemy is close enough to the patrol point
        if (Vector2.Distance(transform.position, currentTarget.position) < 0.2f)
        {
            // Move to the next point in the patrol array
            currentPointIndex++;

            // Loop back to the first patrol point if at the end
            if (currentPointIndex >= patrolPoints.Length)
            {
                currentPointIndex = 0;
            }

            // Set the new patrol target
            currentTarget = patrolPoints[currentPointIndex];
        }
    }
}

```

10.5 Παράρτημα

```

using System.Collections;
using UnityEngine;

public class EnterBossArena : MonoBehaviour
{
    [SerializeField] private GameObject enemyHealth;
    [SerializeField] private AudioSource music1;
    [SerializeField] private AudioSource music2;

    private BoxCollider2D bossCollider;
    private bool bossFightStarted = false;

    // Start is called before any of the Update methods is called for the first time
    private void Start()
    {
        // disables the boss music when loading the level
        music2.enabled = false;
    }
}

```

```

        bossCollider = GetComponent<BoxCollider2D>();
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        /* if the objects that entered the arena collider is the player, it disables
the main music,
        * starts the boss music after some time,
        * enables the boss' healthbar
        */

        if (collision.CompareTag("Player") && !bossFightStarted)
        {
            bossFightStarted = true;

            music1.enabled = false;
            StartCoroutine(MusicDelay(collision.transform));

            enemyHealth.SetActive(true);
        }
    }

    // enables boss music 1 second after entering the arena and makes the arena col-
lider solid so that the player cannot leave
    private IEnumerator MusicDelay(Transform playerTransform)
    {
        yield return new WaitForSeconds(1);

        while (playerTransform.position.x < bossCollider.bounds.center.x)
        {
            yield return null;
        }

        music2.enabled = true;
        bossCollider.isTrigger = false;
    }
}

```

10.6 Παράρτημα

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class EnterBuilding : MonoBehaviour
{
    [SerializeField] private GameObject enterUI;

    private bool player_detection = false;
    private bool EnteredHouse1 => PlayerPrefs.GetInt("EnteredHouse1", 0) == 1;
    private bool EnteredHouse2 => PlayerPrefs.GetInt("EnteredHouse2", 0) == 1;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        // checks if the object taht entered has the "Player" tag
        if (collision.CompareTag("Player"))

```

```
{
    // if the current scene is the Level1 or House and the player hasn't al-
    ready entered House, enables the UI and sets the player_detection to true
    if ((SceneManager.GetActiveScene().name == "Level1" || SceneManager.Get-
ActiveScene().name == "House") && !EnteredHouse1)
    {
        enterUI.SetActive(true);
        player_detection = true;
    }

    // if the current scene is the Level4 or House2 and the player hasn't
    already entered House2, enables the UI and sets the player_detection to true
    if ((SceneManager.GetActiveScene().name == "Level4" || SceneManager.Get-
ActiveScene().name == "House2") && !EnteredHouse2)
    {
        enterUI.SetActive(true);
        player_detection = true;
    }
}

// when exiting, if the object is the Player, disables UI and sets player_detec-
tion to false
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.CompareTag("Player"))
    {
        enterUI.SetActive(false);
        player_detection = false;
    }
}

// Update is called once per frame
private void Update()
{
    // if the player has entered the collider and is pressing the E key then the
    EnterExitHouse function is called
    if (player_detection && Input.GetKeyDown(KeyCode.E))
    {
        EnterExitHouse();
    }
}

// method for loading scenes: Level1, House, Level4, House2
void EnterExitHouse()
{
    // changes the player preferences "PreviousScene" to the current one
    PlayerPrefs.SetString("PreviousScene", SceneManager.GetActiveScene().name);
    PlayerPrefs.Save();

    // if the current scene is the House, it loads Level1 and changes the bool
    Player Preference
    if (SceneManager.GetActiveScene().name == "House")
    {
        PlayerPrefs.SetInt("EnteredHouse1", 1);
        PlayerPrefs.Save();
        SceneManager.LoadScene("Level1");
    }
}
```

```

    // if the current scene is the House2, it loads Level4 and changes the bool
    Player Preference
    else if (SceneManager.GetActiveScene().name == "House2")
    {
        PlayerPrefs.SetInt("EnteredHouse2", 1);
        PlayerPrefs.Save();
        SceneManager.LoadScene("Level4");
    }
    // if the current scene is Level1, it loads the House
    else if (SceneManager.GetActiveScene().name == "Level1")
    {
        if (HeroKnight.Instance != null)
        {
            PlayerPrefs.SetInt("SaveLivesLost", HeroKnight.Instance.livesLost);
            PlayerPrefs.Save();
        }
        SceneManager.LoadScene("House");
    }
    // if the current scene is Level4, it loads the House2
    else
    {
        if (HeroKnight.Instance != null)
        {
            PlayerPrefs.SetInt("SaveLivesLost", HeroKnight.Instance.livesLost);
            PlayerPrefs.Save();
        }
        SceneManager.LoadScene("House2");
    }
}
}
}

```

10.7 Παράρτημα

```

using Unity.Cinemachine;
using UnityEngine;

public class FollowingCameraTarget : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBe-
    // haviour is created
    void Start()
    {
        var cam = GetComponent<CinemachineCamera>();

        // follows the player in every scene
        if (cam != null && HeroKnight.Instance != null)
        {
            cam.Follow = HeroKnight.Instance.transform;
            cam.LookAt = HeroKnight.Instance.transform;
        }
    }
}

```

10.8 Παράρτημα

```
using UnityEngine;
using UnityEngine.UI;

public class HealthBar : MonoBehaviour
{
    public Slider slider;

    [Header("Settings")]
    public bool isPlayerHealthBar = false;

    public static HealthBar Instance;

    // Awake is called when the script instance is being loaded
    private void Awake()
    {
        if (isPlayerHealthBar)
        {
            Instance = this;
        }
    }

    // Start is called just before any of the Update methods is called for the first
    time
    private void Start()
    {
        // sets up the player's healthbar
        if (isPlayerHealthBar && HeroKnight.Instance != null)
        {
            slider.maxValue = HeroKnight.Instance.maxHealth;
            slider.value = HeroKnight.Instance.currentHealth;
        }
    }

    // Update is called once per frame
    private void Update()
    {
        // changes the value of the healthbar
        if (isPlayerHealthBar && HeroKnight.Instance != null)
        {
            slider.value = HeroKnight.Instance.currentHealth;
        }
    }

    public void SetMaxHealth(int health)
    {
        slider.maxValue = health;
        slider.value = health;
    }

    public void SetHealth(int health)
    {
        slider.value = health;
    }
}
```

10.9 Παράρτημα

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using UnityEngine.EventSystems;
using Unity.VisualScripting.Dependencies.NCalc;
using Unity.VisualScripting;

public class HeroKnight : MonoBehaviour
{
    public static HeroKnight Instance { get; private set; }

    [SerializeField] float m_speed = 4.0f;
    [SerializeField] float m_jumpForce = 7.5f;
    [SerializeField] float m_rollForce = 6.0f;
    [SerializeField] bool m_noBlood = false;
    [SerializeField] GameObject m_slideDust;

    [SerializeField] private AudioSource loseMusic;

    private BoxCollider2D bossCollider;

    private Animator m_animator;
    private Rigidbody2D m_body2d;
    private Sensor_HeroKnight m_groundSensor;
    private Sensor_HeroKnight m_wallSensorR1;
    private Sensor_HeroKnight m_wallSensorR2;
    private Sensor_HeroKnight m_wallSensorL1;
    private Sensor_HeroKnight m_wallSensorL2;
    private bool m_isWallSliding = false;
    private bool m_grounded = false;
    private bool m_rolling = false;
    private int m_facingDirection = 1;
    private int m_currentAttack = 0;
    private float m_timeSinceAttack = 0.0f;
    private float m_delayToIdle = 0.0f;
    private float m_rollDuration = 8.0f / 14.0f;
    private float m_rollCurrentTime;

    private bool canTakeDamage = true;

    public bool obtainedUpgrade = false;

    public int damageTaken = 30;

    // fall detection
    private Vector3 respawnPoint;

    // knockback from damage
    [SerializeField] private Rigidbody2D rb;
    [SerializeField] private float knockbackForce = 5f;

    // health
    public int maxHealth = 100;
    public int currentHealth;
```

```

public float respawnDelay = 2f;

private bool isRespawning = false;
private bool isDead = false;

public HealthBar healthBar;
public GameObject LoseScreenUI;

[SerializeField] GameObject life1, life2;

private bool reachCheckpoint = false;
public int livesLost = 0;

/* player is not destroyed so that there is only one player in the whole game
 * and upgrades are kept throughout
 */

private void Awake()
{
    // ensures that a level can be played with the same player, without needing
    to start at level 1
    if (Instance != null && Instance != this)
    {
        Destroy(gameObject);
        return;
    }
    Instance = this;
    DontDestroyOnLoad(gameObject);
}

// Use this for initialization
void Start ()
{
    // ensures the player has the correct stats depending on any upgrades ob-
    tained
    if (Instance != this) return;

    maxHealth = PlayerPrefs.GetInt("SavedMaxHealth", 100);
    damageTaken = PlayerPrefs.GetInt("SavedDamageTaken", 30);

    m_animator = GetComponent<Animator>();
    m_body2d = GetComponent<Rigidbody2D>();
    m_groundSensor = transform.Find("GroundSensor").GetComponent<Sensor_Her-
oKnight>();
    m_wallSensorR1 = transform.Find("WallSensor_R1").GetComponent<Sensor_Her-
oKnight>();
    m_wallSensorR2 = transform.Find("WallSensor_R2").GetComponent<Sensor_Her-
oKnight>();
    m_wallSensorL1 = transform.Find("WallSensor_L1").GetComponent<Sensor_Her-
oKnight>();
    m_wallSensorL2 = transform.Find("WallSensor_L2").GetComponent<Sensor_Her-
oKnight>();

    loseMusic.enabled = false;
    LoseScreenUI.SetActive(false);

    respawnPoint = transform.position;

```

```
currentHealth = maxHealth;

if (HealthBar.Instance != null)
{
    HealthBar.Instance.SetMaxHealth(maxHealth);
    HealthBar.Instance.SetHealth(currentHealth);
}
}

// is called after a level is loaded
private void OnLevelWasLoaded(int level)
{
    /* if scene starts with Level, ensure everything works properly:
    * - healthbar is displayed correctly
    * - lives are reset
    * - player spawns at the start of the level
    * - finds the boss collider for the arena
    * - finds lose screen and lose music
    */

    if (Instance != null && Instance != this)
    {
        return;
    }

    string sceneName = SceneManager.GetActiveScene().name;
    if (sceneName.StartsWith("Level") && sceneName != "LevelSelection")
    {
        currentHealth = maxHealth;
        if (HealthBar.Instance != null)
        {
            HealthBar.Instance.SetHealth(currentHealth);
        }

        livesLost = PlayerPrefs.GetInt("SaveLivesLost", 0);
        if (livesLost >= 1)
        {
            if (life2 != null) life2.SetActive(false);
        }
        else
        {
            if (life2 != null) life2.SetActive(true);
        }

        if (livesLost >= 2)
        {
            if (life2 != null) life1.SetActive(false);
        }
        else
        {
            if (life2 != null) life1.SetActive(true);
        }

        isDead = false;
        isRespawning = false;
        reachCheckpoint = false;
    }
}
```

```
GameObject musicObj = GameObject.FindWithTag("LoseMusic");
if (musicObj != null)
{
    loseMusic = musicObj.GetComponent<AudioSource>();
    if (loseMusic != null) loseMusic.enabled = false;
}

GameObject loseScreenObj = GameObject.FindWithTag("LoseScreen");
if (loseScreenObj != null)
{
    LoseScreenUI = loseScreenObj;
    LoseScreenUI.SetActive(false);
}

Collider2D playerCollider = GetComponent<Collider2D>();
if (playerCollider != null)
{
    playerCollider.enabled = true;
}

if (m_animator != null)
{
    m_animator.Rebind();
    m_animator.Update(0f);
    m_animator.SetInteger("AnimState", 0);
    m_animator.SetBool("Grounded", true);
}

if (m_body2d != null)
{
    m_body2d.bodyType = RigidbodyType2D.Dynamic;
    m_body2d.linearVelocity = Vector2.zero;
}
if (rb != null)
{
    rb.linearVelocity = Vector2.zero;
}

GameObject spawnPointYes = GameObject.FindWithTag("StartSpawnPoint");
GameObject checkpointSpawn = GameObject.FindWithTag("Checkpoint");

if (PlayerPrefs.GetInt(sceneName + "_checkpoint", 0) == 1 && check-
pointSpawn != null)
{
    transform.position = checkpointSpawn.transform.position + Vector3.up
* 2f;
    respawnPoint = checkpointSpawn.transform.position;
    reachCheckpoint = true;
}
else if (spawnPointYes != null)
{
    transform.position = spawnPointYes.transform.position;
    respawnPoint = spawnPointYes.transform.position;
}

if (sceneName.StartsWith("House"))
{
```

```
        obtainedUpgrade = false;
    }

    GameObject bossTriggerYes = GameObject.FindWithTag("BossTrigger");
    if (bossTriggerYes != null)
    {
        bossCollider = bossTriggerYes.GetComponent<BoxCollider2D>();
    }
    else bossCollider = null;
}

// Update is called once per frame
void Update ()
{
    // if player is dead, return
    if (isDead) return;

    // managing lives
    if (life1 == null || life2 == null)
    {
        GameObject foundLife1 = GameObject.Find("Life1");
        GameObject foundLife2 = GameObject.Find("Life2");

        if (foundLife1 != null) life1 = foundLife1;
        if (foundLife2 != null) life2 = foundLife2;
    }

    if (HealthBar.Instance != null)
    {
        HealthBar.Instance.SetHealth(currentHealth);
    }

    if (life1 != null && life2 != null)
    {
        if (livesLost >= 2) life1.SetActive(false);
        else life1.SetActive(true);

        if (livesLost >= 1) life2.SetActive(false);
        else life2.SetActive(true);
    }

    // Increase timer that controls attack combo
    m_timeSinceAttack += Time.deltaTime;

    // Increase timer that checks roll duration
    if(m_rolling)
        m_rollCurrentTime += Time.deltaTime;

    // Disable rolling if timer extends duration
    if(m_rollCurrentTime > m_rollDuration)
        m_rolling = false;

    // Check if character just landed on the ground
    if (!m_grounded && m_groundSensor.State())
    {
        m_grounded = true;
        m_animator.SetBool("Grounded", m_grounded);
    }
}
```

```
//Check if character just started falling
if (m_grounded && !m_groundSensor.State())
{
    m_grounded = false;
    m_animator.SetBool("Grounded", m_grounded);
}

// -- Handle input and movement --
float inputX = Input.GetAxis("Horizontal");

// Swap direction of sprite depending on walk direction
if (inputX > 0)
{
    transform.localScale = new Vector3(1, 1, 1);
    m_facingDirection = 1;
}

else if (inputX < 0)
{
    transform.localScale = new Vector3(-1, 1, 1);
    m_facingDirection = -1;
}

// Move
if (!m_rolling )
    m_body2d.linearVelocity = new Vector2(inputX * m_speed, m_body2d.linear-
Velocity.y);

//Set AirSpeed in animator
m_animator.SetFloat("AirSpeedY", m_body2d.linearVelocity.y);

// -- Handle Animations --
//Wall Slide
m_isWallSliding = (m_wallSensorR1.State() && m_wallSensorR2.State()) ||
(m_wallSensorL1.State() && m_wallSensorL2.State());
m_animator.SetBool("WallSlide", m_isWallSliding);

//Attack
if(Input.GetMouseButtonDown(0) && m_timeSinceAttack > 0.25f && !m_rolling
&& !(EventSystem.current != null && EventSystem.current.IsPointerOver-
GameObject()))
{
    m_currentAttack++;

    // Loop back to one after third attack
    if (m_currentAttack > 3)
        m_currentAttack = 1;

    // Reset Attack combo if time since last attack is too large
    if (m_timeSinceAttack > 1.0f)
        m_currentAttack = 1;

    // Call one of three attack animations "Attack1", "Attack2", "Attack3"
    m_animator.SetTrigger("Attack" + m_currentAttack);

    // Reset timer
    m_timeSinceAttack = 0.0f;
}
```

```
// Block
else if (Input.GetMouseButtonDown(1) && !m_rolling)
{
    m_animator.SetTrigger("Block");
    m_animator.SetBool("IdleBlock", true);
}

else if (Input.GetMouseButtonUp(1))
    m_animator.SetBool("IdleBlock", false);

// Roll
else if (Input.GetKeyDown("left shift") && !m_rolling && !m_isWallSliding)
{
    m_rolling = true;
    m_animator.SetTrigger("Roll");
    m_body2d.linearVelocity = new Vector2(m_facingDirection * m_rollForce,
m_body2d.linearVelocity.y);
}

//Jump
else if (Input.GetKeyDown("space") && m_grounded && !m_rolling)
{
    m_animator.SetTrigger("Jump");
    m_grounded = false;
    m_animator.SetBool("Grounded", m_grounded);
    m_body2d.linearVelocity = new Vector2(m_body2d.linearVelocity.x,
m_jumpForce);
    m_groundSensor.Disable(0.2f);
}

//Run
else if (Mathf.Abs(inputX) > Mathf.Epsilon)
{
    // Reset timer
    m_delayToIdle = 0.05f;
    m_animator.SetInteger("AnimState", 1);
}

//Idle
else
{
    // Prevents flickering transitions to idle
    m_delayToIdle -= Time.deltaTime;
    if(m_delayToIdle < 0)
        m_animator.SetInteger("AnimState", 0);
}

}

// Animation Events
// Called in slide animation.
void AE_SlideDust()
{
    Vector3 spawnPosition;

    if (m_facingDirection == 1)
        spawnPosition = m_wallSensorR2.transform.position;
```

```
        else
            spawnPosition = m_wallSensorL2.transform.position;

        if (m_slideDust != null)
        {
            // Set correct arrow spawn position
            GameObject dust = Instantiate(m_slideDust, spawnPosition, gameOb-
            ject.transform.localRotation) as GameObject;
            // Turn arrow in correct direction
            dust.transform.localScale = new Vector3(m_facingDirection, 1, 1);
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        // if player enters collider tagged: fall detector
        if (collision.CompareTag("FallDetector") && !isRespawning)
        {
            StartCoroutine(HandleFall());
        }
        // if player enters checkpoint, replenish health, renew respawn position
        else if (collision.tag == "Checkpoint")
        {
            if (reachCheckpoint == false)
            {
                currentHealth = maxHealth;
            }
            reachCheckpoint = true;
            respawnPoint = transform.position + Vector3.up * 2f;
        }
        // if player enters boss1 (boss' weapon)
        else if (collision.gameObject.CompareTag("Boss1"))
        {
            // if player is not dead, play the hurt animation, take damage
            if (!isDead)
            {
                m_animator.SetTrigger("Hurt");
                TakeDamage(damageTaken);

                // get knocked back if not dead
                if (!isDead)
                {
                    Vector2 knockBackDirection = (transform.position - colli-
                    sion.transform.position).normalized;
                    rb.AddForce(knockBackDirection * knockbackForce, ForceMode2D.Im-
                    pulse);
                }
            }
        }
        // if player enters boss trigger renew respawn position inside the arena
        else if (collision != null && collision.CompareTag("BossTrigger"))
        {
            respawnPoint = collision.transform.position + new Vector3(2f, 0, 0);
        }
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
```

```
// if player is dead, return
if (isDead) return;

// if player hits the spikes or the base enemies, play the hurt animation and
take damage
if (collision.gameObject.CompareTag("Hazard"))
{
    m_animator.SetTrigger("Hurt");
    TakeDamage(damageTaken - 10);

    // get knocked back if not dead
    if (!isDead)
    {
        Vector2 knockBackDirection = (transform.position - collision.trans-
form.position).normalized;
        rb.AddForce(knockBackDirection * knockbackForce, ForceMode2D.Im-
pulse);
    }
}

// method to take damage
void TakeDamage(int damage)
{
    // if player is already taking damage or is dead, return
    if (!canTakeDamage || isDead) return;

    StartCoroutine(DamageCooldown());

    // if player is blocking, take half the damage
    if (m_animator.GetBool("Block") || m_animator.GetBool("IdleBlock")) {
        currentHealth -= damage / 2;
    }
    else currentHealth -= damage;

    // change health in the healthbar
    if (HealthBar.Instance != null)
    {
        HealthBar.Instance.SetHealth(currentHealth);
    }

    // if health <= 0, call the loselife method and play the death animation
    if (currentHealth <= 0)
    {
        LoseLife();
        isDead = true;

        m_body2d.linearVelocity = Vector2.zero;
        m_animator.SetBool("noBlood", m_noBlood);
        m_animator.SetTrigger("Death");
        StartCoroutine(RespawnAfterDelay());
    }
    // if health is not 0, play the hurt animation
    else
    {
        m_animator.SetTrigger("Hurt");
    }
}
```

```
// exists so that the player doesn't constantly take damage
IEnumerator DamageCooldown()
{
    canTakeDamage = false;
    yield return new WaitForSeconds(0.5f);
    canTakeDamage = true;
}

void LoseLife()
{
    // if player hasn't lost any lives, lose a life and disable the corresponding
    UI element
    if (livesLost == 0)
    {
        life2.SetActive(false);
        livesLost++;
    }
    // if player has lost 1 life, lose the other one, disable the corresponding
    UI element
    else if (livesLost == 1)
    {
        life1.SetActive(false);
        livesLost++;

        // player dies, the UI for the loss screen is activated
        isDead = true;

        PlayerPrefs.SetInt(SceneManager.GetActiveScene().name + "_checkpoint",
0);
        PlayerPrefs.Save();

        ActivateLoseScreen();

        // stops main music and plays the defeat music
        StopMusic();
        StartCoroutine(MusicDelay());
    }
}

// respawn after some seconds
IEnumerator RespawnAfterDelay()
{
    // if the player has lost all their lives, break
    if (livesLost >= 2)
    {
        ActivateLoseScreen();
        yield break;
    }

    yield return new WaitForSeconds(respawnDelay);

    // reset health, animator, isDead bool
    currentHealth = maxHealth;

    if (HealthBar.Instance != null)
    {
        HealthBar.Instance.SetHealth(currentHealth);
    }
}
```

```

        transform.position = respawnPoint;

        m_animator.Rebind();
        m_animator.Update(0f);

        m_animator.SetInteger("AnimState", 0);

        isDead = false;
    }

    // needed so that the player does not lose all their lives at once if they fall
    IEnumerator HandleFall()
    {
        isRespawning = true;

        Collider2D playerCollider = GetComponent<Collider2D>();
        playerCollider.enabled = false;

        LoseLife();

        if (livesLost >= 2)
        {
            isDead = true;
            if (m_body2d != null)
            {
                m_body2d.linearVelocity = Vector2.zero;
                m_body2d.bodyType = RigidbodyType2D.Static;
            }

            if (LoseScreenUI == null)
            {
                GameObject[] allObjects = Resources.FindObjectsOfTypeAll<GameOb-
ject>();
                foreach (GameObject obj in allObjects)
                {
                    if (obj.CompareTag("LoseScreen") && obj.scene.isLoaded)
                    {
                        LoseScreenUI = obj;
                        break;
                    }
                }
            }

            isRespawning = false;
            yield break;
        }

        currentHealth = maxHealth;

        if (HealthBar.Instance != null)
        {
            HealthBar.Instance.SetHealth(currentHealth);
        }

        transform.position = respawnPoint + Vector3.up * 2f;

        yield return new WaitForSeconds(0.2f);
    }

```

```
        if (playerCollider != null) playerCollider.enabled = true;
        isRespawning = false;
    }
    // stops music
    public void StopMusic()
    {
        AudioSource[] allAudioSources = FindObjectsByType<AudioSource>(FindOb-
jectsSortMode.None);

        foreach (AudioSource source in allAudioSources)
        {
            source.Stop();
        }
    }

    // enables defeat music after 1 second
    private IEnumerator MusicDelay()
    {
        yield return new WaitForSeconds(1);

        if (loseMusic == null)
        {
            GameObject[] allObjects = Resources.FindObjectsOfTypeAll<GameObject>();
            foreach (GameObject obj in allObjects)
            {
                if (obj.CompareTag("LoseMusic") && obj.scene.isLoaded)
                {
                    loseMusic = obj.GetComponent<AudioSource>();
                    break;
                }
            }
        }

        if (loseMusic != null)
        {
            loseMusic.enabled = true;
        }
    }

    // activates loss screen
    void ActivateLoseScreen()
    {
        if (LoseScreenUI == null)
        {
            Canvas mainCanvas = FindFirstObjectByType<Canvas>();
            if (mainCanvas != null)
            {
                Transform[] foundTransform = mainCanvas.GetComponentsInChil-
dren<Transform>(true);
                foreach (Transform t in foundTransform)
                {
                    if (t.name == "LoseScreen")
                    {
                        LoseScreenUI = t.gameObject;
                        LoseScreenUI.SetActive(false);
                        break;
                    }
                }
            }
        }
    }
}
```

```

        }
    }

    if (LoseScreenUI != null) LoseScreenUI.SetActive(true);
}

// resets the player
public void ResetPlayerOnRestart()
{
    isDead = false;
    isRespawning = false;

    currentHealth = maxHealth;
    livesLost = 0;

    life1 = null;
    life2 = null;

    if (m_animator != null)
    {
        m_animator.Rebind();
        m_animator.Update(0f);
        m_animator.SetInteger("AnimState", 0);
    }
}
}

```

10.10 Παράρτημα

```

using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LampCheckpoint : MonoBehaviour
{
    public GameObject glow;
    public GameObject lampLit;

    public GameObject life1, life2;

    private bool isActivated = false;
    [SerializeField] public AudioSource soundEffect;

    private HeroKnight player;

    // Start is called once before the first execution of Update after the MonoBe-
    haviour is created
    void Start()
    {
        lampLit.SetActive(false);
        glow.SetActive(false);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {

```

```

        if (isActive) return;

        // if the object's tag is "Player", then it calls the function Activate
        if (collision.tag == "Player")
        {
            player = collision.GetComponent<HeroKnight>();
            StartCoroutine(FlashPlayerGreen(player));
            Activate();
            PlayerPrefs.SetInt(SceneManager.GetActiveScene().name + "_checkpoint",
1);
            PlayerPrefs.Save();
        }
    }

    // lights up the lamp to indicate to the player they have reached the checkpoint
of the level
    void Activate()
    {
        isActive = true;
        glow.SetActive(true);
        lampLit.SetActive(true);

        if (life1.activeInHierarchy && !life2.activeInHierarchy) {
            life2.SetActive(true);
        }

        soundEffect.Play();
    }

    // the player turns green for a moment to indicate they have reached the check-
point
    private IEnumerator FlashPlayerGreen(HeroKnight player)
    {
        if (player != null)
        {
            SpriteRenderer sprite = player.GetComponent<SpriteRenderer>();
            if (sprite != null)
            {
                Color ogColor = sprite.color;

                sprite.color = Color.green;

                yield return new WaitForSeconds(0.3f);

                // reverts to original color
                if (sprite != null)
                {
                    sprite.color = ogColor;
                }
            }
        }
    }
}

```

10.11 Παράρτημα

```
using UnityEngine;
```

```
using UnityEngine.SceneManagement;

public class LoadLevels : MonoBehaviour
{
    [SerializeField] GameObject loadLevelsUI;

    [SerializeField] Bandit boss;
    private bool playerNearby = false;

    // Update is called every frame, if the MonoBehaviour is enabled
    private void Update()
    {
        /* unlocks the corresponding level (using player preferences) depending on:
        * - if the player is nearby
        * - if the user is pressing E
        */

        if (playerNearby && Input.GetKeyDown(KeyCode.E))
        {
            PlayerPrefs.SetInt(SceneManager.GetActiveScene().name + "_checkpoint",
0);
            PlayerPrefs.Save();

            if (SceneManager.GetActiveScene().name == "Level1")
            {
                if (PlayerPrefs.GetInt("UnlockedLevel") <= 1)
                {
                    PlayerPrefs.SetInt("UnlockedLevel", 2);
                    PlayerPrefs.Save();
                }
            }
            else if (SceneManager.GetActiveScene().name == "Level2")
            {
                if (PlayerPrefs.GetInt("UnlockedLevel") <= 2)
                {
                    PlayerPrefs.SetInt("UnlockedLevel", 3);
                    PlayerPrefs.Save();
                }
            }
            else if (SceneManager.GetActiveScene().name == "Level3")
            {
                if (PlayerPrefs.GetInt("UnlockedLevel") <= 3)
                {
                    PlayerPrefs.SetInt("UnlockedLevel", 4);
                    PlayerPrefs.Save();
                }
            }
            else if (SceneManager.GetActiveScene().name == "Level4")
            {
                if (PlayerPrefs.GetInt("UnlockedLevel") <= 4)
                {
                    PlayerPrefs.SetInt("UnlockedLevel", 5);
                    PlayerPrefs.Save();
                }
            }

            // loads level selection scene
            SceneManager.LoadScene("LevelSelection");
        }
    }
}
```

```

    }
}

private void OnTriggerEnter2D(Collider2D collision)
{
    // if the boss is destroyed or it has no health left and the object enter-
    // ing is the player,
    // it enables the UI and sets the playerNearby bool to true
    if (boss == null || boss.currentHealth == 0)
    {
        if (collision.CompareTag("Player"))
        {
            loadLevelsUI.SetActive(true);
            playerNearby = true;
        }
    }
}

private void OnTriggerExit2D(Collider2D collision)
{
    // when the player exits, disables the UI and sets the playerNearby bool to
    false
    if (collision.CompareTag("Player"))
    {
        if (loadLevelsUI != null)
        {
            loadLevelsUI.SetActive(false);
        }
        playerNearby = false;
    }
}
}

```

10.12 Παράρτημα

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class LoseMenu : MonoBehaviour
{
    public GameObject LoseUI;

    // after the player deis, when the corresponding button is pressed it loads the
    level selection scene
    public void GoToLevelSelection()
    {
        SceneManager.LoadScene("LevelSelection");
    }
}

```

10.13 Παράρτημα

```

using UnityEngine;
using UnityEngine.SceneManagement;

```

```
public class MainMenu : MonoBehaviour
{
    // UI elements
    [SerializeField] GameObject mainMenu;
    [SerializeField] GameObject title;
    [SerializeField] GameObject backButton;
    [SerializeField] GameObject optionsMenu;
    [SerializeField] GameObject controls;
    [SerializeField] GameObject difficulty;
    [SerializeField] GameObject progress;
    [SerializeField] GameObject successText;

    private bool pressedStart = false;
    private bool choseDifficulty => PlayerPrefs.GetInt("ChoseDifficulty", 0) == 1;
    // player preferences to change the difficulty (enemies' health)

    // loads the level selection scene after clicking the "Start" button on the main
    menu
    public void StartGame()
    {
        pressedStart = true;

        // if the player hasn't chosen a difficulty already, it forces them to do
        so, then it loads the scene
        if (!choseDifficulty)
        {
            mainMenu.SetActive(false);
            title.SetActive(false);
            difficulty.SetActive(true);

            PlayerPrefs.SetInt("ChoseDifficulty", 1);
            PlayerPrefs.Save();
        }
        else SceneManager.LoadScene("LevelSelection"); // if they have already cho-
        sen a difficulty it loads the scene straight away
    }

    // opens options menu
    public void Options()
    {
        mainMenu.SetActive(false);
        title.SetActive(false);
        backButton.SetActive(true);
        optionsMenu.SetActive(true);
    }

    // quits the application
    public void QuitGame()
    {
        Debug.Log("Quitting...");
        Application.Quit();
    }

    // when pressed, enables and disables the correct buttons and other UI elements
    public void Back()
    {
        if (optionsMenu.activeSelf == true)
        {

```

```
        optionsMenu.SetActive(false);
        backButton.SetActive(false);
        mainMenu.SetActive(true);
        title.SetActive(true);
    }

    if (controls.activeSelf == true)
    {
        controls.SetActive(false);
        optionsMenu.SetActive(true);
    }

    if (difficulty.activeSelf == true)
    {
        difficulty.SetActive(false);
        if (successText.activeSelf == true) successText.SetActive(false);
        optionsMenu.SetActive(true);
    }

    if (progress.activeSelf == true)
    {
        progress.SetActive(false);
        if (successText.activeSelf == true) successText.SetActive(false);
        optionsMenu.SetActive(true);
    }
}

// opens the controls UI
public void OpenControls()
{
    optionsMenu.SetActive(false);
    controls.SetActive(true);
}

// opens the difficulty UI
public void OpenDifficulty()
{
    optionsMenu.SetActive(false);
    difficulty.SetActive(true);
}

// opens the progress UI
public void OpenProgress()
{
    optionsMenu.SetActive(false);
    progress.SetActive(true);
}

// sets the difficulty to easy
public void SetDifficultyEasy()
{
    if (pressedStart)
    {
        SceneManager.LoadScene("LevelSelection");
    }
    else
    {
        PlayerPrefs.SetInt("Difficulty", 0);    // difficulty = 0 (Easy)
    }
}
```

```
        PlayerPrefs.Save();
        successText.SetActive(true);

        PlayerPrefs.SetInt("ChoseDifficulty", 1); // indicates that the player
has chosen difficulty so when pressing start the player doesn't have to do so again
        PlayerPrefs.Save();

        Invoke("Back", 2f);
    }
}

// sets the difficulty to normal
public void SetDifficultyNormal()
{
    if (pressedStart)
    {
        SceneManager.LoadScene("LevelSelection");
    }
    else
    {
        PlayerPrefs.SetInt("Difficulty", 1); // difficulty = 1 (Normal)
        PlayerPrefs.Save();
        successText.SetActive(true);

        PlayerPrefs.SetInt("ChoseDifficulty", 1); // indicates that the player
has chosen difficulty so when pressing start the player doesn't have to do so again
        PlayerPrefs.Save();

        Invoke("Back", 2f);
    }
}

// sets the difficulty to hard
public void SetDifficultyHard()
{
    if (pressedStart)
    {
        SceneManager.LoadScene("LevelSelection");
    }
    else
    {
        PlayerPrefs.SetInt("Difficulty", 2); // difficulty = 2 (Hard)
        PlayerPrefs.Save();
        successText.SetActive(true);

        PlayerPrefs.SetInt("ChoseDifficulty", 1); // indicates that the player
has chosen difficulty so when pressing start the player doesn't have to do so again
        PlayerPrefs.Save();

        Invoke("Back", 2f);
    }
}

// resets all player preferences (progress, choices, difficulty etc)
public void ResetProgress()
{
    PlayerPrefs.DeleteAll();
    PlayerPrefs.Save();
}
```

```

        successText.SetActive(true);

        Invoke("Back", 2f);
    }
}

```

10.14 Παράρτημα

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public static bool Paused = false;

    // UI elements
    [SerializeField] GameObject pauseMenuUI;
    [SerializeField] GameObject pauseImage;
    [SerializeField] GameObject backButton;
    [SerializeField] GameObject controlsButton;
    [SerializeField] GameObject controls;

    // Update is called once per frame
    void Update()
    {
        // if the games is not paused and the user presses the Escape key, it
        // pauses, if already paused, it continues
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (Paused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    // resets to default game speed, resumes music and re-enables camera rotation
    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        pauseImage.SetActive(false);
        Time.timeScale = 1.0f;
        Paused = false;

        AudioListener.pause = false;
    }

    // pauses music and sets game speed to zero so everythinng stops, disables cam-
    // era rotation
    void Pause()
    {
        pauseMenuUI.SetActive(true);
        pauseImage.SetActive(true);
    }
}

```

```

        Time.timeScale = 0f;
        Paused = true;

        AudioListener.pause = true;
    }

    // opens options menu
    public void Options()
    {
        pauseMenuUI.SetActive(false);
        controlsButton.SetActive(true);
        backButton.SetActive(true);
    }

    // shows controls
    public void Controls()
    {
        controlsButton.SetActive(false);
        controls.SetActive(true);
    }

    // enables and disables the correct UI elements depending on which ones are ac-
    tive
    public void Back()
    {
        if (controls.activeSelf)
        {
            controls.SetActive(false);
            controlsButton.SetActive(true);
        }
        else if (controlsButton.activeSelf)
        {
            controlsButton.SetActive(false);
            backButton.SetActive(false);
            pauseMenuUI.SetActive(true);
        }
    }

    // quits the application
    public void GoToMainMenu()
    {
        // resets if the player has reached the checkpoint
        PlayerPrefs.SetInt(SceneManager.GetActiveScene().name + "_checkpoint", 0);
        PlayerPrefs.Save();

        Time.timeScale = 1.0f;
        Paused = false;
        AudioListener.pause = false;
        SceneManager.LoadScene("StartingMenu");
    }
}

```

10.15 Παράρτημα

```

using UnityEngine;

public class SceneSpawnPoint : MonoBehaviour

```

```

{
    [Header("Settings")]
    [Tooltip("Write scene name from which the player comes from")]
    public string comingFromScene;

    // Start is called once before the first execution of Update after the MonoBe-
    haviour is created
    void Start()
    {
        // for entering and exiting the House, House2 scenes
        if (HeroKnight.Instance != null)
        {
            string previousScene = PlayerPrefs.GetString("PreviousScene", "");

            if (previousScene == comingFromScene)
            {
                HeroKnight.Instance.transform.position = this.transform.position;

                PlayerPrefs.SetString("PreviousScene", "");
                PlayerPrefs.Save();
            }
        }
    }
}

```

10.16 Παράρτημα

```

using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class SelectLevel:MonoBehaviour
{
    public GameObject[] buttons;

    // Start is called just before any of the Update methods is called for the first
    time
    void Start()
    {
        // enables the level button depending on how many levels have been unlocked
        (using player preferences)
        int unlockedLevel = PlayerPrefs.GetInt("UnlockedLevel");

        for (int i = 1; i < buttons.Length; i++)
        {
            bool isUnlocked = i < unlockedLevel;

            if (isUnlocked)
            {
                buttons[i].SetActive(true);
            }
            else
            {
                buttons[i].SetActive(false);
            }
        }
    }
}

```

// the following methods load the corresponding level when the one of the level buttons has been pressed

```
public void LoadLevel1()
{
    PlayerPrefs.SetInt("SaveLivesLost", 0);
    PlayerPrefs.Save();
    SceneManager.LoadScene("Level1");
}

public void LoadLevel2()
{
    PlayerPrefs.SetInt("SaveLivesLost", 0);
    PlayerPrefs.Save();
    SceneManager.LoadScene("Level2");
}

public void LoadLevel3()
{
    PlayerPrefs.SetInt("SaveLivesLost", 0);
    PlayerPrefs.Save();
    SceneManager.LoadScene("Level3");
}

public void LoadLevel4()
{
    PlayerPrefs.SetInt("SaveLivesLost", 0);
    PlayerPrefs.Save();
    SceneManager.LoadScene("Level4");
}

public void LoadLevel5()
{
    PlayerPrefs.SetInt("SaveLivesLost", 0);
    PlayerPrefs.Save();
    SceneManager.LoadScene("Level5");
}

// loads the starting scene
public void BackToStartMenu()
{
    SceneManager.LoadScene("StartingMenu");
}
}
```