i

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

“Σχεδιασμός και Ανάπτυξη top-down video game, με χρήση της Unity game engine και της γλώσσας C#”

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ: Μανωλάκης Κυριάκος

ΑΜ:0585

ΕΠΙΒΛΕΠΩΝ: Σωτήρης Χριστοδούλου

ΑΝΤΙΡΡΙΟ 2017

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Αντίρριο, 2017,

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Εισηγητής: Χριστοδούλου Σωτήρης, Υπογραφή

2. Κίτσος Παρασκευάς, Υπογραφή

3. Παρασκευάς Μιχάλης, Υπογραφή

Περίληψη

Το αντικείμενο της παρούσας πτυχιακής εργασίας ήταν ο σχεδιασμός και η ανάπτυξη ενός top-down 2D roleplaying video game με χρήση της μηχανής παιχνιδιού Unity και της γλώσσας προγραμματισμού C#. Σκοπός της εργασίας είναι η εξοικείωση των φοιτητών με την λογική του αντικειμενοστραφούς προγραμματισμού καθώς και τη διαδικασία της σύνθεσης ενός πλήρους ηλεκτρονικού παιχνιδιού υπολογιστή, η οποία συμπεριλαμβάνει την δημιουργία C# scripts, τον χειρισμό του προγράμματος Unity editor και την επεξεργασία ήχου και εικόνας.

Επιπλέον προγράμματα που χρησιμοποιήθηκαν: Visual Studio (IDE), Tiled (tile editor), Tiled to Unity.

Επιπλέον προγράμματα που χρειάστηκαν στην πορεία: Audacity (επεξεργασία ήχου), Photoshop (επεξεργασία εικόνας), Gitshell (command console για αποθήκευση στο Github).

Βοηθητικά site που χρησιμοποιήθηκαν: Trello (οργάνωση/σχεδιασμός), Github (backups, development monitoring).

Επίσης, χρησιμοποιήθηκαν assets με ελεύθερη άδεια (εκτός της μουσικής που απαγορεύει εμπορική χρήση).

Abstract

The aim of this thesis was to design and develop a top-down 2D roleplaying video game using the Unity game engine and the C# programming language. Its scope is the familiarization of students with the philosophy of object-oriented programming, as well as the process of manufacturing a fully featured computer video game, which consists of the creation of scripts in C# and the operation of the Unity editor, including graphics and sound processing.

Additional software that was used: Visual Studio (IDE), Tiled (tile editor), Tiled to Unity.

Additional software that was needed in the process: Audacity (audio processing), Photoshop (image processing), Gitshell (command console for Github repository managing).

Helpful sites which were used: Trello (organizing the project), Github (backups, development monitoring).

Includes free-licensed assets (besides music assets, which restrict commercial use).

Πίνακας Περιεχομένων

Εισαγωγή

Στην σύγχρονη εποχή, η κατασκευή video games αποτελεί μια βιομηχανία πολλών δισεκατομμυρίων ευρώ, μεγαλύτερη και από την βιομηχανία ταινιών του Hollywood, χάρις την γρήγορη άνοδο τους σε δημοτικότητα ως μέσο ψυχαγωγίας. Ένα χαρακτηριστικό, πρόσφατο παράδειγμα είναι το "Fallout 4" της εταιρίας Bethesda, η οποία εισέπραξε 750 εκατομμύρια δολάρια μόλις τις πρώτες 24 ώρες της κυκλοφορίας του.

Πέρα από την άνοδο της τελευταίας τεχνολογίας video games, όπως με το παραπάνω παράδειγμα, τα "retro/indie" ("παλαιάς εποχής") video games κάθε είδους έχουν επανεμφανιστεί στην αγορά με υψηλή ζήτηση και είναι ιδιαίτερα διαδεδομένα σε χρήστες κινητών τηλεφώνων αλλά και PC.

Ένας λόγος της επανεμφάνισης τους είναι η εύκολη κατασκευή τους, σε σχέση με το παρελθόν, δυνατή ακόμη και από ένα άτομο με μέτριες προγραμματιστικές γνώσεις, με editors που αναλαμβάνουν το ρόλο της "game engine" (middleware), η οποία θα εξηγηθεί παρακάτω στον "ρόλο της Unity game engine".

Top-down 2D video games

Η παρούσα πτυχιακή ασχολείται με την κατασκευή ενός 2D top-down ("από πάνω" οπτική γωνία), RPG (role-playing game - περεταίρω χαρακτηρισμός είδους game), παρόμοιο με το αρκετά γνωστό video game, Legend of Zelda.

Άλλες συνήθεις κατηγορίες ενός top-down game είναι τα action/shooter, puzzle, tower-defense, strategy κ.α.

Κύρια χαρακτηριστικά του συγκεκριμένου και βήματα κατασκευής του, θα εξηγηθούν παρακάτω. Ο λόγος της επιλογής του συγκεκριμένου είδους θα εξηγηθεί στο Q&A.

Επίσης κάποιες χρήσιμες ορολογίες που θα χρειαστούν και περιγράφουν μόνο τα 2D γραφικά γενικότερα είναι:

Tiles: Χρησιμοποιείται για στατικά γραφικά π.χ. το περιβάλλον, το έδαφος και άλλα.

Sprites: Η απεικόνιση ενός αντικειμένου. Ένα σύνολο sprites (το οποίο ονομάζεται και "spritesheet") - χρησιμοποιείται για γραφικά που θα "κινούνται" (καρέ-καρέ) προκειμένου να δημιουργηθούν τα animations – η αίσθηση της κίνησης (αντίθετα με τα 3D animations που κατά κύριο λόγο χρησιμοποιούνε 3D models που κινούνται με physics στον χώρο).

Συνοπτική περιγραφή του παιχνιδιού

Ο χρήστης χειρίζεται έναν παίκτη και μπορεί να επιτεθεί και να κινηθεί στον χώρο σε οποιαδήποτε κατεύθυνση. Ο σκοπός του παίκτη είναι να επιβιώσει και να προστατέψει το σπίτι του για 3 μέρες (μέρα ~ 4 λεπτά πραγματικού χρόνου). Ο παίκτης μπορεί να πάρει όπλα και αντικείμενα που τον βοηθάνε σε αυτόν τον σκοπό. Εμφανίζονται 5 διαφορετικά είδη εχθρών κατά την διάρκεια του παιχνιδιού με διαφορετικούς συνδυασμούς κίνησης (όπως περιγράφεται και αργότερα στην ανάλυση των scripts).

Ο ρόλος της Unity game engine

Το πρώτο βήμα κατασκευής ενός video game περιλαμβάνει την κατασκευή της game engine. Η game engine είναι το σύνολο του λογισμικού που αναλαμβάνει βασικές λειτουργίες μέσα στο παιχνίδι, όπως την απεικόνιση γραφικών (renderer), παραγωγή ήχου, μηχανή φυσικής/φυσικών νόμων (physics), animation (όταν πρόκειται για 3D), διαχείριση πόρων του υπολογιστή του χρήστη από το πρόγραμμα και προσαρμογή του λογισμικού του παιχνιδιού για διαφορετικά λειτουργικά συστήματα (build compatibility).

Συνήθως, η κατασκευή μιας game engine, ακόμα και για ένα απλό παιχνίδι και ειδικά για ένα 3D παιχνίδι τελευταίας τεχνολογίας, χρειάζεται τεράστιους πόρους, όσο αφορά τον χρόνο και το ανθρώπινο δυναμικό. Για αυτόν τον λόγο ακόμα και μεγάλες εταιρίες, προτιμούν να χρησιμοποιούν την ίδια game engine για πολλαπλά παιχνίδια που κατασκευάζουν εφόσον αυτή δεν είναι outdated σε πολύ μεγάλο βαθμό σε σχέση με τις καινούριες τεχνολογίες.

Ο editor της Unity περιλαμβάνει όλες αυτές τις λειτουργίες και ενσωματώνει τις περισσότερες σε ένα interface, κάνοντας προσιτή την κατασκευή video games σε μικρότερες εταιρίες και απλούς χρήστες.

Πέρα από την διαδικασία που ακολουθεί ο δημιουργός ενός παιχνιδιού όσο αφορά τον editor, καλείται να γράψει τον κώδικα που αποτελεί την λογική του παιχνιδιού σε ένα σύνολο script που επικοινωνούν μεταξύ τους. Η Unity υποστηρίζει C#, Unityscript (μορφή javascript προσαρμοσμένο σε αυτή) και Boo. Η πιο διαδεδομένη γλώσσα χρήσης της και με το περισσότερο online documentation είναι η C#.

Μέσω της Unity έχει κατασκευαστεί πληθώρα επιτυχημένων εμπορικών παιχνιδιών, (πολλά από τα οποία βρίσκονται και στην πλατφόρμα/διακομιστή “Steam), όπως το 7 days to die, Rust, Firewatch, Pillars of Eternity, Cities Skylines, Kerbal space program, το πολύ δημοφιλές “Pokemon GO”, το Angry birds και πολλά άλλα.

Η ίδια η Unity υποστηρίζει windows, mac και linux (σε πειραματικό βαθμό) για την χρήση της, ενώ τα παιχνίδια που κατασκευάζονται με αυτή μπορούν να εξαχθούν και να λειτουργήσουν σε 27 πλατφόρμες από Windows και Linux μέχρι Android και δημοφιλείς κονσόλες της Nintendo, της Sony και της Microsoft.

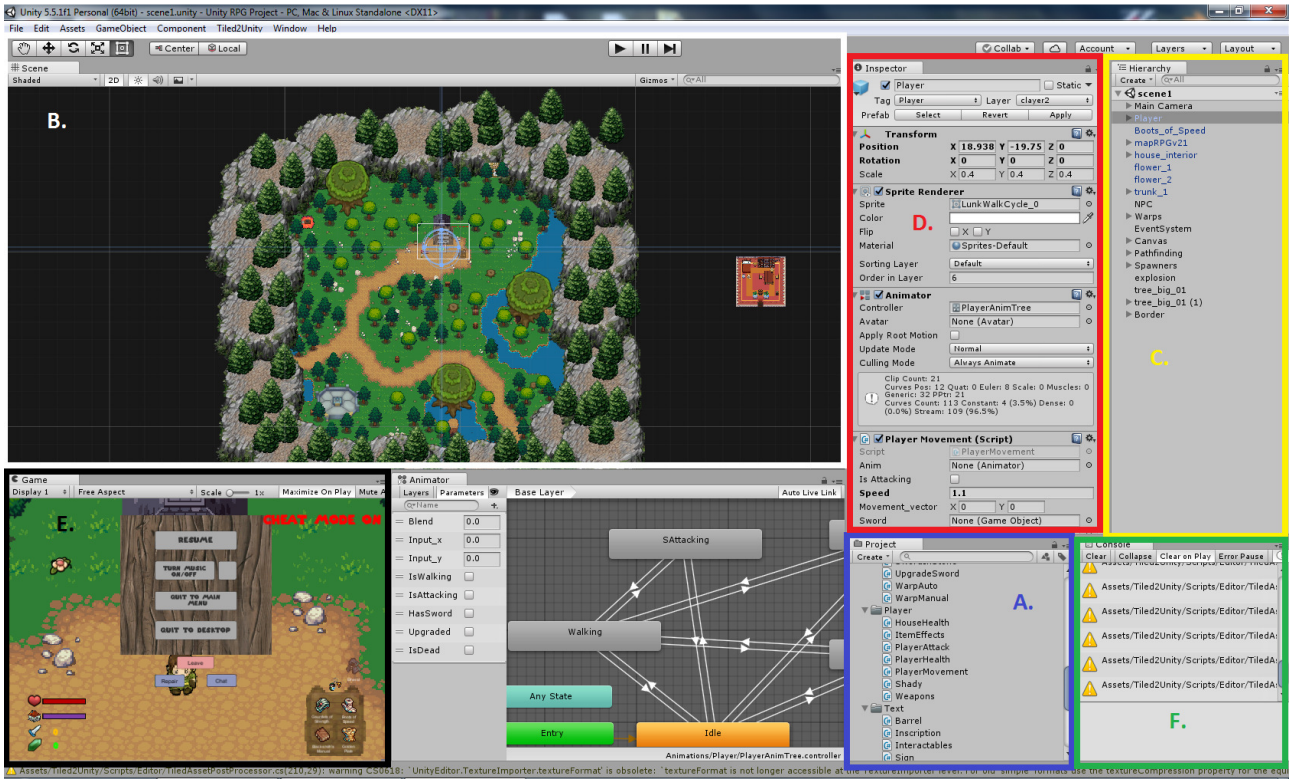
Η Unity, μέχρι πρόσφατα, υποστήριζε επίσημα μόνο την κατασκευή 3D παιχνιδιών. Αυτό άλλαξε στην 4.6v. Επίσης υποστηρίζει και την κατασκευή παιχνιδιών εικονικής πραγματικότητας.

Αξίζει και να σημειωθεί ότι οι χρήστες, μέσω της βάσης δεδομένων του Unity store, έχουν τη δυνατότητα να κατεβάσουν δωρεάν, να αγοράσουν ή και να πουλήσουν διάφορα assets σε άλλους χρήστες, συμπεριλαμβάνοντας έτοιμα plugins ή scripts και γραφικά.

Τέλος υπάρχουν πολλοί editors σαν την Unity, με διαφορετικές δυνατότητες και χαρακτηριστικά ο καθένας, π.χ. η Unreal Game Engine, η οποία χρησιμοποιεί C++ και εναλλακτικά μια μηχανή λογικών καταστάσεων για την χρήση της οποίας δεν είναι απαραίτητη η γνώση συγκεκριμένης γλώσσας προγραμματισμού.

Κύρια χαρακτηριστικά της Unity

Τα κύρια παράθυρα που χρησιμοποιούνται μέσα στον editor (για οποιοδήποτε project):



(Σημειώνεται ότι το interface και τα παράθυρα της Unity είναι πλήρως customizable αναλόγως των προτιμήσεων του χρήστη.)

A. **Assets/Project:** Εδώ αποθηκεύονται τα assets (οτιδήποτε θέλουμε να χρησιμοποιήσουμε για το παιχνίδι – γραφικά/εικόνες, ήχος, animations, scripts κ.α.).

B. **Scene:** Ένας αρχικά άδειος χώρος, στον οποίο τοποθετούνται τα αντικείμενα, κυρίως γραφικά, π.χ. ένα σπίτι (εικόνα σπιτιού που έχουμε κατεβάσει και προσθέσαμε στα assets). Μπορούν να τοποθετηθούν και άλλα αντικείμενα, ανάλογα με τις ανάγκες μας, π.χ. η κάμερα που θα ορίσει την οπτική γωνία/απόσταση (default αντικείμενο της Unity), ή ένα άδειο αντικείμενο (χωρίς γραφικό) που έχει μέσα π.χ. ένα στοιχείο (ιδιότητα του αντικειμένου) της φυσικής μηχανής ή απλά χρησιμοποιείται ως συντεταγμένες για να ενεργοποιήσουμε ένα γεγονός σε εκείνο το σημείο μέσω script). Ένα παιχνίδι μπορεί να περιέχει πολλές σκηνές, οι οποίες συνήθως χρησιμοποιούνται για levels σε ένα παιχνίδι.

C. **Hierarchy:** Το σύνολο/λίστα των αντικειμένων που έχει τοποθετηθεί στην σκηνή.

D. **Inspector:** Το παράθυρο το οποίο δείχνει τα στοιχεία και τις επιλογές του επιλεγμένου αντικειμένου από την ιεραρχία. Τα στοιχεία θα εξηγηθούν αργότερα, ωστόσο, όπως αναφέρθηκε και νωρίτερα, είναι, στην ουσία οι ιδιότητες/χαρακτηριστικά ενός αντικειμένου.

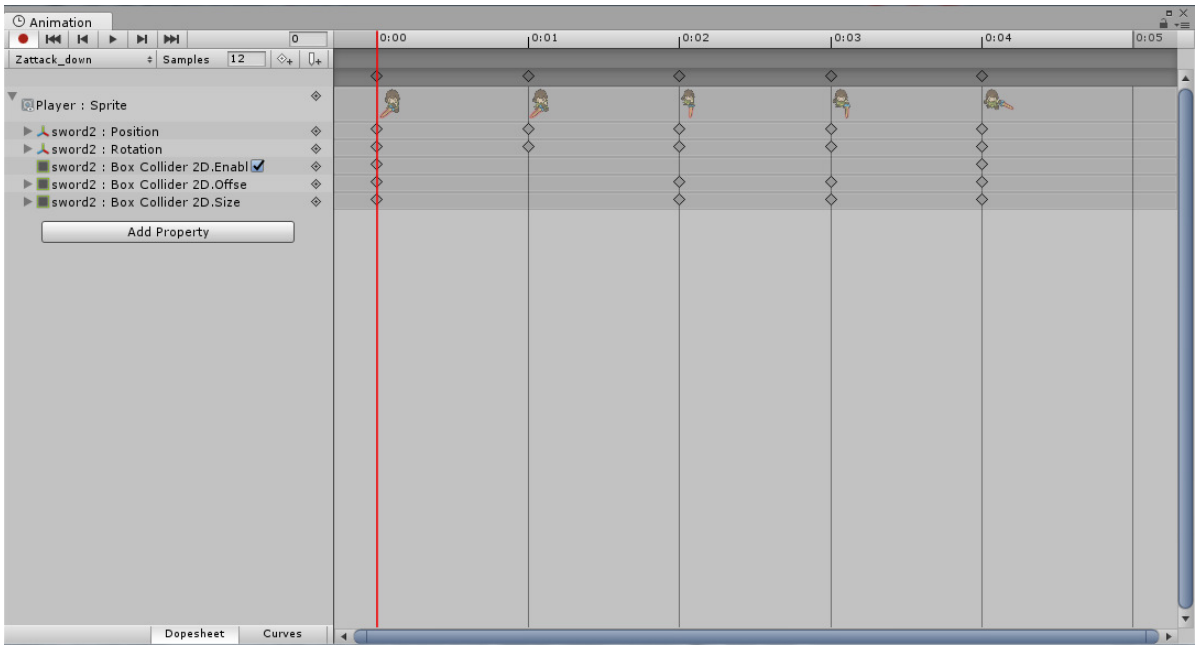
E. **Game preview:** Το παράθυρο που επιτρέπει στον χρήστη να παίξει το παιχνίδι μέσα στον editor χωρίς να χρειαστεί δημιουργήσει executable. Ιδιαίτερα χρήσιμο για debugging, καθώς ο χρήστης μπορεί συγχρόνως να βλέπει τον editor και να παρακολουθεί την μεταβολή κάθε στοιχείου ή μεταβλητής script σε πραγματικό χρόνο.

F. **Debug console:** Το παράθυρο που δείχνει τα errors. Με διπλό κλικ δείχνει σε ποιο script και line βρίσκεται το error (αν είναι compilation error).

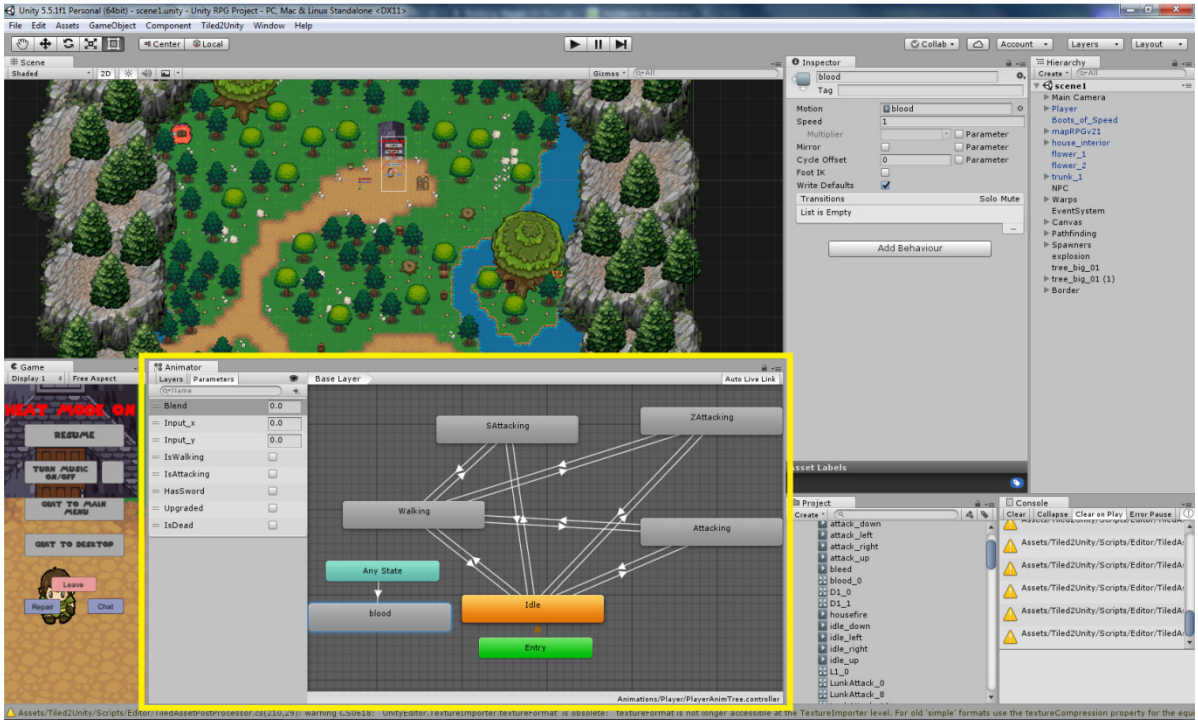
Άλλα σημαντικά εργαλεία (ή παράθυρα) της Unity που χρησιμοποιήθηκαν για την συγκεκριμένη πτυχιακή είναι:

Sprite editor: Συνήθως, τα 2D assets, έρχονται σε μορφή spritesheets (μια εικόνα με πολλά sprites μαζί), και αυτό το εργαλείο απομονώνει το κάθε sprite ώστε να φτιαχτεί ένα σύνολο ξεχωριστών sprites προκειμένου να συνθέσουμε το animation.

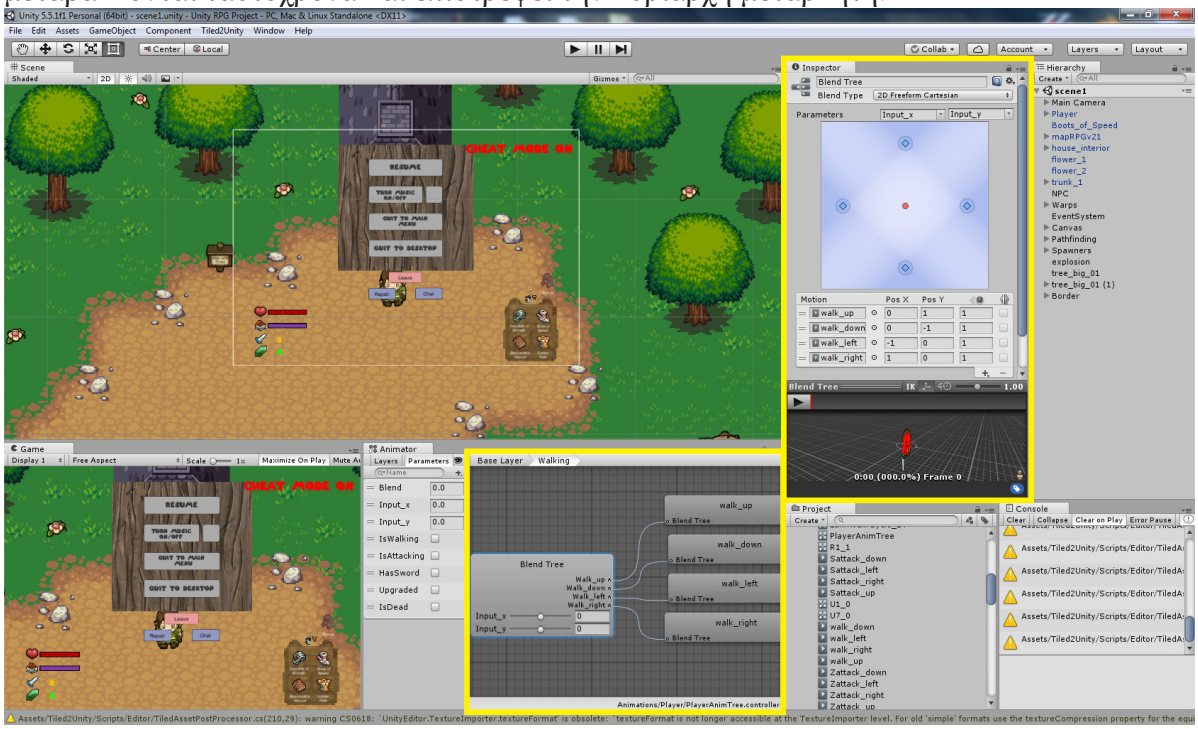
Animation: Εδώ κατασκευάζονται και επεξεργάζονται τα animations χρησιμοποιώντας sprites (μεταβολή σε κάθε καρέ, μήκος καρέ, μεταβολή στοιχείων και μεταβλητών των στοιχείων ενός αντικειμένου σε κάθε καρέ κ.α.).



Animator: Μια στρωματοποιημένη μηχανή λογικών καταστάσεων με μεταβάσεις μεταξύ τους που γίνονται βάσει αλλαγών παραμέτρων (συνθηκών) που δηλώνονται μέσω κώδικα. Π.χ. Έστω ότι έχουμε 2 διαφορετικά animations για 1) περπάτημα, 2) τρέξιμο, ενός αντικειμένου. Προσθέτουμε τα animations στον animator ως states (καταστάσεις), τα ενώνουμε με μια παράμετρο (π.χ. boolean) ως συνθήκη αλλαγής, αλλάζουμε την παράμετρο μέσω κώδικα όταν θελήσουμε (π.χ. εάν πατήσουμε ένα κουμπί η συνθήκη γίνεται αληθής) και έτσι γίνεται η μετάβαση μεταξύ τους.



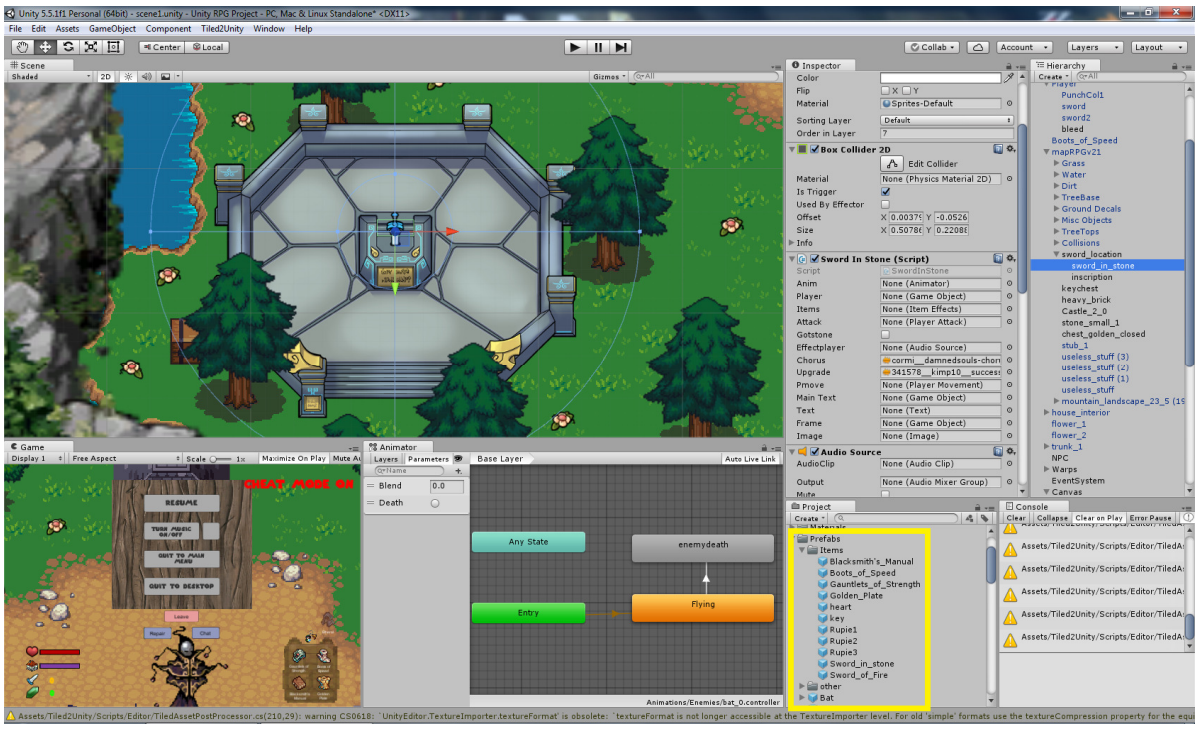
Ένα υπό-στοιχείο που μπορεί να χρησιμοποιηθεί σε κατώτερο στρώμα μίας κατάστασης του animator είναι το Blend Tree το οποίο περιέχει συνδυασμό παραμέτρων που μπορεί να μεταβάλλονται ταυτόχρονα και επιστρέφει την κυρίαρχη μεταβλητή.



Για παράδειγμα, στην συγκεκριμένη πτυχιακή το blend tree, χρησιμοποιήθηκε για την επιλογή του κατάλληλου animation, βάσει της οριζόντιας (x) και κάθετης (y) κατεύθυνσης της κίνησης του παίκτη.

Η μορφή του αρχείου στο οποίο αποθηκεύεται αυτή η μηχανή καταστάσεων για ένα animation ενός αντικειμένου λέγεται animation controller.

Prefabs: Μορφή αντικειμένων. Γενικότερα, προσθέτουμε αντικείμενα στην σκηνή τοποθετώντας είτε default αντικείμενα ή κάποιο asset που έχουμε συλλέξει σε αυτή. Αν κάνουμε το αντίστροφο, δηλαδή, τοποθετήσουμε αντικείμενα τα οποία είναι ήδη στην σκηνή (και που μπορεί να τους έχουμε αλλάξει τις ρυθμίσεις ή να τους έχουμε προσθέσει στοιχεία), στο παράθυρο των asset μας θα δημιουργηθούν τα prefabs.



Το prefab είναι στην ουσία το blueprint (“καλούπι”) ενός αντικειμένου με τις ρυθμίσεις και τα στοιχεία που είχε όταν το φτιάξαμε και χρησιμοποιείται για:

- Όταν θέλουμε να προσθέσουμε ένα αντικείμενο πολλές φορές στη σκηνή με custom ρυθμίσεις και στοιχεία που έχουμε προσθέσει - σέρνουμε το prefab από το παράθυρο των assets στην σκηνή (π.χ. κατεβάζουμε το γραφικό ενός βαρελιού, το προσθέτουμε στην σκηνή, αλλάζουμε το μέγεθος, το αποθηκεύουμε σαν prefab και μετά προσθέτουμε το prefab του βαρελιού στην σκηνή όσες φορές θέλουμε, χωρίς να χρειάζεται κάθε φορά να αλλάζουμε το μέγεθος).
- Όταν θέλουμε να δημιουργήσουμε το στιγμιότυπο του prefab μέσω κώδικα. (Π.χ. Οι εχθροί στο παιχνίδι, είναι αποθηκευμένοι ως prefabs και δημιουργούνται κατά τη διάρκεια του παιχνιδιού μέσω του spawning script).

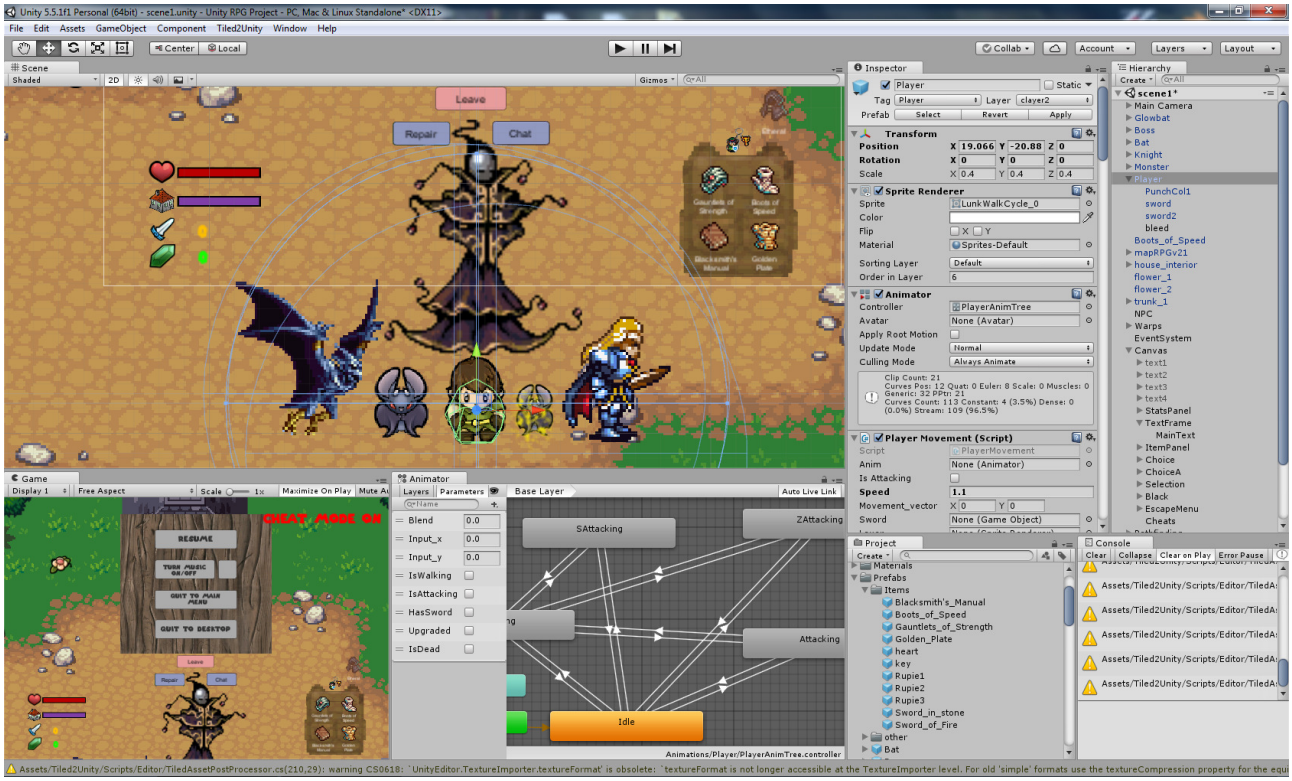
Tags: Τα tags είναι "ταμπέλες" που δημιουργεί ο χρήστης τις οποίες δίνει σε κάποιο αντικείμενο. Είναι ιδιαίτερα χρήσιμα για να βρίσκουμε το αντικείμενο μέσω κώδικα και να το αντιστοιχίσουμε σε κάποια μεταβλητή. Σημειώνεται ότι τα αντικείμενα μπορούν να βρεθούν και μέσω ονόματος.

Render Layer: Επιλογή στον sprite renderer. Το επίπεδο απεικόνισης ενός γραφικού. Γραφικά σε ανώτερο επίπεδο (sorting order) καλύπτουν αυτά σε κατώτερο. Π.χ. τα δέντρα αποτελούνται από δύο διαφορετικά sprite. Ο κορμός είναι σε κατώτερο επίπεδο από τον παίκτη ώστε να δίνεται η εντύπωση πως ο παίκτης είναι μπροστά από τον κορμό και τα φύλλα σε ανώτερο επίπεδο από αυτόν για να δίνεται η εντύπωση ότι ο παίκτης είναι πίσω τους.

Animated Components/Colliders: Μέσω του Animation window μπορούμε να προσθέσουμε, όπως αναφέρθηκε πιο πάνω, μεταβαλλόμενα στοιχεία ενός αντικειμένου ανάμεσα στα καρέ τους. Για παράδειγμα, στο animation επίθεσης του παίκτη, στο $1^{ο}$ καρέ, το σπαθί και ο collider του σπαθιού είναι απενεργοποιημένα. Στο δεύτερο ενεργοποιούνται, στα επόμενα μετακινούνται και στο τελευταίο καρέ απενεργοποιούνται ξανά.

Αντικείμενα και τα στοιχεία τους

Όπως αναφέρθηκε νωρίτερα, το σύνολο των αντικειμένων (GameObject) που υπάρχουν στην σκηνή εμφανίζεται στο παράθυρο της ιεραρχίας για την γρήγορη εύρεση και επεξεργασία τους. Διαλέγοντας ένα αντικείμενο από την ιεραρχία, εμφανίζονται τα στοιχεία του (components) στον Inspector με ότι επιλογές περιλαμβάνει το καθένα.



Ένα αντικείμενο μπορεί να περιέχει πολλαπλά στοιχεία, ανάλογα με την λειτουργικότητά του. Το βασικό στοιχείο κάθε αντικειμένου είναι το "Transform" το οποίο περιλαμβάνει όλες τις πληροφορίες του αντικειμένου σε σχέση με τον τοποθεσία/χώρο του στην σκηνή (μέγεθος, συντεταγμένες, περιστροφή κ.α.), ή το "Rect Transform" εάν είναι graphical user interface element (επιλογές in-game, μενού, στατιστικά, score κ.α.).

Ο χρήστης μπορεί να προσθέσει στοιχεία σε ένα αντικείμενο ανά πάσα στιγμή.

Η ποικιλία των στοιχείων είναι πολύ μεγάλη και παρακάτω θα αναφερθούν τα κύρια στοιχεία που χρησιμοποιήθηκαν για την συγκεκριμένη πρακτική.

-Transform/Rect Transform: Όπως ήδη αναφέρθηκε, είναι το στοιχείο χώρου του αντικειμένου.

-Renderer: Π.χ. "sprite renderer". Η λειτουργία απεικόνισης του στοιχείου (αν έχει, δηλαδή το αν θα φαίνεται ή όχι).

-Physics: Ότι έχει να κάνει με φυσική και φυσικούς νόμους. Δύο βασικά στοιχεία physics που χρησιμοποιούνται είναι τα Colliders2D και το Rigidbody2D (το 2D στο τέλος κάποιων στοιχείων είναι απλά το στοιχείο προσαρμοσμένο από την Unity για 2D παιχνίδια).

- **Colliders2D:** Ένα πλαίσιο το οποίο ορίζει τον χώρο διαδραστικότητας του αντικειμένου σε σχέση με άλλα αντικείμενα. Υπολογίζει την προσκόλληση των αντικειμένων (π.χ. Το εάν ένα αντικείμενο μπορεί να περάσει μέσα από ένα άλλο).

- **Colliders2D as triggers:** Τα colliders μπορούν επίσης να οριστούν στον Inspector ως triggers, απενεργοποιώντας την λειτουργία φυσικής προσκόλλησης, με μόνο σκοπό τον εντοπισμό του πότε δύο αντικείμενα έρχονται σε επαφή. Μέσω κώδικα, χρησιμοποιούμε αυτόν τον εντοπισμό για να δημιουργήσουμε "γεγονότα" που συμβαίνουν όταν γίνεται αυτό. Ένα παράδειγμα χρήσης τους είναι τα colliders του παίκτη και των εχθρών. Όταν έρχονται σε επαφή στο παιχνίδι, μέσω scripts, αφαιρούμε "health" από τον παίκτη και όταν το health του φτάσει στο μηδέν, ο χρήστης χάνει το παιχνίδι.

- **Rigidbody2D:** Η φυσική υπόσταση ενός αντικειμένου. Δίνει στο αντικείμενο μάζα και υπολογίζει την κίνηση του βάσει φυσικών νόμων. Μέσω script μπορούμε να κινήσουμε ένα αντικείμενο εφαρμόζοντας δύναμη (με μια απλή εντολή), εφόσον έχει μάζα.

-Animator: Η λειτουργία του animation του στοιχείου, στο οποίο προσθέτουμε τον animation controller που αντιστοιχεί στο αντικείμενο.

-Audio Source/Listener: Το προσθέτουμε εάν το αντικείμενο είναι πηγή/δέκτης ήχου.

-Script: Το αρχείο κειμένου γλώσσας προγραμματισμού που περιέχει τον κώδικα που ορίζει την συμπεριφορά του, τις παραμέτρους και την επικοινωνία του με άλλα αντικείμενα. Όταν δημιουργήσουμε ένα script πρέπει να το προσθέσουμε σε ένα αντικείμενο για να λειτουργήσει.

Ένα ολοκληρωμένο έργο της Unity περιέχει ένα σύνολο scripts, το καθένα από τα οποία προσκολλούνται από τον χρήστη στο αντικείμενο που θέλει. Ένα αντικείμενο μπορεί να χρησιμοποιήσει πολλαπλά scripts ανάλογα με τις λειτουργίες του. Για παράδειγμα, στην παρούσα πτυχιακή, το αντικείμενο που αναπαριστά τον παίκτη, περιέχει ένα script για κίνηση, ένα για επίθεση, ένα για την "ζωή" και ένα για την αλληλεπίδραση του με άλλα αντικείμενα.

Layers: Δεν έχει σχέση με το render layer. Τα layers είναι ένα χρήσιμο εργαλείο που καθορίζουν τα επίπεδα της διαδραστικότητας των αντικειμένων στην σκηνή. Όταν, για παράδειγμα, οι δύο αντικείμενα βρίσκονται σε διαφορετικό επίπεδο physics και έχουν και τα δύο Colliders δεν συγκρούονται (περνάει το ένα μέσα από το άλλο). Η χρήση των layers δεν ήταν απαραίτητη

Γενική λογική C# στην Unity

Η Unity μπορεί να χρησιμοποιήσει ένα σύνολο script, τα οποία αλληλεπιδρούν μεταξύ τους αλλάζοντας μεταβλητές.

Κατά τη δημιουργία ενός script, μετά τη δήλωση της κλάσης, η οποία πάντα πρέπει να έχει το ίδιο όνομα με το όνομα του script (και γίνεται αυτόματα), δηλώνονται όλες τις μεταβλητές που θέλουμε να χρησιμοποιήσουμε για το αντικείμενο στο οποίο θα προσκολλήσουμε το script.

Ένα script μπορεί να περιέχει πολλές κλάσεις, καθώς και υποκλάσεις οι οποίες κληρονομούν ιδιότητες από άλλες κλάσεις, άλλων, του ίδιου script ή του πηγαίου κώδικα της Unity. Στην παρούσα πτυχιακή χρησιμοποιήθηκε μια κλάση ανά script.

Σε κάθε script που δημιουργούμε υπάρχουν αρχικά δύο default συναρτήσεις, οι οποίες χρησιμοποιούνται για τις περισσότερες λειτουργίες των αντικειμένων. Η Start και η Update.

Ο κώδικας που εμπεριέχεται στην Start εκτελείται μόνο μία φορά, όταν αρχίζει το παιχνίδι (ή όταν εμφανίζεται το αντικείμενο που έχει το script στο παιχνίδι, αν δεν υπάρχει το αντικείμενο από την αρχή στην σκηνή).

Συνήθως στην Start, πέρα άλλων λειτουργιών που μπορεί να χρειαστεί να εκτελεστούν μια φορά μόνο, αρχικοποιούνται οι μεταβλητές που έχουμε δηλώσει, ή αρχικοποιούνται (αντιστοιχούνται) με ένα αντικείμενο ή στοιχείο αντικειμένου που υπάρχει ήδη στην σκηνή.

Η Update τις περισσότερες φορές περιέχει την πλειοψηφία του κώδικα ενός script και τον εκτελεί σε κάθε frame (καρέ), δηλαδή πολλές φορές το δευτερόλεπτο, ανάλογες με το frame rate, κατά την διάρκεια του παιχνιδιού.

Για αυτό το λόγο στην Update προσθέτουμε λειτουργίες με παραμέτρους που θέλουμε να αλλάζουν σε πραγματικό χρόνο όσο λειτουργεί το παιχνίδι. Για παράδειγμα, οι κινήσεις και οι αλληλεπιδράσεις των αντικειμένων.

Ένα loop που αλλάζει ένα στοιχείο, π.χ. ένα while που αλλάζει τις συνθήκες σε έναν animator ενός αντικειμένου, μέσα στην Update είναι αρκετό για να "κρασάρει" την Unity, εφόσον ακόμη και αν δεν είναι "αέναο".

Βασικός τρόπος δήλωσης και αρχικοποίησης των μεταβλητών

Όπως αναφέρθηκε νωρίτερα, το πρώτο πράγμα που κάνει ο χρήστης μετά την δημιουργία ενός script είναι η δήλωση των μεταβλητών που θα χρησιμοποιηθούν (η οποία φυσικά μπορεί να συμπληρωθεί και αργότερα αν ο χρήστης κρίνει ότι χρειάζεται περισσότερους μεταβλητές).

Πέρα από τους κλασικούς τύπους μεταβλητών (int, float, bool κτλπ) υπάρχουν και τύποι (κλάσεις) μεταβλητών της βιβλιοθήκης της Unity. Έστω ότι δηλώνουμε μια μεταβλητή στην οποία θέλουμε να αποθηκεύσουμε ένα αντικείμενο, το οποίο εκπροσωπεί τον παίκτη μας. Στην θέση του τύπου μεταβλητής, συμπληρώνουμε τον όρο τον οποίο η Unity χρησιμοποιεί για να αναφερθεί στα αντικείμενα (GameObject).

Η δήλωση του θα γίνει ως εξής:

- **public GameObject greathero;**

Έστω ότι δηλώνουμε και μια άλλη μεταβλητή στην οποία θέλουμε να αποθηκεύσουμε ένα στοιχείο του παίκτη μας – στην συγκεκριμένη περίπτωση το Transform (που αναφέρθηκε και νωρίτερα) του παίκτη. Ο τύπος δήλωσης θα είναι το όνομα του στοιχείου. Οπότε η δήλωση του θα γίνει ως εξής.

- **public Transform greatherolocation;**

Με αυτές τις δηλώσεις η Unity καταλαβαίνει ότι θέλουμε να δημιουργήσουμε μια μεταβλητή τύπου GameObject και μια μεταβλητή τύπου Trasform με τα αντίστοιχα ονόματα που τους δώσαμε, ωστόσο, εφόσον δεν τις έχουμε αρχικοποιήσει στη Start, δεν την έχει αντιστοιχίσει με τον παίκτη μας ή το στοιχείο του παίκτη μας.

Για να αντιστοιχιστούν οι μεταβλητές μας με το αντικείμενο που εκπροσωπεί τον παίκτη και το στοιχείο του παίκτη Transform, πρέπει να τις αρχικοποιήσουμε στην Start χρησιμοποιώντας συναρτήσεις από τις βιβλιοθήκες της Unity (περισσότερα για αυτές παρακάτω).

Για να αντιστοιχίσουμε το αντικείμενο που εκπροσωπεί τον παίκτη με την μεταβλητή greathero που φτιάξαμε νωρίτερα θα χρησιμοποιήσουμε την εξής συνάρτηση “Gameobject.Find()”, η οποία βρίσκει το αντικείμενο στη σκηνή χρησιμοποιώντας το όνομα του. Έστω ότι το όνομα του στη σκηνή (ιεραρχία) είναι “Player”.

- **greathero = GameObject.Find("Player");**

Τώρα που αντιστοιχίστηκε η μεταβλητή με το αντικείμενο που εκπροσωπεί τον παίκτη, πρέπει να αντιστοιχιστεί και το στοιχείου του αντικειμένου αυτού με την δεύτερη μεταβλητή. Θα χρησιμοποιηθεί μια άλλη συνάρτηση – “GetComponent<name>(), η οποία αντλεί στοιχεία από τα αντικείμενα τους.

- **greatherolocation = greathero.GetComponent<Transform>();**

-Το κομμάτι “greathero.” στην παραπάνω εντολή δεν είναι απαραίτητο εφόσον το script στο οποίο είναι η εντολή, είναι προσκολλημένο στο συγκεκριμένο αντικείμενο που περιέχει το στοιχείο.

Βιβλιοθήκες της Unity:

Η Unity διαθέτει κάποιες βιβλιοθήκες με έτοιμες συναρτήσεις/λειτουργίες που μπορούμε να χρησιμοποιήσουμε στον κώδικα. Όταν ο χρήστης θέλει να πετύχει κάτι συγκεκριμένο μέσω κώδικα, δεν έχει παρά να ανατρέξει στο manual της Unity (https://docs.unity3d.com/Manual/index.html) όπου παρέχονται όλες η συναρτήσεις με παραδείγματα.

Κάποια παραδείγματα συναρτήσεων/λειτουργιών που χρησιμοποιήθηκαν πολύ στην παρούσα πτυχιακή είναι:

- **greathero = GameObject.FindWithTag("Player");** - Εντοπίζει το αντικείμενο με το tag “Player” και το αντιστοιχεί στην μεταβλητή “greathero” που έχουμε δηλώσει στην αρχικοποίηση.

- **mycollider = GetComponent<Collider2D>();** - Εντοπίζει το στοιχείο του αντικειμένου (στην συγκεκριμένη περίπτωση το collider του), στο οποίο βρίσκεται το script που την περιέχει και το αποθηκεύει σε μια μεταβλητή. Οποιοδήποτε στοιχείο μπορεί να

αποθηκευτεί (π.χ. ένα άλλο script, ώστε μετά να μπορούμε να αλλάξουμε τις μεταβλητές του από το παρόν script – περισσότερα στην "ανάλυση δειγμάτων κώδικα").

- **void OnTriggerEnter2D(Collider2D other)** - Μια συνάρτηση η οποία εκτελείται μόνο την στιγμή που ένας άλλος collider ενός αντικειμένου στην σκηνή, έρθει σε επαφή με τον collider του αντικειμένου που έχει το script με αυτή.

- **void OnTriggerStay2D(Collider2D other)** - Σαν την παραπάνω συνάρτηση, που ωστόσο εκτελείται συνεχώς όσο οι colliders των αντικειμένων βρίσκονται σε επαφή.

Επίσης σημειώνεται ότι για να αναγνωριστούν συγκεκριμένες συναρτήσεις π.χ. για κατασκευή UI elements, πρέπει να προσθέσουμε την κατάλληλη βιβλιοθήκη στην αρχή του script.

Βασική επικοινωνία μεταξύ scripts:

Όπως αναφέρθηκε νωρίτερα, όταν θέλουμε να δηλώσουμε μια μεταβλητή για ένα στοιχείο, χρησιμοποιούμε τον τύπο του στοιχείου. Για τα scripts χρησιμοποιούμε το όνομα που τους δώσαμε.

Έστω ότι έχουμε έναν παίκτη που ονομάζεται mitsos στην σκηνή - του έχουμε προσθέσει collider, rigidbody για μάζα και ένα script για την ζωή του εν ονόματι "Health", το οποίο περιέχει μια μεταβλητή τύπου int που ονομάζεται currentHealth και έχει αρχικοποιηθεί με το νούμερο 100.

Τώρα, έστω ότι θέλουμε να φτιάξουμε μια παγίδα που να κάνει ζημιά στον παίκτη όταν την πατάει. Σέρνουμε το γραφικό της παγίδας στη σκηνή, προσθέτουμε collider που μετατρέπουμε σε trigger collider από τις επιλογές του, δημιουργούμε ένα script που ονομάζουμε "Trap" και το προσθέτουμε στο αντικείμενο της παγίδας στην ιεραρχία, μέσω του Inspector.

Εφόσον θέλουμε η παγίδα να κάνει ζημιά στην ζωή του παίκτη, ανοίγουμε το script Trap και δηλώνουμε μεταβλητή που θα εκπροσωπήσει το script Health.

- **public Health zwh;**

Για να βρούμε το στοιχείο/script Health στην σκηνή όμως, πρέπει πρώτα να βρούμε το αντικείμενο στο οποίο βρίσκεται, δηλαδή τον παίκτη και να τον αποθηκεύσουμε σε μια άλλη μεταβλητή (αφού πρώτα τη δηλώσουμε τη μεταβλητή ως GameObject). Θα τον βρούμε βάσει του ονόματος που του είχαμε δώσει. Στην Start γράφουμε:

- **player = GameObject.Find("mitsos");**

Τώρα, αφού έχουμε βρει το αντικείμενο του παίκτη, πρέπει να πάρουμε το στοιχείο του αντικειμένου, το Health script και να το αντιστοιχίσουμε με την μεταβλητή zwh.

- **zwh = mitsos.GetComponent<Health>();**

Εφόσον βρέθηκε το script component Health του mitsos και αποθηκεύτηκε στη "zwh", πρέπει να αλλάξουμε την μεταβλητή τύπου int, το currentHealth μέσα σε αυτό το script. Αναφερόμαστε σε αυτή τη μεταβλητή με αυτόν τον τρόπο (το script που ανήκει και το όνομα της μεταβλητής μετά από τελεία):

- **zwh.currentHealth**

Σημείωση – η μορφή “επίπεδο1”.”επίπεδο2”.”επίπεδο3” χρησιμοποιείται σε πολλές περιπτώσεις και εκτός script για να βρούμε αντικείμενα τα οποία ανήκουν σε κάποια ανώτερα επίπεδα/κλάσεις.

Στην συγκεκριμένη περίπτωση, η συνάρτηση void OnTriggerEnter2D είναι η κατάλληλη εφόσον θα ενεργοποιηθεί μόνο όταν έρθει σε επαφή ένας collider με τον collider της παγίδας.

Επίσης θα προσθέσουμε μέσα σε αυτή μια συνθήκη If, ώστε να ενεργοποιείται μόνο όταν ο collider του παίκτη, συγκεκριμένα, έρθει σε επαφή με τον collider της παγίδας.

```
void OnTriggerEnter2D(Collider2D other)
{
        if (other.gameObject == mitsos)
        {
        zwh.currentHealth = zwh.currentHealth
- 20
        }
}
```

Με αυτή τη διαδικασία, ο παίκτης θα χάνει 20 ζωή κάθε φορά που μπαίνει στην παγίδα.

Ανάλυση δειγμάτων κώδικα του παιχνιδιού

Το πρώτο script θα αναλυθεί εκτενώς, καθώς και οτιδήποτε άλλο θα παρουσιαστεί στα επόμενα script για πρώτη φορά. Υπόψη ότι ο κώδικας δεν είναι τέλειος και σίγουρα υπάρχουν πιο εύκολοι και λειτουργικοί τρόποι για να επιτευχθούν κάποιες λειτουργίες.

Σημείωση: Κάποιοι μεταβλητές στη δήλωση μπορεί να μην χρησιμοποιούνται. Επίσης κάποια κομμάτια κώδικα από δοκιμές κτλπ, μπορεί να μην έχουν σβηστεί και να είναι σε comments.

1. Script κίνησης παίκτη

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class PlayerMovement : MonoBehaviour
5.  {
6.
7.      Rigidbody2D body1;
8.      public Animator anim;
9.      //public Animation anim2;
10.     public bool isAttacking;
11.     public float speed = 1f;
12.     public Vector2 movement_vector;
13.
14.     public GameObject sword;
15.     public SpriteRenderer layer;
16.
17.     public GameObject player;
18.
19.     public float dirx;
20.     public float diry;
21.
22.     // Use this for initialization
23.     void Start()
24.     {
25.         player = GameObject.FindWithTag("Player");
26.         body1 = GetComponent<Rigidbody2D>();
27.         anim = player.GetComponent<Animator>();
28.
29.         //anim2 = GetComponent<Animation>();
30.     }
31.
32.     // Update is called once per frame
33.     void Update()
```

```
34.     {
35.         isAttacking = anim.GetBool("IsAttacking");
36.         movement_vector = new Vector2(Input.GetAxisRaw("Horizontal"),
37.         Input.GetAxisRaw("Vertical")).normalized;
38.         if (movement_vector != Vector2.zero)
39.         {
40.             anim.SetBool("IsWalking", true);
41.             anim.SetFloat("Input_x", movement_vector.x);
42.             anim.SetFloat("Input_y", movement_vector.y);
43.         }
44.         else
45.         {
46.             anim.SetBool("IsWalking", false);
47.         }
48.         //sword layer problem
49.
50.         diry = anim.GetFloat("Input_y");
51.         dirx = anim.GetFloat("Input_x");
52.         sword = GameObject.Find("sword");
53.
54.
55.         if (sword != null)
56.         {
57.             layer = sword.GetComponent<SpriteRenderer>();
58.             if (diry < 0)
59.             {
60.                 layer.sortingOrder = 6;
61.             }
62.             else layer.sortingOrder = 5;
63.         }
64.         if (isAttacking == false)
65.         {
66.             body1.MovePosition(body1.position + movement_vector * Time.deltaTime * 1.0F * speed);
67.         }
68.         else
69.         {
70.             body1.MovePosition(body1.position + movement_vector * Time.deltaTime * 0.5F * speed);
71.         }
72.     }
73. }
```

Το script κίνησης παίκτη που είναι προσκολλημένο στο αντικείμενο του παίκτη και μας επιτρέπει να κινούμε τον παίκτη με το input του χρήστη. Το script περιλαμβάνει και άλλες λειτουργίες οι οποίες προστέθηκαν αργότερα και θα εξηγηθούν παρακάτω.

1-2: Οι default βιβλιοθήκες της Unity.

4: Δήλωση κλάσης που περικλείει όλο το script. Η δομή των κλάσεων εξαρτάται από την λογική του προγραμματιστή για την οργάνωση των λειτουργιών του παιχνιδιού.

7-20: Δήλωση μεταβλητών. Πέρα από τους κλασικούς τύπους μεταβλητών, βλέπουμε και δήλωση μεταβλητών με τύπου στοιχείων/κλάσεων της Unity (GameObject, Spriterenderer, Animator, Rigidbody2D). Επίσης δηλώνεται ο τύπος Vector2D με τη μεταβλητή movement_vector που αρχικοποιείται στην Update και θα εξεταστεί παρακάτω **(36).**

Vectors: Ο Vector είναι ένα διάνυσμα το οποίο δηλώνει κατεύθυνση κίνησης και ισχύ.

Start:

25: Αντιστοιχείται η μεταβλητή player που έχουμε φτιάξει για τον παίκτη με το αντικείμενο του παίκτη στη σκηνή.

26: Αντλείται το στοιχείο Rigidbody2D (από τον παίκτη) και αντιστοιχείται με την μεταβλητή body1.

27: Αντλείται το στοιχείο Animator (από τον παίκτη και αντιστοιχείται με την μεταβλητή anim. Το "player." μπροστά από το "GetComponent" δεν είναι αναγκαίο εφόσον το script αυτό είναι προσκολλημένο στον παίκτη.

Update:

35: Εφόσον έχουμε αποθηκεύσει στην μεταβλητή anim το στοιχείο Animator του παίκτη, μπορούμε να αντλήσουμε περαιτέρω παραμέτρους μέσα σε αυτό. Στην συγκεκριμένη περίπτωση, αντλούμε μια συνθήκη/παράμετρο του Animator που φτιάξαμε στο interface της Unity, το "isAttacking", τύπου bool και το αποθηκεύουμε σε μια μεταβλητή που (τυχαία) έχει το ίδιο όνομα. Χρειαζόμαστε το "anim." μπροστά από την εντολή "GetBool" για να διευκρινίσουμε ότι θέλουμε την παράμετρο του συγκεκριμένου στοιχείου.

36: Εδώ αρχικοποιείται ο movement_vector σε έναν οριζόντιο και έναν κάθετο vector που δέχεται input. Τα inputs μπορούν να οριστούν από το interface ή να οριστούν με μια συνθήκη if μέσα στον κώδικα.

38-43: Κώδικας για τα animation. Εάν ο παίκτης κινείται (δηλαδή αν το διάνυσμα δεν είναι μηδέν), ενεργοποιούνται τα animation του της κίνησης. Στις **41 & 42**, η τιμή του x και y αντίστοιχα του movement_vector μπαίνει στην τιμή των παραμέτρων του animator "Input_x","Input_y" που έχουμε προσθέσει μέσω του interface στο blend tree της κατάστασης "IsWalking". Έτσι ενεργοποιούνται τα κατάλληλα animations κατεύθυνσης, αναλόγως την κατεύθυνση της κίνησης του παίκτη.

50-63: Ένα κομμάτι κώδικα που αλλάζει το επίπεδο απεικόνισης του σπαθιού, ανάλογα με την κατεύθυνση του παίκτη. Αρχικά εντοπίζει την κατεύθυνση και βάζει το σπαθί σε κατώτερο επίπεδο από τον παίκτη, όταν αυτός πρέπει να φαίνεται "πάνω από το σπαθί" (π.χ. όταν κοιτάει προς τα πάνω και πρέπει το κεφάλι του να καλύπτει μέρος του σπαθιού.

64-71: Συγκεκριμένα στην **66** βρίσκεται ο κώδικας που δίνει κίνηση στο αντικείμενο με τη συνάρτηση MovePosition, που κινεί το Rigidbody2D του παίκτη. Στην παρένθεση, προστίθεται το διάνυσμα movement_vector που επηρεάζεται από το input του χρήστη με την παρούσα τοποθεσία του αντικειμένου ανά πάσα στιγμή. Μετά πολλαπλασιάζεται με τον μια αυθαίρετη μεταβλητή (speed), την οποία χρησιμοποιούμε για να ρυθμίσουμε την ταχύτητα του παίκτη μέσα στο παιχνίδι ανάλογα με τις προτιμήσεις μας. Επίσης πολλαπλασιάζεται με το Time.deltaTime.

Time.deltaTime: Εφόσον ο κώδικας τρέχει την Update εκτελείται σε κάθε frame. Επειδή τα frames είναι ασταθή, όταν θέλουμε να πετύχουμε μια ομαλή μεταβολή, όπως την κίνηση ενός αντικειμένου χρησιμοποιούμε πολλαπλασιάζουμε με Time.deltaTime για να κάνουμε τον ρυθμό της μεταβολής εξαρτώμενο από τον χρόνο. Συγκεκριμένα η συνάρτηση αυτή είναι ο χρόνος που πέρασε από το προηγούμενο frame.

Ο σκοπός των υπολοίπων σειρών και της συνθήκης είναι να μειώσουν την ταχύτητα του παίκτη όσο επιτίθεται.

2. Script "ζωής" παίκτη

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using UnityEngine.UI;
4.
5.  public class PlayerHealth : MonoBehaviour {
6.
7.      public bool isDead;
8.      public bool isDamaged;
9.      public bool regen = false;
10.     public float maxHealth = 100f;
11.     public float currentHealth = 100f;
12.
13.     public PlayerMovement playerMovement;
14.     public GameObject player;
15.     //public float damagedtimer;
16.     //public float damagecd;
17.
18.     public AudioSource effectplayer;
19.     public AudioClip splats;
20.
21.     public Animator anim;
22.
23.     //public GameObject blood;
24.     //public Transform blood;
25.     public SpriteRenderer bleed;
26.     public float bleedtimer;
27.
28.     public GameObject house;
```

```
29.     public HouseHealth houseHealth;
30.     public GameObject cheats;
31.     public Text cheatnote;
32.     public AudioSource note;
33.     public AudioClip notecheat;
34.     public bool cheatmode = false;
35.
36.     // Use this for initialization
37.     void Start()
38.     {
39.         anim = GetComponent<Animator>();
40.         playerMovement = GetComponent<PlayerMovement>();
41.         currentHealth = maxHealth;
42.
43.         effectplayer = GetComponent<AudioSource>();
44.
45.         //blood = GameObject.Find("Player.bleed");
46.         //bleed = blood.GetComponent<SpriteRenderer>();
47.
48.         bleed = transform.Find("bleed").GetComponent<SpriteRenderer>();
49.
50.         player = GameObject.FindWithTag("Player");
51.
52.         house = GameObject.FindWithTag("HouseHitbox");
53.         houseHealth = house.GetComponent<HouseHealth>();
54.         cheats = GameObject.Find("Cheats");
55.         cheatnote = cheats.GetComponent<Text>();
56.         note = player.GetComponent<AudioSource>();
57.
58.         cheatnote.enabled = false;
59.     }
60.
61.     // Update is called once per frame
62.     void Update()
63.     {
64.         //damagedtimer += Time.deltaTime;
65.         bleedtimer += Time.deltaTime;
66.
67.
68.         if (regen == true)
69.         {
70.             currentHealth += Time.deltaTime * 2f;
71.             maxHealth = 150;
72.             if (cheatmode == true)
73.             {
```

```
74.                 currentHealth += Time.deltaTime * 20f;
75.             }
76.         }
77.         if (currentHealth > maxHealth)
78.         {
79.             currentHealth = maxHealth;
80.         }
81.
82.         if (bleedtimer > 0.8f)
83.         {
84.             bleed.enabled = false;
85.             bleedtimer = 0;
86.         }
87.
88.
89.         if (Input.GetKeyDown(KeyCode.Backspace))
90.         {
91.             cheatmode = true;
92.             note.clip = notecheat;
93.             note.Play();
94.         }
95.
96.         if (cheatmode == true)
97.         {
98.             houseHealth.currentHealth += Time.deltaTime * 80f;
99.             currentHealth += Time.deltaTime * 30f;
100.            cheatnote.enabled = true;
101.        }
102.    }
103.
104.    public void Damaged(int damage)
105.    {
106.        bleed.enabled = true;
107.        isDamaged = true;
108.        currentHealth -= damage;
109.        Debug.Log(currentHealth);
110.        if (currentHealth <= 0 && !isDead)
111.        {
112.            Death();
113.        }
114.    }
115.
116.    public void Death()
```

```
117.     {
118.         isDead = true;
119.         effectplayer.clip = splats;
120.         effectplayer.volume = 0.5f;
121.         effectplayer.Play();
122.         anim.SetBool("IsDead", true);
123.
124.         //   anim.SetTrigger("Dead");
125.         playerMovement.enabled = false;
126.         foreach (Collider2D c in GetComponents<Collider2D>())
127.         {
128.             c.enabled = false;
129.         }
130.         Invoke("GameoverP", 3);
131.         Destroy(gameObject,2);
132.     }
133.
134.     public void GameoverP()
135.     {
136.         Time.timeScale = 0;
137.     }
138. }
```

7-34: Η δήλωση των μεταβλητών. Το script αυτό, εκτός από την ζωή του παίκτη, περιέχει και κώδικα για sound/visual effects που αφορούν αυτή, για ένα αντικείμενο που την επηρεάζει, για το "cheat mode", καθώς και το για τι γίνεται όταν αυτή εξαντληθεί.

39-58: Η αρχικοποίηση των μεταβλητών. Στην γραμμή **48** σημειώνεται ότι χρησιμοποιείται ένας ιδιαίτερος τρόπος για να βρούμε το στοιχείο ενός child αντικειμένου στην σκηνή.

68-76: Η επίδραση του αντικειμένου στη ζωή του παίκτη. Η μεταβλητή regen μεταβάλλεται από το script "αλληλεπίδραση αντικειμένων" και γίνεται true όταν ο παίκτης έρχεται σε επαφή με το αντικείμενο.

77-80: Ο κώδικας αυτός δεν επιτρέπει να αυξηθεί η ζωή του παίκτη, πέρα από την μέγιστη ζωή που μπορεί να έχει, όταν αυτός παίρνει "καρδιές" από τους εχθρούς.

82-86: Το animation του τραυματισμού του παίκτη παίζει συνεχώς, αλλά η δουλειά του συγκεκριμένου κώδικα είναι να απενεργοποιεί τελείως το αντικείμενο bleed, μετά από λίγη ώρα, αφότου ενεργοποιηθεί από τη συνάρτηση Damaged, ώστε να μην φαίνεται.

89-102: Ο κώδικας του cheat mode.

104-114: Η συνάρτηση Damaged. Καλείται από script επίθεσης εχθρών, όπως θα δούμε στο επόμενο script και μπορεί να καλέσει τη συνάρτηση Death.

116-132: Ενεργοποιεί τα κατάλληλα visual/audio effects για την περίσταση, απενεργοποιεί την κίνηση και τα colliders του παίκτη και καταστρέφει το αντικείμενο. Επίσης στην σειρά **130** καλεί την συνάρτηση GameoverP μέσω της χρήσιμης συνάρτησης της Unity "**Invoke**", με την οποία μπορούμε να εκτελέσουμε κώδικα με καθυστέρηση.

3. Script επίθεσης νυχτερίδας

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class BatAttack : MonoBehaviour {
5.
6.      public int damage = 10;
7.      public int attackCooldown = 1;
8.
9.      public GameObject player;
10.     public GameObject house;
11.     public Health health;
12.     public PlayerHealth playerHealth;
13.     public HouseHealth houseHealth;
14.     public bool inRange;
15.     public bool inRange2;
16.     public float timer;
17.
18.
19.         // Use this for initialization
20.     void Start ()
21.     {
22.         player = GameObject.FindWithTag("Player");
23.         house = GameObject.FindWithTag("HouseHitbox");
24.         playerHealth = player.GetComponent<PlayerHealth>();
25.         houseHealth = house.GetComponent<HouseHealth>();
26.         health = GetComponent<Health>();
27.      }
28.
29.     void OnTriggerStay2D(Collider2D other)
30.     {
31.         if (other.gameObject == player)
32.         {
33.             inRange = true;
34.         }
35.         else if (other.gameObject == house)
36.         {
37.             inRange2 = true;
38.         }
39.     }
```

```
40.
41.     void OnTriggerExit2D(Collider2D other)
42.     {
43.         //if (other.gameObject == player)
44.         //{
45.             inRange = false;
46.         //}
47.         //else if (other.gameObject == house)
48.         //{
49.             inRange2 = false;
50.         //}
51.     }
52.
53.
54.     // Update is called once per frame
55.     void Update()
56.     {
57.         timer += Time.deltaTime;
58.         if (timer >= attackCooldown && inRange && health.currentHealth >=0)
59.         {
60.             AttackP();
61.         }
62.         else if (timer >= attackCooldown && inRange2 && health.currentHealth >= 0)
63.         {
64.             AttackH();
65.         }
66.     }
67.
68.     void AttackP()
69.     {
70.         timer = 0f;
71.         if (playerHealth.currentHealth > 0)
72.         {
73.             playerHealth.Damaged(damage);
74.         }
75.     }
76.     void AttackH()
77.     {
78.         timer = 0f;
79.         if (houseHealth.currentHealth > 0)
80.         {
81.             houseHealth.Damaged(damage);
```

```
82.             }
83.         }
84.     }
```

29-39: Χρησιμοποιούμε μια συνάρτηση OnTriggerStay μεταξύ του παίκτη και της νυχτερίδας για να καθορίσουμε πότε τα αντικείμενα έρχονται σε επαφή. Όταν έρχονται σε επαφή, η μεταβλητή inRange γίνεται αληθής. Σημειώνεται ότι η μεταβλητή inRange2 χρησιμοποιείται για το σπίτι με τον ίδιο τρόπο που η πρώτη χρησιμοποιείται για τον παίκτη.

55-66: Φτιάχνουμε μια συνθήκη που καθορίζει για το πότε θε εκτελεστεί η συνάρτηση της επίθεσης **(60 & 64)** και την τοποθετούμε στην Update γιατί θέλουμε να εκτελείται ανά πάσα στιγμή όταν η συνθήκη γίνεται αληθής. Εκτός του ότι το inRange πρέπει να είναι αληθές, χρησιμοποιούμε ένα timer **(57)** το οποίο μηδενίζεται κάθε φορά που εκτελείται η επίθεση, έτσι ώστε να καθορίσουμε το κάθε πόση ώρα μπορεί να επιτίθεται η νυχτερίδα. Αν δεν το καθορίσουμε αυτό θα επιτίθεται 60 φορές το δευτερόλεπτο – όσες φορές δηλαδή εκτελείται ο κώδικας μέσα στην Update. Τελευταία συνθήκη είναι η ζωή της νυχτερίδας να είναι θετική, ώστε να σταματάει να επιτίθεται όταν την σκοτώσει ο παίκτης.

68-75 (& 76-84): Η συνάρτηση της επίθεσης που καλείται απο την Update. Έχει μια συνθήκη που απλά κοιτάζει εάν ο παίκτης/σπίτι είναι ζωντανός μέσω της μεταβλητής currentHealth στο script ζωής του παίκτη και μετά καλεί την συνάρτηση Damaged που είδαμε σε αυτό.

4. Script κίνησης ιππότη (και τέρατος)

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using System.Linq;
5.
6.  public class KnightMovement : MonoBehaviour
7.  {
8.
9.      public float speed;
10.
11.     //public GameObject[] waypoints;
12.     public List<GameObject> waypoints;
13.     public GameObject closest = null;
14.     public Transform closestPosition;
15.
16.     public GameObject player;
17.     public GameObject house;
18.     public Transform playerpos;
19.     //public Transform playerpos2;
20.     public Transform housepos;
21.     //public float playerposY;
```

```
22.
23.     GameObject finalwaypoint;
24.     Transform stop;
25.
26.     //Vector3 direction;
27.     public float attackCooldown;
28.     //public float attackCooldownM;
29.     public float timerCooldown;
30.     public float timer;
31.     public float timerMax;
32.     public int rand;
33.     float animtimer;
34.     GameObject[] collisions;
35.
36.     public Animator anim;
37.     Rigidbody2D body;
38.     Vector2 knightvector;
39.     int x;
40.     int y;
41.
42.     public float aggroRange;
43.     public float aggroRange2;
44.     public float attackRange;
45.     Transform knightpos;
46.
47.     Vector3 prevPos;
48.
49.     public bool IsAttacking = false;
50.
51.     //public Transform target;
52.     public float xDir;
53.     public float yDir;
54.
55.     private SpriteRenderer sprite;
56.
57.     // Use this for initialization
58.     void Start()
59.     {
60.
61.         player = GameObject.FindWithTag("Player");
62.         house = GameObject.FindWithTag("House");
63.
64.         //playerposY = playerpos.position.y+1;
65.         //playerpos2 = new Vector3(x,y,z);
66.
```

```
        knightpos = GetComponent<Transform>();
        playerpos = player.GetComponent<Transform>();
        housepos = house.GetComponent<Transform>();


        //waypoints = GameObject.FindGameObjectsWithTag("Waypoint");

        waypoints = GameObject.FindGameObjectsWithTag("Waypoint").ToList();
        finalwaypoint = GameObject.Find("C");
        stop = finalwaypoint.GetComponent<Transform>();

        body = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();

        collisions = GameObject.FindGameObjectsWithTag("GenCollisions");

        sprite = this.GetComponent<SpriteRenderer>();
    }


    // Update is called once per frame
    void Update()
    {
        //knight layer
        if (gameObject.name == "Knight(Clone)")
        {
            if (knightpos.position.y > playerpos.position.y)
            {
                sprite.sortingOrder = 5;
            }
            else sprite.sortingOrder = 7;
        }


        /*
            knightvector = (new Vector2(x, y)).normalized;
            body.velocity = knightvector;
            anim.SetFloat("x", knightvector.x);
            anim.SetFloat("y", knightvector.y);
            */

        //target = GameObject.Find("Player").transform;

        animtimer += Time.deltaTime;

        if (Vector2.Distance(knightpos.position, playerpos.position) < aggroRange)
        {
            xDir = playerpos.position.x - transform.position.x;
```

```
115.            yDir = playerpos.position.y - transform.position.y;
116.            if (animtimer > 0.5)
117.            {
118.                anim.SetFloat("x", xDir);
119.                anim.SetFloat("y", yDir);
120.            }
121.        }
122.        else if (Vector2.Distance(knightpos.position, housepos.position) <= attackRange + 0.5)
123.        {
124.            xDir = housepos.position.x - transform.position.x;
125.            yDir = housepos.position.y - transform.position.y;
126.            if (animtimer > 2)
127.            {
128.                anim.SetFloat("x", xDir);
129.                anim.SetFloat("y", yDir);
130.                animtimer = 0;
131.            }
132.        }
133.        else
134.        {
135.            knightvector = transform.position - prevPos;
136.            anim.SetFloat("x", knightvector.x);
137.            anim.SetFloat("y", knightvector.y);
138.            if (animtimer > 0.5)
139.            {
140.                prevPos = transform.position;
141.                animtimer = 0;
142.            }
143.        }
144.
145.        if ((Vector2.Distance(knightpos.position, playerpos.position) <= attackRange && timerCooldown >
146.          attackCooldown) || (Vector2.Distance(knightpos.position, housepos.position) <= attackRange +
147.          0.6f && timerCooldown > attackCooldown))
148.        {
149.            anim.SetBool("IsAttacking", true);
150.            timerCooldown = 0;
151.        }
152.        else anim.SetBool("IsAttacking", false);
153.
154.        timerCooldown += Time.deltaTime;
155.
156.        if (Vector2.Distance(knightpos.position, playerpos.position) < aggroRange &&
```

```
157.        Vector2.Distance(knightpos.position, playerpos.position) > attackRange)
158.    {
159.        waypoints.Clear();
160.        closest = null;
161.        closestPosition = null;
162.
163.        if (gameObject.name == "Knight(Clone)")
164.        {
165.            transform.position = Vector2.MoveTowards(transform.position, playerpos.position –
166.            playerpos.up * 0.3f, speed * Time.deltaTime);
167.        }
168.
169.        else if (gameObject.name == "Monster(Clone)")
170.        {
171.            transform.position = Vector2.MoveTowards(transform.position, playerpos.position +
172.            playerpos.up * 0.3f, speed * Time.deltaTime);
173.        }
174.
175.    }
176.    else if (Vector2.Distance(knightpos.position, housepos.position) < aggroRange2)
177.    {
178.        transform.position = Vector2.MoveTowards(transform.position, housepos.position, speed *
179.        Time.deltaTime);
180.    }
181.    else if (Vector2.Distance(knightpos.position, playerpos.position) > aggroRange)
182.    {
183.
184.        closestWaypoint();
185.        if (closest != null)
186.        {
187.            closestPosition = closest.GetComponent<Transform>();
188.            transform.position = Vector2.MoveTowards(transform.position, closestPosition.position,
189.            speed * Time.deltaTime);
190.            if (closestPosition.position == transform.position)
191.            {
192.                waypoints.Remove(closest);
193.            }
194.            if (transform.position == stop.position)
195.            {
196.                waypoints.Clear();
197.                //closestPosition.position = Vector2.MoveTowards(transform.position,
198.                  playerpos.position, speed * Time.deltaTime);
```

```
199.            }
200.        }
201.
202.        else
203.        {
204.            if (timer > timerMax)// && transform.position == stop.position)
205.            {
206.                rand = Random.Range(1, 9);
207.                timerMax = Random.Range(1, 3);
208.                timer = 0;
209.            }
210.            timer += Time.deltaTime;
211.            switch (rand)
212.            {
213.                case 1:
214.                    transform.Translate(Vector2.up * speed * Time.deltaTime);
215.                    break;
216.                case 2:
217.                    transform.Translate(-Vector2.up * speed * Time.deltaTime);
218.                    break;
219.                case 3:
220.                    transform.Translate(Vector2.right * speed * Time.deltaTime);
221.                    break;
222.                case 4:
223.                    transform.Translate(-Vector2.right * speed * Time.deltaTime);
224.                    break;
225.                case 5:
226.                    transform.Translate((Vector2.up + Vector2.right).normalized * speed *
227.                    Time.deltaTime);
228.                    break;
229.                case 6:
230.                    transform.Translate((Vector2.up + Vector2.left).normalized * speed *
231.                    Time.deltaTime);
232.                    break;
233.                case 7:
234.                    transform.Translate((Vector2.down - Vector2.right).normalized * speed *
235.                    Time.deltaTime);
236.                    break;
237.                case 8:
238.                    transform.Translate((Vector2.down - Vector2.left).normalized * speed *
239.                    Time.deltaTime);
```

```
240.                     break;
241.                 }
242.             }
243.         }
244.         else return;
245.
246.     }
247.
248.     public GameObject closestWaypoint()
249.     {
250.         float distance = Mathf.Infinity;
251.         Vector3 position = transform.position;
252.         foreach (GameObject destination in waypoints)
253.         {
254.             Vector3 diff = destination.transform.position - position;
255.             float curDistance = diff.sqrMagnitude;
256.             if (curDistance < distance)
257.             {
258.                 closest = destination;
259.                 distance = curDistance;
260.             }
261.         }
262.         return closest;
263.     }
264.
265.     public void OnCollisionStay2D(Collision2D other)
266.     {
267.         foreach (GameObject collision in collisions)
268.         {
269.             if (other.gameObject == collision)
270.             {
271.                 rand = Random.Range(1, 9);
272.             }
273.         }
274.     }
275. }
```

AI κίνησης ιππότη/τέρατος - Η κίνηση του ιππότη έχει ως εξής: Αφότου δημιουργείται στην σκηνή (script No.8) πρέπει να ακολουθήσει το αντίστοιχο μονοπάτι και να φτάσει κοντά στο σπίτι. Εάν ο παίκτης/σπίτι είναι αρκετά κοντά, μετακινείται προς αυτά για να τους επιτεθεί και αγνοεί το μονοπάτι. Εάν ο παίκτης έρθει αρκετά κοντά και μετά "ξεφύγει", θα αρχίσει να κάνει τυχαία κίνηση "ψάχνοντας" τον παίκτη. Το τέρας και ο ιππότης χρησιμοποιούν το ίδιο script κίνησης. Ότι αφορά τον ιππότη αφορά και το τέρας στην επεξήγηση αυτή.

12: Για να επιτευχθεί η κίνηση του ιππότη στο μονοπάτι χρησιμοποιήθηκε μια λίστα, στην οποία μπήκαν τα waypoints στην σειρά **74**. Τα waypoints είναι απλά άδεια αντικείμενα τα οποία δημιουργήθηκαν για να χρησιμοποιηθούν οι συντεταγμένες τους.

Σημειώνεται ότι οι περισσότερες μεταβλητές στην δήλωση δημιουργήθηκαν εκ των υστέρων, επειδή χρειάστηκαν από την λογική της κίνησης στην πορεία.

88-98: Το κομμάτι αυτό, αλλάζει το Render Layer του ιππότη, ώστε να είναι κατώτερο από αυτό του παίκτη όταν ο παίκτης είναι κάτω από αυτόν στην σκηνή και ανώτερο όταν είναι πάνω από αυτόν, προκειμένου να υπάρξει μια ψευδαίσθηση τριών διαστάσεων.

112-143: Ο συγκεκριμένος κώδικας ρυθμίζει την κίνηση και τα animations για να έχουν ένα μικρό delay όταν αλλάζουν. Αυτό έγινε στην πορεία, επειδή όταν ο ιππότης προχωρούσε διαγώνια προς έναν στόχο, άλλαζε μέσα σε millisecond μεταξύ των animation π.χ. της δεξιάς κίνησης και της πάνω κίνησης, όταν προχωρούσε διαγώνια πάνω δεξιά. Εμπειρικά, ο κώδικας αυτός ήταν απαραίτητος για να τα ομαλά animations κίνησης των εχθρών με διαφορετικά animations για κάθε κατεύθυνση κίνησης (μόνο ο ιππότης στο συγκεκριμένο παιχνίδι), αλλά είναι αρκετά σίγουρο ότι αυτά μπορούν να επιτευχθούν με πιο απλό τρόπο.

Vector2.Distance(x,y): Μια συνάρτηση που υπολογίζει το διάνυσμα της απόστασης μεταξύ του x και y.

145-152: Ο κώδικας αυτός καθορίζει (με μια απαράδεκτα μεγάλη συνθήκη if) το πότε ο ιππότης θα επιτεθεί σε σχέση με την κίνηση του. Η επίθεση του ιππότη είναι αρκετά διαφορετική από αυτή της νυχτερίδας. Η νυχτερίδα κάνει ζημιά στον παίκτη με την επαφή μόνο, ενώ ο ιππότης επιτίθεται και με σπαθί. Ο ιππότης έχει και αυτός ένα script που καλεί την Damaged του παίκτη (προσκολλημένο στο ίδιο το σπαθί), αλλά ο συγκεκριμένος κώδικας γράφτηκε εδώ γιατί πρέπει η επίθεση να είναι σχετική της απόστασης του απο το αντικείμενο. Το attackRange χρησιμοποιείται για να μην κολαέι ο ιππότης στον παίκτη και να σταματάει για να επιτεθεί με το σπαθί σε μια μικρή απόσταση κοντά του.

156-162: Συνθήκη που στην ουσία λέει ότι εάν ο παίκτης βρίσκεται κοντά στον ιππότη, η λίστα με τα waypoints σβήνει και ο ιππότης ακολουθεί αυτόν.

163-173: (Εμφωλευμένη της παραπάνω). Στην σειρά **165 & 166** βρίσκεται ο κώδικας που κινεί τον ιππότη προς τον παίκτη (171 & 172 για το τέρας). Επιτυγχάνεται με διαφορετικό τρόπο από αυτόν που κινείται ο παίκτης για λόγους οι οποίοι θα εξηγηθούν στο τέλος του εγγράφου. Χρησιμοποιείται η μεταβολή του στοιχείου χώρου τους, **Transform**, παράμετρος της οποίας είναι το **position** (θέση στον χώρο).

Οπότε η συνάρτηση είναι: **trasform.position = Vector2.MoveTowards(x,y,z)**

Σαν το **Vector2.Distance(x,y)** που υπολογίζει απόσταση, το **Vector2.MoveTowards(x,y,z)** δηλώνει ένα διάνυσμα μεταξύ δύο αντικειμένων (x,y) με μία δράση (την κατεύθυνση ενός προς το άλλο), που χρειάζεται και ισχύ (z).

Σαν x θα μπει η τωρινή θέση στον χώρο του αντικειμένου (transform.position). Σαν y θα μπει η θέση του παίκτη (playerpos.position). Η ισχύς καθορίζεται από τον χρήστη και όπως εξηγήθηκε και παραπάνω, ομαλοποιείται πολλαπλασιάζοντας με το Time.deltaTime.

Η μόνη διαφορά κίνησης μεταξύ του τέρατος και του ιππότη είναι η σχετική τους θέση στον άξονα y με τον παίκτη, καθώς το "κέντρο" του sprite τους δεν βρισκόταν στην καλύτερη δυνατή θέση. Γιαυτό και αντί να μπει σκέτο playerpos.position στο x, το αφαιρούμε και με playerpos.up * 0.3f στην περίπτωση του ιππότη για να είναι στην ίδια ευθεία τα sprites, όταν επιτίθεται από πλάγια.

176-180: Ο κώδικας που κινεί τον ιππότη προς το σπίτι. Δεν χρειάζεται να σβηστούν τα waypoints επειδή το σπίτι βρίσκεται έτσι και αλλιώς στο τέλος των waypoints και μόλις ο ιππότης φτάσει στο τελευταίο waypoint σβήνονται τα προηγούμενα.

181: Η συνθήκη για όταν ο ιππότης δεν είναι κοντά στον παίκτη. Περικλείει τον υπόλοιπο κώδικα που μένει στην Update.

248-261: Πριν εξηγηθεί ο υπόλοιπος κώδικας της update, θα εξηγηθεί αυτή η συνάρτηση που καλείται στην σειρά **184** στην Update. Υπολογίζει το κοντινότερο waypoint στον ιππότη ανά πάσα στιγμή. Αποθηκεύουμε την θέση του ιππότη σε ένα Vector3 διάνυσμα (Vector3 και όχι Vector2 από πειραμαστισμό, γιατί ο editor έβγαζε error στην παρακάτω συνθήκη σύγκρισης).

Μετά χρησιμοποιούμε την εντολή foreach **(252)** – για κάθε στοιχείο x, μέσα στη λίστα που είχαμε φτιάξει και φτιάχνουμε ένα διάνυσμα τύπου Vector3 "diff", που δηλώνει την απόσταση του ιππότη από το κάθε destination/waypoint μέσα στη λίστα ανα πάσα στιγμή και το αποθηκεύουμε σε ένα float curDistance. Το sqrMagnitude που χρησιμοποιήθηκε δεν είναι απαραίτητο (είναι απλά πιο γρήγορο σε τέτοιους real-time υπολογισμούς). Έχουμε αποθηκεύσει μια μεταβλητή distance που δηλώνει το άπειρο, για placeholder, όταν ακόμα αυτή δεν έχει αντικατασταθεί με το curDistance **(259)**. Αποθηκεύουμε στην 259 το curDistance στο distance έτσι ώστε κάθε φορά, για κάθε destination, να συγκρίνουμε κάθε παρόν curDistance με το προηγούμενο curDistance στην if στη σειρά **256**. Με αυτό τον τρόπο βρίσκουμε το destination, για το οποίο το curDistance είναι μικρότερο τη συγκεκριμένη στιγμή και το αποθηκεύουμε στη μεταβλητή closest που επιστρέφει η συνάρτηση.

Αντί αυτών, θα μπορούσαμε, αν δεν είχαμε δύο μονοπάτια και οι ιππότες δεν έβγαιναν τυχαία από το κάθε μονοπάτι, να τους πούμε να ακολουθήσουν κάποιες απλές συντεταγμένες στον χάρτη με τη σειρά.

185-200: Με το αυτό το κομμάτι, εφόσον υπάρχει το closest waypoint, ο ιππότης κινείται κάθε φορά στο κοντινότερο waypoint, σβήνοντας από τη λίστα του όποιο waypoint επισκέπτεται (192). Εάν φτάσει στο τελευταίο, τα σβήνει όλα (196).

204-246: Η ψευδο-τυχαία κίνηση του ιππότη, όταν χάνει τον παίκτη. Επιτυγχάνεται με μία τυχαία μεταβλητή για την κατεύθυνση του ιππότη και μια άλλη τυχαία μεταβλητή που καθορίζει πότε θα αλλάξει κατεύθυνση. Η συνάρτηση είναι η **Random.Range(x,y)** και αλλάζει τη μεταβλητή κάθε φορά που εκτελείται ο κώδικας.

265-274: Τέλος, για καλύτερη τυχαία κίνηση, χρησιμοποιείται μία συνάρτηση **OnCollisionStay2D** η οποία εντοπίζει το πότε έρχεται σε επαφή ο ιππότης με τα colliders του περιβάλλοντος (δέντρα, πέτρες, κτλπ) και αλλάζει την κατεύθυνση του ιππότη σε μία από τις 8 τυχαίες κατευθύνσεις που υπάρχουν παραπάνω στο switch case.

5. Script μετάβασης περιοχής (πόρτας σπιτιού)

```
1.  using UnityEngine;
2.  using System.Collections;
3.  using UnityEngine.UI;
4.
5.  public class WarpManual : MonoBehaviour
```

```
6.      {
7.          public Transform target;
8.          public ItemEffects items;
9.          public GameObject player;
10.         public GameObject keychest;
11.         public GameObject mainText;
12.         public Text text;
13.         public GameObject frame;
14.         public Image image;
15.         public PlayerMovement move;
16.         public GameObject black;
17.         public Image blackimage;
18.         void Start()
19.         {
20.             player = GameObject.FindWithTag("Player");
21.             items = player.GetComponent<ItemEffects>();
22.
23.             mainText = GameObject.FindWithTag("text");
24.             text = mainText.GetComponent<Text>();
25.             frame = GameObject.Find("TextFrame");
26.             image = frame.GetComponent<Image>();
27.
28.             move = player.GetComponent<PlayerMovement>();
29.
30.             black = GameObject.Find("Black");
31.             blackimage = black.GetComponent<Image>();
32.         }
33.
34.         void OnTriggerStay2D(Collider2D other)
35.         {
36.             if (other.gameObject == player.gameObject)
37.             {
38.                 if (Input.GetKeyDown(KeyCode.E))
39.                 {
40.                     if (items.gotkey == true)
41.                     {
42.                         move.enabled = false;
43.                         blackimage.CrossFadeAlpha(255f, 1f, true);
44.                         Invoke("Teleport", 1);
45.                     }
46.                     if (items.gotkey == false && items.gotkey2 == false)
47.                     {
48.                         text.canvasRenderer.SetAlpha(255f);
```

```
                    image.canvasRenderer.SetAlpha(255f);
                    //Invoke("Fade", 2);

                    text.text = "It's locked. I think I saw a bat running off with my keys earlier...";
                }
                if (items.gotkey == false && items.gotkey2 == true)
                {
                    text.canvasRenderer.SetAlpha(255f);
                    image.canvasRenderer.SetAlpha(255f);
                    //Invoke("Fade", 2);

                    text.text = "The key doesn't fit the lock anymore...";

                }
            }
        }
    }
    void Fade()
    {
        text.CrossFadeAlpha(1f, 1, false);
        image.CrossFadeAlpha(1f, 1, false);
    }

    public void Teleport()
    {
        Debug.Log("TELEPORT");
        //yield return new WaitForSeconds(2);
        //Time.timeScale = 1;
        player.transform.position = target.position;
        move.enabled = true;

        blackimage.CrossFadeAlpha(1f, 1f, true);
    }
    void OnTriggerExit2D(Collider2D other)
    {
        if (other.gameObject == player)
        {
            text.CrossFadeAlpha(1f, 2, false);
            image.CrossFadeAlpha(1f, 2, false);
        }
    }
}
```

3: Εφόσον το script περιέχει UI elements (μαύρισμα οθόνης, κείμενο) πρέπει να προσθέσουμε και τη συγκεκριμένη βιβλιοθήκη.

34-65: Σημειώνεται ότι το script αυτό δεν χρειάζεται καθόλου thn Update. Η συνάρτηση OnTriggerStay2D ενεργοποιείται όταν ο παίκτης είναι μπροστά στην πόρτα του σπιτιού. Οι συνθήκες ελέγχουν αν έχει το σωστό κλειδί και εμφανίζουν κείμενο αν δεν το έχει. Εάν το έχει **(40-45)**, καλείται η συνάρτηση Teleport και μια αόρατη μαύρη εικόνα που έχουμε τοποθετήσει στη σκηνή και καλύπτει όλη την οθόνη, γίνεται ορατή **(43)** με τη συνάρτηση **CrossFadeAlpha.**
Γενικότερα τα UI elements είναι αντικείμενα που έχουν μηδενικό alpha (100% transparency) και τα κάνουμε ορατά με ανάλογα με τον ρυθμό που ορίζουμε στην CrossFadeAlpha. Όσο για το κείμενο το αλλάζουμε επί τόπου, εφόσον το UI element είναι τύπου Text **(12)**. Αρχικά χρησιμοποιήθηκαν πολλοί διάφοροι τρόποι εμφάνισης και κατασκευής UI elements, αλλά ο συγκεκριμένος αποδείχτηκε ο απλούστερος.

66-69: Αρχικά είχε χρησιμοποιηθεί για να καλείται όταν θέλαμε να γίνει κάτι fade out. Αλλά αν ο παίκτης πάταγε το κουμπί αλληλεπίδρασης E πολύ γρήγορα **(38),** η Fade δεν λειτουργούσε σωστά.

82-88: Εφόσον υπήρχε αυτό το πρόβλημε με τη Fade, αποφασίστηκε να εκτελείται ο συγκεκριμένος κώδικας **(86 & 87)** μόνο όταν ο παίκτης έφευγε από το σημείο αλληλεπίδρασης (π.χ. από την πόρτα).

72-81: Η συνάρτηση Teleport. Το Debug.Log **(74)** είναι ιδιέταιρα χρήσιμο σε πολλές περιπτώσεις και απλά εμφανίζει ένα μήνυμα στο debug console για να δει ο χρήστης αν εκτελείται το σημείο του κώδικα. Η χρήση του σε αυτό το σημείο είχε, αλλά δεν έχει πια κανένα νόημα. Στη σειρά **77** εκτελείται ο κώδικας που τηλεμεταφέρει τον παίκτη. Η τοποθεσία του target έχει οριστεί απο το interface.
Τέλος, στις σειρές **42 & 88** απενεργοποιείται/ενεργοποιείται το script κίνησης του παίκτη.

Σημειώνεται ότι ο κανβάς (Canvas - default αντικείμενο της Unity που εκπροσωπεί την εικόνα που βλέπει ο χρήστης), έχει δικές του παραμέτρους όπως το lighting, color κτλπ και πρέπει να είναι parent και να περικλείει όλα τα UI objects. Παράμετροι του κανβά ρυθμίζονται για τις ανάγκες του script No.7.

6. Script ενός UI element (επιθετικής δύναμης παίκτη)

```
    using System.Collections;
1.      using System.Collections.Generic;
2.      using UnityEngine;
3.      using UnityEngine.UI;
4.
5.      public class APCounter : MonoBehaviour
6.      {
7.          public PlayerAttack playerAttack;
8.          public GameObject player;
9.          public Text text;
```

```
10.     public GameObject apcounter;
11.
12.     public int damage;
13.
14.     // Use this for initialization
15.     void Start ()
16.     {
17.         player = GameObject.FindWithTag("Player");
18.         apcounter = GameObject.Find("APCounter");
19.
20.         text = apcounter.GetComponent<Text>();
21.         playerAttack = player.GetComponent<PlayerAttack>();
22.     }
23.
24.             // Update is called once per frame
25.             void Update ()
26.     {
27.         damage = playerAttack.damage;
28.         text.text ="" + damage;
29.     }
30. }
```

Ο παίκτης κατά τη διάρκεια του παιχνιδιού, μπορεί να πάρει ένα σπαθί και να το αναβαθμίσει μεγαλώνοντας την ζημιά που προκαλεί στους αντιπάλους. Η ζημιά αντικατοπτρίζεται σε έναν αριθμό στο UI με ένα μικρό σπαθί δίπλα του.

Το μόνο που κάνει αυτό το script είναι να παίρνει τη μεταβλητή damage από το script επίθεσης του παίκτη **(27)** και να την τοποθετεί στο UI text **(28).**

7. Script κύκλου μέρας/νύχτας

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class DayNight : MonoBehaviour
6.  {
7.
8.      public Light lighting;
9.      public Color blue;
10.     public Color red;
11.     public Color yellow;
12.     public float timer;
13.     public int x;
14.     public int day;
```

```
15.
16.     // Use this for initialization
17.     void Start()
18.     {
19.         lighting = GetComponent<Light>();
20.         lighting.intensity = 0;
21.         blue = Color.blue;
22.         red = Color.red;
23.         yellow = Color.yellow;
24.         day = 1;
25.     }
26.
27.     // Update is called once per frame
28.     void Update()
29.     {
30.         timer += Time.deltaTime;
31.
32.         if (timer < 35f) x = 1;
33.         if (timer > 35f) x = 2;
34.         if (timer > 70f) x = 3;
35.         if (timer > 105f) x = 4;
36.         if (timer > 140f) x = 5;
37.         if (timer > 180f) x = 6;
38.         if (timer > 215f) x = 7;
39.         if (day > 2 && timer > 105f) x = 8;
40.
41.         switch (x)
42.         {
43.             case 1:
44.                 lighting.color = yellow;
45.                 lighting.intensity += 0.015f * Time.deltaTime;
46.                 break;
47.             case 2:
48.                 //
49.                 break;
50.             case 3:
51.                 lighting.intensity -= 0.015f * Time.deltaTime;
52.                 break;
53.             case 4:
54.                 lighting.color = blue;
55.                 lighting.intensity += 0.02f * Time.deltaTime;
56.                 break;
```

```
57.             case 5:
58.                 //
59.                 break;
60.             case 6:
61.                 lighting.intensity -= 0.02f * Time.deltaTime;
62.                 break;
63.             case 7:
64.                 timer = 0;
65.                 day += 1;
66.                 Debug.Log("Day - " + day);
67.                 break;
68.             case 8:
69.                 lighting.color = red;
70.                 if (lighting.intensity < 0.85f)
71.                 {
72.                     lighting.intensity += 0.03f * Time.deltaTime;
73.                 }
74.                 break;
75.         }
76.     }
77. }
```

Το script αυτό χρησιμοποιεί ένα χρονόμετρο που μηδενίζεται στο τέλος μιας μέρας (case 7). Σε κάθε case (διαφορετική ώρα μέρας) ρυθμίζονται παράμετροι του κανβά αναλόγως. Κάθε φορά που κάνει reset το χρονόμετρο προστίθεται μια μέρα στο day counter για να χρησιμοποιηθεί στο case 8 και στο επόμενο script για τη δημιουργία εχθρών.

8. Script real-time δημιουργίας εχθρών (spawning)

```
1.  using UnityEngine;
2.  using System.Collections;
3.
4.  public class Spawning : MonoBehaviour
5.  {
6.      public DayNight globaltimer;
7.      public DayNight daycount;
8.
9.      public float timer1 = 0.0f;
10.     public float timer2 = 0.0f;
11.     public float timer3 = 0.0f;
12.     public float timer4 = 0.0f;
13.     public float bfreq;
```

```
14.     public float kfreq;
15.     public float mfreq;
16.     public float gfreq;
17.
18.     //public Rigidbody2D fail;
19.
20.     public GameObject spawn1;
21.     public GameObject spawn2;
22.     public GameObject spawn3;
23.     public GameObject spawn4;
24.     public GameObject spawn5;
25.     public GameObject spawn6;
26.     public GameObject spawnA;
27.     public GameObject spawnB;
28.
29.     public Transform tspawn1;
30.     public Transform tspawn2;
31.     public Transform tspawn3;
32.     public Transform tspawn4;
33.     public Transform tspawn5;
34.     public Transform tspawn6;
35.     public Transform tspawnA;
36.     public Transform tspawnB;
37.
38.     public Transform location;
39.     public GameObject[] bsum;
40.     public GameObject[] msum;
41.     public GameObject[] ksum;
42.     public int bmax = 10;
43.     public int mmax = 5;
44.     public int kmax = 3;
45.
46.     public GameObject genericBat;
47.     public GameObject Bat;
48.     public GameObject genericKnight;
49.     public GameObject Knight;
50.     public GameObject genericMonster;
51.     public GameObject Monster;
52.     public GameObject Boss;
53.     public GameObject genericBoss;
54.     public GameObject Glowbat;
55.     public GameObject genericGlow;
56.
```

```
57.     public bool bossup = false;
58.
59.     void Start()
60.     {
61.         globaltimer = GameObject.Find("Canvas").GetComponent<DayNight>();
62.         daycount = GameObject.Find("Canvas").GetComponent<DayNight>();
63.
64.         spawn1 = GameObject.Find("spawn1");
65.         spawn2 = GameObject.Find("spawn2");
66.         spawn3 = GameObject.Find("spawn3");
67.         spawn4 = GameObject.Find("spawn4");
68.         spawn5 = GameObject.Find("spawn5");
69.         spawn6 = GameObject.Find("spawn6");
70.         spawnA = GameObject.Find("spawnA");
71.         spawnB = GameObject.Find("spawnB");
72.
73.         tspawn1 = spawn1.GetComponent<Transform>();
74.         tspawn2 = spawn2.GetComponent<Transform>();
75.         tspawn3 = spawn3.GetComponent<Transform>();
76.         tspawn4 = spawn4.GetComponent<Transform>();
77.         tspawn5 = spawn5.GetComponent<Transform>();
78.         tspawn6 = spawn6.GetComponent<Transform>();
79.         tspawnA = spawnA.GetComponent<Transform>();
80.         tspawnB = spawnB.GetComponent<Transform>();
81.
82.
83.     }
84.
85.     void Update()
86.     {
87.         bfreq = Random.Range(5, 7);
88.         kfreq = Random.Range(15, 20);
89.         gfreq = Random.Range(50, 70);
90.         if (daycount.day < 2)
91.         {
92.             mfreq = Random.Range(13, 16);
93.         }
94.         else
95.         {
96.             mfreq = Random.Range(6, 9);
97.         }
98.
99.
100.        timer1 += Time.deltaTime;
```

```
101.        timer2 += Time.deltaTime;
102.        timer3 += Time.deltaTime;
103.        timer4 += Time.deltaTime;
104.
105.        bsum = GameObject.FindGameObjectsWithTag("Bat");
106.        msum = GameObject.FindGameObjectsWithTag("Monster");
107.        ksum = GameObject.FindGameObjectsWithTag("Knight");
108.
109.        if (timer1 >= mfreq && globaltimer.timer > 120f && globaltimer.timer < 200f)
110.            SpawnMonster();
111.
112.        if (timer2 >= bfreq)
113.            SpawnBat();
114.
115.        if (timer3 >= kfreq && daycount.day > 2)
116.            SpawnKnight();
117.
118.        if (daycount.day > 2 && globaltimer.timer > 170f && bossup == false)
119.            SpawnBoss();
120.
121.        if (timer4 >= gfreq)
122.        {
123.            Spawnglow();
124.        }
125.
126.    }
127.
128.    void Spawnglow()
129.    {
130.        timer4 = 0;
131.
132.        int randomPick = Random.Range(1, 7);
133.
134.        if (randomPick == 1)
135.        {
136.            location = tspawn1;
137.        }
138.        if (randomPick == 2)
139.        {
140.            location = tspawn2;
141.        }
142.        if (randomPick == 3)
143.        {
144.            location = tspawn3;
145.        }
```

```
146.        if (randomPick == 4)
147.        {
148.            location = tspawn4;
149.        }
150.        if (randomPick == 5)
151.        {
152.            location = tspawn5;
153.        }
154.        if (randomPick == 6)
155.        {
156.            location = tspawn6;
157.        }
158.
159.        genericGlow = Instantiate(Glowbat, location.position, location.rotation) as GameObject;
160.        genericGlow.GetComponent<Rigidbody>();
161.    }
162.
163.
164.    void SpawnBat()
165.    {
166.        timer2 = 0;
167.
168.        int randomPick = Random.Range(1, 7);
169.
170.        if (randomPick == 1)
171.        {
172.            location = tspawn1;
173.        }
174.        if (randomPick == 2)
175.        {
176.            location = tspawn2;
177.        }
178.        if (randomPick == 3)
179.        {
180.            location = tspawn3;
181.        }
182.        if (randomPick == 4)
183.        {
184.            location = tspawn4;
185.        }
186.        if (randomPick == 5)
187.        {
188.            location = tspawn5;
```

```
189.        }
190.        if (randomPick == 6)
191.        {
192.            location = tspawn6;
193.        }
194.
195.        if (bsum.Length < bmax)
196.        {
197.            genericBat = Instantiate(Bat, location.position, location.rotation) as GameObject;
198.            genericBat.GetComponent<Rigidbody>();
199.        }
200.        else return;
201.    }
202.
203.    void SpawnMonster()
204.    {
205.        timer1 = 0;
206.
207.        int randomPick = Random.Range(1, 3);
208.
209.        if (randomPick == 1)
210.        {
211.            location = tspawnA;
212.        }
213.        if (randomPick == 2)
214.        {
215.            location = tspawnB;
216.        }
217.
218.        if (msum.Length < mmax)
219.        {
220.            genericMonster = Instantiate(Monster, location.position, location.rotation) as GameObject;
221.            genericMonster.GetComponent<Rigidbody>();
222.        }
223.    }
224.
225.    void SpawnKnight()
226.    {
227.        timer3 = 0;
228.
229.        int randomPick = Random.Range(1, 3);
230.
231.        if (randomPick == 1)
232.        {
```

```
233.             location = tspawnA;
234.         }
235.         if (randomPick == 2)
236.         {
237.             location = tspawnB;
238.         }
239.
240.         if (ksum.Length < kmax)
241.         {
242.             genericKnight = Instantiate(Knight, location.position, location.rotation) as GameObject;
243.             genericKnight.GetComponent<Rigidbody>();
244.         }
245.     }
246.
247.     void SpawnBoss()
248.     {
249.         location = tspawnB;
250.         bossup = true;
251.         genericBoss = Instantiate(Boss, location.position, location.rotation) as GameObject;
252.         genericBoss.GetComponent<Rigidbody>();
253.     }
254.
255. }
```

Με αυτό το script φτιάχνονται (γίνονται instantiate) οι εχθροί κατά τη διάρκεια του παιχνιδιού. Για να γίνει instantiate ένα αντικείμενο πρέπει να έχει αποθηκευτεί ως prefab.

Τα αντικείμενα spawn1-6 που δηλώνονται, είναι τοποθετημένα σε διάφορα σημεία στον χάρτη και προορίζονται για τα σημεία από τα οποία μπορούν να βγουν νυχτερίδες (και μικρές νυχτερίδες), ενώ τα spawn A & B είναι η αφετηρίες των δύο μονοπατιών από τα οποία βγαίνουν οι ιππότες, τα τέρατα και το Boss.

Τα αντικείμενα αυτά είναι άδεια, σαν τα waypoints, αφού το μόνο που μας ενδιαφέρει είναι να πάρουμε την τοποθεσία τους.

42-44: Ορισμός μέγιστου αριθμού κάθε εχθρού που μπορέι να υπάρχει στο παιχνίδι. Τα δηλώσαμε ως πίνακες **(39-41)** ο καθένας από τους οποίους θα περιλαμβάνει/βρίσκει όλους τους εχθρούς του συγκεκριμένου είδους ανά πάσα στιγμή **(105-107).**

87-97: Ο κώδικας για τον ρυθμό με τον οποίο θα βγαίνουν οι εχθροί η οποία είναι σε περιορισμένο βαθμό τυχαία. Η συχνότητα των τεράτων μεγαλώνει την δεύτερη μέρα, χρησιμοποιώντας το daycount του προηγούμενου scrip **(90-97).** Ο τρόπος που χρησιμοποιούμε τη συχνότητα είναι να συγκρίνουμε τον αριθμό που φτιάξαμε με ένα timer που μηδενίζεται κάθε φορά που βγαίνει ένας εχθρός.

109-126: Εδώ καλούνται οι συναρτήσεις κατασκευής κάθε εχθρού μέσα από κάποιες συνθήκες. Π.χ. η συνάρτηση κατασκευής τεράτων καλείται μόνο την περίοδο του timer 120-200 seconds του προηγούμενου script, που αντιστοιχεί στη νύχτα.

Στο υπόλοιπο script υπάρχουν οι συναρτήσεις των εχθρών. Για παράδειγμα στην SpawnKnight():

Μηδενίζεται το timer προκειμένου να μπορεί να ξαναεκτελεστεί μόνο αφότου γίνει μεγαλύτερο απο το αντίστοιχο frequency. Μετά διαλέγεται τυχαία ένα από τα δύο spawn points που μπορεί να βγεί. Τέλος, εάν το μέγεθος του πίνακα (ksum.length) είναι μικρότερο από το kmax που ορίσαμε, εκτελείται η εντολή Instantiate, η οποία αποθηκεύει σε μια μεταβλητή τύπου αντικειμένου που δηλώσαμε στην αρχή (genericKnight), το prefab Knight και το τοποθετεί στο location που είναι η τοποθεσία του spawn point στη σκηνή.

9. Script επιλογών διαλόγου με τον έμπορο

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.  using UnityEngine.UI;
5.
6.  public class Shady : MonoBehaviour
7.  {
8.      public GameObject choice;
9.      public GameObject player;
10.
11.     public GameObject house;
12.     public HouseHealth houseHealth;
13.
14.     public GameObject selection;
15.     public GameObject choiceA;
16.
17.     public ItemEffects items;
18.
19.     public bool goaway1 = true;
20.     public bool goaway2 = false;
21.     public bool goaway3 = false;
22.
23.     public GameObject mainText;
24.     public Text text;
25.     public GameObject frame;
26.     public Image image;
27.
28.     public bool choice2 = false;
29.     public bool choice3 = false;
30.
31.     // Use this for initialization
32.     void Start()
```

```
33.     {
34.         player = GameObject.FindWithTag("Player");
35.
36.         choice = GameObject.Find("Choice");
37.         choiceA = GameObject.Find("ChoiceA");
38.         selection = GameObject.Find("Selection");
39.
40.         items = player.GetComponent<ItemEffects>();
41.
42.         mainText = GameObject.FindWithTag("text");
43.         text = mainText.GetComponent<Text>();
44.         frame = GameObject.Find("TextFrame");
45.         image = frame.GetComponent<Image>();
46.
47.         house = GameObject.FindWithTag("HouseHitbox");
48.         houseHealth = house.GetComponent<HouseHealth>();
49.
50.         choice.SetActive(false);
51.         choiceA.SetActive(false);
52.         selection.SetActive(false);
53.     }
54.
55.     // Update is called once per frame
56.     void Update()
57.     {
58.
59.     }
60.
61.     void OnTriggerStay2D(Collider2D other)
62.     {
63.         if (other.gameObject == player)
64.         {
65.             if (Input.GetKeyDown(KeyCode.E))
66.             {
67.                 text.canvasRenderer.SetAlpha(255f);
68.                 image.canvasRenderer.SetAlpha(255f);
69.
70.                 text.text = "Shady person: What do you want?";
71.
72.                 Cursor.visible = true;
73.                 selection.SetActive(true);
74.                 Time.timeScale = 0;
75.             }
76.         }
77.     }
```

```
78.
79.     void OnTriggerExit2D(Collider2D other)
80.     {
81.         choice.SetActive(false);
82.         choiceA.SetActive(false);
83.         selection.SetActive(false);
84.         Cursor.visible = false;
85.         text.CrossFadeAlpha(1f, 1f, true);
86.         image.CrossFadeAlpha(1f, 1f, true);
87.     }
88.
89.
90.
91.     public void Fade()
92.     {
93.         text.CrossFadeAlpha(1f, 1, false);
94.         image.CrossFadeAlpha(1f, 1, false);
95.     }
96.
97.     public void Yes()
98.     {
99.         if (choice2 == false && choice3 == false)
100.        {
101.            if (items.rupies > 2)
102.            {
103.                items.rupies -= 3;
104.                items.gotkey = false;
105.                items.gotkey2 = true;
106.
107.                text.canvasRenderer.SetAlpha(255f);
108.                image.canvasRenderer.SetAlpha(255f);
109.                //Invoke("Fade", 2);
110.
111.                text.text = "Shady person: Pleasure doing business.";
112.
113.                Cursor.visible = false;
114.                Time.timeScale = 1;
115.                choice.SetActive(false);
116.                choice2 = true;
117.            }
118.            else
119.            {
120.                text.canvasRenderer.SetAlpha(255f);
121.                image.canvasRenderer.SetAlpha(255f);
```

```
            //Invoke("Fade", 2);

            text.text = "Shady person: You don't even have 3 rupies...?";

            Cursor.visible = false;
            choice.SetActive(false);
            Time.timeScale = 1;
        }
    }
    else if (choice2 == true && choice3 == false)
    {
        if (items.rupies > 29)
        {
            items.rupies -= 30;
            items.gotkey = true;
            items.gotkey2 = false;

            text.canvasRenderer.SetAlpha(255f);
            image.canvasRenderer.SetAlpha(255f);
            //Invoke("Fade", 2);

            text.text = "Shady person: Thanks for the transaction.";

            Cursor.visible = false;
            choice.SetActive(false);
            Time.timeScale = 1;

            choice3 = true;
            goaway2 = true;
        }
        else
        {
            text.canvasRenderer.SetAlpha(255f);
            image.canvasRenderer.SetAlpha(255f);
            //Invoke("Fade", 2);

            text.text = "Shady person: Too expensive for you?";

            Cursor.visible = false;
            choice.SetActive(false);
            Time.timeScale = 1;
        }
    }
    else //if (choice3 == true)
```

```
166.        {
167.            if (items.rupies > 39)
168.            {
169.                items.rupies -= 40;
170.                items.gotmanual = false;
171.                items.gotmanualx = true;
172.
173.                text.canvasRenderer.SetAlpha(255f);
174.                image.canvasRenderer.SetAlpha(255f);
175.                //Invoke("Fade", 2);
176.
177.                text.text = "Shady person: Alright, I translated it to english.";
178.
179.                Cursor.visible = false;
180.                choice.SetActive(false);
181.                Time.timeScale = 1;
182.
183.                goaway3 = true;
184.            }
185.            else
186.            {
187.                text.canvasRenderer.SetAlpha(255f);
188.                image.canvasRenderer.SetAlpha(255f);
189.                //Invoke("Fade", 2);
190.
191.                text.text = "Shady person: You don't have enough money. I hope you have fun learning
192.                Chinese.";
193.
194.                Cursor.visible = false;
195.                choice.SetActive(false);
196.                Time.timeScale = 1;
197.            }
198.        }
199.    }
200.
201.    public void No()
202.    {
203.        text.canvasRenderer.SetAlpha(255f);
204.        image.canvasRenderer.SetAlpha(255f);
205.        //Invoke("Fade", 2);
206.
207.        text.text = "Shady person: Your loss...";
208.
209.        Cursor.visible = false;
```

```
210.        choice.SetActive(false);
211.        Time.timeScale = 1;
212.    }
213.
214.
215.
216.    public void Help()
217.    {
218.
219.        if (goaway1 == true)
220.        {
221.            text.canvasRenderer.SetAlpha(255f);
222.            image.canvasRenderer.SetAlpha(255f);
223.            //Invoke("Fade", 3);
224.
225.            text.text = "Shady person: What do you want? Go away!";
226.            selection.SetActive(false);
227.            Cursor.visible = false;
228.            Time.timeScale = 1;
229.        }
230.        if (items.gotkey == true && goaway2 == false)
231.        {
232.            goaway1 = false;
233.            text.canvasRenderer.SetAlpha(255f);
234.            image.canvasRenderer.SetAlpha(255f);
235.            //Invoke("Fade", 3);
236.
237.            text.text = "Shady person: I could modify that key of yours. I'll give you a discount... 3
238.            rupies.";
239.
240.            Cursor.visible = true;
241.            selection.SetActive(false);
242.            choice.SetActive(true);
243.            Time.timeScale = 0;
244.
245.        }
246.        if (items.gotkey2 == true)
247.        {
248.            text.canvasRenderer.SetAlpha(255f);
249.            image.canvasRenderer.SetAlpha(255f);
250.            //Invoke("Fade", 2);
251.
252.            text.text = "Shady person: I could modify your key back to how it was for, say ...30 rupies.";
253.
254.            Cursor.visible = true;
```

```
255.            selection.SetActive(false);
256.            choice.SetActive(true);
257.            Time.timeScale = 0;
258.        }
259.        if (goaway2 == true && items.gotmanual == false)
260.        {
261.            text.canvasRenderer.SetAlpha(255f);
262.            image.canvasRenderer.SetAlpha(255f);
263.            //Invoke("Fade", 2);
264.
265.            text.text = "Shady person: We are done... for now.";
266.            selection.SetActive(false);
267.            Cursor.visible = false;
268.            Time.timeScale = 1;
269.        }
270.        if (goaway2 == true && items.gotmanual == true)
271.        {
272.            text.canvasRenderer.SetAlpha(255f);
273.            image.canvasRenderer.SetAlpha(255f);
274.            //Invoke("Fade", 2);
275.
276.            text.text = "Shady person: Want me to translate that chinese manual you have for 40 rupies?";
277.
278.            Cursor.visible = true;
279.            selection.SetActive(false);
280.            choice.SetActive(true);
281.            Time.timeScale = 0;
282.        }
283.        if (goaway3 == true)
284.        {
285.            text.canvasRenderer.SetAlpha(255f);
286.            image.canvasRenderer.SetAlpha(255f);
287.            //Invoke("Fade", 3);
288.
289.            text.text = "Shady person: Since I took all your money, here's a tip: keep searching useless
290.            stuff, you may find something.";
291.            selection.SetActive(false);
292.            Cursor.visible = false;
293.            Time.timeScale = 1;
294.        }
295.    }
296.
297.    public void Repair()
```

```
    {
        text.canvasRenderer.SetAlpha(255f);
        image.canvasRenderer.SetAlpha(255f);
        //Invoke("Fade", 2);

        text.text = "Shady person: I could fortify your house a little for 20 rupies, no guarantees.";

        Cursor.visible = true;
        selection.SetActive(false);
        choiceA.SetActive(true);
        Time.timeScale = 0;
    }

    public void Yes1()
    {
        if (items.rupies > 19)
        {
            items.rupies -= 20;
            houseHealth.currentHealth += 150;

            text.canvasRenderer.SetAlpha(255f);
            image.canvasRenderer.SetAlpha(255f);
            //Invoke("Fade", 2);

            text.text = "Shady person: There, good as new!";

            Cursor.visible = false;
            choiceA.SetActive(false);
            Time.timeScale = 1;
        }
        else
        {
            text.canvasRenderer.SetAlpha(255f);
            image.canvasRenderer.SetAlpha(255f);
            //Invoke("Fade", 2);

            text.text = "Shady person: Want me to do it for free? Dream on.";

            Cursor.visible = false;
            choiceA.SetActive(false);
            Time.timeScale = 1;
        }
    }

```

```
342.     public void No1()
343.     {
344.         text.canvasRenderer.SetAlpha(255f);
345.         image.canvasRenderer.SetAlpha(255f);
346.         //Invoke("Fade", 2);
347.
348.         text.text = "Shady person: Don't blame me if your house gets destroyed...";
349.
350.         Cursor.visible = false;
351.         choiceA.SetActive(false);
352.         Time.timeScale = 1;
353.
354.     }
355.
356.     public void Leave()
357.     {
358.         selection.SetActive(false);
359.         Cursor.visible = false;
360.         Time.timeScale = 1;
361.     }
362. }
```

Αυτό είναι το script του Shady person, ενός NPC στον οποίο πρέπει να μιλήσει ο παίκτης προκειμένου να τον βοηθήσει να πάρει κάποια αντικείμενα και για να του επισκευάσει το σπίτι. Δεν χρησιμοποιεί καθόλου την Update και είναι ένα πλέγμα/δέντρο από επιλογές διαλόγου. Κάθε επιλογή αντιστοιχεί σε μια συνάρτηση η οποία δεν καλείται από script, αλλά κατευθείαν από το input του παίκτη στο interface της Unity (button UI element).

Για παράδειγμα:

297-309: Η κύρια επιλογή “Repair” είναι ένα απλό παρακλάδι του δέντρου, ακολουθείται από τις επιλογές/συναρτήσεις Yes/No **(311 & 342)** και σε κάθε μία από αυτές εξετάζεται αν ο παίκτης έχει αρκετά χρήματα για τη συναλλαγή. Στην **299 & 300** εμφανίζεται το πλαίσιο του διαλόγου. **303:** Εμφανίζεται ο κατάλληλος διάλογος. **305:** Εμφανίζεται ο κέρσορας του ποντικιού, ο οποίος είναι κρυμμένος γενικότερα στο παιχνίδι από άλλο script. **306:** Όταν εκτελείται εξαφανίζεται το UI της επιλογής και **(307)** εμφανίζεται το UI της επόμενης. **308:** Χρησιμοποιείται όταν θέλουμε να κάνουμε παύση στο παιχνίδι (π.χ. χρησιμοποιείται και στο escape menu).

Το μεγαλύτερο παρακλάδι του script, η κύρια επιλογή στην σειρά 216, λειτουργεί με την ίδια λογική αλλά επειδή υπάρχουν πολλά παρακλάδια που την ακολουθούν, έχουν φτιαχτεί πολλές boolean μεταβλητές για να οδηγούν κάθε φορά στην κατάλληλη περίπτωση (π.χ. αν έχει ο παίκτης ένα αντικείμενο στην κατοχή του).

10. Script αλληλεπίδρασης αντικειμένων με τον παίκτη

```
1.  using System.Collections;
2.  using System.Collections.Generic;
3.  using UnityEngine;
4.
5.  public class ItemEffects : MonoBehaviour
6.  {
7.
8.      public PlayerHealth health;
9.      public PlayerMovement movement;
10.     public PlayerAttack attack;
11.     public GameObject player;
12.
13.     public GameObject barrel;
14.     public Rigidbody2D barrelmass;
15.
16.     public GameObject[] hearts;
17.     public GameObject[] rupies1;
18.     public GameObject[] rupies2;
19.     public GameObject[] rupies3;
20.
21.     public GameObject speedboots;
22.     public GameObject manual;
23.     //public GameObject strengthgloves;
24.     public GameObject goldenplate;
25.     public GameObject stonesword;
26.     public GameObject firesword;
27.     public GameObject key;
28.
29.     public bool gotkey = false;
30.     public bool gotkey2 = false;
31.
32.     public bool gotboots = false;
33.     public bool gotmanual = false;
34.     public bool gotmanualx = false;
35.     public bool gotgloves = false;
36.     public bool gotplate = false;
37.     public bool gotcape = false;
38.     public bool gotcapetrue = false;
39.
40.     public bool gotstone = false;
41.
42.     public int rupies = 0;
43.
```

```
44.     public AudioSource effectplayer;
45.     public AudioClip coin;
46.     public AudioClip heartsound;
47.     public AudioClip item;
48.     public AudioClip keys;
49.     public AudioClip upgrade;
50.
51.     public Collider2D col;
52.
53.     // Use this for initialization
54.     void Start()
55.     {
56.         player = GameObject.FindWithTag("Player");
57.         health = player.GetComponent<PlayerHealth>();
58.         movement = player.GetComponent<PlayerMovement>();
59.         attack = player.GetComponent<PlayerAttack>();
60.
61.         //barrel = GameObject.FindWithTag("barrel");
62.         //barrelmass = barrel.GetComponent<Rigidbody2D>();
63.
64.         effectplayer = GetComponent<AudioSource>();
65.
66.         col = GetComponent<Collider2D>();
67.     }
68.
69.     // Update is called once per frame
70.     void Update()
71.     {
72.         hearts = GameObject.FindGameObjectsWithTag("heart");
73.         rupies1 = GameObject.FindGameObjectsWithTag("rupie1");
74.         rupies2 = GameObject.FindGameObjectsWithTag("rupie2");
75.         rupies3 = GameObject.FindGameObjectsWithTag("rupie3");
76.
77.         speedboots = GameObject.FindWithTag("speedboots");
78.         manual = GameObject.FindWithTag("manual");
79.         key = GameObject.FindWithTag("key");
80.         goldenplate = GameObject.FindWithTag("goldenplate");
81.         stonesword = GameObject.FindWithTag("stonesword");
82.         firesword = GameObject.FindWithTag("firesword");
83.
84.         if (gotcape == true && gotcapetrue == false)
85.         {
86.             gotcapetrue = true;
87.             effectplayer.clip = upgrade;
```

```
            effectplayer.Play();
            attack.damage += 5;
            health.maxHealth += 50;
            col.enabled = false;
            Color tmp = player.GetComponent<SpriteRenderer>().color;
            tmp.a = 0.5f;
            movement.speed += 0.5f;
            player.GetComponent<SpriteRenderer>().color = tmp;
        }
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        foreach (GameObject heart in hearts)
        {
            if (other.gameObject == heart && health.currentHealth < health.maxHealth -3)
            {
                effectplayer.clip = heartsound;
                effectplayer.Play();
                health.currentHealth += 25;
                Destroy(other.gameObject);
            }
        }
        foreach (GameObject rupie1 in rupies1)
        {
            if (other.gameObject == rupie1)
            {
                effectplayer.clip = coin;
                effectplayer.Play();
                rupies += 1;
                Debug.Log("Rupies:" + rupies);
                Destroy(other.gameObject);
            }
        }
        foreach (GameObject rupie2 in rupies2)
        {
            if (other.gameObject == rupie2)
            {
                effectplayer.clip = coin;
                effectplayer.Play();
                rupies += 5;
```

```
            Debug.Log("Rupies:" + rupies);
            Destroy(other.gameObject);
        }
    }
    foreach (GameObject rupie3 in rupies3)
    {
        if (other.gameObject == rupie3)
        {
            effectplayer.clip = coin;
            effectplayer.Play();
            rupies += 15;
            Debug.Log("Rupies:" + rupies);
            Destroy(other.gameObject);
        }
    }
    if (other.gameObject == speedboots)
    {
        effectplayer.clip = item;
        effectplayer.Play();
        movement.speed = 2f;
        gotboots = true;
        Destroy(other.gameObject);
    }
    if (other.gameObject == manual && gotmanual == false && gotmanualx == false)
    {
        effectplayer.clip = item;
        effectplayer.Play();
        gotmanual = true;
        Destroy(other.gameObject);
    }
    if (other.gameObject == key && gotkey == false && gotkey2 == false)
    {
        effectplayer.clip = keys;
        effectplayer.Play();
        gotkey = true;
        Destroy(other.gameObject);
    }
    if (other.gameObject == goldenplate && gotplate == false)
    {
        effectplayer.clip = item;
        effectplayer.Play();
```

```
            gotplate = true;
            health.regen = true;
            Destroy(other.gameObject);
        }
        /*if (other.gameObject == stonesword && gotgloves == true)
        {
            //change animations
            attack.damage += 10;
            Destroy(other.gameObject);
        }
        if (other.gameObject == firesword && gotstone == true && gotmanual == true)
        {
            //change animations
            attack.damage += 12;
            Destroy(other.gameObject);
        }*/
    }
}
```

Ένα script για να "παίρνει" ο παίκτης τα αντικείμενα. Κάποια αντικείμενα δεν συμπεριλαμβάνονται **(174-186)** για λόγους ευκολίας (π.χ. το script που παίρνει και αναβαθμίζει το σπαθί βρίσκεται στα αντίστοιχα αντικείμενα στο παιχνίδι και όχι στον παίκτη).

72-75: Οι καρδιές και οι ρουπίες αποθηκεύονται σε πίνακες (καθώς μπορεί να είναι πολλά αντίτυπα ενός prefab) στην Update, προκειμένου να βρίσκονται ανά πάσα στιγμή.

77-82: Το ίδιο γίνεται και με τα άλλα αντικείμενα, αλλά επειδή είναι μοναδικά και μπορούν να πέσουν μόνο μία φορά από το script που τα κάνει instantiate στο γεγονός του θανάτου ενός εχθρού, δεν χρειάζεται να αποθηκευτούν σε πίνακα.

84-96: Οι επιδράσεις του "μυστικού" αντικειμένου στον παίκτη.

99-173: Ένα μεγάλο onTriggerEnter, με το οποίο όταν ο παίκτης έρχεται σε επαφή με το κάθε αντικείμενο, αλλάζουν ανάλογα οι παράμετροι διαφόρων script του, προστίθενται στο UI ("τσάντα" του παίκτη), ενεργοποιούνται sound effects και τέλος καταστρέφεται το αντικείμενο από τη σκηνή.

Συνοπτικά βήματα που ακολουθήθηκαν για την κατασκευή του

Οι διαδικασίες όπως δρομολογήθηκαν στο Trello για τον σχεδιασμό του video game:

-Συλλογή assets.

-Δημιουργία tiles και περιοχής με το Tiled.

-Δημιουργία σκηνής.

-Importation των tiles με το TiledtoUnity.

-Δημιουργία βασικών animations.

-Ανέβασμα αρχείων στο Github.

-Αλληλεπίδραση παίκτη με trigger colliders στο περιβάλλον.

-Text UI elements (έγιναν με διαφορετικό τρόπο στην αρχή από αυτόν που τελικά χρησιμοποιήθηκε).

-Scripts για warp.

-Animated colliders παίκτη (ενεργοποίηση και κίνηση τους σε κάθε καρέ).

-Bat και Health και Attack scripts για αυτό και τον παίκτη.

-Enemy AI 1: Για τα bats (random movement και όταν φτάσουν κοντά στον παίκτη τον ακολουθούν).

-Monster, Knight (animations, colliders, health/attack scripts etc).

-Spawn system (spawn scripts για τα παραπάνω).

-Enemy AI 2: Για Monster & Knight (pathing με πίνακα waypoints, ακολουθούν τον παίκτη/σπίτι όταν έρχονται σε εμβέλεια σβήνοντας τα waypoints και αν ο παίκτης φύγει μακριά random movement).

-Item interactions: Περιλαμβάνει πολλά, αλλά συνοπτικά είναι όλα τα scripts με τα οποία αλληλεπιδρά ο παίκτης στο περιβάλλον π.χ. τα μπαούλα ή αντικείμενα που πέφτουν.

-Σπίτι ως δευτερεύον στόχος μετά τον παίκτη.

-Random drops και death effects για εχθρούς.

-Κύκλος μέρας/νύχτας και προσαρμογή άλλων scripts βάσει αυτού.

-Διαφορετικά animation sets παίκτη (ανάλογα με το τι όπλο έχει – το 1ο σπαθί έγινε με attach του sprite στο animation του παίκτη. Το 2ο έγινε με διαφορετικό μοντέλο παίκτη στο οποίο προστέθηκε το σπαθί με επεξεργασία στο photoshop.

-Boss (όλα τα scripts που προαναφέρθηκαν για τα άλλα, projectile/fireball και lightning scripts).

-Δημιουργία main menu σε 2^{η} σκηνή.

-Πρόσθεση audio με τα κατάλληλα script που το κάνουν trigger βάσει χρόνου, τοποθεσίας ή αλληλεπίδρασης και επεξεργασία με Audacity όποτε χρειαζόταν.

-UI (life, house life, rupies, attack power, inventory, escape menu).

-NPC με επιλογές/διάλογο και δικό του UI.

Γενικές βοηθητικές πηγές/Tutorials

-Το **manual της Unity** (https://docs.unity3d.com/Manual/index.html). Περιέχει ότι μπορεί να χρειαστεί ο χρήστης σε σχέση με τις βιβλιοθήκες της Unity – συναρτήσεις, εντολές κ.α.

-**Youtube** (το καλύτερο μέρος για tutorials για την Unity και κάθε άλλον editor).΄

- -"rm2kdev" channel

(https://www.youtube.com/channel/UCJvrLzbg4VPRxzf2vhW7G4A)

(Πολύ απλό tutorial για Unity 2D rpg, περιέχει οδηγίες για κατασκευή 2D animations, απλό script κίνησης και τηλεμεταφοράς και οδηγίες για την χρήση του Tiled – χρησιμοποιήθηκε και για τη συγκεκριμένη πτυχιακή.)

- -"Let's code games" channel

(https://www.youtube.com/channel/UCSKZ683Om2H9yHLL1yX5dBQ/featured)

(Πιο προχωρημένο και πολύ καλό tutorial για Unity 2D rpg - τα πρώτα 2 βίντεο επίσης χρησιμοποιήθηκαν και τα πιο σημαντικά – και καλύτερης ποιότητας – free license assets πάρθηκαν από το link στην περιγραφή.)

- -"Jimmy Vegas" channel

https://www.youtube.com/watchv=G9BdFZ2MCXc&list=PLZ1b66Z1KFKik2g8D4wrmYj4yein4rCk8

(Πολλαπλά tutorials σχετικά με την Unity για προχωρημένους και αρχάριους, καλύπτοντας την κατασκευή 3D games, animations, scripting και άλλων λειτουργιών της Unity.)

-Unity Questions http://answers.unity3d.com/questions/

(Βοήθεια από το community σε οτιδήποτε χρησιμοποιείται στην Unity)

-Stack Overflow http://stackoverflow.com

(Βοήθεια, κυρίως για κώδικα, για οποιαδήποτε προγραμματιστική γλώσσα.)

-Slack (chat για developers) https://unitydevs.slack.com

(Βοήθεια με live chat, με πολλές κατηγορίες που καλύπτουν όλες τις λειτουργίες της Unity.)

Text tutorials

- http://scottlilly.com/learn-c-by-building-a-simple-rpg-index/
  (C# αρκετά απλό αλλά εκτενές tutorial για κατασκευή RPG.)

- https://www.raywenderlich.com/category/unity
  (Εκτενές tutorial για προχωρημένους για κατασκευή 2D game με την Unity και C#.)

Πηγές assets που χρησιμοποιήθηκαν

- http://opengameart.org/

(Η πιο δημοφιλής βάση δεδομένων για δωρεάν 2D/3D γραφικά.)

- https://retrogamezone.co.uk/

(Βάση δεδομένων με μεγάλη ποικιλία δωρεάν 2D γραφικών.)

- https://www.freesound.org/

(Βάση δεδομένων με πολύ μεγάλη ποικιλία δωρεάν ηχητικών εφέ.)

- https://www.youtube.com/

(Χρησιμοποιήθηκε για την μουσική – με σχετική άδεια, συνήθως όχι δωρεάν.)

Q&A

-Γιατί επιλέχθηκε το συγκεκριμένο είδος 2D game;

Επιλέχθηκε να γίνει σε 2D λόγω περιορισμένων πόρων του υπολογιστή για 3D περιβάλλοντα που απαιτούν πιο πολλούς πόρους στο game preview της Unity και των ελάχιστων free assets που υπάρχουν για 3D models και art. Επίσης το συγκεκριμένο είδος (role playing game) επιλέχθηκε λόγω των προτιμήσεων του φοιτητή και επίσης γιατί έχει μεγάλη ποικιλία διαφορετικών χαρακτηριστικών και λογικής, όσο αφορά τον προγραμματισμό του.

-Γιατί η μεγάλη πλειοψηφία των μεταβλητών στα scripts είναι public ενώ δεν χρησιμοποιούνται από άλλα scripts;

Γιατί αυτό επιτρέπει πολύ εύκολο debugging και παρακολούθηση των μεταβλητών καθώς είναι ορατά στον Inspector κατά την διάρκεια του game preview. Εφόσον κάθε μεταβλητή είναι μοναδική, δεν υπάρχει εμφανής λόγος για να γίνουν private.

-Γιατί δεν υπάρχει λειτουργία save/load στο παιχνίδι;

Εκτός από το γεγονός ότι δεν αρμόζει στο gameplay και την διάρκεια του παιχνιδιού, η λειτουργία αυτή αποδείχτηκε αρκετά δύσκολη στο συγκεκριμένο είδος, λόγω του ότι η Unity δεν φαίνεται να έχει κάποια έτοιμη λειτουργία για να "σώσει" την κατάσταση/στιγμιότυπο του παιχνιδιού ανά πάσα στιγμή και το τελευταίο έχει πάρα πολλά variables τα οποία αλλάζουν συνέχεια κατά τη διάρκεια του. Συνήθως ακόμα και επαγγελματίες χρησιμοποιούν έτοιμα plugins τα οποία διαθέτουν αυτή τη λειτουργία. Σημειώνεται ότι, εάν το παιχνίδι είχε levels σε ξεχωριστά scenes, η λειτουργία αυτή θα ήταν πολύ απλή υπόθεση.

-Γιατί οι εχθροί δεν γίνονται knockback όταν τους επιτίθεται ο παίκτης;

Το knockback ήταν ένα από τα features μεγάλης προτεραιότητας που είχαν προγραμματιστεί για το παιχνίδι. Ωστόσο εμφανίστηκαν προβλήματα στην πορεία και μετά από πολύ πειραματισμό και δοκιμές δεν μπήκε στο παιχνίδι. Υπάρχουν δύο κύριοι τρόποι κίνησης για τα αντικείμενα στον χώρο του παιχνιδιού. Ο ένας είναι με μεταβολή του Transform που μεταβάλει κατευθείαν τις συντεταγμένες του αντικειμένου στον χώρο. Ο άλλος μέσω physics και ασκεί δύναμη στο αντικείμενο, μέσω του rigidbody του, που όπως έχει αναφερθεί εκπροσωπεί τη μάζα του αντικειμένου. Ο δεύτερος τρόπος (ο οποίος χρησιμοποιήθηκε στον παίκτη μόνο χωρίς προβλήματα) αποδείχτηκε προβληματικός για τους εχθρούς και γενικά φαίνεται ότι δεν προτείνεται για 2D. Η κίνηση των εχθρών με τον δεύτερο τρόπο δεν ήταν ομαλή και μερικές φορές τα colliders τους δεν λειτουργούσαν όπως πρέπει. Η κίνηση τους με τον πρώτο τρόπο (ο οποίος χρησιμοποιήθηκε στους εχθρούς) ήταν μεν ομαλή αλλά το knockback effect δεν ήταν επιθυμητό.

-Ποια είναι τα γνωστά bugs;

1. Κάποιοι πολυγωνικοί animated colliders δεν κινούνται με βάση των κινήσεων που τους δόθηκαν. Δεν βρέθηκε λύση.
2. Η πρώτη μορφή του σπαθιού που δημιουργήθηκε με προσκόλληση, μέσω του animation window, ξεχωριστού texture, έχει lag στην διαγώνια κίνηση (η δεύτερη προσκολλήθηκε με επεξεργασία στο photoshop). Δεν βρέθηκε λύση, εκτός του ότι θα μπορούσε να γίνει και αυτή με τον δεύτερο τρόπο.
3. Οι colliders που δεν είναι ορισμένοι ως triggers και χρησιμοποιούνται μόνο για τις συγκρούσεις, κάνουν trigger γεγονότα που δεν θα έπρεπε. Κατά τα λεγόμενα developer, αυτό θα αλλάξει στο μέλλον.
4. Σε μεγάλες αναλύσεις, 2k+, το πλαίσιο της κάμερας μεγαλώνει πολύ (σαν να απομακρύνεται) και τα πάντα φαίνονται πολύ μικρά, επειδή τα textures/sprites δεν κάνουν scale. Δεν βρέθηκε εφικτή λύση.

-Γιατί δεν φτιάχτηκαν τα γνωστά bugs;

Δεν βρέθηκε λύση, ούτε με την βοήθεια του community (πράγμα σπάνιο). Υπάρχει πιθανότητα να είναι και bugs της Unity καθώς η υποστήριξη 2D είναι σχετικά καινούριο feature.