ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής-Ανάπτυξης Λογισμικού και Τεχνητής Νοημοσύνης»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**TRY TO LIVE: Ανάπτυξη βιντεοπαιχνιδιού επιβίωσης βολών πρώτου προσώπου με την βοήθεια τεχνητής νοημοσύνης** **TRY TO LIVE: Development of a first-person shooter survival video game with artificial intelligence techniques**
Ονοματεπώνυμο Φοιτητή	**Φωτιάδης Φώτιος**
Πατρώνυμο	**Γεώργιος**
Αριθμός Μητρώου	**ΜΠΣΠ19056**
Επιβλέπων	**Παναγιωτόπουλος Θεμιστοκλής, Καθηγητής**

Ημερομηνία Παράδοσης **Ιούλιος 2022**

Τριμελής Εξεταστική Επιτροπή

Θεμιστοκλής Παναγιωτόπουλος Καθηγητής	Διονύσιος Σωτηρόπουλος Επίκουρος Καθηγητής	Ιωάννης Τασούλας Επίκουρος Καθηγητής

Ευχαριστίες

Για την επιτυχή εκπόνηση της παρούσας διπλωματικής διατριβής θα ήθελα πρωτίστως να αποδώσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή κ. Θεμιστοκλή Παναγιωτόπουλο, για την συνεχή επίβλεψη και καθοδήγηση του καθ΄ όλη την διάρκεια της προσπάθειας αυτής σε όλα τα στάδια, καθώς και την ενθάρρυνση για έρευνα και ενασχόληση με το αντικείμενο αυτό.

Εν συνεχεία, θα ήθελα να ευχαριστήσω όλους του καθηγητές του προγράμματος σπουδών για τις σημαντικές γνώσεις και εμπειρίες που αποκόμισα από το παρόν πρόγραμμα σπουδών παρακολουθώντας τις διαλέξεις τους. Επιπρόσθετα, θα ήθελα να ευχαριστήσω ιδιαίτερα τους συμφοιτητές μου για την πολύτιμη βοήθεια και την ανταλλαγή γνώσεων και απόψεων που συνέβαλαν στην επιτυχή ολοκλήρωση των σπουδών μου.

Τέλος, θέλω να ευχαριστήσω τους γονείς μου για την πολύτιμη υποστήριξη, ενθάρρυνση και συμπαράσταση που μου παρέχουν όλα αυτά τα χρόνια.

Περιεχόμενα

Περίληψη

Στην παρούσα διπλωματική παραθέτεται ο τρόπος υλοποίησης του TRY TO LIVE, που είναι βιντεοπαιχνίδι επιβίωσης βολών πρώτου προσώπου (First Person Shooter Game), με την χρήση του Unity και της γλώσσας προγραμματισμού C#. Ύστερα από μια εισαγωγή και μια γενικότερη προσέγγιση πάνω στα βίντεο παιχνίδια, γίνεται μια παρουσίαση για τις μηχανές τρισδιάστατων γραφικών και την γλώσσα προγραμματισμού που χρησιμοποιήθηκε. Κατόπιν, αναφέρονται οι προδιαγραφές που απαιτούνται από έναν υπολογιστή για να τρέξει η εφαρμογή και εξιστορείται η πλοκή (back story) πάνω στην οποία βασίζεται το TRY TO LIVE. Ακολούθως, παρουσιάζεται η σχεδίαση των διαπροσωποιών της εφαρμογής και παρατίθενται τα διάφορα scripts μαζί με τις λειτουργίες τους. Τέλος, γίνεται λεπτομερής ανάλυση του gameplay του παιχνιδιού μαζί με screenshots.

Abstract

In this dissertation we present the TRY TO LIVE first-person shooter survival video game which has been developed by using the Unity 3D Engine 2019.4.20f1 along with the C# programming language. After an introduction and a general approach to video games, we mention the software engines of the gaming industry and a short introduction to C#. Then, we provide the specifications required by a computer to run the application and the back story on which it is based. Next, we present the application interfaces as well as the C# scripts that run behind it, along with their functions. Finally, the dissertation contains a detailed analysis of TRY TO LIVE gameplay with screenshots.

1 Εισαγωγή

1.1 Εισαγωγή στο TRY TO LIVE

Η εργασία είναι εμπνευσμένη από αρκετά διάσημα παιχνίδια επιβίωσης όπως για παράδειγμα το Left 4 Dead 2, το Destiny 2, Alien Isolation και άλλα. Το παιχνίδι αυτό είναι single-player και έτσι θα υπάρχει ένας χαρακτήρας. Ο χρήστης υπολογιστή χειρίζεται μέσω πληκτρολογίου και ποντικιού έναν παίκτη πρώτου προσώπου με σκοπό να κινηθεί στον χώρο προσπαθώντας να φτάσει στον προορισμό του, δηλαδή ένα σπίτι στον χάρτη, αποφεύγοντας εμπόδια και επιβιώνοντας από τους εχθρούς. Η κίνηση του είναι προς όλες τις κατευθύνσεις και επιπρόσθετα μπορεί να κάνει άλμα και να σκύψει. Επιπλέον, μπορεί να επιτεθεί με 6 όπλα με διαφορετικό τρόπο επίθεσης στο καθένα. Κατά την διάρκεια του παιχνιδιού εμφανίζονται τυχαία στον χάρτη δύο είδη εχθρών (NPCs) που περιπολούν την περιοχή και αν εντοπίσουν τον χρήστη του επιτίθονται. Αναλυτικότερα θα δούμε στην συνέχεια της διπλωματικής.

1.2 Εισαγωγή στα video games

Στην εποχή που ζούμε τα video games έχουν γίνει ένα αναπόσπαστο κομμάτι του ελεύθερου μας χρόνου. Ολοένα και περισσότεροι χρήστες ηλεκτρονικών συσκευών παίζουν video games σαν χόμπι. Αυτή η ζήτηση και η δημοτικότητα των βιντεοπαιχνιδιών δεν έχει αφήσει αδιάφορη την βιομηχανία, καθώς παρακολουθούμε να γίνονται επενδύσεις και εξαγορές studio πολλών δισεκατομμυρίων ευρώ ξεπερνώντας και την βιομηχανία ταινιών του Hollywood, με πρόσφατο

παράδειγμα την εξαγορά της Activision από την Microsoft με μια συμφωνία να φτάνει στα 70 δις. δολάρια. Ποσό ρεκόρ για τον κλάδο, που φιγουράρει να γίνεται ο πιο σημαντικός στον τομέα της ψυχαγωγίας.

Ωστόσο, τα βιντεοπαιχνίδια κατακρίνονται για την προβολή της βίας σε νεότερες ηλικίες αλλά και τον εθισμό που προκαλούν. Παρόλα αυτά, τα video games αρκετές φορές παρατηρούμε να έχουν θετικές συνέπειές στην υγεία και στην ψυχολογία. Επίσης, χρησιμοποιούνται για εκπαίδευση αλλά και προσομοίωση καταστάσεων, κοινωνικών προβλημάτων, τεχνολογικών προβλημάτων, ιατρικής κλπ. Αυτό έχει σαν αποτέλεσμα να υπάρχουν πολλά διαφορετικά ήδη βιντεοπαιχνιδιών όπως στρατηγικής, αγώνων, δράσης, ρόλων, περιπέτειας, προσομοίωση ζωής, επιβίωσης και αλλά.

1.3 Ορισμός video game

Κατά την άποψη μου ο πιο ορθός ορισμός που προκύπτει για τα βιντεοπαιχνίδια από την βιβλιογραφία και το διαδίκτυο είναι ο παρακάτω:

Βιντεοπαιχνίδι (αγγλικά: video game) ορίζεται κάθε παιχνίδι που εκτελείται με την χρήση τεχνολογίας. Συσκευές εκτέλεσης των βιντεοπαιχνιδιών είναι οι ηλεκτρονικοί υπολογιστές, κονσόλες βιντεοπαιχνιδιών (PlayStation,Xbox, Nintendo Switch etc.) κινητά τηλέφωνα και άλλα.

2 Εισαγωγή στις Μηχανές παιχνιδιών και στο AI

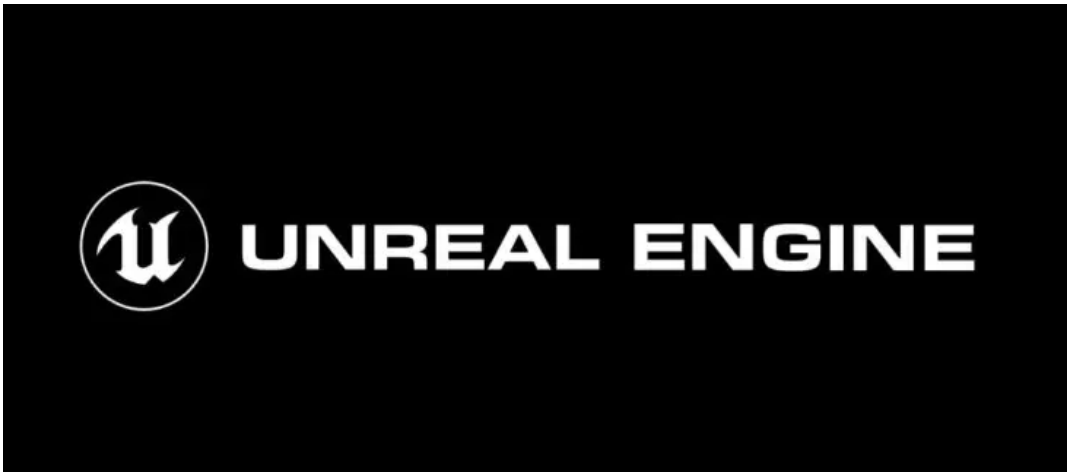
2.1 Μηχανές παιχνιδιών

Για την δημιουργία ενός video game είναι απαραίτητη η χρήση ενός engine.Μια μηχανή παιχνιδιού είναι ένα σύστημα λογισμικού σχεδιασμένο για την ανάπτυξη και την δημιουργία video games .Πιο συγκεκριμένα ένα engine αποτελείται από μια συνισταμένη δράσης και ενεργειών ενός σύνολο λογισμικού που εκτελεί βασικές λειτουργίες εντός της εφαρμογής, όπως animation (3D εφαρμογές), αναπαραγωγή ήχων, μηχανή φυσικής (physics),τεχνητή νοημοσύνη, απεικόνιση γραφικών στοιχείων (renderer), scripting και διαχείριση πόρων υπολογιστή (κάρτα γραφικών, επεξεργαστές, μνήμη κ.α.) από το πρόγραμμα και προσαρμογή του λογισμικού για εκτέλεση σε διαφορετικά λειτουργικά συστήματα.

Η υλοποίηση οποιουδήποτε βιντεοπαιχνιδιού, ανεξαρτήτου μεγέθους, απαιτεί τεράστιους πόρους σε hardware, software, χρόνου και προσωπικού. Για αυτό τον λόγο η διαδικασία κατασκευής πολλές φορές

οικονομικοποιείται, δηλαδή η ίδια engine χρησιμοποιείται για διαφορετικά βιντεοπαιχνίδια. Συνήθως τα studio χρησιμοποιούν διάφορα engines,για παράδειγμα Amazon Lumberyard,CryEngine κ.α. ωστόσο οι πιο διαδεδομένες μηχανές κατασκευής παιχνιδιών, είναι η Unity και η Ureal. Τέλος να αναφέρουμε ότι τα πιο μεγάλα studio να κατασκευάζουν και χρησιμοποιούν την δικιά τους engine (Bungie, CD Project Red κ.α.)

2.2 Unreal Engine



Η Unreal είναι μια από τις κορυφαίες engines για την κατασκευή βιντεοπαιχνιδιών και εκδόθηκε από την Epic Games το 1998.Θεωρείται σχετικά φιλική προς στον χρήστη, ενώ υπάρχει πληθώρα βιβλιογραφίας σχετικά με αυτήν. Χρήζει δυνατή για οποιαδήποτε υλοποίηση παιχνιδιού και σε οποιαδήποτε πλατφόρμα. Επιπρόσθετα, χρησιμοποιείται και σε βιομηχανίες όπως τις αυτοκινητοβιομηχανίες, αρχιτεκτονικές οπτικοποιήσεις κ.α. Αυτό την καθιστά την κορυφαία μηχανή παιχνιδιών για ειδικούς. Παραδείγματα παιχνιδιών που υλοποιήθηκαν με Unreal είναι: PlayersUnknownBattlegrounds, Fortnite, Unreal Tournament κ.α.

Πλεονεκτήματα Unreal:

- Υποστήριξη όλων των πλατφορμών.
- Open source πηγαίος κώδικας.
- Blueprints.
- Ρεαλιστικά γραφικά.
- Πολύ αποδοτικό engine.

Μειονεκτήματα Unreal:

- Σχεδιασμένη κατά βάση για 1st/3rd person shooter.
- Χρήση C++ για προγραμματισμό.
- Δεν υποστηρίζει πλέον 2D εφαρμογές.
- Περιορισμένο asset marketplace.

- Περιορισμένη κοινότητα προγραμματιστών.
- Αρκετά μεγάλο build time.

2.3 Unity Engine

Η Unity είναι μια από τις κορυφαίες engines για την κατασκευή βιντεοπαιχνιδιών και εκδόθηκε από την Unity Technologies το 2005 κυρίως για MacOS, αλλά σήμερα υποστηρίζει ανάπτυξη video games για όλες τις πλατφόρμες. Η ανάπτυξη παιχνιδιών με την Unity είναι πιο προσβάσιμη για το κοινό, με σημαντική υποστήριξη ανάγνωση της οθόνης. Χρησιμοποιείται και αυτή για άλλα είδη δια δραστικών εφαρμογών όπως εκπαιδευτικές εξομοιώσεις, κινηματογράφος κ.α. Όσο αφορά τα παιχνίδια μεταξύ πολλαπλών πλατφορμών(cross-platform) χαρακτηρίζεται η κορυφαία του είδους. Επίσης χαρακτηρίζεται το πρωτοπόρο engine για ανάπτυξη παιχνιδιών για κινητά σε σύστημα Android. Παραδείγματα παιχνιδιών που υλοποιήθηκαν με Unreal είναι: Pokémon Go, Cuphead, Fall guys κ.α.

<u>Πλεονεκτήματα Unity:</u>

- Υποστήριξη όλων των πλατφορμών.
- Μεγάλη κοινότητα προγραμματιστών και διαδικτυακή υποστήριξη,
- Υποστήριξη 2D εφαρμογών.
- Γρήγορο build time.
- Χρήση C# για προγραμματισμό.

<u>Μειονεκτήματα Unity:</u>

- Απαγόρευση στον πηγαίο κώδικα(source code).
- Συχνά προβλήματα στην απόδοση (π.χ. MonoBehaviour).
- Όχι τόσο ρεαλιστικά γραφικά όπως η Unreal,αλλά υπάρχει σημαντική βελτίωση με το παρελθόν.
- Οι νέες λειτουργίες (Visual Scripting) δεν έχει δοκιμαστεί αρκετά.

Βάσει όλων των παραπάνω και υπολογίζοντας την μεγαλύτερη προσωπική μου εξοικείωση με την γλώσσα προγραμματισμού C#, αλλά και την πληθώρα υλικού προς έρευνα (tutorials στο internet, βιβλιογραφία κ.α.) επιλέχθηκε η χρήση του Unity Engine για την υλοποίηση του TRY TO LIVE. Πιο συγκεκριμένα η έκδοση 2019.4.20f1,η οποία περιλαμβάνει και μακροχρόνια υποστήριξη από την Unity.

2.4 Απαιτήσεις συστήματος

Οι ελάχιστες απαιτήσεις του υπολογιστικού συστήματος για την υλοποίηση βιντεοπαιχνιδιών στην έκδοση 2019.4.20f1 σύμφωνα με την Unity παρουσιάζονται στο παρακάτω πίνακα:

Minimum requirements	Windows
Operating system version	Windows 7 (SP1+), Windows 10 and Windows 11, 64-bit versions only.
CPU	X64 architecture with SSE2 instruction set support
Graphics API	DX10, DX11, and DX12-capable GPUs
Additional requirements	Hardware vendor officially supported drivers

Επίσης, οι ελάχιστες απαιτήσεις του υπολογιστικού συστήματος για την εκτέλεση βιντεοπαιχνιδιών στην έκδοση 2019.4.20f1 σύμφωνα με την Unity παρουσιάζονται στο παρακάτω πίνακα:

Operating system	Windows
Operating system version	Windows 7 (SP1+), Windows 10 and Windows 11
CPU	x86, x64 architecture with SSE2 instruction set support.
Graphics API	DX10, DX11, DX12 capable.
Additional requirements	Hardware vendor officially supported drivers.

Το σύστημα στο οποίο αναπτύξαμε και εκτελέσαμε την εφαρμογή έχει τις παρακάτω προδιαγραφές :

- Λειτουργικό σύστημα:Windows 10 Home 64 bit
- Επεξεργαστής: Intel i3-4160 @ 3.60 GHz
- Μνήμη: 8 Ram
- Κάρτα γραφικών: NVIDIA GeForce GTX 750

2.5 Εισαγωγικά στοιχεία για την C#

Όπως αναφέραμε για την υλοποίηση του βιντεοπαιχνιδιού χρησιμοποιήθηκε η γλώσσα προγραμματισμού C# η οποία έτρεξε μέσω του Visual Studio 2019. Η C# δημιουργήθηκε από την Microsoft μέσω της πλατφόρμας .NET με σκοπό να είναι μια απλή αντικειμενοστραφής γλώσσα για γενική χρήση, δανειζόμενη πολλά στοιχεία από την C++ και την Java.Έχει παρόμοια σύνταξη με την C++ και την Java αλλά δεν παράγει κατευθείαν κώδικα αλλά ένα ενδιάμεσο.

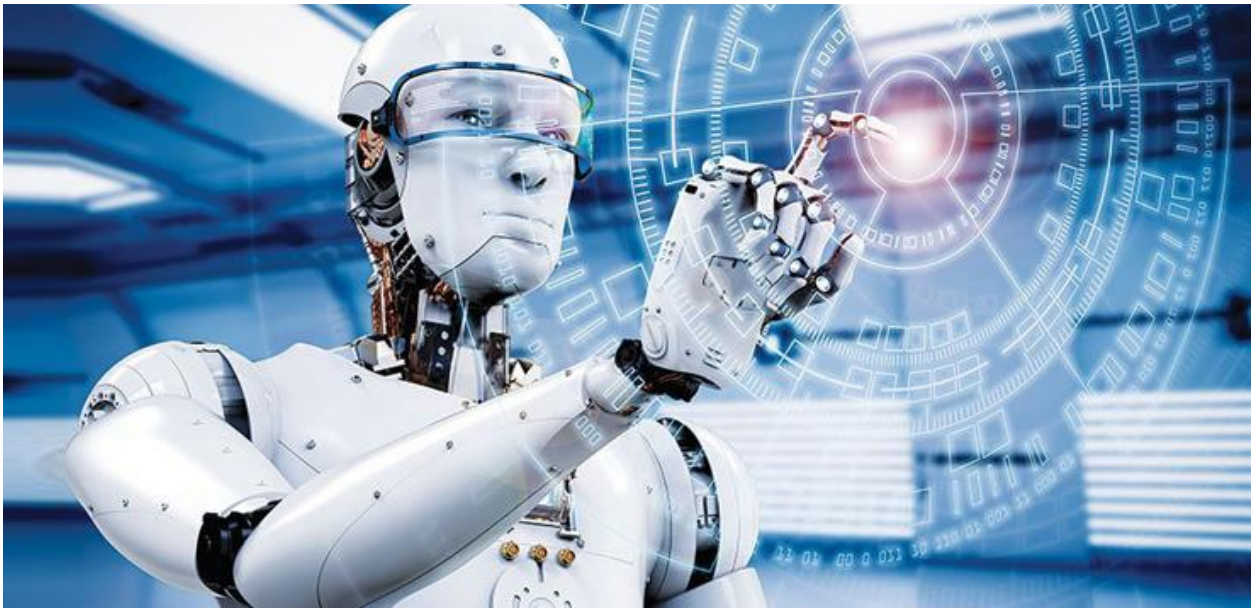
Η ανάπτυξη της λειτουργικότητας διαφόρων game objects στη Unity γίνεται μέσω C# scripts. Αυτά αλληλοεπιδρούν μεταξύ τους με σκοπό την δημιουργία όλων των λειτουργιών των interfaces, της λειτουργικότητας του avatar (κίνηση, επίθεση, κάμερα παίχτη κ.α.) αλλά και την επιθυμητή συμπεριφορά των NPCs. Η συμπεριφορά των NPCs είναι ευφυής, δηλαδή χρησιμοποιούνται τεχνικές Τεχνητής Νοημοσύνης, και συγκεκριμένα Μηχανές Πεπερασμένων Καταστάσεων (Finite State Machines). Θα αναλύσουμε περαιτέρω τα scripts του βιντεοπαιχνιδιού στην συνέχεια της διπλωματικής.

2.6 Τεχνητή νοημοσύνη στα βιντεοπαιχνίδια

Στα βιντεοπαιχνίδια, η τεχνητή νοημοσύνη (AI) χρησιμοποιείται για τη δημιουργία επιθυμητών ή έξυπνων συμπεριφορών κυρίως σε χαρακτήρες χωρίς παίκτες (NPC) παρόμοιες με την ανθρώπινη νοημοσύνη. Η τεχνητή νοημοσύνη αποτελεί αναπόσπαστο μέρος των βιντεοπαιχνιδιών από την έναρξή τους. Το AI στα βιντεοπαιχνίδια είναι ένα ξεχωριστό υπό πεδίο και χρησιμεύει στην βελτίωση της εμπειρίας του παιχνιδιού. Κατά τη χρυσή εποχή των βιντεοπαιχνιδιών “Arcade”,η ιδέα των αντιπάλων της τεχνητής νοημοσύνης διαδόθηκε σε μεγάλο βαθμό με τη μορφή κλιμακωτών επιπέδων δυσκολίας, διακριτών μοτίβων κίνησης και γεγονότων εντός του παιχνιδιού που εξαρτώνται από τη συμβολή του παίκτη. Τα σύγχρονα παιχνίδια συχνά εφαρμόζουν υπάρχουσες τεχνικές όπως η εύρεση μονοπατιών και τα δέντρα αποφάσεων για να καθοδηγήσουν τις ενέργειες των NPC. Η τεχνητή νοημοσύνη χρησιμοποιείται

συχνά σε μηχανισμούς που δεν είναι άμεσα ορατοί στον χρήστη, όπως η εξόρυξη δεδομένων και η παραγωγή διαδικαστικού περιεχομένου.

Ο όρος "παιχνίδι AI" αναφέρεται σε ένα μεγάλο σύνολο αλγορίθμων που περιλαμβάνει επίσης τεχνικές από τη θεωρία ελέγχου, τη ρομποτική, τα γραφικά υπολογιστών και την επιστήμη των υπολογιστών γενικά.



Το AI χρησιμοποιείτε σε μια μεγάλη ποικιλία από αρκετά ανόμοια πεδία μέσα σε ένα παιχνίδι. Το πιο προφανές είναι στον έλεγχο οποιουδήποτε NPC στο παιχνίδι, αν και το "scripting" (δένδρο αποφάσεων) είναι επί του παρόντος το πιο κοινό μέσο ελέγχου. Τα δέντρα αποφάσεων καταλήγουν συχνά σε επαναλαμβανόμενη συμπεριφορά, απώλεια εμβύθισης ή ανώμαλη συμπεριφορά σε καταστάσεις που δεν είχαν σχεδιάσει οι προγραμματιστές. Το Pathfinding, μια άλλη μεθοδολογία του AI, εμφανίζεται ευρέως σε παιχνίδια στρατηγικής σε πραγματικό χρόνο. Το Pathfinding είναι η μέθοδος για τον προσδιορισμό του τρόπου μεταφοράς ενός NPC από ένα σημείο ενός χάρτη σε άλλο, λαμβάνοντας υπόψη το έδαφος και τα εμπόδια. Αρκετά βιντεοπαιχνίδια χρησιμοποιούν συχνά γρήγορη και απλή "διαδρομή βάσει πλέγματος", όπου το έδαφος χαρτογραφείται σε ένα άκαμπτο πλέγμα ομοιόμορφων τετραγώνων και ένας αλγόριθμος εύρεσης μονοπατιών όπως A* ή IDA* εφαρμόζεται στο πλέγμα. Αντί για ένα άκαμπτο πλέγμα, ορισμένα παιχνίδια χρησιμοποιούν ακανόνιστα πολύγωνα και συναρμολογούν ένα πλέγμα πλοήγησης από τις περιοχές του χάρτη στις οποίες μπορούν να περπατήσουν τα NPC. Ως τρίτη μέθοδος, μερικές φορές είναι βολικό για τους προγραμματιστές να επιλέγουν με μη αυτόματο τρόπο "σημεία διαδρομής" που θα πρέπει να χρησιμοποιούν τα NPC για την πλοήγηση. Το πρόβλημα που προκύπτει είναι ότι τέτοια σημεία μπορούν να δημιουργήσουν αφύσικη κίνηση. Επιπλέον, τα σημεία διαδρομής τείνουν να έχουν χειρότερη

απόδοση από τα πλέγματα πλοήγησης σε πολύπλοκα περιβάλλοντα. Πέρα από τη στατική εύρεση μονοπατιών, η πλοήγηση είναι ένα υπό πεδίο της τεχνητής νοημοσύνης παιχνιδιού που εστιάζει στο να δώσει στα NPC τη δυνατότητα να προηγούνται σε ένα δυναμικό περιβάλλον, να βρίσκουν μια διαδρομή προς έναν στόχο αποφεύγοντας συγκρούσεις με άλλες οντότητες (άλλο NPC, παίκτες κ.α.) ή να συνεργάζονται (ομαδική πλοήγηση).

Πολλά σύγχρονα βιντεοπαιχνίδια εμπίπτουν στην κατηγορία της δράσης, βολής πρώτου προσώπου ή της περιπέτειας. Στα περισσότερα από αυτά τα είδη παιχνιδιών, υπάρχει κάποιο επίπεδο μάχης που λαμβάνει χώρα. Η ικανότητα του AI agent να είναι αποτελεσματικός στη μάχη είναι σημαντική σε αυτά τα είδη. Ένας κοινός στόχος των προγραμματιστών σήμερα είναι να γίνει το AI πιο ανθρώπινο ή τουλάχιστον να φαίνεται έτσι. Ένα από τα πιο θετικά και αποτελεσματικά χαρακτηριστικά που υπάρχουν στα σύγχρονο βιντεοπαιχνίδια AI είναι η ικανότητα κυνηγιού. Το AI αρχικά αντέδρασε με πολύ προβληματικό τρόπο. Εάν ο παίκτης βρισκόταν σε μια συγκεκριμένη περιοχή, τότε το AI θα αντιδρούσε είτε με εντελώς επιθετικό τρόπο είτε θα ήταν εντελώς αμυντικό. Τα τελευταία χρόνια έχει εισαχθεί η ιδέα του «κυνηγιού». Σε αυτήν την κατάσταση «κυνηγιού», η τεχνητή νοημοσύνη θα αναζητήσει ρεαλιστικούς δείκτες, όπως ήχους που παράγονται από τον χαρακτήρα ή ίχνη που μπορεί να έχουν αφήσει πίσω τους. Αυτές οι εξελίξεις τελικά επιτρέπουν μια πιο σύνθετη μορφή παιχνιδιού. Με αυτό το χαρακτηριστικό, ο παίκτης μπορεί πραγματικά να σκεφτεί πώς να πλησιάσει ή να αποφύγει έναν εχθρό. Αυτό είναι ένα χαρακτηριστικό που είναι ιδιαίτερα διαδεδομένο στο είδος stealth.

Η ακαδημαϊκή τεχνητή νοημοσύνη μπορεί να παίξει σημαντικό ρόλο εντός του Game AI, εκτός της παραδοσιακής ανησυχίας του ελέγχου της συμπεριφοράς του NPC.

-Μοντελοποίηση εμπειρίας παίκτη: Διάκριση της ικανότητας και της συναισθηματικής κατάστασης του παίκτη, έτσι ώστε να προσαρμόζεται κατάλληλα το παιχνίδι. Αυτό μπορεί να περιλαμβάνει δυναμική εξισορρόπηση δυσκολίας παιχνιδιού, η οποία συνίσταται στην προσαρμογή της δυσκολίας σε ένα βιντεοπαιχνίδι σε πραγματικό χρόνο με βάση την ικανότητα του παίκτη.

-Δημιουργία διαδικαστικού περιεχομένου: Δημιουργία στοιχείων του περιβάλλοντος του παιχνιδιού, όπως περιβαλλοντικές συνθήκες, επίπεδα, ακόμη και μουσική με αυτοματοποιημένο τρόπο. Οι μέθοδοι AI μπορούν να δημιουργήσουν νέο περιεχόμενο ή δια δραστικές ιστορίες.

-Εξόρυξη δεδομένων σχετικά με τη συμπεριφορά των χρηστών: Αυτό επιτρέπει στους σχεδιαστές παιχνιδιών να εξερευνήσουν πώς χρησιμοποιούν οι άνθρωποι το παιχνίδι, ποια μέρη παίζουν περισσότερο και τι τους προκαλεί να σταματήσουν να παίζουν, επιτρέποντας στους προγραμματιστές να συντονίσουν το παιχνίδι ή να το βελτιώσουν.

Εναλλακτικές προσεγγίσεις στα NPC: Αυτές περιλαμβάνουν την αλλαγή της ρύθμισης του παιχνιδιού για τη βελτίωση της αξιοπιστίας του NPC και την εξερεύνηση της κοινωνικής και όχι της ατομικής συμπεριφοράς NPC.

2.7 Τεχνική AI που χρησιμοποιήθηκε:FSM

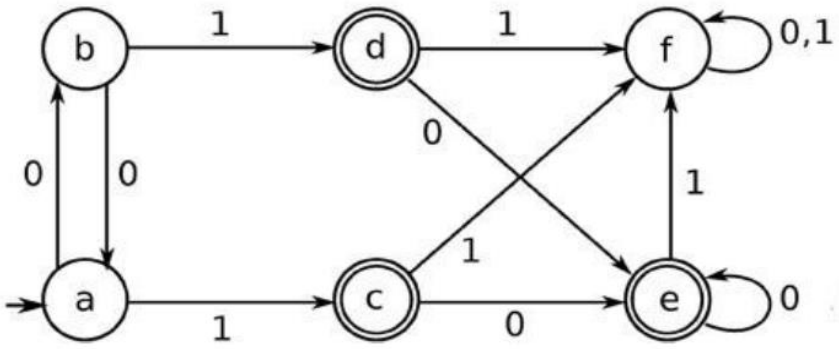
Πιο αναλυτικά για την εισαγωγή τεχνητής νοημοσύνης στα βιντεοπαιχνίδια υπάρχουν τέσσερις τεχνικές:

- Finite State Machines
- Decision Diagrams
- Behavior Trees
- Goal Oriented Action Planning

Στην συγκεκριμένη διπλωματική χρησιμοποιήθηκε **Finite State Machines**.

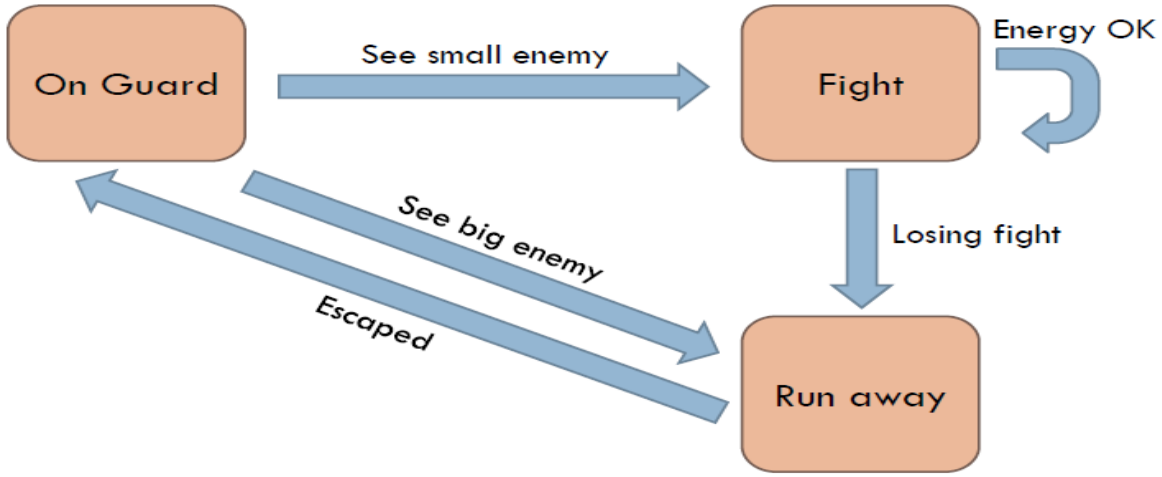
Οι πρώτες πρακτικές εφαρμογές της μηχανής πεπερασμένης κατάστασης (FSM) AI είναι σχεδόν τόσο παλιές όσο και τα ίδια τα βιντεοπαιχνίδια, αφού ακόμη και το ευρέως δημοφιλές "PacMan" χρησιμοποίησε ένα από αυτά τα κλασικά υπολογιστικά μοντέλα. Σχεδόν όλα τα παιχνίδια που παίζαμε και πολλά από τα σύγχρονα παιχνίδια που παίζουμε ακόμα χρησιμοποιούν AI που βασίζεται σε FSM. Χρησιμοποιείται όχι μόνο για τη δημιουργία αξιόπιστων αλληλεπιδράσεων μεταξύ του παίκτη και των άλλων χαρακτήρων, αλλά για την εφαρμογή άλλων κρίσιμων στοιχείων της ανάπτυξης του παιχνιδιού, όπως η γραφική διεπαφή χρήστη (GUI), ο χειρισμός εισόδου, τα στοιχεία ελέγχου του παίκτη και η εξέλιξη στο ιστορικό.

Ένα FSM είναι ένα αφηρημένο μοντέλο υπολογισμού που μπορεί να υπάρχει σε μία κατάσταση κάθε φορά (τρέχουσα κατάσταση), επιλεγμένο από μια πεπερασμένη επιλογή τιμών. Με άλλα λόγια, κάθε κατάσταση είναι μοναδική και αμοιβαία αποκλειστική και επομένως πεπερασμένη. Εισάγει σήμα όταν το μηχάνημα μπορεί να αλλάξει από την αρχική κατάσταση στην επόμενη σε μια μετάβαση που ορίζεται από τη διαδικασία.



Το αποτέλεσμα είναι ένα απίστευτα απλό αλλά κομψό μοντέλο που παρέχει όλες τις απαντήσεις που χρειάζονται οι περισσότεροι προγραμματιστές παιχνιδιών, ειδικά επειδή είναι αρκετά απλό που μπορεί να χρησιμοποιηθεί ακόμη και χωρίς προγραμματισμό. Οι δυνατότητες του παιχνιδιού μπορούν να υλοποιηθούν με τη δημιουργία διαγραμμάτων των FSM σε απευθείας γραφήματα μια απλή και κατανοητή μορφή που θα μπορούσε εύκολα να μετατραπεί ξανά σε πίνακες καταστάσεων.

Πιθανώς η πιο προφανής και ευρέως χρησιμοποιούμενη εφαρμογή των FSM στη βιομηχανία των βιντεοπαιχνιδιών είναι η δημιουργία στοιχειώδους αποτελεσματικής τεχνητής νοημοσύνης. Μια πεπερασμένη κατάσταση μπορεί να χρησιμοποιηθεί για να ορίσει ορισμένες συμπεριφορές χαρακτήρων χωρίς δυνατότητα αναπαραγωγής (NPC), όπως επίθεση, περιαγωγή ή τρέξιμο.



Συνοψίζοντας, στην τεχνική FSM έχουμε τα εξής πλεονεκτήματα:

-Παραδοσιακά μια από τις πρώτες τεχνικές συμπεριφοράς NPC.

-Πολύ απλή στην κατανόηση.

-Πολύ απλή στην εφαρμογή χρησιμοποιώντας δηλώσεις if-then-else.

-Διαχωρισμός μεταξύ της δουλειάς του προγραμματιστή και των σχεδιαστών παιχνιδιών

-Απλοϊκή στις συμπεριφορές που μπορούν να εκφραστούν

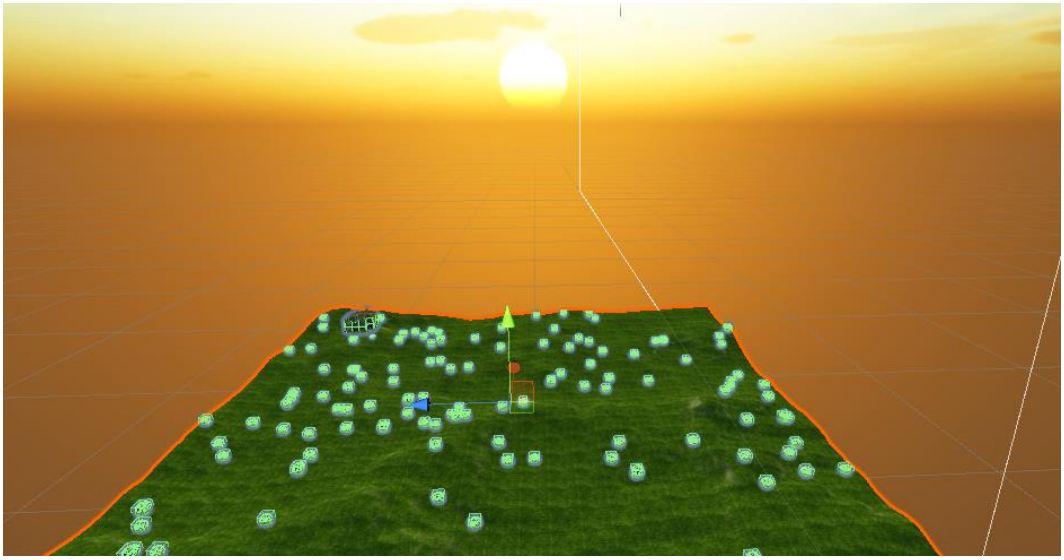
3 Σχεδίαση του TRY TO LIVE

3.1 Πλοκή

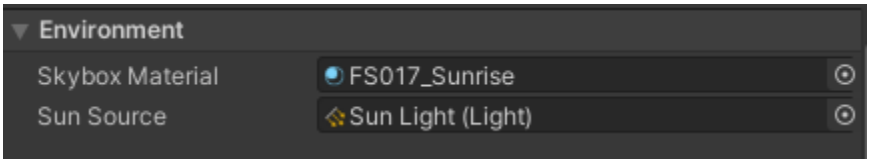
Το παιχνίδι διαδραματίζεται σε ένα δάσος στο οποίο ζουν διάφορα ζώα και ιθαγενείς. Στο δάσος αυτό έχει ξεσπάσει πανδημία η οποία έχει μετατρέψει όλα τα όντα σε νεκροζώντανη κατάσταση(zombies). Πιο συγκεκριμένα έχουν αλλάξει μορφή και έγιναν επιθετικά προς όλα τα όντα που δεν βρίσκονται στην ιδιά κατάσταση. Οι άνθρωποι έγιναν κανίβαλοι και τα ζώα σαρκοβόρα. Ο χαρακτήρας μας είναι από τους μοναδικούς που δεν έχει νοσήσει και προσπαθεί να επιβιώσει. Ο σκοπός του είναι να αποφύγει τα εμπόδια που του θέτουν τα zombies και να βρει κατάλυμα σε ένα σπίτι στο δάσος και να προστατευτεί.

3.2 Εικονικός Κόσμος

Ο κόσμος του βιντεοπαιχνιδιού είναι ένα open-world map το οποίο αναπαριστά δάσος.

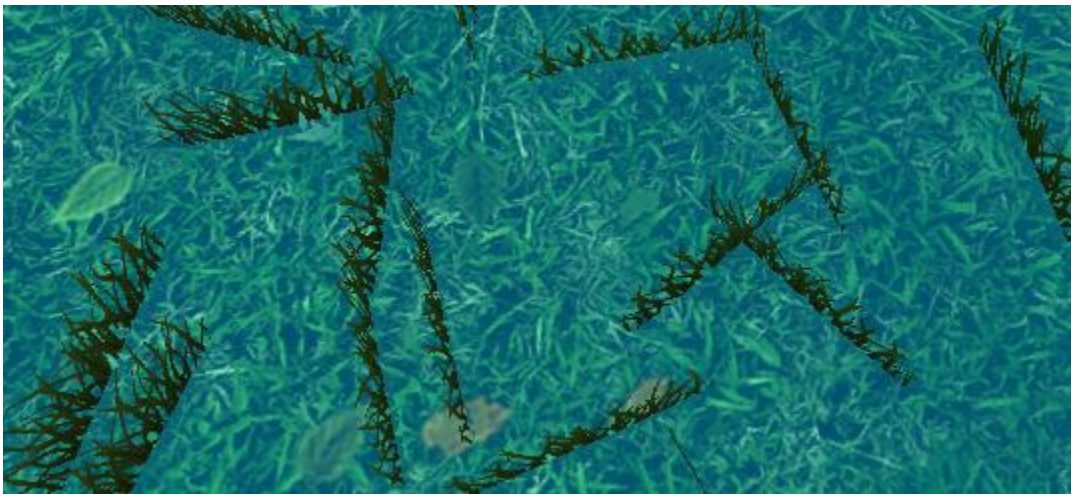


Σαν Skybox επιλέχθηκε μέσω του Asset Store το παρακάτω:



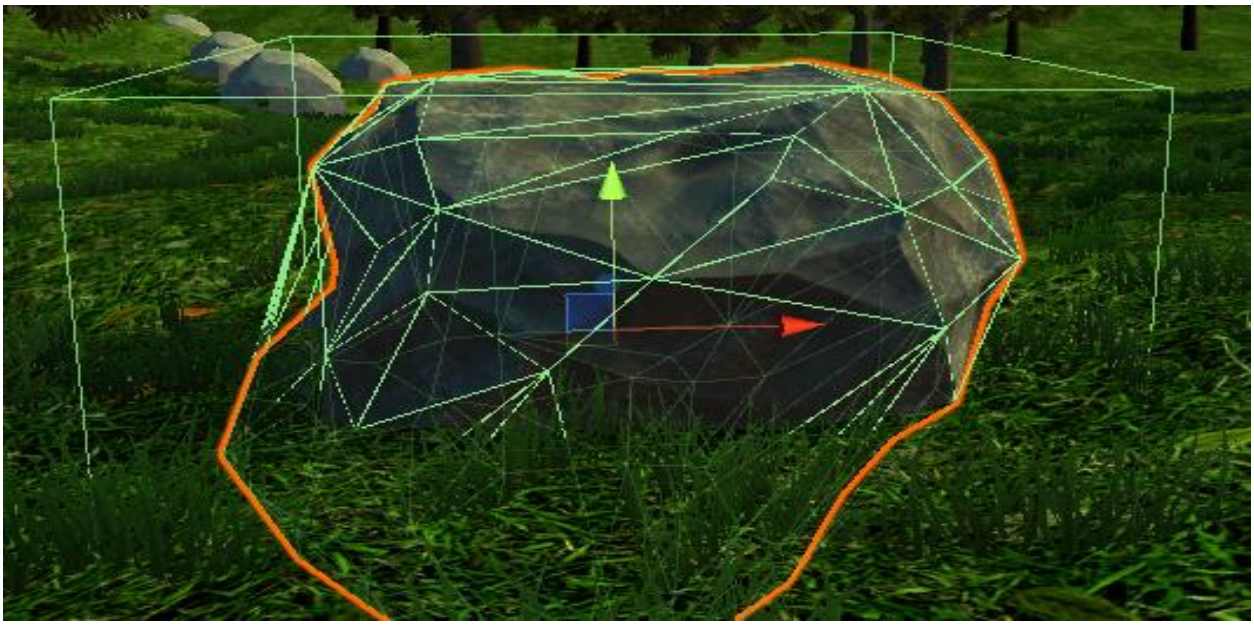
Το open-world map αποτελείται από:

- Το έδαφος, ο τύπος του είναι το γρασίδι.



- Δέντρα, τα οποία γίνονται auto-generate όσο προχωράει στο map ο χαρακτήρας μας, αυτόματη λειτουργία που προϋπήρχε στο asset που χρησιμοποιήθηκε.

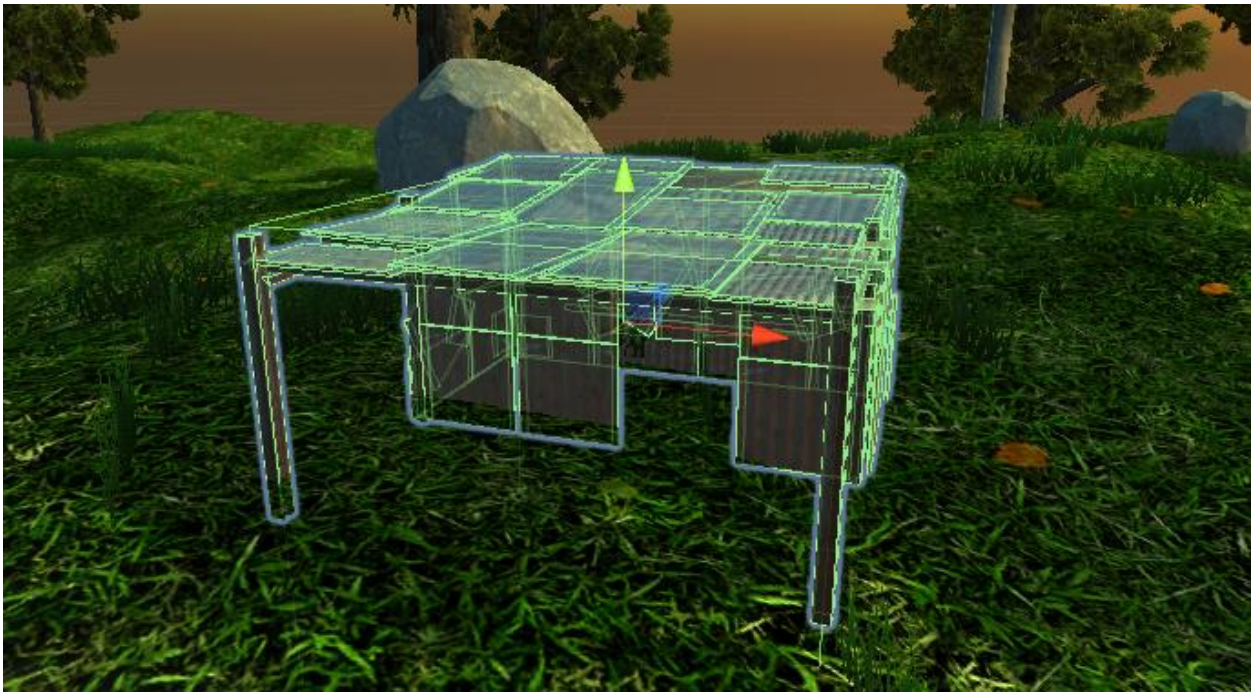
- Βράχους, οι οποίοι όπως φαίνεται και στην παρακάτω εικόνα αποτελούνται ο καθένας από collider σαν αποτέλεσμα να γίνονται εμπόδια στον χαρακτήρα, ο οποίος είτε θα πρέπει να τα προσπεράσει είτε να κάνει άλμα από πάνω τους



- Σπίτι, ο βασικός στόχος και του χαρακτήρα, αποτελείται από πολλά μικρότερα μέρη. Το καθένα μέρος έχει και τον δικό του collider.

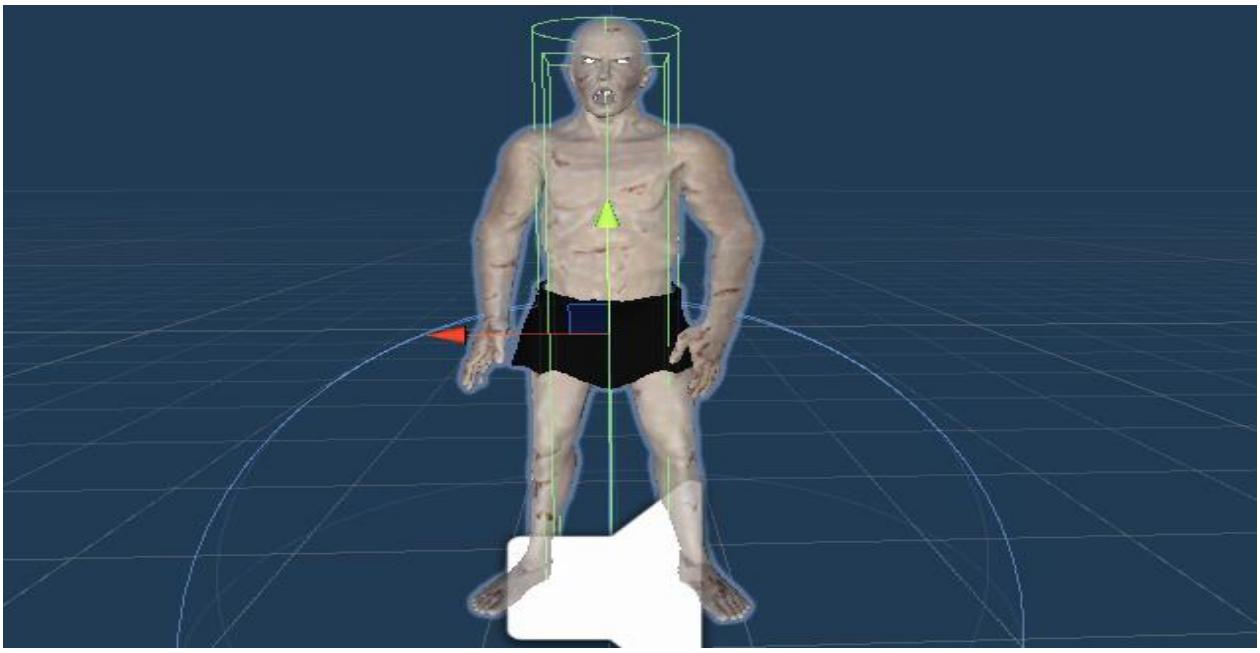
- Φανάρι, το οποίο έχει το δικό του collider.

- Καλύβα, βρίσκεται σε κοντινή απόσταση από το σπίτι και αυτή αποτελείται από πολλά μικρότερα μέρη. Το καθένα μέρος έχει και τον δικό του collider.

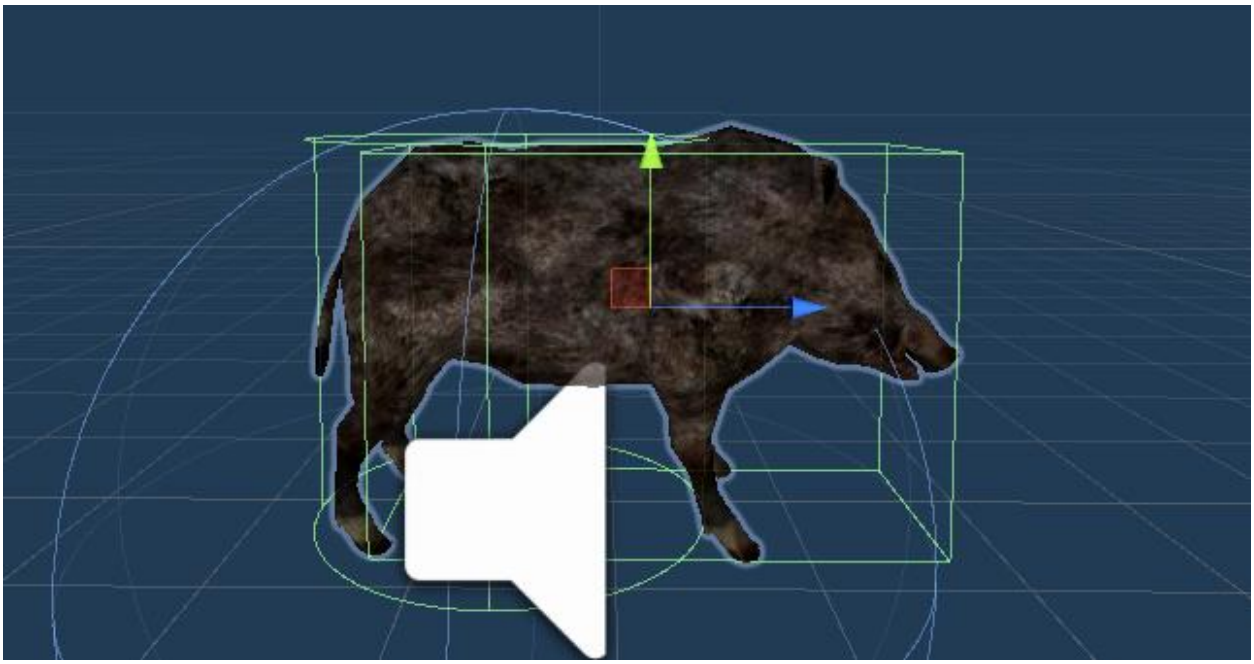


Επιπλέον, περιπολούν τον χάρτη δύο NPCs, οι βασικοί εχθροί του χαρακτήρα μας.

- Ο κανίβαλος, αποτελείται από nav mesh agent, animations, scripts, ηχητικά και collider.



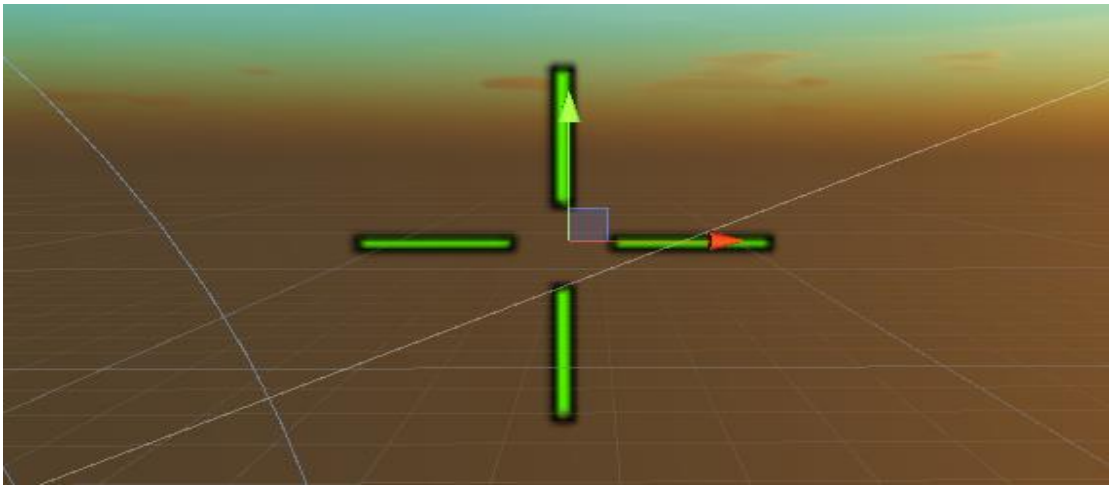
- Το αγριογούρουνο, αποτελείται και αυτό από nav mesh agent, animations, scripts, ηχητικά και collider.



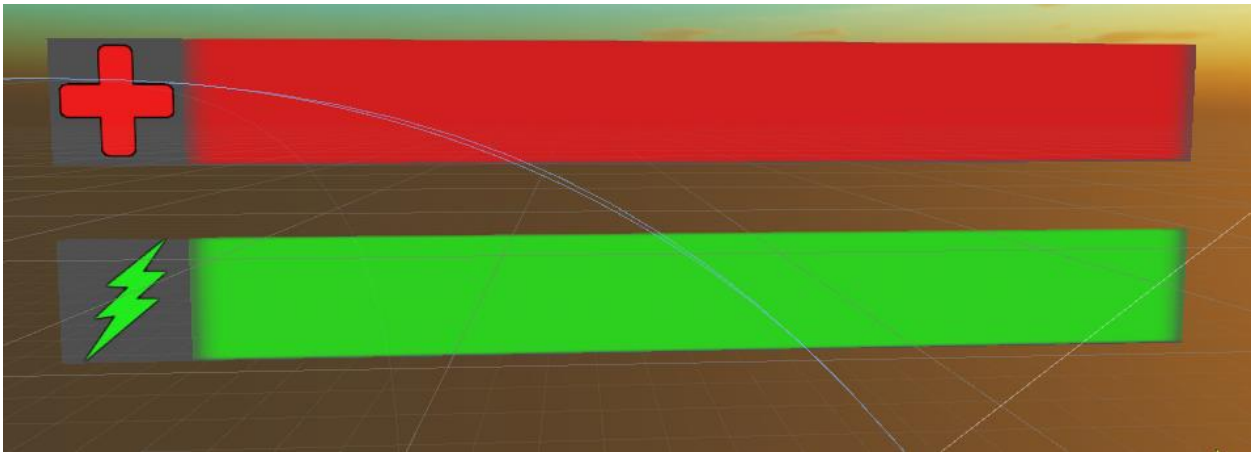
Τέλος, στον εικονικό μας κόσμο έχουμε και τον βασικό μας χαρακτήρα, που αλληλοεπιδρά σε αυτόν από τις επιλογές του χρήστη. Αποτελείται από scripts,ηχητικά αλλά και την βασική κάμερα του video game. Επιπρόσθετα ο χαρακτήρας έχει έξι διαθέσιμα όπλα. Το κάθε όπλο αποτελείται από ένα script και ξεχωριστά animations, ηχητικά, muzzle flash, σφαίρες ή βέλη.

3.3 UI Στοιχεία

Για τον στόχο του χαρακτήρα δημιουργήθηκε το UI Canvas Crosshair για να επιτευχθεί η σκόπευση. Προκειμένου να εμφανίζεται στον παίχτη καθ' όλη την διάρκεια του βιντεοπαιχνιδιού σε σταθερό σημείο και πιο συγκεκριμένα στο κέντρο της οθόνης τοποθετήθηκε εντός του prefab του παίχτη, εφόσον η κεντρική κάμερα έχει "δεθεί" με τον παίχτη. Εντός του UI Canvas ενσωματώθηκε η παρακάτω εικόνα UI:



Επιπρόσθετα, δημιουργήθηκε UI Canvas για την εμφάνιση των στατιστικών ζωής και αντοχής του χαρακτήρα. Κάθε στατιστικό αποτελείται από δύο UI εικόνες μία για το φόντο και μια για το προσκήνιο.Η UI εικόνα στο προσκήνιο θα μεταβάλλεται κατά την διάρκεια του παιχνιδιού βάση scripts. Τέλος, κάθε στατιστικό περιέχει και μια εικόνα συμβολισμού στην αρχή του προσκηνίου που θα δηλώνει την ιδιότητα του(σταυρός για την ζωή, κεραυνός για την αντοχή).



3.4 Ηχητικά Στοιχεία

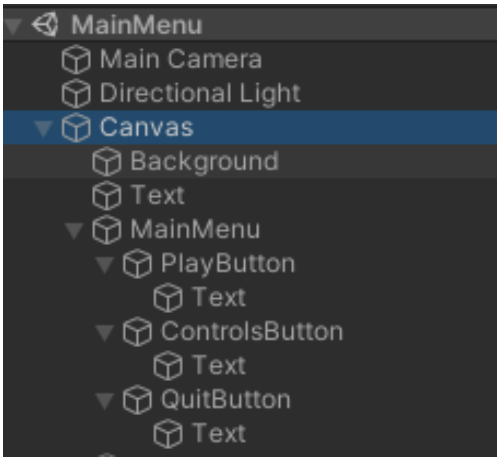
Στην εφαρμογή συναντάμε τα εξής ηχητικά στοιχεία:

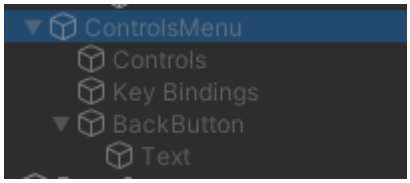
- Ambience sound: είναι η ηχητική απεικόνιση δάσους με ήχους φύλλων και πουλιών και θα παίζει σε επανάληψη κατά την διάρκεια του βιντεοπαιχνιδιού.
- Boar sounds: είναι σύνολο ηχητικών που συναντάμε στο prefab boar (ηχητικό επίθεσης, ηχητικό τραυματισμού, ηχητικό θανάτου, ηχητικό αδράνειας).
- Cannibal sounds: είναι σύνολο ηχητικών που συναντάμε στο prefab cannibal (ηχητικό επίθεσης, ηχητικό κραυγής, ηχητικό θανάτου).
- Footsteps sounds: είναι ηχητική απεικόνιση κίνησης του χαρακτήρα (ηχητικό βηματισμού, ηχητικό γρήγορης κίνησης, ηχητικό αργής κίνησης).
- Guns sounds: ηχητικά για τα όπλα του χαρακτήρα (ήχος πυροβολισμού περίστροφου, ήχος πυροβολισμού αυτόματου, ήχος πυροβολισμού καραμπίνας, ήχος οπλισμού καραμπίνας).
- Woosh sounds: ηχητικά για το τσεκούρι.

4 Αρχιτεκτονική εφαρμογής

4.1 Κεντρικό μενού

Για την υλοποίηση του κεντρικού μενού δημιουργήθηκε το scene Main Menu το οποίο θα τρέχει πριν από το βασικό scene του video game FFPSurvival. Δίνει δια δραστικές επιλογές με αριστερό κλικ στο ποντίκι στον χρήστη. Εντός του scene έχουμε ένα canvas και εντός αυτού ένα panel που τοποθετήθηκε εικόνα σαν φόντο. Επιπλέον εντός του canvas δημιουργήθηκαν τα πλήκτρα του μενού (Play,Controls,Quit) και εντός σε αυτά χρησιμοποιήθηκε ένα asset textmeshpro, για τις ονομασίες και γενικότερα τα κείμενα, καθώς περιέχει περισσότερες επιλογές σε σχέση με το απλό text.Τα τρία πλήκτρα ομαδοποιήθηκαν σε ένα κενό στοιχείο που ονομάστηκε Main Menu, με βάση αυτό δημιουργήθηκε και ένα δεύτερο κενό στοιχείο που θα περιέχει το μενού των πλήκτρων για το βιντεοπαιχνίδι αλλά και ένα πλήκτρο Back. Αυτά τα δύο στοιχεία αποτελούν και το κεντρικό μενού και βάση πλήκτρων θα γίνεται η εναλλαγή τους, η έναρξη του video game ή ο τερματισμός του.

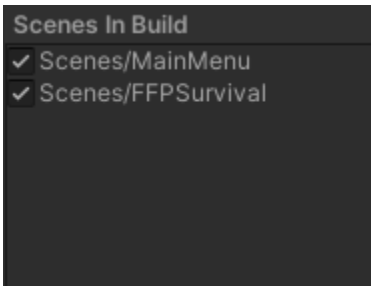




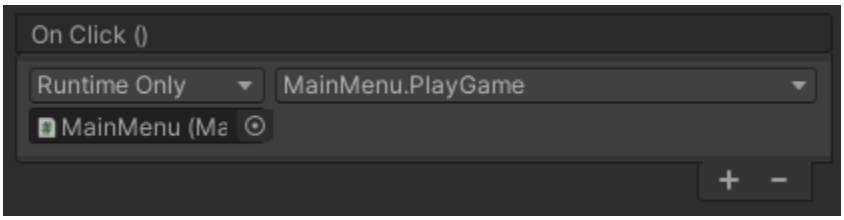
Για το κεντρικό μενού δημιουργήθηκε το script MainMenu. Όπως θα δούμε στην παρακάτω εικόνα στην public void Playgame επιτυγχάνουμε την φόρτωση του επόμενου scene από αυτό που θα τρέξει το script, ενώ με την public void QuitGame τερματίζεται το βιντεοπαιχνίδι.

```
public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

    }

    public void QuitGame()
    {
        Debug.Log("QUIT");
        Application.Quit();
    }
}
```

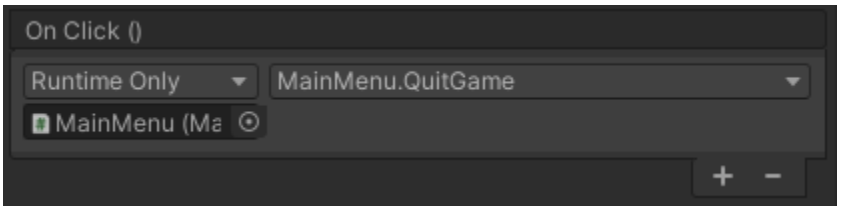
Η σειρά με την οποία ορίστηκαν τα scene της εφαρμογής:



Παρουσιάζεται η συνθήκη που ορίστηκε στο πλήκτρο Play ώστε να γίνεται έναρξη του game πραγματοποιώντας αλλαγή στα scene:

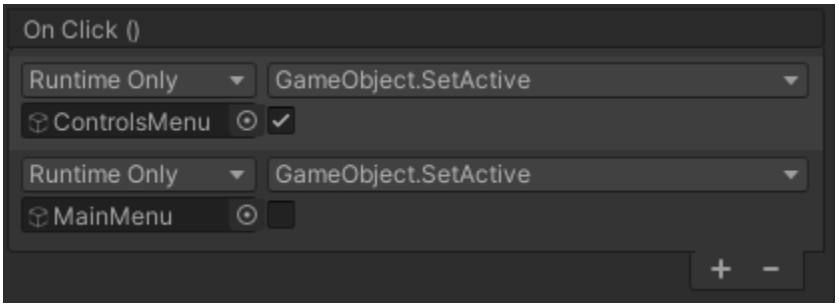


Επίσης παρουσιάζεται η συνθήκη που ορίστηκε στο πλήκτρο Quit ώστε να γίνεται τερματισμός του game:

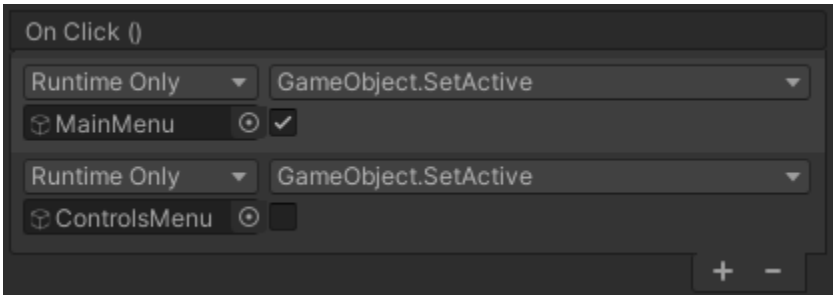


Για τα πλήκτρα Controls και Back δεν χρειάστηκε δημιουργία script αλλά ήταν απαραίτητη η δημιουργία συνθήκης από το ένα μενού προς το άλλο.

Συνθήκη πλήκτρου Controls:



Συνθήκη πλήκτρου Back:



Κεντρικό μενού:

Μενού πλήκτρων:

Με αφορμή την τελευταία εικόνα παραθέτετέ και πίνακας με τα πλήκτρα πλοήγησης και ενεργειών του παίχτη:

Ενέργεια	Πλήκτρο
Πυροβολισμός	LMB
Ζουμ όπλου	RMB
Κίνηση Εμπρός	W
Κίνηση Αριστερά	A
Κίνηση Δεξιά	S
Κίνηση Πίσω	D
Γρήγορη Κίνηση	Shift
Άλμα	Space
Σκυφτή θέση	C
Τσεκούρι	1
Περίστροφο	2
Καραμπίνα	3
Αυτόματο	4
Ακόντιο	5
Τόξο	6

4.2 Κύριο Scene του TRY TO LIVE

Για την διαδικασία υλοποίησης του TRY TO LIVE ήταν απαραίτητη η δημιουργία ενός scene (FFPSurvival) για την εισαγωγή και την εφαρμοσμένη λειτουργία όλων των asset και prefabs,καθώς και την προσθήκη scripts σε αυτά. Πιο συγκεκριμένα έγινε δημιουργία και επεξεργασία του game ground (asset), του χαρακτήρα μέσω του Player (prefab), του UI Canvas (asset), του Enemy Manager (αποτελείται από τα prefab boar και cannibal) ,του Ambient Sound. Πιο αναλυτικά συμπεράσματα δίνονται από την εικόνα που ακολουθεί:

4.3 Scripts για το κύριο Scene

Σε αυτό το σημείο θα γίνει παρουσίαση όλων των script που έδωσαν “ζωή” στο βιντεοπαιχνίδι. Όπως είπαμε και νωρίτερα έγινε χρήση του Visual Studio 2019 και προγραμματισμός σε C#. Για την καλύτερη λειτουργικότητα και διαφοροποίηση των script δημιουργήθηκαν οι παρακάτω ομάδες οι οποίες αποτελούνται:

-Player Scripts:

- AttackScript: Με αυτό το script γίνεται ο προγραμματισμός της επίθεσης του χαρακτήρα, ορίζεται το μέγεθος της βολής (σφαίρα) δημιουργώντας συνθήκη για το collider του με αποτέλεσμα να έρχεται σε επαφή με τα NPCs και να τους προκαλεί damage.
- *HealthScript: Στο συγκεκριμένο script αρχικά έγινε η αρχικοποίηση της ζωής των NPCs και του χαρακτήρα και στην συνέχεια προγραμματίστηκαν οι συνθήκες εξουδετέρωσης για καθένα από αυτά όταν η ζωή τους είναι ίση με μηδέν ή κάτω από μηδέν. Πιο συγκεκριμένα αν εξουδετερωθεί κάποιο NPC ενεργοποιείται από το script EnemyManager η επανεμφάνιση του, ενώ αν εξουδετερωθεί ο παίκτης γίνεται επανεκκίνηση της εφαρμογής.
- MouseLook: Script που αφορά τον προγραμματισμό της κάμερας μετακινώντας το ποντίκι σε κλίμακα 360 μοιρών σε τρις άξονες καθώς και την ευαισθησία κίνησης αυτού.
- **PlayerAttack: Στο παρόν script υλοποιήθηκε η επίθεση του χαρακτήρα, βασικές λειτουργίες των όπλων όπως το fire rate, χρόνος ανάμεσα στις επιθέσεις, zoom in και zoom out.Τα όπλα με σφαίρες ρυθμίστηκαν να πυροβολήσουν είτε με zoom in είτε με zoom out,ενώ το τόξο και το ακόντιο μόνο με zoom in για περισσότερο ρεαλισμό. Επίσης ο στόχος (crosshair) είναι ενεργοποιημένος μόνο με zoom out στα όπλα που πυροβολούν σφαίρες.
- PlayerAxeWooshSound: Με αυτό το script γίνεται ο ορισμός των ηχητικών για το όπλο “τσεκούρι”, καθώς διαφέρει από τα υπόλοιπα όπλα στην επίθεση (melee attack).
- PlayerFootSteps: Script για δημιουργία συνθηκών για αναπαραγωγή ήχου βηματισμού του χαρακτήρα, αλλαγή έντασης και αλλαγή ηχητικού ανάλογα την κίνηση του χαρακτήρα (περπάτημα, τρέξιμο).
- PlayerMovement: Στο συγκεκριμένο script ρυθμίζεται η κίνηση στον χώρο του χαρακτήρα και γίνεται η επίτευξη του άλματος με την προσθήκη βαρύτητας.
- PlayerSprintAndCrouch: Ο κώδικας που κρύβεται πίσω από την πληκτρολόγηση του Shift και του C. Πιο συγκεκριμένα προγραμματίστηκε με το πλήκτρο Sprint να αυξάνεται η ταχύτητα του παίχτη και να μετατρέπεται το ηχητικό σε ταχύτερο και παράλληλα να μειώνεται σταδιακά η μπάρα αντοχής του παίχτη. Μόλις τερματίσει η μπάρα ο παίχτης κινείται υποχρεωτικά αργά μέχρι να επανέλθει. Με το πλήκτρο C μειώνεται το ύψος του παίχτη και η κίνηση γίνεται πιο αργή μέχρι να ξαναπατηθεί το πλήκτρο.
- PlayerStats: Script για την αρχικοποίηση των στατιστικών της ζωής και της αντοχής του παίχτη και προβολή τους στον UI Canvas.

-Weapon Scripts:

- ArrowBowScript: Με αυτό το script δημιουργείται μέσω της επίθεσης του παίχτη, είτε με ακόντιο είτε με τόξο, το βέλος. Ορίζεται η ταχύτητα του, το ποσοστό που αφαιρεί ζωή όταν πετύχει τον στόχο αλλά και η πορεία του.
- WeaponHandler: Στο συγκεκριμένο script γίνονται οι αρχικοποιήσεις όλων των λειτουργιών των όπλων (τύπος πυροβολισμού, εφέ πυροβολισμού, animation πυροβολισμού, ζουμ σκόπευτρου, ήχος όπλισης, ήχος πυροβολισμού).
- WeaponManager: Εδώ ρυθμίζεται η εναλλαγή των όπλων του χρήστη μέσω των πλήκτρων 1 τσεκούρι, 2 περίστροφο, 3 καραμπίνα, 4 αυτόματο, 5 ακόντιο, 6 τόξο.

-Enemy Scripts:

- EnemyAnimator: Με αυτό το script γίνεται ορισμός των animations των NPCs που θα ενεργοποιούνται στο script EnemyConroller (Walk, Run, Attack, Dead).
- EnemyAudio: Με αυτό το script γίνεται ορισμός των ηχητικών των NPCs που θα ενεργοποιούνται στο script EnemyConroller (Scream Sound, Attack Sound, Dead Sound).
- ***EnemyController: Στο συγκεκριμένο script ρυθμίζονται σημαντικές λειτουργίες των εχθρών του παιχνιδιού, πιο συγκεκριμένα ξεκινώντας το παιχνίδι οι εχθροί αρχίζουν να περιπολούν στον χάρτη με ορισμένη ταχύτητα. Όταν ο παίχτης βρεθεί σε ορισμένη απόσταση τα NPCs ξεκινάνε να τρέχουν προς το μέρος του και να τον κυνηγάνε. Αν φτάσουν πάλι σε μια απόσταση που έχει οριστεί στον χαρακτήρα του επιτίθονται. Επίσης αν ο παίχτης βγει από την ορισμένη απόσταση που ξεκινάει η καταδίωξη από τα NPCs αυτά συνεχίζουν να περιπολούν. Παράλληλα στις εναλλαγές λειτουργιών των NPCs ενεργοποιούνται αντίστοιχα τα animations και τα audios. Τέλος, ρυθμίζετε πότε να ενεργοποιείται το σημείο επίθεσης που δημιουργεί damage των NPCs.

-Game Manager Scripts:

- ****EnemyManager: Με το συγκεκριμένο script γίνεται δημιουργία (spawn) σε 10 σημεία που έχουν οριστεί στον χάρτη τα 2 είδη NPCs. Επίσης, αν ο χαρακτήρας εξουδετερώσει κάποιο από τα δέκα NPCs , γίνεται υπολογισμός μέσω του EnemyManager για το ποιο λείπει και γίνεται επαναδημιουργία του στο σημείο που αρχικά δημιουργήθηκε. Τέλος έχει οριστεί συνθήκη ώστε ο αριθμός των NPCs να παραμένει ο ίδιος (5 cannibals,5 boars).

-Helper Scripts:

- TagHolder: Τέλος, στο συγκεκριμένο script ορίστηκαν ετικέτες σε strings που γίνεται χρήση τους πολλές φορές στα παραπάνω scripts, για περισσότερη ευκολία στον προγραμματισμό με σκοπό την αποφυγή λαθών.

4.4 Κώδικας σημαντικότερων Scripts

*** HealthScript**

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

public class HealthScript : MonoBehaviour
{
    private EnemyAnimator enemy_Anim;
    private NavMeshAgent navAgent;
    private EnemyController enemy_Controller;
```

```
public float health = 100f;

public bool is_Player, is_Boar, is_Cannibal;

private bool is_Dead;

private EnemyAudio enemyAudio;

private PlayerStats player_Stats;

void Awake()
{
    if(is_Boar || is_Cannibal)
    {
        enemy_Anim = GetComponent<EnemyAnimator>();
        enemy_Controller = GetComponent<EnemyController>();
        navAgent = GetComponent<NavMeshAgent>();
        // get enemy audio
        enemyAudio = GetComponentInChildren<EnemyAudio>();
    }
    if (is_Player)
    {
        if (is_Player)
        {
            player_Stats = GetComponent<PlayerStats>();
        }
    }
}
public void ApplyDamage(float damage)
{   //if we died don't execute the rest of the code
    if (is_Dead)
        return;
    health -= damage;

    if (is_Player)
    {
        //show the stats(display the health UI value)
        player_Stats.Display_HealthStats(health);
    }
    if(is_Boar || is_Cannibal)
    {
        if(enemy_Controller.Enemy_State== EnemyState.PATROL)
        {
            enemy_Controller.chase_Distance = 50f;
        }
    }

    if (health <= 0f)
    {
        PlayerDied();

        is_Dead = true;
    }


}//apply damage
```

```
    void PlayerDied()
    {
        if (is_Cannibal)
        {
            GetComponent<Animator>().enabled = false;
            GetComponent<BoxCollider>().isTrigger = false;
            GetComponent<Rigidbody>().AddTorque(-transform.forward * 50f);

            enemy_Controller.enabled = false;
            navAgent.enabled = false;
            enemy_Anim.enabled = false;
            StartCoroutine(DeadSound());

            //EnemyManager Spawn more enemies
            EnemyManager.instance.EnemyDied(true);

        }
        if (is_Boar)
        {
            navAgent.velocity = Vector3.zero;
            navAgent.isStopped = true;
            enemy_Controller.enabled = false;

            enemy_Anim.Dead();
            StartCoroutine(DeadSound());

            //EnemyManager Spawn more enemies
            EnemyManager.instance.EnemyDied(false);
        }
        if (is_Player)
        {
            GameObject[] enemies = GameObject.FindGameObjectsWithTag(Tags.ENEMY_TAG);
            for (int i = 0; i< enemies.Length; i++)
            {
                enemies[i].GetComponent<EnemyController>().enabled = false;

            }
            //cal enemy manager to stop spawning enemies
            EnemyManager.instance.StopSpawning();

            GetComponent<PlayerMovement>().enabled = false;
            GetComponent<PlayerAttack>().enabled = false;

GetComponent<WeaponManager>().GetCurrentSelectedWeapon().gameObject.SetActive(false);
        }
        if(tag == Tags.PLAYER_TAG)
        {
            Invoke("RestartGame", 3f);
        }
        else
        {
            Invoke("TurnOffGameObject",3f);

        }
    }// player died
```

```
    void RestartGame()
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene("FFPSurvival");
    }

    void TurnOffGameObject()
    {
        gameObject.SetActive(false);
    }

    IEnumerator DeadSound()
    {
        yield return new WaitForSeconds(0.3f);
        enemyAudio.Play_DeadSound();
    }


}//class
```

**** PlayerAttack**

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

public class PlayerAttack : MonoBehaviour
{
    private WeaponManager weapon_Manager;

    public float fireRate = 15f;
    private float nextTimeToFire;
    public float damage = 20f;

    private Animator zoomCameraAnim;
    private bool zoomed;

    private Camera mainCam;

    private GameObject crosshair;

    private bool is_Aiming;

    [SerializeField]
    private GameObject arrow_Prefab, spear_Prefab;

    [SerializeField]
    private Transform arrow_Spear_StartPosition;

    void Awake()
    {
        weapon_Manager = GetComponent<WeaponManager>();
        zoomCameraAnim                                                    =
transform.Find(Tags.LOOK_ROOT).transform.Find(Tags.ZOOM_CAMERA).GetComponent<Animator>();

        crosshair = GameObject.FindWithTag(Tags.CROSSHAIR);
```

```
        mainCam = Camera.main;
    }



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

    }

    // Update is called once per frame
    void Update()
    {
        WeaponShoot();
        ZoomInAndOut();
    }

    void WeaponShoot()
    {
        if (weapon_Manager.GetCurrentSelectedWeapon().fireType == WeaponFireType.MULTIPLE)
        {   //if we press and hold left mouse click AND
            //if Time is greater than the nextTimeToFire
            if(Input.GetMouseButton(0) && Time.time> nextTimeToFire)
            {
                nextTimeToFire = Time.time + 1f / fireRate;
                weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();

                BulletFired();
            }
        }//if we have a regular weapon that shoots once
        else
        {
            if (Input.GetMouseButtonDown(0))
            {   //handle axe
                if (weapon_Manager.GetCurrentSelectedWeapon().tag == Tags.AXE_TAG)
                {
                    weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();

                }
                //handle shoot
                if        (weapon_Manager.GetCurrentSelectedWeapon().bulletType        ==
WeaponBulletType.BULLET)
                {
                    weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();
                    BulletFired();

                }
                else
                {//we have an arrow or spear
                    if (is_Aiming)
                    {
                        weapon_Manager.GetCurrentSelectedWeapon().ShootAnimation();

                        if    (weapon_Manager.GetCurrentSelectedWeapon().bulletType    ==
WeaponBulletType.ARROW)
                        {
```

```
                            //throw arrow
                            ThrowArrowOrSpear(true);
                        }

                        else if (weapon_Manager.GetCurrentSelectedWeapon().bulletType ==
WeaponBulletType.SPEAR)
                        {

                            //throw Spear
                            ThrowArrowOrSpear(false);
                        }
                    }
                }//else
            }//if input get mouse button 0
        }//else
    }//weapon shoot

    void ZoomInAndOut()
    {   //we are going to aim with our camera on the weapon
        if(weapon_Manager.GetCurrentSelectedWeapon().weapon_Aim== WeaponAim.AIM)
        {   //if we press and hold right mouse button
            if (Input.GetMouseButtonDown(1))
            {
                zoomCameraAnim.Play(AnimationTags.ZOOM_IN_ANIM);

                crosshair.SetActive(false);


            }
            //when we release the right mouse button click
            if (Input.GetMouseButtonUp(1))
            {
                zoomCameraAnim.Play(AnimationTags.ZOOM_OUT_ANIM);

                crosshair.SetActive(true);


            }

        }//if we need to zoom the weapon
        if (weapon_Manager.GetCurrentSelectedWeapon().weapon_Aim == WeaponAim.SELF_AIM)
        {
            if (Input.GetMouseButtonDown(1))
            {
                weapon_Manager.GetCurrentSelectedWeapon().Aim(true);
                is_Aiming = true;
            }

            if (Input.GetMouseButtonUp(1))
            {
                weapon_Manager.GetCurrentSelectedWeapon().Aim(false);
                is_Aiming = false;
            }
        }//weapon self aim

    }//zoom in zoom out

    void ThrowArrowOrSpear(bool throwArrow)
```

```
    {
        if (throwArrow)
        {
            GameObject arrow = Instantiate(arrow_Prefab);
            arrow.transform.position = arrow_Spear_StartPosition.position;

            arrow.GetComponent<ArrowBowScript>().Launch(mainCam);
        }
        else
        {
            GameObject spear = Instantiate(spear_Prefab);
            spear.transform.position = arrow_Spear_StartPosition.position;

            spear.GetComponent<ArrowBowScript>().Launch(mainCam);
        }
    }//throw arrow or spear

    void BulletFired()
    {
        RaycastHit hit;

        if(Physics.Raycast(mainCam.transform.position,    mainCam.transform.forward,    out
hit))//FPS
        {
           if(hit.transform.tag == Tags.ENEMY_TAG)
            {
                hit.transform.GetComponent<HealthScript>().ApplyDamage(damage);

            }
        }

    }

}//class
```

***** EnemyController**

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

public enum EnemyState
{
    PATROL,
    CHASE,
    ATTACK
}


public class EnemyController : MonoBehaviour
```

```
{
    private EnemyAnimator enemy_Anim;
    private NavMeshAgent navAgent;

    private EnemyState enemy_State;

    public float walk_Speed = 0.5f;
    public float run_Speed = 4f;

    public float chase_Distance = 7f;
    private float current_Chase_Distance;
    public float attack_Distance = 1.8f;
    public float chase_After_Attack_Distance = 2f;

    public float patrol_Radius_Min = 20f, patrol_Radius_Max = 60f;
    public float patrol_For_This_Time = 15f;
    private float patrol_Timer;

    public float wait_Before_Attack = 2f;
    private float attack_Timer;

    private Transform target;

    public GameObject attack_Point;

    private EnemyAudio enemy_Audio;


    void Awake()
    {
        enemy_Anim = GetComponent<EnemyAnimator>();
        navAgent = GetComponent<NavMeshAgent>();


        target = GameObject.FindWithTag(Tags.PLAYER_TAG).transform;
        enemy_Audio = GetComponentInChildren<EnemyAudio>();

    }

    // Start is called before the first frame update
    void Start()
    {
        enemy_State = EnemyState.PATROL;

        patrol_Timer = patrol_For_This_Time;

        //when the enemy first gets to the player
        //attack right away
        attack_Timer = wait_Before_Attack;

        //memorize the value of chase distance
        //so that we can put it back

        current_Chase_Distance = chase_Distance;

    }

    // Update is called once per frame
```

```
void Update()
{
    if (enemy_State == EnemyState.PATROL)
    {
        Patrol();
    }
    if (enemy_State == EnemyState.CHASE)
    {
        Chase();
    }
    if (enemy_State == EnemyState.ATTACK)
    {
        Attack();
    }


}

void Patrol()
{
    //tell nav agent that he can move
    navAgent.isStopped = false;
    navAgent.speed = walk_Speed;

    //add to the patrol timer
    patrol_Timer += Time.deltaTime;

    if(patrol_Timer> patrol_For_This_Time)
    {
        SetNewRandomDestination();

        patrol_Timer = 0f;


    }
    if (navAgent.velocity.sqrMagnitude > 0)
    {
        enemy_Anim.Walk(true);


    }
    else
    {
        enemy_Anim.Walk(false);
    }
    //test the distance between the player and the enemy
    if (Vector3.Distance(transform.position, target.position) <= chase_Distance)
    {
        enemy_Anim.Walk(false);
        enemy_State = EnemyState.CHASE;
        // play spotted audio
        enemy_Audio.Play_ScreamSound();

    }


}//patrol
```

```
void Chase()
{
    //enable the agent to move again
    navAgent.isStopped = false;
    navAgent.speed = run_Speed;

    //set the player's position as the destination
    //because we are chasing(running towards) the player

    navAgent.SetDestination(target.position);

    if (patrol_Timer > patrol_For_This_Time)
    {
        SetNewRandomDestination();

        patrol_Timer = 0f;


    }
    if (navAgent.velocity.sqrMagnitude > 0)
    {
        enemy_Anim.Run(true);


    }
    else
    {
        enemy_Anim.Run(false);
    }

    if (Vector3.Distance(transform.position,target.position)<= attack_Distance)
    {
        //stop the animations
        enemy_Anim.Run(false);
        enemy_Anim.Walk(false);
        enemy_State = EnemyState.ATTACK;
        //rest the chase distance
        if (chase_Distance != current_Chase_Distance)
        {
            chase_Distance = current_Chase_Distance;
        }
    }else if(Vector3.Distance(transform.position, target.position) > chase_Distance)
    {
        //player run away from enemy

        //stop running
        enemy_Anim.Run(false);

        enemy_State = EnemyState.PATROL;

        //reset the patrol timer so that the function
        //can calculate the new patrol destination right away
        patrol_Timer = patrol_For_This_Time;

        //rest the chase distance
        if (chase_Distance != current_Chase_Distance)
        {
            chase_Distance = current_Chase_Distance;
```

```
            }

        }//else

    }//chase

    void Attack()
    {

        navAgent.velocity = Vector3.zero;
        navAgent.isStopped = true;

        attack_Timer += Time.deltaTime;

        if(attack_Timer > wait_Before_Attack)
        {
            enemy_Anim.Attack();
            attack_Timer = 0f;

            //play attack sound
            enemy_Audio.Play_AttackSound();

        }
        if(Vector3.Distance(transform.position,target.position)>
            attack_Distance + chase_After_Attack_Distance)
        {
            enemy_State = EnemyState.CHASE;
        }


    }


    void SetNewRandomDestination()
    {
            float rand_Radius = Random.Range(patrol_Radius_Min, patrol_Radius_Max);

            Vector3 randDir = Random.insideUnitSphere * rand_Radius;
            randDir += transform.position;
            NavMeshHit navHit;

            NavMesh.SamplePosition(randDir, out navHit, rand_Radius, -1);

            navAgent.SetDestination(navHit.position);

    }

    void Turn_On_AttackPoint()
    {
        attack_Point.SetActive(true);
    }

    void Turn_Off_AttackPoint()
    {
        if (attack_Point.activeInHierarchy)
        {
            attack_Point.SetActive(false);
        }
```

```
    }
    public EnemyState Enemy_State
    {
        get; set;

    }


}//class
```

***** EnemyManager**

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

public class EnemyManager : MonoBehaviour
{
    public static EnemyManager instance;

    [SerializeField]
    private GameObject boar_Prefab, cannibal_Prefab;

    public Transform[] cannibal_SpawnPoints, boar_SpawnPoints;

    [SerializeField]
    private int cannibal_Enemy_Count, boar_Enemy_Count;

    private int initial_Cannibal_Count, initial_Boar_Count;

    public float wait_Before_Spawn_Enemies_Time = 10f;

    void Awake()
    {
        MakeInstance();
    }

    void Start()
    {
        initial_Cannibal_Count = cannibal_Enemy_Count;
        initial_Boar_Count = boar_Enemy_Count;

        SpawnEnemies();

        StartCoroutine("CheckToSpawnEnemies");


    }

    void MakeInstance()
    {
        if(instance == null)
        {
```

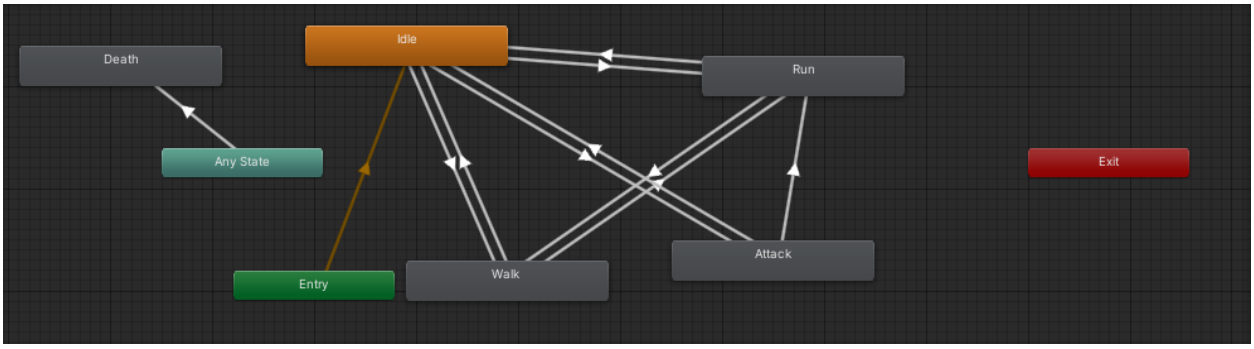
```
            instance = this;
        }

    }

    void SpawnEnemies()
    {
        SpawnCannibals();
        SpawnBoars();

    }

    void SpawnCannibals()
    {
        int index = 0;

        for (int i=0; i < cannibal_Enemy_Count; i++)
        {
            if(index>= cannibal_SpawnPoints.Length)
            {
                index = 0;

            }

            Instantiate(cannibal_Prefab,            cannibal_SpawnPoints[index].position,
Quaternion.identity);
            index++;


        }

        cannibal_Enemy_Count = 0;


    }

    void SpawnBoars()
    {
        int index = 0;

        for (int i = 0; i < boar_Enemy_Count; i++)
        {
            if (index >= boar_SpawnPoints.Length)
            {
                index = 0;

            }

            Instantiate(boar_Prefab,                  boar_SpawnPoints[index].position,
Quaternion.identity);
            index++;


        }

        boar_Enemy_Count = 0;
    }
```

```
    IEnumerator CheckToSpawnEnemies()
    {
        yield return new WaitForSeconds(wait_Before_Spawn_Enemies_Time);
        SpawnCannibals();

        SpawnBoars();

        StartCoroutine("CheckToSpawnEnemies");
    }

    public void EnemyDied(bool cannibal)
    {
        if (cannibal)
        {
            cannibal_Enemy_Count++;
            if(cannibal_Enemy_Count > initial_Cannibal_Count)
            {
                cannibal_Enemy_Count = initial_Cannibal_Count;

            }

        }
        else
        {
            boar_Enemy_Count++;
            if (boar_Enemy_Count > initial_Boar_Count)
            {
                boar_Enemy_Count = initial_Boar_Count;

            }
        }
    }
    public void StopSpawning()
    {
        StopCoroutine("CheckToSpawnEnemies");
    }

}//class
```

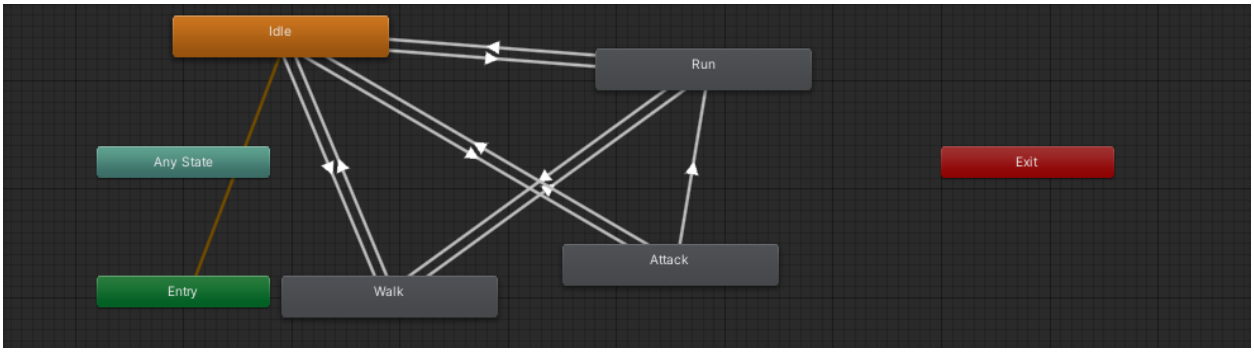
4.5 Animators Εφαρμογής

Εκτός από την δημιουργία scripts εξίσου σημαντική διαδικασία είναι και η δημιουργία animators ώστε να επιτευχθεί η επικοινωνία μεταξύ των scripts, η εναλλαγή καταστάσεων, η εναλλαγή ηχητικών αλλά και η εναλλαγή κινήσεων εντός της εφαρμογής. Τα animators που κατασκευάστηκαν εμφανίζονται παρακάτω:

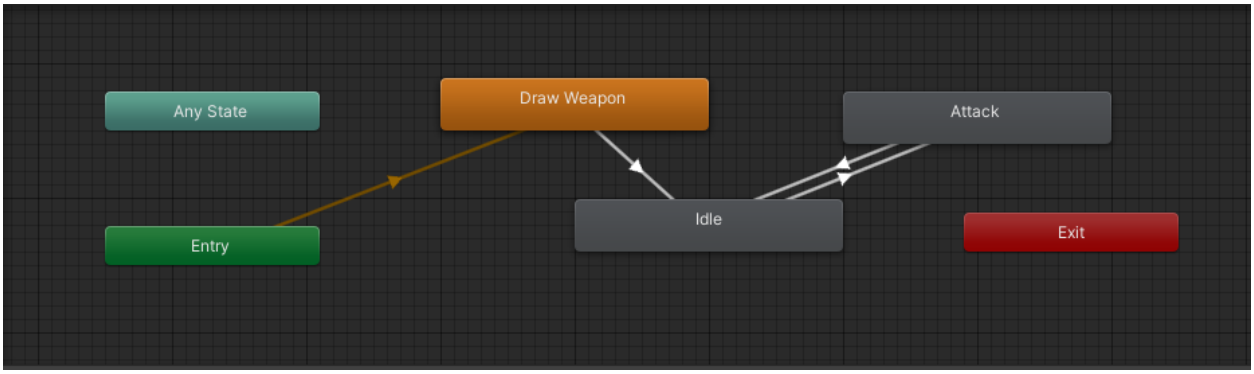
- Boar Animator:



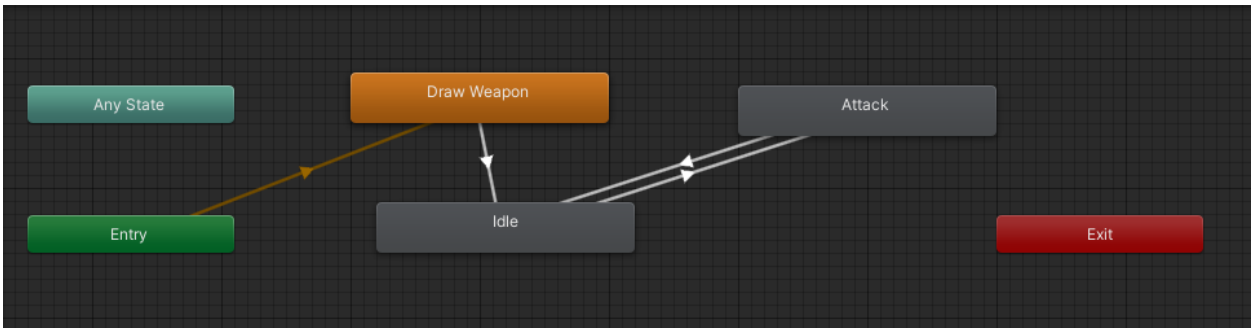
- Cannibal Animator:



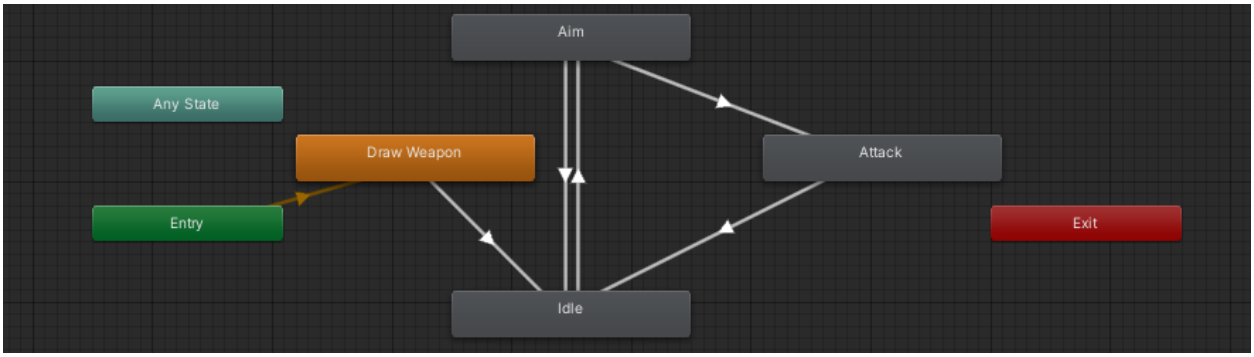
- Axe Animator:



- Weapons Animator:



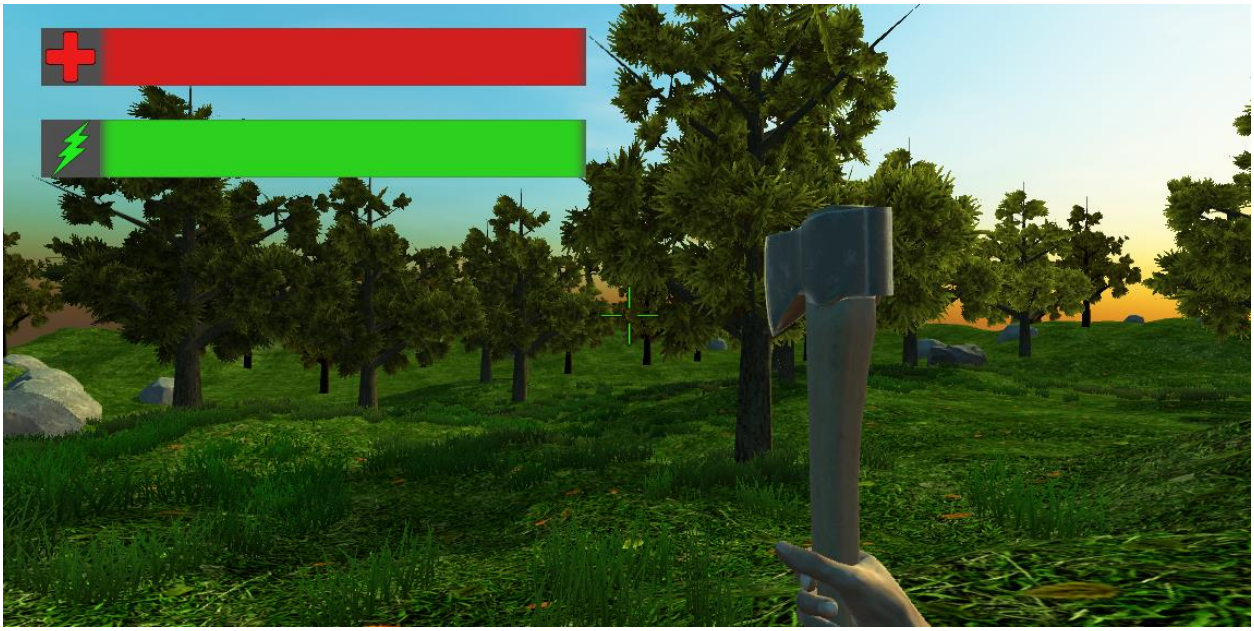
- Spear/Bow Animator:



5 Εκτέλεση Εφαρμογής

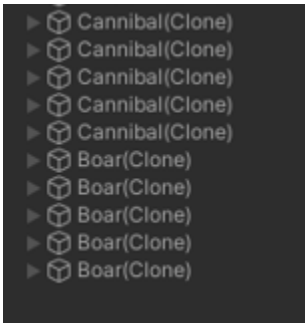
5.1 Παρουσίαση Walkthrough

Ο χρήστης επιλέγοντας από το κεντρικό μενού Start εκκινεί την εφαρμογή. Ο χαρακτήρας εμφανίζεται σε ένα άκρο του χάρτη όπου ξεκινάει και η περιπέτεια του.



Ξεκινώντας ο παίχτης κάνοντας χρήση των πλήκτρων 1,2,3,4,5,6 έχει την δυνατότητα να κάνει αλλαγή στα όπλα του.

Επιπρόσθετα, με την έναρξη απευθείας κάνουν spawn 5 NPCs cannibals και 5 NPCs boars τα οποία ξεκινάνε την περιπολία στο map.



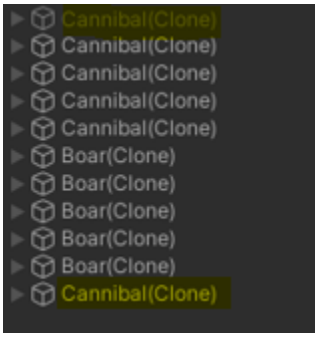
Ένα από τα NPCs cannibal έχει ξεκινήσει την περιπολία του, όπως θα δούμε και παρακάτω.

Μόλις ο χαρακτήρας εισέλθει στο πεδίο καταδίωξης το NPC ξεκινάει να τον κυνηγάει.

Όταν έρθει σε κοντινή απόσταση δημιουργεί damage στον παίχτη, μειώνοντας το ποσοστό ζωής όπως φαίνεται στην παρακάτω εικόνα.

Ο παίχτης με δύο βολές (έτσι έχει ρυθμιστεί) εξουδετερώνει το NPC.

Μετά από λίγο χρονικό διάστημα δημιουργείται ο αντικαταστάτης του στην ίδια θέση εμφάνισης στον χάρτη.



Ο χρήστης επιλέγοντας το πλήκτρο Space ο χαρακτήρας ενεργοποιεί την ικανότητα άλματος.

Με αυτόν τον τρόπο μπορεί να σκαρφαλώσει στους βράχους που τον περιτριγυρίζουν.

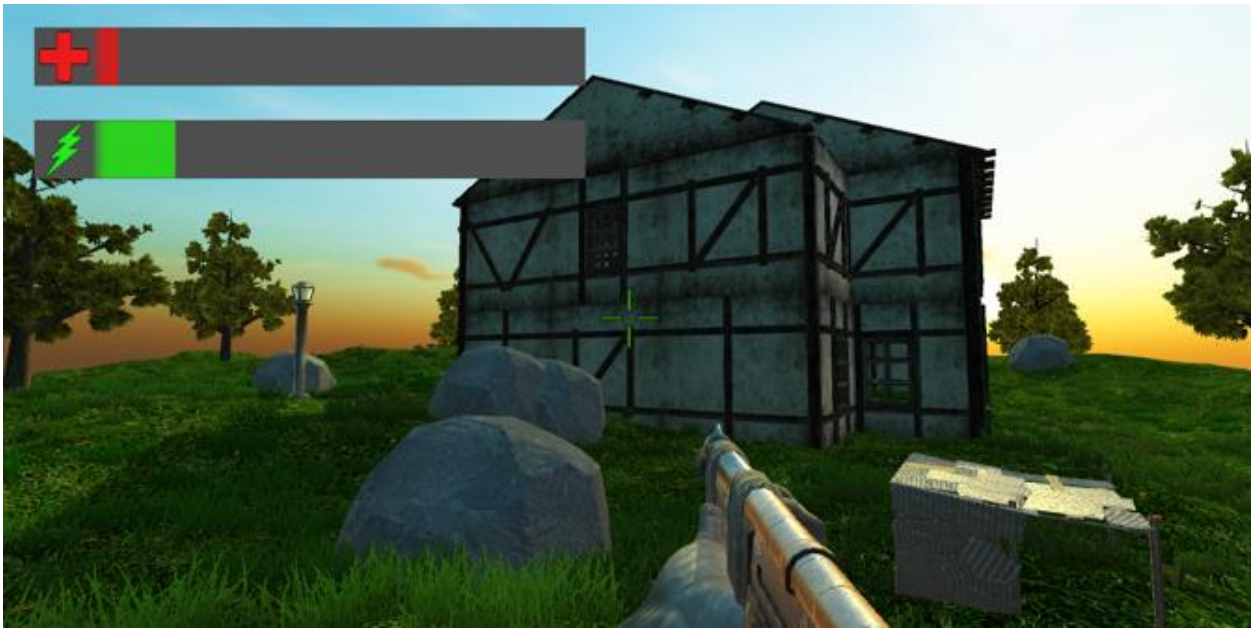
Επίσης, επιλέγοντας C σκύβει και σηκώνεται σε όρθια στάση επιλέγοντας το ίδιο πλήκτρο.

Με την χρήση του key Shift ο χαρακτήρας ξεκινάει να τρέχει μειώνοντας το ποσοστό αντοχής όπως θα δούμε και παρακάτω. Αφήνοντας το Shift επανέρχεται σιγά σιγά και πάλι η αντοχή.

Όσο προχωράει στον χάρτη ο παίχτης συναντάει και τα υπόλοιπα NPCs που του επιτίθονται αν τον εντοπίσουν ή αν τους επιτεθεί αυτός.

Παρακάτω φαίνεται η εξουδετέρωση ενός NPC boar.

Ο βασικός προορισμός είναι το σπίτι που φαίνεται παρακάτω το οποίο θα πρέπει να το πλησιάσει και να κρυφτεί σε αυτό ώστε να προστατευτεί.



Τέλος, να αναφέρουμε πως όταν η ζωή του παίχτη φτάσει στο μηδέν η κάτω από αυτό γίνεται επανεκκίνηση του βιντεοπαιχνιδιού και επαναφορά στην θέση έναρξής του.

6 Επίλογος

6.1 Συμπεράσματα

Κατά την διάρκεια εκπόνησης της εργασίας αποκτήθηκε εμπειρία και κατανόηση σε σημαντικό φάσμα των δυνατοτήτων της Unity όπως τεχνικές τεχνητής νοημοσύνης, scripting, navmeshagents, animation, sound effects κ.α. Η Unity αποτελεί μια εύχρηστη μηχανή βιντεοπαιχνιδιών και εφαρμογών, αρκετά εύκολη στην εκμάθηση καθώς υπάρχει μεγάλη υποστήριξη από την εταιρεία κατασκευής της αλλά και από τους χρήστες της. Η ανάπτυξη ενός βιντεοπαιχνιδιού απαιτεί χρόνο καθώς και αρκετές δοκιμές και ανατροφοδοτήσεις πληροφοριών και εντυπώσεων από τους παίχτες με αποτέλεσμα την συνεχή βελτιστοποίηση του. Όλα τα παραπάνω είχαν σαν αποτέλεσμα μια δυνατή προσωπική εμπειρία και ένα “κυνήγι” γνώσεων για την υλοποίηση του βιντεοπαιχνιδιού.

6.2 Μελλοντικές επεκτάσεις

Με την ολοκλήρωση της διπλωματικές διατριβής προκύπτουν σκέψεις για μελλοντικές επεκτάσεις της. Ορισμένες από αυτές είναι:

- Προσθήκη δεύτερου παίχτη για co-op παιχνίδι.
- Επέκταση του χάρτη και της ιστορίας.
- Προσθήκη παραπάνω και διαφορετικών NPCs εχθρών.
- Προσθήκη επιπλέον όπλων.

-Link στο οποίο υπάρχει διαθέσιμο το exe. του videogame για λήψη:

https://drive.google.com/file/d/1duAHkKzxuENEYj99HPTTgf-8U_SH6EsQ/view?usp=sharing

-Link στο οποίο υπάρχει διαθέσιμο βίντεο με την προσομοίωση του videogame για λήψη:

https://drive.google.com/file/d/1MzBYil1J7phtDeBfFbq7agDinMUPOFdO/view?usp=sharing

7 Βιβλιογραφία-Links

- https://en.wikipedia.org/wiki/Unity_(game_engine)
- https://en.wikipedia.org/wiki/Unreal_Engine
- https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
- https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games
- https://www.techopedia.com/finite-state-machine-how-it-has-affected-your-gaming-for-over-40-years/2/33996
- https://www.youtube.com/channel/UC9XFMlFNvaEzomThNeRaKFg YouTube Channel XyloBetaGames
- https://www.youtube.com/channel/UC8butISFwT-Wl7EV0hUK0BQ YouTube Channel freeCodeCamp.org
- https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA YouTube Channel Brackeys.
- Rob Miles, C# Programming Yellow Book, “Bananas” Edition 7.0, September 2015, University of Hull.
- Rob Miles, C# Programming Yellow Book, “Rubber Duck” Edition 5.1, January 2014, University of Hull.