Τμήμα Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ

Πτυχιακή Εργασία

Σχεδιασμός και Ανάπτυξη εικονικού περιβάλλοντος με την χρήση της μηχανής Unity

Νικόλαος Πετρόπουλος (ΑΜ 2546)
Σωτήριος Ρούσσης (ΑΜ 2422)

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ
Αθανάσιος Κούτρας

Πάτρα, 2022

[Αυτή η σελίδα είναι κενή]

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΜΗ ΛΟΓΟΚΛΟΠΗΣ

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Ακόμα δηλώνω ότι αυτή η γραπτή εργασία προετοιμάστηκε από εμένα προσωπικά και αποκλειστικά και ειδικά για την συγκεκριμένη πτυχιακή εργασία και ότι θα αναλάβω πλήρως τις συνέπειες εάν η εργασία αυτή αποδειχθεί ότι δεν μου ανήκει.

Νικόλαος Πετρόπουλος ΑΜ 2546

Σωτήριος Ρούσσης ΑΜ 2422

[Αυτή η σελίδα είναι κενή]

ΕΥΧΑΡΙΣΤΙΕΣ

Η συγκεκριμένη πτυχιακή εργασία ολοκληρώθηκε έπειτα από αρκετό χρονικό διάστημα, όχι από δυσκολία στο πλαίσιο της υλοποίησης αλλά από τις αμέτρητες προσπάθειες βελτίωσης κάθε γραμμής κώδικα ως προς την απόδοσή του καθώς επίσης από τις στιγμές που υλοποιούσαμε κάτι συγκεκριμένο με πολλούς διαφορετικούς τρόπους ώστε να αποκτήσουμε την ευελιξία και την εμπειρία πάνω στο Unity και στη γλώσσα προγραμματισμού C#.

Θα θέλαμε να ευχαριστήσουμε εκείνους που είχαν την πρόθεση να αναρτήσουν σχετικά άρθρα και videos στο διαδίκτυο τα οποία παρείχαν την κατάλληλη σαφήνεια και καθοδήγηση ώστε να φτάσουμε στο επίπεδο ολοκλήρωσης του στόχου μας. Συγκεκριμένα, ευχαριστούμε θερμά τον Brackeys (YouTube channel) και την development ομάδα του Unity3d για το καθαρό και ευανάγνωστο documentation.

[Αυτή η σελίδα είναι κενή]

ΠΡΟΛΟΓΟΣ

Το περιεχόμενο αυτής της εργασίας περιγράφει τα βασικά χαρακτηριστικά και τις λειτουργίες του Unity καθώς επίσης και τη διαδρομή που ακολουθήσαμε για την υλοποίηση του παιχνιδιού. Τονίζονται σημεία στα οποία, εμείς ως προγραμματιστές ή/και game designers, τα βρήκαμε ενδιαφέροντα αλλά και βασικά για τη δομή ενός single player παιχνιδιού.

[Αυτή η σελίδα είναι κενή]

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία πραγματεύεται την κατασκευή ενός 3D βιντεοπαιχνιδιού μέσω της μηχανής Unity η οποία παρέχει στους προγραμματιστές εργαλεία για την κατασκευή και ανάπτυξη δισδιάστατων και τρισδιάστατων παιχνιδιών. Στο τρισδιάστατο αυτό περιβάλλον έγινε χρήση της γλώσσας προγραμματισμού C# με την οποία δημιουργήθηκαν τα κατάλληλα scripts έτσι ώστε το παιχνίδι να είναι πλήρως λειτουργικό. Για να γραφτούν αυτά τα scripts, δηλαδή, ο κώδικας του παιχνιδιού χρησιμοποιήθηκε η μηχανη Unity σε συνδυασμό με το Visual Studio Code (Source Code Editor). Για την επεξεργασία των φωτογραφιών έγινε χρήση του προγράμματος GIMP (GNU Image Manipulation Program). Ενώ για την επεξεργασία ήχου έγινε η χρήση του προγράμματος Audacity.

Στο πρώτο κεφάλαιο της συγκεκριμένης εργασίας πρόκειται να αναπτυχθεί το θεωρητικό πλαίσιο γύρω από την κατασκευή των βιντεοπαιχνιδιών, την ιστορία τους και τα είδη που έχουν αναπτυχθεί. Στη συνέχεια ακολουθεί το δεύτερο και μεγαλύτερο μέρος της εργασίας το οποίο αφορά στην περιγραφή της διαδικασίας δημιουργίας του παιχνιδιού μας.

ABSTRACT

This thesis deals with the construction of a 3D video game through the Unity engine which provides developers with tools for the construction and development of 2D and 3D games. In this 3D environment, the programming language C# was used, with which the appropriate scripts were created so that the game is fully functional. To write these scripts, that is, the game code, the Unity engine was used in combination with Visual Studio Code (Source Code Editor). The GIMP (GNU Image Manipulation Program) program was used to edit the photos. While the Audacity program was used for sound processing.

In the first chapter of this work, the theoretical framework around the construction of video games, their history and the genres that have been developed will be developed. Then follows the second and largest part of the work, which concerns the description of the process of creating our game.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Unity, c#, video-game, προγραμματισμός, παιχνίδι

[Αυτή η σελίδα είναι κενή]

ΠΕΡΙΕΧΟΜΕΝΑ

[Αυτή η σελίδα είναι κενή]

ΕΙΣΑΓΩΓΗ

Τα βιντεοπαιχνίδια ή ηλεκτρονικά παιχνίδια είναι διαδραστικά παιχνίδια τα οποία λειτουργούν σε κάποιο κύκλωμα υπολογιστή. Οι μηχανές στις οποίες μπορούν να αναπτυχθούν και να λειτουργήσουν αυτού του είδους τα παιχνίδια μπορεί να είναι κοινόχρηστοι ή προσωπική υπολογιστές γενικής χρήσης, κονσόλες arcade, κονσόλες βίντεο συνδεδεμένες σε οικιακές τηλεοράσεις, φορητές κονσόλες και κινητά τηλέφωνα. Επίσης, ο όρος μπορεί να αναφέρεται μόνο σε παιχνίδια τα οποία παίζονται σε συσκευές με οθόνες για αναπαραγωγή βίντεο όπως η τηλεόραση, οι κονσόλες arcade και οι ηλεκτρονικοί υπολογιστές με τους οποίους ασχολείται και η εργασία μας (Henry E. Lowood, 2009).

Το ερώτημα που προκύπτει σε αυτό το σημείο αφορά στον λόγο για τον οποίο υπάρχουν και βρίσκουν τόσο τρομερή ανταπόκριση τα βιντεοπαιχνίδια. Αν και η απάντηση σε αυτό το ερώτημα δεν έχει απαντηθεί με βεβαιότητα από τους επιστήμονες του χώρου υπάρχουν μερικές θεωρίες που θα μπορούσαν να δώσουν ως έναν βαθμό μια λογική εξήγηση. Για παράδειγμα, η επιστήμη της βιολογίας υποστηρίζει πως καθώς υπάρχουν μορφές ηλεκτρονικών παιχνιδιών τα οποία αποτελούν μια εικονική αναπαράσταση της πραγματικότητας, πιθανότατα τα άτομα τα οποία συμμετέχουν σε αυτήν την πραγματικότητα αναπτύσσουν ικανότητες οι οποίες θα τους είναι χρήσιμες στην πραγματική ζωή. Φυσικά, όσο λογική και να φαίνεται μια τέτοια θεωρία δεν θα μπορούσε και πάλι να εξηγήσει τον λόγο για τον οποίο τα παιχνίδια έχουν την μορφή την οποία έχουν καθώς και την διαφορετική συμπεριφορά και στάση των ατόμων-παικτών απέναντι στο βιντεοπαιχνίδι. Όσον αφορά στους λόγους για τους οποίους συγκεκριμένα παιχνίδια αρέσουν σε συγκεκριμένους ανθρώπους και στο πώς αυτοί οι άνθρωποι επηρεάζονται από αυτά τα παιχνίδια, δεν υπάρχει κάποια σαφής εξήγηση παρά μόνο υποθέσης περί ηλικίας, φύλου, κοινωνικής τάξης και διαφόρων άλλων παραγόντων οι οποιοι διαμορφώνουν τους παίκτες (Egenfeldt-Nielsen, Smith & Tosca, 2020).

Το ερώτημα αυτό το οποίο μπορούμε να απαντήσουμε με μια βεβαιότητα είναι αυτό που σχετίζεται με τον τρόπο και τα εργαλεία που χρησιμοποιούμε για να δημιουργήσουμε τα

βιντεοπαιχνίδια. Το συγκεκριμένο ζήτημα έχει μεγάλο ενδιαφέρον καθώς οι τρόποι και τα εργαλεία αυτά συνεχώς ανανεώνονται και εκσυγχρονίζονται ταυτοχρόνως προς την εξέλιξη της τεχνολογίας. Αυτό σημαίνει πως οι μηχανές κατασκευής video games που γνωρίζουμε και χρησιμοποιούμε στις μέρες μας δεν υπήρχαν πριν λίγα χρόνια. Παραδείγματος χάρη, οι πρώτες κονσόλες βιντεοπαιχνιδιών και οι υπολογιστές δεν διέθεταν πλαίσιο για εγγραφή κώδικα βάσει του οποίου να κατασκευαστεί το παιχνίδι αλλά οι προγραμματιστές δημιουργούσαν περιεχόμενο από την αρχή. Όταν πλέον η εποχή άρχισε να απαιτεί καλύτερα γραφικά και περισσότερα επαγγελματικά βιντεοπαιχνίδια οι προγραμματιστές χρειαζόταν να διαθέτουν συγκεκριμένες κατευθυντήριες γραμμές και εργαλεία ώστε να μπορούν να κατασκευάζουν πιο εξελιγμένα και εντυπωσιακά γραφικά έχοντας συγκεκριμένα θεμέλια για να δημιουργήσουν περιεχόμενο. Αυτό έγινε εφικτό με τις μηχανές κατασκευής βιντεοπαιχνιδιών οι οποίες προσέφεραν έτοιμα εργαλεία ακόμα και στους ανεξάρτητους προγραμματιστές για να δημιουργήσουν περιεχόμενο (Nicoll, B., Keogh, B. & Springerlink, 2019).

[Αυτή η σελίδα είναι κενή]

1. Το Ιστορικό Πλαίσιο των Μηχανών Παιχνιδιού και η Μηχανή Unity

1.1 Η Πορεία προς τις Μηχανές Κατασκευής Παιχνιδιών

Πριν την κατασκευή και την εκτεταμένη χρήση των game engines, η τεχνολογία που χρησιμοποιούνταν για την δημιουργία προσομοιώσεων και εικονικής πραγματικότητας ήταν ιδιαίτερα ακριβή χωρίς, ωστόσο, να προσφέρει τα καλύτερα δυνατά γραφικά. Στη σημερινή εποχή τη δυνατότητα για την κατασκευή των καλύτερων γραφικών που δημιουργούν τις πιο ρεαλιστικές εικονικές πραγματικότητες, την προσφέρει η ίδια τεχνολογία η οποία χρησιμοποιείται για την κατασκευή βιντεοπαιχνιδιών (Lewis & Jacobson, 2002). Πριν όμως φτάσουμε στην χρήση των game engines, τα πρώτα βιντεοπαιχνίδια συνδέονταν απευθείας σε αναλογικούς υπολογιστές λυχνίας κενού και συγχρονίζονταν με οθόνες παλμογράφου. Επιπλέον, όπως αναφέρεται και παραπάνω, πολλές κονσόλες και υπολογιστές δεν διέθεταν code frameworks με βάση τα οποία θα μπορούσε να δημιουργηθεί περιεχόμενο αλλά οι προγραμματιστές δημιουργούσαν το περιεχόμενο τους από την αρχή.

Την δεκαετία του 1990, στην δημιουργία των βιντεοπαιχνιδιών άρχισαν να συμμετέχουν έκτος από τους προγραμματιστές καλλιτέχνες και σχεδιαστές, κατασκευάζοντας πιο περίπλοκα και βαριά σε περιεχόμενο παιχνίδια. Για αυτόν τον λόγο προέκυψε η ανάγκη να δημιουργηθούν σταθερά εργαλεία τα οποία θα μπορέσουν να βοηθήσουν την οργάνωση και την αποτελεσματικότερη εργασία όλων αυτών των παραγόντων. Μέσα σε αυτές τις καινούργιες συνθήκες που προέκυψαν από την ανάγκη των καταναλωτών-παικτών για καλύτερα γραφικά και πιο περίπλοκα παιχνίδια άλλαξε και ο ρόλος των προγραμματιστών. Ενώ παλαιότερα έγραφαν κώδικα και σχεδίαζαν το παιχνίδι, πλέον θα έπρεπε να δημιουργήσουν τα κατάλληλα εργαλεία τα οποία θα χρησιμοποιούσαν οι σχεδιαστές και οι καλλιτέχνες για να επεξεργαστούν το περιεχόμενο του παιχνιδιού.

Ο όρος *game engine* χρησιμοποιήθηκε για πρώτη φορά το 1993 και περιγράφει ένα λογισμικό εργαλείο το οποίο επιτρέπει την δημιουργία διαδραστικού ψηφιακού περιεχομένου σε πραγματικό χρόνο. Επίσης περιλαμβάνει code framework για την εγγραφή κώδικα ο οποίος να μπορεί να είναι συμβατός με διαφορες συσκευές και πλατφόρμες όπως κινητά και κονσόλες. Η μηχανή παιχνιδιών δεν κατασκευάζει απλώς βιντεοπαιχνίδια αλλά, όπως είπαμε, διαδραστικό ψηφιακό υλικό σε πραγματικό χρόνο. Για αυτόν τον λόγο δεν χρησιμοποιείται μόνο στον τομέα των video games αλλά και για τη δημιουργία 3D animations, αρχιτεκτονικά μοντέλα, περίπλοκων προσομοιώσεων που χρησιμοποιούνται για εκπαίδευση και εξάσκηση κ.ά. Το σημαντικότερο όμως είναι ότι προσφέρει στους χρήστες της την δυνατότητα να δημιουργήσουν έχοντας ήδη έτοιμες κάποιες βάσεις στις οποίες έχουν προσβασιμότητα χωρίς κάποιο υπέρογκο κόστος.

Αυτή τη στιγμή, στην αγορά υπάρχουν οι υπηρεσίες που προσφέρουν οι εμπορικές μηχανές παιχνιδιού χρησιμοποιούνται από τους περισσότερους προγραμματιστές για την κατασκευή των δικών τους παιχνιδιών. Ωστόσο, μεγάλες εταιρίες κατασκευής παιχνιδιών αν και χρησιμοποιούν μηχανες όπως η Unreal και η Unity, κατασκευάζουν και πάλι τα δικά τους λογισμικά εργαλεία για το τελικό τους έργο. Αυτό συμβαίνει διότι η μηχανές κατασκευής παιχνιδιών γενικής χρήσης ίσως να μην ανταποκρίνονται πάντα στις πολύ υψηλές τεχνολογικές και προγραμματιστικές προδιαγραφές των εταιρειών αυτών. Αυτό αποτελεί και την πρόκληση για αυτές τις μηχανές, δηλαδή, να είναι προσβάσιμες από τους περισσότερους προγραμματιστές που επιθυμούν να δημιουργήσουν ανεξάρτητο δικό τους περιεχόμενο αλλά και διαθέτουν αρκετά υψηλή απόδοση ώστε να τις επιλέγουν μικρομεσαίες αλλά και μεγάλες εταιρίες κατασκευής παιχνιδιών (Nicoll, B., Keogh, B. & Springerlink, 2019).

1.2 Ιστορική αναδρομή Unity Game Engine

Η μηχανή Unity δημιουργήθηκε από τρεις προγραμματιστές οι οποίοι χωρίς να διαθέτουν πολλά χρήματα συγκεντρώθηκαν σε ένα υπόγειο και ξεκίνησαν να γράφουν κώδικα δημιουργώντας ένα λογισμικό το οποίο θα χρησιμοποιούταν ευρέως από την βιομηχανία των

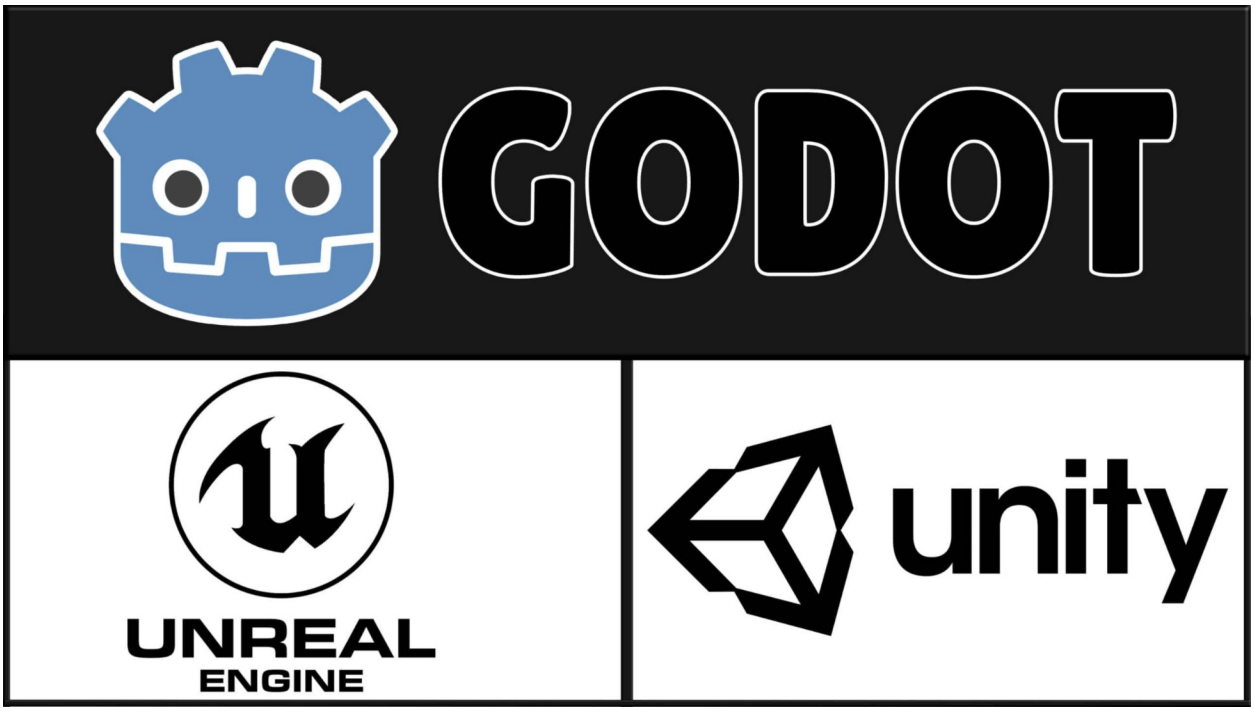
βιντεοπαιχνιδιών. Οι τρεις αυτοί προγραμματιστές ήταν οι David Helgason, Joachim Ante καI Nicholas Francis. Η ομάδα αυτή έπειτα ίδρυσε την εταιρία Over the Edge Entertainment (OTEE) την μετέπειτα Unity Technologies και ξεκίνησε να προσλαμβάνει μηχανικούς λογισμικού. Όταν η μηχανή ήταν πλέον στο δεύτερο στάδιο ελέγχου η OTEE χρησιμοποίησε την μηχανή για να κατασκευάσει ένα βιντεοπαιχνίδι το οποίο θα αναδείκνυε τις δυνατότητες της Unity. Το παιχνίδι αυτό ήταν το *GooBall* και δημιουργήθηκε το 2005 (Nicoll, B., Keogh, B. & Springerlink, 2019). Η δημιουργία της μηχανής ανακοινώθηκε για πρώτη φορά στην Παγκόσμια Συνδιάσκεψη Προγραμματιστών της Apple και από τότε έχει προκαλέσει ραγδαίες αλλαγές στην βιομηχανία των βιντεοπαιχνιδιών και μπορεί να χρησιμοποιηθεί σε πολλές διαφορετικές πλατφόρμες (Windows (PC), Mac, Universal Windows Platform (UWP), Linux Standalone, iOS, Android, ARKit, ARCore, Microsoft HoloLens, Windows Mixed Reality, Magic Leap (Lumin), Oculus, PlayStation VR, PS5,PS4, Xbox One, Xbox X|S, Nintendo Switch, Google Stadia, Linux QNX) (Unity, 2022).

https://unity.com/solutions/multiplatform

2. Εισαγωγή στο Unity

2.1 Επιλέγοντας Game Engine

Το πρώτο στάδιο που κάθε game designer πρέπει να διαβεί, είναι να επιλέξει το κατάλληλο game engine για τη δημιουργία του video game. Από τις μηχανές ανάπτυξης παιχνιδιών, οι πιο δημοφιλείς είναι το Godot, η Unreal (UDK) και φυσικά το Unity3d. Η απόφαση αυτή εξαρτάται κυρίως από το μέγεθος της κοινότητας που το υποστηρίζει, την ποικιλία των γραφικών μοντέλων και scripts τα οποία μπορεί να χρησιμοποιήσει κατά τη διάρκεια του development αλλά και στην προγραμματιστική του εμπειρία. Εμείς επιλέξαμε το Unity3d για την αρκετά μεγάλη κοινότητα του, τα forums αλλά και για την ποικιλία των δωρεάν και μη γραφικών μοντέλων τα οποία ταίριαζαν με το video game που θέλαμε να δημιουργήσαμε, καθώς επίσης την προτίμηση που έχουμε για την γλώσσα scripting και προγραμματισμού η οποία είναι η C#.



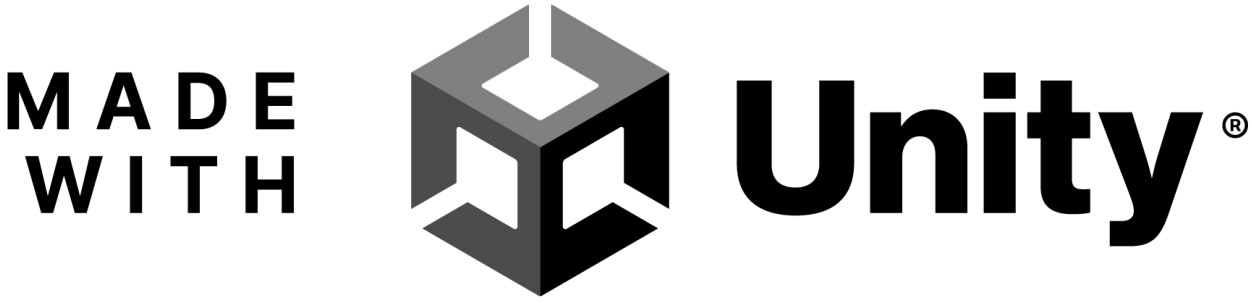
https://gamefromscratch.com/wp-content/uploads/2019/11/GodotVsUnityUnreal4K-scaled.jpg

2.2 Unity Engine

Το Unity κατέχει μεγαλύτερο μερίδιο αγοράς από το Unreal (περίπου 50% έναντι περίπου 15%), επομένως θα πρέπει να υπάρχει μεγαλύτερη ζήτηση για εργασία Unity.
(*Unity: Analyzing the First Game Engine IPO — Naavik*, n.d.)

Το μεγαλύτερο πλεονέκτημα για το Unity είναι ότι είναι ευχάριστο να εισάγεις τον κώδικα. Όλα είναι απλά για κάποιον που είναι γνωστης προγραμματισμού και οι χρόνοι μεταγλώττισης είναι εξαιρετικά γρήγοροι.

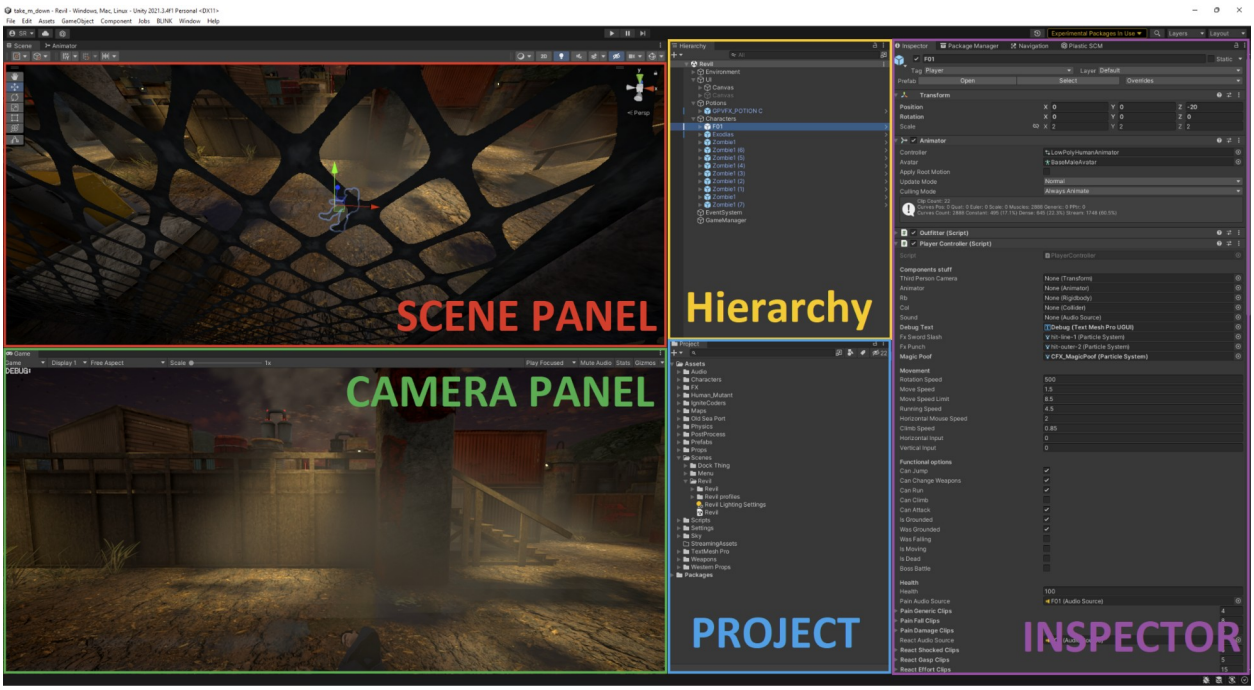
Το Unity Engine χρησιμοποιεί την C# ως κύρια γλώσσα. Αυτή η γλώσσα έχει μια εξαιρετική κοινότητα, χαμηλό επίπεδο εισόδου και ομαλή καμπύλη εκμάθησης. Αλλά ταυτόχρονα, είναι αρκετά αυστηρή και έχει πολλά ισχυρά χαρακτηριστικά για τη σύνταξη σταθερού και αποτελεσματικού κώδικα. Όλα αυτά το καθιστούν ιδανικό για αρχάριους και έμπειρους προγραμματιστές.
Το Unity χρησιμοποιεί μια μικρή προσαρμοσμένη έκδοση του χρόνου εκτέλεσης Mono για την εκτέλεση όλων των σεναρίων σας. Η πιο πρόσφατη έκδοση Unity παρέχει στους προγραμματιστές πρόσβαση σε λειτουργίες και API .NET Standard 2.1 και .NET 4.x. Αυτό σημαίνει ότι πολλές βιβλιοθήκες γραμμένες για κανονικές εφαρμογές .NET ενδέχεται να χρησιμοποιηθούν με το Unity ως τυπικό '.dll' πρόσθετο . Σχεδιάζεται υποστήριξη για πιο σύγχρονες εκδόσεις .NET, γεγονός που θα κάνει την εκδήλωση Unity Engine πιο ενσωματωμένη στο οικοσύστημα .NET.



https://unity3d.com/legal/branding_trademarks

2.2.1 Γραφικό Περιβάλλον Πλατφόρμας

Στο παρακάτω screenshot είναι το Unity Editor, το οποίο αποτελείται από 5 (πέντε) κύρια panels με τα οποία διαχειριζόμαστε εύκολα το project. Το Unity Editor, δίνει τη δυνατότητα στον game designer να οργανώσει τα panels με τη σειρά και την τοποθεσία που επιθυμεί, καθώς επίσης να εξατομικεύσει όλο το Interface, προσθέτοντας ή αφαιρώντας panels και plugins.



2.2.1.1 Scene Panel

Σε αυτό το panel βρίσκεται ο χώρος προσαρμογής του παιχνιδιού, στον οποίο ο χρήστης μπορεί να οργανώσει, τοποθετήσει, στοιχειοθετήσει, μεγεθύνει ή σμικρύνει διάφορα Objects (π.χ. μία σκάλα) καθώς επίσης επεξεργαστεί το περιβάλλον του ίδιου του παιχνιδιού μέσω των κομβίων *Q, W, E, R*.

- Κομβίο “*Q*” - Navigate:

  Επιλέγοντας αυτό το πλήκτρο, γίνεται η πλοήγηση στον χώρο του παιχνιδιού. Κρατώντας το *αριστερό click* μετακινούμε την κάμερα, ενώ πατώντας ταυτόχρονα και το πλήκτρο *Alt* μπορούμε να περιστρέψουμε την κάμερα. Με το *δεξί click*, περιστρέφουμε την κάμερα προς την κατεύθυνση που του ορίζουμε, ενώ πατώντας ταυτόχρονα και το πλήκτρο *Alt* μεγενθύνουμε την κάμερα προς αυτό το σημείο.

- Κομβίο “*W*” - Translate:

  Όταν έχουμε επιλέξει κάποιο Object (πατώντας *αριστερό click*), μπορούμε να το μετακινήσουμε σύμφωνα με τους άξονες X, Y, Z.

- Κομβίο “*E*” - Rotate:

  Μας επιτρέπει να περιστρέψουμε κάποιο Object.

- Κομβίο “*R*” - Scale:

  Μας επιτρέπει τη μεγέθυνση ή σμίκρυνση του Object που έχουμε εστιάσει.

2.2.1.2 Camera Panel

Η αναπαράσταση του παιχνιδιού στην τελική του μορφή, διακρίνεται σε αυτό το panel. Πατώντας τα γραφικά κομβία “*Play*” και “*Pause*”, μπορούμε να ξεκινήσουμε ή να κάνουμε παύση το παιχνίδι στην σκηνή που έχουμε ανοιχτή.

2.2.1.3 Hierarchy

Σε αυτό το παράθυρο, βρίσκονται όλα τα αντικείμενα που είναι τοποθετημένα στην σκηνή του παιχνιδιού. Αρκετές φορές, είναι σημαντικό να τοποθετηθούν τα Objects κάτω ή πάνω από κάποια άλλα, ανάλογα τις λειτουργίες τους καθώς επίσης να μείνουν ανενεργά μέχρι να τα ενεργοποιήσει το ίδιο το παιχνίδι, προγραμματιστικά (π.χ. U.I. ή διάφορα dialogs ή subtitles).

2.2.1.4 Project

Το παράθυρο αυτό περιέχει όλα τα αρχεία που βρίσκονται μέσα στο "root" φάκελο του παιχνιδιού. Assets, Scenes, Fonts, Sounds, Scripts, Prefabs, Licenses μέχρι και plain text αρχεία βρίσκονται στο Project.

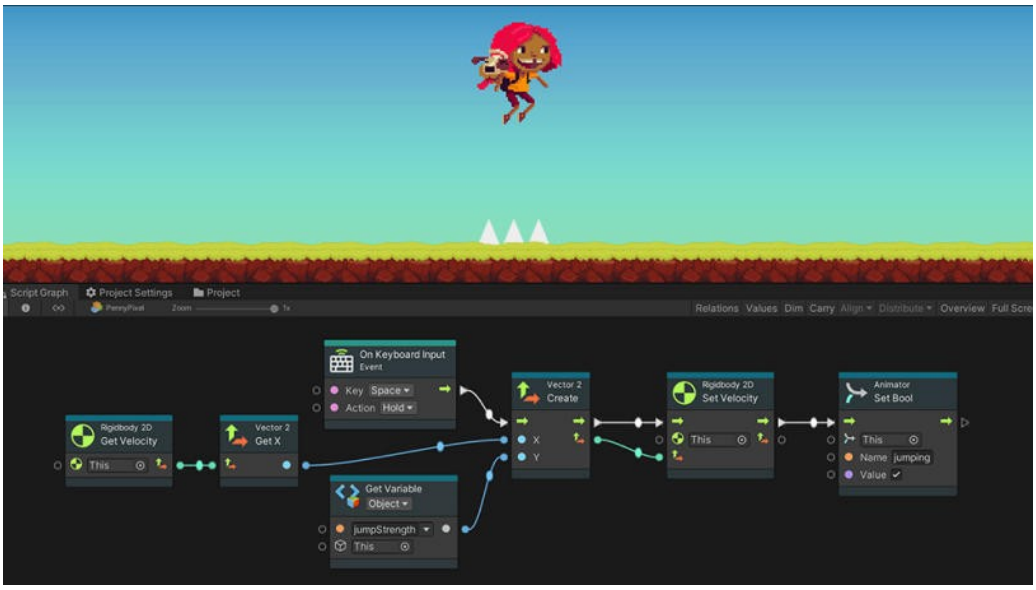
2.2.1.5 Inspector

Στον Inspector διακρίνονται όλα τα χαρακτηριστικά των Objects που βρίσκονται στο παράθυρο Hierarchy. Τα χαρακτηριστικά αυτά μπορεί κάποιος να τους αλλάξει τιμή, να τους μετακινήσει θέση και γενικώς να προσθέσει ή αφαιρέσει λειτουργίες από τα Objects που έχει επιλέξει. Μία αρκετά χρήσιμη λειτουργία, βρίσκεται στο κομβίο "*Lock*" με το οποίο μπορούμε να κλειδώσουμε την εστίαση του Inspector σε συγκεκριμένο Object, ενώ παράλληλα διαχειριζόμαστε άλλα Objects στο Hierarchy, (π.χ. να κάνουμε drag and drop κάποιο Object μέσα σε κάποιο στοιχείο του Inspector που είναι εστιασμένο σε άλλο Object).

2.2.1 Visual Scripting

Το Visual Scripting επιτρέπει την δημιουργία λογικής, στοιχείων σύνδεσης και σεναρίων για παιχνίδια ή εφαρμογές χωρίς την εγγραφή κώδικα, κάνοντας τον συγκεκριμένο τρόπο ιδανικό για καλλιτέχνες, σχεδιαστές και μη προγραμματιστές. Τα σενάρια δημιουργούνται με συνδυασμό κόμβων τα οποία προσφέρουν μια προσομοίωση προγραματιστικής λογικής.

Το Visual Scripting χρησιμοποιεί γραφικά στοιχεία, τα οποία αντιπροσωπεύουν συναρτήσεις, τελεστές ή μεταβλητές. Στη συνέχεια, μπορείτε να συνδέσετε αυτούς τους κόμβους από τις θύρες τους χρησιμοποιώντας άκρες. Αντί να χρειάζεται να γράφετε κώδικα γραμμή προς γραμμή, τα κάνετε όλα οπτικά. Ένα από τα μειονεκτήματα αυτής της δυνατότητας είναι η χειρότερη απόδοση σε σύγκριση με την C# λόγω του χαμηλού επιπέδου ελέγχου στους πόρους του παιχνιδιού , αλλά βελτιώνεται και σχεδιάζεται μια νέα μηχανή εκτέλεσης οπτικών σεναρίων που θα κάνει την απόδοση των οπτικών σεναρίων σχεδόν πανομοιότυπη με τα σενάρια C#.



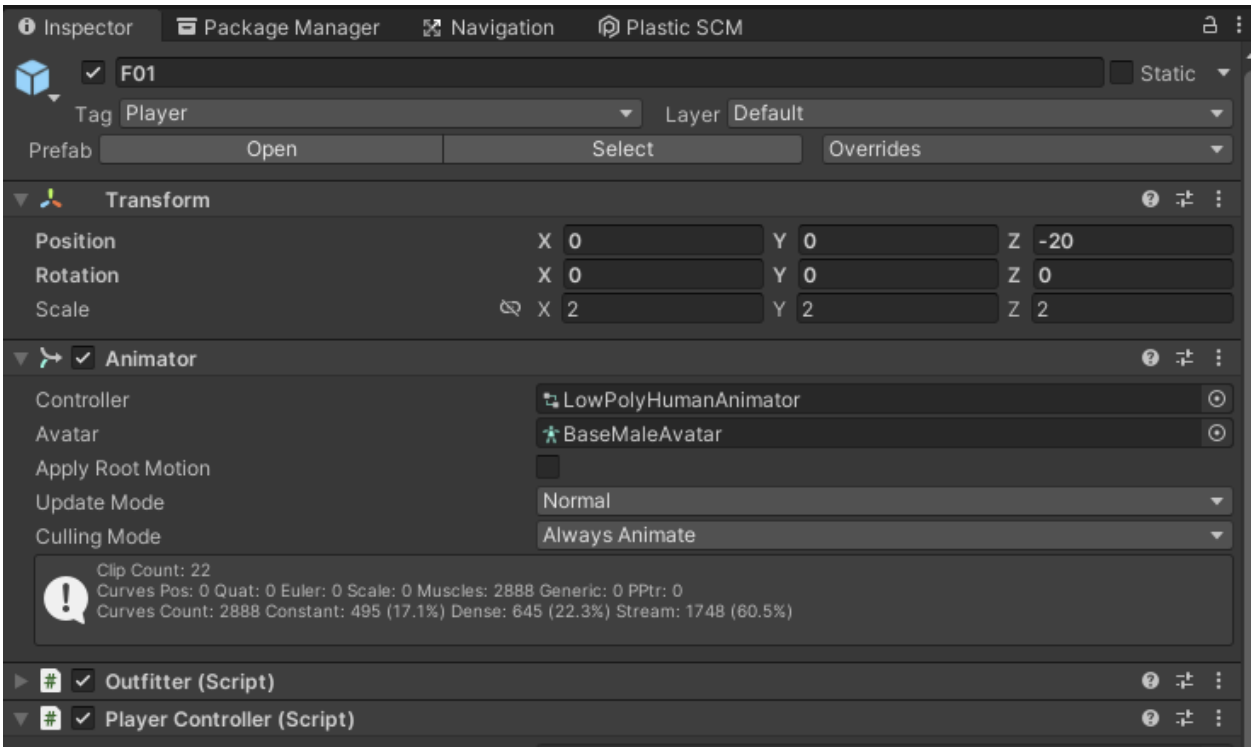
https://unity.com/features/unity-visual-scripting

Μιας και εμείς στοχεύουμε προς το προγραμματιστικό μέρος του Game Design, δεν χρησιμοποιήσαμε την λειτουργία αυτή, επιλέξαμε να γράψουμε κώδικα C#.

2.2.2 GameObject

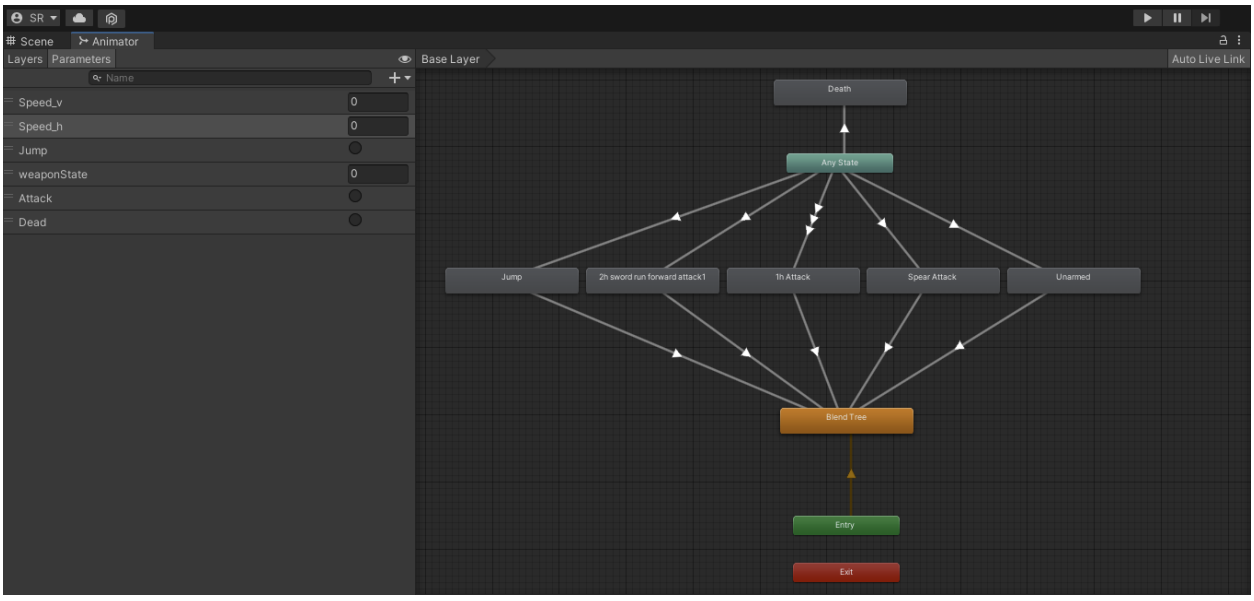
Η κλάση GameObject του Unity χρησιμοποιείται για να αναπαραστήσει οτιδήποτε μπορεί να υπάρχει σε μια σκηνή.Ένα GameObject μπορεί να είναι οτιδήποτε όπως ένα φως η ο ίδιος ο χαρακτήρας του παίκτη,η μπορεί να είναι και σύνολο από αλλά GameObject.Τα GameObjects είναι τα δομικά στοιχεία για σκηνές στο Unity και λειτουργούν ως δοχεία για λειτουργικά στοιχεία που καθορίζουν πώς φαίνεται το GameObject και τι κάνει το GameObject.

Στον προγραμματισμό, η κλάση GameObject παρέχει μια συλλογή μεθόδων που επιτρέπουν να αντιδράσει μαζί με τον κώδικά, συμπεριλαμβανομένου της: εύρεσης, της δημιουργίας συνδέσεων και της αποστολής μηνυμάτων μεταξύ των GameObjects, καθώς και της προσθήκης ή αφαίρεσης στοιχείων που είναι συνδεδεμένα με το GameObject και ορίζοντας τιμές που σχετίζονται με την κατάστασή τους στη σκηνή.



2.2.3 Animations and Animator Controller

Το Animator Controller είναι ένα Unity asset που ελέγχει τη λογική ενός κινούμενου GameObject. Εντός του Animator Controller υπάρχουν καταστάσεις(States) και Sub-State Machines που συνδέονται μεταξύ τους μέσω μεταβάσεων(Transitions). Οι καταστάσεις είναι η αναπαράσταση των Κλιπ κινουμένων σχεδίων(Animations) στο Animator. Οι μεταβάσεις κατευθύνουν τη ροή ενός animation από τη μια κατάσταση στην άλλη. Ενώ οι μεταβάσεις κατευθύνουν τη ροή της λογικής κινούμενων εικόνων, οι Συνθήκες(Conditions) που διέπονται από τις Παραμέτρους(Parameters) ελέγχουν πότε οι καταστάσεις των animations μπορούν να ρέουν στην επόμενη κατάσταση κινούμενης εικόνας. Οι τιμές των παραμέτρων αλλάζουν από συμβάντα σε σενάρια. Για παράδειγμα, ένα χαρακτήρας μπορεί να ελέγχεται από τον χρήστη χρησιμοποιώντας την είσοδο πληκτρολογίου. Αυτές οι τιμές εισόδου τροποποιούν τις τιμές των παραμέτρων και παράγουν ένα animation εικόνα που συνδέεται με αυτές βάσει της συνθήκης που έχει οριστεί στον Animator Controller.



Το Animator Controller προσφέρει μια απεικόνιση στο πως θα εξελιχθούν τα animations, παρόλα αυτά υπάρχει τρόπος να χρησιμοποιήσουμε τα Animations χωρίς την χρήση του

Animator Controller. Μέσω scripting στην C# μπορούμε να καλέσουμε διάφορα functions ώστε να ελέγξουμε τα animations χωρίς να μπλεχτείς με κουτάκια και βελάκια.

```
private void Start() {
    _anim.CrossFade("Walk", 0, 0);
}
```

```
private static readonly int Idle = Animator.StringToHash("Idle");
private static readonly int Walk = Animator.StringToHash("Walk");
private static readonly int Jump = Animator.StringToHash("Jump");
private static readonly int Fall = Animator.StringToHash("Fall");
private static readonly int Land = Animator.StringToHash("Land");
private static readonly int Attack = Animator.StringToHash("Attack");
private static readonly int Crouch = Animator.StringToHash("Crouch");
```

```
var state = _player.Input.x == 0 ? Idle : Walk;
```

Και έτσι μπορούμε να φτιάξουμε ένα παρόμοιο έλεγχο καταστάσεων με Conditions και Transitions μόνο από κώδικα.

```
private int GetState() {
    if (Time.time < _lockedTill) return _currentState;

    // Priorities
    if (_attacking) return LockState(Attack, _attackAnimDuration);
    if (_crouching) return Crouch;
    if (_jumping) return Jump;
    if (_landed) return LockState(Land, _landAnimDuration);

    if (_grounded) return _input.x == 0 ? Idle : Walk;
    return _speed.y > 0 ? Jump : Fall;

    int LockState(int s, float t) {
        _lockedTill = Time.time + t;
        return s;
    }
}
```

2.2.4 PlasticSCM

Το PlasticSCM είναι ένα λογισμικό που εκμεταλλευτήκαμε κατά τη διάρκεια της ανάπτυξης για να μπορέσουμε να δουλεύουμε παράλληλα 2 (δύο) άτομα, τον ίδιο πηγαίο κώδικα. Επιπλέον, μας δίνει τη δυνατότητα του revert που σημαίνει ότι εάν κάποια στιγμή κάνουμε κάποιο λάθος ή σβήσουμε κάποιο αρχείο, μπορούμε ανά πάσα στιγμή να γυρίσουμε πίσω σε αυτό το σημείο. Επίσης μπορούμε να δούμε και την ιστορικότητα των αλλαγών του πηγαίου κώδικα, κάτι το οποίο μας βοήθησε αρκετά στο να κάνουμε track τους μηχανισμούς και τις λειτουργίες τις οποίες ενσωματώνουμε στον πηγαίο κώδικα.

2.3 Επιπρόσθετα προγράμματα

Για τη δημιουργία ενός παιχνιδιού, είναι αρκετά σημαντική η χρήση των προγραμμάτων επεξεργασίας ήχου και εικόνας αλλά και των προγραμμάτων 3D modelling καθώς είναι από τα βασικά στοιχεία για την επιτυχία ενός video game.

Όπως και στις μηχανές ανάπτυξης video game, υπάρχουν αρκετά στην αγορά για την κάλυψη αυτών των αναγκών.

2.3.1 Λογισμικά 3d modeling

https://www.blender.org/about/logo/

Μερικά από τα πιο ευρέως χρησιμοποιούμενα στην κοινότητα είναι:

- Blender - https://www.blender.org/
- Maxon Cinema 4D - https://www.maxon.net/en/cinema-4d
- 3ds max - Autodesk - https://www.autodesk.com/products/3ds-max/overview

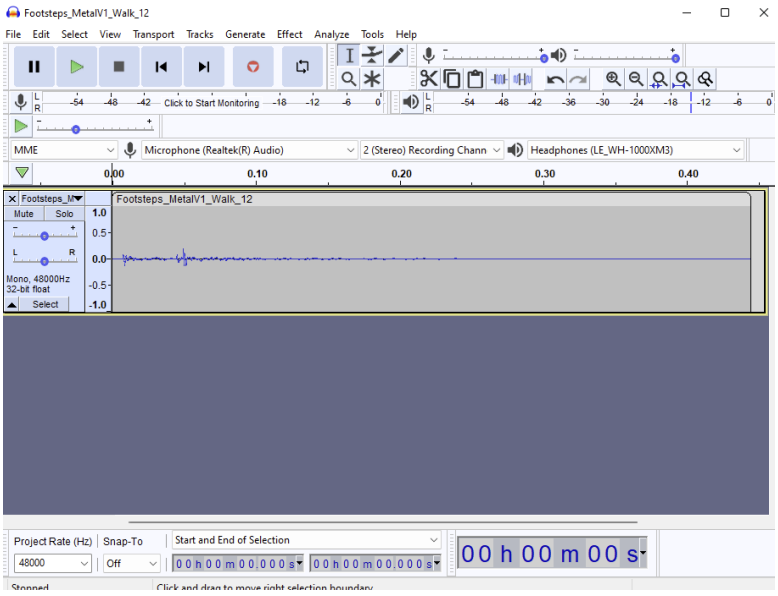
Στην αρχή του development του παιχνιδιού μας, αποφασίσαμε να δημιουργήσουμε κάποιο 3d model για να το συμπεριλάβουμε σε κάποιο level αλλά συμπεράναμε πως για τη δημιουργία του χρειάζεται αρκετά χρόνια εμπειρίας και ταλέντο, παρόλα αυτά, το Blender μας φάνηκε αρκετά εύκολο με σύγκριση με τα υπόλοιπα λογισμικά. Έτσι, καταλήξαμε να χρησιμοποιήσουμε έτοιμα και δωρεάν αλλά και επί πληρωμή assets, από το Unity asset store https://assetstore.unity.com/ και δεν χρειάστηκε η χρήση κάποιου λογισμικού επεξεργασίας 3d modeling.

2.3.2 Λογισμικά επεξεργασίας ήχου

https://www.audacityteam.org/wp-content/themes/wp_audacity/img/logo.png

Για την επεξεργασία του ήχου, επιλέξαμε το Audacity https://www.audacityteam.org/ καθώς είναι δωρεάν και ο κώδικας ανοικτός. Συγκεκριμένα, μας βοήθησε να παραμετροποιήσουμε τις συχνότητες των footsteps του κύριου χαρακτήρα αλλά και των zombies. Η χρήση επίσης του βοήθησε αρκετά στην περικοπή κάποιων σημείων στους ήχους που είχαν μηδενική συχνότητα ώστε να ακούγονται καθαρότερα και χωρίς κάποια καθυστέρηση. Αξιοσημείωτες εναλλακτικές λύσεις είναι και τα:

- FMOD - https://www.fmod.com/
- Steinberg nuendo - https://www.steinberg.net/de/nuendo/game-audio/
- Audiokinetic wwise - https://www.audiokinetic.com/en/products/wwise/



2.3.3 Λογισμικά επεξεργασίας εικόνας

https://www.gimp.org/images/frontpage/wilber-big.png

Μέσα στο πλαίσιο ανάπτυξης του video game, υπήρχε η ανάγκη δημιουργίας στατικών εικόνων όπως για παράδειγμα το ending scene, για το οποίο αξιοποιήσαμε τις δυνατότητες ενός άλλου δωρεάν και ανοικτού κώδικα λογισμικού, το GIMP https://www.gimp.org/, ένα από τα πιο δημοφιλή και user-friendly εργαλεία επεξεργασίας εικόνας και όχι μόνο. Παρόμοιου τύπου λογισμικά είναι:

- Krita - https://krita.org/en/
- Adobe Photoshop - https://www.adobe.com/gr_en/products/photoshop.html
- Microsoft Paint - https://apps.microsoft.com/store/detail/paint/9PCFS5B6T72H

3. Take’ Em Down

3.1 Το παιχνίδι

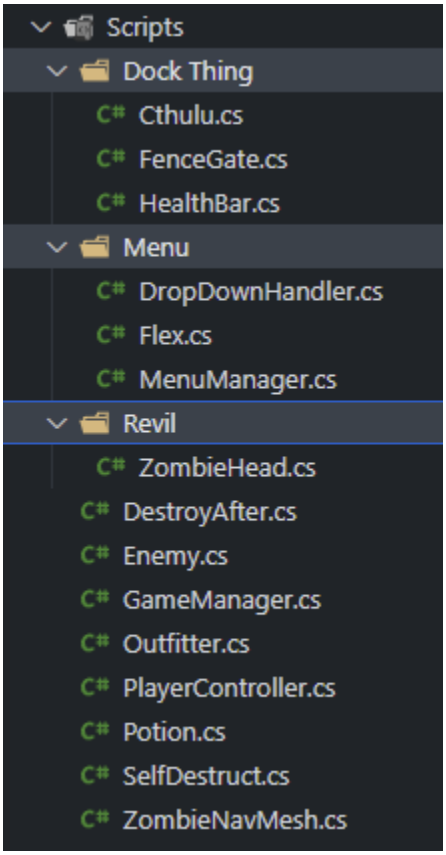
Το game ονομάζεται Take’ Em Down και είναι ένα 3rd person, zombie action hack n’ slash game υλοποιημένο με Unity 2021.3.4f1. Αφορά έναν καπετάνιο, τον Effwan (κύριος χαρακτήρας με καταγωγή την Κορέα), ο οποίος μετά τη συντριβή του αεροπλάνου, βρίσκεται μόνος του ενάντια σε Zombies τα οποία πρέπει να εξουδετερώσει με μπουνιές, σπαθιά ή δόρυ, για να προχωρήσει στην επόμενη πίστα, μέχρι να βρεθεί αντιμέτωπος με τον Azrarg (κύριο boss). Το όνομα του κύριου χαρακτήρα, Effwan, προέρχεται από το Prefab με τίτλο “F01”.

Διαλέξαμε να φτιάξουμε αυτό το είδος παιχνιδιού διότι θεωρήσαμε ότι κατέχει το κατάλληλο επίπεδο δυσκολίας για έναν αρχάριο στο Unity Engine αλλά και γενικώς στο Game Development. Συγκεκριμένα, υποθέσαμε πως η χρήση ενός 3rd Person controller θα μειώσει το χρόνο υλοποίησης ως προς την αλληλεπίδραση των χαρακτήρων με άλλα αντικείμενα. Ως ένα melee-based παιχνίδι, συγκριτικά με άλλες κατηγορίες παιχνιδιών (shooting, racing, sports, κλπ.), παρέχει το μικρότερο βαθμό πολυπλοκότητας και δυσκολίας ως προς την υλοποίησή του. Επιπρόσθετα, κατά τη δημιουργία του παιχνιδιού, παρατηρώντας την πλειοψηφία των δωρεάν assets που παρέχει το Unity Assets Store https://assetstore.unity.com μας έκανε ακόμα πιο σίγουρους για την επιλογή του είδους του game καθώς θα μας επέτρεπε να το τροποποιήσουμε σε όλη τη διάρκεια της υλοποίησης χωρίς να ξεφεύγουμε από τον στόχο μας.

Για το rendering, έχει χρησιμοποιηθεί το URP pipeline καθώς είναι το μόνο που άφηνε περιθώριο συμβατότητας με τα assets που έχουμε βρει και χρησιμοποιήσει σε όλο το παιχνίδι.

3.2 Βασική δομή πηγαίου κώδικα

Προτού περιγραφούν τα ίδια τα scripts, αξίζει να τονιστεί η πρακτική διαχώρισης των scripts που ακολουθήθηκε, σε δύο (2) κατηγορίες. Τα κοινά, που χρησιμοποιούνται ή έπονται να χρησιμοποιηθούν στο μέλλον από πολλά αντικείμενα ή πίστες, και τα map-specific scripts που αφορούν λειτουργίες συγκεκριμένες για κάποια πίστα ή σκηνή. Σε περίπτωση που θα θελήσουμε να καλυτερεύσουμε και να εμπλουτίσουμε το παιχνίδι στο μέλλον, η τακτική αυτή θα μας βοηθήσει αρκετά ως προς την κατανόηση και την διαχείριση πολλών scripts.

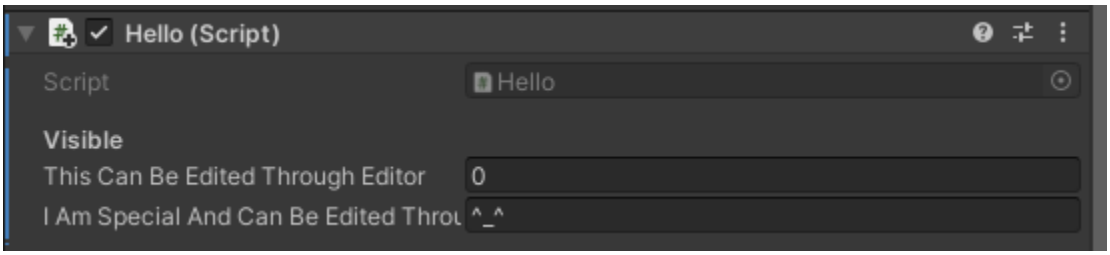


Το κάθε script στο Unity Engine, από προεπιλογή, αποτελείται από ένα βασικό σκελετό, μία κλάση με το όνομα του sciprt που κάνει extend το MonoBehavior (Unity Technologies, 2022) και περιέχει δύο βασικές συναρτήσεις, την Start() και την Update():

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

public class Hello : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

Μέσα στην κλάση αυτή, μπορεί κάποιος να δημιουργήσει δικές του συναρτήσεις ή να χρησιμοποιήσει αυτές που παρέχει η ίδια η μηχανή για συγκεκριμένες λειτουργίες όπως για παράδειγμα την OnTriggerEnter() ή την OnCollisionEnter(). Όπως σε κάθε αντικειμενοστραφή γλώσσα προγραμματισμού έτσι και στη C#, μπορούμε να ορίσουμε χαρακτηριστικά/μεταβλητές της κλάσης για να περιγράψουμε το αντικείμενο. Επίσης, έχει σημασία ο τρόπος που εκτίθεται μία μεταβλητή (public ή private) γιατί έχει αντίκτυπο στο Unity Editor, οι public και οι serialized μεταβλητές μπορούν να αλλάξουν μέσα από το Unity Editor ενώ οι private όχι:

```
public class Hello : MonoBehaviour
{
    [Header("Visible")]
    public int ThisCanBeEditedThroughEditor = default;
    [SerializeField] private string IAmSpecialAndCanBeEditedThroughEditor = "^_^";

    [Header("Hidden")]
    private string ICannotBeEditedThroughEditor = "xtonousou";
```



Η συνάρτηση Start() καλείται μόνο μία φορά κατά την εκκίνηση του script όταν δηλαδή καλείται για πρώτη φορά.

```
119     public void Start() {
120         Vector3 localScale = transform.localScale;
121         thirdPersonCamera = GameObject.Find("ThirdPersonCamera").GetComponent<Transform>();
122         animator = GetComponentInChildren<Animator>();
123         rb = GetComponent<Rigidbody>();
124         col = GetComponent<Collider>();
125         gameManager = GameObject.FindGameObjectWithTag("GameController").GetComponent<GameManager>();
126 
127         // add reactions
128         reactions.Add(reactShockedClips);
129         reactions.Add(reactGaspClips);
130         reactions.Add(reactLaughClips);
131         reactions.Add(reactSighClips);
132 
133         // update distance to ground
134         distToGround = col.bounds.extents.y;
135     }
```

Η συνάρτηση Update() καλείται συνέχεια για κάθε frame per second (εικόνα το δευτερόλεπτο) και εκεί βρίσκεται το μεγαλύτερο ποσοστό του κώδικα καθώς εκεί αλληλεπιδρούν σε πραγματικό χρόνο τα αντικείμενα μεταξύ τους, όσο το παιχνίδι είναι σε δράση.

```
160     public void Update() {
161         if (gameManager.gameIsPaused || isDead || PlayerPrefs.GetInt("wonGame") == 1) return;
162 
163         // update distance to ground
164         distToGround = col.bounds.extents.y+0.3f;
165 
166         // handle weapon states and equipped weapons
167         HandleWeapons();
168 
169         // handle player's attacks
170         PlayerAttack();
171 
172         // handle player's movement
173         PlayerMove();
174 
175         // handle player's footsteps
176         HandleFootsteps();
177 
178         // handle player's reactions
179         HandleReactions();
180 
181         // Debug info on screen
182         GUIUpdate();
183     }
```

Επίσης υπάρχει και η συνάρτηση FixedUpdate(), παρόμοια με την Update(), αλλά καλείται κάθε 0.02 δευτερόλεπτα (50 κλήσεις το δευτερόλεπτο) γιατί χρησιμοποιεί το σύστημα φυσικής του Unity Engine. Κυρίως, χρησιμοποιείται για να επεξεργαστούμε ή παραμετροποιήσουμε τα χαρακτηριστικά των RigidBody components τα οποία έχουμε ορίσει στα αντικείμενα (π.χ. φυσικές δυνάμεις όπως η βαρύτητα).

```
137     public void FixedUpdate() {
138         if (gameManager.gameIsPaused || isDead) return;
139 
140         layerMask = 1 << 26;
141         layerMask = ~layerMask;
142         CheckLife();
143         CheckGround();
144 
145         // take fall damage based on distance to ground
146         if (!wasFalling && isFalling) startOfFall = transform.position.y;
147         if (!wasGrounded && isGrounded) {
148             float fallDistance = startOfFall - transform.position.y;
149             if (fallDistance >= minimumFall) PlayerTakeDamage("fall", fallDistance);
150             if (fallDistance >= minimumJump) StartCoroutine(PlaySound("jump/land"));
151         }
152 
153         wasGrounded = isGrounded;
154         wasFalling = isFalling;
155 
156         // It has to be FixedUpdate, because it applies force to the rigidbody constantly.
157         rb.AddForce(Physics.gravity * gravityScale , ForceMode.Acceleration);
158     }
```

Προηγουμένως, αναφέρθηκαν οι συναρτήσεις OnTriggerEnter() και OnCollisionEnter() οι οποίες καλούνται αυτόματα όταν “κάτι” ισχύει, όπου η λέξη “κάτι” περιγράφεται από το όνομα της ίδιας της συνάρτησης (trigger, collision, κλπ.). Όπως και σε άλλες γλώσσες προγραμματισμού υπάρχον οι όροι events ή listeners που περιγράφουν τη λειτουργία αυτή.

Στο παιχνίδι, εκμεταλλευτήκαμε τη λειτουργία τους για να προγραμματίσουμε τον μηχανισμό “ladder”, δηλαδή το τι συμβαίνει στον κύριο χαρακτήρα όταν ο ίδιος ο χαρακτήρας βρίσκεται στο πλαίσιο της σκάλας, ή όταν βρίσκεται κοντά στο χέρι/πλοκάμι του Azrarg (boss).

```
192     public void OnCollisionEnter(Collision col) {
193         if (gameManager.gameIsPaused || isDead) return;
194 
195         switch (col.gameObject.tag) {
196             case "Ladder":
197                 canJump = false;
198                 canChangeWeapons = false;
199                 canAttack = false;
200                 canRun = false;
201                 canClimb = !canClimb;
202                 break;
203             case "Enemy/Tentacle":
204                 PlayerTakeDamage("tentacle", 10f);
205                 break;
206             default:
207                 break;
208         }
209     }
```

Ακολουθώντας την λογική των listeners, μπορεί κάποιος να υλοποιήσει το input του παιχνιδιού, όπως για παράδειγμα το τι συμβαίνει όταν ο χρήστης πατήσει το πλήκτρο “W”. Αυτή η μέθοδος έχει πλέον καταργηθεί αλλά εξακολουθεί να υπάρχει για λόγους συμβατότητας.

3.3 Περιβάλλον

Το περιβάλλον του παιχνιδιού αφορά οτιδήποτε αντικείμενα (terrain, walls, lights, grass, trees) τα οποία συμβολίζουν την κάθε πίστα ξεχωριστά. Στο παιχνίδι αυτό, δεν χρησιμοποιήθηκε κάποιο ιδιαίτερο terrain ή γενικώς η λειτουργικότητα του terrain σαν αντικείμενο, αλλά στη θέση του χρησιμοποιήθηκαν αντικείμενα τύπου Plane για να μειώσουμε την πολυπλοκότητα καθώς ξεκινήσαμε ως αρχάριοι. Ένας ακόμα λόγος για τον οποίο διαλέξαμε τα απλά αντικείμενα Plane, είναι γιατί το παιχνίδι δεν περιέχει πίστες με μεγάλη έκταση και είναι static. Επίσης, τα αντικείμενα τύπου Plane έχουν αρκετά καλύτερο collision detection και γενικώς καλύτερο rendering.

Το terrain object βοηθάει σε περιπτώσεις που υπάρχει δυναμική βλάστηση (vegetation) και που χρήζει την ανάγκη τυχαίας δημιουργίας πιστών (dynamic map generation). Επιπρόσθετα, όταν η πίστα δεν πρέπει να είναι flat και θα πρέπει να έχει ακαμψίες (π.χ. βουνό), η χρήση του αντικειμένου τύπου Terrain είναι καλύτερη επιλογή από τη χρήση των αντικειμένων Plane.

Σε κάθε πίστα, υπάρχουν περιοχές στις οποίες όταν βρίσκεται ο παίκτης ακούγεται και ο ανάλογος ήχος. Αυτο έχει επιτευχθεί με τη χρήση διαφόρων Plane objects και AudioSource objects. Για παράδειγμα, όταν ο παίκτης βρίσκεται πάνω σε μία λακούβα με νερό και υπάρχει γρασίδι τότε θα ακούγεται ο ήχος του νερού και ότι κάτι μετακινείται στο γρασίδι. Με τον ίδιο τρόπο έχουμε υλοποιήσει και τα Footsteps του κάθε "ζωντανού" object.

3.3.1 Πίστες/Maps

Στην συγκεκριμένη υλοποίηση, υπάρχουν δύο (2) maps στα οποία έχουν αναπτυχθεί μοναδικές λειτουργίες και ξεχωρίζουν μεταξύ τους. Όταν το παιχνίδι ξεκινά, ο χρήστης βλέπει το μενού στο οποίο μπορεί να διαλέξει συγκεκριμένη πίστα για να παίξει. Όταν ο χρήστης ολοκληρώσει την πίστα την οποία έχει επιλέξει, εξουδετερώνοντας τους εχθρούς και ακολουθώντας την πορεία της κάθε πίστας, μεταβαίνει στην επόμενη (progression). Αυτό είναι σημαντικό για να υπάρχει εξέλιξη στην ιστορία ενός παιχνιδιού αλλά και να κρατάει τον χρήστη

σε περιέργεια για τις επόμενες πίστες αλλά και γενικώς για την ιστορία του παιχνιδιού και πως περιμένει να τελειώσει (game ending).

Το Take' Em Down, έχει ως σκοπό την επιστροφή του Effwan στο σπίτι του αφού πρώτα σκοτώσει όλα τα Zombies και τον αρχηγό στην τελευταία πίστα. Οι δύο (2) πίστες μπορούν να χαρακτηριστούν ως bare minimum για να υπάρχει ένα σχετικό progression σε ένα παιχνίδι.

Κατά τη διάρκεια του development αλλά και στη μετέπειτα φάση του παιχνιδιού (αφού έχει ολοκληρωθεί) βρήκαμε αρκετά σημαντική τη χρήση των debug messages για να μας βοηθήσει να εξιχνιάσουμε bugs σε επίπεδο κώδικα. Συγκεκριμένα, έχουμε δημιουργήσει ένα επιπλέον UI που είναι σαν ένα overlay/hud το οποίο μας αποτυπώνει τις boolean μεταβλητές του χαρακτήρα μας (π.χ. Εάν επιτίθεται, εάν βρίσκεται στον αέρα, κλπ.). Το debug UI δεν θα πρέπει να υπάρχει σε public release έκδοση, είναι για λόγους testing.

UI. Debug hud

Η διαχείριση των πιστών, γίνεται από το script GameManager.cs και το MenuManager.cs. Το πως θα μεταβεί ο χρήστης στην επόμενη πίστα, ή στην πίστα που έχει

επιλέξει ο χρήστης από το μενού και πως κλείνει η ιστορία μιας πίστας και γίνεται η μετάβαση στην επόμενη, τα διαχειρίζεται το GameManager.cs.

Αξίζει να τονιστεί, πως η μετάβαση στην επόμενη πίστα γίνεται με βάση τον αριθμό των ζωντανών zombies, δηλαδή ο χρήστης θα πρέπει πρώτα να σκοτώσει όλα τα zombies για να περάσει στο επόμενο map ή ιστορία.

GameManager.cs

Assets > Scripts > GameManager.cs

```
28
29      public void CheckScene() {
30          if (PlayerPrefs.GetInt("wonGame") == 1) return;
31
32          switch (SceneManager.GetActiveScene().name) {
33              case "Revil":
34                  RevilProgression();
35                  break;
36              case "Dock Thing":
37                  DockThingProgression();
38                  DockThingBoss();
39                  break;
40          }
41      }
42
43      public void RevilProgression() {
44          if (GameObject.FindGameObjectsWithTag("Enemy/Zombie").Length == 0) {
45              LoadScene("Dock Thing");
46          }
47      }
48
49      public void DockThingProgression() {
50          if (GameObject.FindGameObjectsWithTag("Enemy/Zombie").Length == 0) {
51              FenceGate fenceGate2 = GameObject.Find("fence_gate 2").GetComponent<FenceGate>();
52              if (!fenceGate2.open) {
53                  fenceGate2.OpenFenceGate();
54                  if (Mathf.Abs(fenceGate2.GetZRotation()) < 0.005f) fenceGate2.open = true;
55              }
56
57              FenceGate fenceGate3 = GameObject.Find("fence_gate 3").GetComponent<FenceGate>();
58              if (!fenceGate3.open) {
59                  fenceGate3.OpenFenceGate();
60                  if (Mathf.Abs(fenceGate3.GetZRotation()) < 0.005f) fenceGate3.open = true;
61              }
62          }
63      }
64
65      public void DockThingBoss() {
66          if (player.bossBattle) {
67              BossHud.SetActive(true);
68
69              if (GameObject.Find("Human_Mutant").GetComponent<Cthulu>().isDead) {
70                  BossHud.SetActive(false);
71                  PlayerPrefs.SetInt("wonGame", 1);
72                  StartCoroutine(WaitForWin());
73              }
74          }
75      }
```

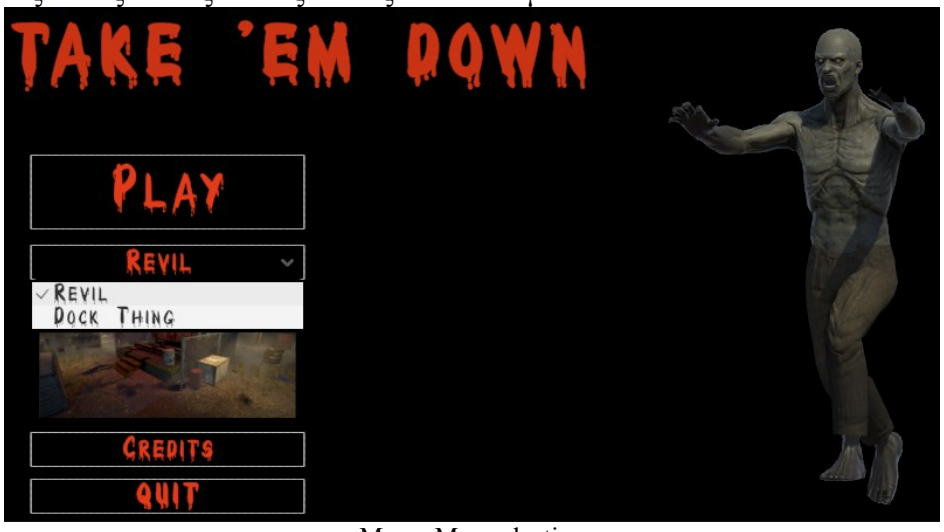
Progression. Game Manager script

```
33      public void Update() {}
34
35      public void ExitGame() {
36          #if UNITY_EDITOR
37              UnityEditor.EditorApplication.isPlaying = false;
38          #else
39              Application.Quit();
40          #endif
41      }
42
43      public void LoadScene() {
44          SceneManager.LoadScene(selectedScene);
45          Time.timeScale = 1;
46      }
47
48      public void LoadCredits() {
49          creditsMenuCanvas.SetActive(true);
50          mainMenuCanvas.SetActive(false);
51      }
52
53      public void ExitCredits() {
54          mainMenuCanvas.SetActive(true);
55          creditsMenuCanvas.SetActive(false);
56      }
```

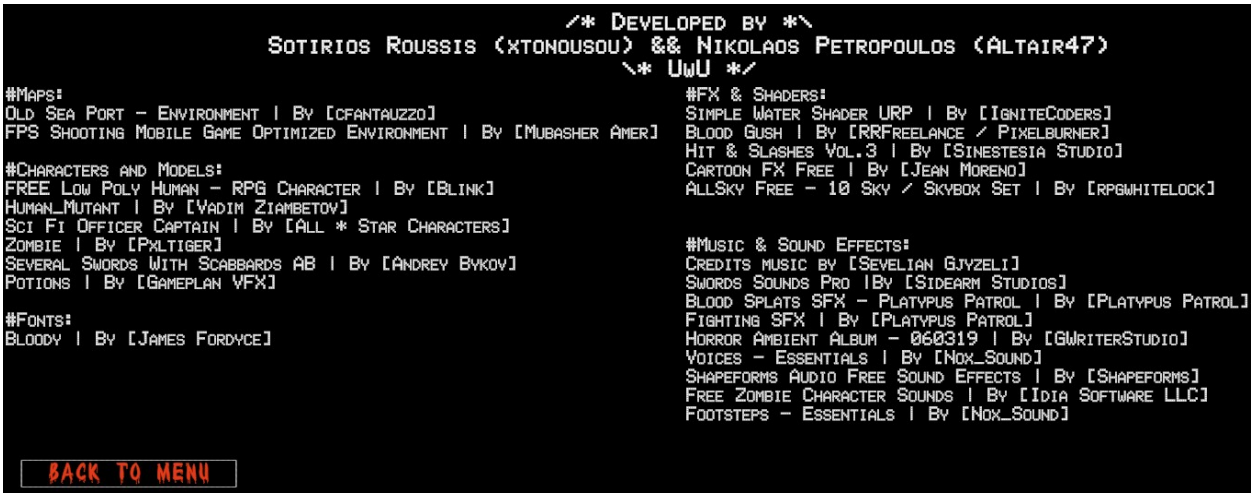
Scenes. Menu Manager script

3.3.1.1 Το μενού “Menu”

Το μενού είναι από τα πιο βασικά στοιχεία σε ένα παιχνίδι καθώς είναι από τα πρώτα πράγματα με το οποίο αλληλεπιδρά ο χρήστης. Μπορεί να μην το έχουμε στο μυαλό μας ως “πίστα”, αλλά στη λογική την οποία ακολουθήσαμε, το μενού είναι ακόμα μία πίστα. Εάν ακολουθήσει κάποιος αυτή την τεχνική θα του είναι πιο εύκολο να συνδέσει, προγραμματιστικά, το μενού με τις άλλες πίστες και τις πίστες πίσω στο μενού.



Menu. Map selection



Menu. Credits showcase

3.3.1.2 Η πίστα “Revil”

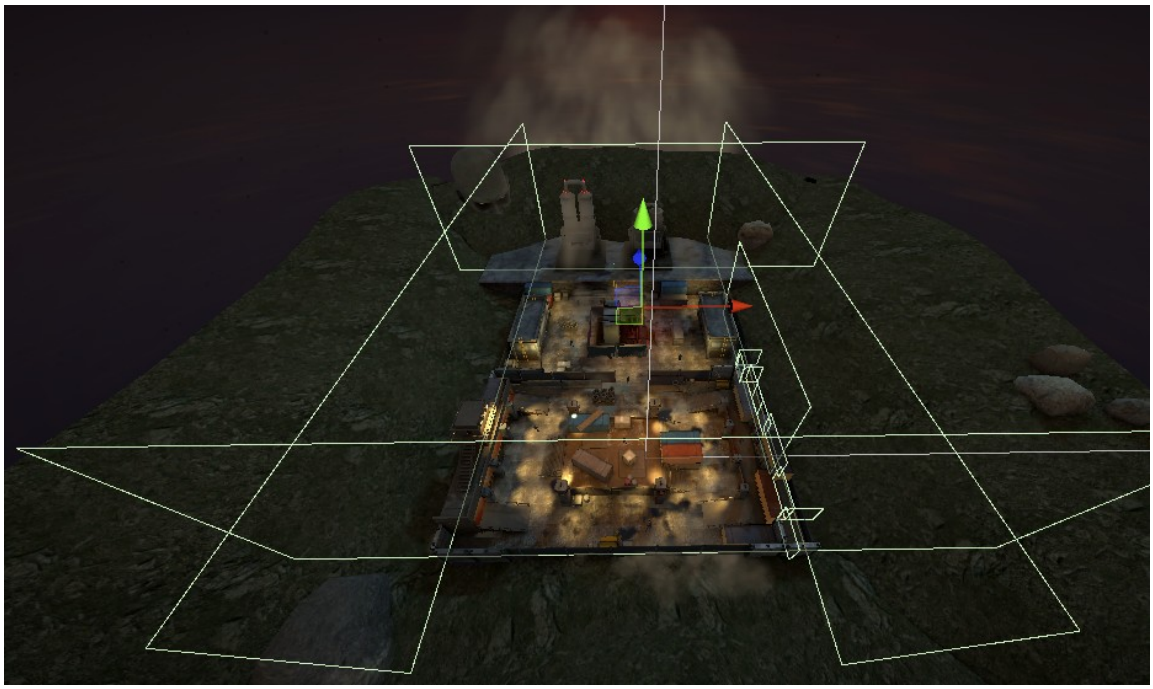

Revil. Map overview

Την πίστα αυτή την βρήκαμε σχεδόν έτοιμη στο Unity assets store. Για να καλύψουμε τις ανάγκες του παιχνιδιού προσθέσαμε παραπάνω βλάστηση, τοποθετήσαμε τα zombies, ένα potion που δίνει στον χαρακτήρα health και κάναμε interactive τα αντικείμενα που βρίσκονται σε αυτή (π.χ. ladders).

Έχουν τοποθετηθεί επιπλέον Plane objects για να κάνουμε λειτουργικό το movement του χαρακτήρα αλλά και των zombies (navmesh agent).

Revil. Helpful plane for movement

Λόγω κάποιων bugs που βρήκαμε στην τοποθέτηση των objects του map, αναγκαστήκαμε να δημιουργήσουμε επιπλέον αντικείμενα τύπου Plane χωρίς το Renderer ενεργοποιημένο, για να περιορίσουμε τον χαρακτήρα από το να βγαίνει εκτός από τα όρια που είναι φτιαγμένο το map (OutOfMapProtection).



Revil. Out of map protection mechanism

Μπορεί να αναρωτηθεί κάποιος, τί ακριβώς είναι το zombie κεφάλι, πάνω αριστερά της πίστας. Για την διερεύνηση των triggers (μηχανισμός του Unity για callbacks εάν μία συνθήκη είναι αληθής) τοποθετήσαμε το κεφάλι του prefab από τα zombies και όταν και μόνο όταν ο χαρακτήρας βρίσκεται στο πλαίσιο που έχουμε ορίσει, τότε το κεφάλι μετακινείται απότομα προς την μεριά του παίκτη για να τον τρομάξει (με ήχο).

Revil. ZombieHead jump scare

Στην ίδια πίστα έχει τοποθετηθεί ένα μικρό prop, με τη λειτουργία του potion ώστε όταν ο χαρακτήρας βρίσκεται σε αυτό, σπάει (εμφανίζοντας πράσινα particles), και του δίνει +10 HP στο health του. Αυτό βοηθάει αρκετά τον παίκτη καθώς έχει αρκετά zombies η πίστα και είναι σχετικά δύσκολο να την ολοκληρώσει.

Revil. Green health potion

```
C# Potion.cs ×
Assets > Scripts > C# Potion.cs > Potion > Start()
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Potion : MonoBehaviour {
6      private PlayerController player;
7      public float potionHealthBoost = 13.37f;
8      public bool potionSuccess = false;
9
10     public void Start() {
11         player = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
12     }
13
14     public void Update() {}
15
16     private void OnTriggerEnter(Collider col) {
17         if (potionSuccess) return;
18
19         player.health += potionHealthBoost;
20         potionSuccess = true;
21         StartCoroutine(player.PlaySound("react/reflexion"));
22
23         transform.GetChild(0).gameObject.SetActive(false);
24         transform.GetChild(1).gameObject.SetActive(false);
25         transform.GetChild(2).gameObject.SetActive(false);
26         transform.GetChild(3).gameObject.SetActive(true);
27         Destroy(gameObject, 5);
28     }
29 }
30
```

Potion. Green health potion script

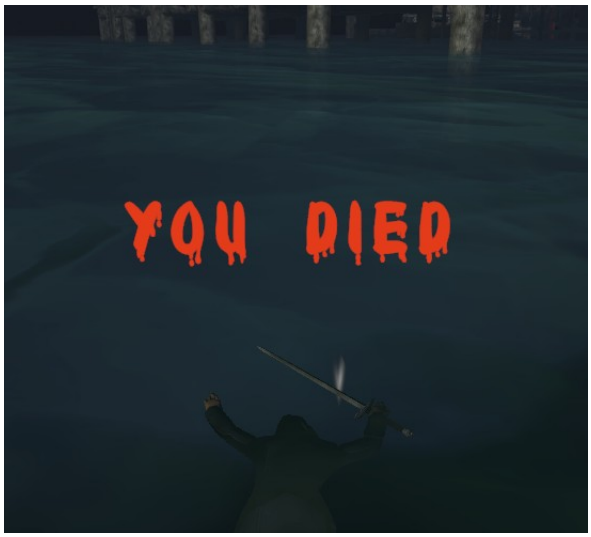
Ένα ακόμα σημαντικό μέρος της πίστας αυτής, είναι ο μηχανισμός των ladders που έχει αναπτυχθεί, ο οποίος δίνει τη δυνατότητα στον παίκτη να ανεβαίνει σκάλες και να πηγαίνει πάνω σε άλλα αντικείμενα.

Revil. Ladder support

3.3.1.3 Η πίστα “Dock Thing”

Dock Thing. Map overview

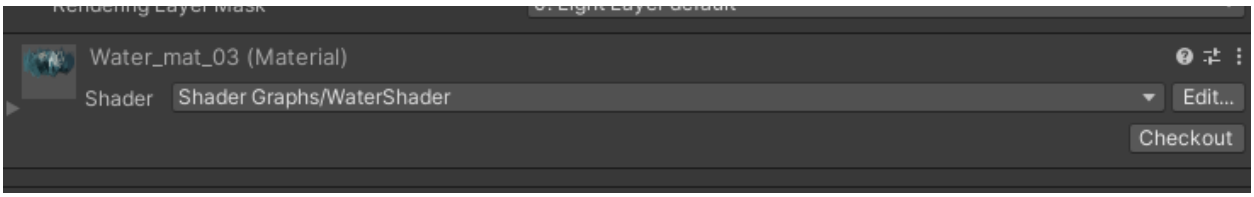
Τη δεύτερη και τελευταία πίστα του παιχνιδιού, βρήκαμε επίσης σχεδόν έτοιμη στο Unity assets store. Γενικώς, δεν έχουν γίνει πολλές αλλαγές συγκριτικά με την “Revil”. Στην πίστα αυτή, μπορεί ο παίκτης να πατάει πάνω στις ξύλινες πλατφόρμες αλλά και να μπει στο κάστρο. Όταν πέσει στο νερό, ο παίκτης χάνει και ξεκινάει το progress από την αρχή της πίστας.



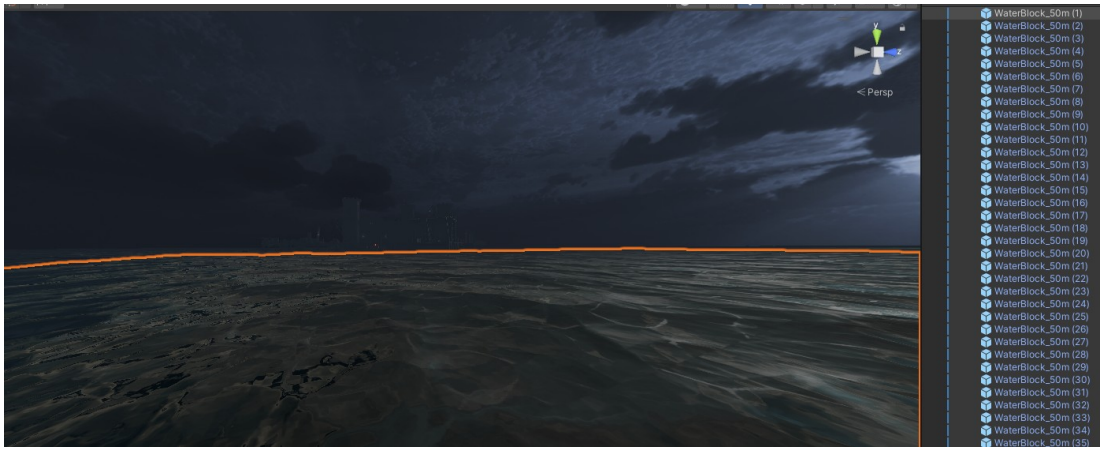
Dock Thing. Water instant death

Τα zombies έχουν τοποθετηθεί στατικά σε συγκεκριμένα σημεία της πίστας τα οποία αποφεύγουν τα όρια της ξύλινης πλατφόρμας και δεν πέφτουν στο νερό.

Για την υλοποίηση και εμφάνιση του νερού, έχουν χρησιμοποιηθεί πολλά blocks/squares στα οποία τους έχουμε ορίσει κάποιο shader.

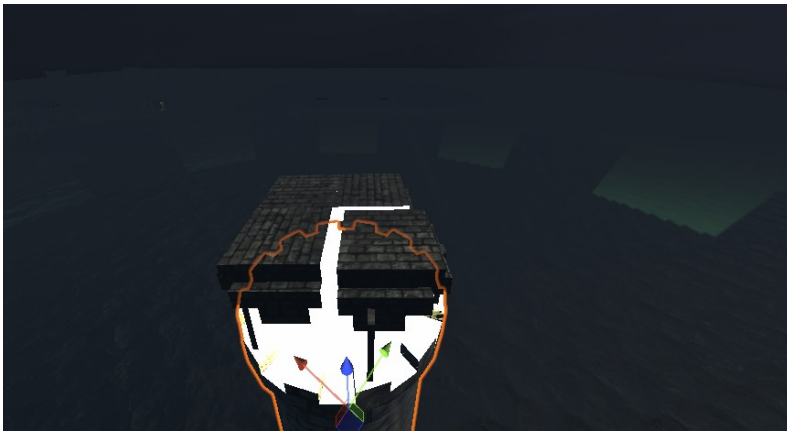


Dock Thing. Water Shader graph



Dock Thing. Water Blocks of shaders

Στον φάρο έχουμε τοποθετήσει φως το οποίο περιστρέφεται για να δίνει την εμφάνιση ενός φάρου, το οποίο φως δεν περνάει μέσα από επιφάνειες.



Dock Thing. Lighthouse overview

Για το progression στην πίστα αυτή, έχουμε προσθέσει έναν μηχανισμό στην κύρια πύλη που χωρίζει το map, η οποία ανοίγει όταν και μόνο όταν ο παίκτης σκοτώσει όλα τα zombies έτσι ώστε να προχωρήσει στο boss (Azrarg). Για το συγκεκριμένο, έχουμε χρησιμοποιήσει μία συνθήκη που ελέγχει τον αριθμό των ενεργών Zombies prefabs αναζητώντας τα “Tags”.

Dock Thing. Fence gate open to boss

FenceGate.cs

Assets > Scripts > Dock Thing > FenceGate.cs > ...

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

public class FenceGate : MonoBehaviour {
    [SerializeField] public bool open = false;
    private float speed = 0.33f;

    public void OpenFenceGate() {
        Quaternion target = Quaternion.Euler(-90, 0, 0);
        transform.rotation = Quaternion.Slerp(transform.rotation, target, speed * Time.deltaTime);
    }

    public float GetZRotation() {
        return transform.rotation.z;
    }
}
```

Dock Thing. Fence gate script

Όταν ο παίκτης περάσει την πύλη, εμφανίζεται το health bar του Azrarg που σημαίνει πως ο παίκτης πρέπει να σκοτώσει το boss.

Dock Thing. Azrarg health bar

Στη μέση περίπου της πίστας, έχει τοποθετηθεί ένα μικρό campfire.

Dock Thing. Campfire

3.4 Player

3.4.1 Input

Το input σύστημα που ακολουθεί το παιχνίδι είναι το legacy. Στο Unity3D από την έκδοση 2019.1, υπάρχει καινούριο σύστημα για το Input (Unity Technologies, 2019) το οποίο παρέχει αρκετά παραπάνω δυνατότητες και είναι πιο εύκολο στη διαχείριση. Διαλέξαμε να ακολουθήσουμε το Legacy System καθώς, ως αρχάριοι, σχεδόν όλοι οι οδηγοί και τα tutorials μας καθοδηγούσαν σε αυτό. Παρόλα αυτά, το Legacy System θα εξακολουθεί να υπάρχει στο μέλλον για να μπορέσουν τα υπάρχοντα παιχνίδια να μεταβιβάσουν τον κώδικά τους στο καινούριο σύστημα.

```
329         if (Input.GetKeyDown(KeyCode.LeftControl)) {
330             transform.localScale -= scaleChange;
331             Instantiate(magicPoof,transform.position,transform.rotation);
332         }
333         if (Input.GetKeyUp(KeyCode.LeftControl)) {
334             transform.localScale += scaleChange;
335             Instantiate(magicPoof,transform.position,transform.rotation);
336         }
```

Input. Legacy system on PlayerMove

Οι δυνατότητες που δίνει το παιχνίδι στον κύριο χαρακτήρα, τον Effwan, είναι να κινείται, να τρέχει, να πηδάει, να σκύβει, να επιτίθεται, να αλλάζει όπλο (μπουνιές, σπαθιά, δόρυ) και να αλλάζει εμφάνιση στα σπαθιά (skins). Με το ποντίκι, μπορεί ο παίκτης να γυρίζει την κάμερα 360 μοίρες.

- Κίνηση με πάτημα πλήκτρων WASD
- Τρέξιμο με παρατεταμένο πάτημα πλήκτρου LEFT_SHIFT ή RIGHT_SHIFT
- Πήδημα με πάτημα πλήκτρου SPACE
- Σκύψιμο με πάτημα πλήκτρου LEFT_CONTROL ή RIGHT_CONTROL
- Επίθεση με πάτημα αριστερού click
- Αλλαγή όπλου σε μπουνιές με το πάτημα πλήκτρου NUM_0
- Αλλαγή όπλου σε σπαθί (που χρησιμοποιείται με το ένα χέρι) με το πάτημα πλήκτρου NUM_1

- Αλλαγή όπλου σε σπαθί (που χρησιμοποιείται με τα δύο χέρια) με το πάτημα πλήκτρου NUM_2
- Αλλαγή όπλου σε δόρυ με το πάτημα πλήκτρου NUM_3
- Αλλαγή εμφάνισης σπαθιού με το WHEEL του ποντικιού
- Αλλαγή κάμερας 360 μοιρών με την κίνηση του ποντικιού, πάνω, κάτω, αριστερά και δεξιά

3.4.2 Controller

```
308     private void PlayerMove() {
309         if (isDead) return;
310 
311         horizontalInput = Input.GetAxis("Horizontal");
312         verticalInput = Input.GetAxis("Vertical");
313 
314         if (Mathf.Abs(horizontalInput) + Mathf.Abs(verticalInput) != 0) {
315             isMoving = true;
316         } else {
317             isMoving = false;
318         }
319 
320         animator.SetFloat("Speed_h", horizontalInput);
321         animator.SetFloat("Speed_v", verticalInput);
322 
323         // rotate object using mouse
324         transform.Rotate(Vector3.up, horizontalMouseSpeed * Input.GetAxis("Mouse X"));
325 
326         thirdPersonCamera.transform.Rotate(Vector3.left, horizontalMouseSpeed * Input.GetAxis("Mouse Y"));
327         // NETI 360 no cap
328 
329         if (Input.GetKeyDown(KeyCode.LeftControl)) {
330             transform.localScale -= scaleChange;
331             Instantiate(magicPoof,transform.position,transform.rotation);
332         }
333         if (Input.GetKeyUp(KeyCode.LeftControl)) {
334             transform.localScale += scaleChange;
335             Instantiate(magicPoof,transform.position,transform.rotation);
336         }
337 
338         float currentSpeed = moveSpeed;
339         if (Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.RightShift)) {
340             currentSpeed = runningSpeed;
341             canRun = false;
342         } else {
343             currentSpeed = moveSpeed;
344             canRun = true;
345         }
346 
347         // move player horizontally
348         rb.AddForce(transform.right * horizontalInput * currentSpeed, ForceMode.VelocityChange);
349         // add vertical force to player to move him forward o backwards
350         rb.AddForce(transform.forward * verticalInput * currentSpeed, ForceMode.VelocityChange);
351 
352         HandleJump();
353         HandleSpeed();
354     }
```

Στο παραπάνω Function PlayerMove() χειριζόμαστε την κατευθυνση την οποία ο παίκτης θέλει να κινηθεί παίρνοντας την είσοδο από το πληκτρολόγιο(WASD) και να κουνήσει τον χαρακτήρα στον άξονα X-Z και παίρνοντας την είσοδο από το ποντίκι για να περιστρέψουμε την κάμερα 360 μοίρες. Κρατώντας πατημένο το Shift η ταχύτητα του χαρακτήρα ανεβαίνει δηλαδή τρέχει πιο γρήγορα.

Επίσεις σε αυτό το function κρατάμε την μεταβλητή isMoving όπου αθροίζουμε της απόλυτες τιμές των inputs για να ξέρουμε ποτέ ο χρήστης σταματάει να δίνει εντολή κίνησης, την οποία θα την χρησιμοποιήσουμε για τους ήχους των βημάτων και πόσο πρέπει να παιχτούν

```
420     private void HandleFootsteps() {
421         if (!isGrounded) return;
422         if (!isMoving) return;
423 
424         footstepTimer -= Time.deltaTime;
425         if (footstepTimer <= 0) {
426             StartCoroutine(PlaySound("footsteps"));
427 
428             if (isMoving) {
429                 if (canRun) footstepTimer = baseStepSpeed;
430                 else footstepTimer = baseStepSpeed * sprintStepMultiplier;
431             }
432         }
433     }
```

καθώς και για το HandleSpeed() στο τέλος του function το οποίο δημιουργήθηκε για να ελέγχει την ταχύτητα του χαρακτήρα όταν σταματάει να κινείται. Λόγω του τρόπου υλοποίησης της

κίνησης με δυνάμεις παρότι ότι είναι πιο ρεαλιστικό έχει ως αποτέλεσμα να κάνει τον χαρακτήρα να γλιστράει όταν προσπαθεί να ακινητοποιηθεί μετά από κίνηση. Με αυτό το function μπορούμε να μειώσουμε την ταχύτητα του χαρακτήρα ομαλά και γρήγορα.

```
409     private void HandleSpeed() {
410         Vector3 flatVelocity = new Vector3(rb.velocity.x, 0, rb.velocity.z);
411         if (flatVelocity.magnitude > moveSpeedLimit) {
412             Vector3 limitedVelocity = flatVelocity.normalized * moveSpeedLimit;
413             rb.velocity = new Vector3(limitedVelocity.x, rb.velocity.y, limitedVelocity.z);
414         }
415 
416         // stop sliding when walk or run and then idle
417         if (!isMoving) rb.velocity = new Vector3(0, rb.velocity.y, 0);
418     }
```

Τέλος έχουμε το HandleJump() το οποίο ελέγχει για άλμα και προστατεύει από κατάχρηση του άλματος (άπειρα άλματα). Περιμένει input το κομβίο (space) και ο χαρακτήρας να είναι isGrounded

```
442     private void HandleJump() {
443         if (!isGrounded || canClimb) return;
444         if (Input.GetKeyDown(KeyCode.Space) && canJump) StartCoroutine(Jump());
445     }
```

για να ξεκινήσει το Coroutine Jump() και να προσθέσει μια δύναμη στον χαρακτήρα από κάτω προς τα πάνω και να τον κάνει να πηδήξει. (Χρησιμοποιούμε το Coroutine γιατί μας δίνει καλύτερη απόδοση και την δυνατότητα να μην κλειδώσουμε το κύριο thread ώστε να μπορούμε να κάνουμε wait ένα timer.)

```
480     public IEnumerator Jump() {
481         canJump = false;
482 
483         jumpTimer -= Time.deltaTime;
484         if (jumpTimer <= 0) {
485             StartCoroutine(PlaySound("jump/start"));
486             if (canJump) jumpTimer = baseJumpSpeed;
487         }
488 
489         animator.SetTrigger("Jump");
490         rb.AddForce(Vector3.up * jumpAmount, ForceMode.Impulse);
491         yield return new WaitForSeconds(0.5f);
492 
493         jumpTimer -= Time.deltaTime;
494         if (jumpTimer <= 0) {
495             StartCoroutine(PlaySound("jump/land"));
496             if (canJump) jumpTimer = baseJumpSpeed;
497         }
498 
499         canJump = true;
500     }
```

Μετά να τρέξει ένα timer για να προλάβει ο χαρακτήρας να περάσει το isGrounded()

Το οποίο ελέγξετε από μια ακτίνα(Raycast) όπου ξεκινάει από το κέντρο του χαρακτήρα προς το έδαφος και μετράμε την απόσταση για να δούμε αν ακουμπάει όντως στο έδαφος.

```
249     private void CheckGround() {
250         isGrounded = Physics.Raycast(transform.position + Vector3.up, -Vector3.up, distToGround);
251     }
```

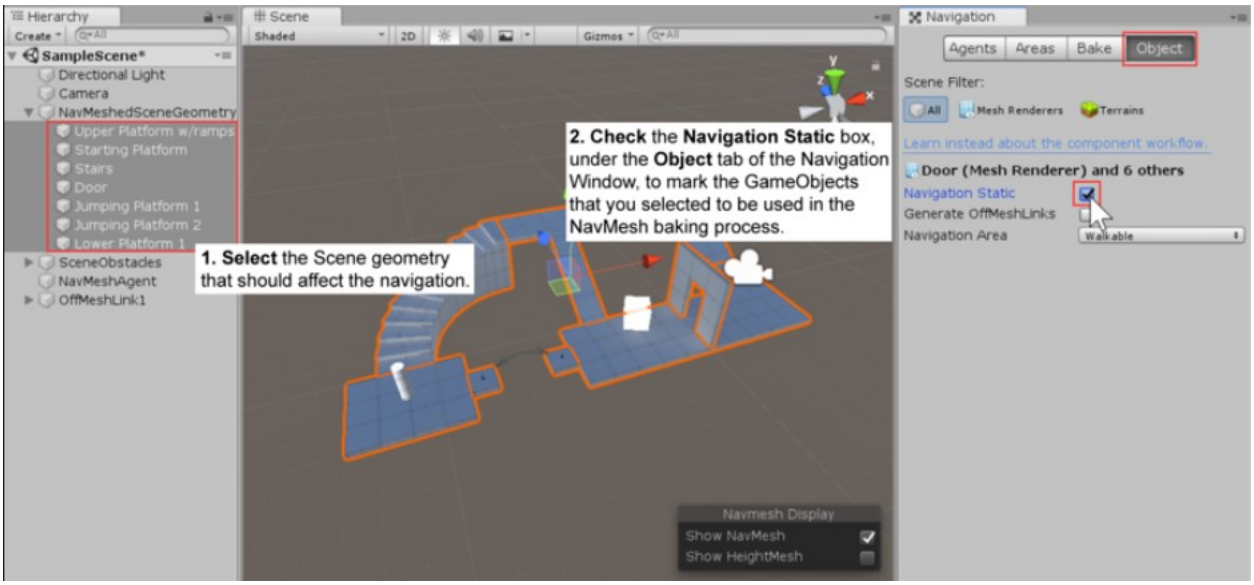
3.5 Εχθροί

Στο παιχνίδι η συμπεριφορά των εχθρών καθορίζεται από κάποια scrpits και πάλι, καθώς και ένα πλέγμα πλοήγησης(navigation mesh ή NavMesh)ώστε να ορίζει που μπορούν οι εχθροί να κινηθούν ανάλογα με την λογική που τους έχουμε δώσει.

3.5.1 NavMesh A.I.

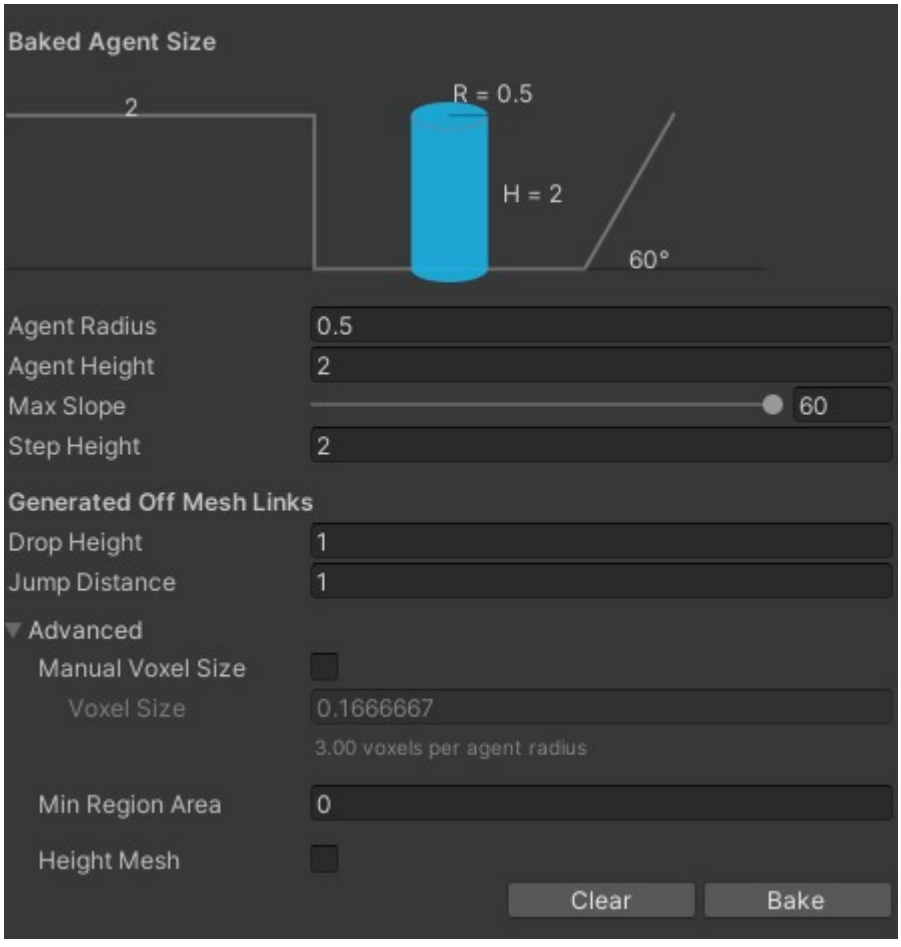
Το NavMesh είναι ένα καθορισμένο πλέγμα(mesh), το οποίο καθορίζει περιοχές με δυνατότητα πλοήγησης στο περιβάλλον, συμπεριλαμβανομένων περιοχών όπου μπορούν να περπατήσουν οι χαρακτήρες, καθώς και να διαβούν εμπόδια. Αυτό είναι χρήσιμο για σενάρια που ενσωματώνουν εύρεση μονοπατιών και πλοήγηση ελεγχόμενη από AI. Το στοιχείο NavMesh Agent βοηθά τους χαρακτήρες να αποφεύγουν ο ένας τον άλλον, να μετακινούνται στη σκηνή προς έναν κοινό στόχο ή οποιοδήποτε άλλο είδος σεναρίου που περιλαμβάνει χωρική συλλογιστική ή εντοπισμό μονοπατιών.

Το NavMesh αυτό δημιουργείται και τροποποιείται πάνω στο Unity Editor όπου μπορείς να διαλέξεις τα αντικειμενα που χρειάζεται και να εφαρμόσεις το NavMesh :



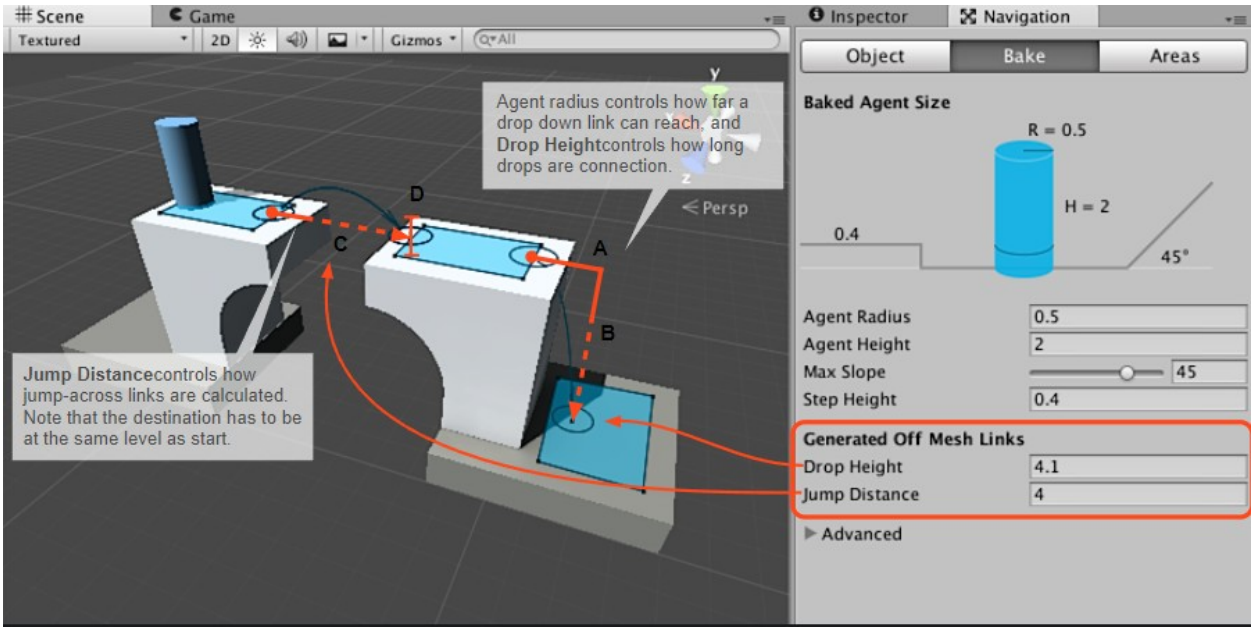
https://docs.unity3d.com/Manual

Στο NavMesh βάζουμε τους Agents οι οποίοι είναι οι χαρακτήρες που θα το χρησιμοποιήσουν.



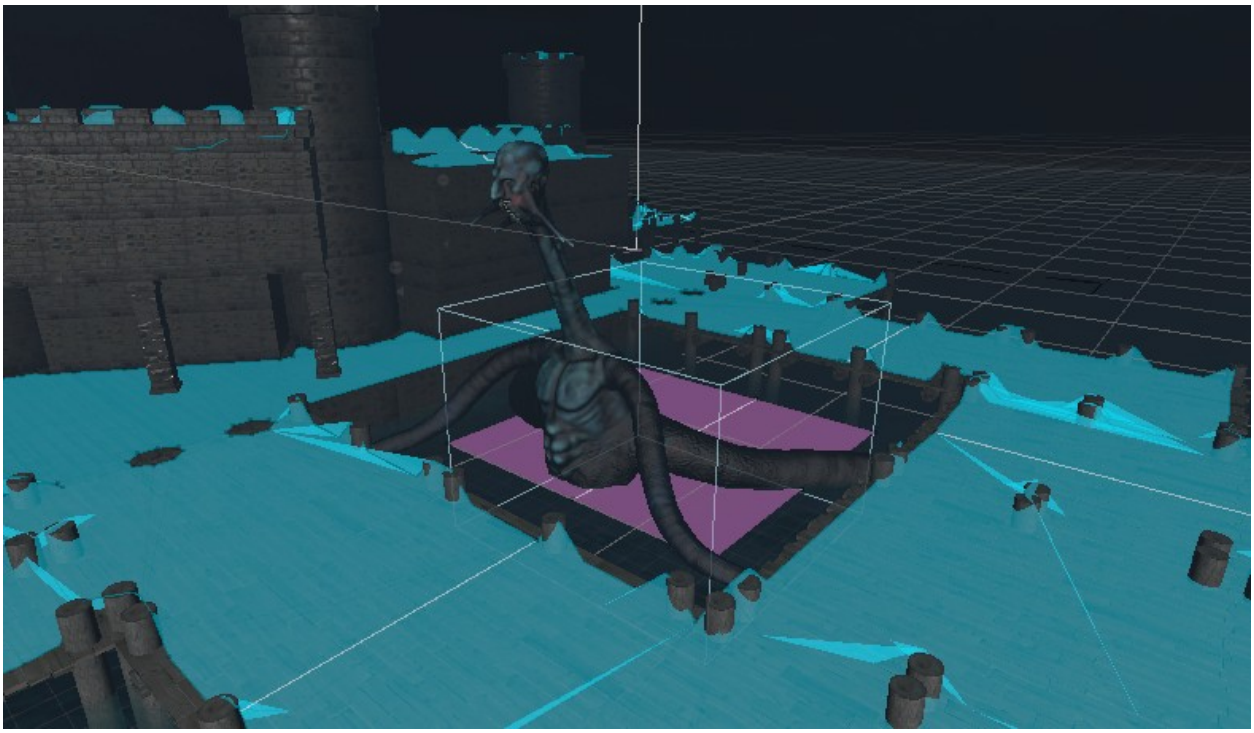
https://docs.unity3d.com/Manual

Πάνω σε αυτό μπορούμε να αλλάξουμε τις διαστάσεις του Agent και την μέγιστη γωνιά την οποία μπορεί να ανέβει καθώς, το μέγεθος του βήματος καθώς και την απόσταση την οποία μπορεί ο χαρακτήρας να ανέβει (να κάνει άλμα) η κατέβει από υψηλή γεωμετρία



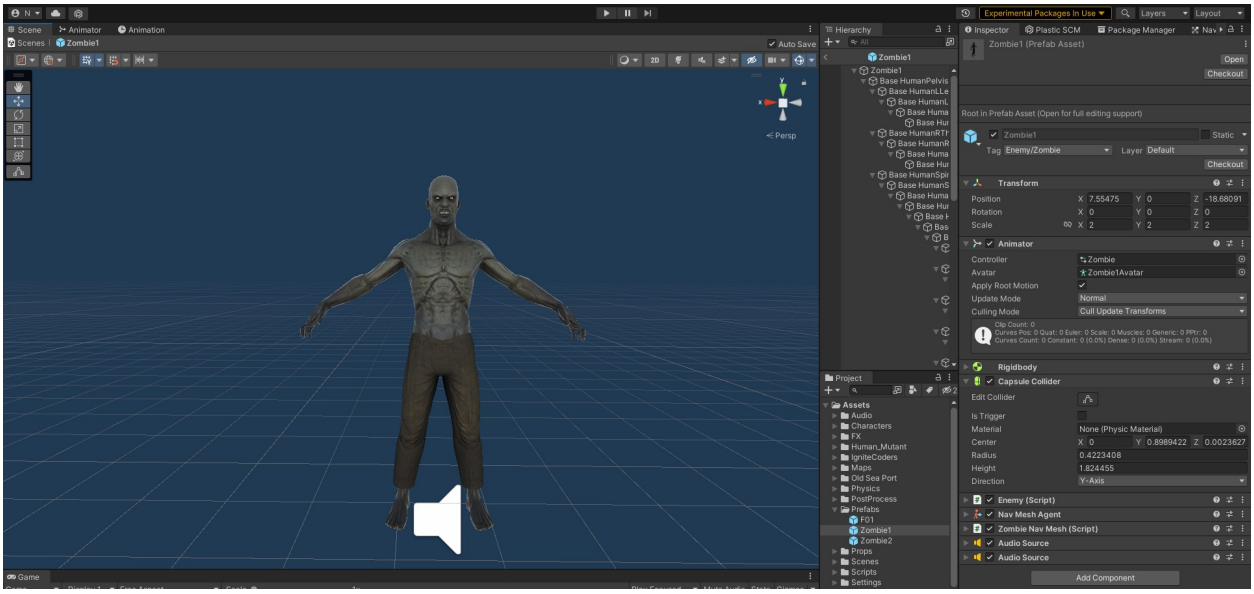
https://docs.unity3d.com/Manual

Είναι δυνατόν επίσεις να φτιάξουμε και διαφορετικά NavMeshes για διαφορετικούς χαρακτήρες όπως για παράδειγμα το Final Boss επειδή είναι στο νερό και χρειάζεται να έχει διαφορετικό NavMesh:



3.5.2 Zombies

Ο κυρίως εχθρός στο Take' Em Down είναι τα Zombies τα οποία ελέγχονται από τα scripts Enemy.cs και το ZombieNavMesh.cs.



Με το ZombieNavMesh.cs χρησιμοποιούμε τον Agent από το NavMesh για να κάνουμε το Zombie να μπορεί και να βρίσκει τον παίκτη να τον ακολουθήσει με μια συγκεκριμένη επιτάχυνση ώστε να του επιτεθούν.

```
public class ZombieNavMesh : MonoBehaviour
{
    // Start is called before the first frame update
    public Enemy zombie;
    [SerializeField] private Transform moveToPositionTransform;
    [SerializeField] private Transform playerPosition;
    private NavMeshAgent navMeshAgent;
    void Start()
    {
        zombie = GetComponent<Enemy>();
        playerPosition = GameObject.FindGameObjectWithTag("Player").
        GetComponent<Transform>();
        navMeshAgent = GetComponent<NavMeshAgent>();
    }

    // Update is called once per frame
    void Update()
    {
            navMeshAgent.destination = playerPosition.position;
            //rotate toward player
            //transform.LookAt(playerPosition);
            navMeshAgent.acceleration = 10f;
    }
}
```

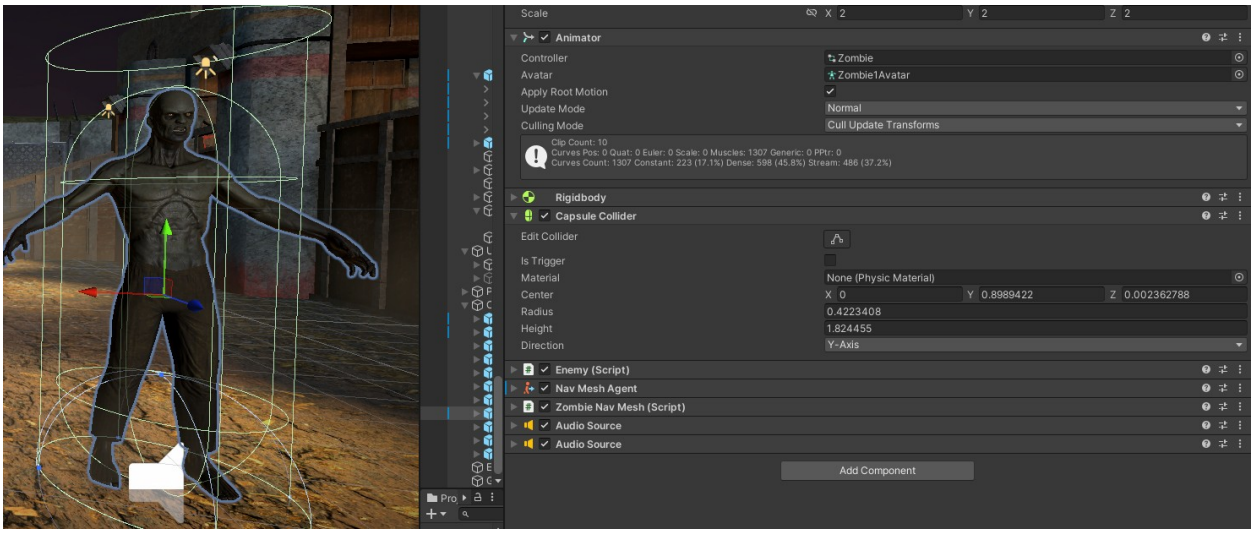
Στο Enemy.cs αρχικά αρχικοποιούμε μερικές μεταβλητές σχετικά με το Zombie, χρησιμοποιούμε 2 μεταβλητές και βγάζουμε μια τυχαία τιμή μεταξύ αυτών ώστε να διαφοροποιήσουμε τα Zombies μεταξύ τους με διαφορετικά χαρακτηριστικά, έτσι κάθε zombie ενώ είναι το ίδιο prefab και έχει το ίδιο script πάνω του, έχει διαφορετικούς πόντους ζωής και διαφορετική ζημιά στην επίθεση τους.

```
26      [SerializeField] private float zHealthRangeMin = 150.0f;
27      [SerializeField] private float zHealthRangeMax = 400.0f;
28      [SerializeField] private float zDamageRangeMin = 5.0f;
29      [SerializeField] private float zDamageRangeMax = 10.0f;
30      [SerializeField] public bool zCanBeDamaged = false;
31      [SerializeField] public bool zCanDamage = false;
32      [SerializeField] public bool zRageMode = false;
33      [SerializeField] public bool zVictory = false;
```

```
35      public void Start() {
36          audioSource = GetComponent<AudioSource>();
37          rb = GetComponent<Rigidbody>();
38          col = GetComponent<Collider>();
39          animator = GetComponent<Animator>();
40          agent = GetComponent<UnityEngine.AI.NavMeshAgent>();
41          playerPosition = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
42          player = GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerController>();
43
44          updateZombieHealth();
45      }
```

```
95      private void updateZombieHealth() => zHealth = Random.Range(zHealthRangeMin, zHealthRangeMax);
```

Επίσης στο Enemy.cs κάνουμε όλους τους ελέγχους σχετικά με την ζωή του Zombie καθώς και collision detection (έλεγχος σύγκρουσης) με τον παίκτη για να αφαιρέσουμε πόντους ζωής. Το Zombie έχει ένα Capsule Collider το οποίο το χρησιμοποιούμε για να ανιχνεύσουμε αν βρίσκεται σε κατάλληλη απόσταση από τον παίκτη ώστε να μπορέσει να κάνει επίθεση



όταν αυτό συγκρουστεί με τον Collider του παίκτη μέσω του OnCollisionEnter τότε όσο είναι εκεί μέσα το zCanDamage θα είναι αληθές

```
84      private void OnCollisionEnter(Collision col) {
85          if (!player.isDead && col.gameObject.tag == "Player") {
86              zCanDamage = true;
87              StartCoroutine(damagePlayer());
88          }
89      }
```

και τότε μπορεί να ξεκινήσει το Coroutine damagePlayer()

```
171     private IEnumerator damagePlayer() {
172         while (zCanDamage) {
173             animator.SetTrigger("Attack");
174             StartCoroutine(playSound("attack", loop: false));
175             yield return new WaitForSeconds(0.8f);
176 
177             if (!zCanDamage) {
178                 Debug.Log("Player escaped before attack was finished, Dodged.");
179                 yield break;
180             }
181             StartCoroutine(player.PlaySound("zombie/hit"));
182             player.health -= Random.Range(zDamageRangeMin, zDamageRangeMax);
183         }
184     }
```

στο οποίο ξεκινάμε το animation για επίθεση και περιμένουμε 0.8 δευτερόλεπτα ώστε το animation να φτάσει στο κατάλληλο καρέ που είναι οράτο ότι χτύπησε τον παίκτη η επίθεση, εκεί ξανά ελέγχει το zCanDamage, το οποίο γίνεται ψευδές σταματήσουν να συγκρούονται τα collider στο OnTriggerExit

```
80      private void OnTriggerExit(Collider col) {
81          if (col.gameObject.tag == "Hitbox") zCanBeDamaged = false;
82      }
```

σε περίπτωση που και παίκτης φύγει από το την απόσταση επιθέσεις πριν ολοκληρωθούν αυτά τα 0.8 δευτερόλεπτα τότε εκτυπώνουμε στο Debug Log ότι απέφυγε την επίθεση και το damagePlayer() τελειώνει πρόωρα. Αν ο παίκτης δεν το αποφύγει τότε θα παιχτεί ο ήχος χτυπήματος και θα αφαιρεθεί από τον παίκτη ένα τυχαίο ποσό πόντων ζωής μεταξύ zDamageRangeMin και zDamageRangeMax.

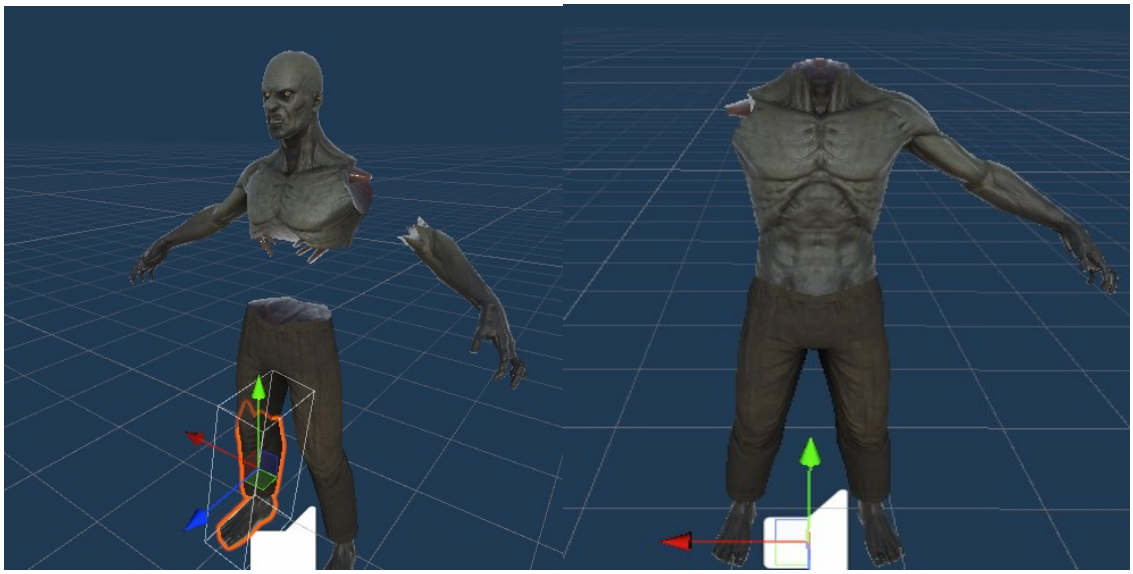
Παρόμοια ο Παίκτης έχει ένα Box Collider μπροστά του όπου είναι Trigger και με τα OnTriggerEnter/OnTriggerStay/OnTriggerExit functions το Zombie καταλαβαίνει όταν βρίσκεται μέσα σε σε αυτό.

```
60      private void OnTriggerEnter(Collider col) {
61          if (!player.isDead && col.gameObject.tag == "Hitbox") zCanBeDamaged = true;
62      }
63
64      private void OnTriggerStay(Collider col) {
65          if (!player.isDead && zCanBeDamaged && Input.GetMouseButtonDown(0)) {
66              animator.SetBool("Pain", true);
67              Instantiate(bloodspray, transform.GetChild(0).position, bloodspray.transform.
                rotation);
68
69              // take zombie body parts off
70              dismember(Random.Range(0, 16));
71              zHealth -= player.playerDamage;
72
73              animator.SetBool("Pain", false);
74
75              rb.AddForce(transform.forward * -10 * 5 * Time.deltaTime, ForceMode.Force);
76          }
77      }
78
79      private void OnTriggerExit(Collider col) {
80          if (col.gameObject.tag == "Hitbox") zCanBeDamaged = false;
81      }
82
83      private void OnCollisionEnter(Collision col) {
84          if (!player.isDead && col.gameObject.tag == "Player") {
85              zCanDamage = true;
86              StartCoroutine(damagePlayer());
87          }
88      }
89
90      private void OnCollisionExit(Collision col) {
91          if (col.gameObject.tag == "Player") zCanDamage = false;
92      }
```

Όσο το zombie βρίσκεται μέσα στο Triggerbox του player τότε αν πατηθεί στο παιχνίδι το αριστερό click του ποντικιού τότε ενεργοποιούνται τα animations για τον "πόνο". Δημιουργούμε επίσεις τα εφέ αιματος και σπρώχνουμε το zombie προς τα πίσω για να έχει την αίσθηση του ότι χτυπήθηκε από κάτι. Με κάθε χτηπημα τρέχει και το function dismember το οποίο έχει μια τυχαία πιθανότητα να κόψη κάποιο μέλος τους σώματος από το zombie, άμα κοπεί το κεφάλι του η πόντοι ζωής του zombie κόβονται στην μέση, γνωστό και ως Critical Hit.

```
136     private void dismember(int bodypart) {
137         switch (bodypart) {
138             case 1:  // left arm
139                 transform.GetChild(5).gameObject.SetActive(false);
140                 transform.GetChild(6).gameObject.SetActive(false);
141                 transform.GetChild(7).gameObject.SetActive(false);
142                 break;
143             case 2:  // left arm palm
144                 transform.GetChild(7).gameObject.SetActive(false);
145                 break;
146             case 3:  // left arm forearm
147                 transform.GetChild(6).gameObject.SetActive(false);
148                 transform.GetChild(7).gameObject.SetActive(false);
149                 break;
150             case 4:  // right arm
151                 transform.GetChild(8).gameObject.SetActive(false);
152                 transform.GetChild(9).gameObject.SetActive(false);
153                 transform.GetChild(10).gameObject.SetActive(false);
154                 break;
155             case 5:  // right arm palm
156                 transform.GetChild(10).gameObject.SetActive(false);
157                 break;
158             case 6:  // right arm forearm
159                 transform.GetChild(9).gameObject.SetActive(false);
160                 transform.GetChild(10).gameObject.SetActive(false);
161                 break;
162             case 10:  // head
163                 transform.GetChild(1).gameObject.SetActive(false);
164                 zHealth /= 2f;
165                 break;
166             default:
167                 break;
168         }
169     }
```

Η επιλογή του συγκεκριμένου μοντέλου για το zombie ήταν προσχεδιασμένη για αυτήν την λειτουργία καθώς προσφέρει κάθε μέλος τους σώματος ξεχωριστά και τα ενώνει σε ένα αντικείμενο “zombie”, έτσι εμείς μπορούμε να απενεργοποιούμε κομμάτια αρκεί να έχει κάποια συνοχή ώστε να κομμάτια χωρίς εμφανής ένωση στο υπόλοιπο σώμα (όπως για παράδειγμα στην πρώτη εικόνα πιο κάτω)

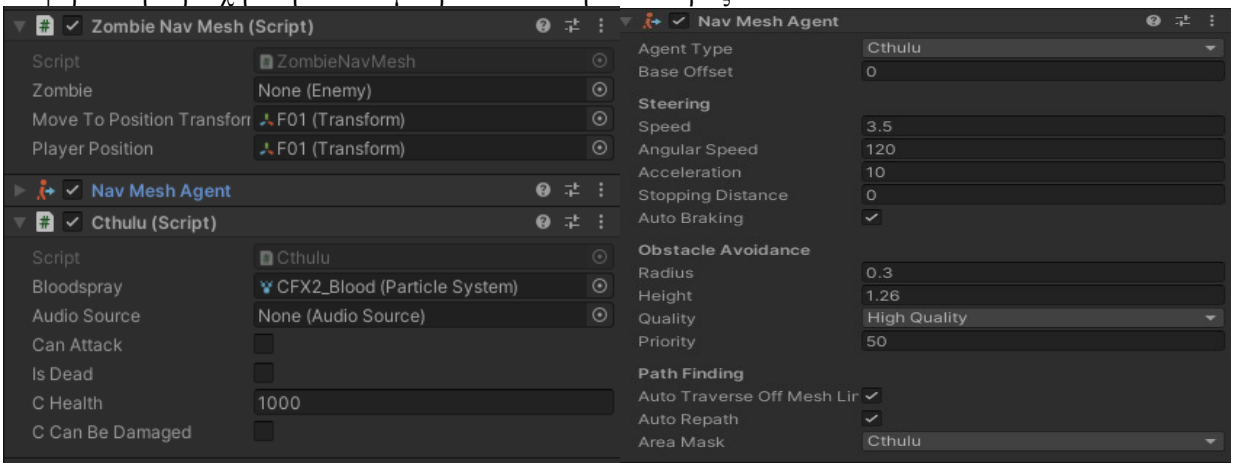


Ένας άλλου είδους μηχανισμός είναι η δυνατότητα να μπαίνουν σε “Rage Mode” δηλαδή να οργίζονται, όπου κάνει τα zombies να τρέχουν με την διπλάσια ταχύτητα αφού πρώτα σταματήσουν κάνουν ένα ηχητικό εφέ και ένα animation για να προϊδεάσουν τον παίκτη και μετά θα αναπαράγουν συνεχόμενα έναν “rage” ήχο όσο κυνηγάνε τον παίκτη.

```
118     private void handleRage() {
119         if (!player.isDead && !zRageMode && zHealth <= zHealthRangeMin / 3) {
120             zRageMode = true;
121 
122             agent.isStopped = true;  // idle it
123             animator.SetBool("isMoving", false);  // idle it
124 
125             animator.SetBool("Attack", true);  // start raging mode
126             StartCoroutine(playSound("idle", loop: false));
127             animator.SetBool("Attack", false);  // start raging mode
128 
129             agent.speed *= 2;  // multiply speed by two
130             agent.isStopped = false;  // start ai again
131             animator.SetBool("isMoving", true);  // start moving again
132             StartCoroutine(playSound("rage", loop: true));
133         }
134     }
```

3.5.3 Final Boss

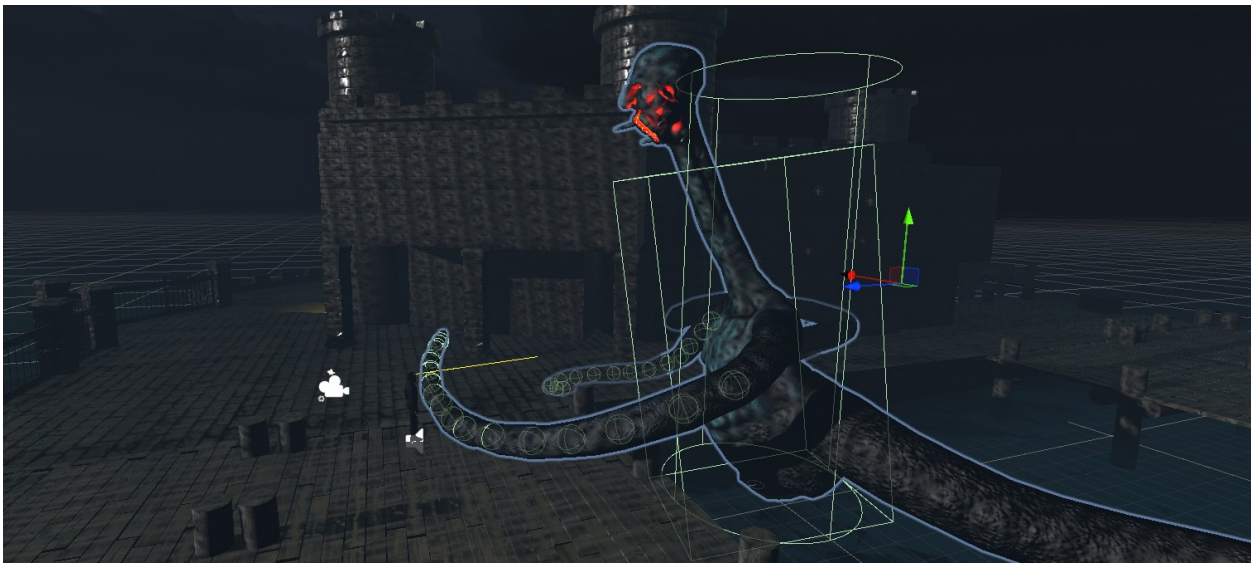
Το final boss του παιχνιδιού χρησιμοποιεί παρόμοιες λειτουργίες με τα zombies αλλά θέλαμε να δημιουργήσουμε κάτι πιο διαφορετικό μιας και η φύση του τέρατος είναι διαφορετική από το ανθρωπόμορφο zombie, και έτσι επιθέσεις του είναι βασισμένες στα animations για να καλύψουν χτυπήματα με τα πλοκάμια τα οποία πραγματοποιεί. Πέρα από αυτό χρησιμοποιεί το ίδιο ZombieNavMesh.cs script για να πηγαίνει προς τον παίκτη αλλά έχουμε άλλο Agent Type γιατί έχει διαφορετική περιοχή στην οποία μπορεί να κουνηθεί το τέρας.

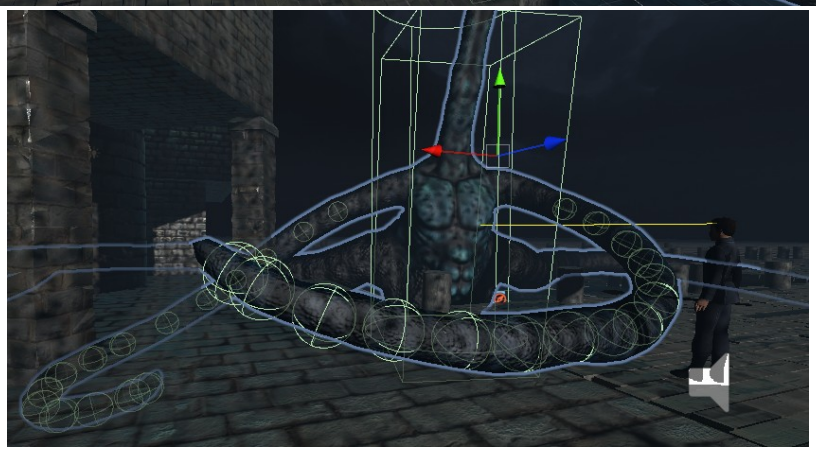


Χρησιμοποιεί όμως διαφορετικό script για την συμπεριφορά του, το Cthulu.cs. Όταν ο παίκτης έρθει σε απόσταση λιγότερη από 20 μέτρα το τέρας θα κοιτάζει μόνιμα τον παίκτη, όταν η απόσταση πέσει κάτω από 15 μέτρα τότε θα ξεκινήσει να κάνει επίθεση. Όταν συμβεί αυτό παίζει ένα animation για χτυπήσει τον παίκτη.

```
61      private IEnumerator Attack() {
62          switch (Random.Range(0, 3)) {
63              case 0:
64                  animator.SetTrigger("R");
65                  break;
66              case 1:
67                  animator.SetTrigger("L");
68                  break;
69          }
```

Κάθε πλοκάμι του τέρατος έχει πολλά Sphere Colliders ώστε να προσομοιώσουν ένα κανονικό χτύπημα στον παίκτη.





Με αυτόν τον τρόπο γίνεται πιο ρεαλιστική η επίθεση καθώς και πιο επικίνδυνη γιατί υπάρχει πιθανότητα να σε πετύχουν πολλά Sphere Colliders με μια μόνο επίθεση και ο παίκτης να χάσει μεγάλο μέρος των πόντων ζωής του. Αλλά αυτός είναι ο σκοπός του Final Boss. Επίσης με αυτόν τον τρόπο ο παίκτης έχει την δυνατότητα να αποφυγή την επίθεση με το να κάνει άλμα την κατάλληλη στιγμή.

4. Συμπεράσματα

Έπειτα από την ολοκλήρωση του παιχνιδιού καταφέραμε να καταλήξουμε σε ορισμένα συμπεράσματα που αφορούν στην δημιουργία των video games. Αρχικά, θα πρέπει να επισημάνουμε πως είναι αρκετά δύσκολο για δύο προγραμματιστες να δημιουργήσουν ένα παιχνίδι χωρίς να έχουν φτιάξει τα δικά τους assets, όπως οι χαρακτήρες και τα διάφορα αντικείμενα του παιχνιδιού. Αυτό συμβαίνει διότι επειδή τα στοιχεία που χρησιμοποιήθηκαν προέρχονται από διαφορετικούς δημιουργούς και πολλές φορές δεν ταιριάζουν μεταξύ τους με αποτέλεσμα να μην είναι επαρκώς ή και καθόλου λειτουργικά και αισθητικά ωραία σε ορισμένες περιπτώσεις. Για παράδειγμα, στην αρχή της διαδικασίας δημιουργίας του βιντεοπαιχνιδιού δυσκολευτήκαμε να βρούμε assets που να είναι συμβατά μεταξύ τους καθώς ορισμένα ήταν φτιαγμένα για URP, άλλα για HDRP και κάποια για Standard.

Επιπλέον, για να μπορέσουμε να χρησιμοποιήσουμε την μηχανή Unity χρειάστηκε να συμβουλευτούμε αρκετα tutorials τα περισσότερα εκ των οποίων προέβαλαν λειτουργίες της μηχανής οι οποίες δεν υπάρχουν πλέον ή δεν λειτουργούν με τον ίδιο τρόπο στις καινούργιες εκδοχές της. Λόγου χάρη, τα περισσότερα tutorials αναφέρονταν στο Legacy Input System το οποίο έχει ανανεωθεί από την έκδοση 2019 και είναι αρκετά διαφορετικό. Για την κατασκευή του παιχνιδιού εμείς ωστόσο, επιλέξαμε το Legacy Input System διότι προσφέρει καλύτερη εκμάθηση του κώδικα της Unity. Σε ένα καινούργιο παιχνίδι όμως με της γνώσεις αυτές θα χρησιμοποιούσαμε το καινούργιο σύστημα.

Επίσης παρότι το παιχνίδι είναι 2 επίπεδα και φαίνεται μικρό σε διάρκεια χρειάστηκε πολύ προσπάθεια στην ολοκλήρωση όλων τον μηχανισμών και αυτό είναι που μας πήρε και τον περισσότερο χρόνο. Πρέπει να ληφθεί υπόψη ότι επειδή είμασταν 2 άτομα έπρεπε να χρησιμοποιήσουμε όπου δεν έβγαζε πόλη νόημα και χρειαζόταν παραμετροποίηση για σωστή λειτουργία καθώς προσπαθούσε να συγχρονίσει πολλές φορές προσωρινά αρχεία. Ανάλογα λοιπόν με το μέγεθος της ομάδας θα χρειαστεί σίγουρα κάποιο τέτοιο εργαλείο καθώς και κάποιο git για την ομαλή λειτουργία και ανάπτυξη του έργου καθώς κάθε κομμάτι στην δημιουργία ενός παιχνιδιού έχει μεγάλο όγκο και χριαζεται συνεργασία.

References

About Visual Scripting | Visual Scripting | 1.7.8. (2022, May 10). Unity - Manual. Retrieved October 2, 2022, from https://docs.unity3d.com/Packages/com.unity.visualscripting@1.7/manual/index.html

Egenfeldt-Nielsen, S., Tosca, S. P., & Smith, J. H. (2019). *Understanding Video Games: The Essential Introduction*. Routledge.

Nicoll, B., & Keogh, B. (2019). *The Unity Game Engine and the Circuits of Cultural Software*. Springer International Publishing.

Unity: Analyzing the First Game Engine IPO — Naavik. (n.d.). Naavik. Retrieved September 17, 2022, from https://naavik.co/deep-dives/unity-analysing-the-first-game-engine-ipo

Unity Technologies. (2019, 10 19). *Introducing the new Input System*. Introducing the new Input System. https://blog.unity.com/technology/introducing-the-new-input-system

Unity Technologies. (2022, 9 23). *Scripting API*. Unity Documentation. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

ΧΡΗΣΙΜΟΙ ΣΥΝΔΕΣΜΟΙ

- https://unity.com
- https://www.audacityteam.org
- https://www.gimp.org
- https://www.blender.org
- https://forum.unity.com
- https://www.youtube.com/c/unity