ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΙΟΥΡΓΙΑ ΤΡΙΣΔΙΑΣΤΑΤΟΥ ΠΑΙΧΝΙΔΙΟΥ ΜΕ ΤΟ UNITY ΣΕ C#

Μπάρμπας Βασίλειος

Επιβλέπων: Γλαβάς Ευριπίδης, Καθηγητής

ΙΩΑΝΝΙΝΑ, Οκτώβριος, 2023

CREATION OF A THREE-DIMENSIONAL GAME WITH

UNITY USING C#

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 26/10/2023

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Ευριπίδης Γλαβάς

2. Μέλος επιτροπής
   Νικόλαος Γιαννακέας

3. Μέλος επιτροπής
   Αλέξανδρος Τζάλλας

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα μεταπτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Μπάρμπας Βασίλειος ΑΜ 1105

ΠΕΡΙΛΗΨΗ

Στα πλαίσια της παρούσας πτυχιακής εργασίας παρουσιάζεται η σχεδίαση και η ανάπτυξη ενός τρισδιάστατου παιχνιδιού τρίτου προσώπου με σκοπό την ψυχαγωγία του χρήστη. Στο πρώτο κεφάλαιο της εργασίας θα αναφερθώ στις μηχανές ανάπτυξης βιντεοπαιχνιδιών, την Ιστορία του Unity και την εξέλιξη της πλατφόρμας βιντεοπαιχνιδιών. Στη συνέχεια ακολουθούν οι δυνατότητες που σου παρέχει η πλατφόρμα, η σχεδίαση και οι κώδικες σε γλώσσα C# που χρειάστηκαν για την υλοποίηση του παιχνιδιού.

Λέξεις-κλειδιά: μηχανές ανάπτυξης βιντεοπαιχνιδιών, τρίτου προσώπου, Unity, κώδικες, C#

ABSTRACT

In the context of this thesis, the design and development of a third-person game is presented with the goal of entertaining the user. In the first chapter of the project I will refer to the videogame development engines, the history of Unity and the evolution of the videogame platform. We continue with the capabilities provided by the platform, the design and the codes in C# that were needed for the implementation of the game.

Key-words: game development engines, third-person, Unity, codes, C#

Περιεχόμενα

Πίνακας Εικόνων

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

1.1 Ιστορική Αναδρομή

Τα προηγούμενα χρόνια η διαδικασία δημιουργίας ηλεκτρονικών παιχνιδιών ήταν ιδιαίτερα δύσκολη διότι απαιτούσε άριστες γνώσεις προγραμματισμού. Εκτός αυτού για να ολοκληρωθεί ένα παιχνίδι μπορεί να χρειαζόταν ακόμα και αρκετούς μήνες. Όλα αυτά έχουν πλέον αλλάξει. Τα τελευταία χρόνια είναι πιο εύκολη η δημιουργία παιχνιδιών από το μηδέν χωρίς να απαιτούνται ιδιαίτερες γνώσεις προγραμματισμού και σε πιο σύντομο χρονικό διάστημα. Οι μηχανές που χρησιμοποιούνται για την δημιουργία των παιχνιδιών αυτών διαθέτουν εργαλεία φωτορεαλιστικής απεικόνισης, προσομοίωσης φυσικών συστημάτων, τεχνητής νοημοσύνης, Animation, επεξεργασίας ήχου.

Παράλληλα με τη δημιουργία των πρώτων βιντεοπαιχνιδιών ξεκίνησε και η δημιουργία των game engines. Αρχικά κάθε εταιρεία είχε τη δική της μηχανή, έπειτα όμως δημιουργήθηκαν διαθέσιμα δωρεάν λογισμικά για την ανάπτυξη παιχνιδιών καθώς επίσης και tutorials με αποτέλεσμα κάθε χρήστης να μπορεί να δημιουργήσει το δικό του παιχνίδι. Χρονολογικά, περίπου στις αρχές της δεκαετίας του 90 έκαναν την εμφάνιση τους οι μηχανές video games. Αρχικά, με 2D γραφικά και μετέπειτα με 3D γραφικά. Δυο παιχνίδια που βοήθησαν πολύ στην εξελικτική πορεία των μηχανών είναι το DOOM και το Quake.

1.2 Μηχανές Ανάπτυξης Βιντεοπαιχνιδιών

Ορισμένες από τις πιο δημοφιλής μηχανές παιχνιδιών είναι:

- Cry Engine
- Unreal Engine
- Unity

Το Cry Engine είναι μηχανή παιχνιδιών που δημιουργήθηκε από τη γερμανική εταιρεία Crytek και έγινε γνωστή ως η αφετηρία των παιχνιδιών Far Cry. Χαρακτηριστικό της πλατφόρμας είναι η δυνατότητα πρόσβασης στον πηγαίο κώδικα του επεξεργαστή του, κάτι που δίνει μεγάλη ευελιξία και προσαρμογή. Συνήθως επιλέγουν την συγκεκριμένη πλατφόρμα

για παιχνίδια τύπου Virtual Reality(VR) και λόγω των εξελιγμένων renderers που παρέχει στους σχεδιαστές.

- Γλώσσα προγραμματισμού: C++, Lua, C#

- Πλατφόρμες που είναι συμβατή: Windows, Linux, PS3, PS4, PS5, Xbox One,

  Xbox Series X and Series S, Nintendo Switch, Wii U

Εικόνα 1.1 Cry Engine Logo

Μια ακόμα πολύ δημοφιλής πλατφόρμα δημιουργίας παιχνιδιών είναι η Unreal Engine. Η συγκεκριμένη μηχανή παιχνιδιών αναπτύχθηκε το 1998 από την Epic Games με πρώτο παιχνίδι το Shooter unreal. Ο αρχικός σκοπός της πλατφόρμας ήταν η δημιουργία παιχνιδιών πρώτου προσώπου(FPS), κατά την εξέλιξη της πλατφόρμα οι χρήστες την επέλεγαν για διάφορα είδη βιντεοπαιχνιδιών λόγω των δυνατοτήτων της. Τέλος ήταν ιδανική για παιχνίδια με πολλαπλούς παίχτες επειδή παρείχε μεγάλη γκάμα εργαλείων για το multiplayer κομμάτι.

- Γλώσσα προγραμματισμού: C++

- Πλατφόρμες που είναι συμβατή: Windows, macOS, PS4, PS5, Xbox One,

  Xbox Series X and Series S. Android, iOS, Nintendo Switch, ARKit, ARcore, SteamVR, Open XR, Oculus, SteamDeck

Εικόνα 1.2 Unreal Engine Logo

Μια ακόμα μηχανή την οποία και επέλεξα για τη δημιουργία του παιχνιδιού που θα αναφέρω στην πορεία της εργασίας είναι το Unity. Το λογισμικό χρησιμοποιείται τόσο για 2D όσο και για 3D γραφικά. Το Unity έκανε την εμφάνιση του το 2002 από τον Δανό Nicholas Francis. Σύμφωνα με το κοινό της η Unity θεωρείται η νούμερο ένα μηχανή για τη δημιουργία παιχνιδιών για κινητές συσκευές με μια από τις πιο γνωστές και μεγάλες επιτυχίες της να είναι το Pokemon Go. Επίσης χρησιμοποιείται αρκετά συχνά για τη δημιουργία βιντεοπαιχνιδιών και ειδικά για τύπου VR.

- Γλώσσα προγραμματισμού: C++, C#

- Πλατφόρμες που είναι συμβατή: Windows, macOS, Linux, PS4, PS5, Xbox One,

  Xbox Series X and Series S, Android, iOS, WebGL, Oculus, Android TV, tvOS,

  Nintendo Switch, ARCore, Stadia, Microsoft HoloLens, Magic Leap.

1.3 Είδη και οφέλη Βιντεοπαιχνιδιών

Μια από τις πιο διασκεδαστικές δραστηριότητες είναι τα video games και αυτό ισχύει τόσο για τους μικρούς όσο και για τους μεγάλους. Εκτός από τη διασκέδαση και την θετική επίδραση στη ψυχολογία κατά την ενασχόληση με τα βιντεοπαιχνίδια προκύπτουν κι άλλα θετικά, όπως η βελτίωση στην ταχύτητα της οπτικής επεξεργασίας πληροφοριών, η λήψη αποφάσεων και η ομαδικότητα. Πιο αναλυτικά θα δούμε παρακάτω τα είδη των παιχνιδιών και πως το καθένα δρα θετικά με την ενασχόληση του.

Τα παιχνίδια στρατηγικής διεγείρουν τη μνήμη, την έκταση προσοχής και τη λογική σκέψη. Αυτό προκύπτει διότι ο παίχτης είναι αναγκασμένος να αναλύσει μια κατάσταση σε περιορισμένο χρονικό διάστημα και να καταστρώσει μια στρατηγική για το βέλτιστο πιθανό αποτέλεσμα.

Τα παιχνίδια ρόλου (Role Playing Games - RPGs) βοηθούν στην γρήγορη λήψη αποφάσεων καθώς ο παίκτης θα χρειαστεί σε πολλά σημεία του παιχνιδιού να πάρει αποφάσεις οι οποίες θα καθορίσουν την πορεία και το τέλος του.

Τα shooting games απαιτούν ταχεία επεξεργασία οπτικής πληροφορίας. Υπάρχουν δυο ειδών shooting games, τα FPS (First Person Shooter) και τα TPS (Third Person Shooter). Η διαφορά μεταξύ τους είναι η οπτική γωνία θέασης του παίχτη από την κάμερα. Σημαντικό ρόλο και στα δυο παίζει η συνεχής επαγρύπνηση του παίχτη. Κατά τη διάρκεια του παιχνιδιού πρέπει να γίνεται γρήγορα η ανάλυση των στοιχείων της οθόνης για την λήψη πληροφοριών με αποτέλεσμα ο χρήστης να εστιάζει συνεχώς σε διάφορα σημεία της. Με αυτόν τον τρόπο ο εγκέφαλος μαθαίνει να μοιράζει την προσοχή του και να βελτιώνει την επεξεργασία των εισερχόμενων οπτικών πληροφοριών.

Ανεξάρτητα από το είδος του παιχνιδιού από τα πιο σημαντικά προτερήματα που έχουν να προσφέρουν είναι η ομαδικότητα. Τα παιχνίδια μπορεί να είναι ομαδικά online ή ακόμα και offline με φίλους. Στόχος τους είναι με τη συνεργασία των ατόμων της ίδια ομάδας να καταστρωθεί ένα σχέδιο με σκοπό να νικήσουν.

ΚΕΦΑΛΑΙΟ 2: UNITY

2.1 Εισαγωγή

Το Unity αποτελεί μια μηχανή που χρησιμοποιείται τόσο για 2D όσο για 3D. Δεν αφορά μόνο παιχνίδια αλλά και εφαρμογές όπως για παράδειγμα εκπαίδευση προσομοιωτών. Το λογισμικό Unity παρέχει δυνατότητα συγγραφής κώδικα αλλά και οπτικών στοιχείων. Η απλή έκδοση είναι δωρεάν, βέβαια υπάρχει και η επαγγελματική η οποία παρέχει εξαιρετικές δυνατότητες και είναι επί πληρωμή. Επιπροσθέτως, το Unity υποστηρίζει πολλές μορφές ήχου και μπορεί να χρησιμοποιηθεί συνδυαστικά με το Photoshop διότι αναγνωρίζει αρχεία με κατάληξη σε .psd .

Εικόνα 2.1 Unity Logo

Ένα από τα πολύ σημαντικά κομμάτια του Unity αποτελεί το Unity Asset Store. Σε αυτό μπορείτε να βρείτε οτιδήποτε, όπως για παράδειγμα:

- Μοντέλα 3D
- Αρχεία κινουμένων σχεδίων
- Εφέ ήχου

2.2 Χαρακτηριστικά Unity

Το Unity είναι μια από τις πιο γνωστές μηχανές παιχνιδιών και έχει διακριθεί για πολλούς λόγους. Ορισμένα από τα κύρια χαρακτηριστικά του Unity που την καθιστούν τόσο δημοφιλή και επιτυχημένη είναι τα εξής.

1. Ευκολία Χρήσης:

   Το Unity προσφέρει μια εύχρηστη διεπαφή και ένα ενιαίο σύστημα δημιουργίας

παιχνιδιών που καθιστούν την ανάπτυξη παιχνιδιών προσβάσιμη για αρχάριους αλλά επίσης αρκετά ισχυρή για πιο έμπειρους προγραμματιστές.

2. Συμβατό με πολλές πλατφόρμες:
   Το Unity υποστηρίζει διαφόρων ειδών πλατφόρμες όπως υπολογιστές, κινητά τηλέφωνα, tablet, VR και άλλες συσκευές επιτρέποντας στους προγραμματιστές να δημιουργούν παιχνίδια που λειτουργούν σε πολλές συσκευές χωρίς μεγάλες αλλαγές στον κώδικα.
3. Κοινότητα και Υποστήριξη:
   Υπάρχει μια μεγάλη και ενεργή κοινότητα προγραμματιστών και αναπτυξιακών ομάδων που παρέχουν υποστήριξη και μοιράζονται γνώσεις μέσω φόρουμ, sites και social media.
4. Γραφικά και Ήχος:
   Το Unity προσφέρει προηγμένα γραφικά και ήχο, συμπεριλαμβανομένων νόμους φυσικής και ειδικού φωτισμού που επιτρέπουν τη δημιουργία ενός πιο ρεαλιστικού περιβάλλοντος.
5. Δωρεάν Έκδοση και Ανοιχτό Κώδικα:
   Υπάρχει μια δωρεάν έκδοση του Unity που είναι διαθέσιμη για όλους και επιπλέον παρέχει πρόσβαση στον πηγαίο κώδικα για περαιτέρω προσαρμογές από προγραμματιστές.

2.3 Ιστορική αναδρομή στο Unity

Τον Μάιο του 2002 ο Δανός προγραμματιστής Nicholas Francis μέσα από μια δημοσίευση σε ένα φόρουμ ζητάει βοήθεια σχετικά με την μηχανή ανάπτυξης παιχνιδιών του. Η απάντηση έρχεται από τον Joachim Ante ο οποίος προσφέρθηκε να τον βοηθήσει καθώς αντιμετώπιζε παρόμοια προβλήματα με την δική του μηχανή. Μετά από όχι και πολύ μεγάλο χρονικό διάστημα όμως αποφασίζουν να δημιουργήσουν από το μηδέν ένα καινούριο game engine. Η είδηση αυτή ενθουσίασε τον τρίτο της παρέας, τον David Helgason, ο οποίος αμέσως γίνεται το τρίτο μέλος της ομάδας.

Μετακομίζουν στην Κοπεγχάγη και ξεκινάνε να γράφουνε μια πολύ σημαντική σελίδα στην ιστορία της βιομηχανίας των βιντεοπαιχνιδιών. Αυτό που έκανε τη πλατφόρμα τους τόσο σημαντική και επιτυχημένη είναι το γεγονός ότι εκείνη την εποχή πολλοί προγραμματιστές όχι

απλά δεν μπορούσαν να δημιουργήσουν τις δικές του μηχανές ανάπτυξης αλλά ούτε να αποκτήσουν άδεια χρήσης κάποιας μηχανής ώστε να υλοποιήσουν τα δικά τους παιχνίδια.

Το 2004 γίνεται και επίσημα η έναρξη της εταιρείας των τριών με επωνυμία Over the Edge Entertainment. Διευθύνων σύμβουλος της εταιρείας είναι ο Helgason, ο οποίος και ήταν ο πιο επικοινωνιακός από τους τρεις. Το επιχειρηματικό σχέδιο που ακολούθησαν ήταν όχι απλά να αναπτύξουν μια μηχανή ανάπτυξης, αλλά και στη δημιουργία μεγάλων ονομάτων που θα την χρησιμοποιούσαν. Θεώρησαν πως οι καταναλωτές δεν θα ήθελαν να αγοράσουν μία ακριβή μηχανή ανάπτυξης χωρίς να ξέρουν ότι μεγάλα ονόματα την χρησιμοποιούσαν. Πίσω από το σκεπτικό αυτό ήταν η Βρετανική εταιρεία ανάπτυξης παιχνιδιών Criterion η οποία ήταν πολύ επιτυχημένη εκείνη την περίοδο και είχε ακριβώς το ίδιο επιχειρηματικό σχέδιο.

Έχοντας φτάσει κοντά στην κυκλοφορία του Unity, αποφασίζουν να δημιουργήσουν ένα εμπορικό παιχνίδι χρησιμοποιώντας τη νέα τους μηχανή. Οι λόγοι για την κίνηση αυτή ήταν δύο, ο πρώτος λόγος ήταν οικονομικός και ο δεύτερος για να δοκιμαστούν τα όρια και οι αντοχές της μηχανής τους στην πράξη. Το 2005 δημοσιεύεται το πρώτο παιχνίδι της Unity, το GooBall και με τα έσοδα από αυτό η Over The Edge Entertainment προσλαμβάνει περισσότερους προγραμματιστές ώστε να κάνει τροποποιήσεις στην μηχανή της και να την βελτιώσει.

Το ξεκίνημα δεν ήταν πολύ καλό για την εταιρεία μας, οι περισσότεροι χρήστες της μηχανής ήταν ερασιτέχνες και ανεξάρτητοι προγραμματιστές ενώ οι εταιρείες που το δοκιμάζουν στα project τους αποτυγχάνουν στις πωλήσεις παταγωδώς. Η εταιρεία δεν το βάζει κάτω και 2 χρόνια αργότερα βγάζει στην αγορά την έκδοση 2.0 της Unity η οποία επικεντρωνόταν στο να ενισχύσει την υποστήριξη σε Windows. Το 2008 βγαίνει στην αγορά το πρώτο iPhone φέρνοντας το AppStore. Η γνωστή με το όνομα πλεόν Unity Technologies αποφασίζει αμέσως να αναπτύξει μία έκδοση της η οποία θα χρησιμοποιηθεί για δημιουργία παιχνιδιών που θα τρέχουν σε iOS λογισμικό. Το 2010 έρχεται η έκδοση 3.0 με μεγάλες προσθήκες και αναβαθμίσεις. Πιο σημαντική αλλαγή από όλες όμως είναι το γεγονός ότι συγκεντρώνεις όλες τις εκδόσεις που απαιτούσαν διαφορετικό εξωτερικό πρόγραμμα επεξεργασίας σε μία. Αμέσως αποκτάει πολλούς εγγεγραμμένους χρήστες και καταφέρνει να γίνει η νούμερο ένα μηχανή για εκπαιδευτικούς σκοπούς αλλά και για δημιουργία περιεχομένου σε κινητά τηλέφωνα. Το 2012 βγαίνει η έκδοση 4.0 και το 2015 η έκδοση 5.0 με μεγάλες αναβαθμίσεις σε ήχους και γραφικά και οι χρήστες πλέον ξεπερνούν τα 5.5 εκατομμύρια. Σήμερα η Unity έχει 6 Version και κυριαρχεί στην ανάπτυξη mobile games τόσο

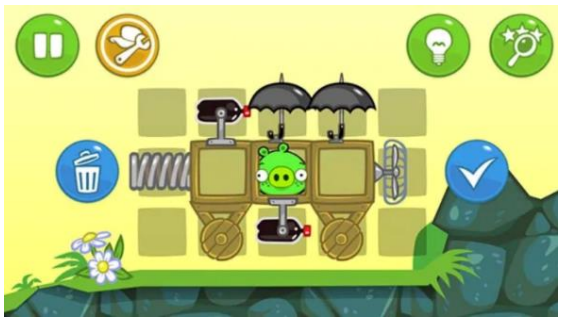
σε iOS όσο και σε Android και σύμφωνα με έρευνες χρησιμοποιείται από ένα ποσοστό προγραμματιστών που ξεπερνάει το 50%.

2.4 Παιχνίδια που έχουν δημιουργηθεί από το Unity

Το πρώτο παιχνίδι το οποίο δημιουργήθηκε από την μηχανή παιχνιδιών Unity ή αλλιώς Over the Edge Entertainment όπως ονομαζόταν η εταιρεία την περίοδο εκείνη είναι το "GooBall" και δημοσιεύθηκε το 2005. Το παιχνίδι βέβαια δεν γνώρισε την επιτυχία που περίμεναν οι δημιουργοί του καθώς ήταν υψηλό το επίπεδο δυσκολίας του. Έως και το 2009 δημιουργούνται άλλα 4 παιχνίδια με το Unity αλλά η πρώτη μεγάλη επιτυχία έρχεται τη χρονιά 2010 στην οποία δημιουργούνται 5 παιχνίδια στο διάστημα ενός χρόνου. Την χρονιά 2011 δημιουργούνται ακόμη 11 και το 2012 δημιουργούνται 21. Φαίνεται λοιπόν πως το Unity πλέον αρχίζει να αποκτάει μεγάλες διαστάσεις στον χώρο του game development. Μερικοί από του πιο σημαντικούς τίτλους που δημιουργήθηκαν μέσα σε αύτη την τριετία και αξίζει να σημειωθούν είναι οι εξής:

- Battlestar Galactica Online (2011)
- Temple Run (2011)
- Bad Piggies (2012)
- Escape Plan (2012)

Εικόνα 2.2 Battlestar Galactica (2011)



Εικόνα 2.3 Bad Piggies (2012)

Τα παιχνίδια που δημιουργήθηκαν γρήγορα γνώρισαν μεγάλη επιτυχία και αναγνώριση. Έτσι το Unity καταφέρνει τα επόμενα χρόνια να γίνει ίσως η κορυφαία μηχανή δημιουργίας παιχνιδιών στον χώρο και από το 2013 μέχρι και σήμερα το Unity έχει χρησιμοποιηθεί για την σχεδίασή και τον προγραμματισμό σε περισσότερα από 400 παιχνίδια.

Μέσα στα παιχνίδια αυτά ανήκουν τίτλοι όπως το Pokemon Go (2016) ίσως ένα από τα πιο γνωστά παιχνίδια παγκοσμίως. Πρόκειται για ένα από τα πρώτα παιχνίδια AR(Augmented Reality) το οποίο σχεδιάστηκε από την Niantic σε συνεργασία με την The

Pokemon Company. Το παιχνίδι κυκλοφόρησε για κινητές συσκευές iOS και Android και κατάφερε να σπάσει αμέτρητα ρεκόρ όπως οι 10 εκατομμύρια λήψεις την πρώτη εβδομάδα κυκλοφορίας.

Εικόνα 2.4 Pokemon Go (2016)

Κλείνοντας άλλος ένας τίτλος που αξίζει να αναφερθεί είναι το Ori and the Blond Forest (2015). Πρόκειται για ένα 2D παιχνίδι που έχει βραβευθεί για το υποδειγματικό level design του καθώς κατά τη διάρκεια του παιχνιδιού με την πρώτη ματιά μαγνητίζεσαι από το μαγευτικά παραμυθένιο εικαστικό του.

Εικόνα 2.5 Ori and the Blind Forest (2015)

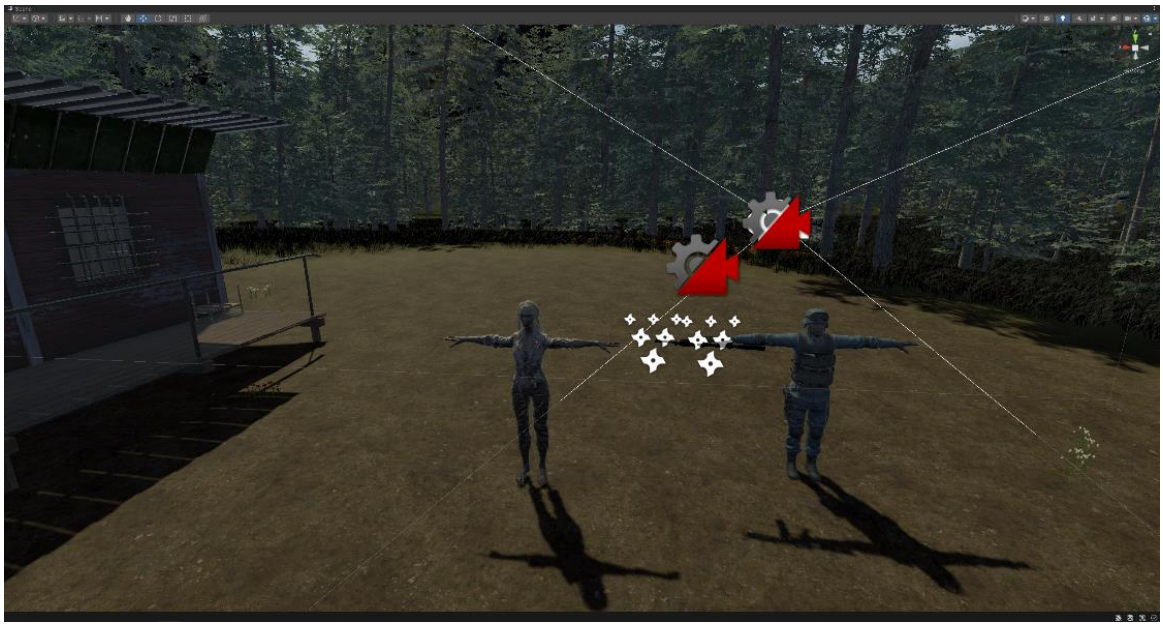
ΚΕΦΑΛΑΙΟ 3: ΔΟΜΗ ΚΑΙ ΕΡΓΑΛΕΙΑ

3.1 Unity Editor

Το κύριο μέρος στο περιβάλλον του Unity είναι το Unity Editor καθώς σε αυτόν δημιουργούνται τα projects. Στο Unity Editor εισάγονται όλα τα assets, scripts και ότι άλλο μπορεί να χρειαστεί για τη δημιουργία του παιχνιδιού και στη συνέχεια μπορούν να τροποποιηθούν αναλόγως. Ο Editor αποτελείται από διάφορα panels με τα οποία ο χρήστης έχει τη δυνατότητα δημιουργίας διεπαφής καθώς επίσης έχει τη δυνατότητα να βλέπει άμεσα την κάθε τροποποίηση που κάνει.

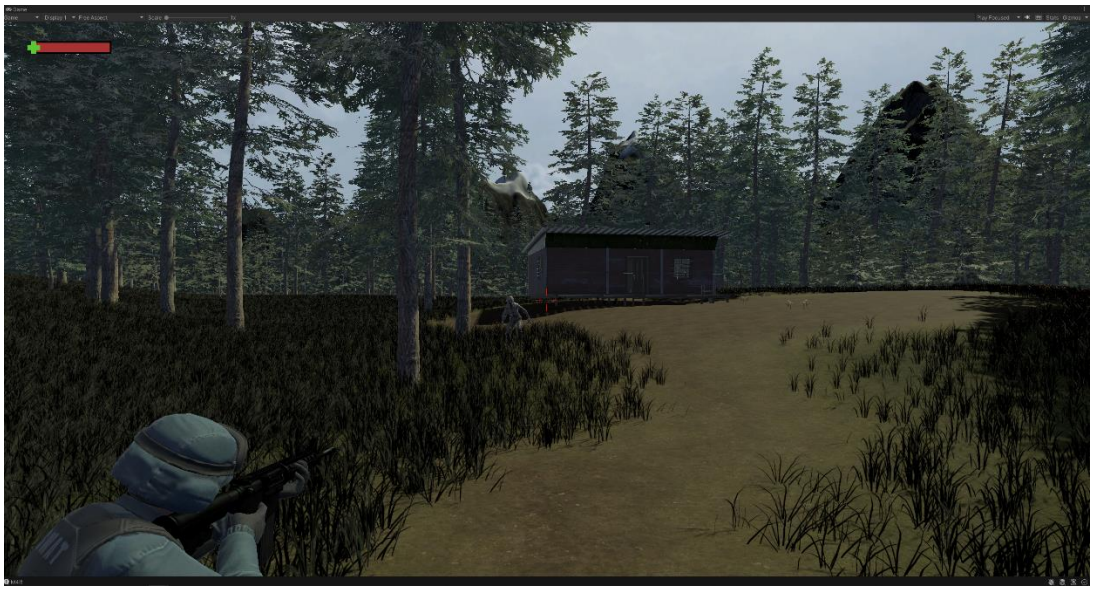
3.2 Unity Editor Panels

Το Scene View panel αποτελεί τη σκηνή, δηλαδή τον χώρο στον οποίο ο χρήστης μπορεί να δημιουργήσει και να επεξεργαστεί τα αντικείμενα που θέλει να τοποθετήσει στο παιχνίδι του.



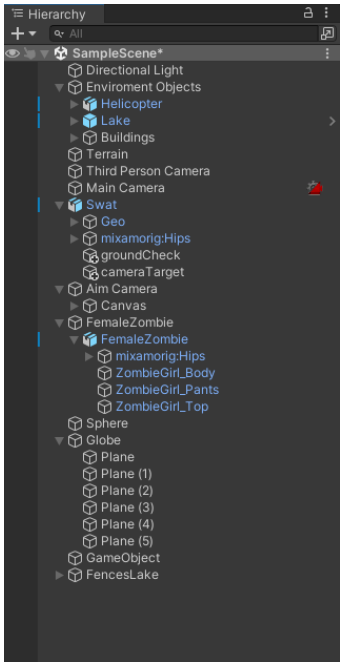
Εικόνα 3.1 Scene View

Στο Game View panel με το που πατήσουμε το κουμπί Play μπορούμε να δούμε ζωντανά πως απεικονίζεται το παιχνίδι που έχουμε δημιουργήσει και να τροποποιήσουμε οτιδήποτε δεν μας αρέσει.



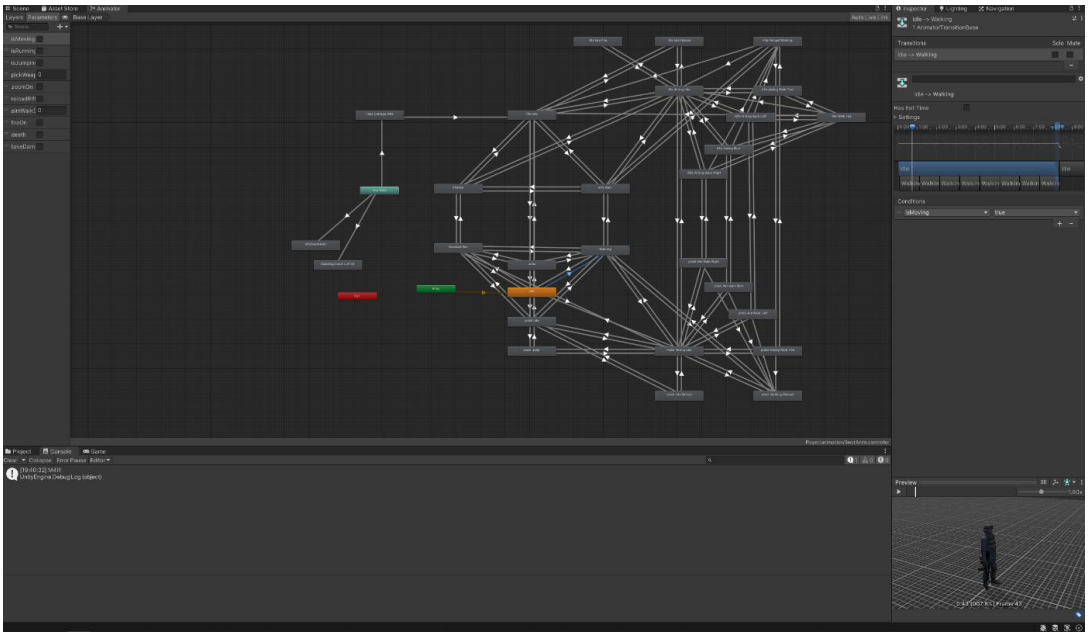
Εικόνα 3.2 Game View

Το Hierarchy panel αποτελεί ένα μενού στο οποίο βρίσκονται όλα τα αντικείμενα που έχουμε τοποθετήσει στην τρέχουσα σκηνή. Αυτή η λίστα ανανεώνεται αυτόματα αν δημιουργήσουμε ένα καινούριο αντικείμενο.



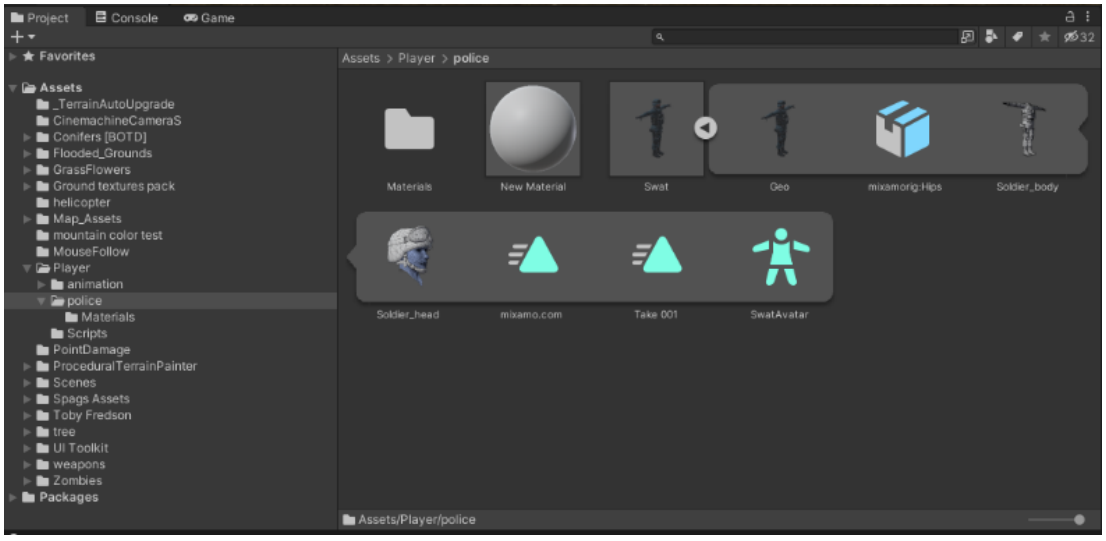
Εικόνα 3.3 Hierarchy

Στο Animator panel θα βάλουμε κινούμενα γραφικά στα αντικείμενα που θέλουμε να τους δώσουμε “ζωή” όπως για παράδειγμα ο χαρακτήρας που χειριζόμαστε όταν μετακινείται να ενεργοποιείται το animation που αναπαριστά το περπάτημα. Για να το επιτύχουμε αυτό εκτός από την εύρεση ή τη δημιουργία του animation πρέπει να ορίσουμε μια συνθήκη η οποία θα το ενεργοποιεί και απενεργοποιεί αναλόγως με την κατάσταση που βρίσκεται το αντικείμενο.



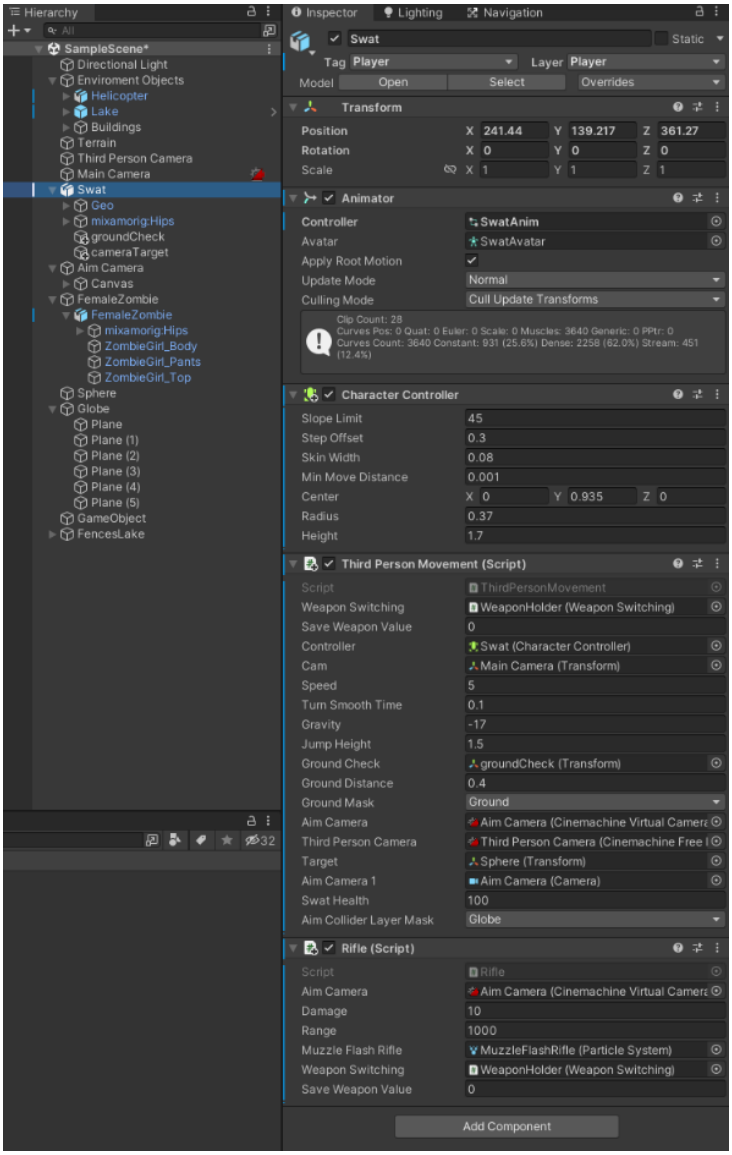
Εικόνα 3.4 Animator

Στο παράθυρο Project μπορούμε να βρούμε τους φακέλους με τα assets που έχουμε κατεβάσει για να χρησιμοποιήσουμε στην εργασία μας.



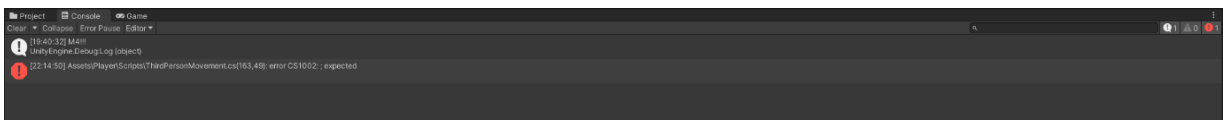
Εικόνα 3.5 Project

Το Inspector panel αποτελεί το τμήμα όπου εμφανίζονται όλα τα χαρακτηριστικά των αντικειμένων τα οποία βρίσκονται στα Hierarchy, Animator και Project panels και μπορούμε να τροποποιήσουμε τις ρυθμίσεις τους.



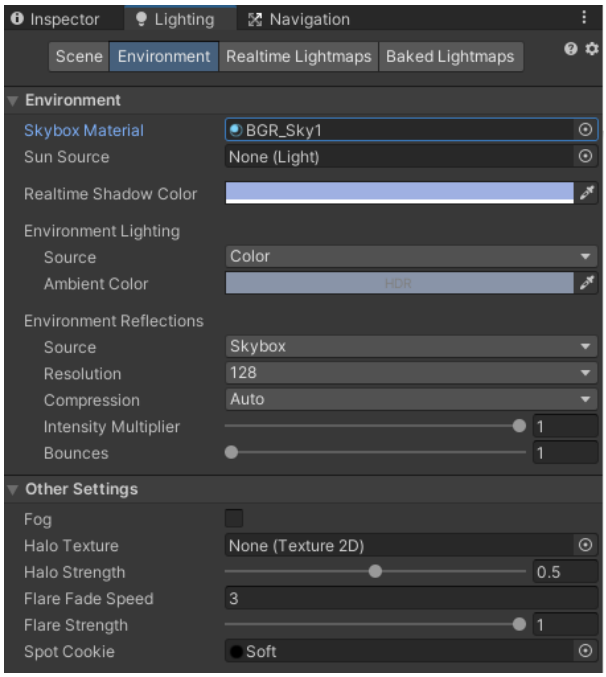
Εικόνα 3.6 Inspector

Το Console panel εμφανίζει τα μηνύματα που προκύπτουν μετά το compile και αν εμφανιστεί κάποιο error μπορούμε να δούμε σε ποιο script και ποια σειρά βρίσκεται αλλά και με διπλό αριστερό κλικ πάνω του θα μας μεταφέρει απευθείας σε αυτό.



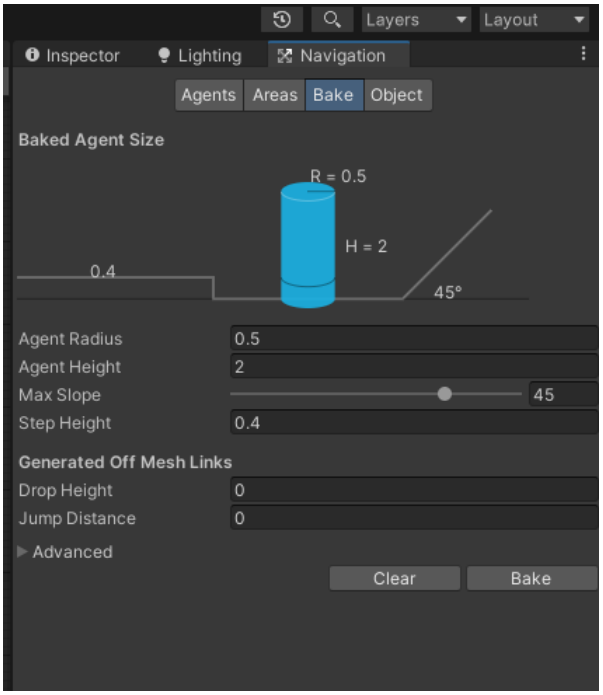
Εικόνα 3.7 Console

Στο lighting panel γίνεται η επεξεργασία των φωτισμών, χρωμάτων, σκιών και επιλογή γραφικών για τον ουρανό.



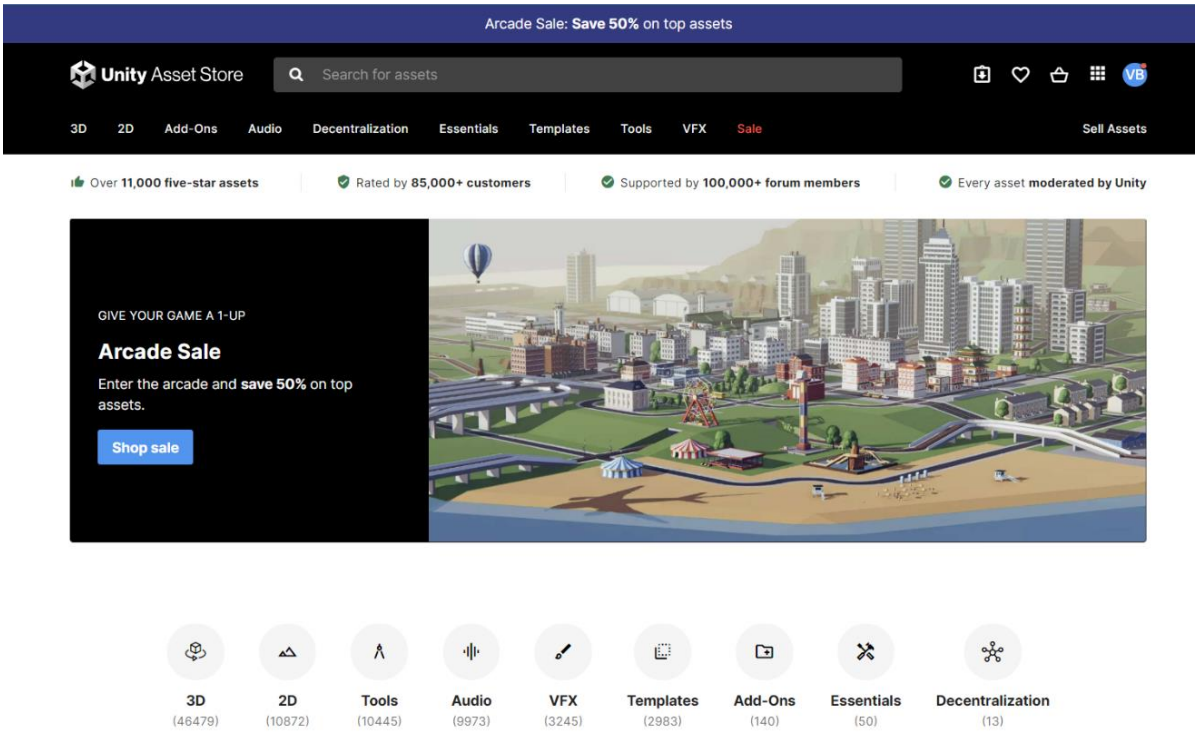
Εικόνα 3.8 Lightning

Το Navigation panel επιτρέπει στους χαρακτήρες του παιχνιδιού να βρίσκουν διαδρομές στις οποίες θα μετακινούνται με βάση τη δομή του χάρτη.



Εικόνα 3.9 Navigation

Το Asset Store είναι σελίδα της Unity στην οποία οι προγραμματιστές μπορούν να κατεβάσουν δωρεάν ή επί πληρωμή assets για να τα εισάγουν στο Project τους. Επίσης, έχουν τη δυνατότητα να δημοσιεύσουν assets τα οποία οι ίδιοι δημιούργησαν.



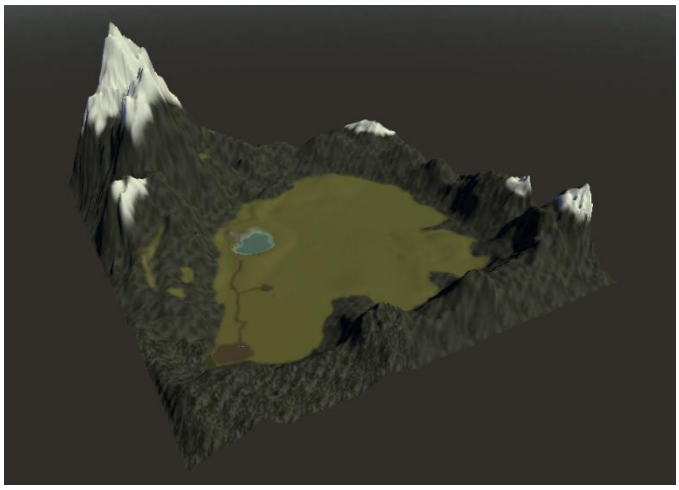
Εικόνα 3.10 Unity Asset Store

ΚΕΦΑΛΑΙΟ 4: ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΟΥ

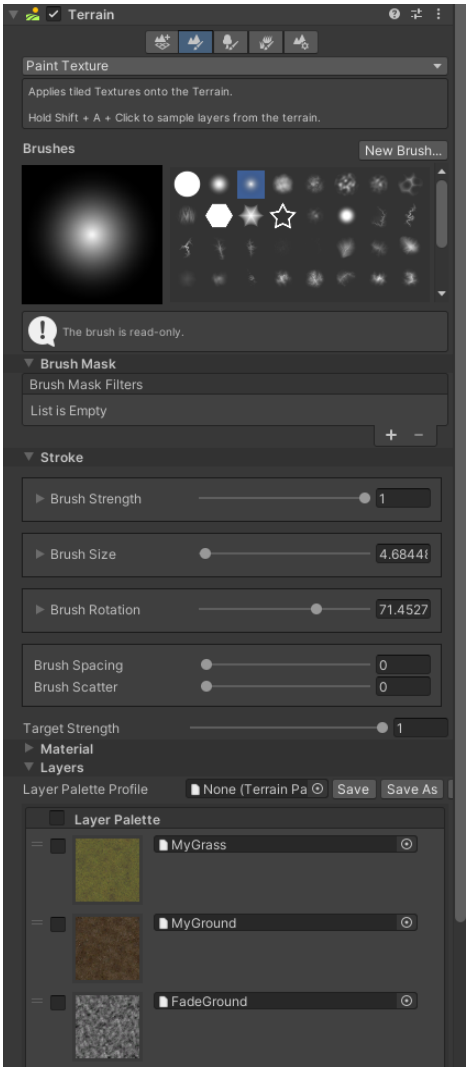
Σε αυτό το κεφάλαιο θα αναλύσουμε το πρακτικό μέρος της εργασίας καθώς εδώ θα περιγράψουμε την διαδικασία σχεδίασης του παιχνιδιού με τη χρήση της πλατφόρμας Unity. Το σενάριο του παιχνιδιού είναι η διάσωση μιας γυναίκας όπου έχει εγκλωβιστεί σε ένα βουνό το οποίο έχει κατακτηθεί από ζόμπι. Σκοπός μας είναι να εντοπίσουμε την τοποθεσία που βρίσκεται η γυναίκα και να την επιστρέψουμε σώα στο ελικόπτερο καθώς ζόμπι περιτριγυρίζουν σε όλο το χάρτη. Στη συνέχεια θα γίνει αναλυτική περιγραφή των βημάτων που ακολούθησα για την υλοποίηση του.

4.1 Map Design

Αρχικά, για την υλοποίηση του παιχνιδιού πρέπει να γίνει η δημιουργία ενός Terrain. Το Terrain είναι ουσιαστικά το "πάτωμα" στο οποίο θα γίνει η αναπαράσταση του παιχνιδιού και με τη χρήση εργαλείων και γραφικών που μας παρέχει το asset store μπορούμε να επιτύχουμε την δημιουργία ενός ρεαλιστικού περιβάλλοντος. Για παράδειγμα μπορούμε να επιλέξουμε διάφορα σημεία του και με την σωστή επιλογή των textures να δημιουργήσουμε βουνά, γρασίδια, χωματόδρομους και άλλα πολλά.



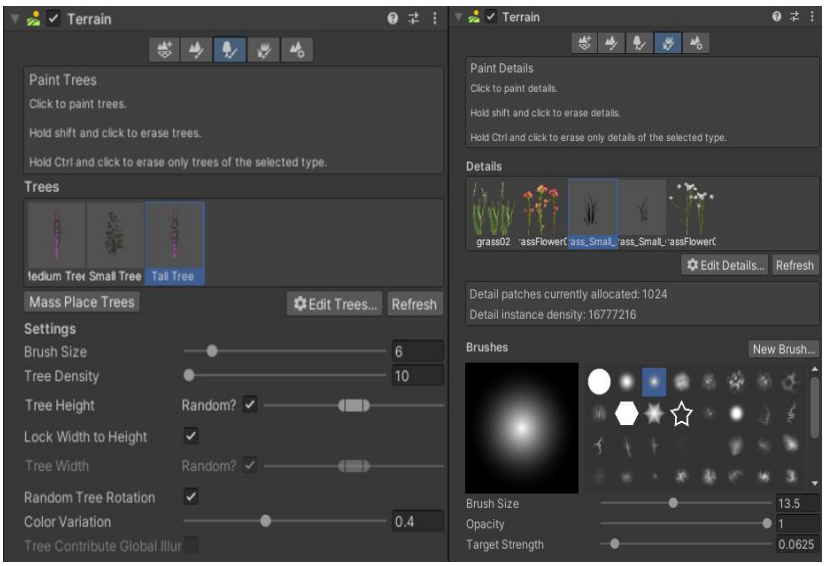
Εικόνα 4.1 το αποτέλεσμα του Terrain



Εικόνα 4.2 Inspector Panel του Terrain

Στη συνέχεια, πάλι με τη βοήθεια του asset store θα τοποθετήσουμε objects για να γεμίσουμε και να κάνουμε πιο ωραία αισθητικά τον χώρο του παιχνιδιού. Υπάρχει η επιλογή να τοποθετήσουμε το κάθε αντικείμενο ξεχωριστά ή μαζικά. Για παράδειγμα στην περίπτωση των δένδρων, επειδή ήθελα να δημιουργηθεί δάσος ο πιο εύκολος τρόπος ήταν να τοποθετήσω το object στο terrain editor και με τη χρήση βούρτσας να επιλέξω σε ποια σημεία θα έχει δένδρα. Τα πακέτα asset που επιλέχθηκαν για τη δημιουργία του παιχνιδιού είναι τα εξής:

- Rock and Boulders 2
- Conifers
- Flooded Grounds
- Rifle
- Pistol 92
- Crosshairs Plus
- Procedural Terrain Painter
- Terrain Sample Asset Pack
- Wasteland Cabin
- Grass Flowers Pack
- Fallen Tree Barrier
- Yaghues Free Ground Materials
- Fog of War Gun Sound FX



Εικόνα 4.3 Επιλογή φυτών και δέντρων

Εικόνα 4.4 Σημείο έναρξης

Εικόνα 4.5 Καλύβα

Εικόνα 4.6 Λίμνη

4.2 Δημιουργία του Swat

Για τη δημιουργία του χαρακτήρα μας θα ξεκινήσουμε με τη δημιουργία ενός 3D object capsule στο hierarchy panel. Στη συνέχεια θα αλλάξουμε το όνομα του αντικειμένου σε swat και θα του προσθέσουμε από το Inspector panel το Character Controller. Η λειτουργεία αυτού του component είναι να δώσει τη δυνατότητα στο αντικείμενο να μπορεί να μετακινείται και να αλληλοεπιδρά με το περιβάλλον που έχουμε δημιουργήσει. Η βασική ιδέα του "Character Controller" είναι το σύστημα του collider και η δυνατότητα που παρέχει στον προγραμματιστή να τροποποιήσει κάποιες από της τιμές του όπως:

Slope Limit: Η μέγιστη γωνία κλίσης (σε μοίρες) που μπορεί να αναρριχηθεί ο χαρακτήρας.

Step Offset: Αναφέρεται στην μέγιστη απόσταση όπου ο χαρακτήρας μπορεί να ανεβεί ή να κατεβεί από μια επιφάνεια όπως σκαλοπάτια ή αντικείμενα.

Skin Width: Η εξωτερική περιοχή γύρω από τον χαρακτήρα η οποία χρησιμοποιείται για την ανίχνευση συγκρούσεων και την ομαλότητα της κίνησης.

Minimum Move Distance: Η ελάχιστη απόσταση που πρέπει να μετακινηθεί ο χαρακτήρας πριν ανιχνευθεί η κίνηση του. Αυτό είναι χρήσιμο για αποφευχθούν ψευδείς σύγκρουσης ή αστάθειες στην κίνηση του χαρακτήρα.

Center: Οι συντεταγμένες x, y, z για τη θέση του κέντρου μάζας του χαρακτήρα.

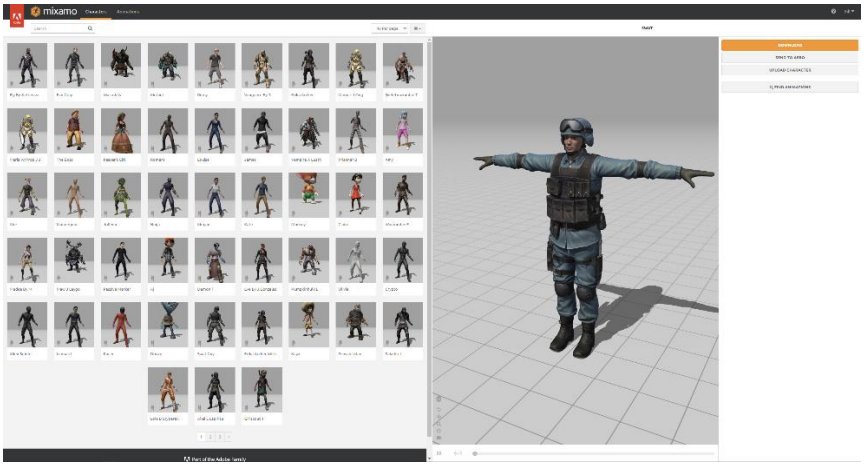
Radius: Καθορίζει το πλάτος του χαρακτήρα γύρω από το κέντρο του.

Height: Καθορίζει το ύψος του χαρακτήρα από τη βάση ως την κορυφή του.

Εικόνα 4.7 Character Controller

Στη συνέχεια θα κάνουμε μη ορατό τον κύλινδρο που δημιουργήσαμε απενεργοποιώντας την επιλογή Mesh Render και στη θέση του θα τοποθετήσουμε την αμφίεση που θέλουμε να έχει ο χαρακτήρας μας. Ένας τρόπος για την εμφάνιση του χαρακτήρα είναι η

δημιουργία των γραφικών του από το μηδέν σε εφαρμογές όπως το photoshop ή το blender και ο άλλος τρόπος είναι να βρεις έτοιμα γραφικά στο διαδίκτυο. Στην παρούσα εργασία η αμφίεση του χαρακτήρα και των αντιπάλων είναι από το site mixamo της Adobe.



Εικόνα 4.8 Mixamo Adobe

Ο παρακάτω κώδικας έχει τοποθετηθεί ως component στο αντικείμενο swat όπου είναι ο χαρακτήρας μας. Οι δύο πιο βασικές συναρτήσεις σε κάθε script είναι η Start και η Update. Η συνάρτηση Start() τρέχει μόνο μια φορά κατά τη διάρκεια του πρώτου frame και αν δεν χρειάζεται μπορεί να παραληφθεί ενώ η συνάρτηση Update() επαναλαμβάνεται σε κάθε frame και είναι απαραίτητη καθώς μέσα σε αυτή γράφουμε τον κύριο κώδικα που χρειάζεται το παιχνίδι. Ξεκινάμε με τη δήλωση τον απαραίτητων βιβλιοθηκών που θα χρειαστούν για τις ακόλουθες εντολές και συνεχίζουμε με τη δήλωση μεταβλητών. Για παράδειγμα η δήλωση μιας πιο σύνθετης μεταβλητής όπως η public AudioSource p_shootingSound δηλώνει μια μεταβλητή με όνομα p_shootingSound τύπου AudioSource, στο Inspector panel του Unity μπορούμε να επιλέξουμε ποιο θα είναι το αρχείο ήχου που θέλουμε να χρησιμοποιήσουμε και στο script θα γράψουμε τον κατάλληλο κώδικα για το πότε θέλουμε να γίνεται η αναπαραγωγή του. Έπειτα μέσα στην Start() γίνεται το κλείδωμα του κέρσορα για να είναι ομαλή η κίνηση της κάμερας και του παίχτη, ακολουθεί η κλήση του animator που θα χρησιμοποιήσουμε για τα κινούμενα γραφικά του χαρακτήρα μας. Στη συνάρτηση Update() καλούμε τις συναρτήσεις WalkingRuningJumping(), cameraPosition(), weaponSystem(), CameraSwitch(), zoomWalkAnim(), takeDamage() και στη πορεία θα εξηγηθεί αναλυτικά τι κάνει η κάθε μια από αυτές.

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

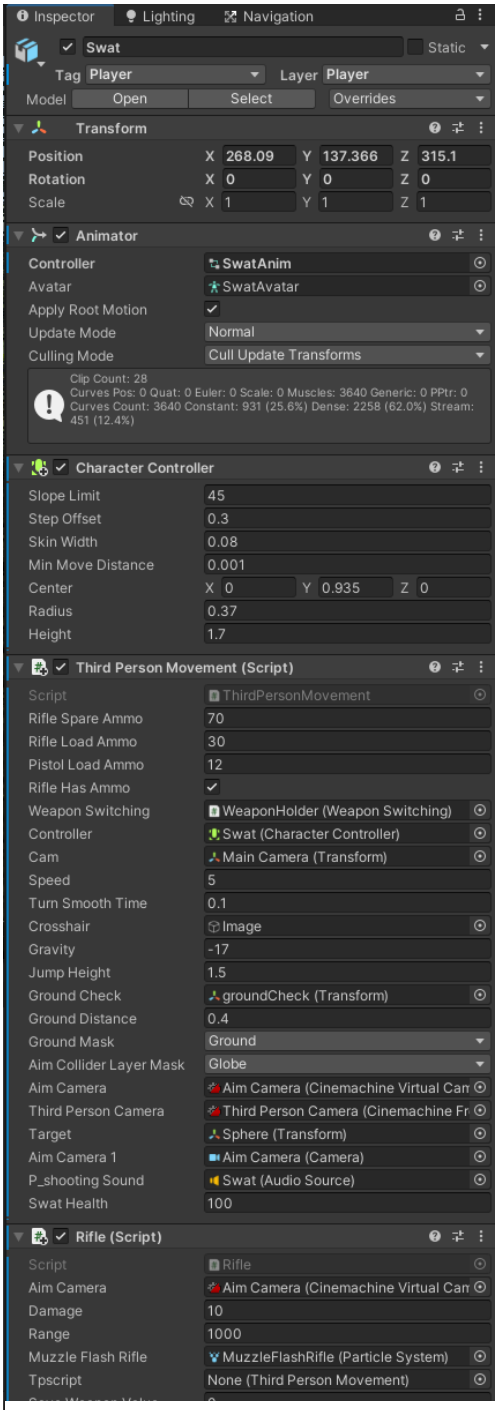
```
using Cinemachine;

public class ThirdPersonMovement : MonoBehaviour
{
        //Δήλωση μεταβλητών για τις σφαίρες
        public float RifleSpareAmmo = 70;
        public float RifleLoadAmmo = 30;
        public float PistolLoadAmmo = 12;
        private float ammoNeeded = 0;
        public bool RifleHasAmmo;

        //Δήλωση μεταβλητών Gravity, Jump και groundCheck
        public float gravity = -16f;
        public float jumpHeight = 1.5f;
        public Transform groundCheck;
        public float groundDistance = 0.4f;
        public LayerMask groundMask;
        [SerializeField]
         public LayerMask aimColliderLayerMask;
        Vector3 velocity;
        bool isGrounded;

        public Transform cam;
        public float speed = 3f;
        public float turnSmoothTime = 0.1f;
        float turnSmoothVelocity;

        public CharacterController controller;
        //Επιστρέφει τιμή από άλλο script
        με αποτέλεσμα να γνωρίζουμε ποιο
        όπλο είναι επιλεγμένο
        public WeaponSwitching weaponSwitching;
        int saveWeaponValue;
```



Εικόνα 4.9 Swat Inspector Panel

```
//UI στόχου
public GameObject crosshair;

//Animator
private Animator animator;

//Κάμερες
public CinemachineVirtualCamera aimCamera;

[SerializeField]
private Camera aimCamera1;

//Στόχος
[SerializeField]
public Transform Target;

//Ηχος
public AudioSource p_shootingSound;

void Start()
{
        Cursor.lockState = CursorLockMode.Locked;
        animator = GetComponent<Animator>();
}

void Update()
{
        WalkingRuningJumping();
        cameraPosition();
        weaponSystem();
        CameraSwitch();
        zoomWalkAnim();
        takeDamage();
```

```
    }
    *Σε αυτό το σημείο έχουν αναπτυχθεί όλες οι συναρτήσεις που είναι μέσα στη
    συνάρτηση Update()*
}
```

Ξεκινάμε με την συνάρτηση που καθορίζει την κίνηση του χαρακτήρα. Δημιουργούμε ένα Vector3 με όνομα direction το οποίο καθορίζει την κίνηση του χαρακτήρα μας, όταν η τιμή του direction δεν είναι μηδενική σημαίνει ότι το αντικείμενο swat είναι σε κίνηση ενώ όταν είναι μηδέν παραμένει ακίνητο. Εάν υπάρχει κίνηση και παράλληλα είναι πατημένο το πλήκτρο “Shift” του πληκτρολογίου τότε θέλουμε το αντικείμενο να αναπτύσσει μεγαλύτερη ταχύτητα για την αναπαράσταση του τρεξίματος. Το επόμενο βήμα είναι να ελέγξουμε αν ο παίχτης βρίσκεται στο έδαφος ή στον αέρα την δεδομένη χρονική στιγμή, μόνο εάν είναι σε επαφή με το έδαφος έχει τη δυνατότητα να κάνει άλμα πατώντας το πλήκτρο space του πληκτρολογίου. Όταν ισχύει κάποια από τις παραπάνω καταστάσεις ορίζουμε αληθής την αντίστοιχη μεταβλητή που χρησιμοποιούμε στον Animator για να επιτύχουμε μια πιο ρεαλιστική απεικόνιση του παιχνιδιού.

```
void WalkingRuningJumping()
{
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");
        Vector3 direction = new Vector3(x, 0f, z).normalized;

        //animation για το περπάτημα
        if (direction != Vector3.zero)
        {
                animator.SetBool("isMoving", true);
        }
        else
        {
                animator.SetBool("isMoving", false);
        }
```

```
            //animation για το τρέξιμο
            if (animator.GetBool("isMoving") == true && Input.GetKey(KeyCode.LeftShift))
            {
                        speed = 5f;
                        animator.SetBool("isRunning", true);
            }
            else
            {
                        speed = 3f;
                        animator.SetBool("isRunning", false);
            }
            //Άλμα
            velocity.y += gravity * Time.deltaTime;
            controller.Move(velocity * Time.deltaTime);
            isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);
            if (isGrounded && velocity.y < 0)
            {
                        velocity.y = -2f;
            }

            //να κάνει άλμα μόνο όταν βρίσκεται στο έδαφος
            if (Input.GetButtonDown("Jump") && isGrounded)
            {
                  //να αλλάζει το velocity y όταν κάνει άλμα
                        velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
                        animator.SetBool("isJumping", true);
            }
            else
            {
                        animator.SetBool("isJumping", false);
            }
```

}

4.3 Camera System

Τέλος, πρέπει να δημιουργήσουμε τον μηχανισμό για τις κάμερες. Επειδή το είδος του παιχνιδιού που επιλέξαμε είναι Third Pearson Shooter οι κάμερες θέλουμε να έχουν το ρόλο ενός “παρατηρητή” και να ακολουθούν τον χαρακτήρα μας. Ονομάσαμε τις κάμερες Aim Camera και Third Person Camera, μέσω της Main Camera θα πραγματοποιείται η εναλλαγή αυτών των δυο. Ενεργή είναι η κάμερα όπου έχει υψηλότερη τιμή στο πεδίο Priority και από τον κώδικα μπορούμε να αλλάζουμε την τιμή της μεταβλητής για να επιλέγουμε ποια από τις δυο θα είναι ενεργή τη δεδομένη χρονική στιγμή.

Εικόνα 4.10 Third Person Shooter και Aim Camera

```
void CameraSwitch()
{
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");
        Vector3 direction = new Vector3(x, 0f, z).normalized;
        if (Input.GetKey(KeyCode.Mouse1) && (saveWeaponValue == 1 || (saveWeaponValue == 2 && RifleHasAmmo == true)))
        {
                aimCamera.Priority = 30;
                crosshair.SetActive(true);
                speed = 1.5f;

                if (Input.GetKeyDown(KeyCode.Mouse0))
                {
                p_shootingSound.Play();
```

```
                speed = 1.2f;
                }
                transform.LookAt(Target);
                animator.SetBool("zoomOn", true);
                if (direction.magnitude >= 0.1f)
                {
                float targetAngle = Mathf.Atan2(direction.x, direction.z) *
Mathf.Rad2Deg + cam.eulerAngles.y;
                Vector3 moveDir = Quaternion.Euler(0f, targetAngle, 0f) *
Vector3.forward;
                controller.Move(moveDir.normalized * speed *
Time.deltaTime);
                }
        }
        else
        {
                aimCamera.Priority = 10;
                crosshair.SetActive(false);
                animator.SetBool("zoomOn", false);
                if (direction.magnitude >= 0.1f && swatHealth > 0)
                {
                float targetAngle = Mathf.Atan2(direction.x, direction.z) *
Mathf.Rad2Deg + cam.eulerAngles.y;
                //περιστροφή χαρακτήρα
                float angle1 =
Mathf.SmoothDampAngle(transform.eulerAngles.y, targetAngle, ref turnSmoothVelocity,
turnSmoothTime);
                transform.rotation = Quaternion.Euler(0f, angle1, 0f);
                Vector3 moveDir = Quaternion.Euler(0f, targetAngle, 0f) *
Vector3.forward;
                controller.Move(moveDir.normalized * speed *
Time.deltaTime);
                }
        }
```

```
}
```

Η cameraPosition() έχει εντολές για την εστίαση και την περιστροφή των δυο καμερών. Η προκαθορισμένη κάμερα του παιχνιδιού είναι η Third Person Camera στην οποία ο χαρακτήρας μας μπορεί να περιστρέφεται ανεξάρτητα με το που εστιάζει αυτή. Η αλλαγή στην δεύτερη κάμερα με όνομα aim Camera γίνεται όταν ο χαρακτήρας κρατάει στα χέρια του ένα από τα δύο όπλα του παιχνιδιού και πατήσουμε το δεξί κλικ του ποντικιού για να στοχεύσει. Η aim Camera είναι τοποθετημένη πιο κοντά στον χαρακτήρα, ο παίκτης εστιάζει στο κέντρο της οθόνης όπου βρίσκεται ο στόχος του όπλου και περιστρέφεται μαζί με αυτή.

```
void cameraPosition()
{
    Vector3 mouseWorldPosition = Vector3.zero;
    Vector2 screenCenterPoint = new Vector2(Screen.width / 2f, Screen.height / 2f);
    Ray ray = aimCamera1.ScreenPointToRay(screenCenterPoint);
    if (Physics.Raycast(ray, out RaycastHit raycastHit, 999f, aimColliderLayerMask))
    {
        Target.position = raycastHit.point;
        mouseWorldPosition = raycastHit.point;
    }

    if (Input.GetKeyDown(KeyCode.Mouse1) && animator.GetBool("zoomOn") == true)
    {
        Vector3 worldAimTarget = mouseWorldPosition;
        worldAimTarget.y = transform.position.y;
        Vector3 aimDirection = (worldAimTarget - transform.position);
        transform.forward = Vector3.Lerp(transform.forward, aimDirection, Time.deltaTime * 20f);
    }
}
```

4.4 Δημιουργία των όπλων

Το αμέσως επόμενο βήμα είναι να τοποθετήσουμε τα γραφικά και το σύστημα των όπλων. Αυτό επιτεύχθηκε με την δημιουργία ενός empty object το οποίο μετονομάστηκε “WeaponHolder”, μέσα σε αυτό τοποθετήσαμε τα όπλα του παιχνιδιού (hands, pistol 92 Dark, rifle) και το ορίσαμε ως child στο δεξί χέρι του αντικειμένου swat.

Εικόνα 4.11 Όπλα του παιχνιδιού

Στον κώδικα ορίζουμε να γίνεται η αλλαγή αυτών με τη ροδέλα του ποντικιού ή με τα πλήκτρα 1,2,3 του πληκτρολογίου από τη στιγμή που ο παίχτης έχει ακόμα πόντους ζωής.

```
using UnityEngine;

public class WeaponSwitching : MonoBehaviour
{
    public ThirdPersonMovement swat;
    public int selectedWeapon = 0;

    void Start()
    {
        SelectWeapon();
    }

    void Update()
    {
        if (swat.swatHealth > 0)
        {
            int previousSelectedWeapon = selectedWeapon;
            //αλλαγή όπλων με τη ροδέλα του ποντικιού
```

```
        if (Input.GetAxis("Mouse ScrollWheel") > 0f)
        {
            if (selectedWeapon >= transform.childCount - 1)
            {
                selectedWeapon = 0;
            }
            else
            {
                selectedWeapon++;
            }
        }
        if (Input.GetAxis("Mouse ScrollWheel") < 0f)
        {
            if (selectedWeapon <= 0)
            {
                selectedWeapon = transform.childCount - 1;
            }
            else
            {
                selectedWeapon--;
            }
        }

        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            selectedWeapon = 2;
        }
        else if (Input.GetKeyDown(KeyCode.Alpha2) &&
transform.childCount >= 2)
        {
            selectedWeapon = 1;

        }
```

```
                else if (Input.GetKeyDown(KeyCode.Alpha3) &&
transform.childCount >= 3)
                {
                        selectedWeapon = 0;
                }

                if (previousSelectedWeapon != selectedWeapon)
                {
                        SelectWeapon();
                }
        }
    }

    void SelectWeapon()
    {
        int i = 0;
        foreach(Transform weapon in transform)
        {
            if(i == selectedWeapon)
            {
                weapon.gameObject.SetActive(true);
            }
            else
            {
                weapon.gameObject.SetActive(false);
            }
            i++;
        }
    }
}
```

Στην weaponSystem() ξεκινάμε παίρνοντας την τιμή του selectedWeapon από το weaponSwitching script και την αποθηκεύουμε στη μεταβλητή saveWeaponValue που

χρησιμοποιούμε στο script του χαρακτήρα μας. Με βάση την τιμή που μας επιστρέφει η saveWeaponValue μπορούμε να καταλάβουμε ποιο όπλο έχουμε επιλεγμένο εκείνη τη στιγμή. Όταν κάποιο από τα όπλα έχει μηδέν σφαίρες στον γεμιστήρα ή πατήσουμε το πλήκτρο R του πληκτρολογίου τότε ενεργοποιείται το animation για το γέμισμα σφαιρών του όπλου και ο γεμιστήρας φτάνει στο μέγιστο αριθμό σφαιρών αν υπάρχει αρκετό απόθεμα.

```
void weaponSystem()
{
        saveWeaponValue = weaponSwitching.selectedWeapon;
        if (saveWeaponValue == 1)
        {
                animator.SetInteger("pickWeapon", 1);
        }
        else if (saveWeaponValue == 2)
        {
                animator.SetInteger("pickWeapon", 2);
        }
        else
        {
                animator.SetInteger("pickWeapon", 0);
        }

        //Losing Ammo
        if (Input.GetKeyDown(KeyCode.Mouse0) && Input.GetKey(KeyCode.Mouse1))
        {
                if (saveWeaponValue == 2 && RifleLoadAmmo > 0)
                {
                RifleLoadAmmo = RifleLoadAmmo - 1;
                }
            if (saveWeaponValue == 1)
                {
                PistolLoadAmmo = PistolLoadAmmo - 1;
                }
        }
```

```
            //Shooting
            if (animator.GetBool("zoomOn") == true && Input.GetKeyDown(KeyCode.Mouse0))
            {
                        animator.SetBool("fireOn", true);
            }
            else
            {
                        animator.SetBool("fireOn", false);
            }

            //pistol reload
            if (saveWeaponValue == 1 && PistolLoadAmmo != 12 && (Input.GetKeyDown(KeyCode.R) || PistolLoadAmmo == 0))
            {
                        animator.SetBool("reloadPistol", true);
                        PistolLoadAmmo = 12;
            }
            else
            {
                        animator.SetBool("reloadPistol", false);
            }
            //Rifle reload
            if (saveWeaponValue == 2 && RifleLoadAmmo != 30 && RifleSpareAmmo > 0 && (Input.GetKeyDown(KeyCode.R) || RifleLoadAmmo == 0))
            {
                        animator.SetBool("reloadRifle", true);
                        //DelayReload();
                        ammoNeeded = 30 - RifleLoadAmmo;

                        if (RifleSpareAmmo < ammoNeeded)
                        {
                        RifleLoadAmmo = RifleLoadAmmo + RifleSpareAmmo;
```

```
                    RifleSpareAmmo = 0;
                    }
                    else
                    {
                    RifleSpareAmmo = RifleSpareAmmo - (30 - RifleLoadAmmo);
                    RifleLoadAmmo = 30;
                    }
            }
            else
            {
                    animator.SetBool("reloadRifle", false);
            }

            if (PistolLoadAmmo == 0 || RifleLoadAmmo == 0)
            {
                    DelayReload();
            }

            if (RifleSpareAmmo == 0 && RifleLoadAmmo == 0)
            {
                RifleHasAmmo = false;
            }
            else
            {
                    RifleHasAmmo = true;
            }
    }
```

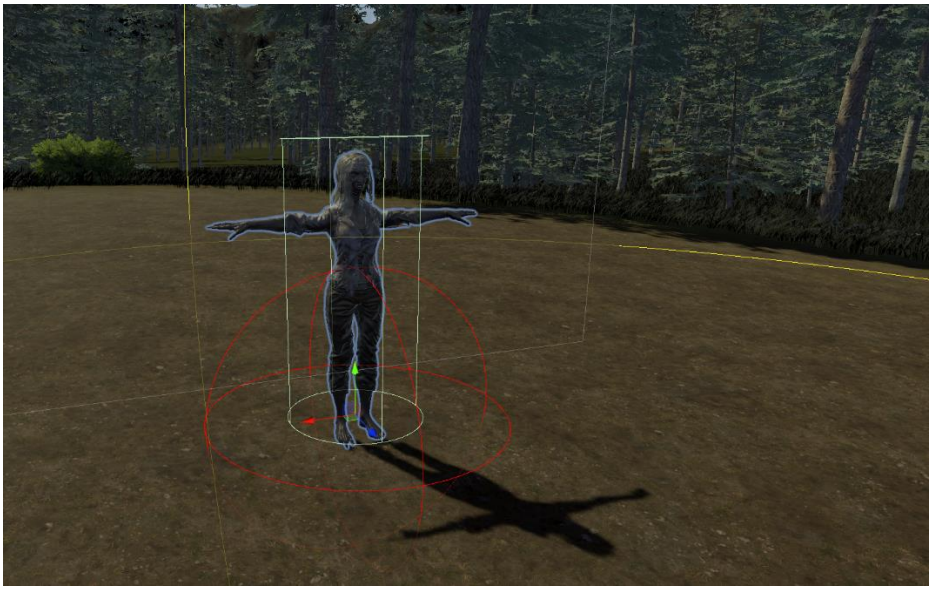
Η zoomWalkAnim() αναλόγως με την τιμή που θα επιστρέψει στην μεταβλητή aimWalkSide μπορούμε να καταλάβουμε προς ποια κατεύθυνση κινείται ο χαρακτήρας κατά τη διάρκεια που στοχεύει με το όπλο με αποτέλεσμα να ενεργοποιεί το κατάλληλο animation για το περπάτημα του.

```
    void zoomWalkAnim()
```

```
        {
            If ((Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow)) &&
animator.GetBool("zoomOn") == true)
            {
                animator.SetInteger("aimWalkSide", 1);
            }
            else if ((Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow)) &&
animator.GetBool("zoomOn") == true)
            {
                animator.SetInteger("aimWalkSide", -2);
            }
            else if ((Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow))
&& animator.GetBool("zoomOn") == true)
            {
                animator.SetInteger("aimWalkSide", -1);
            }
            else if ((Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))
&& animator.GetBool("zoomOn") == true)
            {
                animator.SetInteger("aimWalkSide", 2);
            }
            else
            {
                animator.SetInteger("aimWalkSide", 0);
            }
        }
```

4.5 Δημιουργία των Zombies

Το πρώτο βήμα είναι η δημιουργία ενός 3D αντικειμένου τύπου Cylinder και η ονομασία του σε FemaleZombie. Στη συνέχεια θα απενεργοποιήσουμε το Mesh Render για να μην είναι ορατός ο κύλινδρος και θα τοποθετήσουμε πάνω σε αυτόν τα γραφικά που έχουμε επιλέξει για τα Zombie.



Εικόνα 4.12 Zombie

Όπως φαίνεται στον κώδικα η υλοποίηση για την κίνηση των ζόμπι γίνεται χρησιμοποιώντας το NavMeshAgent, αφού γίνει επιτυχώς η σάρωση του χάρτη μπορούν να αναγνωρίζουν την περιοχή που μπορούν να πλοηγηθούν αποφεύγοντας εμπόδια και παράλληλα να γίνεται η αναζήτηση του αντικειμένου swat. Από τη στιγμή που ο swat παραμένει ζωντανός γίνεται συνεχής έλεγχος των παρακάτω συναρτήσεων.

- TakeDamage(float amount): Κάθε φορά που το ζόμπι δέχεται ζημιά, χάνει πόντους ζωής
- Die(): Όταν χαθούν όλοι οι πόντοι ζωής του σταματά να κινείται και ενεργοποιείται το animation για τον θάνατο του.
- SearchWalkPoint(): Υπολογίζει ένα τυχαίο σημείο στην περιοχή για να κινηθεί το ζόμπι.
- Patrolling(): Εάν το αντικείμενο swat βρίσκεται εκτός της εμβέλειας ορατότητας που έχουμε ορίσει στο ζόμπι, τότε το ζόμπι συνεχίζει να το αναζητά σε τυχαία μονοπάτια.
- ChasePlayer(): Εάν το αντικείμενο swat βρίσκεται εντός του ορατού πεδίου που έχουμε ορίσει για το ζόμπι αλλά ταυτόχρονα δεν είναι σε απόσταση για να δεχτεί επίθεση, τότε το ζόμπι ξεκινά να κυνηγάει τον παίκτη.
- AttackPlayer(): Εάν το αντικείμενο swat βρίσκεται εντός του ορατού πεδίου και ταυτόχρονα είναι εντός της ακτίνα που έχουμε ορίσει για να δεχθεί επίθεση, τότε το ζόμπι ξεκινά να επιτίθεται με μικρές παύσεις μεταξύ των επιθέσεων του εφόσον ο swat εξακολουθεί να έχει πόντους ζωής και παραμένει να βρίσκεται εντός της εμβέλειας για επίθεση.

Η συνάρτηση OnDrawGizmosSelected() χρησιμεύει για την προβολή των εμβελειών μόνο κατά τη διάρκεια της σχεδίασης. Με κίτρινο χρώμα έχουμε ορίσει την εμβέλεια στην οποία το ζόμπι μπορεί να κυνηγήσει τον χαρακτήρα μας, ενώ με κόκκινο χρώμα είναι η ακτίνα στην οποία μπορεί να του επιτεθεί.

```
using UnityEngine;
using UnityEngine.AI;
public class Zombie : MonoBehaviour
{
    public float health = 100f;

    private Animator animator;

    public NavMeshAgent agent;

    public Transform player;

    public LayerMask whatIsGround, whatIsPlayer;

    public Vector3 walkPoint;
    bool walkPointSet;
    public float walkPointRange;

    //Attacking
    public float timeBetweenAttacks;
    bool alreadyAttacked;
    public GameObject projectile;

    //States
    public float sightRange, attackRange;
    public bool playerInSightRange, playerInAttackRange;
```

```
public ThirdPersonMovement swat;

void Start()
{
        animator = GetComponent<Animator>();
        player = GameObject.Find("Swat").transform;
        agent = GetComponent<NavMeshAgent>();
}

void Update()
{
        if(swat.swatHealth <= 0)
        {
                animator.SetBool("idle", true);
        }
        else
        {
                animator.SetBool("idle", false);
        }

        if (health <= 0f)
        {
                Die();
        }
        else if(swat.swatHealth <= 0)
        {
                //game over
        }
        else
        {
                animator.SetBool("zombieDeath", false);
                playerInSightRange = Physics.CheckSphere(transform.position,
sightRange, whatIsPlayer);
```

```
                playerInAttackRange = Physics.CheckSphere(transform.position,
attackRange, whatIsPlayer);

                if (!playerInSightRange && !playerInAttackRange)
                {
                        Patroling();
                }
                if (playerInSightRange && !playerInAttackRange)
                {
                        ChasePlayer();
                }

                if (playerInAttackRange && playerInSightRange)
                {
                        agent.SetDestination(transform.position);
                        AttackPlayer();
                }
        }
}

public void TakeDamage(float amount)
{
        health -= amount;
}

void Die()
{
        agent.SetDestination(transform.position);
        animator.SetBool("zombieDeath", true);
}
private void Patroling()
{
        agent.enabled = true;
        animator.SetBool("inSightRange", false);
```

```
            animator.SetBool("inAttackRange", false);
            agent.speed = 1.3f;
            if (!walkPointSet)
            {
                SearchWalkPoint();
            }
            if (walkPointSet)
            {
                agent.SetDestination(walkPoint);
            }

            Vector3 distanceToWalkPoint = transform.position - walkPoint;

            //Walkpoint reached
            if (distanceToWalkPoint.magnitude < 1f)
            {
                walkPointSet = false;
            }
        }
        private void SearchWalkPoint()
        {
            //Calculate random point in range
            float randomZ = Random.Range(-walkPointRange, walkPointRange);
            float randomX = Random.Range(-walkPointRange, walkPointRange);

            walkPoint = new Vector3(transform.position.x + randomX,
transform.position.y, transform.position.z + randomZ);

            if (Physics.Raycast(walkPoint, -transform.up, 2f, whatIsGround))
            {
                walkPointSet = true;
            }
        }
```

```
private void ChasePlayer()
{
        animator.SetBool("inSightRange", true);
        animator.SetBool("inAttackRange", false);
        agent.speed = 3.5f;
        if(alreadyAttacked == true && animator.GetBool("inAttackRange") == false)
        {
                agent.transform.position = transform.position;
        }
        else
        {
                agent.SetDestination(player.position);
                animator.SetBool("inSightRange", true);
        }
}

private void AttackPlayer()
{
        transform.LookAt(player);
        animator.SetBool("inSightRange", true);
        animator.SetBool("inAttackRange", true);
        agent.SetDestination(transform.position);


        if (!alreadyAttacked)
        {
                alreadyAttacked = true;
                Invoke(nameof(ResetAttack), timeBetweenAttacks);
        }
}

private void ResetAttack()
{
        alreadyAttacked = false;
```

```
    }


    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, attackRange);
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(transform.position, sightRange);
    }
}
```

Η συνάρτηση TakeDamage(float amount) υπολογίζει τη ζημιά που δέχεται το ζόμπι κάθε φορά που το πετυχαίνουμε με σφαίρες. Η τιμή της μεταβλητής amount δεν είναι πάντα η ίδια, διαφέρει ανάλογα με το σημείο που θα δεχτεί επίθεση το ζόμπι αλλά και με το όπλο που θα χρησιμοποιεί ο χαρακτήρας μας. Για παράδειγμα αν η σφαίρα το πετύχει στο κεφάλι χάνει διπλάσιους πόντους από αυτούς που θα έχανε αν το πετυχαίναμε στο υπόλοιπο σώμα. Με παρόμοιο τρόπο λειτουργούν και τα όπλα τα οποία έχουν διαφορετική δύναμη. Το παρακάτω κομμάτι κώδικα συνδέεται με το δεξί χέρι του αντιπάλου και χρησιμεύει στις εξής λειτουργίες. Να δέχεται ζημιά το ζόμπι σε αυτό το σημείο του σώματος αλλά όταν επιτίθεται με αυτό και έρχεται σε επαφή με τον χαρακτήρα μας να προκαλεί ζημιά στον χαρακτήρα μας.

```
using UnityEngine;

public class RightHandHit : MonoBehaviour
{
    public Zombie zombie;
    public ThirdPersonMovement swat;

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            DoDamageToPlayer();
        }
```

```
    }

    void DoDamageToPlayer()
    {
        swat.swatHealth -= 20f;
    }

    public void TakeDamage(float amount)
    {
        zombie.health -= amount;
    }
}
```

4.6 Πόντοι ζωής του swat

Σε αυτή τη συνάρτηση έχουμε τους πόντους ζωής του χαρακτήρα μας, κάθε φορά που ο αντίπαλος μας κάνει ζημιά οι πόντοι μειώνονται και ενεργοποιείται το animation ζημιάς. Όταν χαθούν όλοι οι πόντοι ζωής ενεργοποιείται το animation θανάτου, ο χαρακτήρας παραμένει ακίνητος στο έδαφος, εμφανίζεται στο κέντρο της οθόνης το μήνυμα “You Died” και το παιχνίδι τελειώνει.

```
    void takeDamage()
    {
        if (swatTempHealth > swatHealth)
        {
            animator.SetBool("takeDamage", true);
            swatTempHealth = swatHealth;
        }
        else
        {
            animator.SetBool("takeDamage", false);
        }

        if (swatHealth <= 0)
        {
            velocity.y = 0f;
```

```
                velocity.x = 0f;
                velocity.z = 0f;
                saveWeaponValue = 0;
                thirdPersonCamera.Priority = 100;
                animator.SetBool("death", true);
                animator.SetBool("takeDamage", false);
        }
        else
        {
                animator.SetBool("death", false);
        }
}
```

4.7 Δημιουργία του χαρακτήρα που πρέπει να διασώσουμε

Η διαδικασία για τη δημιουργία του αντικειμένου που θέλουμε να διασώσουμε είναι παρόμοια με αυτή που ακολουθήσαμε για τα ζόμπι. Ξεκινάμε με τη δημιουργία ενός 3D αντικειμένου τύπου Capsule και αλλάζουμε την ονομασία του σε Woman. Στη συνέχεια θα απενεργοποιήσουμε το Mesh Render για να μην είναι ορατή η κάψουλα και θα τοποθετήσουμε στη θέση της τα γραφικά που έχουμε επιλέξει.

Εικόνα 4.13 Woman

Στον κώδικα που έχουμε γράψει για το αντικείμενο woman γίνεται πάλι η χρήση του NavMeshAgent. Η γυναίκα που θέλουμε να διασώσουμε παραμένει ακίνητη στη τοποθεσία που βρίσκεται μέχρι να την εντοπίσουμε με τον χαρακτήρα μας. Για να μας ακολουθήσει μέχρι το σημείο τερματισμού πρέπει καθ' όλη τη διάρκεια να βρισκόμαστε εντός του οπτικού της

πεδίου. Αναλόγως με την κατάσταση που βρίσκεται γίνεται η αναπαραγωγή του κατάλληλου animation.

```
using UnityEngine;
using UnityEngine.AI;
public class WomanScript : MonoBehaviour
{
    public ThirdPersonMovement swat;

    public Animator animator;

    public NavMeshAgent agent;
    public Transform player;

    public Transform WomanLookingPoint;
    public LayerMask whatIsGround, whatIsPlayer;

    public float RunRange, WalkRange, StopRange;
    public bool playerInWalkRange, playerInRunRange, playerInStopRange;

    void Start()
    {
        animator = GetComponent<Animator>();
        WomanLookingPoint = GameObject.Find("LookingPoint").transform;
        agent = GetComponent<NavMeshAgent>();
    }

    void Update()
    {
        playerInStopRange = Physics.CheckSphere(transform.position, StopRange, whatIsPlayer);
        playerInWalkRange = Physics.CheckSphere(transform.position, WalkRange, whatIsPlayer);
        playerInRunRange = Physics.CheckSphere(transform.position, RunRange, whatIsPlayer);
```

```
        if(playerInRunRange && !playerInWalkRange)
        {
            Running();
        }
        if(playerInRunRange && playerInWalkRange && !playerInStopRange)
        {
            Walking();
        }
        if (playerInRunRange && playerInWalkRange && playerInStopRange)
        {
            NotMoving();
        }
        if (!playerInRunRange && !playerInWalkRange)
        {
            Lost();
        }
}
public void Lost()
{
        animator.SetBool("idle", true);
        animator.SetBool("inWalkRange", false);
        animator.SetBool("inRunRange", false);
        agent.SetDestination(transform.position);
}

public void NotMoving()
{
        animator.SetBool("idle", true);
        animator.SetBool("inWalkRange", false);
        animator.SetBool("inRunRange", false);
        agent.SetDestination(transform.position);
}
```

```
public void Walking()
{
        transform.LookAt(WomanLookingPoint);
        animator.SetBool("idle", false);
        animator.SetBool("inWalkRange", true);
        animator.SetBool("inRunRange", false);
        agent.speed = 3f;
        agent.SetDestination(player.position);
}
public void Running()
{
        transform.LookAt(WomanLookingPoint);
        animator.SetBool("idle", false);
        animator.SetBool("inWalkRange", false);
        animator.SetBool("inRunRange", true);
        agent.speed = 4.9f;
        agent.SetDestination(player.position);
}

private void OnDrawGizmosSelected()
{
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, WalkRange);
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(transform.position, RunRange);
        Gizmos.color = Color.green;
        Gizmos.DrawWireSphere(transform.position, StopRange);
}
```

4.8 Σημείο Τερματισμού

Για την επιτυχής ολοκλήρωση του παιχνιδιού πρέπει να εντοπίσουμε την τοποθεσία που βρίσκεται η γυναίκα που θέλουμε να διασώσουμε. Έπειτα, πρέπει να επιστρέψουμε μαζί με αυτή ζωντανοί στο ελικόπτερο όπου είναι και το σημείο έναρξης. Ο τρόπος για την

υλοποίηση του είναι να ορίσω μια εμβέλεια γύρο από το ελικόπτερο και να γίνεται συνεχής έλεγχος για τον αν βρίσκονται εντός αυτής της ακτίνας το αντικείμενο swat και το αντικείμενο woman. Αν είναι και οι δύο εντός έχουμε ολοκληρώσει τον στόχο του παιχνιδιού, εμφανίζεται το μήνυμα “Mission Accomplished” και επιστρέφουμε νικητές στο κύριο μενού.

Εικόνα 4.14 Mission Accomplished

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

public class RescuePoint : MonoBehaviour
{
        public float delayBeforeMainMenu = 3.0f;
        public NavMeshAgent agent;
        public GameObject MissionAccomplished;
        public Transform player;
        public LayerMask whatIsPlayer;
        public LayerMask whatIsWoman;
        public float rescueRange;
        public bool swatInRange;
        public bool womanInRange;
        void Update()
        {
                swatInRange = Physics.CheckSphere(transform.position, rescueRange,
whatIsPlayer);
                womanInRange = Physics.CheckSphere(transform.position, rescueRange,
whatIsWoman);
```

```
            if(swatInRange == true && womanInRange == true)
            {
                MissionAccomplished.SetActive(true);
                StartCoroutine(ReturnToMainMenuCoroutine());
                MissionAccomplished.SetActive(false);
            }
            else
            {
                MissionAccomplished.SetActive(false);
            }
        }

        private IEnumerator ReturnToMainMenuCoroutine()
        {
            yield return new WaitForSeconds(delayBeforeMainMenu);
            SceneManager.LoadScene("MainMenu");
        }

        private void OnDrawGizmosSelected()
        {
            Gizmos.color = Color.green;
            Gizmos.DrawWireSphere(transform.position, rescueRange);
        }
}
```

4.9 UI(User Interface)

Το τελευταίο βήμα είναι να εξετάσουμε τον τρόπου που σχεδιάστηκε το UI του παιχνιδιού οπού περιλαμβάνει το κύριο μενού και το μενού παύση.

Το κύριο μενού λειτουργεί ως το σημείο εκκίνησης του παιχνιδιού. Για το background του μενού επέλεξα ένα στιγμιότυπο οθόνης μέσα από το παιχνίδι. Σε κάθε κουμπί έχουμε προσθέσει να εμφανίζεται μια σκίαση όταν περνάει ο κέρσορας πάνω από αυτό ώστε να γίνεται οπτικά αντιληπτή η επιλογή του. Για κάθε επιλογή που βρίσκεται στο μενού έχει γραφτεί και ο ανάλογος κώδικας.

Οι επιλογές του μενού είναι οι εξής:

Play: Έναρξη του παιχνιδιού.

Options: Ρυθμίσεις του παιχνιδιού.

Quit: Έξοδος από το παιχνίδι.

Εικόνα 4.15 Main Menu

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class MainMenu : MonoBehaviour
{
	public void PlayGame()
	{
		SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
	}

	public void GoToSettingsMenu()
	{
		SceneManager.LoadScene("SettingsMenu");
	}

	public void GoToMainMenu()
	{
		SceneManager.LoadScene("MainMenu");
```

```
    }

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

Κατά τη διάρκεια του παιχνιδιού αν ο χρήστης πατήσει το κουμπί Escape(ESC) στο πληκτρολόγιο, εμφανίζεται το μενού παύσης. Οι επιλογές του μενού είναι:

Resume: Συνέχεια του παιχνιδιού από το σημείο που το σταματήσαμε.

Back to Main Menu: Επιστροφή στο κύριο μενού.

Quit: Έξοδος από το παιχνίδι.

Όπως και στο κύριο μενού κάνουμε ορατή την κάθε επιλογή όταν ο κέρσορας περνάει από πάνω. Στην πορεία αναλύεται ο κώδικας για την υλοποίηση της λειτουργίας κάθε επιλογής.

Εικόνα 4.16 Pause Menu

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public static bool GameIsPaused = false;
```

```
public GameObject pauseMenuUI;

void Update()
{
	if (Input.GetKeyDown(KeyCode.Escape))
	{
		if (GameIsPaused)
		{
			Resume();
		}
		else
		{
			Pause();
		}
	}
}

public void Resume()
{
	pauseMenuUI.SetActive(false);
	Time.timeScale = 1f;
	GameIsPaused = false;
}
void Pause()
{
	pauseMenuUI.SetActive(true);
	Time.timeScale = 0f;
	GameIsPaused = true;
}

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

```
            SceneManager.LoadScene("MainMenu");
        }

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

ΚΕΦΑΛΑΙΟ 5: ΣΥΜΠΕΡΑΣΜΑΤΑ

Στο πλαίσιο της πτυχιακής εργασίας μου ανέπτυξα ένα παιχνίδι τρίτου προσώπου με ζόμπι στην πλατφόρμα Unity. Η διαδικασία ανάπτυξης αυτού του παιχνιδιού με έφερε αντιμέτωπο με πολλές προκλήσεις τις οποίες κατάφερα να ξεπεράσω με δημιουργικότητα και αφοσίωση. Μέσα από αυτή την εμπειρία εξέλιξα τις γνώσεις μου στον προγραμματισμό και με έκανε να καταλάβω πόσο συμπαντική είναι η καλή διαχείριση του χρόνου για ένα project. Τέλος, αναγνωρίζω τις δυνατότητες που μου ανοίγει αυτή η εμπειρία στον κόσμο του game development και είμαι πρόθυμος να εξερευνήσω περαιτέρω αυτόν τον τομέα στην επαγγελματική μου πορεία.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία:

Αναγνώστου Κώστας, 2000, ΒΙΝΤΕΟΠΑΙΧΝΙΔΙΑ Βιομηχανία και Ανάπτυξη.

Δικτυογραφία:

https://www.pcsteps.gr

https://www.gamesindustry.biz/what-is-the-best-game-engine-is-cryengine-the-right-game-engine-for-you

https://learn.microsoft.com/en-us/archive/msdn-magazine/2014/august/unity-developing-your-first-game-with-unity-and-csharp

https://ventionteams.com/blog/unity-pros-and-cons

https://medium.com/@xceltecweb/top-advantages-of-using-unity-for-game-development-bd64b6a3004

https://resources.unity.com/unitenow

https://docs.unity3d.com/Manual/index.html

https://www.youtube.com/watch?v=mkErt53EEFY&t=1s

https://stackskills.com/courses/enrolled/590152

https://www.youtube.com/watch?v=ddy12WHqt-M

https://www.youtube.com/watch?v=U0dlWhB_e0E

https://www.youtube.com/watch?v=0QA2O7juuWQ

https://www.youtube.com/watch?v=_QajrabyTJc

https://www.youtube.com/watch?v=537B1kJp9YQ

https://www.youtube.com/watch?v=ZVrfNivw7PY

https://www.youtube.com/watch?v=FbM4CkqtOuA

https://www.youtube.com/watch?v=Dn_BUIVdAPg

https://devassets.com/

https://www.youtube.com/watch?v=THnivyG0Mvo&t=97s

https://www.youtube.com/watch?v=HZuazvqaYkU

https://www.youtube.com/watch?v=THnivyG0Mvo&t=280s

https://www.youtube.com/watch?v=u5rmP_kAJ2o

https://www.youtube.com/watch?v=UjkSFoLxesw

https://forum.unity.com/threads/setdestination-can-only-be-called-on-an-active-agent-that-is-on-a-navmesh.599221/

https://www.youtube.com/watch?v=SV1eE1hvuqY

https://www.youtube.com/watch?v=JJUnufMLUp0

https://www.youtube.com/watch?v=-GWjA6dixV4

Πτυχιακές εργασίες:

Κωνσταντίνος Ανδρεαδάκης Ιωάννης Εξιλζές. (2015). Ανάπτυξη Παιχνιδιού με το Unity3D σε C#
http://okeanis.lib2.uniwa.gr/xmlui/bitstream/handle/123456789/3479/%CE%A0%CF%84%CF%85%CF%87%CE%B9%CE%B1%CE%BA%CE%AE.pdf?sequence=1&isAllowed=y

Μανωλάκης Κυριάκος . (2017). “Σχεδιασμός και Ανάπτυξη top-down video game, με χρήση της Unity game engine και της γλώσσας C#”.
http://repository.teiwest.gr/xmlui/bitstream/handle/123456789/5724/CIED%20%CE%9C%CE%B1%CE%BD%CF%89%CE%BB%CE%AC%CE%BA%CE%B7%CF%82%20%CE%9A%CF%85%CF%81%CE%B9%CE%AC%CE%BA%CE%BF%CF%82.pdf?sequence=1&isAllowed=y&fbclid=IwAR09s7d8gVS28kaicwRO9fX3EzVPRUaQjDff7BbwJvWRhzQZY0XAHP_Jbcg