Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Τίτλος:

"Παιχνίδι τύπου Survival σε Unity"

Ματαλλιωτάκης Μιχάλης (ΑΜ:3218)

Επιβλέπων Καθηγητής: κ. Παχουλάκης Ιωάννης

Ηράκλειο, Ιανουάριος 2016

Abstract

The present document contains information on the production of a survival-type computer game. Blender was used to create the environment for each game stage(floor,crates, barrels etc.) as well as the texture UV maps for each object using the texture paint tools provided by Blender. The actual 3D rigged models were created using MakeHuman. Each rigged model was imported to Blender and animations were created and recorded where necessary. In addition, the humanoid models were appropriately textured. Those models were subsequently imported into the Unity Game Engine and each object was programmed. It is further discussed how each program was used to develop game functionality, ending with a presentation on the possibilities for future development and extensions.

Σύνοψη

Το ακόλουθο έγγραφο περιγράφει τη διαδικασία που ακολουθήθηκε για την ανάπτυξη ενός παιχνιδιού τύπου «survival». Στα πλαίσια της εργασίας δημιουργήθηκαν τρισδιάστατα μοντέλα ανθρωπόμορφων χαρακτήρων τόσο για την ομάδα των παιχτών όσο και για την ομάδα των εχθρών με την χρήση των προγραμμάτων MakeHuman και Blender. Επιπλέον, το Blender χρησιμοποιήθηκε στη δημιουργία της πίστας, αντικειμένων στο χώρο, αντικειμένων των παικτών, καθώς και στη δημιουργία υφών (textures).Αρχικά δημιουργήθηκαν οι παίκτες και οι εχθροί με τη βοήθεια του προγράμματος MakeHuman.Στην μοντέλα που προέκυψαν εισάχθηκαν στο Blender για τη δημιουργία των απαραίτητων animation sequences και ακολούθησε η δημιουργία υφών.Στη συνέχεια δημιουργήθηκαν οι πίστες και η λειτουργικότητα του παιχνιδιού (gamelogic) στη Unity Game Engine. Τέλος, γίνονται εκτενείς αναφορές στη χρήση του κάθε προγράμματος για το συγκεκριμένο παιχνίδι και παρουσιάζονται προοπτικές εξέλιξης.

Περιεχόμενα

Πίνακας Εικόνων

Κεφάλαιο 1

Εισαγωγή

1.1. Περίληψη

Το βιντεοπαιχνίδι αυτό που θα περιγραφεί αναλυτικότερα πρόκειται για παιχνίδι με 3ου προσώπου οπού ο χρήστης/παίκτης μπορεί να ελέγξει μια ομάδα παικτών (συγκεκριμένα 2 παίκτες).Κάθε παίκτης που είναι μια οποιαδήποτε στιγμή ο "αρχηγός" της ομάδας(μέσω μηχανισμού εναλλαγής ενεργού παίκτη) καθοδηγεί την ομάδα μέσα στην κάθε πίστα και ο σκοπός είναι να επιβιώσουν μέχρι το τέλος κάθε πίστας από τους εχθρούς που χρειάζεται να αντιμετωπίσουν. Ο αρχηγός της ομάδας έχει την δυνατότητα να ανασυντάξει (regroup) την ομάδα, καθώς και να την καθοδηγεί σε επίθεση σε εχθρό που έχει ξεκινήσει να επιτίθεται (assist).

Παρόμοιοι μηχανισμοί υπάρχουν σε ήδη υπάρχοντα παιχνίδια κατηγορίας (action, survival κλπ.).Μερικά παραδείγματα από τέτοιου είδους παιχνίδια είναι:

Figure 1.1-1:Dragon Age : Inquisition

Το DragonAge : Inquisition πρόκειται για ένα τίτλο της εταιρείας βιντεοπαιχνιδιών BioWare όπου και αποτελεί το 3ο παιχνίδι της σειράς DragonAge, όπου αντίστοιχα και το DragonAge 1 και 2 είχαν αντίστοιχους μηχανισμούς διαχείρισης της ομάδας των παικτών όπως φαίνεται στην παραπάνω φωτογραφία. Το DragonAge: Inquisition δημιουργήθηκε το 2014,καθώς και οι προηγούμενοι τίτλοι της σειράς αυτής μερικά χρόνια νωρίτερα. (DragonAge - 2009,Dragon Age-2011) Όλοι οι παραπάνω τίτλοι της σειράς DragonAge εκδόθηκαν από την εταιρεία βιντεοπαιχνιδιών ElectronicArts (EA).

Figure 1.1-2: Freedom Fighters

Το Freedom Fighters είναι ένα άλλο παράδειγμα παιχνιδιού που υποστηρίζει τη διαχείριση ομάδας παικτών όπως φαίνεται στην παραπάνω φωτογραφία. Το εν λόγω βιντεοπαιχνίδι είναι τίτλος της εταιρείας IO Interactive, δημιουργήθηκε το 2003 και επίσης εκδόθηκε από την εταιρεία ElectronicArts (EA).

Τα παραπάνω παραδείγματα έχουν σαν κοινό στοιχείο την δυναμική διαχείριση ομάδας παικτών και αποτέλεσαν σημαντική επιρροή για την εργασία αυτή. Η στρατηγική επιλογής χαρακτήρων / παικτών συναντάται ακόμα και στα πιο πρόσφατα βιντεοπαιχνίδια (2013-2016).Φυσικά στα βιντεοπαιχνίδια αυτά η ομάδα παικτών έχει περισσότερη εφαρμογή στο θέμα επιλογής παίκτη στο μεγαλύτερο μέρος αυτών των παιχνιδιών, καθώς η δυναμική της συνεργασίας συναντάται μόνο σε κάποιες αποστολές για λίγο χρονικό διάστημα και παιχνίδια σαν αυτά είναι το "Grand Theft Auto V" και "Assassin's Creed : Syndicate".

Η συγκεκριμένη εργασία αποσκοπεί στη δημιουργία ενός μικρού μήκους βιντεοπαιχνιδιού που θα εμπεριέχει κάποιες τρισδιάστατες πίστες, μια ομάδα παικτών που ο χρήστης θα έχει την δυνατότητα να διαχειριστεί, καθώς και εχθρούς που θα λειτουργούν υπό ένα είδος τεχνητής νοημοσύνης βασισμένης στην λογική καταστάσεων. Η ομάδα θα κινείται προς διάφορα σημεία σε μια πίστα και κάθε πίστα έχει ένα σκοπό που πρέπει να επιτευχθεί.

Η συγκεκριμένη εφαρμογή αποτελεί ένα demo ενός ολοκληρωμένου παιχνιδιού και δεν πρόκειται για ολοκληρωμένο παιχνίδι κάποιας εταιρείας. Έχει καθαρά επιμορφωτικό χαρακτήρα για προγραμματιστές παιχνιδιών που θέλουν να ασχοληθούν σε αρχάριο επίπεδο με gameplay (μηχανισμοί λειτουργικότητας στο ίδιο το παιχνίδι, αλλά και στην μεριά του χρήστη) και A.I. (τεχνητή νοημοσύνη) programming.Η εργασία αυτή επιπλέον περιέχει στοιχεία διεπαφής με σκοπό την ενημέρωση του χρήστη κατά τη διάρκεια του παιχνιδιού.

Η εν λόγω εφαρμογή δεν περιορίζεται σε κάποια θεματολογία και ύφος μιας και θεωρητικά πρόκειται για κάποιου είδους Cyberpunk βιντεοπαιχνίδι οπού σε ένα τέτοιο genre (είδος) η επιλογή για το περιβάλλον μέχρι και τους χαρακτήρες είναι εντελώς ελεύθερη καθώς μπορεί να συνδυάσει παραπάνω από μία εποχές και στυλ. Το είδος του

βιντεοπαιχνιδιού είναι κατηγορίας shooter που σημαίνει πως προορίζεται για μάχη εξ αποστάσεως και όχι melee με βάση τον προγραμματισμό που έχει γίνει.

1.2. Σκοπός και Στόχοι Εργασίας

Οι στόχοι της εργασίας είναι οι εξής:

(I)Δημιουργία ανθρωποειδών μοντέλων για την ομάδα των παικτών και για τους εχθρούς.

1). Χρήση του MakeHuman για δημιουργία μοντέλων.

2). Χρήση του Blender για επιπλέον αντικείμενα πάνω στους χαρακτήρες όπως (όπλα, διακοσμητικά).

3). Χρήση του Blender για δημιουργία υφών όπου απαιτούνταν.

4). Χρήση του Blender για δημιουργία animation όπου απαιτούνταν.

(II)Δημιουργία αντικειμένων που συντελούν τα επίπεδα που πρέπει να περάσει η ομάδα.

1). Χρήση του Blender για δημιουργία γραφικών στις πίστες.

(III)Προγραμματισμός μηχανισμών στην ομάδα των παικτών και στους εχθρούς.

1). Προγραμματισμός συμπεριφοράς μηχανισμών ομαδικής συμπεριφοράς

2). Προγραμματισμός συμπεριφοράς της τεχνητής νοημοσύνης των εχθρών

(IV)Προγραμματισμός μηχανισμών των επιπέδων.

1). Προγραμματισμός μηχανισμών των επιπέδων (checkpoints, triggers).

(V)Προγραμματισμός της Διεπαφής.

1). Προγραμματισμός της διεπαφής (μπάρα ζωής(παίκτες/εχθροί),κατάσταση παύσης, κατάσταση νίκης, κατάσταση ήττας, κατάσταση αρχής παιχνιδιού)

(VI)Προγραμματισμός στην κίνηση του παίκτη.

1). Προγραμματισμός κίνησης του ενεργού παίκτη.

(VII) Προγραμματισμός σε ρουτίνες που ελαχιστοποιούν σε κάποια σημεία τον φόρτο που προκαλεί η φυσική της Unity.

1). Προγραμματισμός στο κάλεσμα της φυσικής στο θέμα πυροβολισμού της με laser.

2). Προγραμματισμός σε ρουτίνα ελέγχου ύψους του ενεργού παίκτη για καθορισμό πτώσης.

1.3. Δομή Εργασίας

Το παρόν έγγραφο περιέχει την Εισαγωγή (1ο Κεφάλαιο) καθώς και άλλα 3 κεφάλαια. Το 2ο κεφάλαιο περιλαμβάνει τη μεθοδολογία που ακολουθήθηκε για την ανάπτυξη του παιχνιδιού, ενώ το 3ο κεφάλαιο παρουσιάζει την ανάπτυξη του παιχνιδιού. Τέλος αναλύουμε τα αποτελέσματα της εργασίας αυτής και αν το αποτέλεσμα ήταν μέσα στις προσδοκίες της εργασίας αυτής, καθώς και τα προβλήματα που παρουσιάστηκαν και επιλύθηκαν κατά την διάρκειά της. Στο τέλος του εγγράφου εμπεριέχονται γενικές πληροφορίες για το κάθε πρόγραμμα που χρησιμοποιήθηκε στα πλαίσια της πτυχιακής εργασίας.

1.4. Επεξήγηση Βασικών Όρων

Game Engine

Πρόκειται για ένα πρόγραμμα που αποτελείται από πολλά άλλα ενσωματωμένα προγράμματα και το κάθε ένα από αυτά είναι υπεύθυνο για κάποια συγκεκριμένη λειτουργία. Πρόκειται καθαρά για πρόγραμμα που χρησιμοποιούν όλοι οι προγραμματιστές παιχνιδιών για την κατασκευή κάποιου παιχνιδιού. Όλες οι game engines περιέχουν προγράμματα που καλύπτουν τις βασικές ιδιότητες των παιχνιδιών, δηλαδή αποτύπωση γραφικών (rendering), animation tools (εργαλεία που βοηθάνε στη διαχείριση των animations), ήχο, υφές (textures), φωτισμό, δικτύωση (networking / multiplayer) , φυσική, διαχείριση μνήμης, διεπαφές (UI), τεχνητή νοημοσύνη και το βασικότερο από όλα μία σκηνή όπου όλα αυτά τα στοιχεία μαζεύονται αναλόγως τις ανάγκες και στο τελικό τους αποτέλεσμα αποτελούν το εκάστοτε βιντεοπαιχνίδι. Μία game engine μπορεί να είναι κατασκευασμένη για 2D(δισδιάστατα) γραφικά, για 3D(τρισδιάστατα) γραφικά ή και να αποτελεί συνδυασμό και των δύο. Παράδειγμα 2D Game Engine είναι η Cocos2D, ενώ για 3D έχουμε τις Unreal Engine και Cry Engine, ενώ παράδειγμα συνδυασμού των 2 ειδών αποτελεί η Unity που περιέχει εργαλεία και για τους 2 τύπους γραφικών. Οι Game Engines έχουν την ιδιότητα να εξάγουν το συνολικό αποτέλεσμα τους σε διάφορες μορφές (κονσόλες βιντεοπαιχνιδιών, κινητά τηλέφωνα και tablets, web / διαδικτυακά παιχνίδια, desktop / laptop).

Mesh

Mesh ή αλλιώς πλέγμα αποκαλούμε την γεωμετρία ενός 2D / 3D αντικειμένου η οποία περιέχει πληροφορίες για την θέση των vertices (σημεία του mesh), faces (επιφάνεια πάνω στο mesh που χρειάζεται 3 ή παραπάνω vertices για να σχηματιστεί). Χρησιμοποιείται από τις υφές (textures), τους χάρτες φωτισμού επιφάνειας (normal maps) κυρίως και αυτά τα 2 αποτελούν τον λόγο της φαινομενικής λεπτομέρειας πάνω σε ένα αντικείμενο.

Animation

Το animation αποτελεί μια σειρά κινήσεων σε ένα χρονικό διάστημα με τη βοήθεια σημείων αναφοράς (keyframes) και να αλλάξει θέσεις, γωνίες ή μέγεθος σε ένα ή παραπάνω αντικείμενα. Το clip που παράγεται στο τέλος της παραγωγής του περιέχει όλες τις πληροφορίες που αφορούν θέση, γωνία, μέγεθος για κάθε ορισμένη χρονική στιγμή, αλλά και για τα διαστήματα που μεσολαβούν μεταξύ των keyframes. Το animation όταν παραχθεί είναι έτοιμο να εισαχθεί στη Game Engine και να χρησιμοποιηθεί για να καθορίσει κάποια κίνηση που επιθυμούμε να περιλαμβάνει το αντικείμενο στο οποίο αναφέρεται το animation clip.

Shaders

Ο shader είναι αλγόριθμος/οι που εφαρμόζεται πάνω σε υλικό (material) το οποίο περιέχει μία ή παραπάνω υφές (textures) και ανάλογα τις ιδιότητες του εκάστοτε shader αλλάζει τον τρόπο που επιδρά μια πηγή φωτός πάνω σε μια επιφάνεια γραφικού. Μπορεί να κάνει ένα αντικείμενο να φαίνεται πιο παιδικό (toon), μπορεί να το κάνει να φαίνεται σαν ύφασμα ή σαν μέταλλο. Στην γενική του μορφή ο shader μπορεί να εφαρμοστεί σε οποιαδήποτε μορφή γραφικού και να δώσει ένα πιο ρεαλιστικό ή διαφορετικό αποτέλεσμα.

Artificial Intelligence

Artificial intelligence ή αλλιώς τεχνητή νοημοσύνη είναι μια προγραμματιστική τεχνική για να αποδώσουμε συμπεριφορά σε αντικείμενα μη ελεγχόμενα από τον ίδιο τον παίκτη κατά την διάρκεια του παιχνιδιού. Επιθυμούμε συνήθως να περιέχει μια ρουτίνα εντολών και κανόνων ως προς τον τρόπο λειτουργίας του και αναλόγως τις ανάγκες. Η τεχνητή νοημοσύνη στα games είναι ευρείας χρήσης και μπορεί να δεχτεί ερεθίσματα από απλές κινήσεις και ήχους μέχρι και μαθησιακή συμπεριφορά σε σχέση με το περιβάλλον.

Material

Το υλικό ή αλλιώς material περιλαμβάνει 2 κύρια χαρακτηριστικά: την υφή και τον shader.Η υφή χρησιμοποιείται για να τυλίξει (wrap) το αντικείμενο στο οποίο αναφέρεται και ο shader για τον τρόπο που θα επιδρά πάνω του το φως. Το material αποτελεί ένα υποχρεωτικό στοιχείο για τα αντικείμενα μέσα σε μια σκηνή ενός παιχνιδιού, καθώς ο renderer της game engine αναλαμβάνει να αναγνωρίσει το material και με βάση τις ρυθμίσεις που έχει να αποδώσει το επιθυμητό αποτέλεσμα κατά την διάρκεια απεικόνισης του.

Collider

Ο collider είναι βασικό συστατικό των videogames και αναφέρεται στην ιδιότητα ενός αντικειμένου να συγκρούεται με άλλα αντικείμενα που φέρουν την ίδια ιδιότητα. Υπάρχουν 4 βασικά είδη collider: Sphere Collider, Box Collider, Capsule Collider και Mesh Collider.Όπως καταλαβαίνει κάποιος διαβάζοντας τα ονόματα οι 3 πρώτοι colliders υποδεικνύουν την μορφή που θα έχουν αφού εφαρμοστούν σε κάποιο αντικείμενο (σφαίρα, κύβος, κάψουλα), ενώ ο 4ος (Mesh Collider) υλοποιεί την γεωμετρία του αντικείμενου στο οποίο υπάγεται.Αποτελεί την καλύτερη λύση σε θέμα αξιοπιστίας, αλλά είναι πιο ακριβός υπολογιστικά καθώς χρησιμοποιεί πολύ περισσότερα σημεία για να εντοπίζει και να επιλύει συγκρούσεις. Η χρήση του αποφεύγεται στην πλειοψηφία των περιπτώσεων καθώς οι περιπτώσεις χρήσης του είναι ικανοποιητικές σε στατικά αντικείμενα που δεν επιβαρύνουν τον υπολογιστή.

Texture Map

Υπάρχουν περιπτώσεις που θέλουμε να βάψουμε (texture paint) ένα αντικείμενο με διάφορους τρόπους και να κρατήσουμε το αποτέλεσμα σε κάποια εικόνα που θα μπορούμε να χρησιμοποιήσουμε αργότερα για το αντικείμενο μας. Η δουλειά του βαψίματος γίνεται σε ένα σχεδιαστικό πρόγραμμα όπως τα Blender, Maya, ή 3d StudioMax . Αρχικά παίρνουμε το αρχικό χρώμα / υφή του αντικειμένου και κάνουμε unwrap (το ξετυλίγουμε). Κατόπιν επιλέγουμε να βάψουμε τον αντικείμενο μας με χρώμα ή με κάποια άλλη υφή σημειακά. Κατά την διάρκεια αυτή δημιουργείται ένα UV Map που αναγνωρίζει τις συντεταγμένες(θέση) του κάθε σημείου που έγινε unwrap. Όσο βάφουμε παρατηρούμε αλλαγές και στο δημιουργημένο UV Map. Όταν η διαδικασία του βαψίματος ολοκληρωθεί επιλέγουμε να σώσουμε το UV Map σαν εικόνα. Αυτή η εικόνα μπορεί να χρησιμοποιηθεί οποιαδήποτε στιγμή στο αντικείμενο το οποίο αφορά και να δώσει το αποτέλεσμα που είχαμε μετά το τέλος του βαψίματος. Αυτό είναι το texture map ή και απλώς texture.

Κεφάλαιο 2

Μεθοδολογία

2.1. Μέθοδος Ανάλυσης & Ανάπτυξης Πτυχιακής Εργασίας

Η δημιουργία του τρισδιάστατου παιχνιδιού έγινε από το μηδέν. Σχετικά με τα animations: κάποια υπήρχαν έτοιμα από τη Unity και κάποια άλλα ήταν αναγκαίο να δημιουργηθούν εκ του μηδενός. Πιο αναλυτικά τα μοντέλα παικτών και εχθρών ρυθμίστηκαν μέσα από το πρόγραμμα MakeHuman εφόσον η εργασία δεν έχει κύριο σκοπό την γραφιστική πλευρά στην δημιουργία των βιντεοπαιχνιδιών και εξήχθηκαν (export) για να μπουν σε πρόγραμμα παραμετροποίησης (authoring tool). Έπειτα αποφασίσαμε για λόγους ευκολίας και καλύτερης συνεργασίας μεταξύ των προγραμμάτων που υπάρχουν να χρησιμοποιήσουμε το Blender για τις βελτιώσεις γραφικών, δημιουργία γραφικών για τις πίστες και animation σε χαρακτήρες. Η Unity κρίθηκε ως καταλληλότερη game engine για τις 2 προσφερόμενες επιλογές σε γλώσσες προγραμματισμού (C# και Javascript).

2.2. Απαιτούμενο λογισμικό για την δημιουργία παιχνιδιών

Για την δημιουργία ενός βιντεοπαιχνιδιού απαιτούνται πράγματα όπως γραφικά, animations τα οποία μπορούμε να δημιουργήσουμε ή να πάρουμε έτοιμα αναλόγως τον στόχο της εκάστοτε εργασίας, αλλά φυσικά απαιτείται και η επιλογή κατάλληλης game engine βασισμένης στις ανάγκες μας και τους στόχους μας ώστε να αποφευχθούν σχεδιαστικά προβλήματα κατά την διάρκεια της υλοποίησης.

2.2.1. Προγράμματα δημιουργίας γραφικών και διαχείρισης τους

Ας εξετάσουμε τα σχεδιαστικά προγράμματα γραφικών και animations που εξετάστηκαν για την υλοποίηση αυτής της εργασίας.

1) **3D Studio Max**

Το 3D StudioMax αποτελεί προϊόν της εταιρείας Autodesk και θεωρείται ένα καταξιωμένο σχεδιαστικό πρόγραμμα γραφικών και animation στην αγορά. Όπως φαίνεται στην παρακάτω εικόνα βλέπουμε το αρχικό περιβάλλον το όποιο περιλαμβάνει διάφορα menu για τις διαφορετικές λειτουργίες που παρέχει. Λόγω προβλημάτων και έλλειψης ικανοποιητικών εκπαιδευτικών υλικών στο internet η χρήση του στην εργασία αυτή κρίθηκε χρονικά ακατάλληλη.

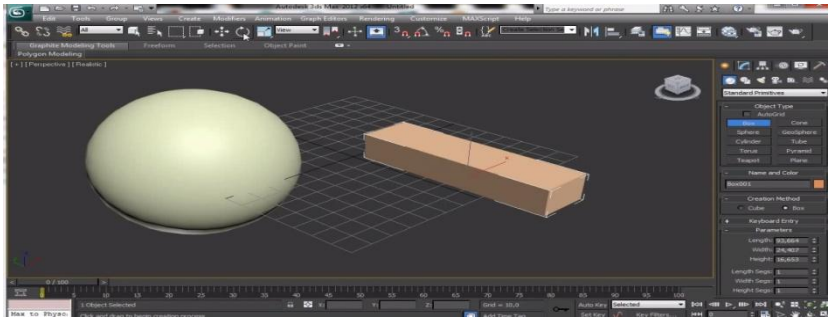

Figure 2.2.1-1: 3D Studio Max

2) **Blender**

Το Blender αποτελεί ένα πολυμήχανο εργαλείο μιας και πέρα από εργαλείο δημιουργίας γραφικών και animation είναι και μια αρκετά δημοφιλής game engine. Είναι αρκετά εύχρηστο σε σχέση με άλλα και πολύ πιο εύκολο στην μάθησή του μιας και υπάρχει πληθώρα από υλικό είτε στο επίσημο website του, αλλά και στο youtube. Η απόφαση επιλογής του πάρθηκε αμέσως λόγω της φιλικότητας με τον χρήστη και της συνεργασίας του με το MakeHuman, καθώς έχει μέσα πρόσθετο λογισμικό που αναγνωρίζει αντικείμενα που έχουν προέλθει από εκεί.

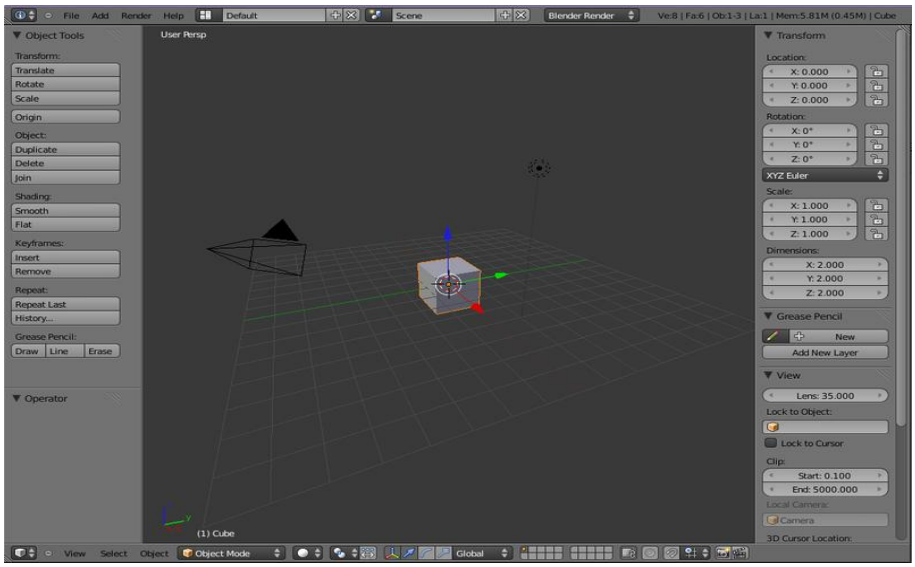

Figure 2.2.1-2:Blender

3) **MakeHuman**

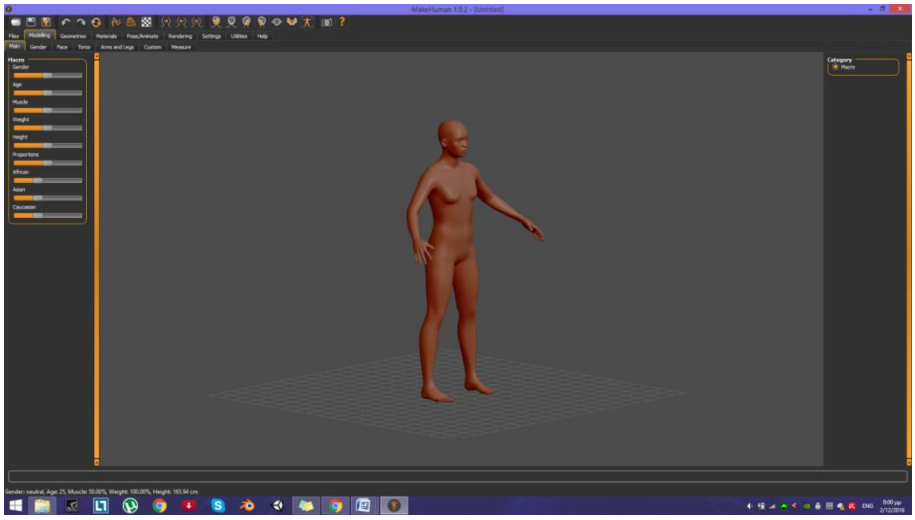

Figure 2.2.1-3:MakeHuman

Το MakeHuman είναι το λογισμικό που ανέλαβε την διαδικασία μοντελοποίησης και rigging των ανθρωποειδών (humanoids) μοντέλων, καθώς και κάποιες παρεχόμενες δυνατότητες σε ρούχα, μαλλιά, αυξομείωση σωματικών χαρακτηριστικών, αλλαγή χρωματικών χαρακτηριστικών και καθορισμός format εξαγωγής αρχείου για περαιτέρω βελτιώσεις. Το MakeHuman είναι προϊόν μιας μικρής κοινότητας που το υποστηρίζει αδιάκοπα, είναι πρόγραμμα ανοιχτού λογισμικού (opensource) και δεν απαιτείται ειδική άδεια για εμπορική χρήση.

2.2.2. Επιλογή Game Engine

Η επιλογή της game engine ήταν και θα είναι πάντα το σημαντικότερο στοιχείο καθώς αποτελεί το σημείο σταθμό οπού όλα τα τμήματα σε ένα βιντεοπαιχνίδι καταλήγουν εκεί και γίνονται ένα ενιαίο αποτέλεσμα. Ας εξετάσουμε τις επιλογές από τις οποίες έπρεπε να επιλέξουμε.

1) **Unreal Engine 4**

Η Unreal Engine 4 αποτελεί την καλύτερη game engine στα πλαίσια του προγραμματισμού των videogames και συγκεκριμένα σε videogames μεγάλης κλίμακας οπού συνήθως απαιτείται μεγάλος αριθμός ατόμων για την επίτευξη τους. Η Unreal Engine 4 χρησιμοποιεί την γλώσσα προγραμματισμού C++ λόγω της ελευθερίας χρήσης της και των πολλών δυνατοτήτων που προσφέρει. Η πολυπλοκότητα της και η έλλειψη μαθησιακού υλικού την καθιστούν αρκετά δύσκολη στην κατασκευή έστω και μικρού μεγέθους βιντεοπαιχνιδιού και έτσι αποθαρρύνθηκε η χρήση της στην εργασία αυτή.

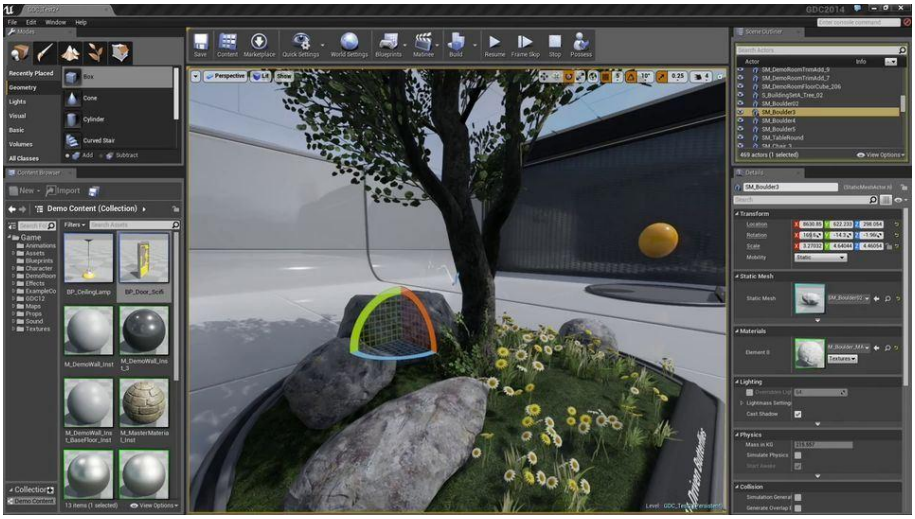

Figure 2.2.2-1:Unreal Engine 4

2) **Unity Game Engine**

Η Unity Game Engine αποτελεί την καλύτερη λύση για όσους ενδιαφέρονται να μάθουν να προγραμματίζουν videogames σε αρκετά αρχάριο έως και επαγγελματικό βαθμό. Καλύπτει μεγάλο φάσμα διαφορετικών αναγκών στα βιντεοπαιχνίδια, καθώς είναι ευέλικτη στις εκδόσεις και υποστηρίζεται σε πάρα πολλές διαφορετικές συσκευές. Τα επίπεδα λεπτομέρειας που επιτυγχάνει είναι αρκετά ικανοποιητικά αλλά σε καμία περίπτωση δεν είναι ισάξια με την Unreal Engine 4. Πρόκειται όμως για την καλύτερη μαθησιακή λύση και αυτός είναι ο λόγος που επιλέχθηκε, όπως επίσης και η διευκόλυνση που δίνει στην επιλογή γλώσσας προγραμματισμού μεταξύ C# και Javascript.

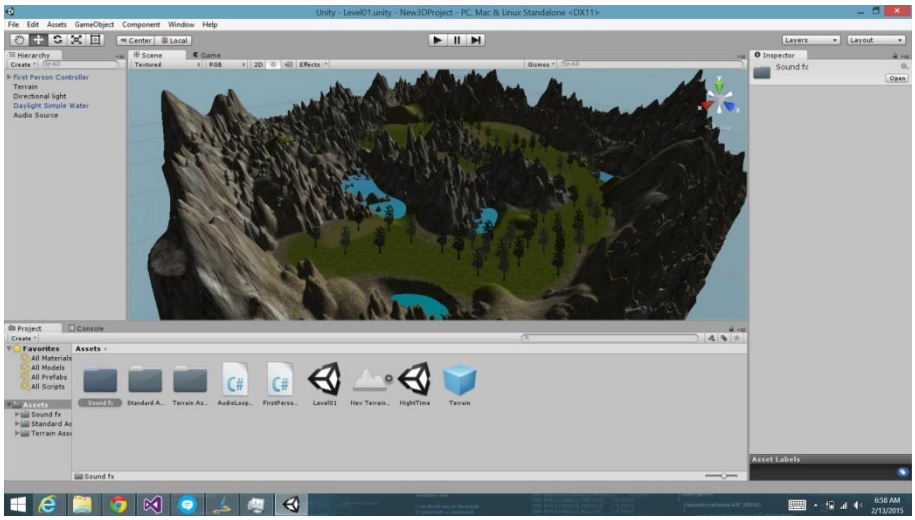

Figure 2.2.2-2:Unity Game Engine

2.3. Ροή Εργασίας

Η ροή εργασίας περιγράφεται στο παρακάτω γράφημα :

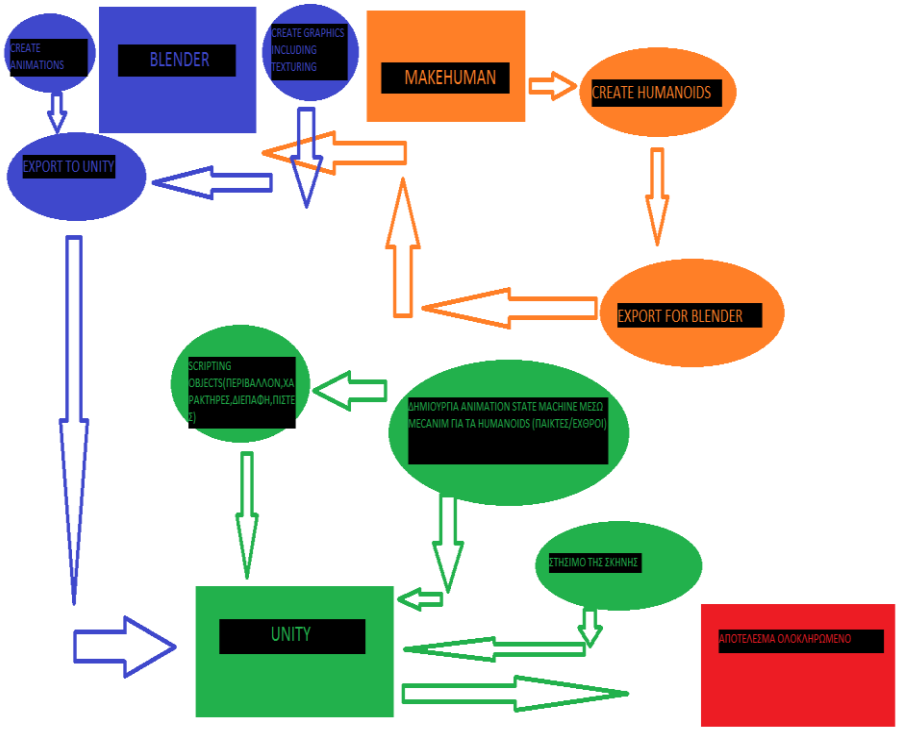

Figure 2.3-1: Ροή Εργασίας

2.3.1 Περιγραφή ροής εργασίας

Όπως φαίνεται στο παραπάνω γράφημα αρχικά δημιουργούμε μέσα από το MakeHuman τα μοντέλα ανθρωποειδών που θα χρησιμοποιήσουμε για παίκτες και εχθρούς. Τους ορίζουμε τις ιδιότητες που επιθυμούμε και αφού τελειώσουμε εξάγουμε το αποτέλεσμα σε μορφή που θα χρησιμοποιηθεί για περαιτέρω μορφοποίηση στο Blender.

Αφού παραχθεί το αρχείο για το Blender, ανοίγουμε το Blender για να διαχειριστούμε το εισερχόμενο μοντέλο μας. Η διαδικασία αυτή περιλαμβάνει διορθώσεις μορφολογικές που μπορεί να προέκυψαν κατά την εισαγωγή του, βάψιμο (texture paint) και δημιουργία animation όπου κρίνεται απαραίτητο. Παράλληλα στο ίδιο πρόγραμμα δημιουργούμε και τα γραφικά της πίστας ή και επιπρόσθετα αξεσουάρ στους παίκτες / εχθρούς, τα βάφουμε και αυτά με τον ίδιο τρόπο αντίστοιχα. Αφού γίνει αυτή η διαδικασία σε όλα τα αντικείμενα εξάγουμε τα αποτελέσματα μας στην Unity. Στο σημείο αυτό η Unity έχει λάβει όλα τα αρχεία που εμείς θέλουμε και είναι έτοιμη να διαχειριστεί το κάθε αντικείμενο ξεχωριστά. Αρχικά θέλουμε να χρησιμοποιήσουμε το animation mecanim της Unity για να ελέγξουμε τους παίκτες / εχθρούς για τυχόν σκελετικά λάθη προτού προβούμε στην διαδικασία να προχωρήσουμε. Αφού γίνει αυτό και βρεθεί ότι δεν υπάρχει κάποιο σφάλμα στον αριθμό των οστών, τότε μπορούμε να προβούμε στην διαδικασία δημιουργίας μιας μηχανής καταστάσεων (State Machine) οπού εκεί περνάμε τα animations αλλά και τις καταστάσεις που θα πρέπει να υπάρχουν για να μπορεί να συμβεί το εκάστοτε animation.

Αφού γίνουν αυτά μπορούμε να χτίσουμε το περιβάλλον και να ανανεώνουμε συνεχώς με νέα αντικείμενα έως ότου μείνουμε ικανοποιημένοι με το αποτέλεσμα μας. Αφού

γίνουν τα παραπάνω είμαστε έτοιμοι να πάμε στο κύριο σημείο της δουλειάς μας, δηλαδή να δημιουργήσουμε τον κατάλληλο κώδικα που θα περιγράψει την συμπεριφορά της πίστας, των χαρακτήρων μας αλλά και την τεχνητή νοημοσύνη των εχθρών που θα ακολουθούν. Τελευταίο κομμάτι αφήνουμε τον προγραμματισμό της διεπαφής μας αφού έχουμε καταλήξει στις τεχνικές ρυθμίσεις των υπόλοιπων κομματιών.

Όταν όλοι οι παραπάνω στόχοι επιτευχθούν τότε μπορούμε να εξάγουμε το τελικό αποτέλεσμά μας σε ένα εκτελέσιμο αρχείο στα πλαίσια αυτών που επιτρέπει η Unity για να ελέγξουμε τυχόν ανωμαλίες στην συμπεριφορά κάποιου αντικειμένου (alpha test), καθώς η διαδικασία αυτή επαναλαμβάνεται μέχρι να μείνουμε ευχαριστημένοι με την απόδοση της εφαρμογής μας.

Κεφάλαιο 3

Υλοποίηση

3.1. Χρήση του MakeHuman

Εξ αρχής ήταν απαραίτητο να γίνει χρήση κάποιου προγράμματος που μπορεί να μας δώσει έτοιμα μοντέλα ανθρωποειδούς μορφής και στα οποία θα δουλέψουμε προσθήκες και λεπτομέρειες. Το πρόγραμμα που χρησιμοποιήσαμε ονομάζεται MakeHuman. Όπως φαίνεται στην παρακάτω εικόνα πρόκειται για το αρχικό περιβάλλον κατά την εκκίνησή του.

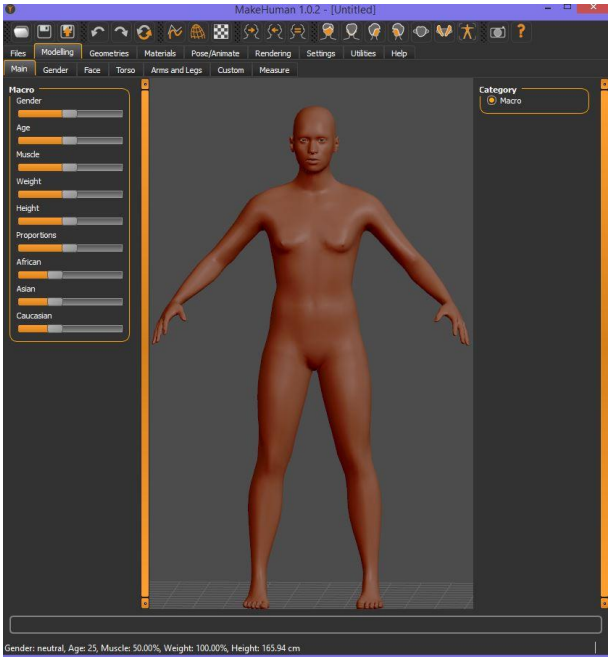

Figure 3.1-1:MakeHuman Starting Screen

3.1.1 Επιλογή Χαρακτηριστικών

Όπως βλέπουμε και στην παραπάνω εικόνα το πρόγραμμα αρχικά μας δίνει αρχικοποιημένες συνθήκες ενός ανθρωποειδούς και στο αριστερό μέρος της οθόνης έχει ένα σετ ρυθμίσεων που αφορούν το φύλο, ηλικία, μυϊκό όγκο, βάρος, ύψος, κατανομή διαστάσεων (proportions), και στις τελευταίες τρεις ρυθμίσεις μας δίνει την επιλογή για την φυλή στη οποία υπάγεται το δημιούργημά μας, ώστε να μπορέσουμε να προβούμε σε περαιτέρω ρυθμίσεις.

Στην παρακάτω εικόνα βλέπουμε τις επιλογές που μας δίνει το πρόγραμμα για την μορφοποίηση χαρακτηριστικών που αφορούν το φύλο που επιλέξαμε.

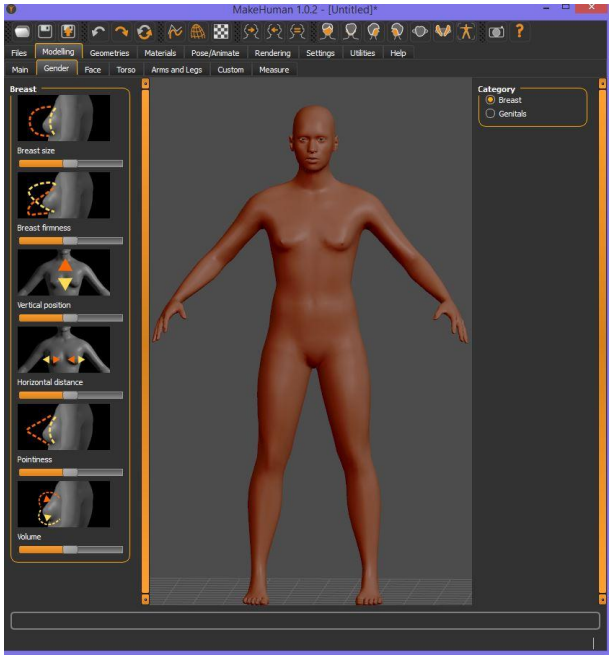

Figure 3.1.1-1: Gender Tab

Προχωράμε στην επόμενη καρτέλα μορφοποίησης του μοντέλου μας η οποία αφορά τα χαρακτηριστικά του προσώπου. Στην αριστερή πλευρά της παρακάτω εικόνας έχουμε τις διαθέσιμες μορφοποιήσεις που αφορούν το επιλεγμένο στοιχείο της δεξιάς λίστας, καθώς φαίνεται ότι παρέχει μορφοποίηση σε όλα τα δυνατά σημεία του προσώπου και επικεντρώνεται σε καθένα από αυτά.

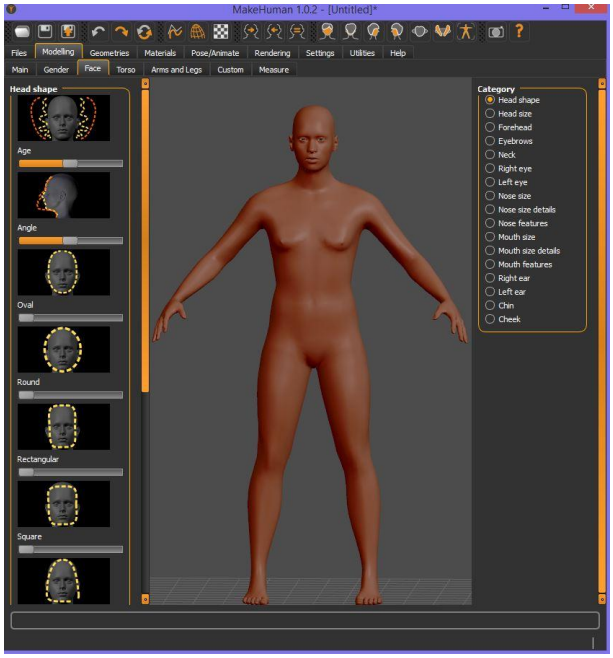

Figure 3.1.1-2: Face Tab

Έπειτα βλέπουμε στην παρακάτω εικόνα τις ρυθμίσεις που αφορούν τις μορφολογικές ρυθμίσεις για τον κορμό του σώματος οπού η λειτουργία είναι όμοια και με τις παραπάνω καρτέλες ρυθμίσεων έχοντας την αριστερή μπάρα για ρυθμίσεις του αντίστοιχου κομματιού που έχουμε επιλέξει από τον επιλογέα εστίασης στα δεξιά.

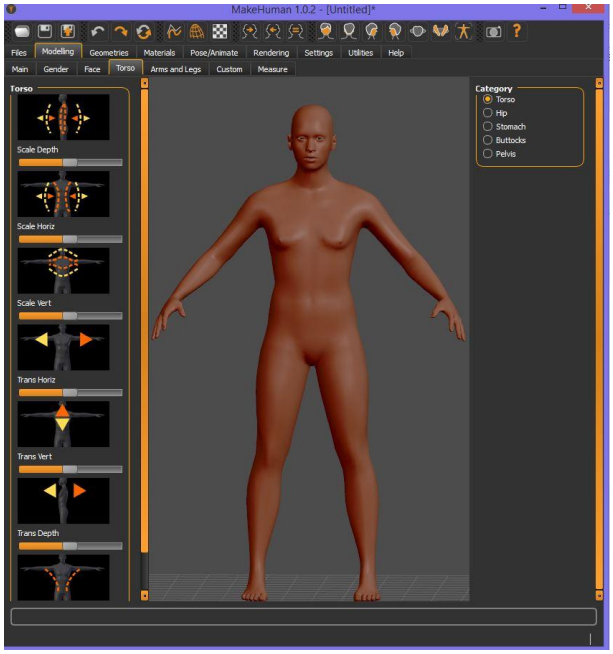

Figure 3.1.1-3:Torso Tab

Στην συνέχεια μπορούμε να δούμε αντίστοιχα στην παρακάτω εικόνα την καρτέλα μορφοποίησης για χέρια και πόδια που ισχύουν ακριβώς οι ίδιες συνθήκες στις επιλογές και δυνατότητες με τις παραπάνω καρτέλες. Στο σημείο αυτό πρέπει να αναφέρουμε ότι το πρόγραμμα μας προσφέρει την επιλογή να έχουμε διαφορετικά χαρακτηριστικά στο αριστερό πόδι από το δεξί. Το ίδιο ισχύει αντίστοιχα και για τα χέρια. Έχουμε την δυνατότητα ασυμμετρίας . Για να αποφευχθούν πιθανά λάθη συμμετρίας το πρόγραμμα έχει την επιλογή αυτόματης συμμετρίας το οποίο βρίσκεται στο σημείο που έχουμε μαρκάρει με κόκκινο περίγραμμα.

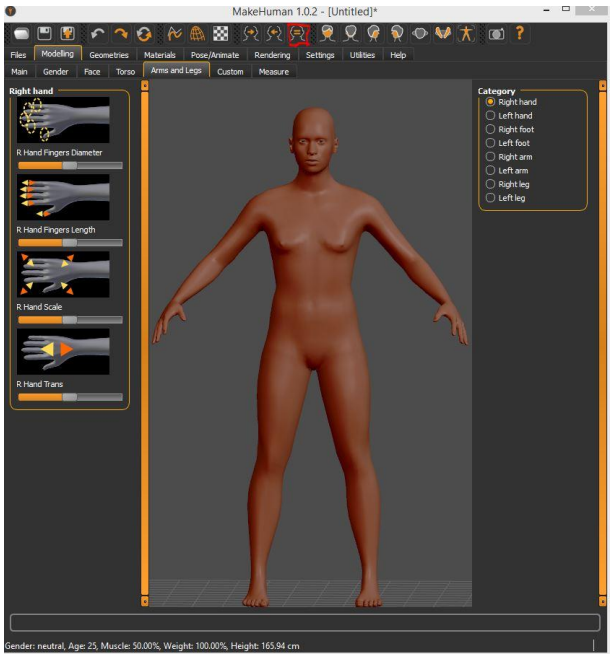

Figure 3.1.1-4: Arms and Legs Tab

Η τελική καρτέλα σωματικής μορφοποίησης για το ανθρωποειδές αφορά τις διαστάσεις ανά τμήμα που επιλέγουμε με τον ίδιο τρόπο που κάναμε και στις παραπάνω περιπτώσεις όπως φαίνεται στην παρακάτω εικόνα.

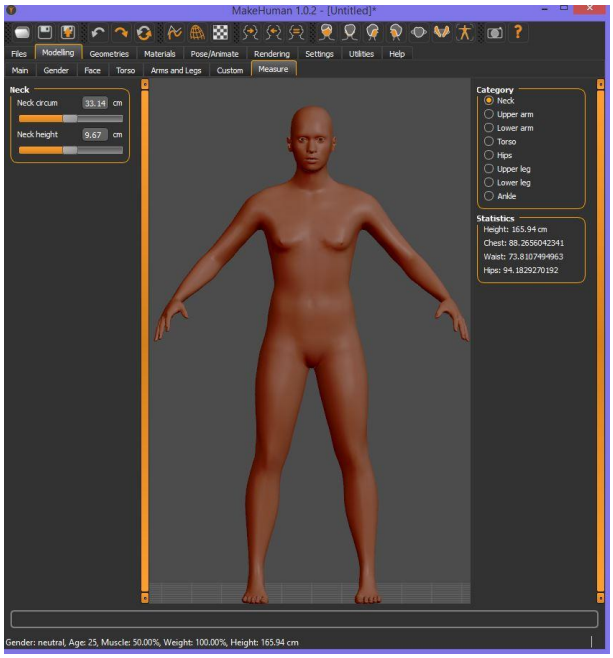

Figure 3.1.1-5: MeasureTab

3.1.2 Επιλογή Accessories

Η επιλογή αξεσουάρ (accessories) μας δίνει την δυνατότητα για προσθήκη ρούχων, μορφοποίηση ματιών, προσθήκη μαλλιών, προσθήκη γεννητικών οργάνων, προσθήκη φρυδιών, προσθήκη βλεφαρίδων, προσθήκη οδοντοστοιχίας και γλώσσας που κάθε καρτέλα αναλαμβάνει την αντίστοιχη λειτουργία. Ένα παράδειγμα μορφοποίησης φαίνεται στην παρακάτω εικόνα.

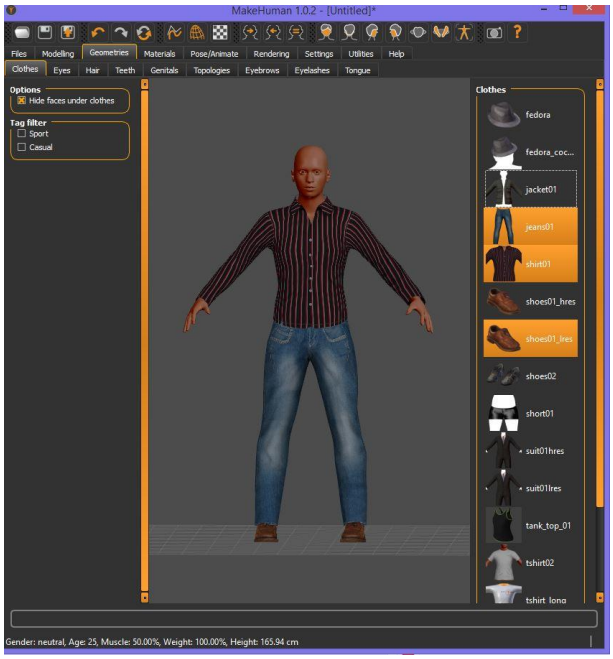

Figure 3.1.2-1: Accessories Tab

3.1.3 Επιλογές Χρώματος σε σώμα και μάτια

Στο MakeHuman έχουμε την δυνατότητα να ρυθμίσουμε (πέρα από την αρχική σελίδα) το χρώμα του σώματος και των ματιών μέσω κάποιων προκαθορισμένων λιστών επιλογής όπως φαίνεται στην παρακάτω εικόνα, καθώς και το χρώμα των μαλλιών (εφόσον έχουμε ορίσει μαλλιά στον χαρακτήρα μας).

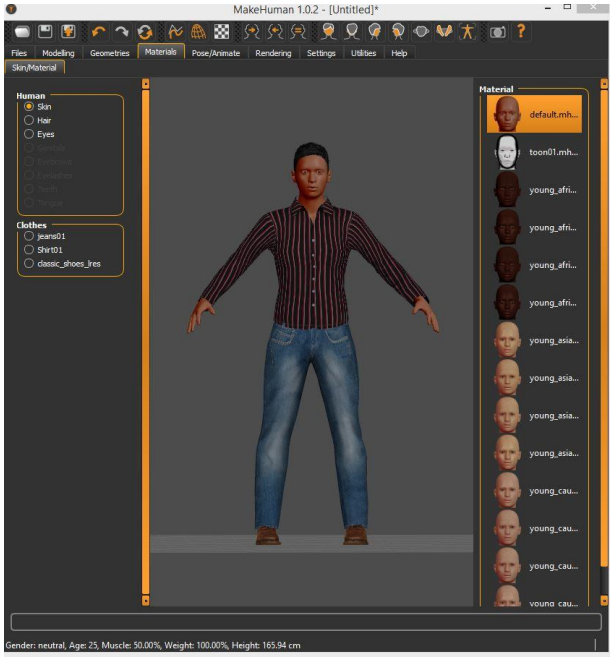

Figure 3.1.3-1: Materials Tab

3.1.4ΕπιλογέςΤύπου Rig

Το MakeHuman αποτελεί εξαιρετική λύση σε όσους δεν έχουν αρκετές γνώσεις στο να στήσουν σύστημα σκελετικής γεωμετρίας (rigging) και αυτοματοποιεί την διαδικασία αυτή για να επιταχύνει την πρόοδο της παραγωγής σε όσους δεν έχουν την γνώση ή και τον απαιτούμενο χρόνο. Μέσω της καρτέλας "Pose / Animate" έχουμε μια γκάμα επιλογών ως προς το ποιο είδος σκελετικής γεωμετρίας έχουμε ανάγκη. Στα πλαίσια αυτής της εργασίας χρησιμοποιήσαμε το game.json καθώς ήταν η καταλληλότερη επιλογή για την εισαγωγή στο Blender οπού και θα αναφερθούμε αργότερα.Στην παρακάτω εικόνα βλέπουμε ένα παράδειγμα της καρτέλας επιλογής rigformat.

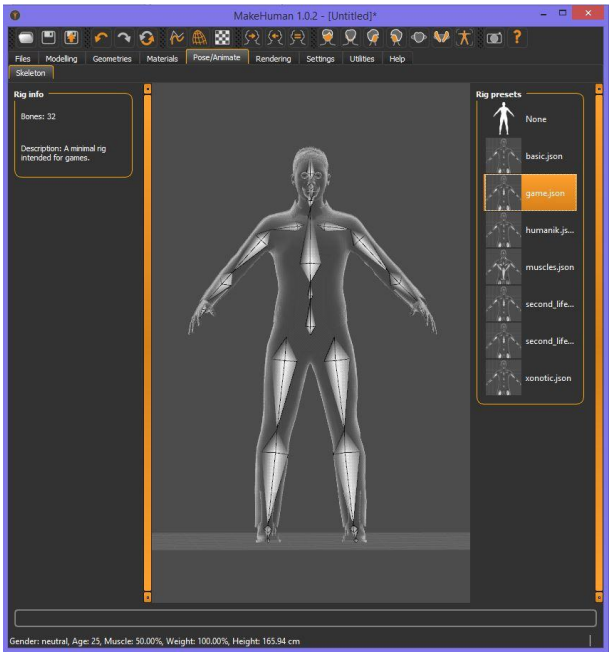

Figure 3.1.4-1: Rig Format Tab

3.1.5 Επιλογέςexport format

Το τελευταίο βήμα στην παραγωγή των ανθρωποειδών μοντέλων μας είναι η εξαγωγή τους για να χρησιμοποιηθούν στο Blender ή και την απευθείας εισαγωγή τους στην Unity. Αρχικά μας ενδιαφέρει στον επιλογέα τι μέρος αυτού του αντικειμένου θέλουμε να σώσουμε, δηλαδή αν θέλουμε να είναι Mesh Format, RigFormat ή Maps. Αν θέλουμε και από τις 3 κατηγορίες δεδομένα αποθηκεύουμε σε ξεχωριστά αρχεία για το συγκεκριμένο μοντέλο την κάθε πληροφορία.

Το Mesh Format μας παρέχει μια σειρά επιλογών ως προς την χρήση που θέλουμε να κάνουμε μετά το export. Στην περίπτωση μας για αυτή την εργασία χρησιμοποιήθηκαν στην κατηγορία του Mesh Format οι επιλογές "Blender Exchange (mhx)" για το Blender, καθώς και "Filmbox (.fbx)" για την απευθείας χρήση στην Unity. Στην κατηγορία του RigFormat χρησιμοποιήσαμε την επιλογή "Biovision Hierarchy BVH" για τον παίκτη στον οποίο ορίσαμε animations μέσα από το Blender.Παράδειγμα της καρτέλας Export φαίνεται στην παρακάτω φωτογραφία:

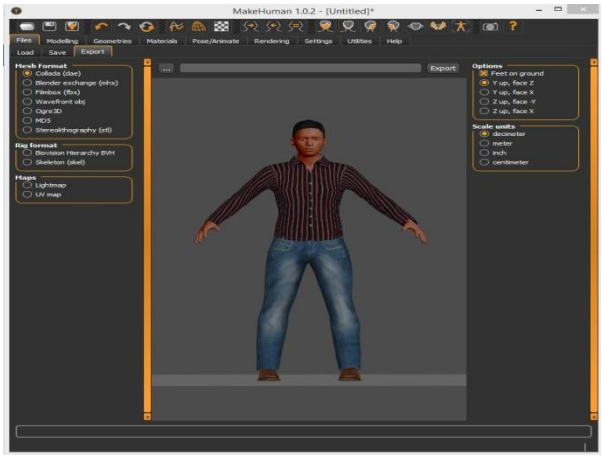

Figure 3.1.5-1: Export format options Tab

3.2. Χρήση του Blender

Όπως αναφέρθηκε και σε παραπάνω κεφάλαιο η επιλογή του προγράμματος στο οποίο και τελικά δημιουργήσαμε τα γραφικό περιεχόμενο του παιχνιδιού αυτού είναι το Blender. Στο κεφάλαιο αυτό μπορείτε να βρείτε πληροφορίες για τις τεχνικές με τις οποίες υλοποιήθηκαν τα γραφικά, καθώς και κάποια animations στον ένα από τους 2 χαρακτήρες(παίκτες) μας εφόσον δεν μας κάλυπτε το πακέτο της Unity με προδιαγεγραμμένα animations.Προτού ξεκινήσουμε την ανάλυση θα δούμε το αρχικό περιβάλλον του προγράμματος Blender.

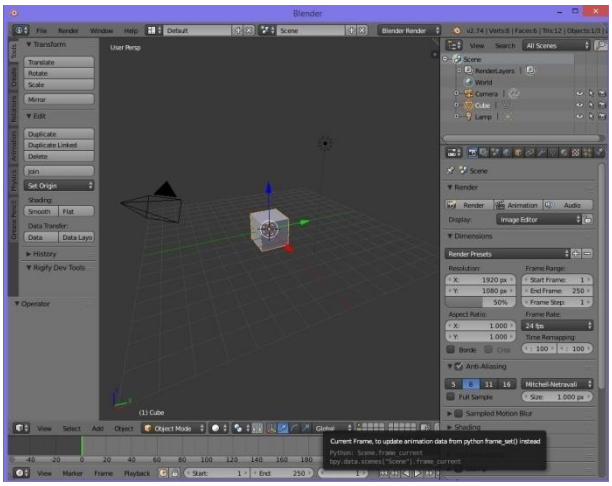

Figure 3.2-1:Blender Starting Screen

Στην παραπάνω φωτογραφία βλέπουμε την αρχική οθόνη που μας ανοίγει το πρόγραμμα κατά την εκκίνησή του. Περιλαμβάνει στο κέντρο του μία σκηνή που είναι ο κόσμος που βρίσκεται οτιδήποτε δημιουργούμε για το κάθε αντικείμενο/α που επιθυμούμε. Στην αριστερή πλευρά της εικόνας έχουμε μια εργαλειοθήκη που περιλαμβάνει δημιουργία νέων αντικειμένων για προσθήκη στην σκηνή μας αλλά και ιδιότητες των αντικειμένων που έχουμε επιλέξει.

Στο κάτω μέρος της οθόνης έχουμε μερικές ακόμα ιδιότητες για το/τα αντικείμενό/ά μας, πληροφορίες που αφορούν την κατάσταση του αντικειμένου μας και ένα animation playback μενού που αναπαράγει τα animations που έχουμε προδιαγράψει για το επιλεγμένο αντικείμενο οπού θα μελετήσουμε αναλυτικότερα παρακάτω.

Στην δεξιά πλευρά της οθόνης έχουμε (με σειρά από πάνω προς τα κάτω) την σκηνή μας ιεραρχικά δομημένη ώστε να γνωρίζουμε τι περιέχει οποιαδήποτε στιγμή και γενικές πληροφορίες για το κάθε αντικείμενο. Ακριβές από κάτω βρίσκεται ο επιλογέας ιδιοτήτων που περιέχει καρτέλες για render (απεικόνιση της σκηνής), render layer (επίπεδο απεικόνισης ιεραρχικά), scene (πληροφορίες για την κατάσταση της σκηνής μας),world (πληροφορίες για τον κόσμο που υπάγεται η σκηνή, το αντικείμενο το οποίο έχουμε επιλέξει με τις ιδιότητες του, περιορισμοί, μετατροπείς αντικειμένων (θα αναφερθούμε παρακάτω σε κάποιους), περισσότερες πληροφορίες για το αντικείμενό μας, υλικό (υφή και Shader), επιλογή υφής, particles (συμπεριφορά σωματιδίων / special effect) και τέλος physics (φυσική στην οποία υπάγεται το αντικείμενο (αν υπάγεται)).

3.2.1 Object Modeling

Το object modeling ήταν απαραίτητο για την δημιουργία αντικειμένων στις πίστες αλλά και για αντικείμενα που αφορούν τους παίκτες μας αλλά και τους εχθρούς. Ακολουθήθηκαν διαφορετικές διαδικασίες για την παραγωγή τους οπού θα αναδείξουμε.

Αρχικά πρέπει να παραθέσουμε ένα κομμάτι δημιουργίας γραφικού που βασίζεται σε απλά transformation operations (χειρισμοί μετατροπών), δηλαδή scale (αυξομείωση μεγέθους στον τρισδιάστατο χώρο), rotate (περιστροφή και στους 3 άξονες), translate (μετατόπιση και στους 3 άξονες).

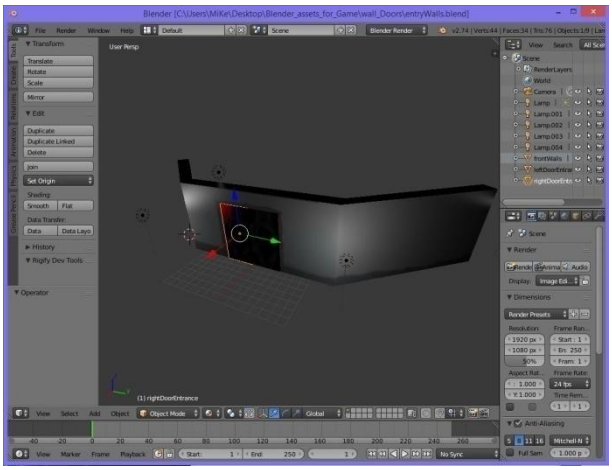

Figure 3.2.1-1:Wall with doors

Στην παραπάνω εικόνα βλέπουμε έναν τοίχο με 2 πόρτες στο κεντρικό του σημείο. Αρχικά δημιουργήσαμε ένα κύβο και μέσω των βασικών transform operations δημιουργήσαμε το κομμάτι του τοίχου συνθέτοντας την γεωμετρία του κατάλληλα. Επιπλέον βλέπουμε 2 πόρτες οι οποίες δημιουργήθηκαν αντίστοιχα με τον ίδιο τρόπο. Για να δώσουμε όμως την αίσθηση στον χρήστη που θα δοκιμάσει την εφαρμογή ότι οι πόρτες καταλαμβάνουν ένα κενό χώρο χρησιμοποιήσαμε μία διαφορετική τεχνική επάνω στην γεωμετρία του τοίχου όπως φαίνεται στην παρακάτω εικόνα.

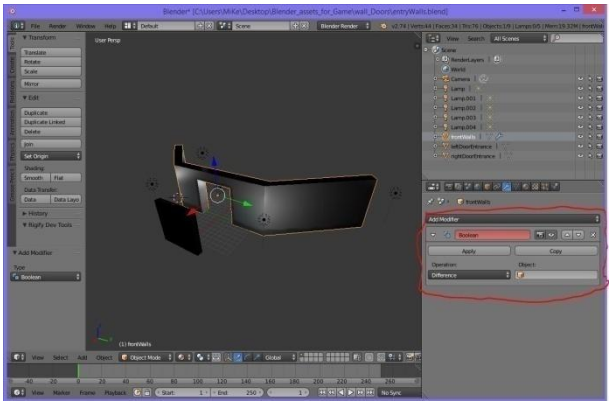

Figure 3.2.1-2: Boolean Modifier on Walls

Η τεχνική αυτή όπως φαίνεται μέσα στο κόκκινο περίγραμμα ονομάζεται Boolean Modifier. Η τεχνική αυτή μας επιτρέπει τρεις δυνατότητες : την αφαίρεση ενός αντικειμένου από ένα άλλο (Difference),την ένωση 2 αντικειμένων (Union) και την διασταύρωση 2 αντικειμένων (Intersection).Για τις δικές μας ανάγκες χρησιμοποιήσαμε στα πλαίσια αυτής εργασίας την τεχνική Difference για να δημιουργήσουμε το άνοιγμα της πόρτας πάνω στον

τοίχο. Η ίδια διαδικασία έγινε και για να επιτύχουμε σε άλλα αντικείμενα που δημιουργήσαμε όπως φαίνεται στις παρακάτω εικόνες.

Figure 3.2.1-3: Boolean Modifier on Barrel

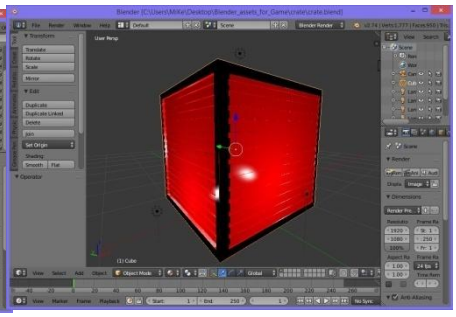

Figure 3.2.1-4: Boolean Modifier on Crate

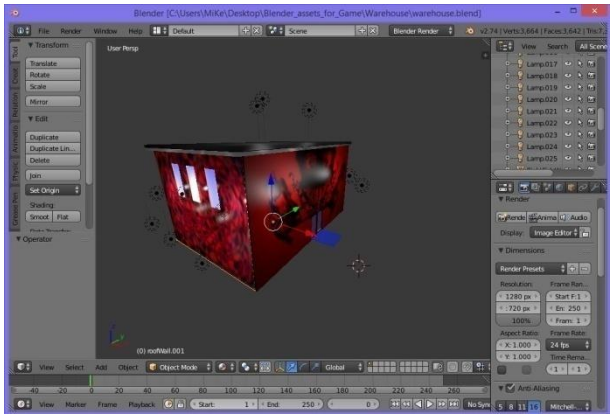

Figure 3.2.1-5: Boolean Modifier on Warehouse

Οι παραπάνω εικόνες αποτελούν αποτέλεσμα χρήσης του Boolean Modifier tool με την επιλογή Difference, καθώς για να επιτευχθεί το αντίστοιχο αποτέλεσμα χρειάστηκε να γίνει συνεχής εφαρμογή του με κατάλληλα σχήματα όπως διαμορφωμένα αρχικά σχήματα κύβου και κυλίνδρου.

Στα πλαίσια αυτής της εργασίας χρησιμοποιήθηκε και η τεχνική μοντελοποίησης ελεύθερης διαμόρφωσης γεωμετρίας (Edit Mesh) . Για να προβούμε σε αυτή την διαδικασία δεν έχουμε παρά να προβούμε έχοντας επιλεγμένο το αντικείμενο που επιθυμούμε να διαμορφώσουμε πηγαίνοντας στο κάτω μέρος στο πρόγραμμά μας στο σημείο που λέει "Object Mode" και να πατήσουμε ανοίγοντας το μενού και επιλέγοντας "Edit Mode" όπως φαίνεται στην παρακάτω εικόνα.

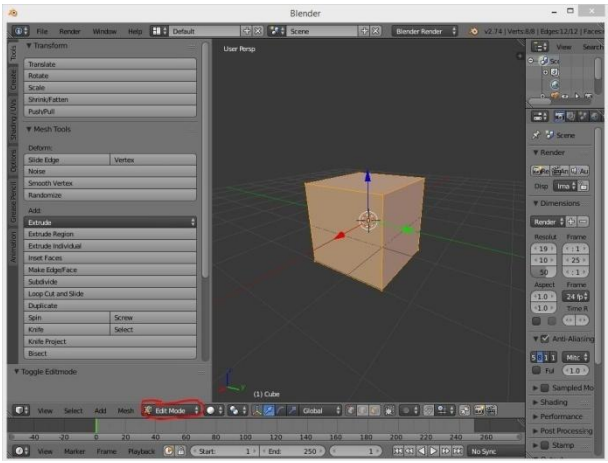

Figure 3.2.1-6: Edit Mode

Έχοντας φτάσει σε αυτό το σημείο βλέπουμε στα αριστερά μια καρτέλα με tools τα οποία μας προσφέρουν πάρα πολλές επιλογές να διαμορφώσουμε την γεωμετρία μας όπως εμείς επιθυμούμε. Χρησιμοποιώντας το πλήκτρο "a" επιλέγουμε όλα τα σημεία (points) του αντικειμένου μας, καθώς αν το ξαναπατήσουμε δεν αφήνει κανένα σημείο επιλεγμένο. Για σημειακή επιλογή πολλών σημείων πατάμε το πλήκτρο "c" και επιλέγουμε όσα σημεία επιθυμούμε να επηρεάσουμε.

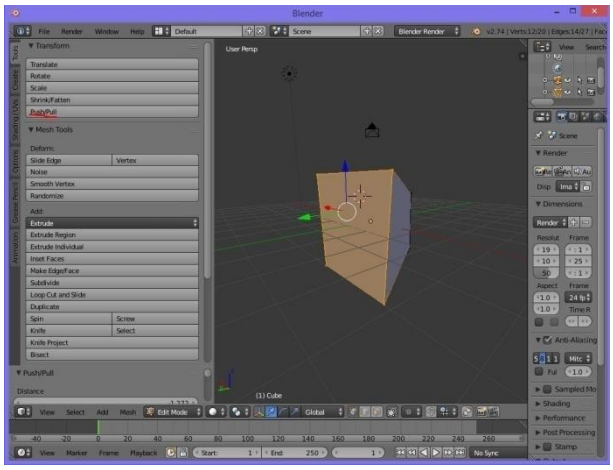

Figure 3.2.1-7: Push / Pull Edit Mesh

Στην παραπάνω εικόνα χρησιμοποιήσαμε το εργαλείο "push / pull" το οποίο μεγαλώνει ή συρρικνώνει ένα face (επιφάνεια/ σύνολο σημείων).Γενικότερα μέσω αυτής της εργαλειοθήκης πετύχαμε κάποια από τα αντικείμενα για τους χαρακτήρες της εργασίας με συνδυασμό εργαλείων όπως φαίνεται στις παρακάτω εικόνες.

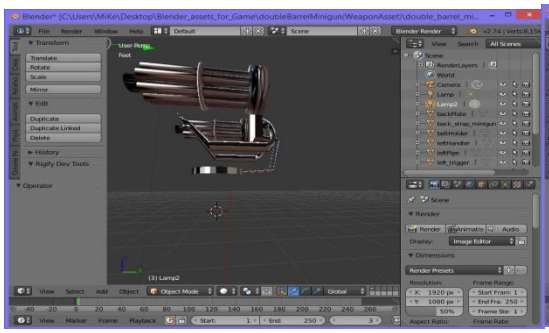

Figure 3.2.1-8:Minigun Weapon

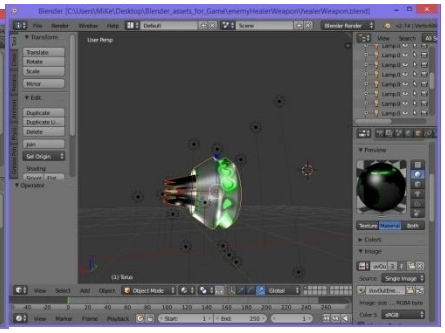

Figure 3.2.1-9: Healer's Weapon

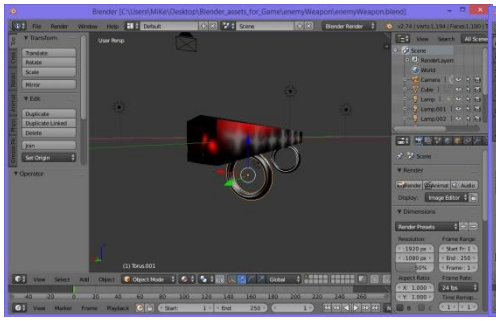

Figure 3.2.1-10: Enemy's Weapon

Figure 3.2.1-11: Sunglasses

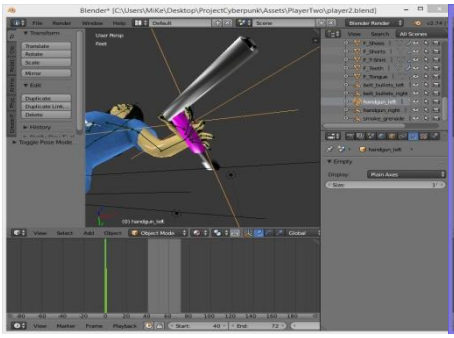

Figure 3.2.1-12:Handgun

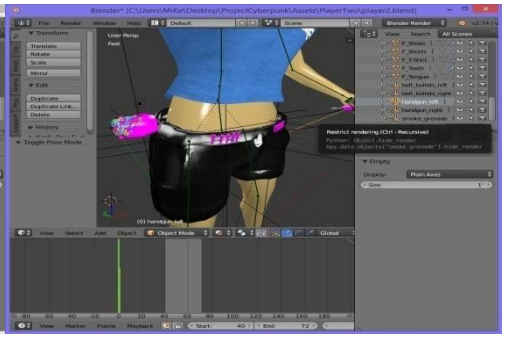

Figure 3.2.1-13:Belt with grenade and bullets

Σε όλες τις παραπάνω φωτογραφίες υπέστησαν αλλαγές μέσω του "Edit Mode" που αναφέραμε παραπάνω.(Τα ροζ χρώματα σε κάποιες από τις φωτογραφίες οφείλονται στην αλλαγή υλικού μέσα στην Unity).

3.2.2 Animation Creation

Το κομμάτι του animation στο Blender χρησιμοποιήθηκε στα πλαίσια της πτυχιακής για ανάπτυξη βασικών κινήσεων walk / run (περπάτημα / τρέξιμο) και συγκεκριμένα στον χαρακτήρα με τα διπλά handguns. Η ποιότητα των animation είναι ερασιτεχνική καθώς αποτελούσε επιπρόσθετη εργασία από τα προβλεπόμενα και η γνώση για δημιουργία καλών αισθητικά και ρεαλιστικών animations απαιτεί πάρα πολύ χρόνο στην μάθησή τους και στην εφαρμογή τους. Στην παρακάτω εικόνα βλέπουμε ένα στιγμιότυπο του animation για την δράση "Run".

Figure 3.2.2-1: Run Animation

Αρχικά πρέπει να αναφέρουμε ότι για την κατασκευή των animations χρειαζόμαστε την τεχνική των keyframes. Πρόκειται για μια μέθοδο όπου λειτουργεί παρόμοια σε όλα τα προγράμματα παραγωγής animation και βασίζεται στην λογική της καταγραφής συγκεκριμένων στιγμών ως στιγμές-κλειδιά. Για ένα πεπερασμένο διάστημα του χρόνου που

θέτουμε εμείς μπορούμε μέσα στην ζώνη εκείνη να δημιουργήσουμε την οποιαδήποτε κίνηση θέλουμε σε όσα διαστήματα θέλουμε. Κάθε τιμή μεταξύ 2 keyframes παράγει όλες τις ενδιάμεσες τιμές για να ικανοποιήσει την μετάβαση που μπορεί να αφορά την οποιαδήποτε πράξη κίνησης, περιστροφής ή αυξομείωσης μεγέθους που κάναμε. Στα ανθρωποειδή μοντέλα λόγω της αυξημένης πολυπλοκότητας χειριζόμαστε τα animations (συνήθως) μέσω του σκελετικού συστήματος (rig) που μας παρέχουν για να αποφύγουμε άσκοπη δουλειά που πιθανότατα θα αλλοιώσει το αποτέλεσμα.

Όπως φαίνεται στην παραπάνω φωτογραφία στο δεξί μέρος της εικόνας έχουμε τα keyframes για κάθε κίτρινο ρόμβο που περιέχει οριζόντια και το μέλος του σώματος που αφόρα στην λίστα αριστερά για όλο το διάστημα που καταγράφηκε και αριστερά κάτω το timeline που περιγράφει την διάρκεια μέσα στο ανοιχτό γκρίζο διάστημα και οι κίτρινες κατακόρυφες γραμμές αντιπροσωπεύουν τα σημεία στα οποία καταγράψαμε κομμάτι της κίνησης που θέλουμε (keyframes), ενώ στην σκηνή βλέπουμε το αντικείμενο στο οποίο υπάγεται το animation ένα στιγμιότυπο της προσομοίωσης της κίνησης.

3.2.3 Texture Paint

Η τεχνική Texture Paint βασίζεται στην λογική βαψίματος με πινέλα (brushes), είναι αρκετά χρονοβόρα ανάλογα το επίπεδο λεπτομέρειας που θέλουμε να επιτύχουμε. Η διαδικασία αυτή ξεκινάει επιλέγοντας το αντικείμενο που θέλουμε να βάψουμε και επιλέγοντας στο σημείο που αναγράφεται "Object Mode" αλλάζοντας την επιλογή σε "Texture Paint" και δεξιά από αυτό αλλάζουμε μέσα από την λίστα την αρχική επιλογή σε "Material" όπως θα δούμε στην παρακάτω εικόνα.

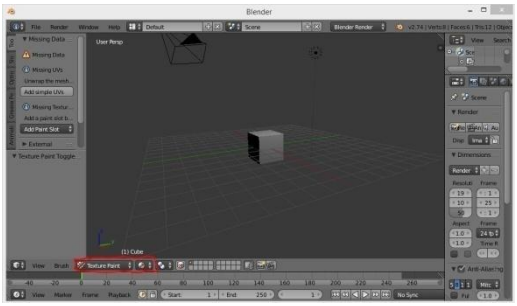

Figure 3.2.3-1: Texture Paint Step 1

Αφού γίνει αυτό θα εμφανιστούν καινούργιες επιλογές στην αριστερή καρτέλα. Πατάμε στο κουμπί "Add simple UVs" που θα μας δώσει αρχικές συνθήκες για να μπορούμε να βάψουμε. Μετέπειτα επιλέγουμε το μενού επιλογών που θα εμφανιστεί με τίτλο "Add Paint Slot" για να τοποθετήσουμε πινέλο και επιλέγουμε "Diffuse Color". Θα μας ζητήσει να επιλέξουμε αρχικό χρώμα που θα το βάψει μέσα από παλέτα χρωμάτων. Επιλέγουμε οποιοδήποτε χρώμα μιας και αυτό θα είναι προσωρινό.Στην ακόλουθη εικόνα βλέπουμε το αποτέλεσμα των επιλογών μας στο οποίο θα πάμε να χρωματίσουμε.

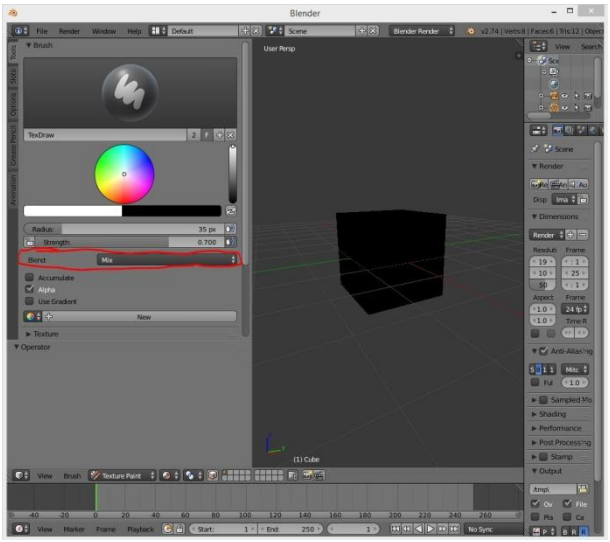

Figure 3.2.3-2:Texture Paint Step 2

Αφού έγινε το παραπάνω βήμα μπορούμε να βάψουμε σημειακά το αντικείμενό μας επιλέγοντας μέσω των επιλογών "Radius" και "Strength" το πάχος του πινέλου και την ποσότητα χρώματος . Έπειτα βλέπουμε αρχικοποιημένη συνθήκη "Mix" στην κατηγορία "Blend" όπως φαίνεται στην παραπάνω εικόνα μέσα στο κόκκινο περίγραμμα. Αυτό μενού περιέχει τύπους που μπορούμε να εμπλουτίσουμε το χρώμα μας μιας και προσφέρει μια γενναία ποσότητα εργαλείων χρωματισμού. Ας δούμε ένα παράδειγμα βαφής του κύβου αυτού χρησιμοποιώντας το εργαλείο αυτό και συνδυάζοντας πινέλα και κάποια από τα διαφορετικά mode τους.

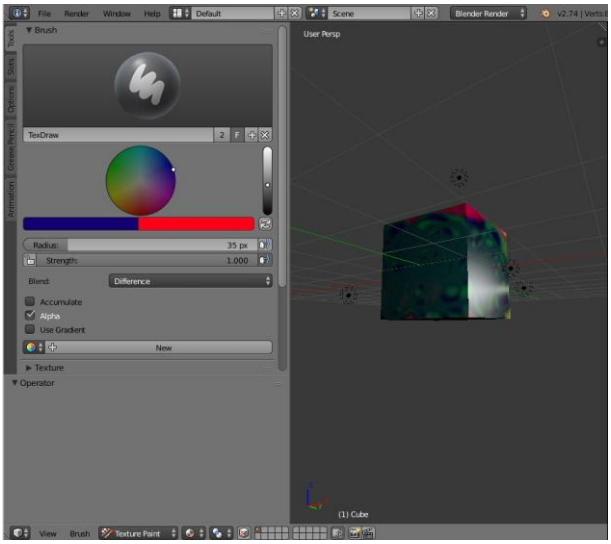

Figure 3.2.3-3:Texture Paint Step 3

Το συγκεκριμένο αποτέλεσμα παράχθηκε χρησιμοποιώντας μόνο μερικά από τα είδη χρωματισμού στον επιλογέα και έχουμε αυτό το αποτέλεσμα το οποίο θα αποτελέσει texture (υφή) τύπου graffiti. Οι βαφές στις εικόνες του υποκεφαλαίου (3.2.1) έχουν υλοποιηθεί με αυτό τον τρόπο στα αντικείμενα του βιντεοπαιχνιδιού που αντιπροσωπεύουν.

3.2.4 UV Mapping

Ακολουθώντας το παραπάνω παράδειγμα με τον κύβο θα εξετάσουμε πως μπορούμε να εξάγουμε το αποτέλεσμα του χρωματισμού σε ύφη που θα χρησιμοποιηθεί στην Unity για να δώσει το τελικό αποτέλεσμα. Με βάση την παρακάτω εικόνα επιλέγουμε στην δεξιά πλευρά την λίστα μέσα στο κόκκινο περίγραμμα και αλλάζουμε προσωρινά την επιλογή σε "UV / Image Editor".

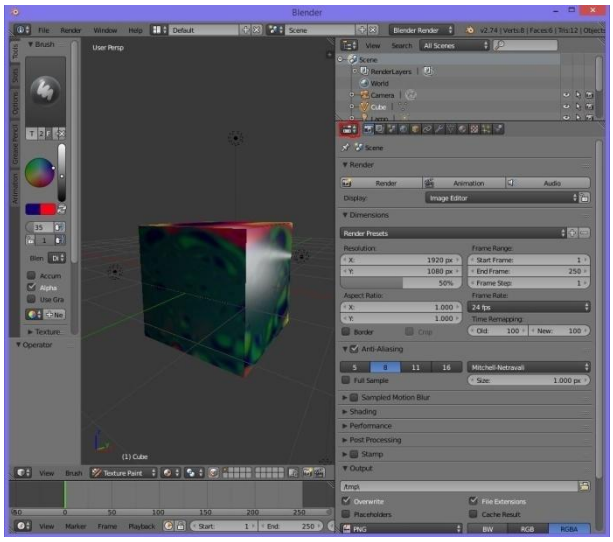

Figure 3.2.4-1:UV Mapping Step 1

Στην συνέχεια εάν όλα έχουν γίνει σωστά θα πάρουμε το παρακάτω αποτέλεσμα στην οθόνη μας. Η λίστα στην ακόλουθη εικόνα με το κόκκινο περίγραμμα περιέχει χαρτογραφημένες συντεταγμένες σε μορφή material του αντικειμένου που χρωματίσαμε.(Αν δεν εμφανιστεί με την ενεργοποίηση που περιγράψαμε παραπάνω τότε επιλεγούμε χειροκίνητα ανοίγοντας την μαρκαρισμένη λίστα όπως φαίνεται παρακάτω.)

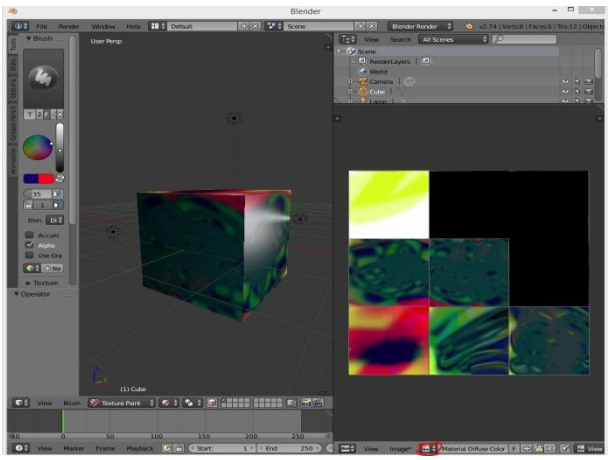

Figure 3.2.4-2:UV Mapping Step 2

Σε αυτό το σημείο βλέπουμε ξεκάθαρα την χαρτογράφηση του χρωματισμού και έχουμε το UV Map έτοιμο για να τον αποθηκεύσουμε προκειμένου αυτή η ύφη να είναι διαθέσιμη αργότερα και στην Unity. Πατάμε κάτω δεξιά στο πλαίσιο "Image" και έπειτα "Save As Image".

3.2.5 Εξαγωγή σε μορφή FBX

Σε κάποια σημεία της εργασίας χρειάστηκε να εξάγουμε αντικείμενα σε μορφή ".fbx" και στις παρακάτω απεικονίσεις θα εξηγήσουμε πως μπορεί να γίνει αυτή η διαδικασία. Η εξαγωγή σε μορφή .fbx έγινε συγκεκριμένα σε ένα κομμάτι γραφικών μιας και προοριζόταν για ανθρωποειδές που είχε εξαχθεί από το MakeHuman κατευθείαν στην Unity. Για λόγους αποφυγής αταξίας δεδομένων μεταξύ διαφορετικών περιβαλλόντων και για την καλύτερη μεταξύ τους συνοχή κρίθηκε ότι ήταν καλύτερο να γίνει η παρακάτω διαδικασία.

Έχοντας έτοιμη την σκηνή του Blender με φορτωμένο το αρχείο προς εξαγωγή επιλέγουμε πάνω αριστερά "File", πάμε επάνω από το σημείο "Export" και αφού ανοίξει το μενού επιλέγουμε "FBX (.fbx)" και βλέπουμε την παρακάτω εικόνα στην οθόνη μας.

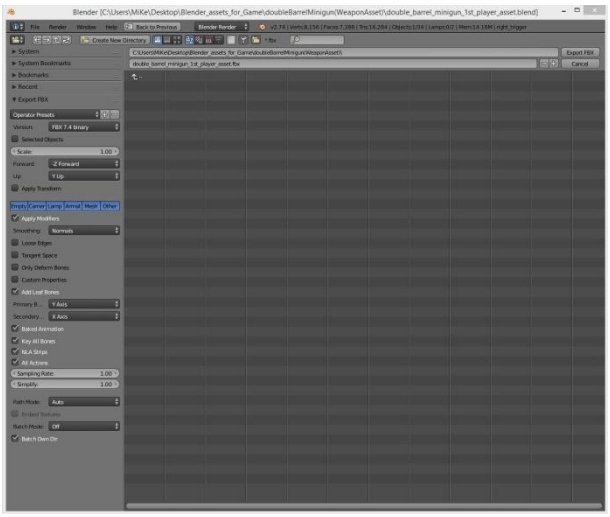

Figure 3.2.5-1:Export to FBX File Format

Στην καρτέλα αριστερά της οθόνης με όνομα "Export FBX" παρατηρούμε τις επιλογές που μας δίνει για την εξαγωγή. Από αυτές τις επιλογές μας ενδιαφέρουν τα πεδία "Version", "Scale", "Forward", "Up" και το κουτάκι "Apply Transform" για τις δικές μας ανάγκες.

Στο πεδίο "Version" αφήνουμε την επιλογή που μας έχει προκαθορίσει και συνεχίζουμε στο πεδίο "Scale". Αυτό το πεδίο μας επιτρέπει να αλλάξουμε το μέγεθος του εξαγόμενου αρχείου και αναλόγως τις ανάγκες μας μπορούμε να το τροποποιήσουμε, αλλά για την εργασία αυτή το κρατήσαμε σταθερό. Τα πεδία "Forward" και "Up" είναι πολύ σημαντικά διότι αποτελούν την αρχή των αξόνων για το πρόγραμμα στο οποίο στοχεύουμε και επειδή η Unity έχει το Z άξονα σε Forward κατάσταση αλλάζουμε από "-Z Forward" σε "Z Forward" και το πεδίο "Up" το αφήνουμε σταθερό και πατάμε tick στο πεδίο "Apply Transform" για να θέσει σε εφαρμογή τις ρυθμίσεις μας.

3.3. Χρήση της Unity

Σε αυτό το κεφάλαιο θα αναλύσουμε την εργασία που έγινε στα πλαίσια αυτής της εργασίας και αποτελεί το βασικότερο και σημαντικότερο κομμάτι . Στις παρακάτω σελίδες θα

περιγραφεί η διαδικασία δεσίματος των αντικειμένων για την παραγωγή ενός ενιαίου αποτελέσματος σε κάθε πτυχή που και τελικά θα είναι το βιντεοπαιχνίδι.

3.3.1 Εισαγωγή

Προτού προβούμε σε ανάλυση τμηματική θα περιγράψουμε το αρχικό περιβάλλον της Unity.Το αρχικό περιβάλλον που αντικρίζουμε μετά την δημιουργία ενός άδειου project της Unity φαίνεται στην παρακάτω εικόνα.

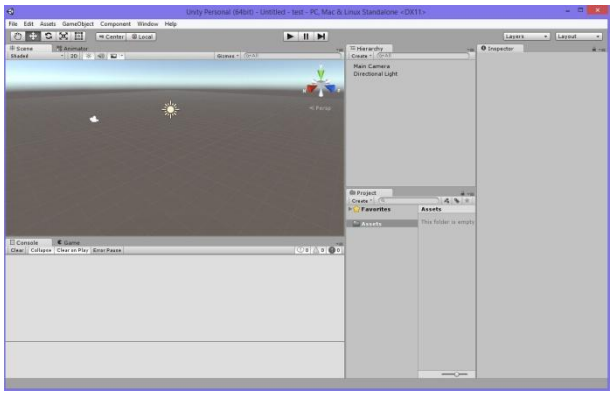

Figure 3.3.1-1:Home Screen in Unity

Αρχικά παρατηρούμε στην κορυφή μια εργαλειοθήκη με 7 καρτέλες που περιέχουν κατηγοριοποιημένες πληροφορίες για πράγματα που μπορούμε μέσω της Unity να κάνουμε. Στο πεδίο "File" περιέχονται επιλογές για την σκηνή όπως φόρτωμα σκηνής, σώσιμο σκηνής, δημιουργία νέας σκηνής, αλλά και δυνατότητες δημιουργίας, σωσίματος και φορτώματος project.

Στο πεδίο "Edit" περιέχονται βασικές λειτουργικές μετατροπές για το αντικείμενο που θέλουμε να επηρεάσουμε όπως αποκοπή, αντιγραφή, επικόλληση, αλλά και δημιουργία πανομοιότυπου αντικειμένου.

Αμέσως μετά βρίσκεται το πεδίο "Assets" που περιέχει την δημιουργία αντικειμένων που προσφέρει μέσα στο πακέτο της η Unity για οποιαδήποτε χρήση(ήχος, υλικό, φυσική, animation, colliders, scripts κλπ.). Μέσα από αυτή την καρτέλα μπορεί κάποιος να δημιουργήσει δικό του asset ή να φέρει κάποιο άλλο που προμηθεύτηκε από το "Asset Store" της Unity.

Στο πεδίο "GameObject" περιέχονται βασικά στοιχεία αντικειμένων που χρησιμοποιούνται συχνά όπως βασικά 3D και 2D γεωμετρικά σχήματα, πηγές ήχου, συστήματα διεπαφής, φωτισμός, κάμερα καθώς και άδεια αντικείμενα που προορίζονται για οποιαδήποτε χρήση.

Στο πεδίο "Component" μας παρέχονται στοιχεία συμπεριφορών αντικειμένου/ων που έχουν ευρεία χρήση σε όλων των ειδών εργασίες (παρόμοια με αυτά που περιέχει στο πεδίο "Assets").

Έπειτα έχουμε το πεδίο "Window" που περιέχει ιδιότητες μορφοποίησης του χώρου εργασίας μας, δηλαδή τον τρόπο που θέλουμε να βλέπουμε το αρχείο μας μαζί με όλα τα αντικείμενα που το αφορούν, καθώς και επιπρόσθετες καρτέλες που επιτρέπουν επισκόπηση σε άλλες πτυχές της εργασίας.Τελευταίο στην λίστα είναι το πεδίο "Help" όπου περιέχει πληροφορίες σχετικά με την game engine.

Στην συνέχεια έχουμε στο αριστερό μέρος της οθόνης τις καρτέλες "Scene" και ακριβώς από κάτω τις "Console" και "Game". Η καρτέλα "Scene" περιέχει όλο το οπτικό

υλικό που έχουμε προσθέσει μια οποιαδήποτε στιγμή στην σκηνή μας, ενώ η καρτέλα "Game" μας δείχνει τι βλέπει η κάμερα που έχουμε ορίσει ως ενεργή (να τονίσουμε σε αυτό το σημείο ότι αν δεν υπάρχει αντικείμενο κάμερας στην σκηνή μας δεν θα μπορούμε να δούμε απολύτως τίποτα κατά την διάρκεια του παιχνιδιού), καθώς η καρτέλα περιέχει πληροφορίες που αφορούν λάθη σε κώδικα που προέκυψαν κατά την διάρκεια της προσομοίωσης, συντακτικά λάθη στον κώδικα μας και άλλες πληροφορίες σχετικές με τον προγραμματισμό των scripts.

Στην δεξιά πλευρά της οθόνης βλέπουμε την καρτέλα "Hierarchy" που περιλαμβάνει τα αντικείμενα που έχουμε προσθέσει στην σκηνή μας μια οποιαδήποτε στιγμή, ενώ στα δεξιά της η καρτέλα "Inspector" δείχνει αναλυτικά τα στοιχεία του αντικειμένου που έχουμε επιλέξει μέσα από το hierarchy ή scene.

Τέλος έχουμε την καρτέλα "Project" που περιλαμβάνει τον βασικό φάκελο "Assets" ο οποίος χρησιμοποιείται για να αποθηκεύουμε αντικείμενα που θέλουμε να κρατήσουμε για γενική χρήση που δημιουργήσαμε στην σκηνή μας, για αντικείμενα που στην περίπτωση της εργασίας αυτής φέραμε μέσω του Blender, αλλά και πιθανότατα κάποιων assets που κατεβάσαμε μέσω του Asset Store.

3.3.2 Το Περιβάλλον των επιπέδων

Στα πλαίσια της εργασίας αυτής δημιουργήσαμε τρεις σκηνές οι οποίες αποτελούν το τελικό έργο της εργασίας σε μορφή βιντεοπαιχνιδιού. Η 1η σκηνή αποτελεί το αρχικό μενού που ξεκινάει η εφαρμογή μας, καθώς η 2η και η 3η σκηνή είναι τα επίπεδα τα οποία πρέπει να περάσουμε για να τερματίσουμε το παιχνίδι μας και κάθε επίπεδο από αυτά έχει διαφορετικούς σκοπούς που πρέπει να πετύχει η ομάδα των παικτών για να ολοκληρωθεί επιτυχώς.

Αρχικά έχουμε το κύριο μενού όπου ξεκινάει το παιχνίδι μας και περιλαμβάνει κουμπιά με τα οποία επιλέγουμε την εκκίνηση (μετάβαση στην 1η πίστα μέσω του "Start New Game") και την έξοδο μας από το παιχνίδι (μέσω του "Exit") όπως φαίνεται στην παρακάτω εικόνα.

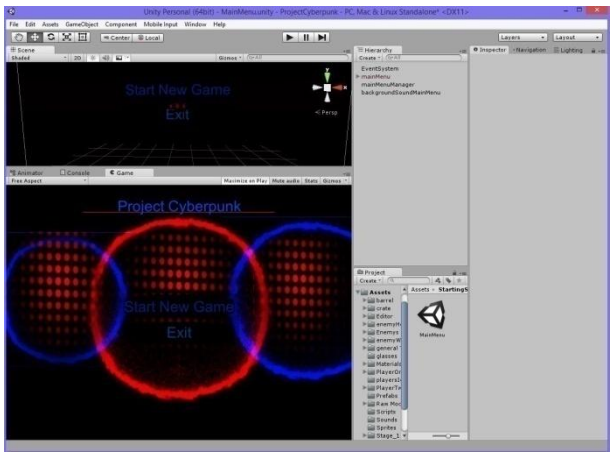

Figure 3.3.2-1: Main Menu

Στο μενού αυτό προσθέσαμε πέρα από τα γραφικά χαρακτηριστικά που περιλαμβάνει και ηχητική πηγή που παίζει αδιάκοπα σαν background ήχος (όπως αντίστοιχα έγινε και στις πίστες μας), καθώς το μενού κατά την εκκίνησή του έχει την δυνατότητα της περιστροφής του background όπως βλέπουμε στις 3 παρακάτω φωτογραφίες.

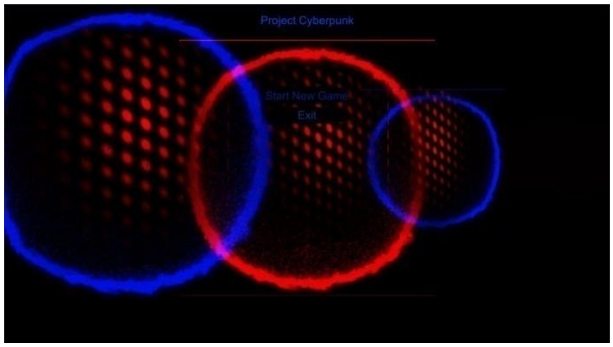

Figure 3.3.2-2:Main Menu Rotation Step 1

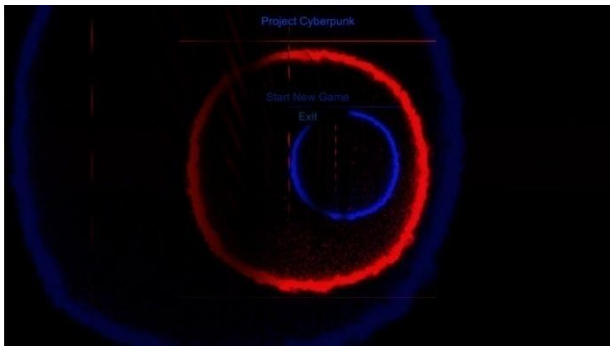

Figure 3.3.2-3:Main Menu Rotation Step 2

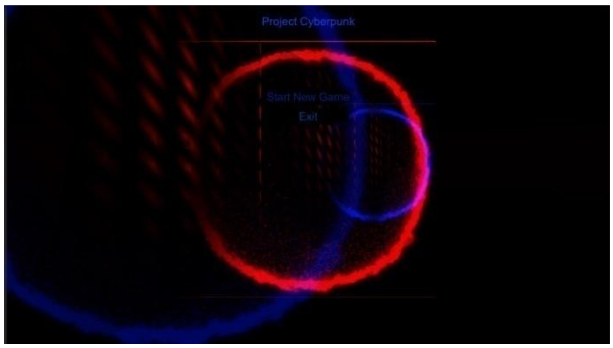

Figure 3.3.2-4: Main Menu Rotation Step 3

Οι τρεις παραπάνω φωτογραφίες αποτελούν τον τρόπο με τον οποίο περιστρέφεται το αρχικό μας μενού κατά την εκκίνηση του παιχνιδιού συνεχόμενα έως ότου επιλέξουμε μια από τις 2 επιλογές (εκκίνηση / έξοδο). Η διαδικασία περιστροφής της εικόνας του background έγινε μέσω κώδικα που επιτρέπει στο μενού να περιστρέφεται συνεχώς για όλο το διάστημα του χρόνου με προκαθορισμένη ταχύτητα ως προς τον άξονα Y (ύψος) .

Στην συνέχεια θα δούμε την μορφή της κάθε πίστας και το στήσιμο των γραφικών στοιχείων που αποτελούνται. Για την 1η πίστα η εκκίνησή μας είναι μέσα σε μια φουτουριστικού τύπου αποθήκη και πρέπει οι ομάδα μας να διασχίσει μια συγκεκριμένη πορεία για να ολοκληρώσει την πίστα, καθώς για κάθε σημείο με κόκκινο πίδακα καπνού αποτελεί σημείο κλειδί που η ομάδα πρέπει να περάσει για να συνεχίσει στο επόμενο. Σε κάθε σημείο που εμφανίζεται ο κόκκινος καπνός όταν το προσεγγίσουμε σε αρκετά κοντινό σημείο ενεργοποιείται μια ομάδα εχθρών που πρέπει να αντιμετωπίσουμε για να μπορέσουμε να προχωρήσουμε. Η διαδικασία αυτή επαναλαμβάνεται μέχρι το τέλος της πίστας όπως θα δούμε στις παρακάτω εικόνες.

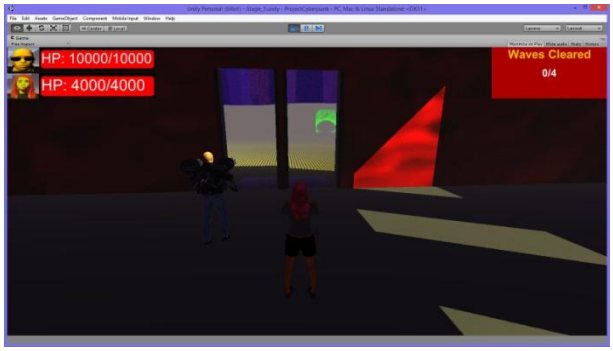

Figure 3.3.2-5:Stage 1 Starting Point

Στην παραπάνω εικόνα παρατηρούμε το σημείο στο οποίο ξεκινάει η ομάδα στο παιχνίδι στο 1ο επίπεδο, ενώ πάνω αριστερά βλέπουμε τις ζωές για το κάθε μέλος της ομάδας μας με βάση την εικόνα αριστερά τους, καθώς πάνω δεξιά βλέπουμε την πρόοδο που έχει γίνει στον σκοπό του επιπέδου αυτού.

Στην επόμενη εικόνα θα δούμε τα σημεία κλειδιά με κόκκινο καπνό που καλείται η ομάδα να περάσει και τις πόρτες που ανοίγονται ανάλογα το σημείο που βρισκόμαστε με πράσινο laser.

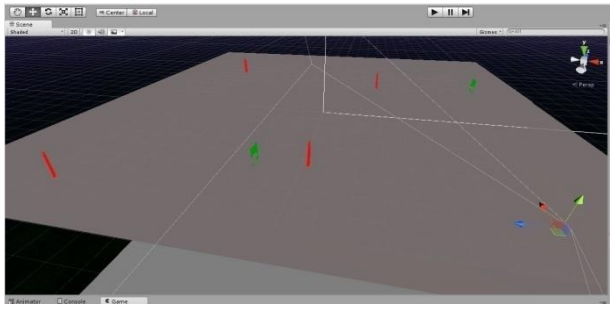
Figure 3.3.2-6: Stage 1 Overview Objective Indicators

Κλείνοντας για την 1η πίστα θα δούμε μερικά από τα τμήματα του γραφικού περιεχομένου και για τα οποία αναφερθήκαμε εκτενώς στο υποκεφάλαιο για το Blender.

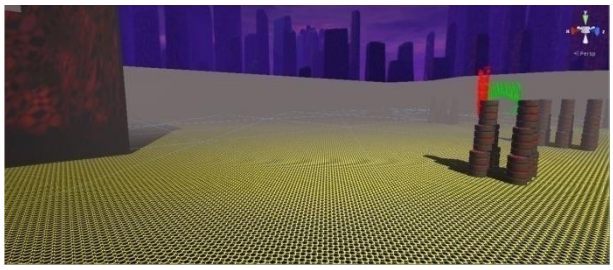
Figure 3.3.2-7:Stage 1 Graphic Content 1

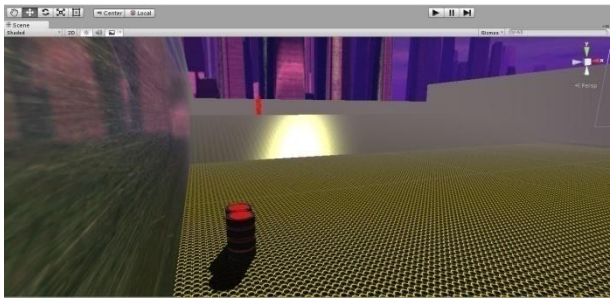

Figure 3.3.2-8:Stage 1 Graphic Content 2

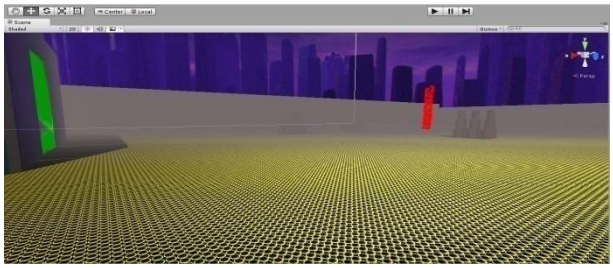

Figure 3.3.2-9:Stage 1 Graphic Content 3

Όμοια θα αναφερθούμε και για την 2η πίστα που ακολουθεί όμοια λογική στο χτίσιμο, αλλά διαφέρει στο κομμάτι της προγραμματιστικής υλοποίησης που θα αναφερθούμε αργότερα. Αρχικά ας εξετάσουμε τα κύρια στοιχεία της πίστας. Αφού ολοκληρωθεί η 1η πίστα τότε καθοδηγούμαστε στην 2η πίστα και το σημείο εκκίνησης βρίσκεται στην παρακάτω εικόνα.

Figure 3.3.2-10:Stage 2 Starting Point

Στην 2η πίστα έχουμε πιο απλό σκοπό από ότι στην 1η, δηλαδή πρέπει η ομάδα να εξολοθρεύσει 50 ή περισσότερους εχθρούς για να ξεκλειδώσει την πόρτα που οδηγεί στο τέλος του παιχνιδιού. Ο σκοπός παρότι είναι απλός απαιτεί έξυπνη διαχείριση της ομάδας για την επιβίωση, κάτι που κ στην 1η πίστα είναι απαραίτητο. Στις παρακάτω φωτογραφίες μπορούμε να δούμε γραφικό περιεχόμενο της πίστας πάνω στην οποία καλούνται οι μονάδες των εχθρών.

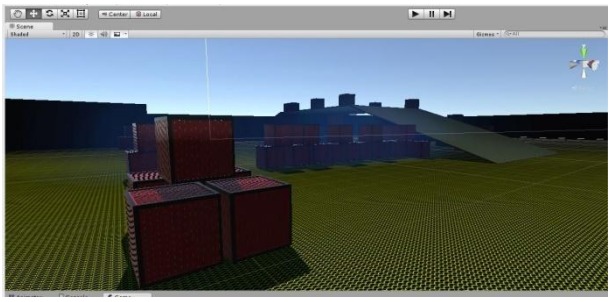

Figure 3.3.2-11:Stage 2 Graphic Content 1

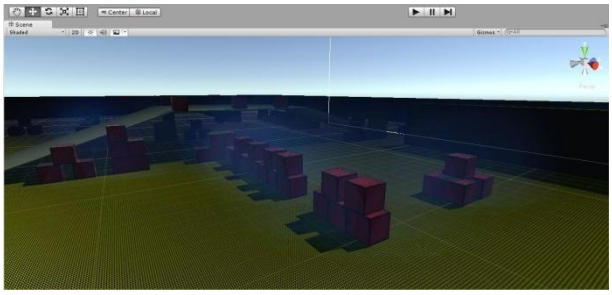

Figure 3.3.2-12:Stage 2 Graphic Content 2

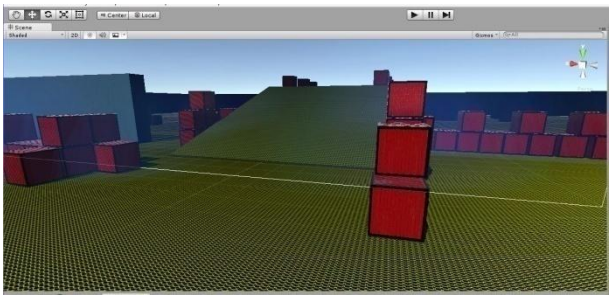

Figure 3.3.2-13:Stage 2 Graphic Content 3

Η παρακάτω εικόνα δείχνει 2 πίδακες κόκκινου καπνού που ενεργοποιούνται όταν η ομάδα τηρεί τα κριτήρια για να τερματίσει το παιχνίδι στην συγκεκριμένη πίστα, καθώς όπως και στην 1η πίστα πρέπει να διασχίσει στην περιοχή της πόρτας για να ολοκληρωθεί η πίστα.

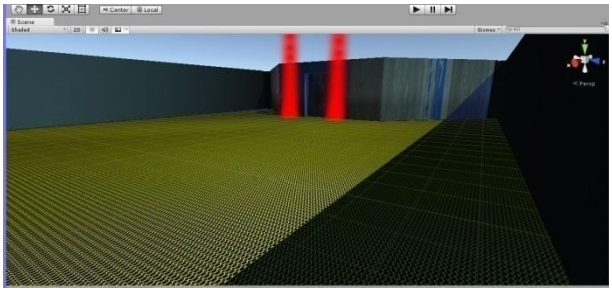

Figure 3.3.2-14: Stage 2 Ending Point Trigger

Η διεπαφή (U.I / User Interface) και στις 2 πίστες περιλαμβάνει όπως φαίνεται στις παρακάτω εικόνες μενού παύσης, τερματισμού και αποτυχίας με τις κατάλληλες επιλογές αντίστοιχα.

Figure 3.3.2-15: Pause Menu

Figure 3.3.2-16:Success Popup Menu

Figure 3.3.2-17:Game Over Menu

Ο προγραμματισμός των μενού αυτών και γενικότερα της διεπαφής του παιχνιδιού έγινε συνδυαστικά χρησιμοποιώντας πληροφορίες μέσα από την ιστοσελίδα της Unity, άλλα και βοήθεια μέσω της γνωστής ιστοσελίδας youtube (Στο τέλος του εγγράφου εμπεριέχονται πληροφορίες στο τμήμα "Βιβλιογραφία" με υπερσύνδεσμο από κάθε σημείο που χρειάστηκε να αντλήσουμε πληροφορίες για την κατασκευή της). Ο κώδικας που δημιουργήσαμε για την κατασκευή της λειτουργικότητας της διεπαφής φαίνεται στις παρακάτω εικόνες και αποτελεί συνδυασμό των βοηθητικών υλικών, αλλά και επινόησης συνδυασμών από την μεριά μας.

```
public void resume(){
    pauseMenu.gameObject.SetActive(false);
    Cursor.visible = false;
    activePlayer.GetComponent<player_inactive_Movement>().enabled=true;
    Time.timeScale=1f;
}
public void returnToMainMenu(){
    Time.timeScale = 1f;
    Application.LoadLevel ("MainMenu");
}
public void exitGame(){
    System.Diagnostics.Process.GetCurrentProcess().Kill();//works for export .Comment this line when working with editor.
}
public void retry(){
    Application.LoadLevel (Application.loadedLevelName);
}
public void continueGame(){
    if (Application.loadedLevelName == "Stage_1") {
        Application.LoadLevel("Stage_2");
    }
    if (Application.loadedLevelName == "Stage_2") {
        Application.LoadLevel("MainMenu");
    }
}
```

Figure 3.3.2-18: UI Functions

Ο παραπάνω κώδικας αφορά την λειτουργικότητα των συναρτήσεων στις επιλογές των μενού.

```
27      void Update () {
28          //check for playerOne to focus pause,death,success
29          if (GameObject.Find ("humanMake_Model_One")!=null) {
30              if(GameObject.Find ("humanMake_Model_One").GetComponent<NavMeshAgent>().isActiveAndEnabled==false)
31              activePlayer=GameObject.Find("humanMake_Model_One");
32          }
33          //check for playerTwo to focus pause,death,success
34          if (GameObject.Find ("player2")!=null) {
35              if(GameObject.Find ("player2").GetComponent<NavMeshAgent>().isActiveAndEnabled==false)
36                  activePlayer=GameObject.Find("player2");
37          }
38          //pause menu.
39          if (Input.GetKeyDown (KeyCode.Escape) && activePlayer!=null) {
40              if (pauseMenu.gameObject.activeInHierarchy == false){
41                  Cursor.visible = true;
42                  pauseMenu.gameObject.SetActive (true);
43                  Time.timeScale=0f;
44                  activePlayer.GetComponent<player_inactive_Movement>().enabled=false;
45              }
46              else{
47                  Cursor.visible = false;
48                  pauseMenu.gameObject.SetActive(false);
49                  activePlayer.GetComponent<player_inactive_Movement>().enabled=true;
50                  Time.timeScale=1f;
51              }
52          }
53          //game over menu.
54          if (activePlayer == null) {
55              Cursor.visible = true;
56              gameOverPopUp.gameObject.SetActive (true);
57          }
58          //success stage menu.
59          //case stage 1
60          if(Application.loadedLevelName=="Stage_1"){
61              if(stageManagerStatus.GetComponent<stageOneManager>().countWaves==4){
62                  Cursor.visible = true;
63                  winPopUp.gameObject.SetActive(true);
64              }
65          }
66          //case stage 2
67          if(Application.loadedLevelName=="Stage_2"){
68              if(guardsKilled>=50){
69                  if(exitDoorLeft.activeSelf==false && exitDoorRight.activeSelf==false){
70                      Cursor.visible = true;
71                      winPopUp.gameObject.SetActive(true);
72                  }
73              }
74          }
75      }
```

Figure 3.3.2-19:UI Popup Manager

Ο παραπάνω κώδικάς αφοράει την διαχείριση των μενού από τον ενεργό παίκτη, καθώς και global μεταβλητές της σκηνής που αφορούν τον κέρσορα και την χρονική κλίμακα.

Το κομμάτι που αφοράει τις μπάρες ζωής των παικτών και εχθρών βρίσκεται στις παρακάτω εικόνες.

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

public class playerOneHealth : MonoBehaviour {
    public int health;//current health
    public float initHealth;//starting health
    private string enemyName;
    private GameObject aliveState;

    void Start () {
        if (Application.loadedLevelName == "Stage_1") {
            health = 10000;
            initHealth = health;//total health for UI purpose.
        }
        if (Application.loadedLevelName == "Stage_2") {
            health = 10000;
            initHealth = health;//total health for UI purpose.
        }
        enemyName = "";
        aliveState = GameObject.Find ("player_Manager");
    }
    void Update () {
        if (health <= 0) {
            aliveState.GetComponent<genPlayerManager>().setAlivePlOne=false;
            Destroy(GameObject.Find("humanMake_Model_One"));
        }
    }
    void OnCollisionEnter(Collision col){
        if (col.gameObject.tag == "enemyBullet") {
            health-=2; //apply dmg each time being hit.
            GameObject dmg=GameObject.Find("playerOneHealthBar");
            dmg.GetComponent<Image>().fillAmount-=2/initHealth;
            enemyName=col.gameObject.GetComponent<enemyBullet>().getHitName(); //gets the enemy name
        }
    }
    public GameObject getEnemyByHit(){
        GameObject enemyByHit=GameObject.Find(enemyName);
        return enemyByHit;
    }
}
```

Figure 3.3.2-20:Player1 Health Part 1

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

public class playerOneLogo : MonoBehaviour {
    private string health;
    private GameObject playerOne;
    private playerOneHealth initHealth;
    private Text msg;
    void Start () {
        msg = GameObject.Find ("playerOneHealthLogo").GetComponent<Text>();
        playerOne = GameObject.Find ("humanMake_Model_One"); //get Object of reference
        initHealth = playerOne.GetComponent<playerOneHealth> (); //get Starting health.
        health="HP: "+initHealth.health+"/"+"";
        msg.text = health;
    }
    void Update () {
        if (playerOne != null) {
            health = "HP: " + initHealth.health + "/"+initHealth.initHealth;
            msg.text = health;
        }
        if (playerOne == null) {
            health = "HP:0/10000";
            msg.text = health;
        }
    }
}
```

Figure 3.3.2-21:Player1 Health Part2

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

public class playerTwoHealth : MonoBehaviour {
    public int health; //current health
    public float initHealth;//starting health
    private string enemyName;
    private GameObject aliveState;

    void Start () {
        if (Application.loadedLevelName == "Stage_1") {
            health = 4000;
            initHealth = health;
        }
        if (Application.loadedLevelName == "Stage_2") {
            health = 4000;
            initHealth = health;//total health for UI purpose.
        }
        enemyName = "";
        aliveState = GameObject.Find ("player_Manager");
    }
    void Update () {
        if (health <= 0) {
            aliveState.GetComponent<genPlayerManager>().setAlivePlTwo=false;
            Destroy(GameObject.Find("player2"));
        }
    }
    void OnCollisionEnter(Collision col){
        if (col.gameObject.tag == "enemyBullet") {
            health-=3; //apply dmg each time being hit.
            GameObject dmg=GameObject.Find("playerTwoHealthBar");
            dmg.GetComponent<Image>().fillAmount-=3/initHealth;
            enemyName=col.gameObject.GetComponent<enemyBullet>().getHitName(); //gets the enemy n
        }
    }
    public GameObject getEnemyByHit(){
        GameObject enemyByHit=GameObject.Find(enemyName);
        return enemyByHit;
    }
}
```

Figure 3.3.2-22:Player2 Health Part1

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

public class playerTwoLogo : MonoBehaviour {
    private string health;
    private GameObject playerTwo;
    private playerTwoHealth initHealth;
    private Text msg;
    void Start () {
        msg = GameObject.Find ("playerTwoHealthLogo").GetComponent<Text>();
        playerTwo = GameObject.Find ("player2"); //get Object of reference
        initHealth = playerTwo.GetComponent<playerTwoHealth> (); //get Starting health.
        health="HP: "+initHealth.health+"/"+"";
        msg.text = health;
    }
    void Update () {
        if (playerTwo != null) {
            health = "HP: " + initHealth.health + "/"+initHealth.initHealth;
            msg.text = health;
        }
        if (playerTwo == null) {
            health = "HP:0/4000";
            msg.text = health;
        }
    }
}
```

Figure 3.3.2-23:Player2 Health Part2

Οι 4 παραπάνω φωτογραφίες αποτελούν υλοποιήσεις για την διαχείριση των ζωών, δηλαδή τις μεταβολές και τις αλλαγές στην διεπαφή μας που αφορούν την ομάδα των παικτών.

Ομοίως δουλέψαμε και στους εχθρούς μας το σύστημα ζωής τους στα πλαίσια της διεπαφής όπως φαίνεται παρακάτω στις εικόνες που περιέχουν τον κώδικά μας.

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

public class enemyGeneralHealth : MonoBehaviour {
    private Vector3 lookPlayer;//look the player.
    private GameObject healthBar;//object to look the player.
    private GameObject activePlayer;//active gameobject of player unit.
    public GameObject visualHealth; //object where you refer to.
    private Image visualHealthImg; //component you need to access to increase/decrease hp.
    private float maxHp;
    void Start () {
        healthBar = this.gameObject;
        visualHealthImg = visualHealth.GetComponent<Image> (); //init full health.
    }
    void Update () {
        //look at Player 1 if active.
        if (GameObject.Find("humanMake_Model_One")!=null) {
            if(GameObject.Find("humanMake_Model_One").GetComponent<NavMeshAgent> ().isActiveAndEnabled==false){
                activePlayer=GameObject.Find("humanMake_Model_One"); //active player per frame.
                lookPlayer=activePlayer.transform.position;//location of player per frame.
                healthBar.transform.LookAt(lookPlayer);
            }
        }
        //look at Player 2 if active.
        if (GameObject.Find("player2")!=null) {
            if(GameObject.Find("player2").GetComponent<NavMeshAgent> ().isActiveAndEnabled==false){
                activePlayer=GameObject.Find("player2"); //active player per frame.
                lookPlayer=activePlayer.transform.position;//location of player per frame.
                healthBar.transform.LookAt(lookPlayer);
            }
        }
    }
    public void setDamageTaken(){
        visualHealthImg.fillAmount -= (1 / maxHp);
    }
    public void setHealingTaken(){
        visualHealthImg.fillAmount += (5 / maxHp);
    }
    public void setMaxHP(float hp){
        maxHp = hp;
    }
}
```

Figure 3.3.2-24:Enemy General, Lieutenant, Guard Health

Η παραπάνω φωτογραφία αποτελεί τον κώδικα που γράφτηκε για την κατάσταση ζωής όλων των επιθετικών μονάδων του εχθρού και λειτουργεί αποκλειστικά για την διεπαφή στην 1η πίστα.

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

public class enemyHealerHealth : MonoBehaviour {
    private Vector3 lookPlayer;//look the player.
    private GameObject healthBar;//object to look the player.
    private GameObject activePlayer;//active gameobject of player unit.
    public GameObject visualHealth; //object where you refer to.
    private Image visualHealthImg; //component you need to access to increase/decrease hp.
    private float maxHp;
    void Start () {
        healthBar = this.gameObject;
        visualHealthImg = visualHealth.GetComponent<Image> (); //init full health.
    }
    void Update () {
        //look at Player 1 if active.
        if (GameObject.Find ("humanMake_Model_One") != null) {
            if (GameObject.Find ("humanMake_Model_One").GetComponent<NavMeshAgent> ().isActiveAndEnabled == false) {
                activePlayer = GameObject.Find ("humanMake_Model_One"); //active player per frame.
                lookPlayer = activePlayer.transform.position;//location of player per frame.
                healthBar.transform.LookAt (lookPlayer);
            }
        }
        //look at Player 2 if active.
        if (GameObject.Find ("player2") != null) {
            if (GameObject.Find ("player2").GetComponent<NavMeshAgent> ().isActiveAndEnabled == false) {
                activePlayer = GameObject.Find ("player2"); //active player per frame.
                lookPlayer = activePlayer.transform.position;//location of player per frame.
                healthBar.transform.LookAt (lookPlayer);
            }
        }
    }
    public void setDamageTaken(){
        visualHealthImg.fillAmount -= (1 / maxHp);
    }
    public void setMaxHP(float hp){
        maxHp = hp;
    }
}
```

Figure 3.3.2-25:Enemy Healer Health

Αντίστοιχα και εδώ έχουμε τον κώδικα που αφορά την συμπεριφορά της διεπαφής στον θεραπευτή της αντίπαλης ομάδας, που η μόνη διαφορά είναι ότι δεν υπάρχει δυνατότητα επούλωσης ζημιάς στον εαυτό του παρά μόνο στα μέλη της ομάδας του.

Η παρακάτω εικόνα αντιπροσωπεύει το κομμάτι της διεπαφής των εχθρών, δηλαδή τις μπάρες ζωής για τις οποίες γράψαμε τον κώδικα στις παραπάνω εικόνες.

Figure 3.3.2-26:Enemy Health Bars

3.3.3 Εισαγωγή των Μοντέλων

Σε αυτό εδώ το κομμάτι θα εξετάσουμε την διαδικασία που έγινε για να εισαγάγουμε τα μοντέλα των παικτών, αντιπάλων, αλλά και των αντικειμένων που δημιουργήθηκαν στο Blender και αποτελούν ένα μεγάλο μέρος του γραφικού στα επίπεδα που δημιουργήσαμε.

Η διαδικασία ξεκίνα μεταφέροντας τα αρχεία του Blender και στην περίπτωση του male παίκτη μας το αρχείο ".fbx" μέσα στο φάκελο Assets που αναφέραμε παραπάνω ότι απαιτείται για την διαχείριση των αντικειμένων μας συνοδευόμενα πάντα με τις υφές (UV / Textures) που ασχοληθήκαμε στο υποκεφάλαιο "3.2" (Χρήση του Blender).

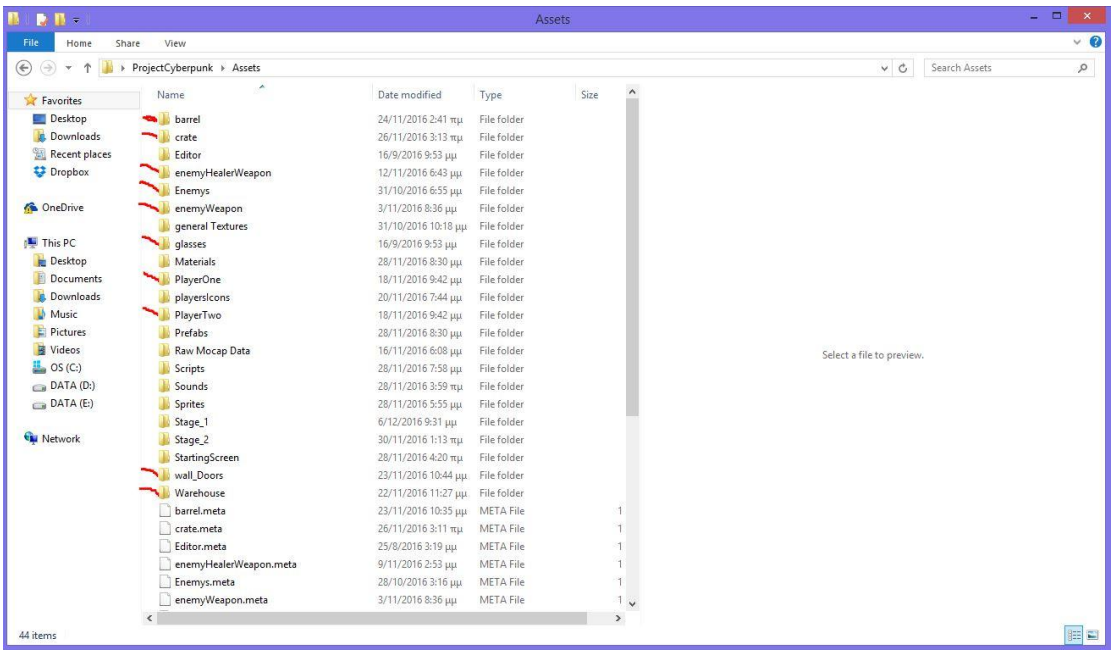

Figure 3.3.3-1: Import Files in Unity from File Explorer

Στην παραπάνω φωτογραφία βλέπουμε δεξιά από τα κόκκινα σημάδια τα αρχεία που φέραμε από Blender και MakeHuman χωρισμένα σε ξεχωριστά αρχεία για κάθε αντικείμενο με όλα τα απαραίτητα αρχεία.

Το επόμενο βήμα μας είναι να ανοίξουμε την Unity η οποία θα ανανεώσει τα αρχεία του Project μας προσθέτοντας τα καινούργια αρχεία . Αφού γίνει αυτή η διαδικασία επιλέγουμε κάθε φάκελο μέσα στο φάκελο Assets στην καρτέλα Project και για κάθε αρχείο που κάναμε import η Unity δημιούργησε έναν φάκελο που ονομάζεται Materials για το οποίο θα αναφερθούμε αργότερα.

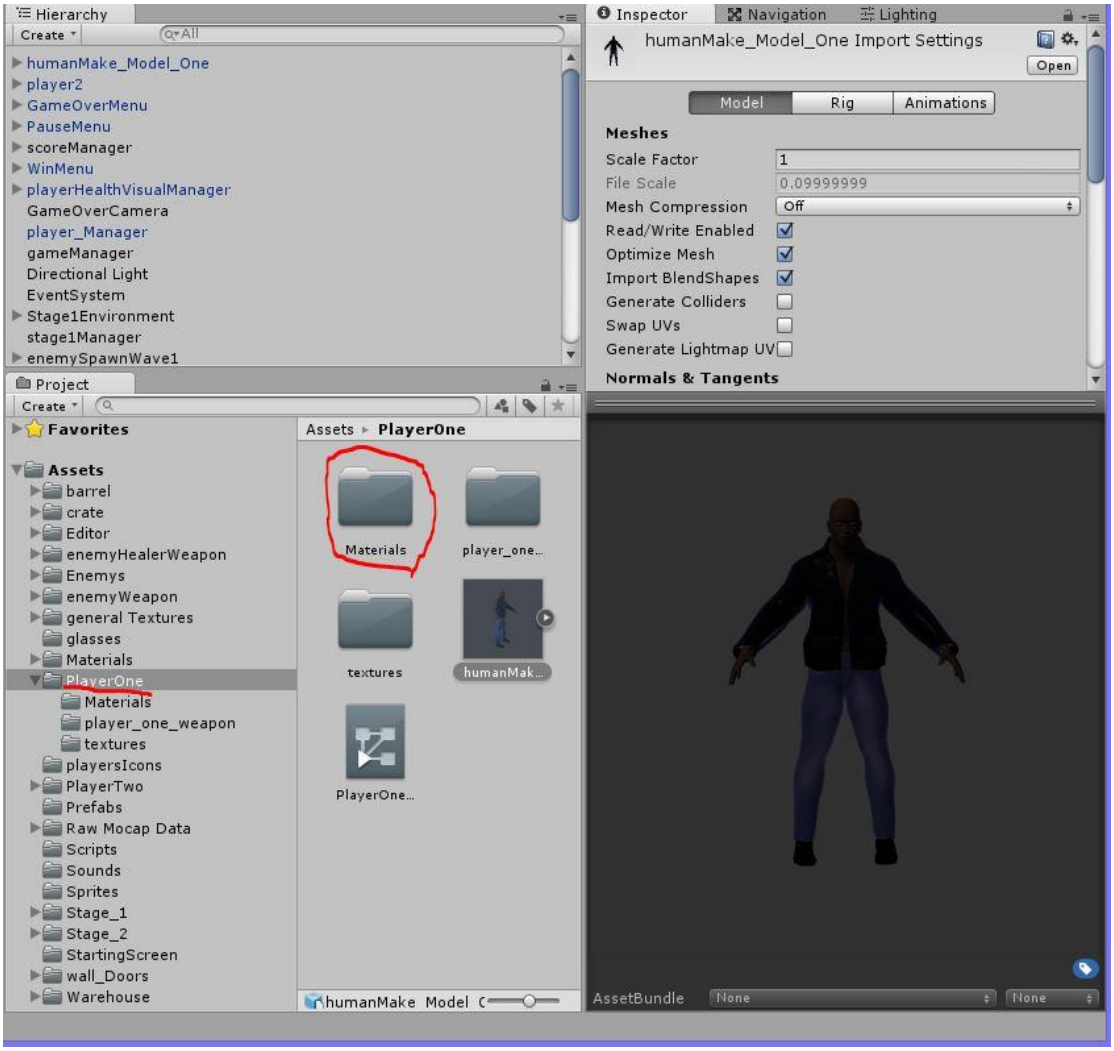

Figure 3.3.3-2: Imported File inside Unity

Η παραπάνω εικόνα αποτελεί ένα παράδειγμα των αρχείων που κάναμε import στο project μας. Η κόκκινη γραμμή στην καρτέλα του Project υποδεικνύει το αρχείο που έχουμε επιλέξει και ο φάκελος δεξιά μέσα στο κόκκινο περίγραμμα (Materials) το αρχείο που δημιουργεί αυτόματα η Unity που περιέχει όλες τις υφές που φέραμε μαζί με το μοντέλο τροποποιημένες για να εφαρμοστούν επάνω του. Φυσικά έχουμε την επιλογή να τροποποιήσουμε τον Shader στο Material ώστε να μας δώσει κάποιο διαφορετικό οπτικό αποτέλεσμα. Η λίστα με τα material που παράχθηκε για αυτό το μοντέλο που θέσαμε ως παράδειγμα φαίνεται στην ακόλουθη εικόνα.

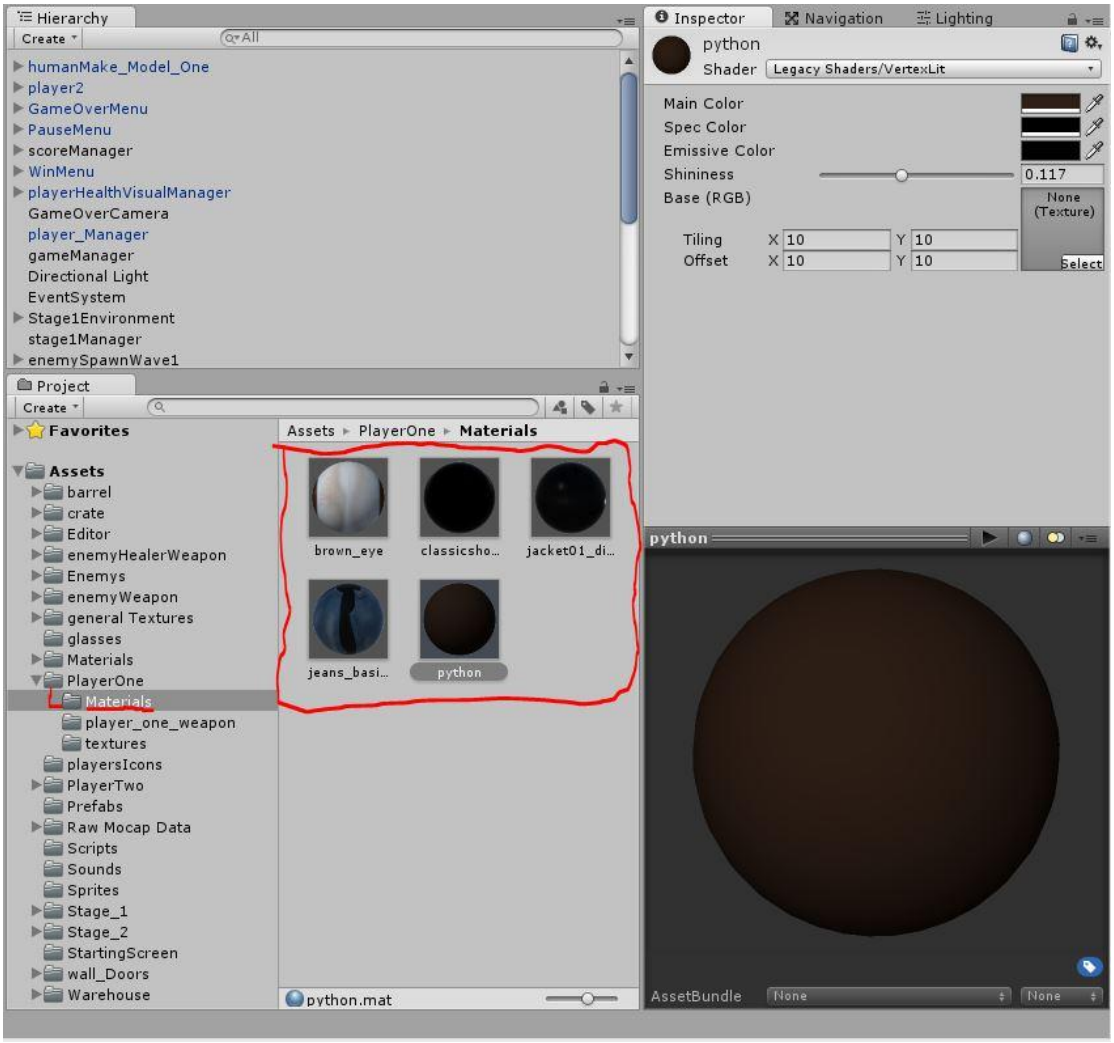

Figure 3.3.3-3: Material Folder inside Model Folder

3.3.4 Animation Controller και Διαχείριση κινήσεων

Στο παραπάνω υποκεφάλαιο εξηγήσαμε την φόρτωση των μοντέλων μέσα στην Unity. Τα στατικά αντικείμενα μας δεν χρειάστηκαν κάποια περαιτέρω επέμβαση στο κομμάτι του animation. Οι χαρακτήρες της ομάδας και των εχθρών αποτελούν το βασικό κινητό μέρος του βιντεοπαιχνιδιού. Σε αυτό το σημείο πρέπει να εξηγήσουμε την προετοιμασία και την εφαρμογή των animations στα μοντέλα μας.

Αρχικά κάθε μοντέλο που περιέχει animation που σκοπεύουμε να διαχειριστούμε μέσα στην σκηνή πρέπει να έχει το component "Animator" . Ο Animator για να γίνει λειτουργικός πάνω στο αντικείμενο που υπάγεται χρειάζεται ένα αντικείμενο (που συνήθως για λόγους αποφυγής αταξίας δημιουργούμε μέσα στον φάκελο που βρίσκεται το αντικείμενο που εισαγάγαμε) που ονομάζεται "Animator Controller".

Προτού όμως εξηγήσουμε την χρήση του Animator θα πρέπει να ελέγξουμε για τυχόν προβλήματα στην δομή του σκελετού του αντικείμενου. Για να το πετύχουμε αυτό επιλέγουμε το αντικείμενο που επιθυμούμε να ελέγξουμε και επιλέγουμε το κουμπί Rig που μας εμφανίζει καινούργιες επιλογές όπως βλέπουμε στην παρακάτω εικόνα.

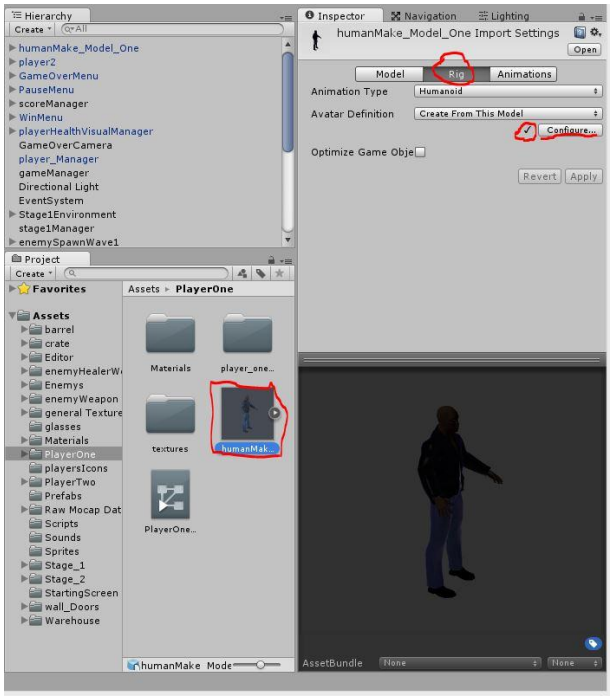

Figure 3.3.4-1: Humanoid Rig Part 1

Όπως βλέπουμε στα δεξιά της εικόνας το πατημένο κουμπί "Rig" επιλέγουμε το κουμπί "Configure" έχοντας θέσει το "Animation Type" σε Humanoid για να μεταβούμε στο mecanim της Unity που μας βοηθάει να ρυθμίσουμε σωστά το Rig μας για να μπορέσουμε να μεταβούμε στην διαδικασία να προσθέσουμε animations. Στην παρακάτω εικόνα βλέπουμε την νέα οθόνη που μας δημιουργεί η Unity για να ελέγξουμε αν υπάρχουν λάθη στην αυτόματη χαρτογράφηση της σκελετικής ιεραρχίας που έχει γίνει.

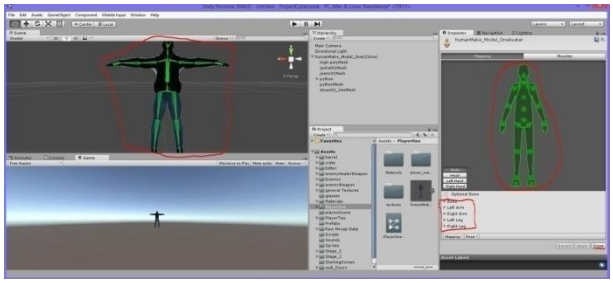

Figure 3.3.4-2:Humanoid Rig Part 2

Στην δεξιά πλευρά της οθόνης βλέπουμε ένα πράσινο ανθρωποειδές, ενώ λίγο πιο κάτω βλέπουμε ιεραρχίες που περιλαμβάνουν τα οστά που αντιστοίχησε η Unity και ελέγχουμε για τυχόν λανθασμένη αντιστοίχηση. Εφόσον όλα είναι σωστά παρατηρούμε και την αριστερή πλευρά της οθόνης στην οποία βλέπουμε την σκελετική διάταξη οπού σε περίπτωση οστού που δεν βρέθηκε το σημείο που λείπει χρωματίζεται με κόκκινο χρώμα.

Εφόσον δεν παρατηρούμε κανένα πρόβλημα επιλέγουμε το κουμπί "Done" για να επιστρέψουμε στην προηγούμενη κατάσταση της εργασίας μας. Τοποθετούμε τον Animator Controller που δημιουργήσαμε στον Animator του αντικειμένου που θέλουμε να ορίσουμε τα animations. Έπειτα ανοίγουμε τον Animator Controller μας και βλέπουμε την ακόλουθη εικόνα

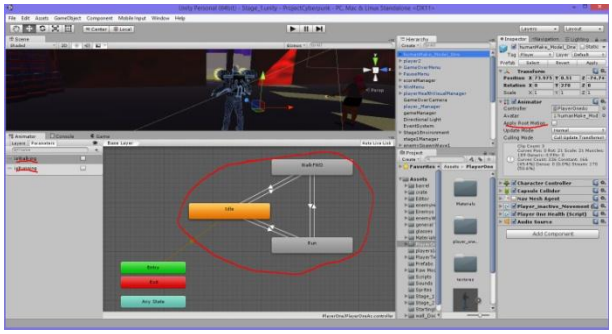

Figure 3.3.4-3:Humanoid RigPart 3

Σε αυτό το σημείο να αναφέρουμε πως χρησιμοποιήσαμε τα animation clips από το πακέτο της Unity μέσω του Asset Store (Raw Mocap Data) που περιέχει διαφόρων είδη animation (walk, run ,idle ,interact etc.). Από αυτά τα animations χρησιμοποιήσαμε 3 animations τα οποία ήταν κοινά για την ομάδα των αντιπάλων και για τον παίκτη με τα διπλά miniguns. Στον συγκεκριμένο animation controller προσθέσαμε τα 3 clips (walk, run, idle) και τα βελάκια μεταξύ των clips υποδεικνύουν τις εναλλαγές μεταξύ τους και περιέχουν συνθήκες εναλλαγών που βρίσκονται στο αριστερό μέρος της οθόνης . Έχουμε λοιπόν 2 κριτήρια τύπου Boolean (isWalking, isRunning) τα οποία όπως θα δούμε σε παρακάτω υποκεφάλαιο καθορίζουν το πότε θα γίνεται η μετάβαση από ένα animation σε ένα άλλο.

Για τον 2ο παίκτη (female) που εξηγήσαμε πως ήταν απαραίτητο να δημιουργήσουμε από την αρχή τα animations διότι δεν υπήρχαν κατάλληλα για τις ανάγκες μας ακολουθήθηκε η ίδια ακριβώς διαδικασία με την μόνη διαφορά ότι τα animation βρίσκονταν μέσα στα αρχεία του μοντέλου μας όπως φαίνεται στην παρακάτω εικόνα.

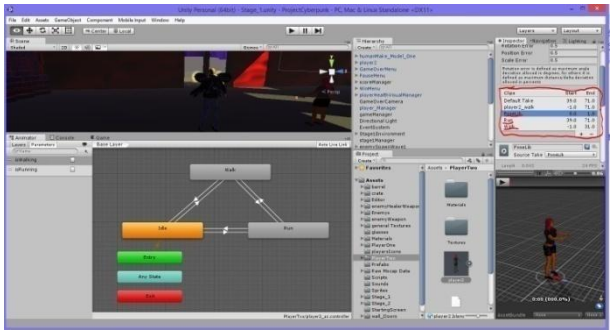

Figure 3.3.4-4: Humanoid Rig Part 4 (Player 2)

Στο κόκκινο περίγραμμα βλέπουμε με υπογράμμιση τα animations που χρησιμοποιήσαμε στα πλαίσια κατασκευής του 2ου παίκτη τα οποία καθόρισαν την οπτική της προγραμματιστικής συμπεριφοράς που θα αναφερθούμε αργότερα.

3.3.5 Υλικά σε αντικείμενα

Τα υλικά (Materials) αποτελούν τον βασικό παράγοντα που καθορίζει την ποιότητα εμφάνισης των γραφικών μας στην σκηνή. Γενικότερα τα υλικά όπως αναφέραμε και σε παραπάνω κεφάλαιο αποτελούνται από την υφή (texture) και τον Shader, δηλαδή τον τρόπο που επιδρά το φως πάνω σε μία επιφάνεια. Ο συνδυασμός αυτών των 2 καθορίζει την βασική δομή του material. Πέρα από τα βασικά αυτά χαρακτηριστικά υλικό θεωρείται και ένα απλό χρώμα ή συνδυασμός χρώματος και υφής . Υπάρχουν πάρα πολλά είδη Shader που μπορούν να εφαρμοστούν σε material, καθώς υπάρχει και η δυνατότητα να γράψει κάποιος πηγαίο κώδικα μέσα από την Unity για να υλοποιήσει την εμφάνιση στις δικές του ανάγκες.

Στα πλαίσια της εργασίας αυτής δεν κρίθηκε απαραίτητο να δημιουργήσουμε τους δικούς μας Shaders και επιλέξαμε κάποιους από αυτούς που προσφέρει η Unity όπως θα δούμε στις παρακάτω εικόνες για να δημιουργήσουμε τα αντίστοιχα υλικά (materials).

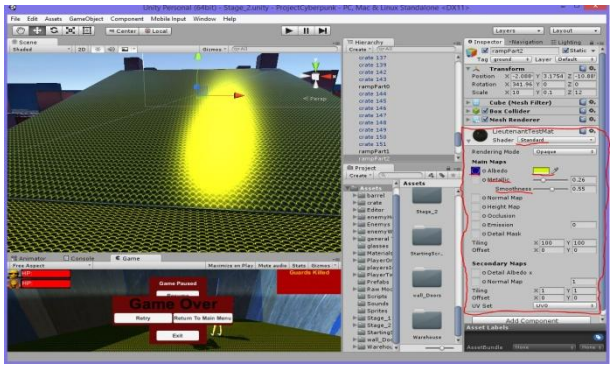

Figure 3.3.5-1:Standard Shader Material

Στην παραπάνω εικόνα βλέπουμε έναν από τους βασικούς Shaders της game engine ο οποίος περιέχει μια υφή με επιλογή απόχρωσης του κίτρινου χρώματος και 2 ιδιότητες που περιγράφουν το πόσο μεταλλικό φαίνεται, αλλά και την εξομάλυνση που επιθυμούμε. Ρυθμίζοντας αυτές τις ιδιότητες παίρνουμε διαφορετικά αποτελέσματα και αλλάζουμε συνεχώς αυτές τις ρυθμίσεις έως ότου μείνουμε ικανοποιημένοι. Στο πεδίο του material να αναφέρουμε πως η επιλογή "Tiling" αλλάζοντας τη μας δίνει την υφή που επιλέξαμε πολλαπλασιασμένη κατά X και Y άξονες και με αυτό τον τρόπο δημιουργήσαμε το υλικό που φαίνεται στην σκηνή τόσο για την ράμπα, αλλά και για το έδαφος.

Μερικά ακόμα παραδείγματα αυτού του είδους υλικού φαίνονται στις παρακάτω εικόνες.

Figure 3.3.5-2:Standard Shader Material Part 2

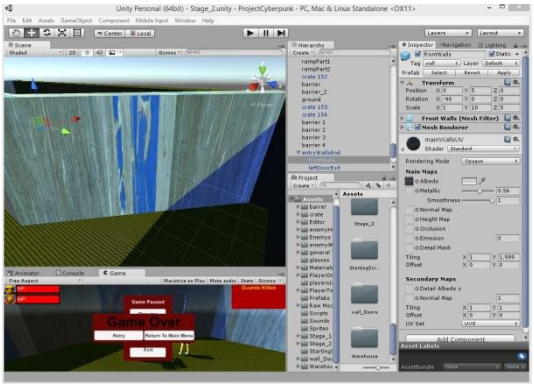

Figure 3.3.5-3:Standard Shader Material Part 3

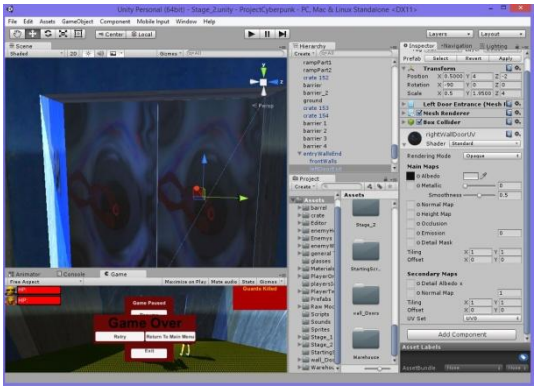

Figure 3.3.5-4:Standard Shader Material Part 4

Στα πλαίσια της εργασίας μας χρησιμοποιήσαμε και άλλους Shaders όπως τον "Legacy Shader/ Bumped Specular", ο οποίος μας δίνει την δυνατότητα να έχουμε μία υφή ως βάση και άλλη μία υφή για την αντανάκλαση του φωτισμού με δυνατότητα χρωματικής αλλαγής, καθώς εδώ έχουμε και την δυνατότητα επιλογής στο χρώμα αντανάκλασης επάνω στην επιφάνεια (specular color) και με την επιλογή "Shininess" ορίζουμε την λαμπρότητα της αντανάκλασης. Παράδειγμα της χρήσης του φαίνεται στην ακόλουθη εικόνα.

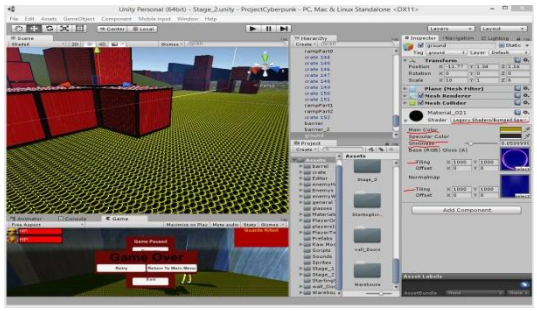

Figure 3.3.5-5:Legacy Shader / Bumped Specular Material

Το υλικό αυτό χρησιμοποιήθηκε για να ορίσουμε την υφή του εδάφους μας και στις 2 πίστες. Με τον ίδιο Shader ορίσαμε αντίστοιχα και κάποια από τα γραφικά των παικτών μας κυρίως σε ρούχα, όπως για παράδειγμα στο παντελόνι της παίκτριας μας.

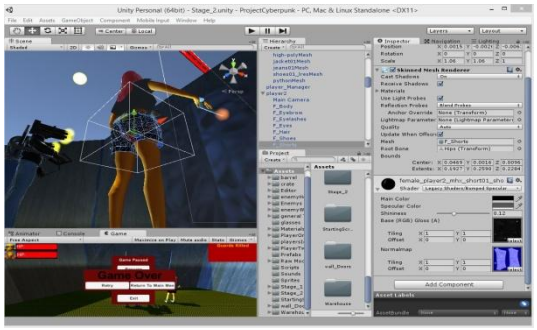

Figure 3.3.5-6:Legacy Shader / Bumped Specular Material2

Επιπλέον χρησιμοποιήσαμε το είδος Shader: "Legacy Shader / Diffuse" που είναι αρκετά όμοιος με τον παραπάνω με την μόνη βασική διαφορά ότι δεν περιλαμβάνει την έννοια της αντανάκλασης του φωτός και το χρώμα αυτού, ενώ χρησιμοποιεί ένα βασικό χρώμα και μια υφή. Η παρακάτω εικόνα αποτελεί ένα παράδειγμα της χρήσης του και το αποτέλεσμα που προσφέρει .

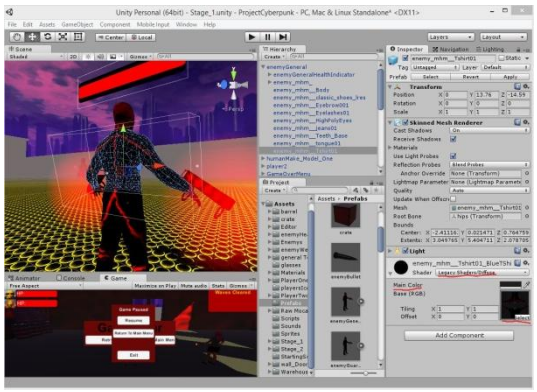

Figure 3.3.5-7:Legacy Shader / Diffuse Material

Στο σημείο αυτό να επισημάνουμε πως για τα αντικείμενα που εισήγαμε από το Blender χρησιμοποιήθηκαν οι υφές που αναλύσαμε σε παραπάνω κεφάλαιο.

3.3.6 Τεχνητή Νοημοσύνη καθοδήγησης κίνησης(Navmesh Agent)

Στα πλαίσια της εργασίας μας ήταν απαραίτητη η δημιουργία τεχνητής νοημοσύνης τόσο στην ομάδα των παικτών μας όσο και στους αντιπάλους μας. Για κάθε αντικείμενο που φέρει το component "Navmesh Agent" έχει την δυνατότητα να μπορεί να χρησιμοποιήσει το περιβάλλον στο οποίο δρα με βάση κάποιες ρυθμίσεις που πρέπει να γίνουν πιο μπροστά όπως θα δούμε στις παρακάτω εικόνες.

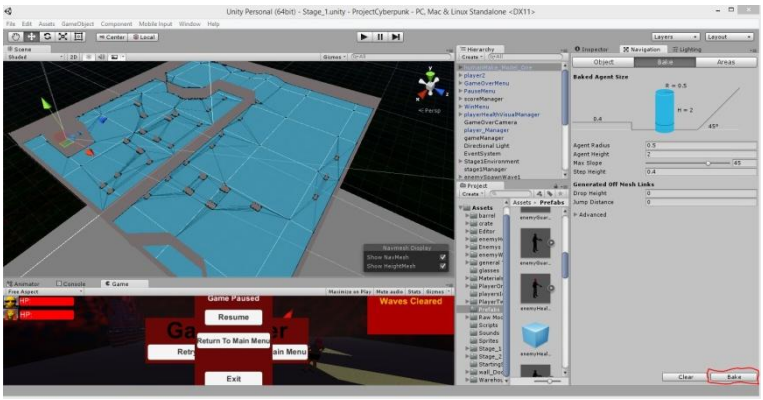

Figure 3.3.6-1:Navmesh Area

Στην Unity υπάρχει κάτι που ονομάζεται Navmesh και αποτελεί τον χώρο που μπορεί να κινηθεί κάθε μονάδα που έχει το Navmesh Agent component επάνω της. Όπως φαίνεται στην παραπάνω εικόνα το γαλάζιο χρώμα αντιπροσωπεύει την περιοχή στην οποία μπορεί να κινηθεί μια τέτοια μονάδα, ενώ η γκρίζα περιοχή αντίστοιχα υποδεικνύει την περιοχή που δεν μπορεί να διασχίσει. Τις γκρίζες ζώνες σε μια οποιαδήποτε πίστα μπορούμε να τις δημιουργήσουμε επιλέγοντας κάθε στοιχείο και κάνοντας tick στο κουτάκι "Static" για να υποδείξουμε ότι δεν πρόκειται για κινητό μέρος. Αφού ολοκληρώσουμε αυτή την διαδικασία για κάθε αντικείμενο που επιθυμούμε μέσα από την καρτέλα στα δεξιά της οθόνης στην καρτέλα "Navigation" επιλέγουμε στο κουμπί "Bake". Αφού γίνει αυτό πλέον κάθε μονάδα γνωρίζει την περιοχή που επιτρέπεται να κινηθεί.

Για την ομάδα των παικτών η τεχνητή νοημοσύνη αποσκοπούσε να πετύχει σε αποτέλεσμα ομαδικότητας. Η ομάδα των παικτών για κάθε μονάδα που δεν ελέγχει ο παίκτης έχει προγραμματιστεί κατάλληλη συμπεριφορά έτσι ώστε ο ενεργός παίκτης να μπορεί να δώσει διαταγή για επίθεση ή για οπισθοχώρηση, ενώ όταν δεν υπάρχει κάποιος εχθρός ο μη χειριζόμενος παίκτης ακολουθάει τον ενεργό παίκτη.

Το παιχνίδι επίσης μας επιτρέπει να εναλλάξουμε τους χαρακτήρες μας για να επιλέξουμε κάθε φορά ποιον θέλουμε να χειριστούμε. Ο παίκτης που πριν ήταν το αντικείμενο που χειριζόμασταν τώρα αποκτά συμπεριφορά της τεχνητής νοημοσύνης που ορίσαμε μέσω κώδικα. Όταν ο ένας από τους 2 παίκτες της ομάδας σκοτωθεί μέσα στο παιχνίδι δεν υπάρχει πλέον η δυνατότητα εναλλαγής. Στις 2 παρακάτω εικόνες φαίνονται οι 2 παίκτες μας, καθώς κάθε ένας από αυτούς έχει το component "Navmesh Agent" για να λειτουργήσει η ιδέα που περιγράψαμε.

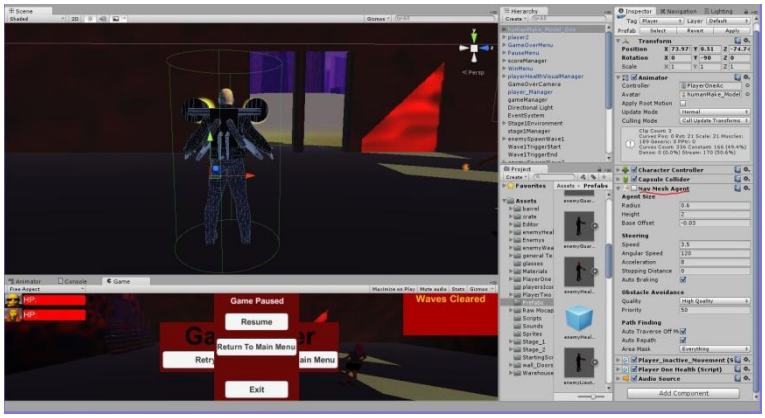

Figure 3.3.6-2:Player 1 Navmesh Agent

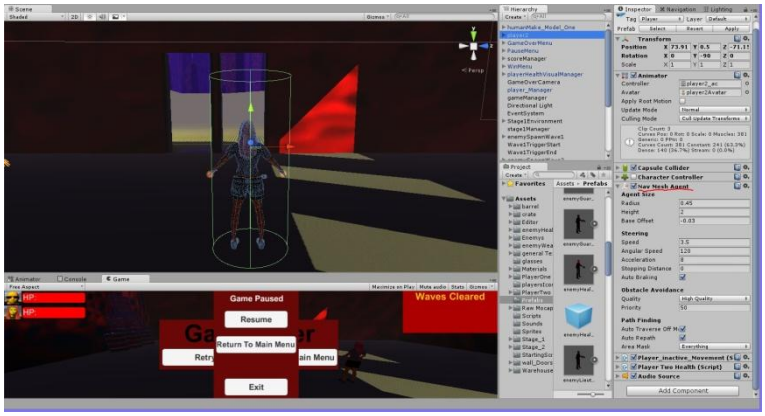

Figure 3.3.6-3:Player 2 Navmesh Agent

Σε αυτό το σημείο θα παραθέσουμε τον κώδικα που δημιουργήθηκε στα πλαίσια της πτυχιακής για την τεχνητή νοημοσύνη της ομάδας και την εναλλαγή ενεργού παίκτη. Στην παρακάτω εικόνα παρατηρούμε τον κώδικα που τρέχει σε κάθε ένα από τα μέλη της ομάδας και αφορά τις συνθήκες και τον διαχωρισμό μεταξύ ενεργού και καθοδηγούμενου παίκτη.

```
public class genPlayerManager : MonoBehaviour {
    public GameObject playerOne;
    public GameObject playerTwo;
    public Camera plOneCam;
    private Vector3 plOneCamPos;
    public Camera plTwoCam;
    private Vector3 plTwoCamPos;
    public CharacterController ctrlPlayerOne;
    public CharacterController ctrlPlayerTwo;
    private GameObject lastPlayerActiveCam;
    private Camera gameOverCam;
    public bool setAlivePlOne = true;
    public bool setAlivePlTwo = true;
    void Start(){
        lastPlayerActiveCam = GameObject.Find ("GameOverCamera"); // find object camera to use for game over
        gameOverCam = lastPlayerActiveCam.GetComponent<Camera> (); //get the component of the gameobject(Camera)
        gameOverCam.transform.position = new Vector3 (0, 0, 0); //set a random pos for starting
    }
    void Update () {
        //player one
        if (Input.GetKeyDown (KeyCode.F1) && setAlivePlOne==true && setAlivePlTwo==true) {
            plOneCam.enabled = true;
            plTwoCam.enabled = false;
            Animator anim = playerOne.GetComponent<Animator> ();
            anim.SetBool ("isWalking", false);
            anim.SetBool ("isRunning", false);
            if (playerOne.GetComponent<NavMeshAgent> ().isActiveAndEnabled == true) {
                playerOne.GetComponent<NavMeshAgent> ().enabled = false;
                ctrlPlayerOne.GetComponent<CharacterController> ().enabled = true;
            }
            ctrlPlayerTwo.GetComponent<CharacterController> ().enabled = false;
            playerTwo.GetComponent<NavMeshAgent> ().enabled = true;
        }
        //player two
        if (Input.GetKeyDown (KeyCode.F2) && setAlivePlTwo==true && setAlivePlOne==true) {
            plTwoCam.enabled = true;
            plOneCam.enabled = false;
            Animator anim = playerTwo.GetComponent<Animator> ();
            anim.SetBool ("isWalking", false);
            anim.SetBool ("isRunning", false);
            if (playerTwo.GetComponent<NavMeshAgent> ().isActiveAndEnabled == true) {
                playerTwo.GetComponent<NavMeshAgent> ().enabled = false;
                ctrlPlayerTwo.GetComponent<CharacterController> ().enabled = true;
            }
            ctrlPlayerOne.GetComponent<CharacterController> ().enabled = false;
            playerOne.GetComponent<NavMeshAgent> ().enabled = true;
        }
        //case player1 dead.
        if (playerOne == null && playerTwo!=null) {
            plTwoCam.enabled = true;
            plTwoCamPos=plTwoCam.transform.position;
            gameOverCam.transform.position=plTwoCamPos;
            playerTwo.GetComponent<NavMeshAgent> ().enabled = false;
            ctrlPlayerTwo.GetComponent<CharacterController> ().enabled = true;
        }
        //case player2 dead.
        if (playerTwo == null && playerOne!=null) {
            plOneCam.enabled = true;
            plOneCamPos=plOneCam.transform.position;
            gameOverCam.transform.position=plOneCamPos;
            playerOne.GetComponent<NavMeshAgent> ().enabled = false;
            ctrlPlayerOne.GetComponent<CharacterController> ().enabled = true;
        }
        //Game Over State.Cam enable.
        if (playerOne == null && playerTwo == null) {
            gameOverCam.enabled=true;
        }
    }
}
```

Figure 3.3.6-4: Player Switch Code

Στην παραπάνω φωτογραφία φαίνεται ο κώδικας για την εναλλαγή των παικτών και στην ακόλουθη φωτογραφία φαίνεται ο κώδικας της λειτουργικότητας των διαταγών που δίνει ο ενεργός παίκτης στην ομάδα για να ακολουθήσει.

```
    //Call Back Teammate -player1 or player2(CALL BACK REST OF THE TEAM WHILE HAVING ANY OF THEM AS ACTIVE PLAYER).
    if(Input.GetKey(KeyCode.R)  && callBack==false){
        status="regroup";
        callBack=true;
    }
    if (myAgent.enabled==true && callBack == true && playerOne!=null && playerTwo!=null) {//CALLBACK FUNCTION
        myAgent = this.GetComponent<NavMeshAgent> ();
        //call back teammate player2
        if(myAgent.name==playerTwo.name){
            myAgent.SetDestination(playerOne.transform.position);
            distancePlayer=Vector3.Distance(this.transform.position,playerOne.transform.position);
            if (distancePlayer < 3) {
                myAgent.SetDestination (this.transform.position);
                anim.SetBool("isRunning",false);
                anim.SetBool("isWalking",false);
            }
            if (distancePlayer > 3) {
                myAgent.SetDestination (playerOne.transform.position);
                anim.SetBool("isRunning",false);
                myAgent.speed=2.0f;
                anim.SetBool("isWalking",true);
            }
        }
        //call back teammate player1
        if(myAgent.name==playerOne.name){
            myAgent.SetDestination(playerTwo.transform.position);
            distancePlayer=Vector3.Distance(this.transform.position,playerTwo.transform.position);
            if (distancePlayer < 3) {
                myAgent.SetDestination (this.transform.position);
                anim.SetBool("isRunning",false);
                anim.SetBool("isWalking",false);
            }
            if (distancePlayer > 3) {
                myAgent.SetDestination (playerTwo.transform.position);
                anim.SetBool("isRunning",false);
                myAgent.speed=2.0f;
                anim.SetBool("isWalking",true);
            }
        }
    }//end of player callback function
```

Figure 3.3.6-5:Regroup Order Code

```
//assisting active player
if (myAgent.enabled == true && callBack==false && playerOne!=null && playerTwo!=null) {
    myAgent = this.GetComponent<NavMeshAgent> ();

    if(myAgent.name==playerOne.name){//player one A.I case
        enemyPos=playerTwo.GetComponentInChildren<playerTwoShooting>().getEnemyPos();
        if(enemyPos.gameObject==null){//case where enemy is killed while assisting
            if(playerTwo.GetComponent<NavMeshAgent>().isActiveAndEnabled==false){
                status="regroup";
                callBack=true;
            }
        }
        if(enemyPos!=null){
            distanceEnemy =Vector3.Distance (this.transform.position,enemyPos.transform.position);//distance from player A.I to enemy
            if(enemyPos.gameObject.tag=="enemy" && enemyPos!=null){
                myAgent.SetDestination(enemyPos.transform.position);
                anim.SetBool("isWalking",true);
                myAgent.speed=3.5f;
                anim.SetBool("isRunning",true);
            }
            if (distanceEnemy<=10 && enemyPos!=null) {
                anim.SetBool("isRunning",false);
                myAgent.speed=2.0f;
                anim.SetBool("isWalking",false);
                myAgent.SetDestination (this.transform.position);
                if(enemyPos.name==playerTwo.name)
                    myAgent.transform.LookAt(new Vector3(this.transform.position.x,enemyPos.transform.position.y,this.transform.position.z));
                if(enemyPos.name!=playerTwo.name)
                    myAgent.transform.LookAt(new Vector3(enemyPos.transform.position.x,this.transform.position.y,enemyPos.transform.position.z));
            }
            if (distanceEnemy>20 && enemyPos!=null) {
                myAgent.SetDestination (enemyPos.transform.position);
                anim.SetBool("isWalking",true);
                myAgent.speed=3.5f;
                anim.SetBool("isRunning",true);
            }
        }
    }
    if(myAgent.name==playerTwo.name){//player two A.I case
        enemyPos=playerOne.GetComponentInChildren<playerOneShooting>().getEnemyPos();
        if(enemyPos.gameObject==null){//case where enemy is killed while assisting
            if(playerOne.GetComponent<NavMeshAgent>().isActiveAndEnabled==false){
                status="regroup";
                callBack=true;
            }
        }
        if(enemyPos!=null){
            distanceEnemy =Vector3.Distance (this.transform.position,enemyPos.transform.position);//distance from player A.I to enemy
            if(enemyPos.gameObject.tag=="enemy" && enemyPos!=null){
                myAgent.SetDestination(enemyPos.transform.position);
                anim.SetBool("isWalking",true);
                myAgent.speed=3.5f;
                anim.SetBool("isRunning",true);
            }
            if (distanceEnemy<=7 && enemyPos!=null) {
                anim.SetBool("isRunning",false);
                myAgent.speed=2.0f;
                anim.SetBool("isWalking",false);
                myAgent.SetDestination (this.transform.position);
                if(enemyPos.name==playerOne.name)
                    myAgent.transform.LookAt(new Vector3(this.transform.position.x,enemyPos.transform.position.y,this.transform.position.z));
                if(enemyPos.name!=playerOne.name)
                    myAgent.transform.LookAt(new Vector3(enemyPos.transform.position.x,this.transform.position.y,enemyPos.transform.position.z));
            }
            if (distanceEnemy>14 && enemyPos!=null) {
                myAgent.SetDestination (enemyPos.transform.position);
                anim.SetBool("isWalking",true);
                myAgent.speed=3.5f;
                anim.SetBool("isRunning",true);
            }
        }
    }
}//end of assist function
```

Figure 3.3.6-6:Assist Order Code

Η τεχνητή νοημοσύνη των αντιπάλων έχει διαφορετική λειτουργία για την κάθε πίστα . Αρχικά θα εξετάσουμε την τεχνητή νοημοσύνη ως προς την συμπεριφορά της στην 1η πίστα. Η ομάδα των αντιπάλων αποτελείται από ένα αρχηγό, ένα υπαρχηγό, δύο φρουρούς και ένα θεραπευτή. Κάθε μονάδα βασίζεται στην λογική ιεραρχίας και συνεπώς έχει μεγαλύτερη ανοχή στα χτυπήματα. Όταν χτυπηθεί κάποια μονάδα στρέφονται όλες οι μονάδες κατευθείαν στον στόχο και τον κυνηγάνε έως ότου αυτός εξολοθρευτεί. Κάθε μονάδα που δέχεται ζημία και η ζημιά αυτή φτάσει στο 50% ή περισσότερο κάνει αίτημα στον θεραπευτή για επούλωση ζημιάς.

Ο θεραπευτής ακολουθάει (εφόσον δεν τον έχουν καλέσει για να θεραπεύσει ζημιά) την μεγαλύτερη ιεραρχικά μονάδα. Στην περίπτωση που ο θεραπευτής χτυπηθεί πρώτος ενημερώνει την ομάδα του για τον παίκτη που του έκανε ζημία μέσω του αρχηγού. Η παραπάνω διαδικασία περιγράφεται από τον παρακάτω κώδικα που αφορά το κάθε μέλος της ομάδας ξεχωριστά.

```
//healing request.
if (count <=500 && healer!=null && general.gameObject!=null ) {
    healer.GetComponentInChildren<enemyHealerHealing>().setAidTarget(general.gameObject,true);
}
if (healer != null) {
    if (healer.GetComponentInChildren<enemyHealerHealing> ().getTargetName () == general.gameObject) {//checking if currently being healed and hea
        //healing break.
        if (((count > 500) && healer != null)) {
            healer.GetComponentInChildren<enemyHealerHealing> ().setAidTarget (healer, false);
        }
    }
}

//player active routine check
if (enemyName!=null && enemyName!="" && general.isActiveAndEnabled==true) {
    player=GameObject.Find(enemyName);//checking for active player if shoot the general
    if(player!=null && general.isActiveAndEnabled==true){
        playerDistance = Vector3.Distance (this.transform.position, player.transform.position);//distance from general to active player.
        general.gameObject.transform.LookAt(player.transform.position);
        general.SetDestination(player.transform.position);
        general.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
        if(playerDistance<5){
            general.gameObject.transform.LookAt(player.transform.position);
            general.SetDestination(this.transform.position);
            general.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
        }
    }
    if(player==null && general.isActiveAndEnabled==true){
        general.SetDestination(this.transform.position);
        general.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
    }
}
//if player recognized is not Dead send units to pursuit player.
if (player != null && (playerDistance>8 || playerDistance<=25)) {
    if(GameObject.Find ("enemyLieutenant(Clone)")!= null && lieutenant.GetComponent<NavMeshAgent> ().isActiveAndEnabled ==true){
        lieutenantDistance=Vector3.Distance(lieutenant.transform.position,player.transform.position);
        lieutenant.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
        lieutenant.GetComponent<NavMeshAgent>().SetDestination (player.transform.position);
        if(lieutenantDistance<5){
            lieutenant.transform.LookAt(player.transform.position);
            lieutenant.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
            lieutenant.GetComponent<NavMeshAgent>().SetDestination(lieutenant.transform.position);
        }
    }
    if(GameObject.Find ("enemyGuard_1(Clone)")!= null && guardOne.GetComponent<NavMeshAgent> ().isActiveAndEnabled ==true){
        guardOneDistance=Vector3.Distance(guardOne.transform.position,player.transform.position);
        guardOne.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
        guardOne.GetComponent<NavMeshAgent>().SetDestination(player.transform.position);
        if(guardOneDistance<5){
            guardOne.transform.LookAt(player.transform.position);
            guardOne.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
            guardOne.GetComponent<NavMeshAgent>().SetDestination(guardOne.transform.position);
        }
    }
    if(GameObject.Find ("enemyGuard_2(Clone)")!= null && guardTwo.GetComponent<NavMeshAgent> ().isActiveAndEnabled ==true){
        guardTwoDistance=Vector3.Distance(guardTwo.transform.position,player.transform.position);
        guardTwo.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
        guardTwo.GetComponent<NavMeshAgent>().SetDestination(player.transform.position);
        if(guardTwoDistance<5){
            guardTwo.transform.LookAt(player.transform.position);
            guardTwo.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
            guardTwo.GetComponent<NavMeshAgent>().SetDestination(guardTwo.transform.position);
        }
    }
}
```

Figure 3.3.6-7: Enemy General Behavior Code Part 1

```
        //if player recognized is Dead or is far away from general return lower units to follow general
        if (player == null  || playerDistance>30 || lieutenantGeneralDistance>30) {
            enemyName="";
            if(GameObject.Find ("enemyLieutenant(Clone)")!= null && lieutenant.GetComponent<NavMeshAgent> ().isActiveAndEnabled ==true){
                lieutenant.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                lieutenant.GetComponent<NavMeshAgent>().SetDestination (general.transform.position);
                float bossDistance=Vector3.Distance(lieutenant.transform.position,general.transform.position);
                if(bossDistance<5){
                    lieutenant.transform.LookAt(general.transform.position);
                    lieutenant.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    lieutenant.GetComponent<NavMeshAgent>().SetDestination(lieutenant.transform.position);
                }
            }
            if(GameObject.Find ("enemyGuard_1(Clone)")!= null && guardOne.GetComponent<NavMeshAgent> ().isActiveAndEnabled ==true){
                guardOne.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guardOne.GetComponent<NavMeshAgent>().SetDestination(general.transform.position);
                float bossDistance=Vector3.Distance(guardOne.transform.position,general.transform.position);
                if(bossDistance<5){
                    guardOne.transform.LookAt(general.transform.position);
                    guardOne.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    guardOne.GetComponent<NavMeshAgent>().SetDestination(guardOne.transform.position);
                }

            }
            if(GameObject.Find ("enemyGuard_2(Clone)")!= null && guardTwo.GetComponent<NavMeshAgent> ().isActiveAndEnabled ==true){
                guardTwo.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guardTwo.GetComponent<NavMeshAgent>().SetDestination(general.transform.position);
                float bossDistance=Vector3.Distance(guardTwo.transform.position,general.transform.position);
                if(bossDistance<5){
                    guardTwo.transform.LookAt(general.transform.position);
                    guardTwo.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    guardTwo.GetComponent<NavMeshAgent>().SetDestination(guardTwo.transform.position);
                }
            }
        }
    }
    void OnCollisionEnter(Collision col){
        if (col.gameObject.tag == "playerBullet") {
            count-=1; //damage 1 per hit.
            GameObject dmg=this.gameObject; //DAMAGE FLAG
            dmg.GetComponentInChildren<enemyGeneralHealth>().setDamageTaken(); //WARN U.I CHANGE
            enemyName=col.gameObject.GetComponent<playerBulletScript>().getSourceHitName(); //gets the enemy name that hit this a.i unit.
        }
        if (col.gameObject.tag == "healingBeam") {
            count+=5; //heal 1 per cast
            GameObject healing=this.gameObject; //HEALING FLAG
            healing.GetComponentInChildren<enemyGeneralHealth>().setHealingTaken();//WARN U.I CHANGE.

        }
    }
```

Figure 3.3.6-8:Enemy General Behavior Code Part 2

Ο παραπάνω κώδικας αφοράει τον αρχηγό και αναφέρεται σε αίτημα για επούλωση προς τον θεραπευτή της ομάδας, καθώς και για την εύρεση και ενημέρωση της ομάδας για τυχόν ζημιά.

```
        //healing request.
        if (count <=350 && healer!=null && lieutenant.gameObject!=null ) {
            healer.GetComponentInChildren<enemyHealerHealing>().setAidTarget(lieutenant.gameObject,true);
        }
        if (healer != null) {
            if (healer.GetComponentInChildren<enemyHealerHealing> ().getTargetName () == lieutenant.gameObject) {
                //healing cancel.
                if (((count > 350) && healer != null)) {
                    healer.GetComponentInChildren<enemyHealerHealing> ().setAidTarget (healer, false);
                }
            }
        }
        //direct hit notify general and pursuit
        if (enemyName!=null && enemyName!="") {
            tempPl = GameObject.Find (enemyName);
            if(general!=null && tempPl!=null){
                general.GetComponent<enemyGeneral>().setEnemyName(tempPl.name);
            }
        }
        //general orders to pursuit enemy.
        if(general!=null && (general.GetComponent<enemyGeneral>().getEnemyName()!=null && general.GetComponent<enemyGeneral>().getEnemyName()!="")){
            enemyName=general.GetComponent<enemyGeneral>().getEnemyName();
            tempPl=GameObject.Find(enemyName);
        }
        //lieutenant orders
        if (general == null && this.isActiveAndEnabled==true) {
            GameObject player=GameObject.Find(enemyName);
            lieutenant.gameObject.GetComponent<Animator>().SetBool("isWalking",false); //init state immobilized . (temporal)
            lieutenant.SetDestination(this.transform.position);
            if(guardOne!=null && guardOne.GetComponent<NavMeshAgent>().isActiveAndEnabled==true && player!=null){
                playerDistanceOne=Vector3.Distance(guardOne.transform.position,tempPl.transform.position);
                guardOne.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guardOne.GetComponent<NavMeshAgent>().SetDestination(player.transform.position);
                if(playerDistanceOne<5){
                    guardOne.transform.LookAt(tempPl.transform.position);
                    guardOne.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    guardOne.GetComponent<NavMeshAgent>().SetDestination(guardOne.transform.position);
                }
            }
            if(guardTwo!=null && guardTwo.GetComponent<NavMeshAgent>().isActiveAndEnabled==true && player!=null){
                playerDistanceTwo=Vector3.Distance(guardTwo.transform.position,tempPl.transform.position);
                guardTwo.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guardTwo.GetComponent<NavMeshAgent>().SetDestination(player.transform.position);
                if(playerDistanceTwo<5){
                    guardTwo.transform.LookAt(tempPl.transform.position);
                    guardTwo.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    guardTwo.GetComponent<NavMeshAgent>().SetDestination(guardTwo.transform.position);
                }
            }
            if((guardOne==null  && guardTwo==null) && player!=null && this.isActiveAndEnabled==true){
                playerDistance=Vector3.Distance(this.transform.position,player.transform.position);
                lieutenant.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                lieutenant.SetDestination(player.transform.position);
                if(playerDistance<5){
                    lieutenant.transform.LookAt(player.transform.position);
                    lieutenant.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    lieutenant.GetComponent<NavMeshAgent>().SetDestination(lieutenant.transform.position);
                }
            }
            if(player==null){
                enemyName = "";
            }
        }
    }
    void OnCollisionEnter(Collision col){
        if (col.gameObject.tag == "playerBullet") {
            count-=1; //apply dmg each time being hit.(debug method).
            GameObject dmg=this.gameObject; //DAMAGE FLAG
            dmg.GetComponentInChildren<enemyLieutenantHealth>().setDamageTaken(); //WARN U.I CHANGE
            enemyName=col.gameObject.GetComponent<playerBulletScript>().getSourceHitName(); //gets the enemy name that hit this a.i unit.
        }
        if (col.gameObject.tag == "healingBeam") {
            count+=3; //heal 1 per cast
            GameObject healing=this.gameObject; //HEALING FLAG
            healing.GetComponentInChildren<enemyLieutenantHealth>().setHealingTaken();//WARN U.I CHANGE.
        }
```

Figure 3.3.6-9:Enemy Lieutenant Behavior Code

Αντίστοιχα παραπάνω περιγράφεται και ο κώδικας συμπεριφοράς του υπαρχηγού της ομάδας των αντιπάλων μας.

```
        //healing request
        if (count <= 250 && healer != null && guard.gameObject != null) {
            healer.GetComponentInChildren<enemyHealerHealing> ().setAidTarget (guard.gameObject, true);

        }
        if (healer != null) {
            if (healer.GetComponentInChildren<enemyHealerHealing> ().getTargetName () == guard.gameObject) {
                //healing break
                if (((count > 250) && healer != null)) {
                    healer.GetComponentInChildren<enemyHealerHealing> ().setAidTarget (healer, false);
                }
            }
        }

        //self order
        if((lieutenant==null && general==null && healer==null)){
            enemyName=this.GetComponent<enemyGuard>().getEnemyName();
            if(enemyName!=null && enemyName!=""){
                GameObject tempPl=GameObject.Find(enemyName);
                if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true && tempPl!=null){
                    playerDistance=Vector3.Distance(guard.transform.position,tempPl.transform.position);
                    guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                    guard.SetDestination (tempPl.transform.position);//chase player
                    if(playerDistance<5){
                        guard.transform.LookAt(tempPl.transform.position);
                        guard.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                        guard.SetDestination(this.transform.position);
                    }
                }
            }
            if(enemyName==null){
                if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true){
                    guard.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    guard.SetDestination (this.transform.position); //root position
                }
            }
        }
        //orders from lieutenant
        if (lieutenant !=null && enemyName=="") {
            enemyName=lieutenant.GetComponent<enemyLieutenant>().getEnemyName();//enemyLieutenant
            if(enemyName!=null && enemyName!=""){
                GameObject tempPl=GameObject.Find(enemyName);
                if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true){
                    playerDistance=Vector3.Distance(guard.transform.position,tempPl.transform.position);
                    guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                    guard.SetDestination (tempPl.transform.position);//chase player
                    if(playerDistance<5){
                        guard.transform.LookAt(tempPl.transform.position);
                        guard.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                        guard.SetDestination(this.transform.position);
                    }
                }
            }
            if(enemyName==null && enemyName==""){
                enemyName="";
                if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true){
                    guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                    guard.SetDestination(lieutenant.transform.position);//follow lieutenant
                }
            }
        }
```

Figure 3.3.6-10:Enemy Guard Behavior Code Part 1

```
    //orders from general
    if (lieutenant == null && general != null && enemyName=="") {
        enemyName=general.GetComponent<enemyGeneral>().getEnemyName();//enemyGeneral
        if(enemyName!=null && enemyName!=""){
            GameObject tempPl=GameObject.Find(enemyName);
            if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true){
                playerDistance=Vector3.Distance(guard.transform.position,tempPl.transform.position);
                guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guard.SetDestination (tempPl.transform.position);//chase player
                if(playerDistance<5){
                    guard.transform.LookAt(tempPl.transform.position);
                    guard.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    guard.SetDestination(this.transform.position);
                }
            }
        }
        if(enemyName==null  && enemyName==""){
            if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true){
                guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guard.SetDestination (general.transform.position);//follow general
            }
        }
    }
    //orders from healer.
    if (lieutenant == null && general == null && healer != null) {
        enemyName=healer.GetComponent<enemyHealer>().getEnemyName();
        if(enemyName!=null && enemyName!=""){
            GameObject tempPl=GameObject.Find(enemyName);
            if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true && tempPl!=null){
                playerDistance=Vector3.Distance(guard.transform.position,tempPl.transform.position);
                guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guard.SetDestination (tempPl.transform.position);//chase player
                if(playerDistance<5){
                    guard.transform.LookAt(tempPl.transform.position);
                    guard.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
                    guard.SetDestination(this.transform.position);
                }
            }
        }
        if(enemyName==null && enemyName==""){
            if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true){
                guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
                guard.SetDestination (healer.transform.position); //follow healer
            }
        }
    }
}
void OnCollisionEnter(Collision col){
    if (col.gameObject.tag == "playerBullet") {
        count-=1; //apply dmg each time being hit.(debug method).
        GameObject dmg=this.gameObject; //DAMAGE FLAG
        if(dmg==GameObject.Find("enemyGuard_1(Clone)")){
            dmg.GetComponentInChildren<enemyGuardOneHealth>().setDamageTaken(); //WARN U.I CHANGE
        }
        if(dmg==GameObject.Find("enemyGuard_2(Clone)")){
            dmg.GetComponentInChildren<enemyGuardTwoHealth>().setDamageTaken(); //WARN U.I CHANGE
        }
        enemyName=col.gameObject.GetComponent<playerBulletScript>().getSourceHitName(); //gets the enemy nam
    }
    if (col.gameObject.tag == "healingBeam") {
        count+=2; //heal 1 per cast
        GameObject healing=this.gameObject; //HEALING FLAG
        if(healing==GameObject.Find("enemyGuard_1(Clone)")){
            healing.GetComponentInChildren<enemyGuardOneHealth>().setHealingTaken(); //WARN U.I CHANGE
        }
        if(healing==GameObject.Find("enemyGuard_2(Clone)")){
            healing.GetComponentInChildren<enemyGuardTwoHealth>().setHealingTaken(); //WARN U.I CHANGE
        }

    }
}
```

Figure 3.3.6-11: Enemy Guard Behavior Code Part 2

Αντίστοιχα και εδώ βλέπουμε στις 2 παραπάνω εικόνες την συμπεριφορά των φρουρών μας και την αλληλεπίδραση του με τον θεραπευτή.

```
//healer healing method.
if (healer.isActiveAndEnabled == true) {
    if(healer.GetComponentInChildren<enemyHealerHealing>().getTargetName()!=null && healer.GetComponentInChildren<enemyHealerHealing>().getTargetName()!=healer){
        teammate=healer.GetComponentInChildren<enemyHealerHealing>().getTargetName().name; //constantly checking
        if(GameObject.Find(teammate)!=null){
            if(GameObject.Find(teammate)!=null && GameObject.Find(teammate)==general ){//general requesting heal.
                int health=general.GetComponent<enemyGeneral>().count;
                if(health<500){
                    if(Vector3.Distance(healer.gameObject.transform.position,general.transform.position)>7){
                        healer.GetComponent<Animator>().SetBool("isRunning",true);
                    }
                    if(Vector3.Distance(healer.gameObject.transform.position,general.transform.position)>3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",true);
                    }
                    healer.transform.LookAt (general.transform.position); //lock on target.
                    healer.GetComponent<NavMeshAgent>().SetDestination(general.transform.position);
                    if(Vector3.Distance(healer.gameObject.transform.position,general.transform.position)<3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",false);
                        healer.transform.LookAt (general.transform.position); //lock on target.
                        healer.GetComponent<NavMeshAgent>().SetDestination(healer.gameObject.transform.position);
                    }
                }
                if(health<=0 || general==null){
                    healer.GetComponent<Animator>().SetBool("isRunning",false);
                    healer.GetComponent<Animator>().SetBool("isWalking",false);
                    healer.GetComponentInChildren<enemyHealerHealing>().setHealingState(false);
                }
            }

            if(GameObject.Find(teammate)!=null && GameObject.Find(teammate)==lieutenant ){//lieutenant requesting heal.
                int health=lieutenant.GetComponent<enemyLieutenant>().count;
                if(health<350){
                    if(Vector3.Distance(healer.gameObject.transform.position,lieutenant.transform.position)>7){
                        healer.GetComponent<Animator>().SetBool("isRunning",true);
                    }
                    if(Vector3.Distance(healer.gameObject.transform.position,lieutenant.transform.position)>3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",true);
                    }
                    healer.transform.LookAt (lieutenant.transform.position); //lock on target.
                    healer.GetComponent<NavMeshAgent>().SetDestination(lieutenant.transform.position);
                    if(Vector3.Distance(healer.gameObject.transform.position,lieutenant.transform.position)<3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",false);
                        healer.transform.LookAt (lieutenant.transform.position); //lock on target.
                        healer.GetComponent<NavMeshAgent>().SetDestination(healer.gameObject.transform.position);
                    }
                }
                if(health<=0 || lieutenant==null){
                    healer.GetComponent<Animator>().SetBool("isRunning",false);
                    healer.GetComponent<Animator>().SetBool("isWalking",false);
                    healer.GetComponentInChildren<enemyHealerHealing>().setHealingState(false);
                }
            }
```

Figure 3.3.6-12:Enemy Healer Healing Behavior Part 1

```
            if(GameObject.Find(teammate)!=null && GameObject.Find(teammate)==guardOne){//guardOne requesting heal.
                int health=guardOne.GetComponent<enemyGuard>().count;
                if(health<250){
                    if(Vector3.Distance(healer.gameObject.transform.position,guardOne.transform.position)>7){
                        healer.GetComponent<Animator>().SetBool("isRunning",true);
                    }
                    if(Vector3.Distance(healer.gameObject.transform.position,guardOne.transform.position)>3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",true);
                    }
                    healer.transform.LookAt (guardOne.transform.position); //lock on target.
                    healer.GetComponent<NavMeshAgent>().SetDestination(guardOne.transform.position);
                    if(Vector3.Distance(healer.gameObject.transform.position,guardOne.transform.position)<3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",false);
                        healer.transform.LookAt (guardOne.transform.position); //lock on target.
                        healer.GetComponent<NavMeshAgent>().SetDestination(healer.gameObject.transform.position);
                    }
                }
                if(health<=0 || guardOne==null){
                    healer.GetComponent<Animator>().SetBool("isRunning",false);
                    healer.GetComponent<Animator>().SetBool("isWalking",false);
                    healer.GetComponentInChildren<enemyHealerHealing>().setHealingState(false);
                }
            }

            if(GameObject.Find(teammate)!=null && GameObject.Find(teammate)==guardTwo){//guardTwo requesting heal.
                int health=guardTwo.GetComponent<enemyGuard>().count;
                if(health<250){
                    if(Vector3.Distance(healer.gameObject.transform.position,guardTwo.transform.position)>7){
                        healer.GetComponent<Animator>().SetBool("isRunning",true);
                    }
                    if(Vector3.Distance(healer.gameObject.transform.position,guardTwo.transform.position)>3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",true);
                    }
                    healer.transform.LookAt (guardTwo.transform.position); //lock on target.
                    healer.GetComponent<NavMeshAgent>().SetDestination(guardTwo.transform.position);
                    if(Vector3.Distance(healer.gameObject.transform.position,guardTwo.transform.position)<3){
                        healer.GetComponent<Animator>().SetBool("isRunning",false);
                        healer.GetComponent<Animator>().SetBool("isWalking",false);
                        healer.transform.LookAt (guardTwo.transform.position); //lock on target.
                        healer.GetComponent<NavMeshAgent>().SetDestination(healer.gameObject.transform.position);
                    }
                }
                if(health<=0 || guardTwo==null){
                    healer.GetComponent<Animator>().SetBool("isRunning",false);
                    healer.GetComponent<Animator>().SetBool("isWalking",false);
                    healer.GetComponentInChildren<enemyHealerHealing>().setHealingState(false);
                }
            }
        }
        else{
            teammate="";
        }
    }
}
```

Figure 3.3.6-13:Enemy Healer Healing Behavior Part 2

Τα 2 παραπάνω κομμάτια αποτελούν τον τρόπο με τον οποίο ο θεραπευτής δρα με βάση τις ανάγκες της ομάδας για επούλωση ζημιάς.

```
//healer alerting party.
if(general!=null){
    if(enemyName!=null && enemyName!="")
        general.GetComponent<enemyGeneral> ().setEnemyName (enemyName);//update general
}
if(general==null && lieutenant!=null){
    if(enemyName!=null && enemyName!="")
        lieutenant.GetComponent<enemyLieutenant>().setEnemyName(enemyName); //update lieutenant
}
if(general==null && lieutenant==null && (guardOne!=null || guardTwo!=null)){
    if(guardOne!=null){
        if(enemyName!=null && enemyName!="")
            guardOne.GetComponent<enemyGuard>().setEnemyName(enemyName);//update guard1
    }
    if(guardTwo!=null){
        if(enemyName!=null && enemyName!="")
            guardTwo.GetComponent<enemyGuard>().setEnemyName(enemyName);//update guard2
    }
}
//healer health
if (count < 0) {
    healer.SetDestination(this.transform.position);
    Destroy(this.gameObject);
}
```

Figure 3.3.6-14:Enemy Healer Alerting Team Behavior

Εδώ βλέπουμε το σημείο του κώδικα που ο θεραπευτής ειδοποιεί την ομάδα του όταν χτυπηθεί.

```
//healer following methods.
if (general != null && this.gameObject != null && healer.isActiveAndEnabled==true && (teammate=="" || teammate==healer.gameObject.name)) {//&& healer.GetComponentInChildren<enemyHealerHealing>().getHealingState()==false
    generalDistance=Vector3.Distance(healer.transform.position,general.transform.position);
    healer.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
    healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
    healer.SetDestination (general.transform.position); // follow general
    if(generalDistance<5){
        healer.transform.LookAt(general.transform.position);
        healer.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
        healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
        healer.SetDestination(healer.transform.position);
    }
}
if (general == null && lieutenant != null && healer.isActiveAndEnabled == true && (teammate=="" || teammate==healer.gameObject.name)) { //&& healer.GetComponentInChildren<enemyHealerHealing>().getHealingState()==false
    lieutenantDistance=Vector3.Distance(healer.transform.position,lieutenant.transform.position);
    healer.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
    healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
    healer.SetDestination (lieutenant.transform.position); //follow lieutenant
    if(lieutenantDistance<5){
        healer.transform.LookAt(lieutenant.transform.position);
        healer.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
        healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
        healer.SetDestination(healer.transform.position);
    }
}
if(general==null && lieutenant==null && healer.isActiveAndEnabled==true && (teammate=="" || teammate==healer.gameObject.name)){//&& healer.GetComponentInChildren<enemyHealerHealing>().getHealingState()==false
    if(guardOne!=null){
        guardOneDistance=Vector3.Distance(healer.transform.position,guardOne.transform.position);
        healer.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
        healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
        healer.SetDestination(guardOne.transform.position);//follow guard 1
        if(guardOneDistance<5){
            healer.transform.LookAt(guardOne.transform.position);
            healer.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
            healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
            healer.SetDestination(healer.transform.position);
        }
    }
    if(guardTwo!=null){
        guardTwoDistance=Vector3.Distance(healer.transform.position,guardTwo.transform.position);
        healer.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
        healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
        healer.SetDestination(guardTwo.transform.position);//follow guard 2
        if(guardTwoDistance<5){
            healer.transform.LookAt(guardTwo.transform.position);
            healer.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
            healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
            healer.SetDestination(healer.transform.position);
        }
    }
    if (guardOne == null && guardTwo == null && healer.isActiveAndEnabled==true) {
        healer.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
        healer.gameObject.GetComponent<Animator>().SetBool("isRunning",false);
        healer.SetDestination (this.transform.position); //self navigation
    }
}
}
void OnCollisionEnter(Collision col){
    if(col.gameObject.tag=="playerBullet"){
        count-=1; //apply dmg each time being hit.
        GameObject dmg=this.gameObject; //DAMAGE FLAG
        dmg.GetComponentInChildren<enemyHealerHealth>().setDamageTaken(); //WARN U.I CHANGE
        this.setEnemyName(col.gameObject.GetComponent<playerBulletScript>().getSourceHitName()); //gets the enemy name that hit this a.i unit.
    }
}
```

Figure 3.3.6-15:Enemy Healer Following Behavior

Τελευταίο κομμάτι για την λειτουργία του θεραπευτή είναι η ιεραρχία που ακολουθεί την ομάδα του με βάση τον παραπάνω κώδικα.

Στις προηγούμενες σελίδες αναφερθήκαμε στην συμπεριφορά των αντιπάλων για την 1η πίστα. Στην 2η πίστα έχουμε μια πιο απλοποιημένη μορφή τεχνητής νοημοσύνης όπου μέσα στο χώρο της πίστας έχουμε 4 σημεία στα οποία δημιουργούνται φρουροί στο κάθε ένα από αυτά. Όταν χτυπηθεί κάποιος από τους φρουρούς αυτόματα ενημερώνονται όλοι οι φρουροί και καλούνται να επιτεθούν στον παίκτη που χτύπησε τελευταία φορά. Από την στιγμή εκείνη και μετά ο παίκτης που επικεντρώνει τα χτυπήματά του σε οποιαδήποτε μονάδα των εχθρών συνεχώς καταδιώκεται μέχρι να ολοκληρώσει τον σκοπό του. Παρακάτω θα δούμε τον κώδικα που γράφτηκε με κάποιες τροποποιήσεις που μοιάζουν αρκετά στην συμπεριφορά του φρουρού της 1ης πίστας.

```
4
5 //guardBehaviour in StandAlone Mode.
6 public class enemyGuardStandAlone : MonoBehaviour {
7     private NavMeshAgent guard; //get Orders or act independently
8     private float guardDistance;
9     private float playerDistance;
0     public string enemyName;
1     public float count;
2     void Start () {
3         guard = this.GetComponent<NavMeshAgent> ();
4         enemyName = "";
5         count = 500;
6     }
7     void Update () {
8         //health check.
9         if (count <= 0) {
0             GameObject scoreUpdate=GameObject.Find("gameManager");
1             scoreUpdate.GetComponent<gameManager>().guardsKilled+=1;
2             guard.SetDestination (this.transform.position);
3             Destroy (this.gameObject);
4         }
5         //guard standalone behaviour movement.
6         enemyName=GameObject.FindGameObjectWithTag("enemy").GetComponent<enemyGuardStandAlone>().getEnemyName();
7         if(enemyName!=null && enemyName!=""){
8             GameObject tempPl=GameObject.Find(enemyName);
9             if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true && tempPl!=null){
0                 playerDistance=Vector3.Distance(guard.transform.position,tempPl.transform.position);
1                 guard.gameObject.GetComponent<Animator>().SetBool("isWalking",true);
2                 guard.SetDestination (tempPl.transform.position);//chase player
3                 if(playerDistance<5){
4                     guard.transform.LookAt(tempPl.transform.position);
5                     guard.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
6                     guard.SetDestination(this.transform.position);
7                 }
8             }
9         }
0         if(enemyName==null){
1             if(this.GetComponent<NavMeshAgent>().isActiveAndEnabled==true){
2                 guard.gameObject.GetComponent<Animator>().SetBool("isWalking",false);
3                 guard.SetDestination (this.transform.position); //root position
4                 }
5             }
6     }
7     void OnCollisionEnter(Collision col){
8         if (col.gameObject.tag == "playerBullet") {
9             count-=5; //apply dmg each time being hit.(debug method).
0             enemyName=col.gameObject.GetComponent<playerBulletScript>().getSourceHitName(); //gets the enemy name that hit this a.i unit.
1             GameObject[] enemiesNotified;
2             enemiesNotified=GameObject.FindGameObjectsWithTag("enemy");
3             foreach(GameObject i in enemiesNotified){
4                 i.GetComponent<enemyGuardStandAlone>().setEnemyName(enemyName);
5             }
6         }
7     }
8     public void setEnemyName(string enemy){
9         this.enemyName = enemy;
0     }
1     public string getEnemyName(){
2         return enemyName;
3     }
4 }
```

Figure 3.3.6-16:Enemy Guard Behavior Code (Stage 2)

3.3.7 Particle Systems

Τα Particle Systems είναι ένας μηχανισμός της Unity για να δημιουργήσουμε αλληλουχίες από μικρά (συνήθως) σωματίδια που αναλόγως τις ρυθμίσεις που έχουμε θέσει σαν προγραμματιστές μπορούν να μας δώσουν πολλά διαφορετικά αποτελέσματα. Πολύ γνωστό παράδειγμα που λειτουργεί σε αυτή την μορφή είναι ο καπνός που παράγει η εξάτμιση του αυτοκινήτου ή κάποιας καμινάδας σε αρκετά videogames . Υπάρχουν γενικότερα πολλά ειδικά εφέ που μπορούν να επιτευχθούν με αυτό τον τρόπο.

Τα particle systems αποτέλεσαν βοηθητικό υλικό στις πίστες μας, αλλά και στην αποτύπωση κατεύθυνσης της σφαίρας όπως θα δούμε στην παρακάτω φωτογραφία.

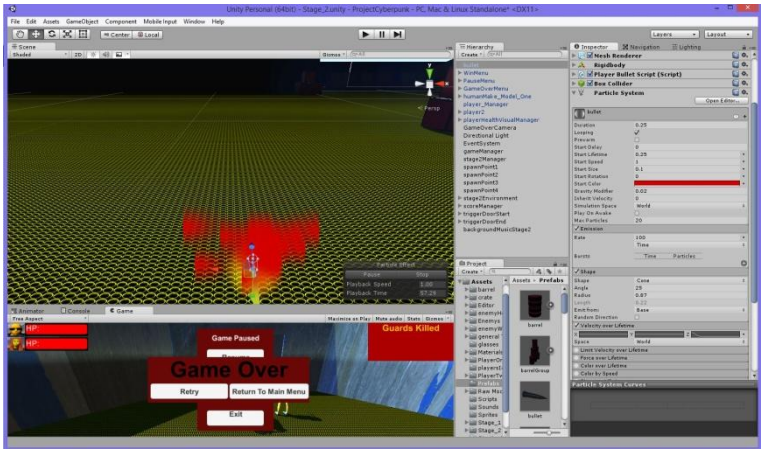

Figure3.3.7-1:Bullet Particle System

Όπως φαίνεται στην παραπάνω εικόνα στην σκηνή ο κόκκινος καπνός είναι η αρχικοποιημένη συνθήκη στην μορφή της σφαίρας, καθώς έχουμε ρυθμίσει την σφαίρα που χρησιμοποιεί η ομάδα των παικτών να κινεί μαζί της τον καπνό μέχρι και το Particle System ενεργοποιείται μόνο όταν χτυπήσει συγκεκριμένα αντικείμενα και αλλάζει χρώμα αναλόγως τι θα χτυπήσει (μαύρο για τοίχους, κόκκινο για εχθρούς και ανενεργό για όλα τα υπόλοιπα στοιχεία). Οι ρυθμίσεις για τα Particle Systems φαίνονται στα δεξιά της παραπάνω εικόνας και επηρεάζουν στο κύριο μέρος τους την μορφή του συστήματος (σχήμα παραγωγής), τον ρυθμό παραγωγής, καθώς και βασικές λειτουργικές ιδιότητες όπως διάρκεια ζωής, χρόνο εκκίνησης, επιρροή δύναμης με την μορφή της βαρύτητας, περιστροφή, ταχύτητα, μέγεθος κλπ. . Όλες οι ρυθμίσεις αυτές μπορούν να επηρεαστούν στα παρακάτω properties που καθορίζουν προαιρετικά συμπεριφορά των σωματιδίων σε σχέση με τον χρόνο.

Στο σημείο αυτό να παραθέσουμε τον κώδικα που τροποποιεί το χρώμα κατά την σύγκρουση με αντικείμενα.

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

public class playerBulletScript : MonoBehaviour {
    ParticleSystem myParticle;
    private string sourceHit; //get the enemy's name.
    Behaviour halo;

    void Start(){
        halo = (Behaviour)this.GetComponent ("Halo");
        myParticle= this.GetComponent<ParticleSystem>();
    }
    void OnCollisionEnter(Collision col){
        if (col.gameObject.tag == "enemy") {
            myParticle.startColor=Color.red;
            myParticle.Play();
            halo.enabled=false;
            Destroy(this.gameObject,0.1f);
        }
        if (col.gameObject.tag == "wall") {
            myParticle.startColor=Color.black;
            myParticle.Play();
            halo.enabled=false;
            Destroy(this.gameObject,0.1f);
        }
        if (col.gameObject.tag == "player") {
            halo.enabled=false;
            Destroy(this.gameObject,0.1f);
        }
    }
    public void setSourceHitName(string name){
        this.sourceHit = name;
    }
    public string getSourceHitName(){
        return sourceHit;
    }
}
```

Figure 3.3.7-2:Bullet Particle System Behavior

Ομοίως δουλέψαμε και το laser effect στα όπλα των αντιπάλων εισάγοντας μια νέα ακόμα έννοια της Unity η οποία ονομάζεται Trail Renderer. Αυτή η ιδιότητα έδωσε την αίσθηση που θα δούμε στην παρακάτω εικόνα.

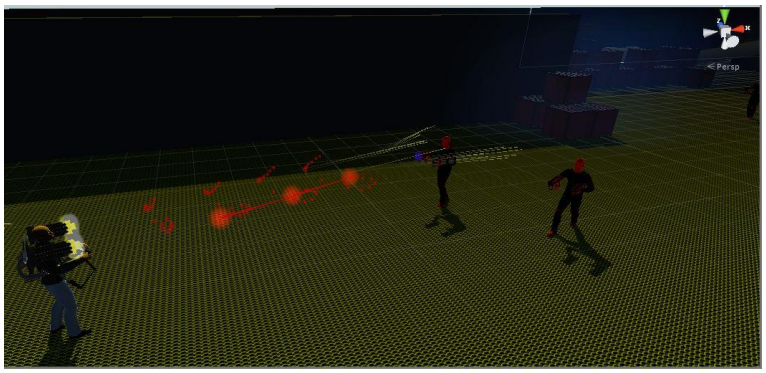

Figure 3.3.7-3:Combat Bullet / Laser Capture

Η παραπάνω εικόνα αποτελεί στιγμιότυπο μάχης μεταξύ της ομάδας και των αντιπάλων στην 2η πίστα. Οι κόκκινες γραμμές αποτελούν τις ακτίνες laser και οι γραμμές στα πλάγια του εχθρού σφαίρες που αστόχησαν. Το σφαιρικό φως που υπάρχει επάνω στο όπλο του παίκτη και στις ακτίνες του εχθρού είναι ένα effect που υποστηρίζουν όλα τα είδη φωτός και ονομάζεται "Draw Halo" (θα δοθεί εξήγηση στο επόμενο υποκεφάλαιο). Την παρουσία των Particles βλέπουμε και στο όπλο του παίκτη μας, που παράγει τα σωματίδια κατά την διάρκεια περιστροφής των κυλίνδρων και δημιουργεί τον καπνό.

Παρακάτω θα δούμε ακόμα σημεία στις πίστες μας που χρησιμοποιήσαμε την συμπεριφορά των Particle Systems με διαφορετικά χαρακτηριστικά.

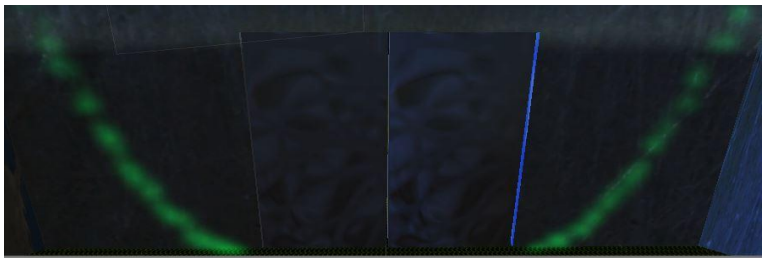

Figure 3.3.7-4: Side Walls Particles

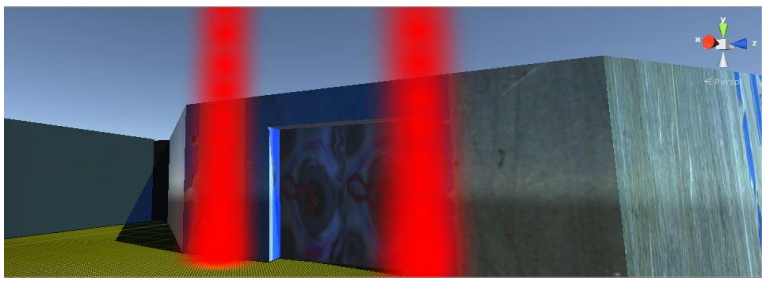

Figure 3.3.7-5: Side Walls Particles 2

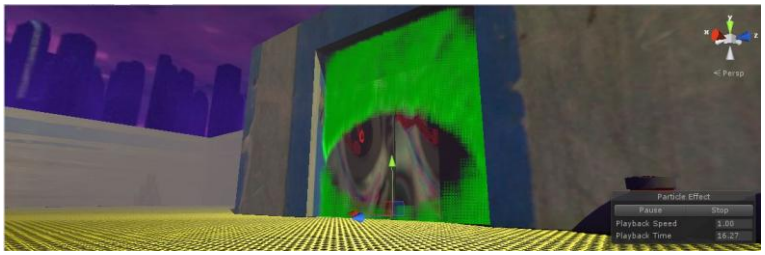

Figure 3.3.7-6:Entrance Door Locker Particle

3.3.8 Lighting

Ο φωτισμός στα βιντεοπαιχνίδια παίζει πολύ σημαντικό ρόλο για την απόδοση ρεαλισμού τόσο στο περιβάλλον όσο και στα κινητά αντικείμενα μέσα στο χώρο, όπως παίκτες, αντίπαλοι κλπ. Η Unity περιέχει 4 είδη φωτισμού όπου κάθε είδος αντιπροσωπεύει διαφορετικό στυλ φωτισμού. Τα είδη φωτισμού είναι: Spot, Directional, Point, Area.

Στα πλαίσια της πτυχιακής εργασίας μας χρησιμοποιήσαμε μόνο Spot και Directional είδη φωτός, καθώς φάνηκε ότι κάλυψαν τις ανάγκες μας. Το Spotlight πρόκειται για είδος φωτός που παράγει διάχυτο φως προς μία κατεύθυνση και λειτουργεί όπως ένας φακός. Η χρήση του αφορά το κομμάτι με τις σφαίρες καθώς εξυπηρετεί παρά πολύ καλά στο κομμάτι του εφέ του πυροβολισμού. Η απόδοση φωτός είναι πολύ μικρή και υπερκαλύπτεται από άλλα είδη φωτός .Στην παρακάτω εικόνα παρουσιάζουμε ένα γενικό παράδειγμα του.

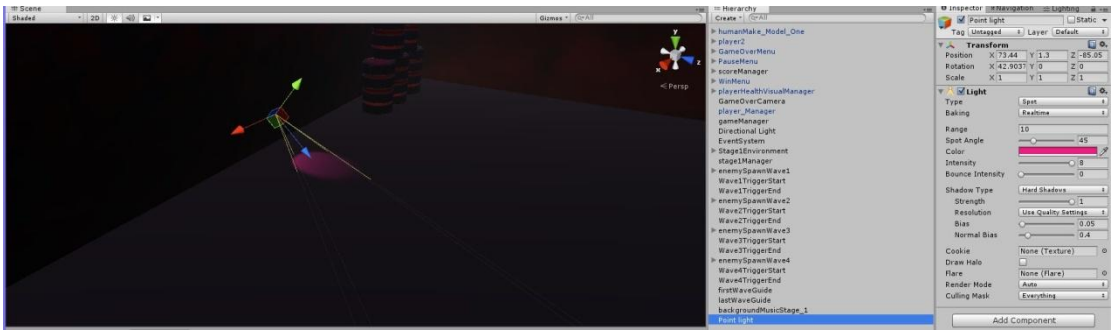

Figure 3.3.8-1: Spotlight

Το Directional Light έπαιξε σημαντικό ρόλο στην εργασία μας καθώς το είδος αυτό αντιπροσωπεύει το πως λειτουργεί ο ήλιος, καθώς υπάρχει δυνατότητα να δημιουργήσει κάποιος Day / Night Cycle, δηλαδή να προσομοιώσει την λειτουργία μέρας / νύχτας. Στα πλαίσια της εργασίας μας το κρατήσαμε σαν στατικό φωτισμό για το περιβάλλον κρατώντας τις ίδιες ρυθμίσεις και στις 2 πίστες . Η παρακάτω εικόνα αποτελεί την επιρροή του Directional Light στο παιχνίδι μας.

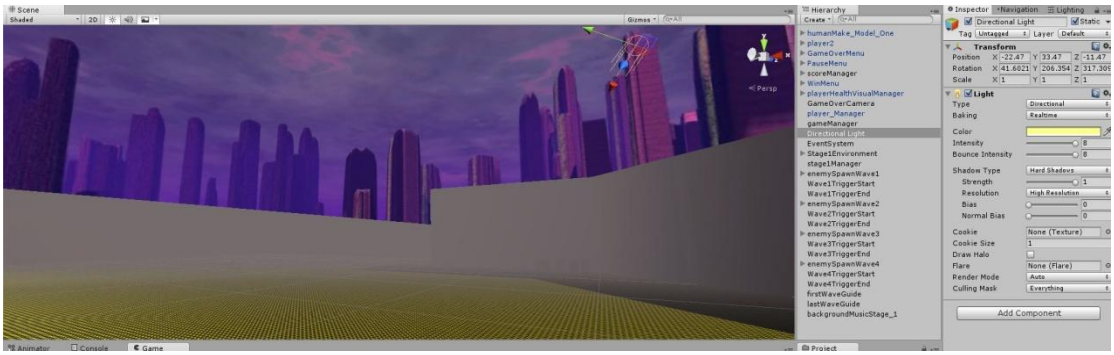

Figure 3.3.8-2:Directional Light

Η επιλογή "Draw Halo" που περιέχει κάθε είδος φωτός δημιουργεί την αίσθηση της έντονης λαμπρότητας που φαίνεται όταν αντικρύσουμε την πηγή φωτός κατευθείαν προς αυτήν και την επίδραση που έχει σε σχέση με το ανθρώπινο μάτι.

3.3.9 Scripting

Σε αυτό το κομμάτι θα ασχοληθούμε με όλα τα τμήματα της εργασίας μας στα οποία καθορίσαμε την συμπεριφορά τους μέσω κώδικα. Ο προγραμματισμός των αντικειμένων σε ένα παιχνίδι είναι από τις βασικότερες πτυχές, καθώς είναι η διαδικασία για να δώσουμε "ζωή " σε ένα γραφικό αντικείμενο. Σε παραπάνω κεφάλαιο περιγράψαμε μέρος της χρήσης κώδικα στην συμπεριφορά της τεχνητής νοημοσύνης τόσο στην ομάδα των παικτών όσο και στην συμπεριφορά των αντιπάλων μας, ενώ στην αρχή του κεφαλαίου συμπεριλάβαμε τον κώδικα που χρησιμοποιήσαμε για την διεπαφή του παιχνιδιού.

Παρακάτω θα παραθέσουμε κώδικα που χρειάστηκε να γράψουμε για την διαδικασία του πυροβολισμού και για τους παίκτες μας, αλλά και για τις μονάδες των εχθρών, καθώς και κώδικα που περιγράφει τον τρόπο με τον οποίο κινείται ο ενεργός μας παίκτης.

```
//active player section--------------------
if (myAgent.enabled == false) {

    //ROTATE CAMERA
    if(Input.mousePresent){
        horizontalMove = horizontalSpeed * Input.GetAxis("Mouse X");
        playerCtrl.transform.Rotate(0, horizontalMove, 0);
    }
    //FORWARD MOVEMENT
    if(Input.GetKey(KeyCode.W)){
        if(Input.mousePresent)
            horizontalMove = horizontalSpeed * Input.GetAxis("Mouse X");

        anim.SetBool("isWalking",true);
        anim.SetBool("isRunning",false);
        //SPRINT FORWARD MOVEMENT
        if(Input.GetKey(KeyCode.CapsLock) && Input.GetKey(KeyCode.W) ){
            anim.SetBool("isRunning",true);
            playerCtrl.transform.Rotate(0, horizontalMove, 0);
            moveDirection = new Vector3(0,0,Input.GetAxis("Vertical"));
            moveDirection=transform.TransformDirection(2*moveDirection*Time.deltaTime);
            playerCtrl.Move(moveDirection);
        }
        playerCtrl.transform.Rotate(0, horizontalMove, 0);
        moveDirection = new Vector3(0,0,Input.GetAxis("Vertical"));
        moveDirection=transform.TransformDirection(moveDirection*Time.deltaTime);
        playerCtrl.Move(moveDirection);
    }
    //IDLE
    if(Input.GetKeyUp(KeyCode.W)){
        anim.SetBool("isWalking",false);
        anim.SetBool("isRunning",false);
    }
}
```

Figure 3.3.9-1:Active Player Movement Behavior

Η παραπάνω εικόνα περιγράφει τον χειρισμό από την μεριά του χρήστη σε σχέση με τον ενεργό παίκτη.

```
//artificial gravity behaviour system for active player obj.
if(playerOne!=null)
    playerOnePos = playerOne.GetComponent<Transform>().position;
if(playerTwo!=null)
    playerTwoPos = playerTwo.GetComponent<Transform>().position;

if (myAgent.enabled == false) {
    if(playerOne!=null){
        if(myAgent.name==playerOne.name){
            RaycastHit hit;
            float currentY=playerOnePos.y;
            Vector3 down = transform.TransformDirection (Vector3.down)*4f;
            if (Physics.Raycast (playerOnePos, down, out hit)) {
                if(hit.collider.gameObject.tag=="ground"){
                    currentY=playerOnePos.y-hit.point.y;
                    playerCtrl.SimpleMove(new Vector3(0,currentY,0 ));
                }
            }
        }
    }
    if(playerTwo!=null){
        if(myAgent.name==playerTwo.name){
            RaycastHit hit;
            float currentY=playerTwo.transform.position.y;
            Vector3 down = transform.TransformDirection (Vector3.down)*4f;
            if (Physics.Raycast (playerTwoPos, down, out hit)) {
                if(hit.collider.gameObject.tag=="ground"){
                    currentY=playerTwoPos.y-hit.point.y;
                    playerCtrl.SimpleMove(new Vector3(0,currentY,0 ));
                }
            }
        }
    }
}
//end of artificial gravity behaviour system.
```

Figure 3.3.9-2: Active Player Gravity Behavior

Σε αυτό το σημείο να εξηγήσουμε τι αντιπροσωπεύει ο παραπάνω κώδικας. Στα πλαίσια της εργασίας μας χρησιμοποιήσαμε ένα component που ονομάζεται Character Controller και περιέχει αρκετές ιδιότητες που απλοποιούν την πολυπλοκότητα της κίνησης (κυρίως σε σχέση με την φυσική). Για λόγους αποφυγής προβλημάτων χρειάστηκε να ορίσουμε τον παραπάνω κώδικα για να κατανοεί ο ενεργός παίκτης την έννοια της βαρύτητας και να προβεί σε υψομετρικές πράξεις που στο αποτέλεσμα τους μας δίνουν μια αληθοφανή συμπεριφορά πτώσης από ένα ψηλότερο επίπεδο σε χαμηλότερο και το ανάποδο σε περίπτωση ανηφορικής κλίσης. Η τεχνική αυτή κοστίζει λιγότερο σε υπολογιστική ισχύ από ότι η κλασσική μέθοδος με την χρήση του component "Rigidbody" που κάνει συνεχή χρήση της βαρύτητας.

Μία ακόμα αλλαγή που κάναμε και υπάγεται στην κατηγορία της συμπεριφοράς στην κίνηση και αφορά τον 2ο παίκτη μας (female) . Προσθέσαμε ένα component στην γεωμετρία των μαλλιών που ονομάζεται "Cloth". Το component αυτό ρυθμίζοντας το κατάλληλα μας επιτρέπει να εφαρμόσουμε συμπεριφορά κίνησης σε ρούχα και οποιαδήποτε επιφάνεια προορίζεται για υφασμάτινη επιφάνεια. Μετά από αρκετές δοκιμές και με κατάλληλες ρυθμίσεις κατάφερε να μας δώσει μια αίσθηση ρεαλισμού στην κίνηση των μαλλιών κατά την διάρκεια της προσομοίωσης της εργασίας μας χωρίς παρέμβαση του προγραμματισμού.

```
private GameObject shoot(){
    RaycastHit hit;
    Vector3 forward = transform.TransformDirection (Vector3.forward)*7f;
    ParticleSystem smokeEffect=this.GetComponent<ParticleSystem>();
    AudioSource minigunAudio= this.GetComponentInParent<AudioSource>();
    if(Input.GetMouseButton(0) && playerAI.gameObject.GetComponent<Animator>().GetBool("isRunning")==false){
        if(minigunAudio.isPlaying==false){
            minigunAudio.volume=1f;
            minigunAudio.Play();
        }
        Cursor.visible = false;
        minigunBarrelOne.transform.Rotate(0,0,45*Time.deltaTime);
        minigunBarrelTwo.transform.Rotate(0,0,45*Time.deltaTime);
        myLight.enabled=true;
        myLight.range= Mathf.Lerp(myLight.range,0.35f,8f*Time.deltaTime);

        if(myLight.range>0.29f)
            myLight.range=0.1f;

        if(myLight.range==0.1f){
            smokeEffect.Play();
            GameObject tempPrefab;
            tempPrefab=Instantiate(bulletPrefab,bulletPos.transform.position,bulletPos.transform.rotation) as GameObject;
            tempPrefab.GetComponent<playerBulletScript>().setSourceHitName("humanMake_Model_One");
            Rigidbody tempRigid;
            tempRigid=tempPrefab.GetComponent<Rigidbody>();
            tempRigid.AddForce(forward*380);
            Destroy(tempPrefab,0.5f);
        }
        if (Physics.Raycast (transform.position, forward, out hit)) {
            if(hit.collider.gameObject.tag=="enemy"){
                pos=hit.collider.gameObject; //position for teammate A.I to seek and destroy
            }
        }
    }
    if(Input.GetMouseButtonUp(0)){
        minigunAudio.Stop();
        smokeEffect.Stop();
        myLight.enabled=false;
    }
    return pos; //returns the pos of enemy the a.i will assist main player.
}//end of shoot
```

Figure 3.3.9-3: Player 1 Active Shooting Behavior

```
//playerShoot-A.I
private void shootAI(){
    RaycastHit hit;
    float distanceHit;
    Vector3 forward = transform.TransformDirection (Vector3.forward) *7f;
    AudioSource minigunAudio= this.GetComponentInParent<AudioSource>();

    if (Physics.Raycast (transform.position, forward, out hit)) {
        if(hit.collider.gameObject.tag=="enemy" && playerAI.gameObject.GetComponent<Animator>().GetBool("isRunning")==false){
            distanceHit=hit.distance;
            if(distanceHit<=10){
                if(minigunAudio.isPlaying==false){
                    minigunAudio.volume=1f;
                    minigunAudio.Play();
                }
                minigunBarrelOne.transform.Rotate(0,0,45*Time.deltaTime);
                minigunBarrelTwo.transform.Rotate(0,0,45*Time.deltaTime);
                myLight.enabled=true;
                myLight.range= Mathf.Lerp(myLight.range,0.35f,4f*Time.deltaTime);
                if(myLight.range>0.29f)
                    myLight.range=0.1f;

                if(myLight.range==0.1f){
                    ParticleSystem smokeEffect=this.GetComponent<ParticleSystem>();
                    smokeEffect.Play();
                    GameObject tempPrefab;
                    tempPrefab=Instantiate(bulletPrefab,bulletPos.transform.position,bulletPos.transform.rotation) as GameObject;
                    tempPrefab.GetComponent<playerBulletScript>().setSourceHitName("humanMake_Model_One");
                    Rigidbody tempRigid;
                    tempRigid=tempPrefab.GetComponent<Rigidbody>();
                    tempRigid.AddForce(forward*380);
                    Destroy(tempPrefab,0.5f);
                }
            }
            if(distanceHit>10){
                if(minigunAudio.isPlaying==true){
                    minigunAudio.Stop();
                }
                myLight.range=0f;
                minigunBarrelOne.transform.Rotate(0,0,0*Time.deltaTime);
                minigunBarrelTwo.transform.Rotate(0,0,0*Time.deltaTime);
            }
        }
    }
}//end of ShootAI
```

Figure 3.3.9-4:Player 1 A.I. Shooting Behavior

Οι 2 παραπάνω φωτογραφίες περιγράφουν τον τρόπο που γίνεται η διαδικασία πυροβολισμού στον 1ο παίκτη σε ενεργή κατάσταση και αντίστοιχα σε κατάσταση τεχνητής νοημοσύνης.

```
//playerShoot-Active
private GameObject shoot(){
    RaycastHit hit;
    Vector3 forward = transform.TransformDirection (Vector3.forward)*7f;
    AudioSource handGunAudio= this.GetComponentInParent<AudioSource>();
    if(Input.GetMouseButton(0)){
        if(handGunAudio.isPlaying==false){
            handGunAudio.volume=1f;
            handGunAudio.Play();
        }
        Cursor.visible = false;
        myLight.enabled=true;
        myLight.range= Mathf.Lerp(myLight.range,0.35f,4f*Time.deltaTime);
        if(myLight.range>0.29f)
            myLight.range=0.1f;
        if(myLight.range==0.1f || myLight.range<=0.2f){
            GameObject tempPrefab;
            tempPrefab=Instantiate(bulletPrefab,bulletPos.transform.position,bulletPos.transform.rotation)as GameObject;
            tempPrefab.GetComponent<playerBulletScript>().setSourceHitName("player2");
            Rigidbody tempRigid;
            tempRigid=tempPrefab.GetComponent<Rigidbody>();
            tempRigid.AddForce(forward*300);
            Destroy(tempPrefab,0.5f);
        }
        if (Physics.Raycast (transform.position, forward, out hit)) {
            if(hit.collider.gameObject.tag=="enemy"){
                pos=hit.collider.gameObject; //enemy's location send to teammates
            }
        }
    }
    if(Input.GetMouseButtonUp(0)){
        handGunAudio.Stop();
        myLight.enabled=false;
    }
    return pos;
}//end of shoot
```

Figure 3.3.9-5:Player 2 Active Shooting Behavior

```
//playerShoot-A.I
private void shootAI(){
    RaycastHit hit;
    float distanceHit;
    Vector3 forward = transform.TransformDirection (Vector3.forward) *7f;
    AudioSource handGunAudio= this.GetComponentInParent<AudioSource>();

    if (Physics.Raycast (transform.position, forward, out hit)) {
        if(hit.collider.gameObject.tag=="enemy"){
            distanceHit=hit.distance;
            if(distanceHit<=7){
                if(handGunAudio.isPlaying==false){
                    handGunAudio.volume=1f;
                    handGunAudio.Play();
                }
                myLight.enabled=true;
                myLight.range= Mathf.Lerp(myLight.range,0.35f,4f*Time.deltaTime);
                if(myLight.range>0.29f)
                    myLight.range=0.1f;

                if(myLight.range==0.1f || myLight.range<=0.2f){
                    GameObject tempPrefab;
                    tempPrefab=Instantiate(bulletPrefab,bulletPos.transform.position,bulletPos.transform.rotation)as GameObject;
                    tempPrefab.GetComponent<playerBulletScript>().setSourceHitName("player2");
                    Rigidbody tempRigid;
                    tempRigid=tempPrefab.GetComponent<Rigidbody>();
                    tempRigid.AddForce(forward*300);
                    Destroy(tempPrefab,0.5f);
                }
            }
            if(distanceHit>7){
                if(handGunAudio.isPlaying==true)
                    handGunAudio.Stop();
                myLight.range=0f;
            }
        }
    }
}//end of ShootAI
```

Figure 3.3.9-6:Player 2 A.I. Shooting Behavior

Αντίστοιχα και εδώ βλέπουμε τις 2 περιπτώσεις πυροβολισμού για τον 2ο παίκτη μας. Στις ακόλουθες εικόνες βλέπουμε τον τρόπο με τον οποίο πυροβολούν οι αντίπαλοι μας και στην περίπτωση των φρουρών λειτουργούν σε κοινό κώδικα που ελέγχει την πίστα που παίζουμε εκείνη την στιγμή, καθώς και τον τρόπο που λειτουργεί ο θεραπευτής κατά την διάρκεια της θεραπείας σε μονάδα της ομάδας του.

```
//enemyShoot
private void shootAI(){
    RaycastHit hit;
    float distanceHit;
    Vector3 forward = transform.TransformDirection (Vector3.forward) *12f;
    AudioSource laserAudio= this.GetComponentInParent<AudioSource>();

    if (Physics.Raycast (transform.position, forward, out hit)) {
        if(hit.collider.gameObject.tag=="Player"){
            enemyGeneral.transform.LookAt(hit.transform.position); //lock on target.
            enemyGeneral.GetComponentInParent<enemyGeneral>().setEnemyName(hit.collider.gameObject.name);
            distanceHit=hit.distance;
            if(distanceHit<=15){
                if(laserAudio.isPlaying==false){
                    laserAudio.volume=1f;
                    laserAudio.Play();
                }
                myLight.enabled=true;
                myLight.range= Mathf.Lerp(myLight.range,0.35f,4f*Time.deltaTime);
                if(myLight.range>0.29f)
                    myLight.range=0.1f;

                if(myLight.range==0.1f || myLight.range<=0.2f){
                    GameObject tempPrefab;
                    tempPrefab=Instantiate(bulletPrefab,bulletPos.transform.position,bulletPos.transform.rotation)as GameObject;
                    tempPrefab.GetComponent<enemyBullet>().setHitName(enemyGeneral.name);
                    Rigidbody tempRigid;
                    tempRigid=tempPrefab.GetComponent<Rigidbody>();
                    tempRigid.AddForce(forward*150);
                    Destroy(tempPrefab,0.5f);
                }
            }
            if(distanceHit>15){
                if(laserAudio.isPlaying==true){
                    laserAudio.Stop();
                }
                myLight.range=0f;
            }
        }
    }
}//end of ShootAI
```

Figure 3.3.9-7:General Shooting Behavior

```
//enemyShoot
private void shootAI(){
    RaycastHit hit;
    float distanceHit;
    Vector3 forward = transform.TransformDirection (Vector3.forward) *12f;
    AudioSource laserAudio= this.GetComponentInParent<AudioSource>();

    if (Physics.Raycast (transform.position, forward, out hit)) {
        if(hit.collider.gameObject.tag=="Player"){
            enemyLieutenant.transform.LookAt(hit.transform.position); //lock on target.
            enemyLieutenant.GetComponentInParent<enemyLieutenant>().setEnemyName(hit.collider.gameObject.name);
            distanceHit=hit.distance;
            if(distanceHit<=15){
                if(laserAudio.isPlaying==false){
                    laserAudio.volume=1f;
                    laserAudio.Play();
                }
                myLight.enabled=true;
                myLight.range= Mathf.Lerp(myLight.range,0.35f,4f*Time.deltaTime);
                if(myLight.range>0.29f)
                    myLight.range=0.1f;

                if(myLight.range==0.1f || myLight.range<=0.2f){
                    GameObject tempPrefab;
                    tempPrefab=Instantiate(bulletPrefab,bulletPos.transform.position,bulletPos.transform.rotation)as GameObject;
                    tempPrefab.GetComponent<enemyBullet>().setHitName(enemyLieutenant.name);
                    Rigidbody tempRigid;
                    tempRigid=tempPrefab.GetComponent<Rigidbody>();
                    tempRigid.AddForce(forward*150);
                    Destroy(tempPrefab,0.5f);
                }
            }
            if(distanceHit>15){
                if(laserAudio.isPlaying==true){
                    laserAudio.Stop();
                }
                myLight.range=0f;
            }
        }
    }
}//end of ShootAI
```

Figure 3.3.9-8:Lieutenant Shooting Behavior

```
//enemyShoot
private void shootAI(){
    RaycastHit hit;
    float distanceHit;
    Vector3 forward = transform.TransformDirection (Vector3.forward) *12f;
    AudioSource laserAudio= this.GetComponentInParent<AudioSource>();

    if (Physics.Raycast (transform.position, forward, out hit)) {
        if(hit.collider.gameObject.tag=="Player"){
            enemyGuard.transform.LookAt(hit.transform.position); //lock on target.
            if(Application.loadedLevelName=="Stage_1")
                enemyGuard.GetComponentInParent<enemyGuard>().setEnemyName(hit.collider.gameObject.name);
            if(Application.loadedLevelName=="Stage_2")
                enemyGuard.GetComponentInParent<enemyGuardStandAlone>().setEnemyName(hit.collider.gameObject.name);

            distanceHit=hit.distance;
            if(distanceHit<=15){
                if(laserAudio.isPlaying==false){
                    laserAudio.volume=1f;
                    laserAudio.Play();
                }
                myLight.enabled=true;
                myLight.range= Mathf.Lerp(myLight.range,0.35f,4f*Time.deltaTime);
                if(myLight.range>0.29f)
                    myLight.range=0.1f;

                if(myLight.range==0.1f || myLight.range<=0.2f){
                    GameObject tempPrefab;
                    tempPrefab=Instantiate(bulletPrefab,bulletPos.transform.position,bulletPos.transform.rotation)as GameObject;
                    tempPrefab.GetComponent<enemyBullet>().setHitName(enemyGuard.name);
                    Rigidbody tempRigid;
                    tempRigid=tempPrefab.GetComponent<Rigidbody>();
                    tempRigid.AddForce(forward*150);
                    Destroy(tempPrefab,0.5f);
                }
            }
            if(distanceHit>15){
                if(laserAudio.isPlaying==true){
                    laserAudio.Stop();
                }
                myLight.range=0f;
            }
        }
    }
}//end of ShootAI
```

Figure 3.3.9-9:Guard Shooting Behavior(Stage 1 & 2)

```
//enemyHeal
private void healAI(){
    RaycastHit hit;
    GameObject tempTeamMember=unit;
    if (Vector3.Distance (enemyHealer.transform.position, unit.transform.position) < 10) {
        Vector3 forward = transform.TransformDirection (Vector3.forward).normalized * 5f;
        Vector3 backwards =transform.TransformDirection(Vector3.back)*5f;
        if (Physics.Raycast (transform.position, forward, out hit)) {
            if (hit.collider.gameObject.name == unit.name) {
                enemyHealer.transform.LookAt(unit.transform.position);
                GameObject tempPrefab;
                tempPrefab = Instantiate (healPrefab, healPos.transform.position, healPos.transform.rotation)as GameObject;
                Rigidbody tempRigid;
                tempRigid = tempPrefab.GetComponent<Rigidbody> ();
                //soundclip implement
                tempRigid.AddForce (forward *50);
                if(Physics.Raycast(tempTeamMember.transform.position,backwards,out hit)){
                    if(hit.collider.gameObject.name==enemyHealer.name){
                        tempRigid.AddForce (backwards *50);
                    }
                }
                Destroy (tempPrefab, 0.5f);
            }
        }
    }
}//end of healAI
```

Figure 3.3.9-10:Healer Healing Behavior

Προγραμματισμός απαιτήθηκε ακόμα και στις διαχείριση των επίπεδων μας για να εξυπηρετήσουν τον σκοπό της κάθε πίστας, όπως το πότε θα εμφανίζονται οι αντίπαλοί μας στην πίστα κλπ. Για την 1η πίστα προγραμματίσαμε το πότε και που θα εμφανίζονται οι ομάδες των εχθρών, καθώς και τα κριτήρια για να προχωρήσουμε βαθύτερα στην πίστα αυτή μέχρι και τα κριτήρια για τον τερματισμό της. Παρακάτω θα δούμε εικόνες που περιγράφουν τον κώδικα που γράψαμε για αυτές τις ανάγκες.

```
public class stageOneManager : MonoBehaviour {
    private GameObject wallDoorLeft; //vanish doorLeft
    private GameObject wallDoorRight;//vanish doorRight
    private GameObject wallDoorLeftExit;
    private GameObject wallDoorRightExit;
    private ParticleSystem frontDoorParticle;
    private ParticleSystem exitDoorParticle;
    private ParticleSystem waveOneSpawn;
    private ParticleSystem waveTwoSpawn;
    private ParticleSystem waveThreeSpawn;
    private ParticleSystem waveFourSpawn;
    public int countWaves; //count the wave number passed.

    void Start () {
        countWaves = 0; //counter to trigger Success Pop up.
        frontDoorParticle = GameObject.Find ("firstWaveGuide").GetComponent<ParticleSystem> ();
        exitDoorParticle = GameObject.Find ("lastWaveGuide").GetComponent<ParticleSystem> ();
        waveOneSpawn = GameObject.Find ("enemySpawnWave1").GetComponent<ParticleSystem> ();
        waveTwoSpawn = GameObject.Find ("enemySpawnWave2").GetComponent<ParticleSystem> ();
        waveThreeSpawn = GameObject.Find ("enemySpawnWave3").GetComponent<ParticleSystem> ();
        waveFourSpawn = GameObject.Find ("enemySpawnWave4").GetComponent<ParticleSystem> ();
        wallDoorLeft = GameObject.Find ("leftDoorEntrance");
        wallDoorRight = GameObject.Find ("rightDoorEntrance");
        wallDoorLeftExit = GameObject.Find ("leftDoorExit");
        wallDoorRightExit = GameObject.Find ("rightDoorExit");
    }
    void Update () {
        if (countWaves==1) { //what happens when complete 1st wave
            //Front Gate.
            frontDoorParticle.Stop();
            waveOneSpawn.Stop();
            wallDoorLeft.GetComponent<BoxCollider>().enabled=false;
            wallDoorRight.GetComponent<BoxCollider>().enabled=false;
            wallDoorLeft.GetComponent<MeshRenderer>().enabled=false;
            wallDoorRight.GetComponent<MeshRenderer>().enabled=false;
        }
        if (countWaves == 2) {
            waveTwoSpawn.Stop ();
        }
        if (countWaves == 3) {
            waveThreeSpawn.Stop();
        }
        if (countWaves == 4) { //what happens when complete 4th wave
            //Exit Gate.
            waveFourSpawn.Stop();
            exitDoorParticle.Stop();
            wallDoorLeftExit.GetComponent<BoxCollider>().enabled=false;
            wallDoorRightExit.GetComponent<BoxCollider>().enabled=false;
            wallDoorLeftExit.GetComponent<MeshRenderer>().enabled=false;
            wallDoorRightExit.GetComponent<MeshRenderer>().enabled=false;
        }
    }
    public void setCountWaves(int count){
        countWaves = count;
    }
}
```

Figure 3.3.9-11:Stage1 Manager

```
using UnityEngine;
using System.Collections;

public class playerTriggerWaveOne : MonoBehaviour {
    public GameObject enemyGeneralSpPoint;
    public GameObject enemyHealerSpPoint;
    public GameObject enemyLieutenantSpPoint;
    public GameObject enemyGuardOneSpPoint;
    public GameObject enemyGuardTwoSpPoint;
    public GameObject enemyGeneralPref;
    public GameObject enemyLieutenantPref;
    public GameObject enemyHealerPref;
    public GameObject enemyGuardOnePref;
    public GameObject enemyGuardTwoPref;
    private GameObject stageManager;
    private GameObject myCollider;
    void Start () {
        stageManager=GameObject.Find("stage1Manager");
        myCollider = this.gameObject;
    }
    void OnTriggerEnter(Collider col){
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave1TriggerStart")){
            myCollider.transform.Translate(0,5,0);
            Instantiate(enemyLieutenantPref,enemyLieutenantSpPoint.transform.position,enemyLieutenantSpPoint.transform.rotation);
            Instantiate(enemyGeneralPref,enemyGeneralSpPoint.transform.position,enemyGeneralSpPoint.transform.rotation);
            Instantiate(enemyHealerPref,enemyHealerSpPoint.transform.position,enemyHealerSpPoint.transform.rotation);
            Instantiate(enemyGuardOnePref,enemyGuardOneSpPoint.transform.position,enemyGuardOneSpPoint.transform.rotation);
            Instantiate(enemyGuardTwoPref,enemyGuardTwoSpPoint.transform.position,enemyGuardTwoSpPoint.transform.rotation);
            Destroy(this.gameObject);
        }
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave1TriggerEnd")){
            if (GameObject.Find("enemyGeneral(Clone)")== null && GameObject.Find("enemyLieutenant(Clone)") == null && GameObject.Find("enemyHealer(Clone)") == null && GameObject.Find("enemyGuard_1(Clone)") == null && GameObject.Find("enemyG
                stageManager.GetComponent<stageOneManager>().setCountWaves(1);
                myCollider.transform.Translate(0,5,0);
                Destroy(this.gameObject);
            }
        }
    }
}
```

Figure 3.3.9-12: Wave 1 Function Trigger

```
using UnityEngine;
using System.Collections;

public class playerTriggerWaveTwo : MonoBehaviour {
    public GameObject enemyGeneralSpPoint;
    public GameObject enemyHealerSpPoint;
    public GameObject enemyLieutenantSpPoint;
    public GameObject enemyGuardOneSpPoint;
    public GameObject enemyGuardTwoSpPoint;
    public GameObject enemyGeneralPref;
    public GameObject enemyLieutenantPref;
    public GameObject enemyHealerPref;
    public GameObject enemyGuardOnePref;
    public GameObject enemyGuardTwoPref;
    private GameObject stageManager;
    private GameObject myCollider;
    void Start () {
        stageManager=GameObject.Find("stage1Manager");
        myCollider = this.gameObject;
    }
    void OnTriggerEnter(Collider col){
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave2TriggerStart")){
            myCollider.transform.Translate(0,5,0);
            Instantiate(enemyLieutenantPref,enemyLieutenantSpPoint.transform.position,enemyLieutenantSpPoint.transform.rotation);
            Instantiate(enemyGeneralPref,enemyGeneralSpPoint.transform.position,enemyGeneralSpPoint.transform.rotation);
            Instantiate(enemyHealerPref,enemyHealerSpPoint.transform.position,enemyHealerSpPoint.transform.rotation);
            Instantiate(enemyGuardOnePref,enemyGuardOneSpPoint.transform.position,enemyGuardOneSpPoint.transform.rotation);
            Instantiate(enemyGuardTwoPref,enemyGuardTwoSpPoint.transform.position,enemyGuardTwoSpPoint.transform.rotation);
            Destroy(this.gameObject);
        }
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave2TriggerEnd")){
            if (GameObject.Find("enemyGeneral(Clone)")== null && GameObject.Find("enemyLieutenant(Clone)") == null && GameObject.Find("enemyHealer(Clone)") == null && GameObject.Find("enemyGuard_1(Clone)") == null && GameObject.Find("enemyGuard_2(Clone)") == null) {
                stageManager.GetComponent<stageOneManager>().setCountWaves(2);
                myCollider.transform.Translate(0,5,0);
                Destroy(this.gameObject);
            }
        }
    }
}
```

Figure 3.3.9-13:Wave 2 Function Trigger

```
using UnityEngine;
using System.Collections;

public class playerTriggerWaveThree : MonoBehaviour {
    public GameObject enemyGeneralSpPoint;
    public GameObject enemyHealerSpPoint;
    public GameObject enemyLieutenantSpPoint;
    public GameObject enemyGuardOneSpPoint;
    public GameObject enemyGuardTwoSpPoint;
    public GameObject enemyGeneralPref;
    public GameObject enemyLieutenantPref;
    public GameObject enemyHealerPref;
    public GameObject enemyGuardOnePref;
    public GameObject enemyGuardTwoPref;
    private GameObject stageManager;
    private GameObject myCollider;
    void Start () {
        stageManagerwGameObject.Find("stage1Manager");
        myCollider = this.gameObject;
    }
    void OnTriggerEnter(Collider col){
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave3TriggerStart")){
            myCollider.transform.Translate(0,5,0);
            Instantiate(enemyLieutenantPref,enemyLieutenantSpPoint.transform.position,enemyLieutenantSpPoint.transform.rotation);
            Instantiate(enemyGeneralPref,enemyGeneralSpPoint.transform.position,enemyGeneralSpPoint.transform.rotation);
            Instantiate(enemyHealerPref,enemyHealerSpPoint.transform.position,enemyHealerSpPoint.transform.rotation);
            Instantiate(enemyGuardOnePref,enemyGuardOneSpPoint.transform.position,enemyGuardOneSpPoint.transform.rotation);
            Instantiate(enemyGuardTwoPref,enemyGuardTwoSpPoint.transform.position,enemyGuardTwoSpPoint.transform.rotation);
            Destroy(this.gameObject);
        }
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave3TriggerEnd")){
            if (GameObject.Find("enemyGeneral(Clone)")== null && GameObject.Find("enemyLieutenant(Clone)") == null && GameObject.Find("enemyHealer(Clone)") == null && GameObject.Find("enemyGuard_1(Clone)") == null && GameObject.Find("enemyGuard_2(Clone)") == null) {
                stageManager.GetComponent<stageOneManager>().setCountWaves(3);
                myCollider.transform.Translate(0,5,0);
                Destroy(this.gameObject);
            }
        }
    }
}
```

Figure 3.3.9-14:Wave 3 Function Trigger

```
using UnityEngine;
using System.Collections;

public class playerTriggerWaveFourth : MonoBehaviour {
    public GameObject enemyGeneralSpPoint;
    public GameObject enemyHealerSpPoint;
    public GameObject enemyLieutenantSpPoint;
    public GameObject enemyGuardOneSpPoint;
    public GameObject enemyGuardTwoSpPoint;
    public GameObject enemyGeneralPref;
    public GameObject enemyLieutenantPref;
    public GameObject enemyHealerPref;
    public GameObject enemyGuardOnePref;
    public GameObject enemyGuardTwoPref;
    private GameObject stageManager;
    private GameObject myCollider;
    void Start () {
        stageManager=GameObject.Find("stageIManager");
        myCollider = this.gameObject;
    }
    void OnTriggerEnter(Collider col){
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave4TriggerStart")){
            myCollider.transform.Translate(0,5,0);
            Instantiate(enemyLieutenantPref,enemyLieutenantSpPoint.transform.position,enemyLieutenantSpPoint.transform.rotation);
            Instantiate(enemyGeneralPref,enemyGeneralSpPoint.transform.position,enemyGeneralSpPoint.transform.rotation);
            Instantiate(enemyHealerPref,enemyHealerSpPoint.transform.position,enemyHealerSpPoint.transform.rotation);
            Instantiate(enemyGuardOnePref,enemyGuardOneSpPoint.transform.position,enemyGuardOneSpPoint.transform.rotation);
            Instantiate(enemyGuardTwoPref,enemyGuardTwoSpPoint.transform.position,enemyGuardTwoSpPoint.transform.rotation);
            Destroy(this.gameObject);
        }
        if(col.gameObject.tag=="Player" && myCollider==GameObject.Find("Wave4TriggerEnd")){
            if (GameObject.Find("enemyGeneral(Clone)")== null && GameObject.Find("enemyLieutenant(Clone)") == null && GameObject.Find("enemyHealer(Clone)") == null && GameObject.Find("enemyGuard_1(Clone)") == null && GameObject.Find("enemyGuard_2(Clone)") == null) {
                stageManager.GetComponent<stageOneManager>().setCountWaves(4);
                myCollider.transform.Translate(0,5,0);
                Destroy(this.gameObject);
            }
        }
    }
}
```

Figure 3.3.9-15:Wave 4 Function Trigger

Οι παραπάνω εικόνες περιγράφουν τον κώδικα στις 4 ζώνες που πρέπει να περάσει με την σειρά η ομάδα και τον διαχειριστή της πίστας που ελέγχει τα κριτήρια για να τερματίσουμε με επιτυχία το επίπεδο αυτό.

Στην 2η πίστα έχουμε το σημείο εκκίνησης που απενεργοποιεί την πόρτα στην αρχή και υποδεικνύει την εκκίνηση της πίστας και το σημείο τερματισμού που ξεκλειδώνει μόνο όταν επιτύχουμε τον σκοπό μας, καθώς και ένα διαχειριστή που ανά διαστήματα φέρνει αντιπάλους στο παιχνίδι στα 4 σημεία που έχουμε ορίσει να αρχικοποιούνται όπως θα δούμε στις παρακάτω εικόνες.

```
using UnityEngine;
using System.Collections;

public class stageTwoManager : MonoBehaviour {
    public GameObject spOne;
    public GameObject spTwo;
    public GameObject spThree;
    public GameObject spFour;
    public GameObject guardSolo;
    private GameObject enemyGuardSpOne;
    private GameObject enemyGuardSpTwo;
    private GameObject enemyGuardSpThree;
    private GameObject enemyGuardSpFour;
    private float timeCounter;
    private float interval;
    void Start () {
        interval = 1;
        timeCounter = 0f;
    }
    void Update () {
        timeCounter = Time.timeSinceLevelLoad;//time count.

        if (timeCounter > (10f*interval)){ //time to instantiate each time
            interval+=1; //incrementation of intervals
            if(enemyGuardSpOne==null){
                enemyGuardSpOne = Instantiate (guardSolo, spOne.transform.position, spOne.transform.rotation)as GameObject;
            }
            if(enemyGuardSpTwo==null){
                enemyGuardSpTwo = Instantiate (guardSolo, spTwo.transform.position, spTwo.transform.rotation)as GameObject;
            }
            if(enemyGuardSpThree==null){
                enemyGuardSpThree = Instantiate (guardSolo, spThree.transform.position, spThree.transform.rotation)as GameObject;
            }
            if(enemyGuardSpFour==null){
                enemyGuardSpFour = Instantiate (guardSolo, spFour.transform.position, spFour.transform.rotation)as GameObject;
            }
        }
    }
}
```

Figure 3.3.9-16:Stage 2 Manager

```
using UnityEngine;
using System.Collections;

public class startTrigger : MonoBehaviour {
    public GameObject startDoorLeft;
    public GameObject startDoorRight;
    private ParticleSystem initParticleLeft;
    private ParticleSystem initParticleRight;
    void Start(){
        initParticleLeft = GameObject.Find("particleStartLeft").GetComponent<ParticleSystem> ();
        initParticleRight = GameObject.Find("particleStartRight").GetComponent<ParticleSystem> ();
    }
    void OnTriggerEnter(Collider col){
        if (col.gameObject.tag == "Player") {
            initParticleLeft.Stop();
            initParticleRight.Stop();
            startDoorLeft.SetActive (false);
            startDoorRight.SetActive (false);
        }
    }
}
```

Figure 3.3.9-17:Stage 2 Starting Behavior

```
using UnityEngine;
using System.Collections;

public class endTrigger : MonoBehaviour {
    public GameObject endDoorLeft;
    public GameObject endDoorRight;
    private ParticleSystem endParticleLeft;
    private ParticleSystem endParticleRight;
    private int countGuard;
    void Start(){
        endParticleLeft = GameObject.Find ("particleEndLeft").GetComponent<ParticleSystem> ();
        endParticleLeft.Pause ();
        endParticleRight = GameObject.Find ("particleEndRight").GetComponent<ParticleSystem> ();
        endParticleRight.Pause ();
    }

    void Update(){
        countGuard = GameObject.Find ("gameManager").GetComponent<gameManager> ().guardsKilled;
        if (countGuard >= 50) {
            endParticleLeft.Play();
            endParticleRight.Play();
        }
    }
    void OnTriggerEnter(Collider col){
        countGuard = GameObject.Find ("gameManager").GetComponent<gameManager> ().guardsKilled;
        if (col.gameObject.tag == "Player" && countGuard>=50) {
            endDoorLeft.SetActive (false);
            endDoorRight.SetActive (false);
        }
    }

}
```

Figure 3.3.9-18:Stage 2 End Behavior

Τέλος έχουμε τον κώδικα που διαφοροποιεί το πεδίο της διεπαφής μας για τον σκοπό σε κάθε πίστα αντίστοιχα και ενημερώνεται για την πρόοδο του στόχου μας όπως βλέπουμε στην ακόλουθη εικόνα

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

public class score : MonoBehaviour {
    //stage 1 variables.
    private int wavesCleared;
    private string finalTextStgOne;
    //stage 2 variables.
    private int numOfGuards;
    private string finalTextStgTwo;
    //generic variables.
    private Text text;
    private Text aboveText;

    void Start () {
        if(Application.loadedLevelName=="Stage_1")
        wavesCleared = GameObject.Find ("stage1Manager").GetComponent<stageOneManager> ().countWaves;//number of waves crossed.
        if(Application.loadedLevelName=="Stage_2")
        numOfGuards = GameObject.Find ("gameManager").GetComponent<gameManager> ().guardsKilled;//number of guards killed.

        aboveText = GameObject.Find ("fixedText").GetComponent<Text> (); //fixed Text logo
        text = this.GetComponent<Text> (); //gets the field of score text
        finalTextStgOne="/4";
        finalTextStgTwo="/50";
    }
    void Update () {
        if (Application.loadedLevelName == "Stage_1") {
            wavesCleared = GameObject.Find ("stage1Manager").GetComponent<stageOneManager> ().countWaves;
            text.text = wavesCleared + finalTextStgOne;
            if (wavesCleared >=4) {
                aboveText.text = "Stage Complete";
                finalTextStgTwo = "Guide to the Exit";
                text.text = finalTextStgTwo;
            }
        }
        if (Application.loadedLevelName == "Stage_2") {
            numOfGuards = GameObject.Find ("gameManager").GetComponent<gameManager> ().guardsKilled;
            text.text = numOfGuards + finalTextStgTwo;
            if (numOfGuards >= 50) {
                aboveText.text = "Stage Complete";
                finalTextStgTwo = "Guide to the Exit";
                text.text = finalTextStgTwo;
            }
        }
    }

}
```

Figure 3.3.9-19:Score Manager

3.3.10 Skybox

Το Skybox πρόκειται για μία μέθοδο που χρησιμοποιείται στην Unity και πιθανότατα σε άλλες game engines που δημιουργεί γύρω από την πίστα μας μια αίσθηση τρισδιάστατου background. Το Skybox χρησιμοποιεί 6 επιφάνειες, όσες δηλαδή περιλαμβάνει ο κύβος (box) και κάθε μία από αυτές αντιπροσωπεύει μία πλευρά του. Συνήθως επιλέγουμε σωστά κομμένη εικόνα για να μην παρουσιάσει σφάλμα στην αποτύπωση της. Στα πλαίσια της εργασίας μας στην 1η πίστα πήραμε έτοιμο skybox από ένα μέλος της κοινότητας της Unity που στο τέλος του εγγράφου θα παραθέσουμε link για την ιστοσελίδα, ενώ για την 2η πίστα μας χρησιμοποιήσαμε το αρχικό που έχει μέσα η Unity.Παρακάτω θα δούμε μία εικόνα αποτύπωσης skybox μηχανισμού.

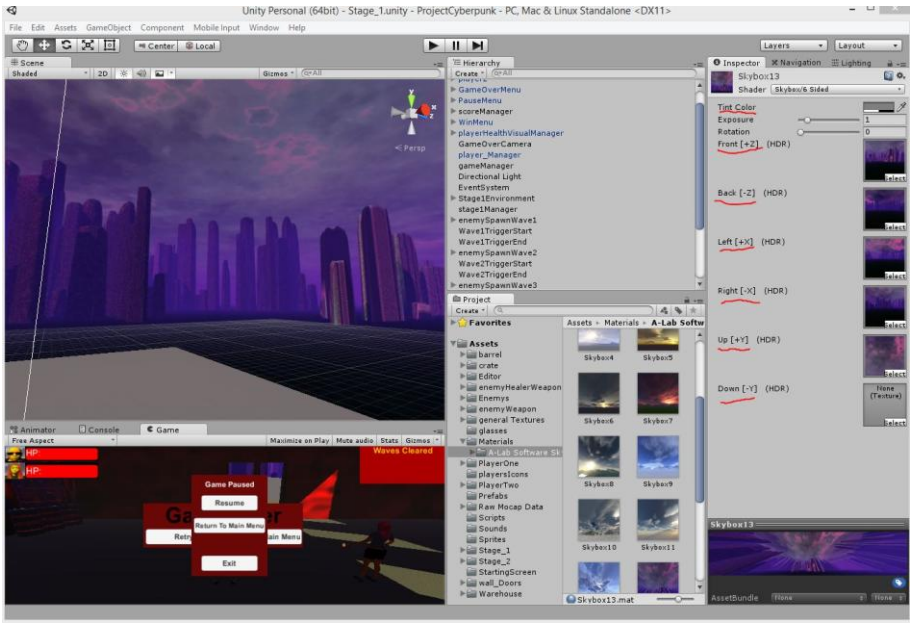

Figure 3.3.10-1:Skybox Stage 1

Όπως βλέπουμε στα δεξιά της φωτογραφίας κάθε εικόνα με κόκκινη υπογράμμιση αντιστοιχεί στην πλευρά που θα προβληθεί και το σύνολο τους αποτελεί μία εξομαλυμένη ενιαία εικόνα, ενώ η ιδιότητα "TintColor" επιτρέπει να δώσουμε απόχρωση στο σύνολο του skybox.Ομοίως και στην 2η πίστα έχουμε skybox το αρχικό που δίνεται από την game engine όπως φαίνεται παρακάτω.

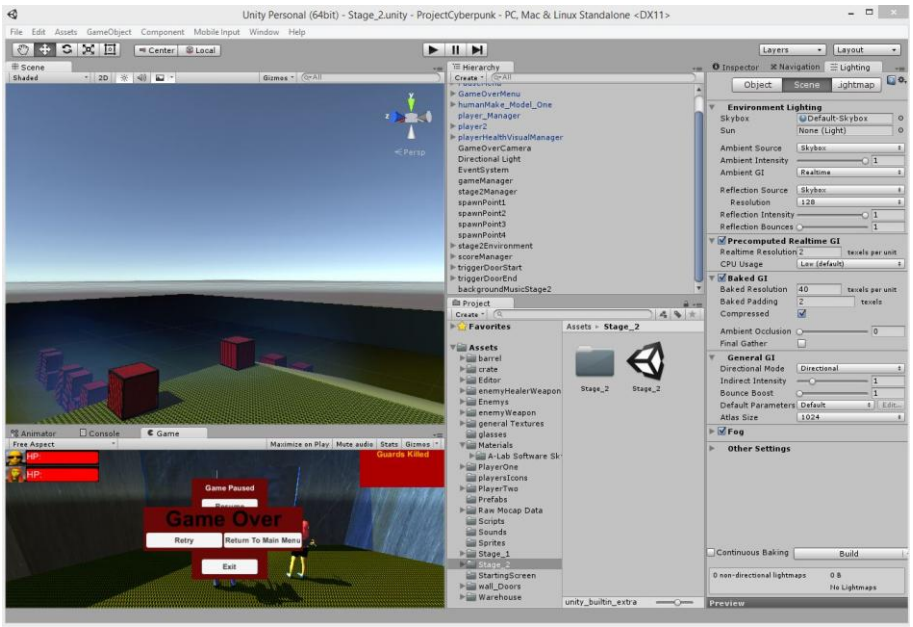

Figure 3.3.10-2: Skybox Stage 2

Κεφάλαιο 4

Αποτελέσματα

4.1. Δυνατότητες της Εφαρμογής

Το παιχνίδι ξεκινάει φέρνοντας στο προσκήνιο το κύριο μενού με την επιλογή να ξεκινήσουμε το παιχνίδι ή να κλείσουμε την εφαρμογή. Πατάμε στο κουμπί να ξεκινήσουμε το παιχνίδι και αυτόματα μας μεταφέρει στην 1η πίστα που βλέπουμε την ομάδα μας από κάμερα 3ου προσώπου. Χρησιμοποιώντας το κουμπί "W" προχωράμε ευθεία ενώ για κατευθύνουμε τον ενεργό χαρακτήρα μας αριστερά ή δεξιά κινούμε το ποντίκι μας προς την αντίστοιχα επιθυμητή κατεύθυνση. Επίσης έχουμε την δυνατότητα να κάνουμε τον ενεργό χαρακτήρα μας να τρέχει έχοντας ταυτόχρονα πατημένα τα κουμπιά "W" και "Caps Lock" .

Για τον πυροβολισμό επιλέγουμε το αριστερό κουμπί του ποντικιού μας και το κρατάμε πατημένο για όση διάρκεια θέλουμε να πυροβολούμε. Σε κάθε πίστα επιλέξαμε ένα κομμάτι από το youtube και το χρησιμοποιήσαμε σαν μουσική επένδυση για κάθε μία από τις 2 πίστες, όπως αντίστοιχα κάναμε και για τους ήχους πυροβολισμού στους εχθρούς και στους χαρακτήρες μας. Στην 1η πίστα ξεκινάμε μέσα σε μία αποθήκη και σκοπός μας είναι να εξουδετερώσουμε σταδιακά όλες τις ομάδες αντιπάλων που θα συναντήσουμε μέχρι το τέλος της πίστας.

Η 2η πίστα ξεκινά σε σημείο που αποτελεί συνέχεια της 1ης πίστας όμως ο σκοπός είναι διαφορετικός. Πρέπει να αντέξει η ομάδα για αρκετό χρόνο καθώς πρέπει να εξουδετερώσει ένα μεγάλο αριθμό από αντιπάλους. Στο παιχνίδι αυτό μας δίνεται η δυνατότητα εναλλαγής χαρακτήρων μέσω των κουμπιών "F1" για επιλογή ενεργού παίκτη τον 1ο (male) και "F2" για επιλογή ενεργού παίκτη τον 2ο (female). Ο ενεργός παίκτης έχει την δυνατότητα να δώσει διαταγές για επίθεση και ανασύνταξη της ομάδας. Το παιχνίδι σε πολλά σημεία θα απαιτήσει γρήγορες εναλλαγές των παικτών για επανατοποθέτηση στον χώρο προκειμένου να γλιτώσουν, καθώς χρειάζεται στρατηγική σκέψη (κυρίως στην 1η πίστα) για την σειρά με την οποία πρέπει να εξουδετερώσουμε τους εχθρούς μας.

Τέλος η εφαρμογή μας προσφέρει την δυνατότητα μενού παύσης σε περίπτωση που θέλουμε να κλείσουμε το παιχνίδι, να επιστρέψουμε στο αρχικό μενού ή και απλώς να σταματήσουμε προσωρινά την ροή του παιχνιδιού. Η μετάβαση στο μενού παύσης γίνεται μέσω του κουμπιού "Esc" και χρησιμοποιώντας το ίδιο κουμπί επαναφέρουμε την ροή σε πραγματικό χρόνο, καθώς εναλλακτικά μέσω του κουμπιού "Resume" μέσα από το μενού αυτό.

4.2. Δυσκολίες που αντιμετωπίστηκαν

Το 1ο κομμάτι που αντιμετωπίσαμε δυσκολία ήταν στο κομμάτι των ανθρωποειδών μοντέλων μας, καθώς υπήρχε έλλειψη γνώσης και χρειάστηκε συνεχής προσπάθεια για ευρύτερη κατανόηση.

Το 2ο κομμάτι που αντιμετωπίσαμε δυσκολία ήταν η δημιουργία των animations στον 2ο χαρακτήρα μας (female) . Η γνώση πάνω στην κατασκευή humanoid animation ήταν ελάχιστη και ήταν απαραίτητο να μελετήσουμε πάνω σε αυτό προκειμένου να φτιάξουμε μια ερασιτεχνικού επιπέδου σειρά από animations.

Το 3ο κομμάτι που αντιμετωπίσαμε αρκετές δυσκολίες ήταν τα λογικά λάθη που προέκυπταν στην κατασκευή της τεχνητής νοημοσύνης των εχθρών και συγκεκριμένα στην

1η πίστα. Η δυναμικότητα που προσπαθήσαμε να επιτύχουμε δυσκόλεψε την κατασκευή της διότι αποσκοπούσαμε σε ένα σύστημα αυτόματης διαχείρισης μέσω ιεραρχίας.

4.3. Μελλοντική εξέλιξη και επεκτάσεις στον χώρο των videogames

Η συγκεκριμένη εργασία έχει προοπτικές να γίνει ακόμα μεγαλύτερη και με προσθήκη σε πολλά κομμάτια όπως καινούργια όπλα, περισσότερους παίκτες, περισσότερες πίστες αρκετά πιο εμπλουτισμένες και πιθανότατα αφήνει την δυνατότητα στον κάθε game developer να αναπτύξει το δικό του κομμάτι μέσα σε αυτό καθώς περιέχει δυναμικότητα στον κώδικα. Το είδος του παιχνιδιού περιορίζεται στο είδος 3rd Person Shooter καθώς έχει σχεδιαστεί να έχει όπλα μεγάλου βεληνεκούς, όμως η θεματολογία μπορεί να αλλάξει εύκολα σε ότι επιθυμούμε εμείς (Western, Cyberpunk, Scifi). Πιθανή είναι επίσης η εξέλιξη του και σε παιχνίδι Multiplayer καθώς η ομάδα των παικτών μπορεί να αλλαχτεί σε multiplayer cooperation αντί για single player cooperation όπως είναι σχεδιασμένο αυτή την στιγμή.

5. Παράρτημα

1. MakeHuman

1.1 Γενικά

Το MakeHuman είναι ένα πρόγραμμα δημιουργίας βασικών ανθρωποειδών μοντέλων με δυνατότητα προσθήκης ρουχισμού, μαλλιών και άλλες διάφορες γεωμετρίες επέκτασης του μοντέλου. Μέσω της επίσημης ιστοσελίδας μπορούμε να κατεβάσουμε αρχεία με προσθήκες που έχουν φτιάξει άλλοι χρήστες και να το εφαρμόσουμε στο πρόγραμμα πάνω για να εμπλουτίσουμε την συλλογή με accessories. Το MakeHuman μέσω του plug-in που έχει το Blender μας δίνει την δυνατότητα να δημιουργήσουμε τα δικά μας ρούχα που θέλουμε να βάλουμε στους χαρακτήρες μας. Το plug-in αυτό ονομάζεται "MakeClothes" και παρέχεται στο βασικό πακέτο του Blender.Το μεγάλο πρόβλημα που λύνει το MakeHuman είναι η δημιουργία Rig, καθώς εξάγει έτοιμα μοντέλα με εφαρμοσμένη σκελετική ιεραρχία. Η τελευταία έκδοση του είναι "MakeHuman 1.1.0" .

2. Blender

2.1 Γενικά

Το Blender αποτελεί ένα πλέον μεγάλο πρόγραμμα, καθώς δεν είναι μόνο σχεδιαστικό εργαλείο για γραφικά και animation. Περιέχει και μέρος που ανταποκρίνεται ως game engine προσφέροντας αρκετές από τις βασικές λειτουργίες που προσφέρουν γενικά οι game engines. Σαν εργαλείο σχεδιασμού γραφικών, υφών είναι ιδανικό για αρχάριους αλλά και για επαγγελματίες μιας και προσφέρει μια πληθώρα επιλογών και ανταποκρίνεται σε αρκετές απαιτήσεις κάτι που το καθιστά ανταγωνίσιμο ανάμεσα στα υπόλοιπα σχεδιαστικά προγράμματα, καθώς επίσης συνεργάζεται άριστα με όλες τις γνωστές game engines όπως Unity, Unreal4 κλπ. Η τελευταία έκδοση του Blender είναι η 2.78a .

3. Unity

3.1 Γενικά

Η Unity είναι μια game engine που προσεγγίζει στο πακέτο της τόσο τις εφαρμογές 3D αλλά και 2D. Περιέχει πάρα πολλές δυνατότητες στους developers και είναι αρκετά απλή στην χρήση της, καθώς μέσω της επίσημης ιστοσελίδας προσφέρει εκπαιδευτικά αρχεία για την μάθηση της με ολοκληρωμένα παραδείγματα . Καλύπτει όλες τις πτυχές σχεδιασμού ενός videogame τόσο σε αρχικό επίπεδο όσο και στο επαγγελματικό . Η έκδοση εφαρμογών είναι δωρεάν και δεν απαιτεί ειδική άδεια παρά μόνο στην περίπτωση που τα κέρδη της εφαρμογής ξεπεράσουν ένα μεγάλο ποσό. Οι πλατφόρμες δράσης που επιτρέπει για σχεδιασμό εφαρμογών είναι πολλές και συνεχώς αυξάνονται. Υπάρχουν ήδη αρκετά εκατομμύρια εφαρμογές δημοσιευμένες. Οι γλώσσες προγραμματισμού που πλέον υποστηρίζονται είναι C# και Javascript. Η τελευταία έκδοση της Unity είναι η 5.5.0.

Βιβλιογραφία

https://docs.unity3d.com/ScriptReference/

http://answers.unity3d.com/questions/604442/need-help-with-shooting-script-c.html

http://www.makehuman.org/

https://www.youtube.com/watch?v=f6s9RO8pZZc

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

https://www.youtube.com/watch?v=PyEmRVRHWL8

https://www.youtube.com/watch?v=FD9HZB0Jn1w

https://www.youtube.com/watch?v=6agwCUaMNWI

http://gamedev.stackexchange.com/questions/104693/how-to-use-input-getaxismouse-x-y-to-rotate-the-camera

https://www.youtube.com/watch?v=h8oI0n5kAIg

https://www.youtube.com/watch?v=gvDse2H3B7s

https://unity3d.com/learn/tutorials/topics/physics/detecting-collisions-oncollisionenter

https://www.youtube.com/watch?v=rRak-Hn7rgM

https://www.youtube.com/watch?v=i7a_wlAKB7I

https://www.youtube.com/watch?v=4oIoh4x7qH0

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

https://unity3d.com/learn/tutorials/topics/user-interface-ui

https://unity3d.com/learn/tutorials/topics/graphics

https://docs.unity3d.com/Manual/class-AnimationClip.html

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

https://docs.unity3d.com/Manual/class-AnimatorController.html

https://docs.unity3d.com/Manual/class-NavMeshAgent.html

https://docs.unity3d.com/ScriptReference/CharacterController.html

https://docs.unity3d.com/ScriptReference/Light.html

https://docs.unity3d.com/ScriptReference/Material.html

https://docs.unity3d.com/Manual/class-LineRenderer.html

https://docs.unity3d.com/ScriptReference/ParticleSystem.html

https://docs.unity3d.com/Manual/class-TrailRenderer.html

https://docs.unity3d.com/Manual/class-Skybox.html

https://forum.unity3d.com/threads/18-free-skyboxes-unitypackage.27513/

Προγράμματα που χρησιμοποιήθηκαν

MakeHuman v1.0.2

Blender v2.74

Unity Game Engine v5.0.0f4