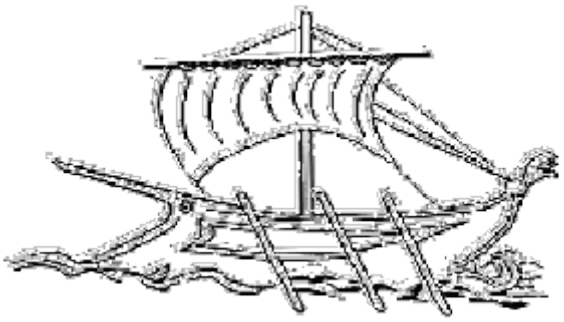



# ΑΕΙ ΠΕΙΡΑΙΑ Τ.Τ.
## ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
### ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ Τ.Ε.


**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Ανάπτυξη εφαρμογής ηλεκτρονικού πολεμικού παιχνιδιού.


**Μελέτιος Κατρακούλης**
**Γεώργιος Μανίκας**

**Εισηγητής: Ματιάτος Σπυρίδων , Καθηγητής Εφαρμογών**



**ΑΘΗΝΑ**
**ΜΑΙΟΣ 2016**

Ανάπτυξη εφαρμογής ηλεκτρονικού πολεμικού παιχνιδιού.



Μελέτιος Γ. Κατρακούλης & Γεώργιος Ε. Μανίκας

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# Συσκευή αυτόματης παροχής τροφής για κατοικίδια ζώα.

**Μελέτιος Γ. Κατρακούλης**
**Α.Μ. 41788**

**Γεώργιος Ε. Μανίκας**
**Α.Μ. 42073**

**Εισηγητής:**

**Ματιάτος Σπυρίδων , Καθηγητής Εφαρμογών**

**Εξεταστική Επιτροπή:**

**Ημερομηνία εξέτασης**

# ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Οι κάτωθι υπογεγραμμένοι Μελέτιος Κατρακούλης, του Γεωργίου, με αριθμό μητρώου 41788 και Γεώργιος Μανίκας, του Ευαγγέλου, με αριθμό μητρώου 42073 φοιτητές του Τμήματος Μηχανικών Η/Υ Συστημάτων Τ.Ε. του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβουμε την εκπόνηση της Πτυχιακής Εργασίας μας, δηλώνουμε ότι ενημερωθήκαμε για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

# ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία ολοκληρώθηκε έπειτα από αρκετές δοκιμές, αλλαγές και συζητήσεις για το καλύτερο δυνατό αποτέλεσμα. Σε αυτό το σημείο, μιας και όλα κύλησαν ομαλά θα θέλαμε να ευχαριστήσουμε αρχικά τον καθηγητή μας, κ. Σπυρίδωνα Ματιάτο, για την πολύτιμη βοήθεια και στήριξη που μας παρείχε όποτε του ζητήθηκε. Επίσης θα θέλαμε να ευχαριστήσουμε τις οικογένειες μας για την υποστήριξη τους καθ'όλη την διάρκεια της εκπόνησης της πτυχιακής εργασίας.

## ΠΕΡΙΛΗΨΗ

Η Πτυχιακή έχει ως θέμα την δημιουργία ενός FPS(First – Person Shooting) παιχνιδιού σε περιβάλλον Unity και σε γλώσσα C# με τη χρήση των σχεδιαστικών προγραμμάτων Unity (5.1.2f1 free) και Blender (2.75). Τα πρόγραμμα δεν είναι αγορασμένα με αποτέλεσμα να υπάρχουν κάποιοι περιορισμοί στις δυνατότητες ανάπτυξης. Η C# ως αντικειμενοστραφής γλώσσα προγραμματισμού είναι κατάλληλη για τον σχεδιασμό τέτοιου είδους παιχνιδιών. Τα δύο σχεδιαστικά προγράμματα επιλέχθηκαν ως προσιτά προς τους χρήστες και εύκολα στην χρήση τους. Σκοπός του παιχνιδιού, μετά την επιλογή ομάδας ή τυχαίο ορισμό της ομάδας, είναι η εξόντωση της αντίπαλης ομάδας με την χρήση των οπλών των παικτών. Τα στάδια υλοποίησης της πτυχιακής περιλαμβάνουν: (α) Σχεδιασμό της πίστας στο Blender, (β) Σχεδιασμό γραφικού περιβάλλοντος στο Unity, (γ) Εγκατάσταση του Server (διακομιστής) για την συνδεσιμότητα των χρηστών, (δ) Συγχρονισμό των χρηστών με τον Server (διακομιστής), (ε) Εγκατάσταση γραφικών των παικτών και (στ) Προγραμματισμός χειρισμού του παίκτη.

## ABSTRACT

The subject for our thesis was the development of an FPS(First-Person Shooting) game, created in the environment of Unity and the programming language was C#. The programs that were used for this development were Unity(5.1.2f1) and Blender(2.75). Those programs were not purchased and that as a result was the limitation to the growth potential of the application. C# as an oriented programming language it's the best choice and best suited for the design of a development for a game like this. Unity and Blender were the two programs preferred for this development because of their good accessibility and easiness on use. The purpose of the game, after the selection or random definition of the teams, is the extermination of all the opponents with the use of each player’s weapons. The stages of developing the game are:(a) Design the map using Blender, (b) Design the GUI using Unity ,(c)Install a server (διακομιστής) for user connectivity, (d) Synchronizing the users with the server (διακομιστής), (e) Installing graphics of players, (f) Develop the player handling programming.

Ανάπτυξη εφαρμογής ηλεκτρονικού πολεμικού παιχνιδιού.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Προγραμματισμός C#.
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Unity, Blender, FPS, διαδικτυακό ηλεκτρονικό παιχνίδι.

# ΠΕΡΙΕΧΟΜΕΝΑ

# ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

# ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

# ΚΕΦΑΛΑΙΟ 1

# ΕΙΣΑΓΩΓΗ

## 1.1 Συνοπτική Εισαγωγή

Η παρούσα πτυχιακή εργασία έχει ως θέμα την ανάπτυξη εφαρμογής ηλεκτρονικού πολεμικού παιχνιδιού με στόχο την ψυχαγωγία του χρήστη. Η εφαρμογή αυτή απευθύνεται σε ανθρώπους που ασχολούνται με διαδικτυακά ηλεκτρονικά παιχνίδια και κυρίως με παιχνίδια τύπου First-Person Shooter(FPS). Τα παιχνίδια τύπου FPS είναι παιχνίδια μάχης πολεμικού περιεχομένου όπου οι χρήστες λαμβάνουν μέρος στην δράση μέσα από τα μάτια του πρωταγωνιστή. Με άλλα λόγια τα παιχνίδια τέτοιου τύπου δίνουν την προοπτική της μάχης σε πρώτο πρόσωπο. Ο χρήστης χρησιμοποιώντας την εφαρμογή αυτή γεμίζει τον ελεύθερο χρόνο του διασκεδάζοντας με ένα ομαδικό παιχνίδι το οποίο προσφέρει και εκτόνωση αφού η μία ομάδα προσπαθεί να ‘εξοντώσει’ την αντίπαλή της. Τέλος η εφαρμογή αυτή οξύνει τα αντανακλαστικά του παίκτη και του μαθαίνει την επίτευξη ενός στόχου μέσω της ομαδικής δουλειάς.

## 1.2 Ιστορική Αναδρομή

Τα ηλεκτρονικά παιχνίδια εμφανίστηκαν στο τέλος της δεκαετίας του 1940, όταν πανεπιστημιακοί ξεκίνησαν να σχεδιάζουν απλής μορφής παιχνίδια και προσομοιωτές, ως τμήματα υπολογιστικών επιστημονικών ερευνών. Μέχρι την δεκαετία του 1970 ο όρος βιντεοπαιχνίδια δεν ήταν δημοφιλής. Με το πέρασμα των χρόνων όμως, εξελίχθηκε από έναν απλό τεχνικό όρο σε μια γενική έννοια, η οποία αναφέρεται σε μια καινοτομία, σε ένα νέο είδος διαδραστικής ψυχαγωγίας. Στις μέρες μας ένα βιντεοπαιχνίδι ορίζεται ως οποιοδήποτε παιχνίδι που μπορεί να αναπαραχθεί σε ένα λειτουργικό σύστημα που αποτελείται από ηλεκτρονικά λογικά κυκλώματα και εμπεριέχει το στοιχείο της διαδραστικότητας, αποδίδοντας τα αποτελέσματα των δράσεων των παικτών σε μια οθόνη.

Επί 1950, τα βιντεοπαιχνίδια ταξινομούνταν σε τρεις κατηγορίες.

- Προγράμματα εκμάθησης και προπόνησης

- Ερευνητικά προγράμματα στον τομέα της τεχνητής νοημοσύνης
- Προγράμματα επίδειξης που είχαν σκοπό να εντυπωσιάσουν και να ψυχαγωγήσουν το κοινό-χρήστες.

Το πρώτο παιχνίδι που αναπτύχθηκε για τον ηλεκτρονικό υπολογιστή, ήταν μία προσομοίωση σκακιού από τον Alan Turing και τον David Champernowne. Ονομάστηκε Turochamp, όμως δεν εφαρμόστηκε ποτέ σε κάποιον υπολογιστή. Έγιναν πολλές απόπειρες σχεδιασμών παιχνιδιών, αλλά οι περισσότερες ήταν για την ανάδειξη συγκεκριμένων τεχνολογιών ή για την ολοκλήρωση ερευνών. Το πρώτο παιχνίδι που σχεδιάστηκε καθαρά για ψυχαγωγικούς λόγους ήταν το “Tennis for Two”, που σχεδιάστηκε από τον William Hginbotham και υλοποιήθηκε από τον Robert Dvorak το 1958.

Με το πέρασμα των χρόνων, άρχισαν να δημιουργούνται κονσόλες σχεδιασμένες να αναπαράγουν ένα συγκεκριμένο παιχνίδι με αρκετά περιορισμένες δυνατότητες. Χαρακτηριστικό παράδειγμα στην εποχή 1970-1980 ήταν τα arcade games, κονσόλες που λειτουργούσαν με την εισαγωγή κέρματος. Με την εξέλιξη της τεχνολογίας, νέες εταιρείες έκαναν την εμφάνισή τους. Σχεδιάζονταν νέες, εκσυγχρονισμένες κονσόλες με δυνατότητα αναπαραγωγής περισσότερων από ένα παιχνίδι με όλο και καλύτερα, βελτιωμένα γραφικά.

Στα μέσα της δεκαετίας του 1970, έκαναν την εμφάνισή τους οι προσωπικοί υπολογιστές, αντικαθιστώντας τις μεμονωμένες κονσόλες, καθώς έδιναν τη δυνατότητα στους χρήστες να προγραμματίζουν και να παίζουν παιχνίδια απλής μορφής. Τα προγράμματα αυτά διανέμονταν έτοιμα ως κείμενο μέσω βιβλίων και περιοδικών, καθώς και σε μορφή δισκέτας ή κασέτας στο κοινό προκειμένου να μπορεί να παίξει ο καθένας το παιχνίδι στον δικό του, προσωπικό Η/Υ. Στα τέλη της ίδιας δεκαετίας και στις αρχές της επόμενης, με την εισχώρηση του διαδικτύου στην καθημερινή ζωή, έκαναν την εμφάνισή τους συστήματα τα οποία παρείχαν τη δυνατότητα στον χρήστη να παίζει τα παιχνίδια αυτά διαδικτυακά, ταυτόχρονα με άλλους, σε κοινό γραφικό περιβάλλον. Αυτά είναι τα σημερινά Online Games.

Στη συνέχεια, άρχισαν να κυκλοφορούν κονσόλες βιντεοπαιχνιδιών όλο και πιο εξελιγμένες. Αυτό σήμαινε πως η αναπαραγωγή τους χαρακτηριζόταν από βελτιωμένο ήχο, καλύτερης ποιότητας γραφικά και εικόνα και γενικότερα η μορφή τους ήταν πιο απλουστευμένη με μεγάλη ευχρηστία. Παράλληλα, στους Η/Υ

γινόταν ακριβώς το ίδιο, με την κατασκευή καλύτερων επεξεργαστών, καρτών ήχου και γραφικών όπως και άλλων λειτουργικών συστημάτων που συνέβαλαν στη βέλτιστη αναπαραγωγή βιντεοπαιχνιδιών, φτάνοντας μέχρι και στις μέρες μας, χαρίζοντας όλο και περισσότερες εμπειρίες και ώρες ψυχαγωγίας στους χρήστες. Με τους ρυθμούς και τις ταχύτητες εξέλιξης των Η/Υ και κατ' επέκταση των βιντεοπαιχνιδιών, υπολογίζεται πως και στα επόμενα χρόνια η ανάπτυξή τους θα είναι ραγδαία.

Στις μέρες μας έχουν δημιουργηθεί διάφορες κατηγορίες παιχνιδιών. Κάθε μία από αυτές τις κατηγορίες έχει τα δικά της χαρακτηριστικά. Λόγω του μεγάλου αριθμού βιντεοπαιχνιδιών κάθε κατηγορία έχει τις δικές της υποκατηγορίες. Μερικές από αυτές αναφέρονται παρακάτω.[5]

### 1.2.1 Παιχνίδια Δράσης (Action Games)

Τα action games είναι παιχνίδια δράσης και απαιτούν τον συντονισμό των ματιών, των χεριών και του μυαλού του χρήστη ο οποίος χειρίζεται έναν χαρακτήρα και αντιμετωπίζει διάφορες δοκιμασίες. [1] [2] Οι υποκατηγορίες είναι:

- Παιχνίδια πλατφόρμας (Platform games)

  Τα παιχνίδια πλατφόρμας εκτυλίσσονται σε ένα τρισδιάστατο περιβάλλον. Οι παίκτες καλούνται να καθοδηγήσουν έναν χαρακτήρα μέσω εμποδίων, πηδώντας πάνω σε πλατφόρμες και αντιμετωπίζοντας αντιπάλους.
- Παιχνίδια σκοποβολής (Shooter games)

  Στα παιχνίδια σκοποβολής οι παίκτες χρησιμοποιούν όπλα από απόσταση για να λάβουν μέρος στην δράση. Τα περισσότερα παιχνίδια της κατηγορίας αυτής είναι πολύ βίαια, υπάρχουν όμως και κάποια πιο ήπια.
- Παιχνίδια μάχης (Fighting games and beat 'em ups)

  Τα παιχνίδια αυτής της υποκατηγορίας εξομοιώνουν μάχες κοντινής αποστάσεως εναντίον μερικών αντιπάλων με την χρήση βίαιων επιθέσεων. Συχνά περιλαμβάνουν την χρήση όπλων αλλά δίνουν έμφαση στην μάχη σώμα με σώμα.

### 1.2.2 Παιχνίδια περιπέτειας (Adventure Games)

Πρόκειται για παιχνίδια περιπέτειας η οποία λαμβάνει χώρα σε κάποιο γραφικό περιβάλλον. Τα παιχνίδια αυτά δεν απαιτούν τόσο τα αντανακλαστικά του παίκτη, όπως η προηγούμενη κατηγορία, αλλά απαιτούν από τον χρήστη να λύνει γρίφους και να δίνει ροή στην περιπέτεια ερχόμενος σε επαφή τόσο με άλλους χαρακτήρες όσο και με το γραφικό περιβάλλον στο οποίο βρίσκεται.[1] Οι υποκατηγορίες είναι:

- Παιχνίδια που δεν γίνεται αντιληπτός ο παίκτης (Stealth game)

  Τα παιχνίδια αυτά εστιάζουν στα τεχνάσματα και στα χτυπήματα ακρίβειας για την εξόντωση του αντιπάλου χωρίς όμως ο αντίπαλος να αντιλαμβάνεται την παρουσία του παίκτη.
- Παιχνίδια επιβίωσης (Survival honor)

  Τα παιχνίδια επιβίωσης εστιάζουν στην προσπάθεια να φοβίσουν τον παίκτη με τη χρήση παραδοσιακών φανταστικών στοιχείων τρόμου. Σε πολλά από αυτά τα παιχνίδια τα αντικείμενα που μπορεί να χρησιμοποιήσει ο παίκτης είναι λίγα πράγμα που τον αναγκάζει να επιβιώσει με όποιο τρόπο μπορεί.
- Metroidvania

  Στα παιχνίδια αυτά ο παίκτης έχει την δυνατότητα να εξερευνήσει έναν τεράστιο κόσμο. Υπάρχουν όμως μέρη σε αυτόν τον κόσμο όπου έχει πρόσβαση μόνο αφού έχει αποκτήσει κάποια εργαλεία, όπλα ή κάποιες ικανότητες στη διάρκεια του παιχνιδιού.

### 1.2.3 Παιχνίδια Δράσης-Περιπέτειας (Action-Adventure Games)

Η κατηγορία αυτή αποτελεί συνδυασμό των δύο προηγούμενων κατηγοριών. Τα παιχνίδια αυτής της κατηγορίας εστιάζουν στην εξερεύνηση κάποιου γραφικού περιβάλλοντος καθώς επίσης και την συλλογή αντικειμένων που χρησιμεύουν αργότερα στο παιχνίδι για την επίλυση γρίφων η για την μάχη.[1] Οι υποκατηγορίες είναι:

- Παιχνίδια δράσης-περιπέτειας με τη χρήση εντολών (Text adventures)

  Σε αυτή την υποκατηγορία παιχνιδιών ανήκουν παιχνίδια περιπέτειας στα οποία ο παίκτης δίνει εντολή μέσω του πληκτρολογίου στον χαρακτήρα του, ο οποίος αντιδράει αναλόγως.
- Παιχνίδια δράσης-περιπέτειας με τη χρήση του γραφικού περιβάλλοντος (Graphic adventures)

  Τα παιχνίδια αυτά αναπτύχθηκαν όσο τα γραφικά άρχιζαν να γίνονται πιο συνηθισμένα. Με το πέρασμα των χρόνων και με την ανάπτυξη των γραφικών, ήταν πιο εύκολο για τον παίκτη να χρησιμοποιεί τον κέρσορα αντί να δίνει εντολές μέσω του πληκτρολογίου
- Εικονικό μυθιστόρημα (Visual novels)

  Τα παιχνίδια αυτά παρουσιάζουν την ιστορία τους με στατικά γραφικά, συνήθως με την χρήση σχεδίων κινούμενων σχεδίων. Ο παίκτης για να αναπτύξει την πλοκή του παιχνιδιού πρέπει να ανεβάζει συνεχώς τα στατιστικά του χαρακτήρα του.
- Διαδραστικές ταινίες (Interactive movie)

  Οι διαδραστικές ταινίες εμπεριέχουν προσχεδιασμένες σκηνές στις οποίες ο παίκτης καλείται να επιλέξει κάποιες από τις δράσεις του χαρακτήρα του. Σε αυτά τα παιχνίδια, δηλαδή, ο παίκτης πρέπει να επιλέξει ή να μαντέψει τις δράσεις που οι σχεδιαστές έχουν ήδη προεπιλέξει.
- Τρισδιάστατες περιπέτειες (Real-time 3D adventures)

  Πρόκειται για τρισδιάστατα παιχνίδια δράσης-περιπέτειας στα οποία ο παίκτης έχει ελευθερία κινήσεων και δράσεων.

### 1.2.4 Παιχνίδια Επιλογής Ρόλου (Role-Playing Games)

Τα παιχνίδια αυτής της κατηγορίας παρουσιάζουν στον χρήστη έναν φανταστικό κόσμο συνήθως γεμάτο με πόλεις, κάστρα, τέρατα και διάφορες περιοχές, τον οποίο μπορεί να εξερευνήσει με τον χαρακτήρα του ακολουθώντας μία συγκεκριμένη σειρά γεγονότων. Ο χρήστης έχει την επιλογή να δημιουργήσει τον χαρακτήρα του δίνοντάς του συγκεκριμένη μορφή και ένα σύνολο ικανοτήτων που θα τον βοηθήσουν.[1] Οι υποκατηγορίες είναι:

- Παιχνίδια δράσης επιλογής ρόλων (Actions RPGs)

  Τα παιχνίδια αυτής της υποκατηγορίας είναι παιχνίδια που συνδυάζουν την δημιουργία χαρακτήρα με επιλογές του παίκτη και στοιχεία παιχνιδιών δράσης.
- Διαδικτυακά παιχνίδια επιλογής ρόλων (MMORPGs)

  Σε αυτή την υποκατηγορία ανήκουν τα παιχνίδια επιλογής ρόλων που παίζονται μαζικά διαδικτυακά από πολλούς παίκτες. Σε αυτά τα παιχνίδια οι παίκτες αναβαθμίζουν τα στατιστικά του χαρακτήρα τους μέσω αναζητήσεων αντικειμένων η μέσω μάχης με άλλους παίκτες.
- Roguelikes

  Το είδος αυτού του παιχνιδιού δίνει έμφαση στην ανάπτυξη των στατιστικών του χαρακτήρα σε ένα δισδιάστατο περιβάλλον.
- Παιχνίδια τακτικής επιλογής ρόλων (Tactical RPGs)

  Η υποκατηγορία αυτή αναφέρεται σε παιχνίδια που συνδυάζουν παιχνίδια στρατηγικής με παιχνίδια επιλογής ρόλων. Ο παίκτης ελέγχει μια ομάδα με περιορισμένο αριθμό χαρακτήρων αντιμετωπίζοντας τους αντιπάλους του που είναι ομάδες με παρόμοια χαρακτηριστικά. Κάθε παίκτης έχει συγκεκριμένο χρόνο για να σκεφτεί και να κάνει την στρατηγική ανάπτυξη της ομάδας του.
- Παιχνίδια επιλογής ρόλου ανοικτού κόσμου (Sandbox RPGs)

  Στην υποκατηγορία αυτή τα παιχνίδια ανοικτού κόσμου, τα παιχνίδια δηλαδή που ο χαρακτήρας δεν είναι περιορισμένος όσον αφορά τα μέρη που μπορεί να εξερευνήσει και δεν είναι απαραίτητο να ακολουθεί ένα συγκεκριμένο μονοπάτι. Τα παιχνίδια αυτά συνήθως περιέχουν έναν μεγάλο αριθμό χαρακτήρων τεχνητής νοημοσύνης με τους οποίους ο

παίκτης μπορεί να έρθει σε επαφή και να βοηθηθεί στην εξέλιξη του παιχνιδιού.

- Παιχνίδια διαφορετικής κουλτούρας (Cultural differences)

  Τα παιχνίδια που ανήκουν σε αυτή την υποκατηγορία χωρίζονται σε δύο κατηγορίες. Στην μία ο παίκτης φτιάχνει τον δικό του χαρακτήρα και είναι σχετικά ελεύθερος στις επιλογές και στις δράσεις του ενώ στην άλλη ο παίκτης ελέγχει μια ομάδα προκατασκευασμένων χαρακτήρων και η ιστορία που εκτυλίσσεται έχει ένα προκαθορισμένο σενάριο.
- Παιχνίδια επιλογών (Choices)

  Σε αυτή την υποκατηγορία ανήκουν τα παιχνίδια στα οποία ο παίκτης έχει την επιλογή του αν θα νικήσει τους αντιπάλους του με θανατηφόρα αποτελέσματα ή όχι.
- Παιχνίδια φαντασίας (Fantasy)

  Τα παιχνίδια αυτής της υποκατηγορίας, όπως λέει και ο τίτλος, είναι παιχνίδια φαντασίας τα οποία λαμβάνουν χώρα φανταστικούς κόσμους συνήθως μεσαιωνικού τύπου.

### 1.2.5 Παιχνίδια Προσομοίωσης (Simulation Games)

Πρόκειται για μια μεγάλη κατηγορία παιχνιδιών τα οποία είναι σχεδιασμένα να προσομοιώνουν πιστά πτυχές της πραγματικότητας ή ακόμα και υποθετικών καταστάσεων.[1] Οι υποκατηγορίες είναι:

- Προσομοίωση κατασκευής και διαχείρισης (Construction and management simulation)

  Τα παιχνίδια της υποκατηγορίας αυτής είναι παιχνίδια προσομοίωσης τα οποία ζητούν από τους παίκτες να κατασκευάσουν, να επεκτείνουν και να διοικήσουν φανταστικές κοινότητες ή προγράμματα με συγκεκριμένο αριθμό πόρων.

- Προσομοίωση ζωής (Life simulation)

Σε αυτή την υποκατηγορία ανήκουν παιχνίδια που προσομοιώνουν την ζωή. Ο παίκτης ελέγχει έναν χαρακτήρα τεχνητής νοημοσύνης ο οποίος αναπτύσσει σχέσεις με άλλους χαρακτήρες τεχνητής νοημοσύνης.

- Προσομοίωση οχημάτων (Vehicle simulation)

  Πρόκειται για παιχνίδια προσομοίωσης τα οποία εξομοιώνουν τον τρόπο λειτουργίας διάφορων οχημάτων με ρεαλιστικό τρόπο.

### 1.2.6 Παιχνίδια Στρατηγικής (Strategy Games)

Σε αυτή την κατηγορία ανήκουν τα παιχνίδια που θέλουν προσεκτική στρατηγική και μεθοδική σκέψη με σκοπό τη νίκη. Συνήθως ο κάθε παίκτης περιμένει την σειρά του για να παίξει και έχει στον έλεγχο του έναν ολόκληρο στρατό ή γενικότερα μια ομάδα χαρακτήρων που έχουν σκοπό να εξαλείψουν τον αντίπαλο.[1] Οι υποκατηγορίες είναι:

- Παιχνίδια 4E (4X game)

  Τα παιχνίδια αυτής της κατηγορίας έχουν αυτό το όνομα διότι η στρατηγική τους βασίζεται σε 4 στόχους: εξερεύνηση, επέκταση, εκμετάλλευση, εξολόθρευση.
- Παιχνίδια πυροβολικού (Artillery game)

  Σε αυτή την κατηγορία ανήκουν τα παιχνίδια στρατηγικής όπου οι παίκτες χρησιμοποιούν πυρομαχικά για να νικήσουν τον αντίπαλο, είτε με μορφή όπλων είτε με μορφή πολεμικών οχημάτων.
- Παιχνίδια στρατηγικής πραγματικού χρόνου (Real-time strategy(RTS))

  Τα παιχνίδια που ανήκουν σε αυτή την υποκατηγορία είναι παιχνίδια στρατηγικής στα οποία οι παίκτες πρέπει να αναπτύσσουν τις στρατηγικές τους ενώ η κατάσταση του παιχνιδιού αλλάζει συνεχώς.
- Διαδικτυακά παιχνίδια στρατηγικής πραγματικού χρόνου (MMORTS)

  Πρόκειται για παιχνίδια στρατηγικής που συνδυάζουν στρατηγική πραγματικού χρόνου με μία πίστα συγκεκριμένου γεωγραφικού πλάτους όπου οι παίκτες έχουν τον δικό τους στρατό και αναπτύσσουν την στρατηγική τους.

- Παιχνίδια τακτικής πραγματικού χρόνου (Real-time tactics)

  Η διαφορά των παιχνιδιών αυτής της υποκατηγορίας και των παιχνιδιών στρατηγικής πραγματικού χρόνου είναι ότι οι παίκτες δεν χρειάζεται να ασχολούνται με την κατασκευή κτηρίων και την διοίκηση της οικονομίας τους. Εστιάζουν μόνο στην ανάπτυξη της στρατηγικής τους.
- Παιχνίδια υπεράσπισης με τη χρήση πύργων (Tower defense)

  Στα παιχνίδια αυτής της υποκατηγορίας υπάρχουν χαρακτήρες τεχνητής νοημοσύνης συνήθως στη μορφή τεράτων τα οποία ακολουθούν ένα συγκεκριμένο μονοπάτι και ο παίκτης καλείται να κτίσει πύργους σε αυτό το μονοπάτι με τα οποία μπορεί να εξολοθρεύσει αυτά τα τέρατα.
- Παιχνίδια στρατηγικής εναλλασσόμενης σειράς (Turn-based strategy)

  Στα παιχνίδια αυτής της υποκατηγορίας οι παίκτες έχουν συγκεκριμένο χρόνο για να αναλύσουν την κατάσταση στην οποία βρίσκονται πριν προβούν σε κάποια πράξη και έχουν περιορισμένες κινήσεις σε σχέση με την πίστα που βρίσκονται.
- Παιχνίδια τακτικής εναλλασσόμενης σειράς (Turn-based tactics)

  Σε αυτή την υποκατηγορία ανήκουν τα παιχνίδια στρατηγικής στα οποία οι παίκτες καλούνται να ολοκληρώσουν τα ζητούμενα του παιχνιδιού με τη χρήση των ένοπλων δυνάμεων που τους παρέχει το παιχνίδι.
- Παιχνίδια πολέμου (War game)

  Τα παιχνίδια που ανήκουν σε αυτή την υποκατηγορία, όπως λέει και ο τίτλος, είναι παιχνίδια στρατηγικής που στηρίζονται στην ανάπτυξη πολεμικών στρατηγικών σε ένα συγκεκριμένο γεωγραφικό εύρος.

### 1.2.7 Αθλητικά Παιχνίδια (Sports Games)

Τα παιχνίδια αυτής της κατηγορίας προσομοιώνουν πραγματικά αλλά και φανταστικά αθλήματα. Ο χρήστης έχει την δυνατότητα να παίξει και εναντίον άλλων χρηστών αλλά και εναντίον του υπολογιστή, δηλαδή εναντίον τεχνητής νοημοσύνης.[1] Οι υποκατηγορίες είναι:

- Παιχνίδια αγώνα δρόμου (Racing)

  Σε αυτή την υποκατηγορία ανήκουν παιχνίδια στα οποία ο παίκτης χρησιμοποιεί κάποιο μέσο μεταφοράς για να ανταγωνιστεί με τους αντιπάλους του.
- Αθλητικά παιχνίδια (Sports game)

  Όπως αναφέρει και ο τίτλος, τα παιχνίδια αυτής της κατηγορίας εξομοιώνουν τα παραδοσιακά αθλήματα.
- Ανταγωνιστικά παιχνίδια (Competitive)

  Πρόκειται για παιχνίδια με υψηλό βαθμό ανταγωνισμού των οποίων το περιεχόμενο είναι αθλητικά παιχνίδια φαντασίας και όχι των παραδοσιακών αθλημάτων που γνωρίζουμε.
- Παιχνίδια αθλητικών πολεμικών τεχνών (Sports-based fighting)

  Τα παιχνίδια αυτής της κατηγορίας είναι παιχνίδια που βασίζονται στην μάχη σώμα με σώμα.

Η εφαρμογή της πτυχιακής εργασίας ανήκει στην πρώτη κατηγορία των Action Games και συγκεκριμένα στην υποκατηγορία First-Person Shooter Games (FPS). Πρόκειται για παιχνίδια στα οποία ο χρήστης συμμετέχει στο πεδίο μάχης σε πρώτο πρόσωπο, δηλαδή από την οπτική του πρωταγωνιστή. Χρησιμοποιεί όπλα, είτε μακρινής είτε κοντινής απόστασης, με σκοπό την εξάλειψη των πόντων ζωής του αντιπάλου. Τα παιχνίδια της κατηγορίας αυτής απαιτούν την συνεχή προσοχή του χρήστη και την γρήγορη αντίδραση του.

# ΚΕΦΑΛΑΙΟ 2

## ΠΕΡΙΒΑΛΛΟΝ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΗΣ

Τα προγράμματα που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής ήταν το σχεδιαστικό πρόγραμμα 3D γραφικών Blender και το πρόγραμμα cross-platform παιχνιδιών Unity. Το Blender επιλέχθηκε με σκοπό τον σχεδιασμό όλων των 3D αντικειμένων που χρησιμοποιούνται στην εφαρμογή μας. Δηλαδή τον σχεδιασμό της πίστας, όλων των αντικειμένων που βρίσκονται πάνω της καθώς και του όπλου που χρησιμοποιούν οι παίκτες. Επιλέξαμε αυτό το πρόγραμμα καθώς ξέραμε κάποιους από τους χειρισμούς του, αφού έχουμε ξαναχρησιμοποιήσει το ίδιο πρόγραμμα στο παρελθόν για απλούστερους σχεδιασμούς. Το Unity επιλέχθηκε με σκοπό την δημιουργία και την πλήρη επεξεργασία του περιβάλλοντος της εφαρμογής μας. Δηλαδή των animation (κινούμενων σχεδίων), των ήχων, των γραφικών, του χειρισμό και όλων των χαρακτηριστικών που αποτελούν την εφαρμογή αυτή. Επιλέξαμε αυτό το πρόγραμμα διότι είναι από τα πιο ολοκληρωμένα ‘εργαλεία’ για την δημιουργία βιντεοπαιχνιδιών. Επίσης, είναι εύχρηστο και εύκολο στην εκμάθηση του καθώς υπάρχουν βοηθητικά αρχεία για την χρήση του στο διαδίκτυο. Ένας ακόμα παράγοντας που μας οδήγησε στην επιλογή των δύο αυτών προγραμμάτων ήταν ότι και τα δύο προγράμματα διανέμονται στους χρήστες διαδικτυακά και δωρεάν.

## 2.1 Σχεδιαστικό πρόγραμμα 3D γραφικών Blender

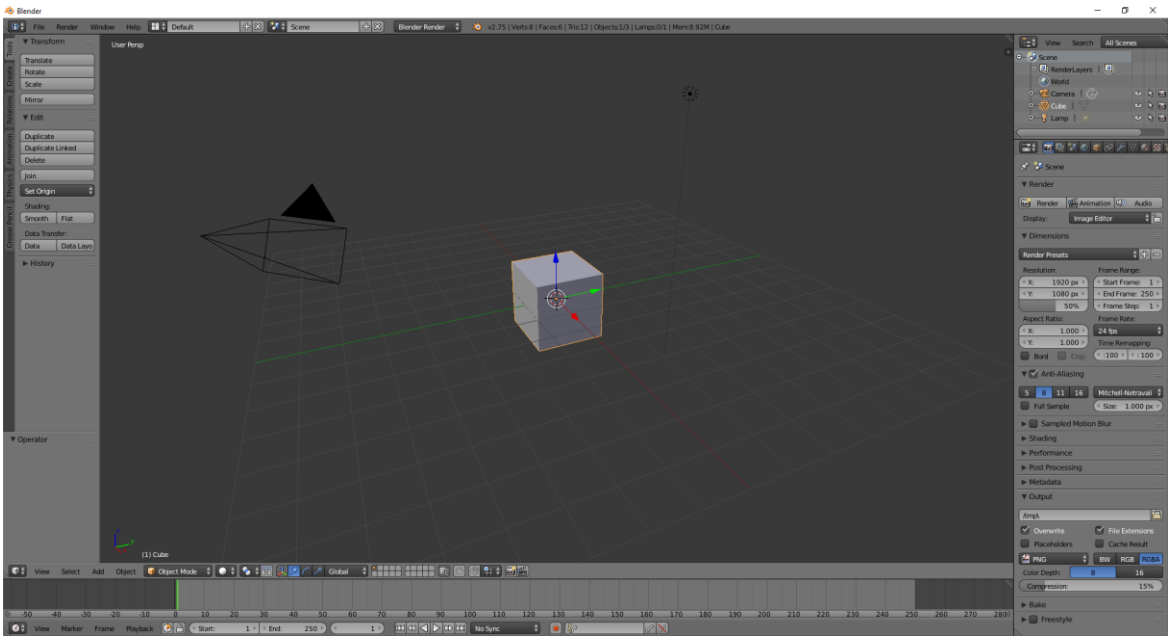
### 2.1.1 Παρουσίαση του Blender – κύρια χαρακτηριστικά

Το Blender είναι ένα πρόγραμμα σχεδίασης 3D γραφικών. Είναι ελεύθερο λογισμικό που σημαίνει ότι μπορεί να το αποκτήσει οποιοσδήποτε έχει πρόσβαση στο διαδίκτυο λόγω της άδειας χρήσης GNU- General Public License. Χρησιμοποιείται για modeling (μοντελοποίηση), animation (κινούμενα σχέδια), rendering (απόδοσης) , και για δημιουργία αλληλεπιδραστικών 3D εφαρμογών όπως τα βιντεοπαιχνίδια. Το Blender, μεταξύ άλλων, διαθέτει προχωρημένα εργαλεία για animation (κινούμενα σχέδια) καθώς επίσης και διάφορα εργαλεία για την δημιουργία υλικού. Είναι διαθέσιμο για όλα τα κύρια λειτουργικά συστήματα όπως είναι τα Windows και τα Linux.

Μερικά από τα κύρια χαρακτηριστικά του είναι: ευέλικτη δυνατότητα εσωτερικού rendering, μη γραμμική επεξεργασία βίντεο/ήχου (μοντάζ), η γλώσσα προγραμματισμού Python για προγραμματισμό λογικής και επιπλέον scripting. Την ανάπτυξη του Blender την πραγματοποίησε ο Tom Rosendaal. [4]

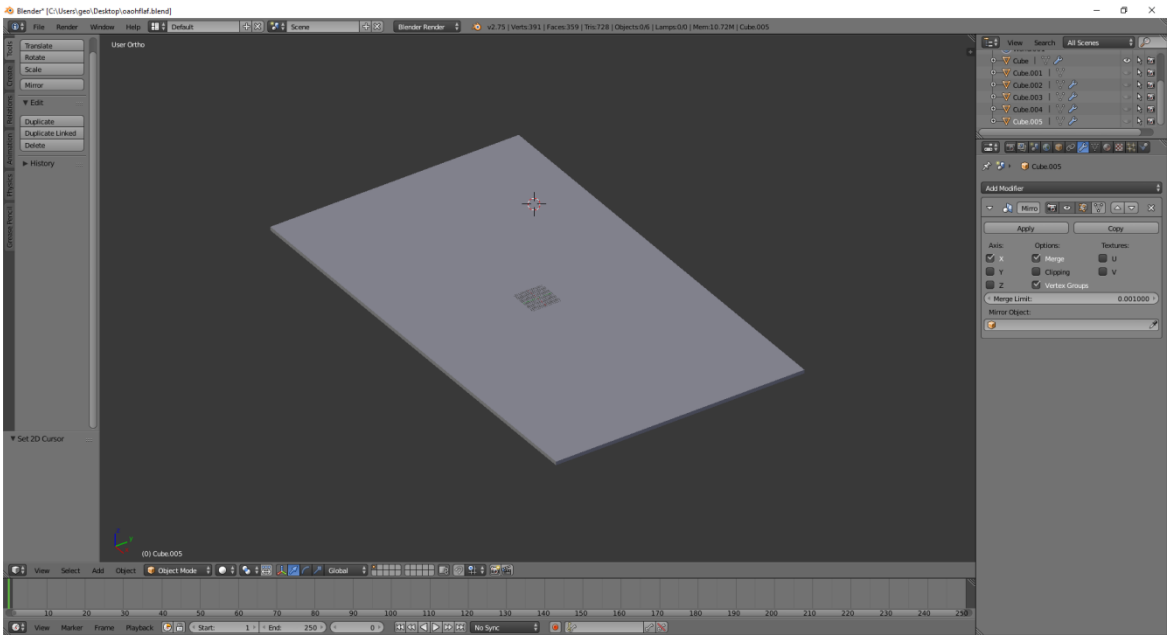
### 2.1.2 Περιβάλλον Blender – χρήση προγράμματος στην εφαρμογή

Το περιβάλλον εργασίας του Blender είναι αρκετά εύχρηστο. Αποτελείται από δύο κύρια μενού, το “Object Mode” (μενού αντικειμένου) και το “Edit Mode” (μενού επεξεργασίας). Το πρώτο μενού χρησιμοποιείται για τον χειρισμό κάθε αντικειμένου σαν μια μονάδα ενώ το δεύτερο για τον χειρισμό των δεδομένων του αντικειμένου. Για παράδειγμα επιλέγοντας το “Object Mode” (μενού αντικειμένου) μπορούμε να μετακινήσουμε, να περιστρέψουμε και να δώσουμε διαστάσεις σε ολόκληρα πολυγωνικά πλέγματα, ενώ επιλέγοντας το “Edit Mode” (μενού επεξεργασίας) μπορούμε να επεξεργαστούμε ξεχωριστά τις κορυφές και τις άκρες από κάθε πλέγμα. Η εναλλαγή μεταξύ των μενού αυτών γίνεται με το κουμπί “Tab”. Οι περισσότερες εντολές στο πρόγραμμα δίνονται μέσω συντομεύσεων με τα κουμπιά του πληκτρολογίου. Το περιβάλλον αποτελείται από διαφορετικού μεγέθους παράθυρα το καθένα με τις δικές του ενδείξεις και κουμπιά για την λειτουργία του προγράμματος όπως φαίνεται στην εικόνα παρακάτω.

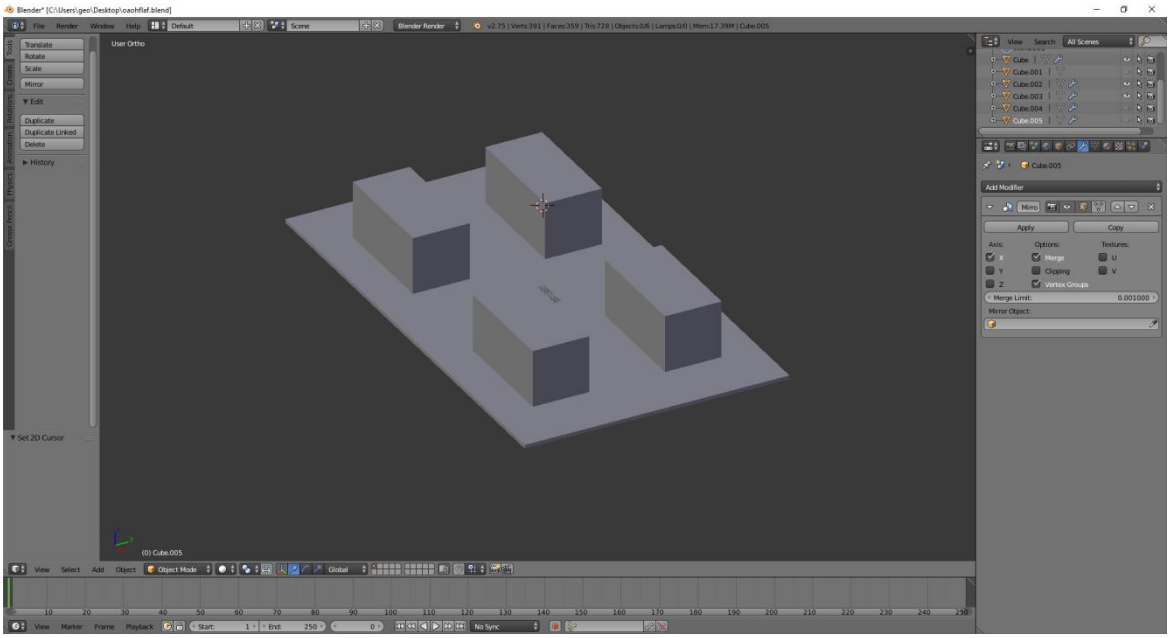


**Εικόνα 2.1.1:** Περιβάλλον Blender.

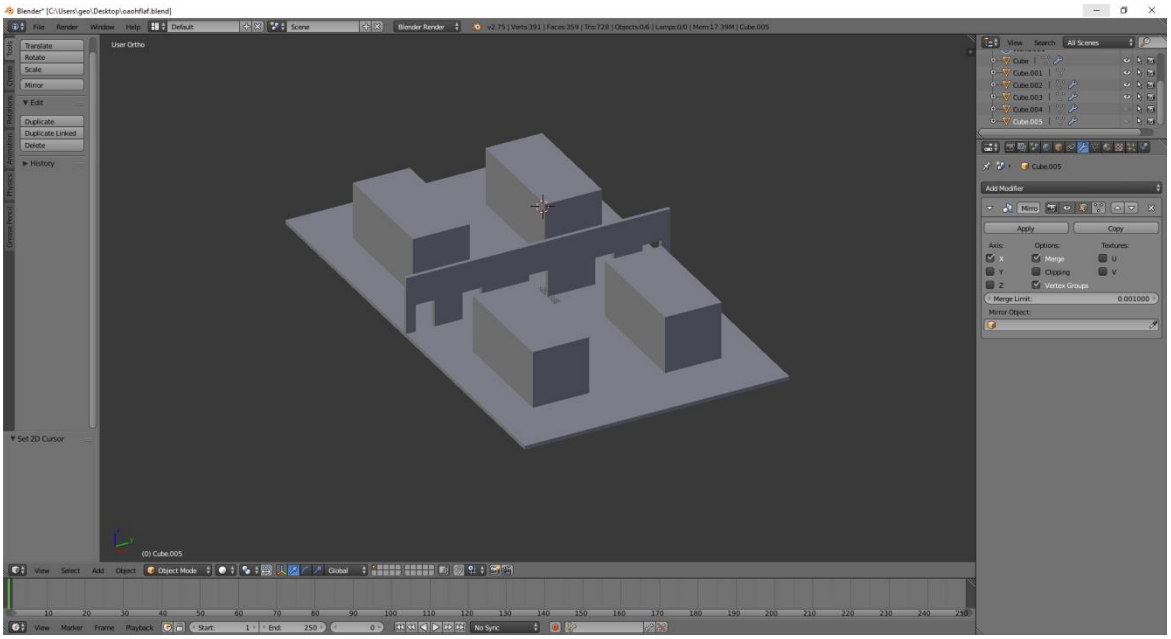
Στις παρακάτω εικόνες βλέπουμε τον σχεδιασμό της πίστας στο Blender με την χρήση γεωμετρικών στερεών σχημάτων (3D).



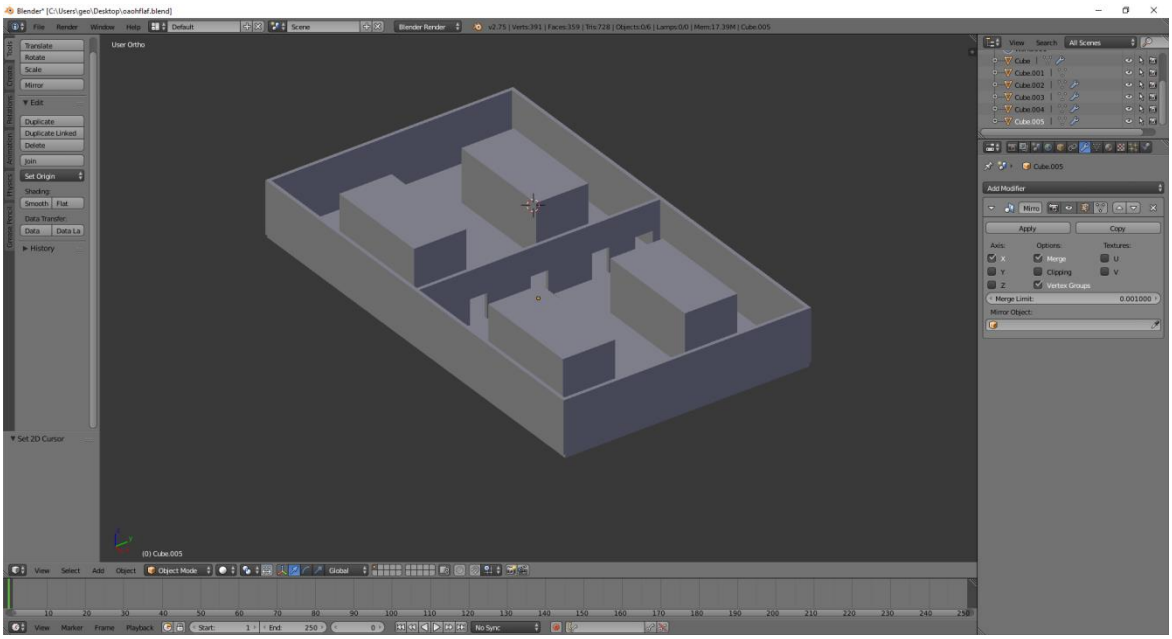
**Εικόνα 2.1.2:** Σχεδιασμός εδάφους.



**Εικόνα 2.1.3:** Σχεδιασμός εμποδίων.



**Εικόνα 2.1.4:** Σχεδιασμός διαχωριστικού τοίχους.



**Εικόνα 2.1.5:** Σχεδιασμός περίφραξης πίστας.

### 2.1.3 Παρόμοια προγράμματα

Όπως και το Blender 3D, υπάρχουν προγράμματα με παρόμοιες λειτουργίες που χρησιμοποιούνται για την σχεδίαση και μοντελοποίηση 3D αντικειμένων. Μερικά από αυτά είναι το SketchUp, to AutoDesk 3DS Max, to AutoDesk Maya Sculptris κλπ. Παρακάτω θα δούμε μερικά από τα πλεονεκτήματα και τα μειονεκτήματα των προγραμμάτων αυτών. [6] [7]

| | Πλεονεκτήματα | Μειονεκτήματα |
|---|---|---|
| SketchUP | • Ιδανικό για σχεδίαση κτιρίων.<br>• Η δωρεάν έκδοση επιτρέπει στον χρήστη να σχεδιάσει το 3D μοντέλο και να το εξάγει σε άλλο πρόγραμμα για περεταίρω επεξεργασία. | • Η δωρεάν έκδοση με την έκδοση επί πληρωμής έχουν μικρές διαφορές.<br>• Η έκδοση επί πληρωμής είναι άνω των 500$. |
| AutoDesk 3DS Max | • Δημιουργία 3D μοντέλων για ταινίες και βιντεοπαιχνίδια.<br>• Έχει χαρακτηριστικά και εργαλεία που εστιάζουν στην δημιουργία λεπτομερειών των χαρακτήρων και διαφορετικών οπτικών γωνιών του εικονικού κόσμου που έχει σχεδιαστεί.<br>• Υπάρχει δοκιμαστική έκδοση που δίνει την δυνατότητα στο χρήστη να αποφασίσει αν το πρόγραμμα καλύπτει τις ανάγκες του ή όχι. | • Η τιμή του προγράμματος είναι 3.675$.<br>• Υπάρχει μηνιαία ενοικίαση για άδεια χρήσης 195$ τον μήνα. |

| | Πλεονεκτήματα | Μειονεκτήματα |
|---|---|---|
| AutoDesk Maya | • Είναι ένα σχεδιαστικό πρόγραμμα που εστιάζει στο animation των χαρακτήρων και χρησιμοποιείται κυρίως για animated ταινίες.<br>• Υπάρχει δωρεάν έκδοση για μαθητές και προσφέρει μοναδικούς τρόπους εκμάθησης σε αυτούς. | • Η βασική έκδοση κοστίζει 4.000$.<br>• Είναι δύσκολο στην εκμάθηση. |
| Sculptris | • Είναι ιδανικό για άπειρους αλλά και για έμπειρους χρήστες.<br>• Τεχνικές πτυχές άλλων προγραμμάτων εκτελούνται αυτόματα από τον υπολογιστή με την χρήση αυτού του προγράμματος. | • Το πρόγραμμα κολλάει περιστασιακά.<br>• Δεν μπορεί να γίνει επεξεργασία της σχεδίασης του αντικειμένου από την στιγμή που ξεκινήσει η χρωματική επεξεργασία του. |
| Blender | • Χρήσιμο για την δημιουργία 3D animation.<br>• Είναι δωρεάν και εξίσου καλό με το Maya που είναι επί πληρωμής.<br>• Υπάρχουν πολλοί χρήσιμοι οδηγοί στην ιστοσελίδα του.<br>• Δυνατότητα εισαγωγής και εξαγωγής αρχείων και αντικειμένων για επεξεργασία. | • Ο υπολογιστής πρέπει να έχει σύγχρονο υλικό για να λειτουργήσει το πρόγραμμα χωρίς να κολλάει.<br>• Το περιβάλλον εργασίας του είναι λίγο περίπλοκο. |

**Πίνακας 1:** Σύγκριση Σχεδιαστικών Προγραμμάτων.

## 2.2 Πρόγραμμα cross-platform παιχνιδιών Unity

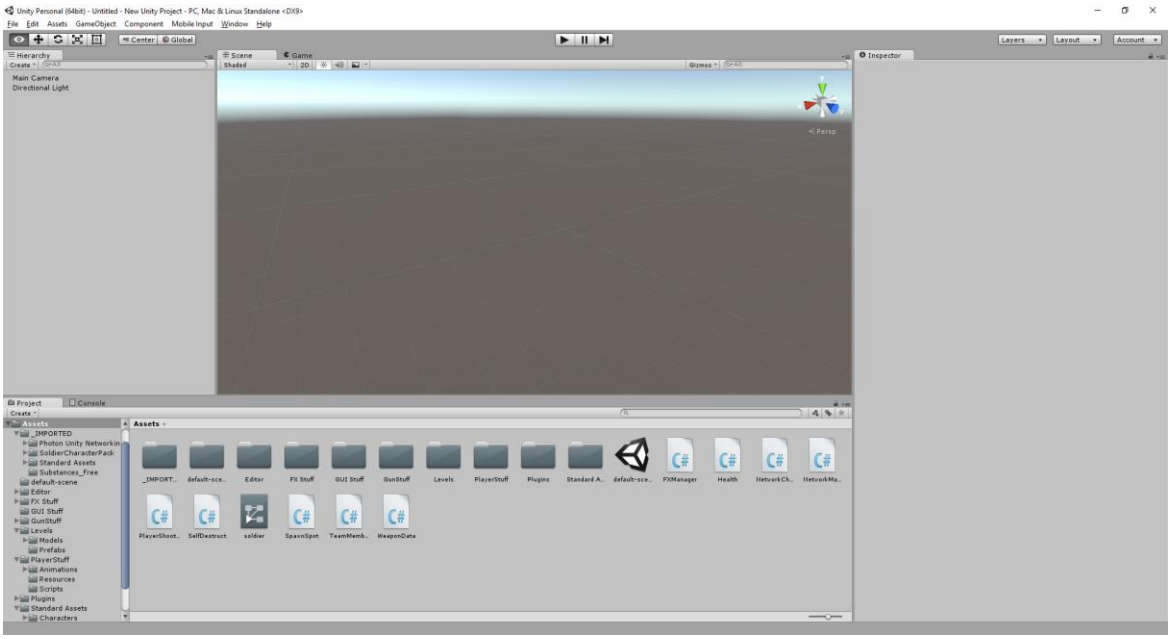
### 2.2.1 Παρουσίαση Unity

Το Unity είναι ένα πρόγραμμα, ή καλύτερα μια μηχανή cross-platform παιχνιδιών (πρόκειται για παιχνίδια που μπορούν να τα παίξουν ταυτόχρονα πολλοί χρήστες, ο καθένας από την δική του κονσόλα) η οποία αναπτύχθηκε από την Unity Technologies. Χρησιμοποιείται για την ανάπτυξη βιντεοπαιχνιδιών για Η/Υ, κονσόλες, κινητά τηλέφωνα και ιστοσελίδες.

Το Unity περιλαμβάνει γραφικά, ήχους, animations (κινούμενα σχέδια), σύστημα φυσικής (βαρύτητα, κίνηση, κλπ.), και πολλά ακόμα χαρακτηριστικά που είναι αναγκαία για την ολοκληρωμένη ανάπτυξη βιντεοπαιχνιδιών. Οι γλώσσες ανάπτυξης του Unity είναι C, C++ και C#. Οι γλώσσες προγραμματισμού των Script είναι C# και Javascript.[3]

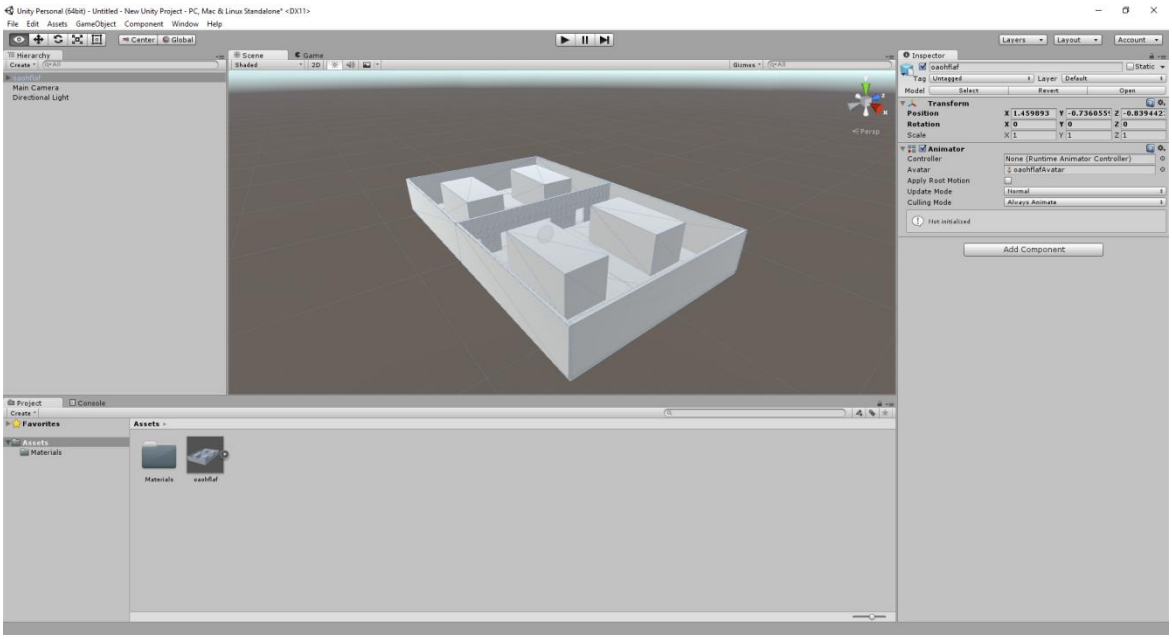
### 2.2.2 Περιβάλλον Unity - χρήση προγράμματος στην εφαρμογή

Το περιβάλλον του Unity είναι σχεδιασμένο έτσι ώστε οι χρήστες να έχουν στην διάθεσή τους οποιαδήποτε εργαλεία χρειάζονται σε όλη την διάρκεια της επεξεργασίας. Αποτελείται από διαφορετικού μεγέθους παράθυρα εκ των οποίων το ένα είναι η οθόνη επεξεργασίας. Γύρω από αυτό βρίσκονται παράθυρα με ενδείξεις τιμών και παραμέτρων των αντικειμένων που χρησιμοποιούνται καθώς επίσης οι ονομασίες και οι θέσεις των αντικειμένων και των αρχείων που επεξεργάζονται. Στο κάτω μέρος βρίσκεται το παράθυρο της κονσόλας όπου εμφανίζονται τα μηνύματα του συστήματος καθώς και τα πιθανά σφάλματα των Script. Η εισαγωγή αντικειμένων γίνεται εύκολα με την χρήση του ποντικιού.

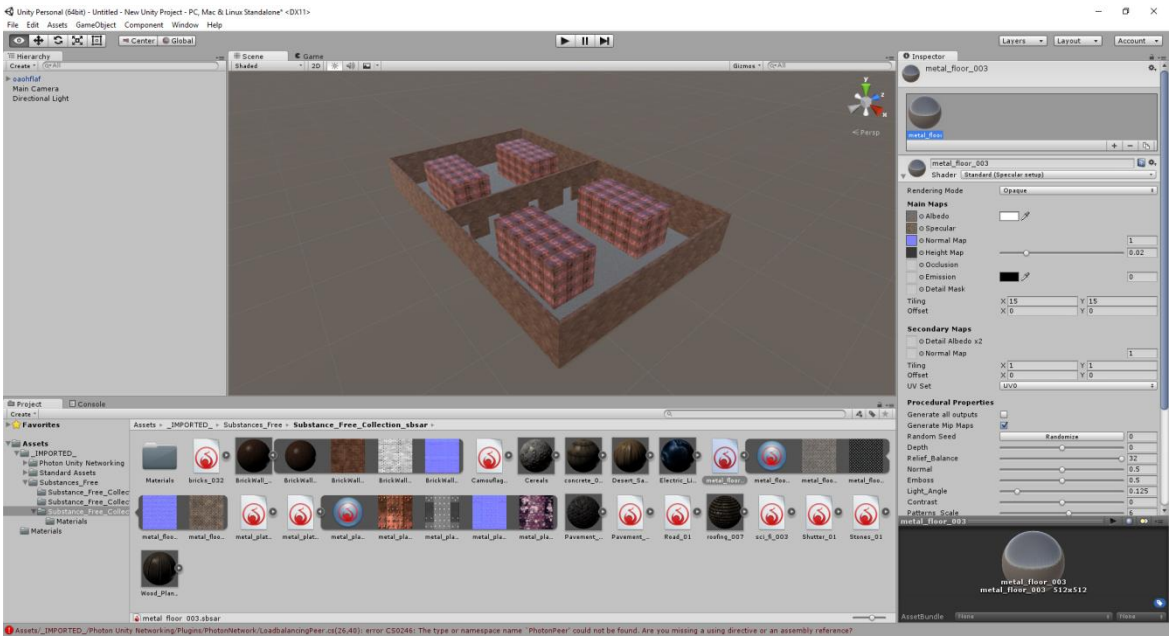


**Εικόνα 2.2.1:** Περιβάλλον Unity.

Ξεκινώντας, αφού σχεδιάσαμε την πίστα στο Blender, αρχίσαμε την επεξεργασία της στο Unity προσθέτοντας χρώματα στο περιβάλλον της χρησιμοποιώντας υλικά τα οποία βρήκαμε στο Unity Store όπως φαίνεται παρακάτω. (Τα υλικά χρησιμοποιούνται με drug & drop κατευθείαν στο αντικείμενο που θέλουμε να ‘χρωματίσουμε’)



**Εικόνα 2.2.2:** Πίστα στο Unity.



**Εικόνα 2.2.3:** Χρήση materials.

### 2.2.3 Παρόμοια Προγράμματα

Στις μέρες μας έχουν αναπτυχθεί κι άλλες μηχανές δημιουργίας cross-platform παιχνιδιών. Μερικές από αυτές είναι, η Source, η Unreal Engine, η CryENGINE κλπ. Όπως και η Unity έτσι και οι υπόλοιπες μηχανές χρησιμοποιούνται για την δημιουργία παιχνιδιών σε κονσόλες. Η Source, όπως και η CryENGINE, είναι μηχανές που χρησιμοποιούνται κυρίως για την ανάπτυξη παιχνιδιών συμβατά με υπολογιστές και κονσόλες σε αντίθεση με τη Unity που χρησιμοποιείται κυρίως για κινητά τηλέφωνα και υπολογιστές. Η Unreal Engine από την άλλη χρησιμοποιείται για όλες τις κονσόλες, τους υπολογιστές αλλά και για κινητά τηλέφωνα.

Η καθεμία έχει τα δικά της χαρακτηριστικά, το δικό της περιβάλλον εργασίας και τη δική της συμβατότητα μεταξύ των υπολογιστικών συστημάτων των χρηστών. Όλες οι μηχανές διανέμονται δωρεάν στους προγραμματιστές μέσω διαδικτύου δίνοντας την δυνατότητα σε οποιονδήποτε χρήστη να ασχοληθεί με την ανάπτυξη παιχνιδιών, προσφέροντας μοναδικά γραφικά χαρακτηριστικά και υλικά. [8]

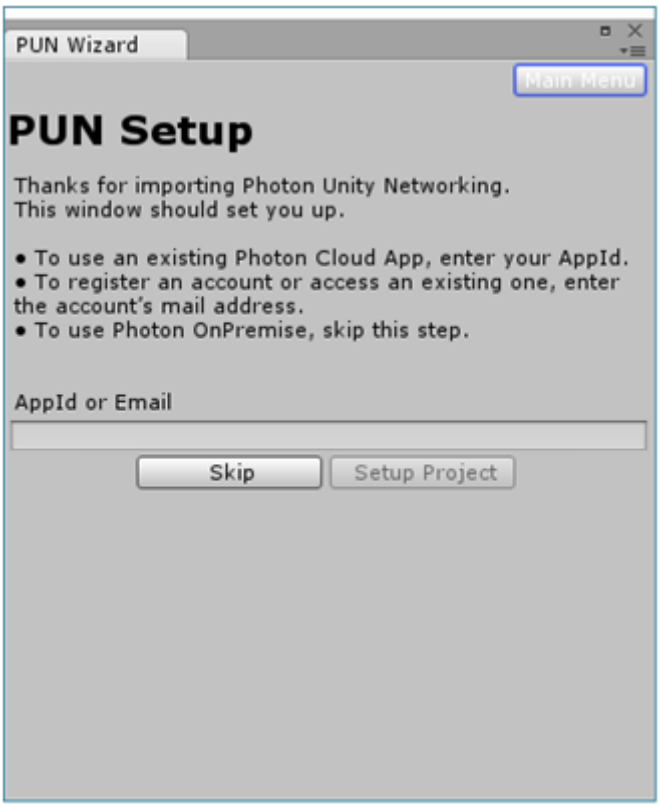
# ΚΕΦΑΛΑΙΟ 3

# ΑΝΑΛΥΣΗ ΔΙΑΔΙΚΑΣΙΑΣ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΗΣ

## 3.1 Εγκατάσταση Server Διασύνδεσης Χρηστών

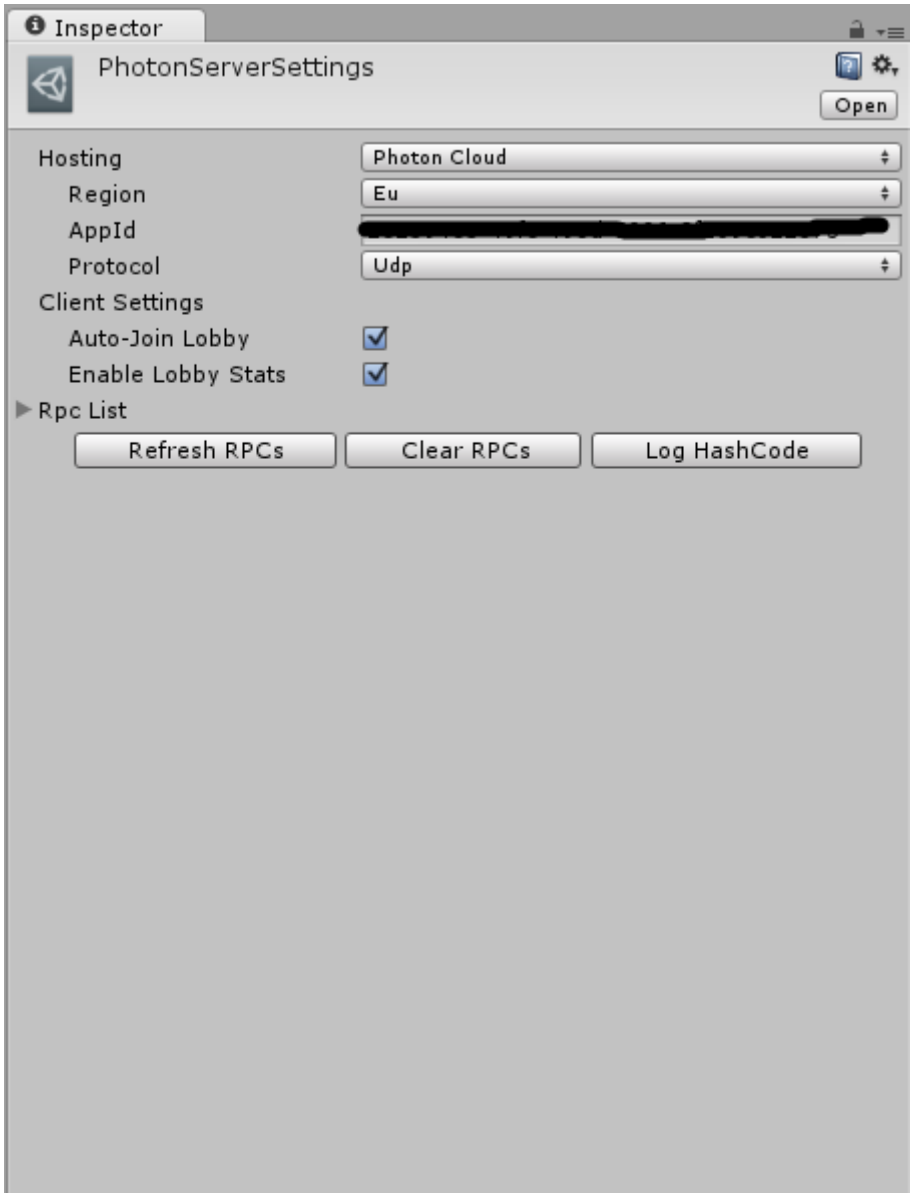
Για την συνδεσιμότητα των χρηστών στην εφαρμογή χρησιμοποιείται ένας Server (διακομιστής) . Πρόκειται για μια κονσόλα η οποία είναι υπεύθυνη για την αμφίδρομη μεταφορά δεδομένων μεταξύ των χρηστών αλλά και μεταξύ του χρήστη με τον Server έτσι ώστε όλοι οι χρήστες να έχουν την ίδια έκδοση της εφαρμογής. Αυτό έχει ως αποτέλεσμα τον συντονισμό της εφαρμογής με τα δεδομένα κάθε χρήστη.

Το Unity παρέχει στους χρήστες δωρεάν plugins τα οποία βοηθούν στην καλύτερη ανάπτυξη των προγραμμάτων. Ένα από αυτά είναι και το Photon Unity Networking το οποίο παρέχει στον χρήστη ένα Cloud (photon cloud) το οποίο λειτουργεί σαν Server (διακομιστής) για την διευκόλυνση ανάπτυξης διαδικτυακών εφαρμογών όπως η συγκεκριμένη. Έτσι ρυθμίζοντας το Cloud στις προδιαγραφές της δικής μας εφαρμογής εγκαταστήσαμε τον Server (διακομιστής) όπως βλέπουμε παρακάτω.




**Εικόνα 3.1.1:** Εγκατάσταση του Photon Cloud (Server).

Για την εγκατάσταση συμπληρώνουμε την φόρμα με το email μας, όπως φαίνεται παραπάνω, και επιλέγουμε “Setup Project”. Στην συνέχεια επιλέξαμε τις δικές μας ρυθμίσεις και ολοκληρώθηκε η εγκατάσταση. Παρακάτω βλέπουμε τις ρυθμίσεις αυτές: περιοχή, αναγνωριστικός αριθμός(AppId), πρωτόκολλο, κλπ (Για λόγους ασφαλείας το AppId αποκρύπτεται).

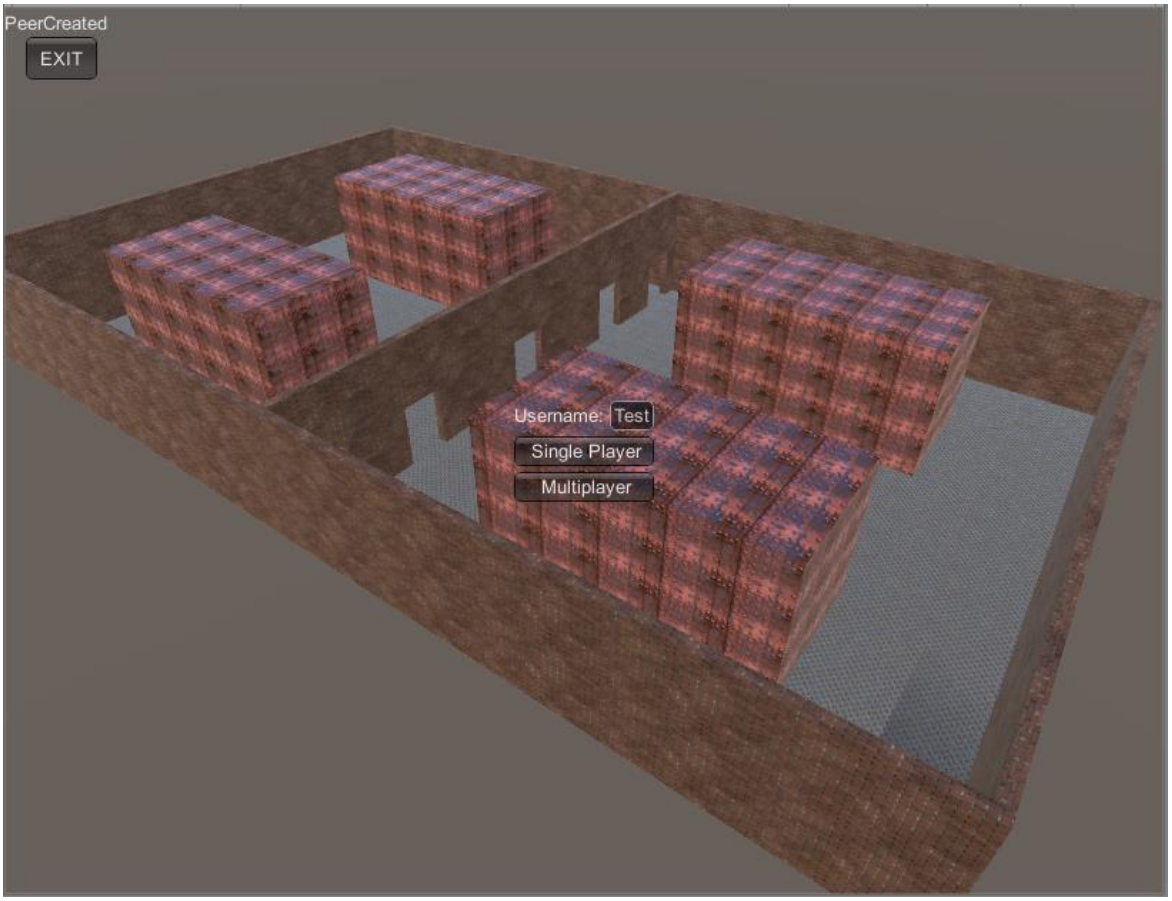



**Εικόνα 3.1.2:** Εγκατάσταση του Photon Cloud (Server).

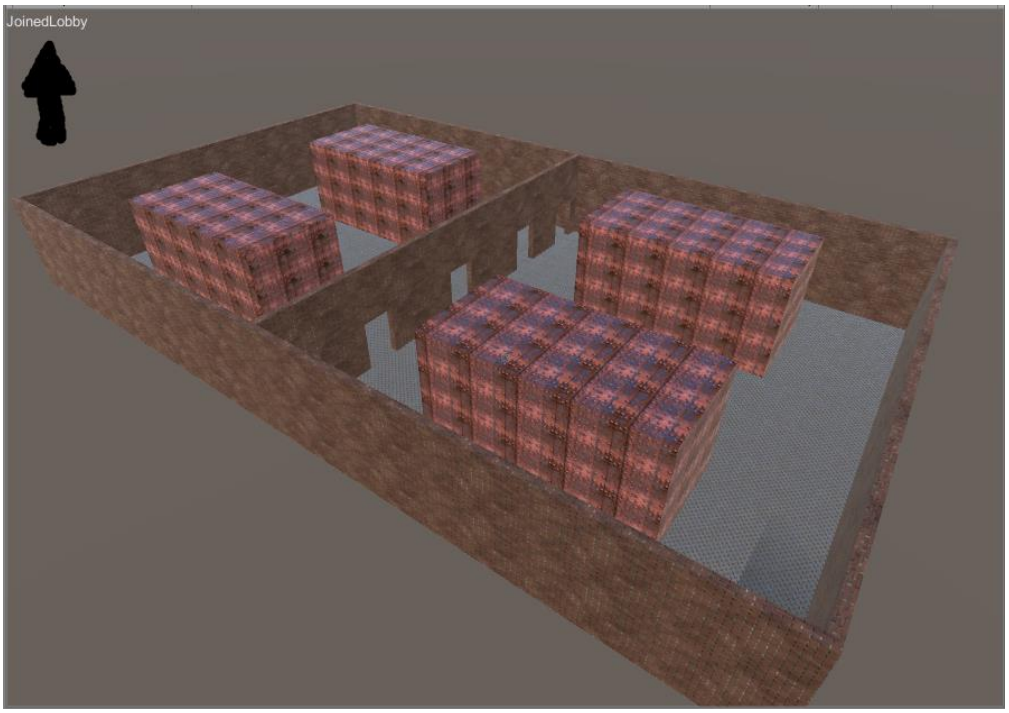
### 3.2 Συγχρονισμός των χρηστών με τον Server

Τα διαδικτυακά ηλεκτρονικά παιχνίδια αναπαράγονται ταυτόχρονα από τις κονσόλες των χρηστών. Αυτό σημαίνει ότι οι χρήστες παίζοντας το ίδιο παιχνίδι μοιράζονται ένα κοινό γραφικό περιβάλλον. Αυτό με την σειρά του απαιτεί τον συγχρονισμό μεταξύ των χρηστών έτσι ώστε οι δράσεις κάθε παίκτη να εμφανίζονται στην οθόνη των υπόλοιπων κάθε στιγμή. Με αυτόν τον τρόπο επιτυγχάνεται η καλύτερη αναπαραγωγή της εφαρμογής. Αξίζει να αναφερθεί ότι σε κάθε χρήστη υπάρχουν διαφορές στην ώρα εκτέλεσης κάθε δράσης οι οποίες όμως είναι πολύ μικρές, σε επίπεδο millisecond.

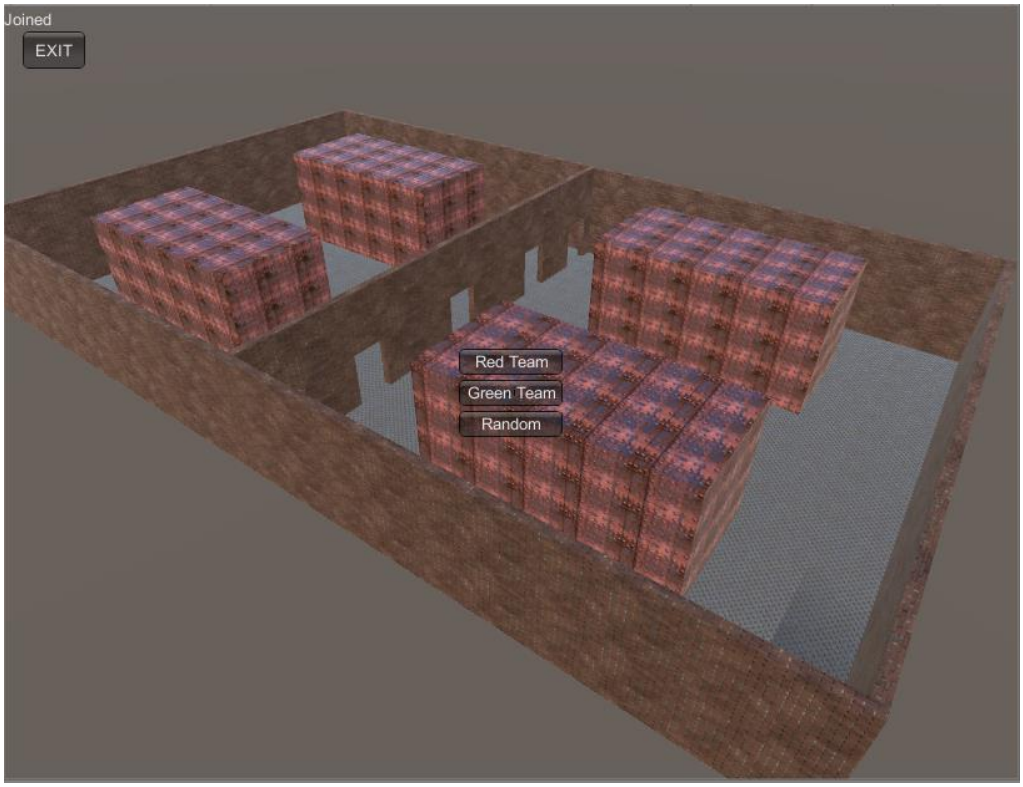
Για τον συγχρονισμό των χρηστών δημιουργήσαμε δύο Script. Το πρώτο Script είναι το '**NetworkManager**' το οποίο εμπεριέχει: συναρτήσεις για την συνδεσιμότητα των χρηστών σε 'δωμάτιο αναμονής', το γραφικό μενού με τις επιλογές για την σύνδεση του χρήστη (User name, mode, κλπ) καθώς και τα χαρακτηριστικά κάθε παίκτη. Παρακάτω βλέπουμε ενδεικτικές φωτογραφίες από δοκιμές.




**Εικόνα 3.2.1:** Γραφικό μενού τρόπου σύνδεσης (GUI).




**Εικόνα 3.2.2:** Σύνδεση χρήστη στο ‘δωμάτιο αναμονής’.




**Εικόνα 3.2.3:** Επιλογή ομάδας (GUI).

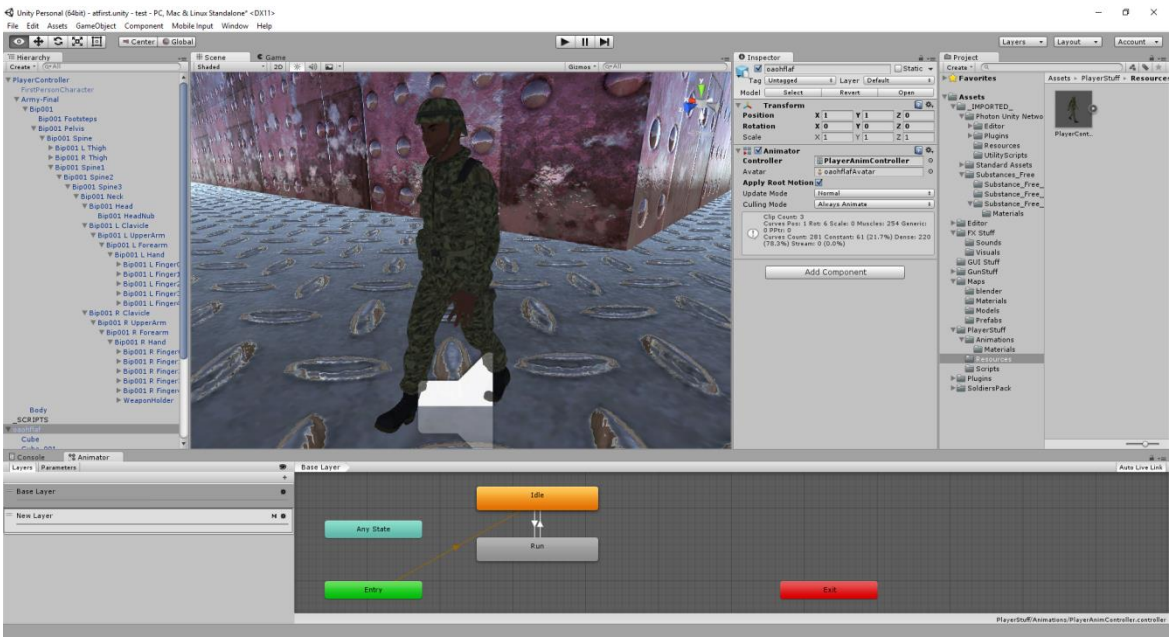
Το δεύτερο Script είναι το '**NetworkCharacter**' το οποίο εμπεριέχει συναρτήσεις για τον συγχρονισμό των παικτών μέσω του Server (διακομιστής). Αυτό γίνεται με την αμφίδρομη μεταφορά των θέσεων και των animation που χρησιμοποιεί κάθε παίκτης κάθε στιγμή, με τον Server (διακομιστής), έτσι ώστε όλοι οι παίκτες να βλέπουν τα ίδια πράγματα. Η ανάλυση και τον δύο Script γίνεται πιο λεπτομερής στο επόμενο κεφάλαιο.

### 3.3 Γραφικά των παικτών

Σε κάθε παιχνίδι χρησιμοποιεί ένα συγκεκριμένο σύνολο γραφικών για την δημιουργία και την ροή της εικόνας. Τα γραφικά συμβάλλουν στην καλαισθησία του περιβάλλοντος της εφαρμογής. Όσο πιο εξελιγμένα και λεπτομερή είναι τα γραφικά τόσο πιο απαιτητική γίνεται η εφαρμογή. Αυτό σημαίνει ότι για την καλύτερη αναπαραγωγή της εφαρμογής θα πρέπει να είναι εξίσου εξελιγμένα και τα συστήματα που θα την εκτελέσουν. Έτσι τα γραφικά θα έχουν καλύτερη ροή και θα δημιουργούν πιο ολοκληρωμένες εικόνες.

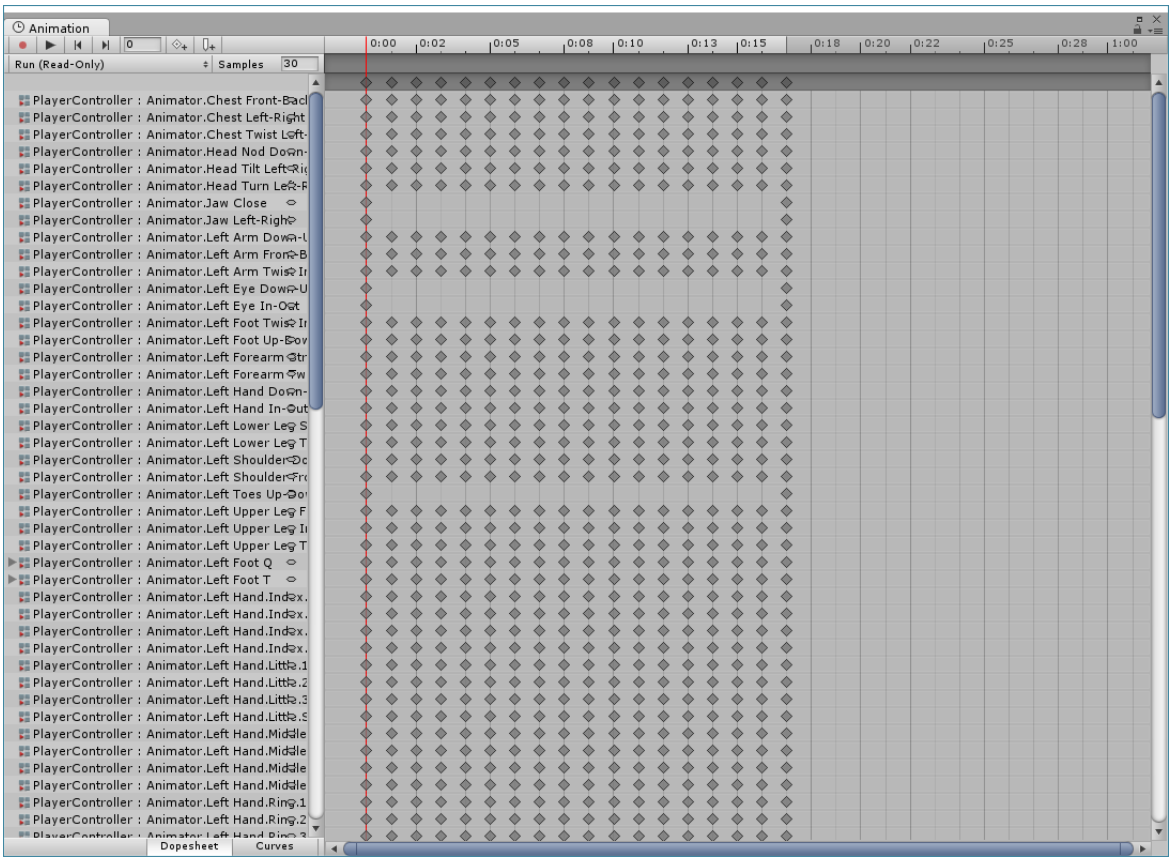
Το Unity παρέχει ορισμένα γραφικά δωρεάν στους χρήστες. Για να δημιουργήσουμε τα animation των χαρακτήρων εφαρμόσαμε τα γραφικά του Unity στο σκελετό του χαρακτήρα, αφού πρώτα επιλέξαμε την κατάλληλη μορφή του χαρακτήρα μας.

.



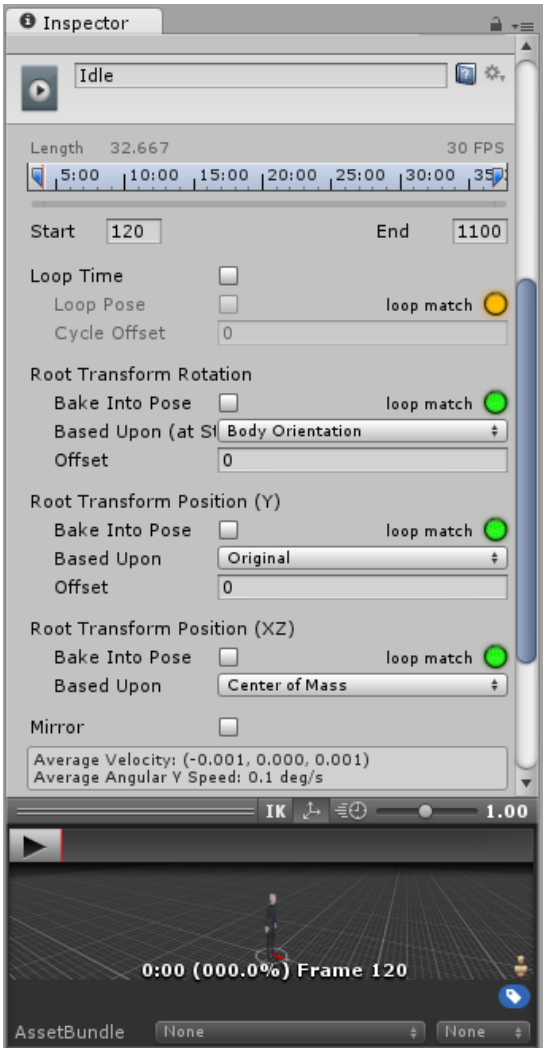

**Εικόνα 3.3.1:** Μορφή του χαρακτήρα.

Στην παραπάνω εικόνα βλέπουμε ένα στιγμιότυπο του χαρακτήρα που χρησιμοποιεί κάθε παίκτης. Στην στήλη που βρίσκεται αριστερά από τον χαρακτήρα βλέπουμε αναλυτικά κάθε ‘κομμάτι’ του σώματος. Στην συνέχεια δημιουργήσαμε το animation (κινούμενο σχέδιο) ακινησίας και τρεξίματος, εφαρμόζοντας σε κάθε κομμάτι του σώματος τις κατάλληλες κινήσεις, όπως φαίνεται στην παρακάτω εικόνα.

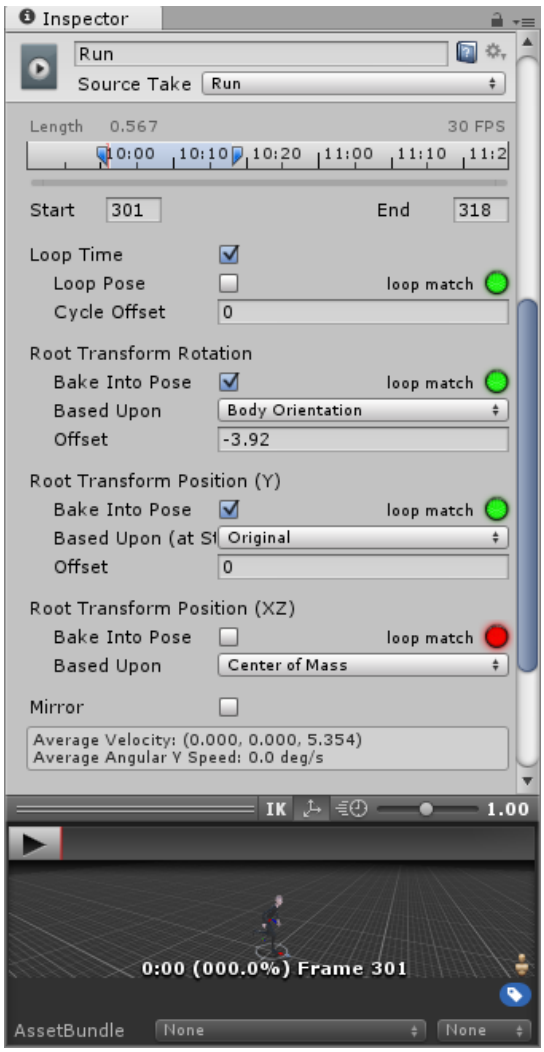



**Εικόνα 3.3.2:** Αναλυτική εφαρμογή κινήσεων.

Στην συνέχεια συγχρονίσαμε τα animation της ακινησίας και του τρεξίματος με το σώμα του χαρακτήρα βάζοντας σε κάθε κίνηση αρχή και τέλος έτσι ώστε η κίνηση να γίνεται επαναλαμβανόμενα ξεκινώντας από την αρχή κάθε φορά που φτάνει στο τέλος της. Με αυτόν τον τρόπο πετυχαίνουμε μια πιο απαλή ροή στην ολοκληρωμένη κίνηση του παίκτη.

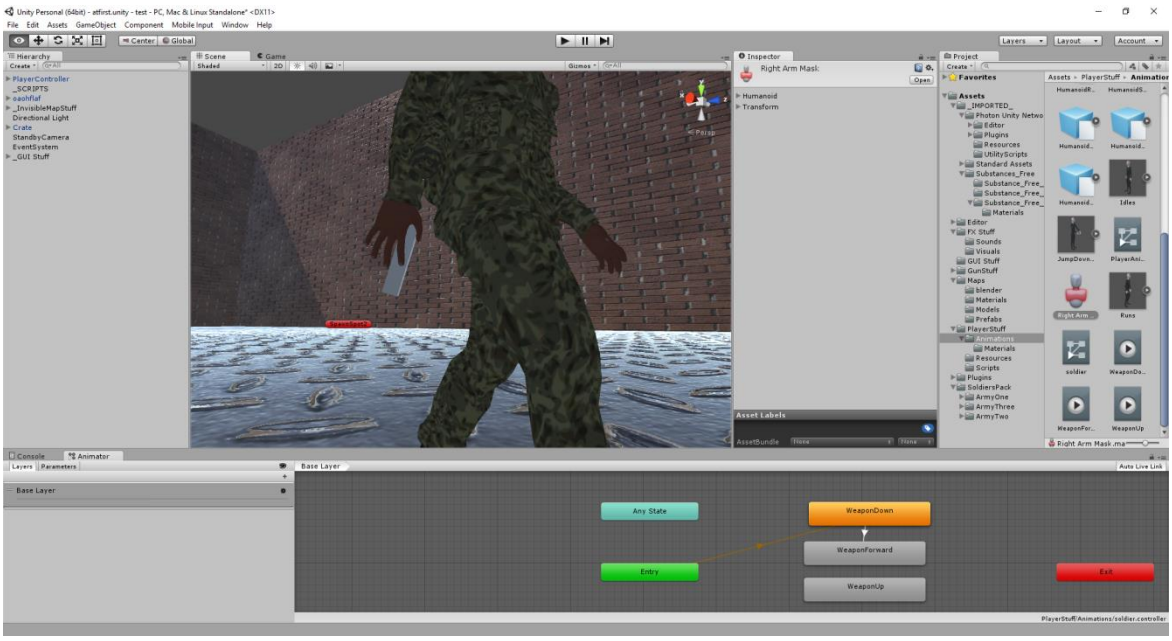



**Εικόνα 3.3.3:** Συγχρονισμός Ακινησίας.




**Εικόνα 3.3.4:** Συγχρονισμός Τρεξίματος.

Τέλος σχεδιάσαμε το όπλο που χρησιμοποιούν οι παίκτες και το εφαρμόσαμε στο δεξί χέρι του χαρακτήρα μας όπως φαίνεται παρακάτω. (Η προσκόλληση του όπλου δεν μπορεί να γίνει με απόλυτη λεπτομέρεια λόγω περιορισμού ελέγχου του χεριού του χαρακτήρα)




**Εικόνα 3.3.5:** Μορφή του όπλου.

### 3.4 Προγραμματισμός χειρισμού των παικτών

Για τον χειρισμό του παίκτη δημιουργήσαμε ένα Script (**PlayerMovement**) με το οποίο ελέγχουμε οποιαδήποτε κίνηση γίνεται στον παίκτη. Η λεπτομερής ανάλυση της λειτουργίας του συγκεκριμένου Script γίνεται σε επόμενο κεφάλαιο. Ο παίκτης κινείται εξ' ορισμού με τα κουμπιά: W,S,A,D ή τα βελάκια του πληκτρολογίου. Με το W κινείται μπροστά, με το S πίσω, με το A αριστερά και με το D δεξιά. Η κάμερα και ο στόχος του όπλου κινούνται με το ποντίκι του υπολογιστή και ο πυροβολισμός γίνεται με το αριστερό κλικ του ποντικιού. Ο κώδικας βρίσκεται αναλυτικά στο παράρτημα και μπορείτε να το Δείτε εδώ.

### 3.5 Προβλήματα που αντιμετωπίστηκαν

Κατά τη διάρκεια της εκπόνησης της πτυχιακής εργασίας αντιμετωπίσαμε διάφορα προβλήματα τα οποία αφορούσαν περισσότερο την εγκατάσταση και παρουσίαση των animation (κινούμενων σχεδίων) της εφαρμογής.

#### 3.5.1 Animation άλματος

Ένα από τα προβλήματα που αντιμετωπίσαμε ήταν το άλμα του παίκτη. Κάθε φορά που κάποιος παίκτης κάνει άλμα πρέπει να ενεργοποιείται το αντίστοιχο animation (κινούμενο σχέδιο), δηλαδή ο παίκτης να απομακρύνεται από το έδαφος στον κατακόρυφο άξονα, τα πόδια του να λυγίζουν και λίγο πριν επιστρέψει στο έδαφος τα πόδια να τεντώνουν στην αρχική τους θέση. Το πρόβλημα που αντιμετωπίσαμε σε αυτό το κομμάτι ήταν ότι τα πόδια του παίκτη μετά το άλμα δεν τέντωναν στην αρχική τους θέση, με αποτέλεσμα ο παίκτης να φαίνεται σκυφτός συνέχεια. Αυτό είχε ως αποτέλεσμα, με την σειρά του, να αφαιρέσουμε τελείως την κίνηση και το animation του άλματος, καθώς δεν καταφέραμε να βρούμε λύση.

### 3.5.2 Animation τρεξίματος

Ένα ακόμα πρόβλημα που αντιμετωπίσαμε ήταν το animation (κινούμενο σχέδιο) του τρεξίματος του παίκτη. Κάθε παίκτης μπορεί να τρέξει προς οποιαδήποτε κατεύθυνση κάνοντας βήματα προς τα μπρος, πίσω, δεξιά, αριστερά και διαγώνια. Το πρόβλημα που αντιμετωπίσαμε σε αυτό το κομμάτι είναι ότι οποιαδήποτε κίνηση κάνει ο παίκτης ενεργοποιείται το animation (κινούμενο σχέδιο) της κίνησης που θα έκανε αν κινούταν προς τα μπρος. Επίσης όταν ένας παίκτης επιλέξει κάποια άλλη κίνηση εκτός από την μπροστινή τότε στην κίνηση αυτή προστίθεται η ταχύτητα και η κατεύθυνση της μπροστινής κίνησης. Για παράδειγμα, αν ο παίκτης επιλέξει να τρέξει προς τα πίσω τότε ενεργοποιείται και η μπροστινή κίνηση με αποτέλεσμα να μειώνεται η ταχύτητα κίνησης αφού ο παίκτης κινείται προς τα πίσω αλλά ταυτόχρονα προσπαθεί να κινηθεί μπροστά. Και αυτό το πρόβλημα δεν έχει λυθεί.

### 3.5.3 Περιορισμός υλικού ανάπτυξης

Τα animation (κινούμενα σχέδια) των παικτών τα αποκτήσαμε από το Unity Store, το οποίο παρέχει δωρεάν υλικό για την ανάπτυξη παιχνιδιών σε περιβάλλον Unity. Αυτό σημαίνει ότι ήμασταν περιορισμένοι στο τι animation θα χρησιμοποιήσουμε αφού το υλικό αυτό είναι συγκεκριμένο και για την απόκτηση περαιτέρω υλικού έπρεπε να γίνει πληρωμή.

### 3.5.4 Κέρσορας

Κατά την διάρκεια του παιχνιδιού, από την στιγμή που συνδέεται ο παίκτης στην πίστα και έπειτα, ο κέρσορας του ποντικιού είναι ακόμα εμφανής και ενεργοποιημένος. Το πρόβλημα που δημιουργείται με αυτό τον τρόπο είναι ότι μπορεί άθελά του ο παίκτης κατά την διάρκεια του παιχνιδιού να πατήσει καταλάθος το κουμπί “EXIT” που βρίσκεται στο επάνω και αριστερά μέρος της οθόνης. Δεν καταφέραμε να βρούμε τρόπο να απενεργοποιήσουμε τον κέρσορα κατά την διάρκεια της μάχης.

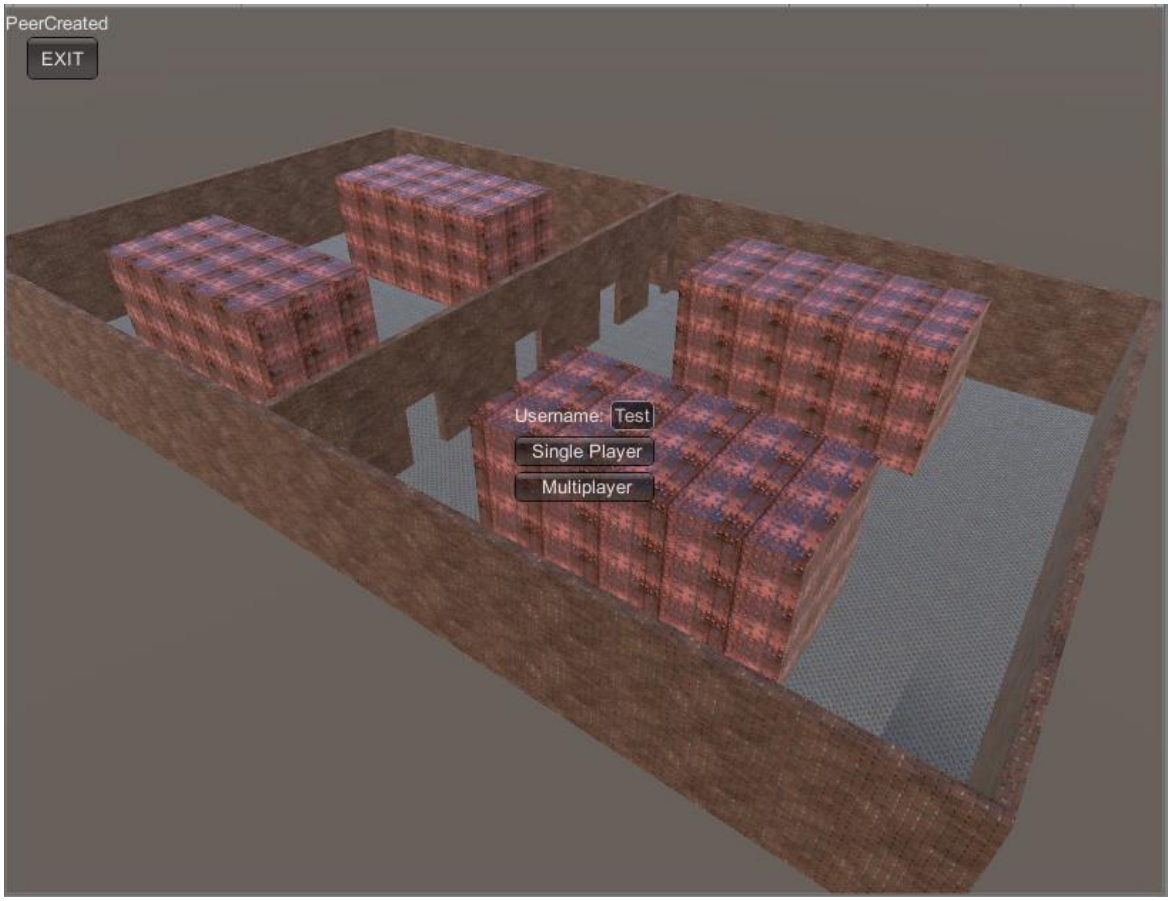
# ΚΕΦΑΛΑΙΟ 4

# ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ

## 4.1 Παρουσίαση Εφαρμογής

Στα προηγούμενα κεφάλαια είδαμε από πού προέρχεται το είδος του παιχνιδιού μας, τι εργαλεία χρησιμοποιήθηκαν καθώς και όλα τα βήματα σχεδιασμού, ανάπτυξης και υλοποίησης της εφαρμογής μας. Στο κεφάλαιο αυτό λοιπόν θα δούμε την ίδια την εφαρμογή σε εκτέλεση.

Εκτελώντας την εφαρμογή, κάθε χρήστης, στην αρχή έρχεται αντιμέτωπος με το κύριου μενού του παιχνιδιού. Οι επιλογές που έχει είναι ποικίλες. Αρχικά καλείται να συμπληρώσει το όνομα του ή κάποιο ψευδώνυμο της αρεσκείας του με το οποίο θα αναγνωρίζεται από τους άλλους παίκτες όταν συνδέεται στο πεδίο μάχης. Στην συνέχεια έχει την δυνατότητα να επιλέξει μεταξύ των μενού “Single Player” και “Multiplayer”. Επίσης υπάρχει και η επιλογή “EXIT” η οποία, σε αντίθεση με τις υπόλοιπες επιλογές που έχει ο χρήστης, μπορεί να γίνει ανά πάσα στιγμή και είναι η επιλογή που οδηγεί στην έξοδο της εφαρμογής.




**Εικόνα 4.1:** Αρχικό μενού επιλογών.

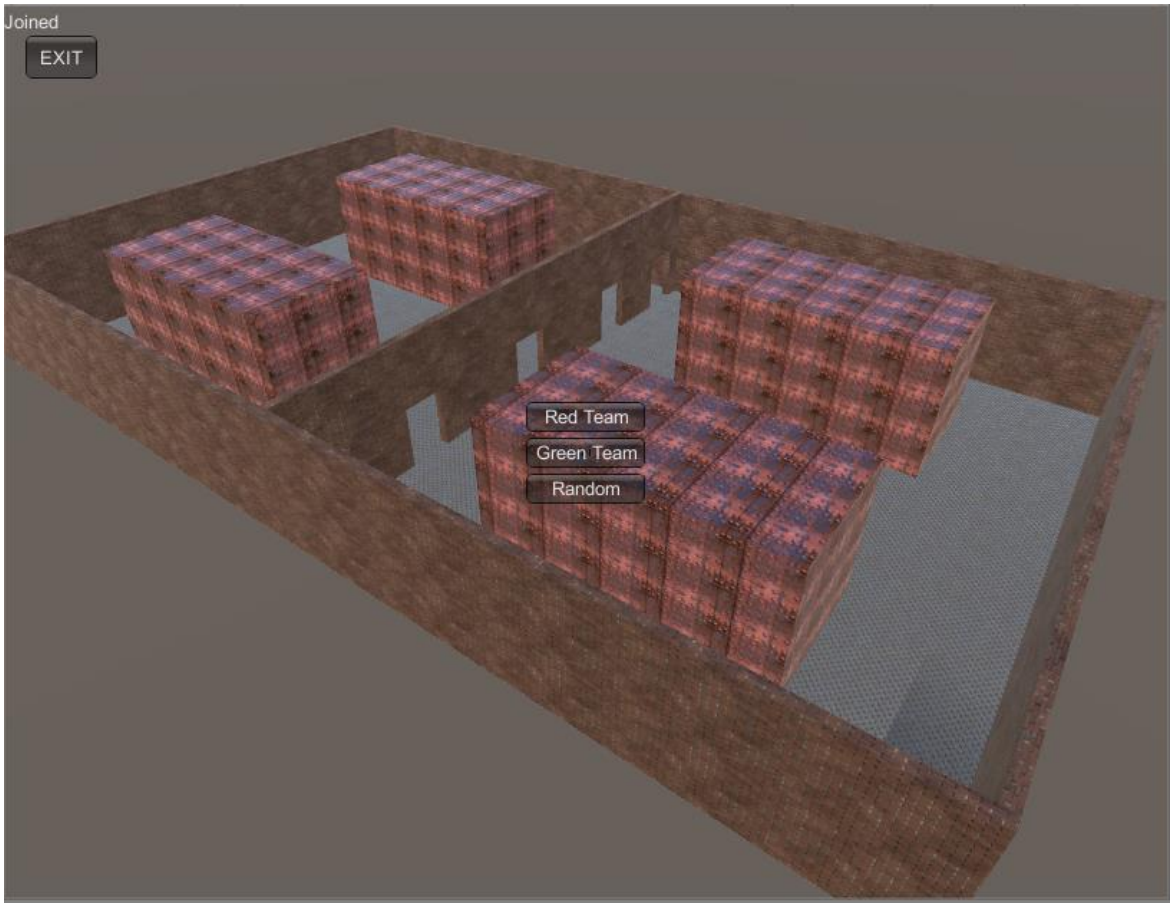
Οποιοδήποτε επιλογή από τις άλλες δύο κάνει ο χρήστης, το επόμενο μενού που θα εμφανιστεί είναι το ίδιο. Ένα μενού από το οποίο μπορεί ο χρήστης να επιλέξει μία εκ των δύο ομάδων, την κόκκινη ή την πράσινη. Έχει επίσης και μια τρίτη επιλογή, τον τυχαίο ορισμό της ομάδας του. Εδώ αξίζει να σημειωθεί όμως μία σημαντική διαφορά μεταξύ του μενού επιλογής ομάδας όταν ο χρήστης έχει επιλέξει “Single Player” και όταν έχει επιλέξει “Multiplayer”.

Στην πρώτη περίπτωση ο χρήστης συνδέεται αμέσως στην πίστα και αφού επιλέξει ομάδα εμφανίζεται πάνω σε αυτήν μόνος του. Δεν είναι συνδεδεμένος διαδικτυακά που σημαίνει ότι δεν έχει την δυνατότητα σύνδεσης σε αυτή την πίστα άλλος χρήστης. Το μενού αυτό εξυπηρετεί σε δοκιμές του παίκτη και έχει προοπτικές βελτίωσης καθώς μελλοντικά μπορούν να προστεθούν παίκτες τεχνητής νοημοσύνης για να παίζει ο χρήστης μόνος του εκτός σύνδεσης.

Στην δεύτερη περίπτωση ο χρήστης πρώτα συνδέεται με τον Server (διακομιστής) και δημιουργεί ένα “δωμάτιο”, εξασφαλίζει δηλαδή έναν χώρο στον

Server (διακομιστής) στον οποίο μπορούν να συνδεθούν και άλλοι χρήστες και να παίξουν στην ίδια πίστα. Στην περίπτωση μας ο Server (διακομιστής) είναι πολύ μικρός που σημαίνει ότι και ταυτόχρονοι χρήστες είναι λίγοι. Αφού λοιπόν γίνει η σύνδεση με τον Server (διακομιστής) και δημιουργηθεί το “δωμάτιο”, τότε και μόνο τότε ο χρήστης μπορεί να επιλέξει την ομάδα του.

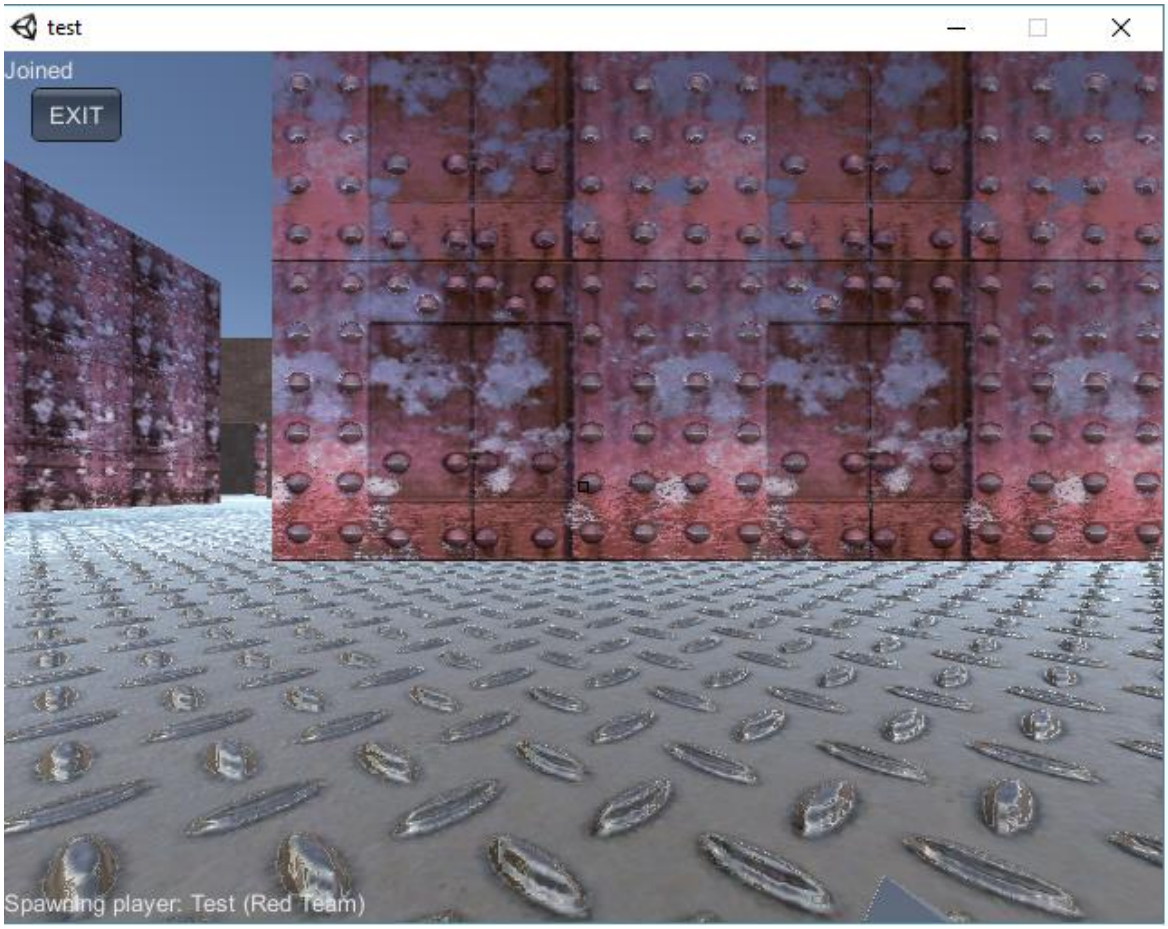
Σε περίπτωση που δεν πραγματοποιηθεί η σύνδεση τότε το πρόγραμμα τερματίζεται και η διαδικασία σύνδεσης ξεκινάει από την αρχή την επόμενη φορά που ο χρήστης θα δοκιμάσει να επιλέξει το μενού “Multiplayer” αφού εκτελέσει την εφαρμογή.




**Εικόνα 4.2:** Μενού επιλογής ομάδας.

Μόλις λοιπόν ο χρήστης επιλέξει την ομάδα του εμφανίζεται στο πεδίο μάχης με το χρώμα της αντίστοιχης ομάδας που επέλεξε. Το σύστημα φροντίζει να ενημερώσει όλους τους υπόλοιπους παίχτες με ένα μήνυμα συστήματος στο κάτω αριστερό μέρος της οθόνης, εμφανίζοντας κάθε φορά το όνομα και την ομάδα του

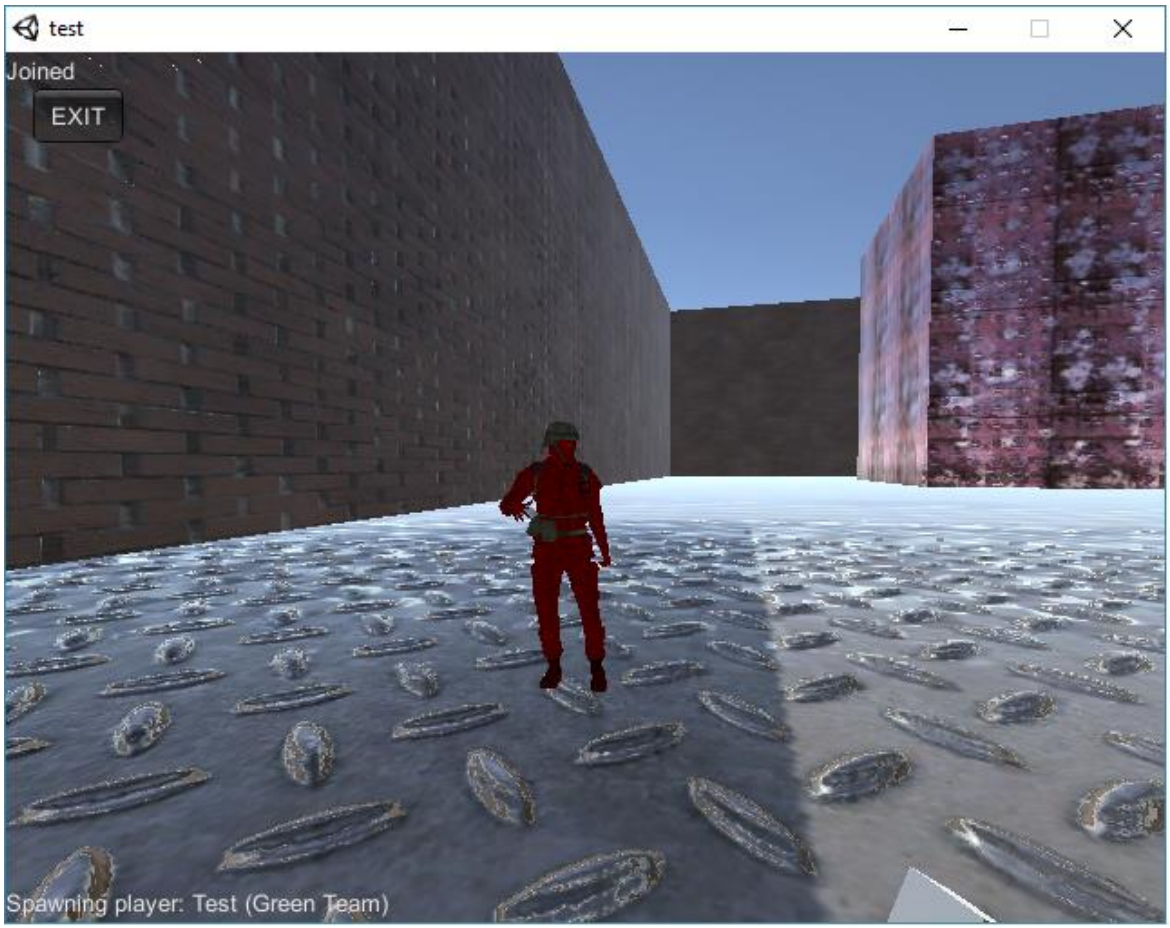
παίκτη που συνδέθηκε. Την ίδια στιγμή ενεργοποιούνται διάφορα Scripts που έχουν προγραμματιστεί για τον χειρισμό του παίκτη. Έτσι ο χρήστης χρησιμοποιώντας τα κουμπιά W, S, A, D, κινεί τον παίκτη του μπρος, πίσω, αριστερά και δεξιά αντίστοιχα ενώ χρησιμοποιώντας το ποντίκι αλλάζει την οπτική του και τον στόχο του ελεύθερα προς όλες τις κατευθύνσεις.




**Εικόνα 4.3:** Αρχική θέση του παίκτη.

Με αυτό τον τρόπο λοιπόν το παιχνίδι μας ξεκινά. Στόχος του παίκτη είναι να αρχίσει να εξερευνεί την πίστα προσπαθώντας να εντοπίσει τους παίκτες της αντίπαλης ομάδας και να τους εξοντώσει με το όπλο του πριν του τελειώσουν οι πόντοι ζωής του. Κάθε παίκτης έχει 100 πόντους ζωής και κάθε ακτίνα/σφαίρα κάνει 25 πόντους ζωής ζημιά. Αυτό σημαίνει ότι ο παίκτης πρέπει να πετύχει 4 φορές τον αντίπαλό του για να τον εξοντώσει. Ο παίκτης συνεχίζει και ψάχνει συνεχώς την πίστα μέχρι να εντοπίσει τον αντίπαλο του και αν τον εξοντώσει πριν

του τελειώσουν οι πόντοι ζωής του τότε συνεχίζει να ψάχνει τον επόμενο αντίπαλο του.




**Εικόνα 4.4:** Μάχη με αντίπαλο.

Στην περίπτωση που ο παίκτης εξοντωθεί πρώτος από τον αντίπαλό του τότε μετά από 5 δευτερόλεπτα εμφανίζεται ξανά στην αρχική θέση που εμφανίστηκε όταν συνδέθηκε. Στο ίδιο σημείο, δηλαδή, που εμφανίζονται όλοι οι παίκτες που ανήκουν στην ομάδα του. Το παιχνίδι δεν έχει τέλος εκτός κι αν ο παίκτης επιλέξει, να αποσυνδεθεί από αυτό. Από την στιγμή που εκτελείται η εφαρμογή αλλά και κατά την διάρκεια της μάχης, ο παίκτης μπορεί οποιαδήποτε στιγμή να επιλέξει το κουμπί “EXIT” που βρίσκεται επάνω αριστερά στην οθόνη έτσι ώστε να αποσυνδεθεί από το παιχνίδι και να τερματίσει την εφαρμογή.

## 4.2 Δομή Προγράμματος

Για τη δημιουργία της εφαρμογής αυτής χρησιμοποιήθηκαν διάφορα Scripts. Το καθένα από αυτά έχει την δική του λειτουργικότητα και όλα μαζί συνολικά αποτελούν την επιθυμητή λειτουργία της εφαρμογής μας. Παρακάτω θα δούμε αναλυτικά κάθε Script και πως συμβάλλει το καθένα στον προγραμματισμό της εφαρμογής, με την σειρά που παρουσιάζονται και στο "**Διάγραμμα ροής 2**" που βρίσκεται στο τέλος του κεφαλαίου.

Όπως είδαμε και στο προηγούμενο κεφάλαιο, με την εκτέλεση της εφαρμογής, εμφανίζεται το αρχικό μενού με τις δυνατές επιλογές που μπορεί να κάνει ο χρήστης. Για την εμφάνιση του μενού αυτού είναι υπεύθυνο το Script "**Network Manager**". Το Script αυτό περιέχει όλα τα μενού που εμφανίζονται στον χρήστη μέχρι την εκκίνηση του παιχνιδιού. Επίσης είναι υπεύθυνο για την συνδεσιμότητα μεταξύ χρήστη και Server (διακομιστής). Όταν δηλαδή ο χρήστης εισάγει το ψευδώνυμό του και επιλέξει το μενού "multiplayer", τότε μέσω του Script αυτού γίνεται η σύνδεση με τον Server (διακομιστής) και με την ολοκλήρωσή της ο χρήστης καλείται να επιλέξει ομάδα, εμφανίζεται δηλαδή το δεύτερο μενού.

Σε αυτό το μενού ο χρήστης, αφού επιλέξει ομάδα, εμφανίζεται μέσα στην πίστα στην μεριά της ομάδας που επέλεξε και με το αντίστοιχο χρώμα. Για την εμφάνιση στην σωστή μεριά της πίστας και με το σωστό χρώμα χρησιμοποιήθηκαν δύο ακόμα Scripts. Το πρώτο είναι το "**Spots**", το οποίο με την χρήση μιας ακεραίας μεταβλητής (1 για την κόκκινη ομάδα και 2 για την πράσινη ομάδα) εμφανίζει τον χρήστη στην αντίστοιχη μεριά. Το δεύτερο είναι το "**TeamMember**" το οποίο έχει παρόμοια χρήση με το προηγούμενο Script. Χρησιμοποιείται, δηλαδή, μια ακέραια μεταβλητή και ανάλογα την τιμή της τότε η στολή του παίκτη παίρνει και το αντίστοιχο χρώμα (1 για την κόκκινη ομάδα και 2 για την πράσινη ομάδα).

Μετά τις προσωπικές επιλογές του, ο χρήστης εμφανίζεται στην πίστα στην μεριά της ομάδας που επέλεξε και με το αντίστοιχο χρώμα. Την ίδια στιγμή ενεργοποιούνται τέσσερα Scripts τα οποία εμπεριέχουν κώδικα που αφορά τις ιδιότητες του παίκτη καθώς και τον συγχρονισμό του με τον Server (διακομιστής) και τους υπόλοιπους παίκτες. Τα βήματα αυτά, καθώς και τα αντίστοιχα Script

που χρησιμοποιήθηκαν, παρουσιάζονται στο διάγραμμα ροής από την “ΣΥΝΔΕΣΗ” έως και την “ΕΚΚΙΝΗΣΗ”. Παρακάτω παρουσιάζονται τα τέσσερα Scripts που προαναφέρθηκαν.

Το πρώτο Script που αναπτύξαμε σε αυτό το σημείο ήταν το “**PlayerMovement**” το οποίο περιέχει κώδικα που καθορίζει τις κινήσεις που μπορεί να κάνει ο παίκτης καθώς και το animation (κινούμενο σχέδιο) κάθε κίνησης αντίστοιχα. Δηλαδή χρησιμοποιείται για την μετακίνηση του παίκτη στην πίστα, την οπτική του και τον τρόπο στόχευσης.

Έπειτα αναπτύξαμε το Script “**Health**” το οποίο είναι υπεύθυνο για τους πόντους ζωής του παίκτη καθώς και για το τι γίνεται όταν αυτοί οι πόντοι μηδενιστούν. Κάθε φορά που ο παίκτης εμφανίζεται στην πίστα έχει 100 πόντους ζωής και κάθε φορά που δέχεται χτύπημα αυτοί μειώνονται. Θα δούμε στην συνέχεια τι γίνεται όταν αυτοί μηδενιστούν.

Στη συνέχεια αναπτύξαμε το Script “**WeaponData**” το οποίο περιέχει τα χαρακτηριστικά του όπλου που χρησιμοποιεί ο παίκτης. Δηλαδή πόσο γρήγορα πυροβολεί και πόση ζημιά κάνει κάθε χτύπημα στον αντίπαλο. Το όπλο είναι συγχρονισμένο με την οπτική του παίκτη έτσι ώστε να στοχεύει ταυτόχρονα ότι βλέπει και κάνει ζημιά ίση με 25 πόντους ζωής.

Τέλος, κατά την εκκίνηση ενεργοποιείται και το Script “**NetworkCharacter**” το οποίο περιέχει κώδικα για τον συγχρονισμό των παικτών μεταξύ τους. Όταν δηλαδή κάποιος παίκτης εκτελεί μια ενέργεια, όποιος παίκτης έχει οπτική επαφή μαζί του, βλέπει την ενέργεια αυτή την ίδια στιγμή που γίνεται.

Βλέπουμε λοιπόν ότι κατά την εκκίνηση του παιχνιδιού ενεργοποιούνται τα τέσσερα αυτά Script που είναι σημαντικά για την λειτουργικότητα του παίκτη και για την αλληλεπίδρασή του με το περιβάλλον στο οποίο βρίσκεται. Με αυτό τον τρόπο το παιχνίδι είναι έτοιμο να ξεκινήσει. Ο παίκτης ξεκινάει την εξερεύνηση της πίστας ψάχνοντας τους αντιπάλους του. Ερχόμαστε λοιπόν στο σημείο που ο παίκτης εντοπίζει τον εχθρό και καλείται να πυροβολήσει. Η παράγραφος αυτή αναφέρεται στην συνθήκη “ΕΝΤΟΠΙΣΜΟΣ ΕΧΘΡΟΥ” του διαγράμματος ροής.

Σε αυτό το σημείο ενεργοποιείται το Script “**PlayerShooting**” το οποίο είναι υπεύθυνο για όλα τα χαρακτηριστικά του πυροβολισμού καθώς και για το τι γίνεται

σε κάθε πυροβολισμό. Όπως προαναφέρθηκε κάθε πυροβολισμός κάνει στον αντίπαλο ζημιά ίση με 25 πόντους ζωής. Αυτό το Script λοιπόν κάθε φορά που ο παίκτης πετυχαίνει τον αντίπαλο αφαιρεί 25 πόντους από τους πόντους ζωής του. Ταυτόχρονα με κάθε πυροβολισμό ενεργοποιείται το Script “**FXManager**” το οποίο εμπεριέχει κώδικα για την εμφάνιση της σφαίρας/ακτίνας και για τον ήχο κάθε πυροβολισμού. Σε αυτό το σημείο βρισκόμαστε στο βήμα “ΠΥΡΟΒΟΛΙΣΜΟΣ” του διαγράμματος ροής.

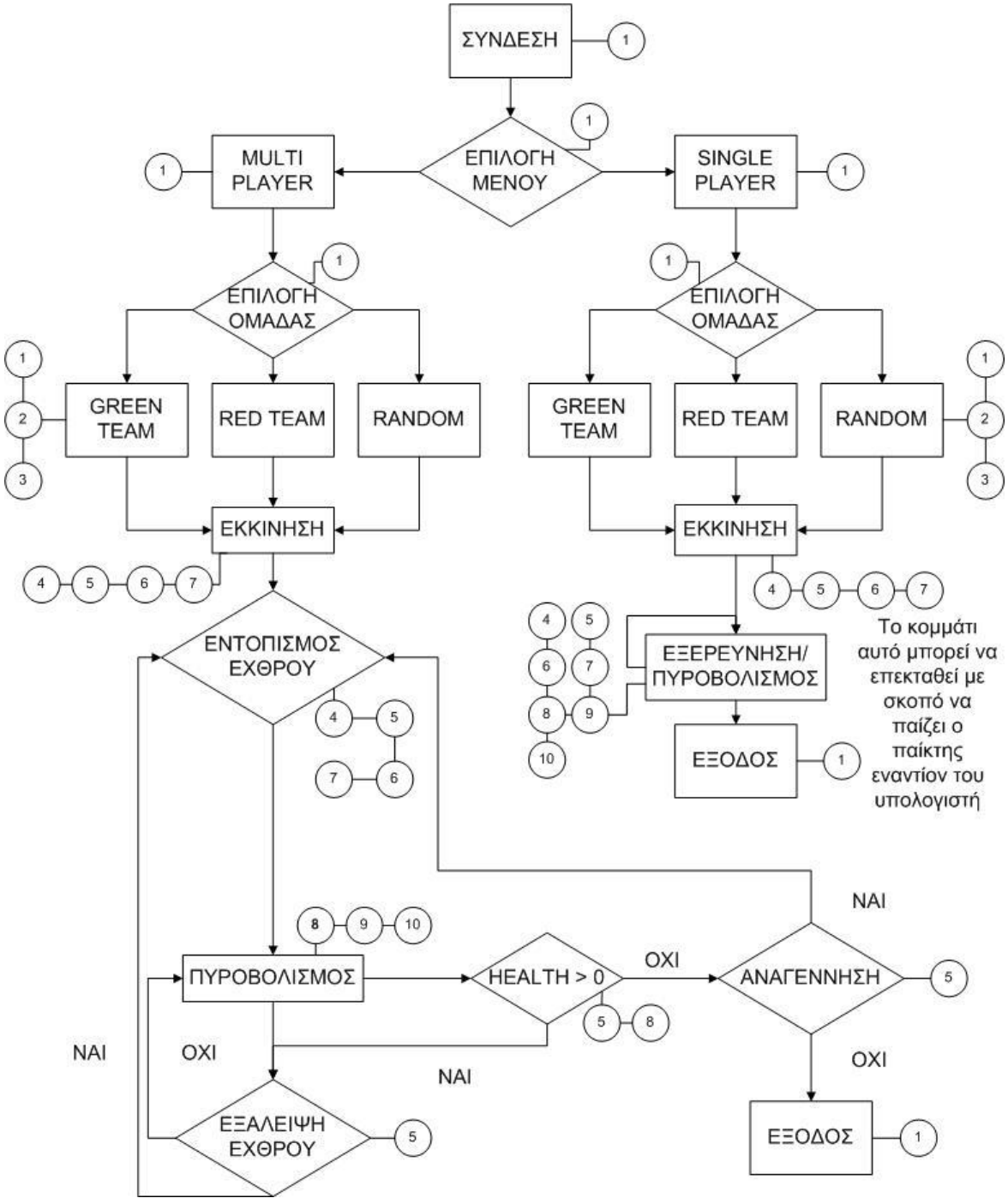
Αν ο παίκτης σκοτώσει τον αντίπαλό του, αν δηλαδή οι πόντοι ζωής του φτάσουν στο μηδέν πριν από τους δικούς του τότε ο εχθρός εξαλείφεται και ο παίκτης συνεχίζει την εξερεύνηση της πίστας, όπως φαίνεται και στο διάγραμμα ροής στην συνθήκη “ΕΞΑΛΛΕΙΞΗ ΕΧΘΡΟΥ”. Σε περίπτωση όμως που δεν καταφέρει να νικήσει την μάχη και τελειώσουν πρώτα οι δικοί του πόντοι ζωής τότε ο παίκτης καλείται να επιλέξει αν θέλει να αναγεννηθεί ή να βγει τελείως από το παιχνίδι, όπως φαίνεται και στο διάγραμμα ροής στην συνθήκη “HEALTH > 0”. Αυτό τον ρόλο τον αναλαμβάνουν τα Script “**Health**” και “**PlayerShooting**” τα οποία περιέχουν συναρτήσεις που ελέγχουν τους πόντους ζωής του παίκτη σε κάθε πυροβολισμό. Έτσι αν οι πόντοι ζωής του παίκτη μηδενιστούν τότε ενεργοποιείται η συνάρτηση αναγέννησης, όπου ο παίκτης εμφανίζεται ξανά στην αρχική του θέση έπειτα από ένα μικρό χρονικό διάστημα, και το παιχνίδι συνεχίζεται. Διαφορετικά ο παίκτης έχει την δυνατότητα να βγει από το παιχνίδι τερματίζοντας την εφαρμογή, όπως φαίνεται στο διάγραμμα ροής στην συνθήκη “ΑΝΑΓΕΝΝΗΣΗ”.

Τέλος ένα ακόμα Script που αναπτύξαμε είναι το “**SelfDestruct**” το οποίο χρησιμοποιείται για την καταστροφή αντικειμένων μέσα στην πίστα. Στην εφαρμογή συγκεκριμένα χρησιμοποιείται για την απαλοιφή της σφαίρας/ακτίνας του όπλου. Αν δεν είχαμε αυτό το Script τότε μετά από κάθε πυροβολισμό η σφαίρα/ακτίνα θα έμενε ορατή στην έκταση της πίστας.

Οι αριθμοί που βρίσκονται δίπλα στα βήματα του διαγράμματος ροής αντιστοιχούν στα Script του παρακάτω πίνακα και δηλώνουν ποια από αυτά χρησιμοποιούνται σε κάθε βήμα.

| **Α/Α** | **Ονόματα Script** |
|---|---|
| ① | Network Manager |
| ② | Spots |
| ③ | TeamMember |
| ④ | PlayerMovement |
| ⑤ | Health |
| ⑥ | WeaponData |
| ⑦ | NetworkCharacter |
| ⑧ | PlayerShooting |
| ⑨ | FXManager |
| ⑩ | SelfDestruct |

**Πίνακας 4.2.1:** Αρίθμηση των Script




**Διάγραμμα Ροής 1:** Αναλυτική Λειτουργία Εφαρμογής.

Οι αριθμοί που βρίσκονται σε κύκλο παρουσιάζονται στον παραπάνω πίνακα και δηλώνουν ποια από τα Script χρησιμοποιούνται σε κάθε βήμα.

# ΚΕΦΑΛΑΙΟ 5

## ΠΡΟΟΠΤΙΚΕΣ

Παρόλα τα προβλήματα που αντιμετωπίσαμε η εφαρμογή ολοκληρώθηκε και το παιχνίδι είναι έτοιμο. Το παιχνίδι αυτό όμως έχει πολλά περιθώρια βελτίωσης, καθώς τα γραφικά μπορούν να αναπτυχθούν περαιτέρω και το πλήθος των παικτών μπορεί να μεγαλώσει με την αγορά κάποιου server (διακομιστής). Επίσης μπορεί να αναπτυχθεί ένα μενού όπου κάθε χρήστης μπορεί να παίζει εναντίον του Η/Υ, εναντίον δηλαδή παικτών τεχνητής νοημοσύνης (Artificial Intelligence Bots). Μία ακόμα προσθήκη που μπορεί να γίνει είναι η ποικιλία εξοπλισμού των παικτών καθώς επίσης και η ποικιλία των πιστών. Δηλαδή κάθε φορά που κάποιος χρήστης συνδέεται στο παιχνίδι να έχει την δυνατότητα επιλογής στολής, πυρομαχικών και πίστας. Επίσης, όπως στα περισσότερα διαδικτυακά βιντεοπαιχνίδια, μπορεί να προστεθεί ένας πίνακας αποτελεσμάτων που θα εμπεριέχει το σκορ κάθε παίκτη. Τέλος θα μπορούσε να αναπτυχθεί ένας τρόπος επικοινωνίας μεταξύ των παικτών που ανήκουν στην ίδια ομάδα. Όλα αυτά δεν πραγματοποιήθηκαν σε αυτή την πτυχιακή λόγω έλλειψης γνώσεων και οικονομικού περιορισμού. Η εφαρμογή βρίσκεται στην πιο απλή μορφή της και, όπως προαναφέρθηκε, τα περιθώρια βελτίωσης της είναι πολλά.

# ΠΑΡΑΡΤΗΜΑ ΚΩΔΙΚΑ

### 1 WeaponData (Script)

```
using UnityEngine;

using System.Collections;


public class WeaponData : MonoBehaviour {

	public float fireRate = 0.5f;

	public float damage = 25f;

	// Use this for initialization

	void Start () {


	}
```

Σε αυτό το script δηλώνουμε τα χαρακτηριστικά του όπλου που χρησιμοποιούν οι χρήστες. Τα χαρακτηριστικά είναι:

1) πόσο γρήγορα πυροβολεί (‘fireRate’).
2) πόση ζημιά κάνει το χτύπημα του στον αντίπαλο (‘damage’).

**2 SelfDestruct (Script)**

```
using UnityEngine;
using System.Collections;


public class SelfDestruct : MonoBehaviour {


	public float selfDestructTime = 1.0f;


	void Update () {
		selfDestructTime -= Time.deltaTime;


		if (selfDestructTime <= 0) {


			PhotonView pv = GetComponent<PhotonView>();
			if(pv != null && pv.instantiationId!=0) {
				PhotonNetwork.Destroy (gameObject);
			}
			else{
				Destroy(gameObject);
			}
		}
	}
}
```

Η λειτουργία του Script αυτού είναι να καταστρέφει αντικείμενα (‘Destroy(gameObject);’) μετά από περίπου 1 δευτερόλεπτο (‘selfDestructTime -= Time.deltaTime’). Στην συγκεκριμένη εφαρμογή το χρησιμοποιήσαμε για να εξαφανίζουμε την ακτίνα/σφαίρα του όπλου αφού γίνει πυροβολισμός, διαφορετικά η ακτίνα/σφαίρα θα έμενε ορατή συνεχώς.

**3 Spots(Scripts)**

**Πρώτο Script (SpawnSpotRed)**

```
using UnityEngine;
using System.Collections;

public class SpawnSpotRed : MonoBehaviour {

	public int teamId=1;
}
```

**Δεύτερο Script (SpawnSpotGreen)**

```
using UnityEngine;
using System.Collections;

public class SpawnSpotGreen : MonoBehaviour {

	public int teamId=2;


}
```

Και τα δύο Scripts έχουν ίδια λειτουργία τα καθένα όμως λειτουργεί ξεχωριστά για κάθε ομάδα. Ανάλογα με την επιλογή ομάδας του χρήστη η μεταβλητή ‘teamId’ παίρνει την αντίστοιχη τιμή, 1 για την κόκκινη ομάδα και 2 για την πράσινη. Η τιμή αυτή χρησιμοποιείται σε Script που θα δούμε παρακάτω για να γίνει η εμφάνιση του παίκτη στην πίστα στην σωστή μεριά, στην μεριά δηλαδή που εμφανίζονται οι παίκτες της ομάδας του.

**4 TeamMember(Script)**

```
using UnityEngine;
using System.Collections;
public class TeamMember : MonoBehaviour {
	int _teamID = 0;
	public int teamID {
		get { return _teamID;}
	}
	[PunRPC]
	void SetTeamID(int id) {
		_teamID = id;
		SkinnedMeshRenderer mySkin =
		this.transform.GetComponentInChildren<SkinnedMeshRenderer> ();

		if (mySkin == null) {
			Debug.LogError("Could not find a SkinnedMeshRenderer");
		}
		if (_teamID == 1)
			mySkin.material.color = Color.red;
		if (_teamID == 2)
			mySkin.material.color = Color.green;
	}
}
```

Ο σκοπός αυτού του script είναι να χρωματίζει την στολή του κάθε παίκτη ανάλογα με το ποια ομάδα έχει επιλέξει. Για παράδειγμα, αν ο παίκτης επιλέξει την κόκκινη ομάδα τότε η στολή του θα γίνει κόκκινη, ενώ αν έχει επιλέξει την πράσινη ομάδα η στολή του γίνεται πράσινη.

Η συνάρτηση ‘SetTeamID’, η οποία εκτελείται ταυτόχρονα σε όλους τους χρήστες, ορίζει έναν ακέραιο σε κάθε ομάδα και αναλόγως με το ποια ομάδα θα επιλέξει ο χρήστης τότε δίνεται και το αντίστοιχο χρώμα.

**5 FXManager(Script)**

```
using UnityEngine;
using System.Collections;
public class FXManager : MonoBehaviour {
	public GameObject sniperBulletFXPrefab;
	[PunRPC]
	void SniperBulletFX(Vector3 startPos,Vector3 endPos){
		if (sniperBulletFXPrefab != null) {
			GameObject sniperFX = (GameObject)Instantiate
			(sniperBulletFXPrefab, startPos, Quaternion.LookRotation
			(endPos-startPos));
			LineRenderer lr = sniperFX.transform.Find
			("LineFX").GetComponent<LineRenderer> ();
			if (lr != null) {
				lr.SetPosition (0, startPos);
				lr.SetPosition (1, endPos);
			} else {
				Debug.LogError ("sniperBulletFXPrefab's linerenderer
				is missing.");
			}
		}
		else {
			Debug.LogError ("sniperBulletFXPrefab is missing!");
		}
```

```
        }
}
```

Αυτό το Script είναι υπεύθυνο για την δημιουργία της ακτίνας που βγαίνει από το όπλο κάθε φορά που πυροβολεί ο χρήστης. Η αγκύλη [PunRPC] (Remote Procedure Call) που βρίσκεται πάνω από την συνάρτηση ‘SniperBulletFXPrefab’ είναι ένα πρωτόκολλο το οποίο επιτρέπει στην συνάρτηση αυτή να εκτελείται ξεχωριστά σε όλους τους υπολογιστές που έχουν συνδεθεί στον ίδιο Server (διακομιστής).

**6 Health(Script)**

```
using UnityEngine;
using System.Collections;


public class Health : MonoBehaviour {
	public float hitPoints = 100f;
	float currentHitPoints;
	bool showGUI=false;


	// Use this for initialization
	void Start () {

		currentHitPoints = hitPoints;
	}


	[PunRPC]
	public void TakeDamage(float amt) {
		currentHitPoints -= amt;
		if (currentHitPoints <= 0) {
			Die();
		}
	}
```

```
        void Die() {


                if (GetComponent<PhotonView> ().instantiationId == 0) {
                        Destroy(gameObject);
                }
                else {
                        if(GetComponent<PhotonView> ().isMine){
                                if(gameObject.tag == "Player"){


                                        NetworkManager nm =
                                        GameObject.FindObjectOfType<Network
                                        Manager>();
                                        nm.standbyCamera.SetActive(true);
                                        nm.respawnTimer = 5f;


                                }
                        }
                        PhotonNetwork.Destroy(gameObject);
                }
        }
}
```

Το Script Health είναι υπεύθυνο για τους πόντους ζωής κάθε παίκτη και για το τι γίνεται όταν αυτοί οι πόντοι φτάσουν στο μηδέν (όταν δηλαδή ο παίκτης πεθάνει).

Αρχικά, αφού γίνουν οι κατάλληλες δηλώσεις των μεταβλητών που θα χρησιμοποιηθούν, με την συνάρτηση ‘Start’, αρχικοποιούμε τους πόντους ζωής κάθε παίκτη. Στη συνέχεια η, η συνάρτηση ‘TakeDamage’, μειώνει τους πόντους αυτούς κάθε φορά που ο παίκτης δέχεται ζημιά και αν φτάσουν στο μηδέν τότε καλείται η συνάρτηση ‘Die’.

Η συνάρτηση ‘Die’ καταστρέφει το ‘αντικείμενο’ που πυροβολήθηκε και έχασε όλους τους πόντους ζωής του. Στη συνέχεια, αφού γίνει έλεγχος αν το ‘αντικείμενο’ που καταστράφηκε είναι παίκτης, ενεργοποιεί ξανά την κάμερά του και τον επαναφέρει στην πίστα στο σημείο που εμφανίζονται όλοι οι παίκτες της ομάδας του.

**7 PlayerMovement(Script)**

```
using UnityEngine;
using System.Collections;


public class PlayerMovement : MonoBehaviour {
	public float speed = 10f;
	Vector3 direction = Vector3.zero;
	CharacterController cc;
	Animator anim;


	void Start () {
		cc = GetComponent<CharacterController>();
		anim = GetComponent<Animator>();
	}
	void Update () {
		direction = transform.rotation * new Vector3(
		Input.GetAxis("Horizontal") , 0, Input.GetAxis("Vertical") );


		if(direction.magnitude > 1f) {
				direction = direction.normalized;
		}


		anim.SetFloat("Speed", direction.magnitude);
```

```
        AdjustAimAngle();
    }

    void AdjustAimAngle() {
        Camera myCamera = this.GetComponentInChildren<Camera>();

        if(myCamera==null) {
            Debug.LogError("Why doesn't my character have a camera?
            return;
        }

        float AimAngle = 0;

        if(myCamera.transform.rotation.eulerAngles.x <= 90f) {

            AimAngle = -myCamera.transform.rotation.eulerAngles.x;
        }
        else {
            AimAngle = 360 -
            myCamera.transform.rotation.eulerAngles.x;
        }
    }
```

```
        void FixedUpdate () {


                Vector3 dist = direction * speed * Time.deltaTime;


                cc.Move( dist );

        }
}
```

Το Script ‘PLayerMovement’ είναι υπεύθυνο για τον χειρισμό των κινήσεων του παίκτη καθώς και για την ενεργοποίηση των animation (κινούμενων σχεδίων) κάθε κίνησης αντίστοιχα.

Ξεκινώντας αρχικοποιούμε τις μεταβλητές: ‘speed’, ‘direction’, ‘verticalVelocity’, ‘cc’ και ‘anim’. Η μεταβλητή ‘speed’ ορίζει την ταχύτητα κίνησης και η μεταβλητή ‘direction’ ορίζει την κίνηση του παίκτη στον άξονα χ. Οι μεταβλητές ‘cc’ και ‘anim’ δεν έχουν αρχική τιμή αλλά χρησιμοποιούνται για τον χειρισμό του παίκτη και του animation (κινούμενου σχεδίου) αντίστοιχα. Στην συνέχεια στην συνάρτηση ‘Start’ αρχικοποιούμε τις μεταβλητές ‘cc’ και ‘anim’. Η συνάρτηση ‘Update’, η οποία καλείται μια φορά ανά πλαίσιο (per frame), αποθηκεύει στην μεταβλητή ‘direction’ τις κινήσεις του παίκτη, ενεργοποιεί το animation (κινούμενο σχέδιο) της κίνησης και τέλος καλεί την συνάρτηση ‘AdjustAimAngle’ η οποία αναλύεται παρακάτω.

Η συνάρτηση ‘AdjustAimAngle’ είναι υπεύθυνη για την κίνηση της κάμερας του παίκτη ανάλογα με το που κοιτάει ο παίκτης κάθε στιγμή. Αυτό το πετυχαίνουμε με την μεταβλητή ‘AimAngle’, αφού αρχικοποιήσουμε την μεταβλητή ‘myCamera’ με την κάμερα του παιχνιδιού.

Τέλος έχουμε την συνάρτηση ‘FixedUpdate’ η οποία εκτελείται σε κάθε ‘κύκλο φυσικής κατάστασης’ (δηλαδή κάθε φορά που κινείται ο παίκτης και χρειαζόμαστε τους νόμους της φυσικής). Αποθηκεύουμε την συνολική κίνηση του παίκτη στην μεταβλητή ‘dist’ και με την συνάρτηση ‘Move()’ την ενεργοποιούμε. Η ιδιότητα ‘deltaTime’ χρησιμοποιείται για να έχουμε πιο λεπτομερή αποτελέσματα στην ‘απαλότητα’ (smoothness) της κίνησης.

**8 NetworkCharacter (Script)**

```
using UnityEngine;
using System.Collections;

public class NetworkCharacter : Photon.MonoBehaviour {

	Vector3 realPosition = Vector3.zero;
	Quaternion realRotation=Quaternion.identity;

	Animator anim;

	bool gotFirstUpdate = false;

	float lastUpdateTime;

	void Start () {
		CacheComponents ();
	}
	void CacheComponents() {
		if (anim == null) {
			anim = GetComponent<Animator> ();
			if (anim == null) {
				Debug.LogError ("no animator component put");
```

```
                }
        }
}


void Update () {

        if (photonView.isMine) {

        } else {
                transform.position=Vector3.Lerp
                (transform.position,realPosition,0.1f);
                transform.rotation=Quaternion.Lerp
                (transform.rotation,realRotation,0.1f);
        }

}

public void OnPhotonSerializeView(PhotonStream stream,
PhotonMessageInfo info) {

        CacheComponents();
        if (stream.isWriting) {
```

```
                stream.SendNext (transform.position);
                stream.SendNext (transform.rotation);
                stream.SendNext (anim.GetFloat("Speed"));









        } else {

                realPosition = (Vector3)stream.ReceiveNext();
                realRotation = (Quaternion)stream.ReceiveNext();
                anim.SetFloat("Speed", (float)stream.ReceiveNext());

                if(gotFirstUpdate == false) {
                        transform.position = realPosition;
                        transform.rotation = realRotation;
                        gotFirstUpdate = true;
                }
        }
    }
}
```

Το Script αυτό είναι υπεύθυνο για τον συγχρονισμό των χρηστών στο διαδίκτυο. Με άλλα λόγια όταν ένας παίκτης κάνει κάποια ενέργεια στο παιχνίδι (κίνηση, πυροβολισμός κλπ) να το βλέπουν ταυτόχρονα όλοι οι παίκτες την ώρα που γίνεται.

Αρχικά γίνονται οι αρχικοποιήσεις των μεταλητών: ‘realPosition’ και ‘realRotation’ που χρησιμοποιούνται για τις συντεταγμένες του παίκτη, ‘anim’ που χρησιμοποιείται για το αντίστοιχο animation (κινούμενο σχέδιο) του παίκτη και ‘gotFirstUpdate’ που χρησιμοποιείται για τον έλεγχο λήψης ορισμένων δεδομένων που θα εξηγηθούν παρακάτω. Στη συνέχεια ξεκινάει το Script με την συνάρτηση ‘CacheComponents’ η οποία ελέγχει αν έχουν οριστεί τα κατάλληλα animation (κινούμενα σχέδια) αλλιώς τα ορίζει στην μεταβλητή ‘anim’.

Παρακάτω έχουμε την συνάρτηση ‘Update’ η οποία ελέγχει αν ο παίκτης έχει συντεταγμένες θέσης και περιστροφής αλλιώς του ορίζει καινούριες.

Τέλος έχουμε την συνάρτηση ‘OnPhotonSerializeView’ η οποία είναι υπεύθυνη για τον συγχρονισμό μεταξύ των παικτών και του Server (διακομιστής). Ξεκινώντας καλούμε την συνάρτηση ‘CacheComponents’ που εξηγήθηκε παραπάνω. Έπειτα οι παίκτες στέλνουν με την εντολή ‘stream.SendNext()’ τις συντεταγμένες και τα animation τους και λαμβάνουν με την εντολή ‘stream.RecieveNext()’ τις συντεταγμένες και τα animation των υπόλοιπων παικτών. Τέλος επειδή μπορεί να συνδεθεί κάποιος οποιαδήποτε στιγμή λαμβάνει την πιο πρόσφατη ενημέρωση για τις συντεταγμένες των παικτών.

**9 NetworkManager(Script)**

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

public class NetworkManager : MonoBehaviour {

    public bool offline = false;
    bool connecting = false;
    public GameObject standbyCamera;
    List<string> chatMessages;
    int maxChatMessages = 5;
    public float respawnTimer = 0;
    bool pickedTeam = false;
    int teamID = 0;

    // Αρχικοποίηση
    void Start () {

        PhotonNetwork.player.name = PlayerPrefs.GetString("Username",
        "Player");
        chatMessages = new List<string> ();

    }
```

```
void OnDestroy() {
        PlayerPrefs.SetString ("Username", PhotonNetwork.player.name);
}



public void AddChatMessage(string m) {
        GetComponent<PhotonView> ().RPC ("AddChatMessage_RPC",
        PhotonTargets.All, m);
}



[PunRPC]

public void AddChatMessage_RPC(string m) {
        while (chatMessages.Count >= maxChatMessages) {
                chatMessages.RemoveAt(0);
        }
        chatMessages.Add(m);
}



void Connect() {
```

```
        PhotonNetwork.ConnectUsingSettings ("shooting game v1.0");
    }
    void OnGUI() {
        if (GUI.Button (new Rect (15, 20, 50, 30),"EXIT")) {
            Application.Quit ();
        }
        GUILayout.Label (PhotonNetwork.connectionStateDetailed.ToString
        ());
        if (PhotonNetwork.connected == false && connecting == false) {
            GUILayout.BeginArea(new Rect(0, 0, Screen.width,
            Screen.height));
            GUILayout.BeginHorizontal();
            GUILayout.FlexibleSpace();
            GUILayout.BeginVertical();
            GUILayout.FlexibleSpace();

            GUILayout.BeginHorizontal();
            GUILayout.Label("Username: ");
            PhotonNetwork.player.name =
            GUILayout.TextField(PhotonNetwork.player.name);
            GUILayout.EndHorizontal();

            if (GUILayout.Button ("Single Player")) {
                connecting = true;
```

```
                PhotonNetwork.offlineMode = true;
                OnJoinedLobby ();
            }
            if (GUILayout.Button ("Multiplayer")) {
                connecting = true;
                Connect ();
            }

            GUILayout.FlexibleSpace();
            GUILayout.EndVertical();
            GUILayout.FlexibleSpace();
            GUILayout.EndHorizontal();
            GUILayout.EndArea();
        }
        if (PhotonNetwork.connected == true && connecting == false) {

            if(pickedTeam) {

                GUILayout.BeginArea(new Rect(0,0, Screen.width,
                Screen.height));
                GUILayout.BeginVertical();
                GUILayout.FlexibleSpace();
```

```
                foreach(string msg in chatMessages) {
                        GUILayout.Label (msg);
                }

                GUILayout.EndVertical();
                GUILayout.EndArea();
        }
        else {
                GUILayout.BeginArea(new Rect(0, 0, Screen.width,
                Screen.height));
                GUILayout.BeginHorizontal();
                GUILayout.FlexibleSpace();
                GUILayout.BeginVertical();
                GUILayout.FlexibleSpace();

                GUILayout.BeginHorizontal();
                GUILayout.Label("Username: ");
                PhotonNetwork.player.name =
                GUILayout.TextField(PhotonNetwork.player.name);
                GUILayout.EndHorizontal();

                if (GUILayout.Button ("Red Team")) {
                        SpawnMyPLayer (1);
```

```
                        }
                        if (GUILayout.Button ("Green Team")) {
                                SpawnMyPLayer (2);
                        }
                        if (GUILayout.Button ("Random")) {
                                SpawnMyPLayer (Random.Range(1,3));
                        }

                        GUILayout.FlexibleSpace();
                        GUILayout.EndVertical();
                        GUILayout.FlexibleSpace();
                        GUILayout.EndHorizontal();
                        GUILayout.EndArea();
                }
        }

}

void OnJoinedLobby() {
        PhotonNetwork.JoinRandomRoom ();
}

void OnPhotonRandomJoinFailed() {
```

```
        PhotonNetwork.CreateRoom (null);
    }

    void OnJoinedRoom() {

        connecting = false;
    }
    void SpawnMyPLayer(int teamID) {
        this.teamID = teamID;
        pickedTeam = true;
        if(teamID==1){
            AddChatMessage ("Spawning player: " +
            PhotonNetwork.player.name + " (Red Team)");
        }else{
            AddChatMessage ("Spawning player: " +
            PhotonNetwork.player.name + " (Green Team)");
        }
        if (teamID == 1) {
            SpawnSpotRed MySpot =
            GameObject.FindObjectOfType<SpawnSpotRed> ();
            GameObject myPlayerGO =
            (GameObject)PhotonNetwork.Instantiate ("PlayerController",
            MySpot.transform.position, MySpot.transform.rotation,0);
```

```
            ((MonoBehaviour)myPlayerGO.GetComponent("PlayerMove
            ment")).enabled = true;

            standbyCamera.SetActive (true);

            ((MonoBehaviour)myPlayerGO.

            GetComponent("PlayerShooting")).enabled = true;

            myPlayerGO.GetComponent<PhotonView>().RPC("SetTeamI
            D", PhotonTargets.AllBuffered, teamID);


                  myPlayerGO.transform.FindChild

            ("FirstPersonCharacter").gameObject.SetActive (true);

      }

      if(teamID == 2) {

            SpawnSpotGreen MySpot =
            GameObject.FindObjectOfType<SpawnSpotGreen> ();

            GameObject myPlayerGO =
            (GameObject)PhotonNetwork.Instantiate ("PlayerController",
            MySpot.transform.position, MySpot.transform.rotation,0);

            ((MonoBehaviour)myPlayerGO.GetComponent("PlayerMove
            ment")).enabled = true;

            standbyCamera.SetActive (true);

            ((MonoBehaviour)myPlayerGO.

            GetComponent("PlayerShooting")).enabled = true;

            myPlayerGO.GetComponent<PhotonView>().RPC("SetTeamI
            D", PhotonTargets.AllBuffered, teamID);


      myPlayerGO.transform.FindChild
```

```
                ("FirstPersonCharacter").gameObject.SetActive (true);
            }
        }
        void Update() {
            if (respawnTimer > 0) {
                respawnTimer -= Time.deltaTime;
                if(respawnTimer <= 0) {
                    SpawnMyPLayer(teamID);
                }
            }
        }
}
```

Η λειτουργία αυτού του Script είναι αρχικά να εμφανίζει το μενού με τις επιλογές του παίκτη (interface), στην συνέχεια να τον συνδέει με το παιχνίδι και τέλος να τον εμφανίζει στην πίστα, δίνοντας του την επιλογή να αποσυνδεθεί από το παιχνίδι οποιαδήποτε στιγμή θελήσει.

Πιο αναλυτικά στην πρώτη σελίδα γίνονται οι αρχικοποιήσεις των μεταβλητών και των μηνυμάτων του συστήματος. Η συνάρτηση ‘OnDestroy’ αποθηκεύει τα στοιχεία που έχει δώσει ο χρήστης έτσι ώστε να μην χρειάζεται να τα δίνει κάθε φορά. Στην συνέχεια η συνάρτηση ‘AddChatMessage_RPC’ είναι η συνάρτηση στην οποία αποθηκεύονται τα μηνύματα συστήματος και η συνάρτηση ‘AddChatMessage’ εμφανίζει αυτά τα μηνύματα. Η αγκύλη [PunRPC] (Remote Procedure Call) που βρίσκεται πάνω από την συνάρτηση ‘AddChatMessage_RPC’ είναι ένα πρωτόκολλο το οποίο επιτρέπει στην συνάρτηση αυτή να εκτελείται σε όλους τους υπολογιστές που έχουν συνδεθεί στον ίδιο Server (διακομιστής).

Οι επόμενες τρεις σελίδες περιέχουν τον κώδικα που δημιουργεί το μενού (interface) με τις επιλογές που μπορεί να κάνει ο χρήστης ως προς τον τρόπο σύνδεσης και την ομάδα, καθώς και το κουμπί εξόδου που μπορεί να το επιλέξει ο χρήστης οποιαδήποτε στιγμή.

Οι τρεις επόμενες συναρτήσεις είναι υπεύθυνες για την δημιουργία ενός εικονικού ‘δωματίου αναμονής’ όπου οι χρήστες περιμένουν αφού συνδεθούν με τον Server (διακομιστής) για να παίξουν στην συγκεκριμένη πίστα.

Η συνάρτηση ‘SpawnMyPLayer()’ είναι υπεύθυνη για να εμφανίζεται ο κάθε παίκτης στο σωστό σημείο της πίστας ενεργοποιώντας κάθε φορά τις κατάλληλες διεργασίες για την σωστή λειτουργία του χαρακτήρα. Τέλος η συνάρτηση ‘Update()’ καλείται μια φορά ανά πλαίσιο (per frame) έτσι ώστε οποιαδήποτε στιγμή κάποιος παίκτης εξοντωθεί να μπορεί να ξαναεμφανιστεί.

**10 PlayerShooting(Script)**

```
using UnityEngine;
using System.Collections;


public class PlayerShooting : MonoBehaviour {
	float cooldown = 0;
	FXManager fxManager;
	WeaponData weaponData;
	void Start(){
		fxManager = GameObject.FindObjectOfType<FXManager> ();
		if (fxManager == null) {

			Debug.LogError ("FXManager missing");
		}
	}

	void Update () {

		cooldown -= Time.deltaTime;

		if (Input.GetButton ("Fire1")) {

			Fire();
```

```
            }


        }


        void Fire() {
            if (weaponData == null) {
                weaponData =
                gameObject.GetComponentInChildren<WeaponData> ();
                if (weaponData == null) {
                    Debug.LogError("Didn't find any WeaponData in our
                    children");
                    return;
                }
            }
            if (cooldown > 0) {
                return;
            }
            Debug.Log ("Firing our gun!");
            Ray ray = new Ray (Camera.main.transform.position,
            Camera.main.transform.forward);
            Transform hitTransform;
            Vector3 hitPoint;
            hitTransform = FindClosestHitObject (ray, out hitPoint);
            if (hitTransform != null) {
```

```
Debug.Log ("We hit: " + hitTransform.name);
Health h = hitTransform.GetComponent<Health>();
while(h == null && hitTransform.parent) {
        hitTransform = hitTransform.parent;
        h = hitTransform.GetComponent<Health>();
}



if (h != null) {
        PhotonView pv = h.GetComponent<PhotonView>();
        if ( pv == null) {
                Debug.LogError("Freak out");
        }
        else{
                TeamMember tm =
                hitTransform.GetComponent<TeamMember>();
                TeamMember myTm =
                this.GetComponent<TeamMember>();
                if(tm==null || tm.teamID==0 ||
                myTm.teamID==null || myTm.teamID==0 ||
                tm.teamID != myTm.teamID ) {
                        h.GetComponent<PhotonView>().RPC("
                        TakeDamage",
                        PhotonTargets.AllBuffered,
                        weaponData.damage);
```

```
                        }
                    }
                }
                if(fxManager != null) {
                    DoGunFX(hitPoint);
                }
            }
            else {
                if(fxManager != null) {
                    hitPoint = Camera.main.transform.position +
                    (Camera.main.transform.forward*100f);
                    DoGunFX(hitPoint);
                }
            }
            cooldown = weaponData.fireRate;
        }


        void DoGunFX(Vector3 hitPoint) {

        fxManager.GetComponent<PhotonView>().RPC("SniperBulletFX",
        PhotonTargets.All, weaponData.transform.position, hitPoint);
```

```
        }
        Transform FindClosestHitObject(Ray ray, out Vector3 hitPoint){
                RaycastHit[] hits = Physics.RaycastAll (ray);
                Transform closestHit = null;
                float distance = 0;
                hitPoint = Vector3.zero;
                foreach(RaycastHit hit in hits) {
                        if(hit.transform != this.transform && (closestHit == null ||
                        hit.distance < distance) ) {
                                closestHit = hit.transform;
                                distance = hit.distance;
                                hitPoint = hit.point;
                        }
                }
                return closestHit;
        }
}
```

Αυτό το Script είναι υπεύθυνο για τον πυροβολισμό και για το τι συμβαίνει όταν πυροβολείται κάποιο αντικείμενο.

Αφού γίνει η δήλωση μεταβλητών ‘cooldown’, ‘fxManager’ και ‘weaponData’, με την συνάρτηση ‘Start’ αρχικοποιούμε την μεταβλητή ‘fxManager’ με τα αντικείμενα που αναφέρθηκαν στο Script ‘FXManager’. Στη συνέχεια, στην συνάρτηση ‘Update’, με την μεταβλητή ‘cooldown’ ορίζουμε την διαφορά χρόνου μεταξύ κάθε πυροβολισμού αφαιρώντας κάθε φορά τον χρόνο ‘deltaTime’ που είναι ο χρόνος που διήρκεσε το τελευταίο πλαίσιο (frame). Έπειτα, αν πατηθεί το αριστερό κλικ (“fire1”), καλείται η συνάρτηση ‘Fire’ που θα αναλυθεί παρακάτω.

Στη συνέχεια, με την συνάρτηση ‘Fire’ που είναι υπεύθυνη για τον πυροβολισμό, ορίζουμε στην μεταβλητή ‘weaponData’ τα χαρακτηριστικά του όπλου μας και δημιουργούμε την ακτίνα (ray). Αποθηκεύουμε στην μεταβλητή ‘hitTransform’ το πρώτο αντικείμενο που χτύπησε η ακτίνα με την βοήθεια της συνάρτησης ‘FindClosestHitObject’ η οποία αναλύεται παρακάτω. Στην συνέχεια, αν αυτό που χτυπήσαμε δεν είναι το κενό, ελέγχουμε αν έχει πόντους ζωής (δηλαδή αν δεν είναι αντικείμενο) και αν έχει σημαίνει πως είναι κάποιος άλλος παίκτης (μεταβλητή ‘tm’). Έπειτα, αν αυτός ο παίκτης δεν ανήκει στην ίδια ομάδα με εμάς, τότε δέχεται ζημιά και οι πόντοι ζωής του μειώνονται με την βοήθεια της εντολής ‘h.GetComponent<PhotonView>().RPC("TakeDamage",PhotonTargets.AllBuffered, weaponData.damage);’, η οποία τρέχει ταυτόχρονα ξεχωριστά στην κονσόλα κάθε χρήστη (Remote Procedure Call) έτσι ώστε να είναι όλοι ενημερωμένοι για τους πόντους ζωής του συγκεκριμένου παίκτη. Τέλος καλείται η συνάρτηση ‘DoGunFX’ που είναι υπεύθυνη για την εμφάνιση των ακτινών έτσι όπως τις βλέπει κάθε χρήστης ξεχωριστά.

Τέλος, έχουμε την συνάρτηση ‘DoGunFX’ η οποία, εκτός από την λειτουργία που προαναφέρθηκε, εμπεριέχει και την συνάρτηση ‘FindClosestHitObject’. Η συνάρτηση αυτή αποθηκεύει σε έναν πίνακα όλα τα αντικείμενα που χτύπησε η ακτίνα και επιστρέφει στο κύριο πρόγραμμα το πρώτο αντικείμενο με όλα του τα χαρακτηριστικά.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

**[1]** https://en.wikipedia.org/wiki/List_of_video_game_genres

**[2]** https://en.wikipedia.org/wiki/Shooter_game

**[3]** https://en.wikipedia.org/wiki/Unity_%28game_engine%29

**[4]** https://en.wikipedia.org/wiki/Blender_%28software%29

**[5]** https://en.wikipedia.org/wiki/History_of_video_games#Gaming_computers

**[6]** https://teacherreid.wordpress.com/2012/10/01/pros-and-cons-of-different-software-in-the-classroom-blender-and-xna/

**[7]** http://windows7themes.net/en-us/3d-modelling-software-5-best-alternatives-to-blender-3d/

**[8]** http://blog.digitaltutors.com/unity-udk-cryengine-game-engine-choose/