ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση και ανάπτυξη διαδικτυακού ηλεκτρονικού παιχνιδιού σε πλατφόρμα Unity

Design and development of an online video game on the Unity platform

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΚΟΝΤΟΓΙΑΝΝΗΣ ΔΗΜΗΤΡΙΟΣ

Βόλος, ΜΑΙΟΣ 2017

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση και ανάπτυξη διαδικτυακού ηλεκτρονικού παιχνιδιού σε πλατφόρμα Unity

Design and development of an online video game on the Unity platform

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ
ΚΟΝΤΟΓΙΑΝΝΗΣ ΔΗΜΗΤΡΙΟΣ

Επιβλέποντες :

ΤΣΟΜΠΑΝΟΠΟΥΛΟΥ ΠΑΝΑΓΙΩΤΑ	ΠΟΤΑΜΙΑΝΟΣ ΓΕΡΑΣΙΜΟΣ
ΑΝΑΠΛΗΡΩΤΡΙΑ ΚΑΘΗΓΗΤΡΙΑ Π.Θ	ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ Π.Θ

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την ????????????

(Υπογραφή)

.....

ΤΣΟΜΠΑΝΟΠΟΥΛΟΥ ΠΑΝΑΓΙΩΤΑ

ΑΝΑΠΛΗΡΩΤΡΙΑ ΚΑΘΗΓΗΤΡΙΑ Π.Θ

(Υπογραφή)

.....

ΠΟΤΑΜΙΑΝΟΣ ΓΕΡΑΣΙΜΟΣ

ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ Π.Θ

(Υπογραφή)

.....

ΚΟΝΤΟΓΙΑΝΝΗΣ ΔΗΜΗΤΡΙΟΣ

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστημίου Θεσσαλίας

© 2017– All rights reserved

Η σελίδα αυτή είναι σκόπιμα λευκή.

Περίληψη

Καθώς η βιομηχανία ανάπτυξης ηλεκτρονικών παιχνιδιών εξελίσσεται παράλληλα με τις τεχνολογίες δικτύων, οι προγραμματιστές παιχνιδιών καλούνται να συνδυάσουν στοιχεία και τεχνικές που δίνουν έμφαση στη διαδραστικότητα, προσφέροντας μεγαλύτερη αλληλεπίδραση μεταξύ των παικτών και καλλιεργώντας στοιχεία ανταγωνισμού και συνεργασίας. Επομένως, τα διαδικτυακά παιχνίδια πολλών παικτών έγιναν γρήγορα μία από τις πιο δημοφιλείς κατηγορίες παιχνιδιών διότι η εμπειρία που προσφέρουν διαφέρει αρκετά από τα ηλεκτρονικά παιχνίδια ενός παίκτη, έχοντας μεγαλύτερη αξία επανάληψης. Στόχος αυτής της εργασίας είναι η σχεδίαση και ανάπτυξη ενός παιχνιδιού First Person Shooter σε λειτουργικό σύστημα Windows που υποστηρίζει ανταγωνιστικό διαδικτυακό παιχνίδι πολλών παικτών, χρησιμοποιώντας τα εργαλεία ανάπτυξης και δικτύωσης που παρέχει η πλατφόρμα προγραμματισμού παιχνιδιών Unity. Συνεπώς, πρόκειται να γίνει παρουσίαση της πλατφόρμας, της διαδικασίας ανάπτυξης του παιχνιδιού καθώς και της υπηρεσίας Unity Networking (UNet) η οποία προσφέρει βιβλιοθήκες κατάλληλες για προγραμματισμό δικτυακών εφαρμογών και δίνει τη δυνατότητα εκτέλεσης των ηλεκτρονικών παιχνιδιών στους εξυπηρετητές της Unity κατά την κυκλοφορία του παιχνιδιού. Επειδή η πρόσβαση στους εξυπηρετητές της Unity αποτελεί συνδρομητικό πρόγραμμα, τα πειράματα και ο έλεγχος του παιχνιδιού έγιναν σε οικιακό τοπικό δίκτυο έτσι ώστε η διαδικασία να έχει μηδενικό κόστος και να μπορούμε εύκολα να παρατηρήσουμε τη συμπεριφορά των μηχανισμών που σταδιακά προσαρτώνται στο παιχνίδι που υλοποιούμε χωρίς τις ανακριβείς ενδείξεις θορυβώδους καναλιού ,εκμηδενίζοντας έτσι τις επικοινωνιακές καθυστερήσεις οι οποίες θα είναι αναπόφευκτες εφόσον αποφασίσουμε να φιλοξενήσουμε το παιχνίδι μας σε έναν απομακρυσμένο εξυπηρετητή. Στις τελευταίες ενότητες θα αναπτυχθεί μία εκτενής συζήτηση των αποτελεσμάτων της εργασίας και θα υπάρξουν προτάσεις βελτίωσης του ηλεκτρονικού παιχνιδιού με την αξιοποίηση τεχνικών και χαρακτηριστικών που χρησιμοποιούνται στα πιο σύγχρονα First Person Shooters.

Abstract

As the video game industry evolves in conjunction with network technologies, game developpers are called to combine elements and techniques that focus on interactivity in order to offer increased interaction among players and cultivate competitive and cooperative aspects. Hence, multiplayer games promptly became one of the most popular gaming genres because they offer a drastically different experience from single player games while having greater replay value. The aim of this project is to design and develop an online competitive multiplayer First Person Shooter which uses the development and networking tools of Unity game engine and runs on Windows. Therefore, the Unity platform and the development process will be presented as well as the Unity Networking (UNet) service which offers libraries suitable for the development of network applications while providing the ability to execute video games on the Unity servers when they are released. Due to the fact that access to the Unity servers is part of a paid subscription program, experimentation and testing were carried out in a local home network in order to nullify the cost and be able to easily observe the behavior of each mechanism being gradually implemented without receiving any inaccurate indications due to noisy data channels while also nullifying any communication delay that would occur if we tried to host our game on a remote server. The final chapters contain an extensive discussion concerning the results of the project as well as suggestions aimed at improving the video game with the utilization of techniques and characteristics that apply to modern First Person Shooters.

Περιεχόμενα

Λίστα Εικόνων

Λίστα Πινάκων

1. Εισαγωγή

1.1 Ιστορική Αναδρομή

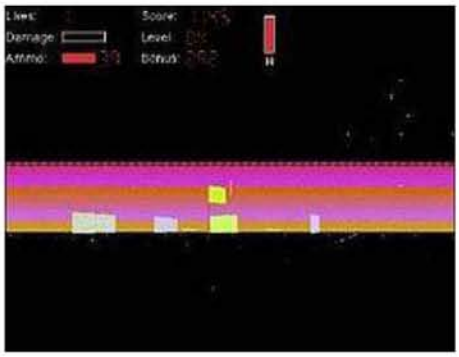
Τα ηλεκτρονικά παιχνίδια που υποστηρίζουν ταυτόχρονη συμμετοχή πολλών παικτών αποτελούν μία από τις πιο δημοφιλείς κατηγορίες παιχνιδιών και η εξέλιξή τους με την πάροδο του χρόνου είναι ραγδαία. Το 1958 η ανάπτυξη του παιχνιδιού Tennis for Two υποστήριξε μη-δικτυακό παιχνίδι δύο παικτών και σύντομα ακολούθησαν παιχνίδια όπως το Spacewar! το 1962 και το Astro Race (Εικόνα 1) που αναπτύχθηκε το 1973. Τα παιχνίδια πολλών παικτών έδωσαν την ευκαιρία στους παίκτες να αλληλεπιδρούν πιο δυναμικά με το περιβάλλον του παιχνιδιού καθώς η ύπαρξη ενός συμπαίκτη ή αντιπάλου που δεν ελέγχεται από τεχνητή νοημοσύνη, οδηγεί σε περισσότερες προκλήσεις και εμπειρίες οι οποίες δίνουν στο παιχνίδι μεγαλύτερη αξία επανάληψης και καλλιεργούν παράλληλα σχέσεις συνεργασίας και ανταγωνισμού μεταξύ των παικτών. Ωστόσο, μερικές από τις πρώτες μεγάλες προκλήσεις που αντιμετώπισαν οι προγραμματιστές παιχνιδιών αυτού του είδους είναι η ανάγκη των παικτών να μοιράζονται δεδομένα σε πραγματικό χρόνο και η δυνατότητα να παίζουν μεταξύ τους ανεξαρτήτως της απόστασης.

[i] [ii] [iii]

Εικόνα 1:Πρώτα παιχνίδια πολλών παικτών i)Tennis For Two, ii) Spacewar, iii)Astro Race

Η εξέλιξη των δικτυακών τεχνολογιών έδωσε απαντήσεις στις παραπάνω προκλήσεις, οδηγώντας σε μία νέα εποχή δικτυακών παιχνιδιών τα οποία σταδιακά εγκαταλείπουν τα

χαρακτηριστικά τοπικού παιχνιδιού σε ένα υπολογιστή. Η ανάπτυξη παιχνιδιών τοπικού δικτύου (LAN) ήταν μία επιτυχής προσπάθεια συγχρονισμού της κατάστασης του παιχνιδιού σε υπολογιστές κοινού δικτύου. Με αυτή την προσέγγιση, η ανταλλαγή πληροφοριών γινόταν χωρίς καθυστερήσεις και κατά συνέπεια η απόκριση του παιχνιδιού ήταν άμεση. Παιχνίδια όπως το Spectre (1990) ακολούθησαν αποκλειστικά αυτή την προσέγγιση. [1]



Εικόνα 2:Spectre παιχνίδι LAN για το Macintosh της Apple

Η τοπική δικτύωση αποτέλεσε προσωρινή λύση και σύντομα έγινε η μετάβαση στην ανάπτυξη διαδικτυακών παιχνιδιών που στηρίζονται στην επικοινωνία των προγραμμάτων-πελατών (clients) με απομακρυσμένους εξυπηρετητές (servers).Εξαιρετικά δημοφιλή παραδείγματα ηλεκτρονικών παιχνιδιών που ακολουθούν αυτή την προσέγγιση είναι το Battlefield 1 (2016) που υλοποιήθηκε στην πλατφόρμα Frostbite και το Overwatch (2016) που υλοποιήθηκε σε ειδικά κατασκευασμένη πλατφόρμα της Blizzard Entertainment. Βέβαια, το μοντέλο τοπικής δικτύωσης δεν έχει εγκαταλειφθεί αλλά χρησιμοποιείται σήμερα κυρίως για έλεγχο ποιότητας και λειτουργικότητας των παιχνιδιών πριν αυτά φιλοξενηθούν στους εξυπηρετητές και γίνουν διαθέσιμα στο ευρύ κοινό.[2][3]

[i] [ii]

Εικόνα 3: Σύγχρονα Διαδικτυακά Παιχνίδια i) Battlefield 1, ii) Overwatch

1.2 Διαδικτυακά Παιχνίδια First Person Shooter

Η ταχύτατη εξέλιξη των διαδικτυακών παιχνιδιών βοήθησε στην επέκταση πολλών επιμέρους κατηγοριών παιχνιδιών αλλά ιδιαίτερα οφέλη παρατηρούνται στην κατηγορία των First Person Shooters καθώς αυτή η εξέλιξη αποτέλεσε ένα από τους πιο καθοριστικούς παράγοντες ακμής τους. Τα παιχνίδια αυτής της κατηγορίας χρησιμοποιούν κάμερα πρώτου προσώπου προσφέροντας στους παίκτες μία πιο ρεαλιστική θεώρηση του χώρου παιχνιδιού μέσα από το πεδίο όρασης του μοντέλου του χαρακτήρα που καλούνται να χειριστούν. Η βασική ιδέα στην οποία στηρίζονται τα παιχνίδια First Person Shooter είναι η συμμετοχή των παικτών σε μάχες με τη χρήση όπλων για την επίτευξη στόχων αλλά και την ατομική πρόοδο κάθε παίκτη ξεχωριστά. Επιπροσθέτως, τα παιχνίδια αυτά προσφέρουν μία ιδιαίτερη εμπειρία δράσης καθώς η χρήση της κάμερας σε συνδυασμό με τους μηχανισμούς χρήσης όπλων οξύνουν τα αντανακλαστικά των παικτών και την ικανότητα συγχρονισμού των κινήσεων του ματιού με τις κινήσεις των χεριών. Αρχικά, τα First Person Shooters ήταν περιορισμένα στην αναπαράσταση σεναρίων πολέμου με χαρακτήρες ελεγχόμενους από τον υπολογιστή και προκαθορισμένες αλληλεπιδράσεις οι οποίες ενεργοποιούνταν σε συγκεκριμένα χρονικά διαστήματα χωρίς να δίνουν στους παίκτες την αίσθηση της ελευθερίας στο επίπεδο του παιχνιδιού. Όμως αυτή η πρώτη προσέγγιση θα άλλαζε ριζικά με την εξάπλωση των διαδικτυακών παιχνιδιών.

Με την ενσωμάτωση διαδικτυακών δυνατοτήτων στα First Person Shooters, οι προγραμματιστές έχουν την ευκαιρία να αναπαραστήσουν ρεαλιστικά σενάρια στα οποία οι παίκτες πρέπει να δράσουν είτε ομαδικά σε συνεργασία με άλλους παίκτες είτε ατομικά δρώντας σε ένα ανταγωνιστικό περιβάλλον μάχης. Αξίζει να σημειωθεί ότι ο ανταγωνιστικός χαρακτήρας αυτών των παιχνιδιών έχει συμβάλλει στην εξέλιξη του ηλεκτρονικού αθλητισμού καθώς η παρουσίαση των ικανοτήτων των παικτών αναδείχθηκε ως δημοφιλής δραστηριότητα στο χώρο του θεάματος. Η δημοτικότητα των παιχνιδιών αυτών αυξήθηκε ραγδαία καθώς ο χειρισμός και ο μέσος βαθμός δυσκολίας τους τα καθιστούν προσιτά στο μεγαλύτερο ποσοστό των παικτών. Επίσης, η απουσία προγραμματισμένων αντιπάλων συμβάλει στην δυναμική ροή του παιχνιδιού εισάγοντας στοιχεία τυχαιότητας στην κίνηση των παικτών. Συνεπώς, τα παιχνίδια αυτά προσφέρουν απεριόριστη αξία επανάληψης λόγω των απρόβλεπτων ενεργειών των ανθρώπινων αντιπάλων δημιουργώντας διαφορετικά σενάρια μάχης. Χαρακτηριστικά παραδείγματα των πιο δημοφιλών διαδικτυακών First Person Shooters αποτελούν οι σειρές παιχνιδιών Counter Strike, Call Of Duty και Battlefield

οι οποίες σημειώνουν εκατομμύρια πωλήσεων ετησίως και έχουν διαμορφώσει τη σύγχρονη σκηνή του ηλεκτρονικού αθλητισμού.

Τα σύγχρονα First Person Shooters, ακολουθώντας τα πρότυπα των περισσότερων διαδικτυακών παιχνιδιών, χρειάζεται να αντιμετωπίσουν τους παίκτες ως ξεχωριστούς χρήστες αντιστοιχίζοντάς τους σε ένα λογαριασμό ο οποίος θα αντικατοπτρίζει την πρόοδό τους στο παιχνίδι. Επίσης, παρέχεται η δυνατότητα στους παίκτες είτε να φιλοξενήσουν είτε να συνδεθούν σε αγώνες άλλων χρηστών ώστε να παίξουν με άλλους χρήστες. Η σύνδεση στο διαδίκτυο είναι υποχρεωτική για την πρόσβαση στο παιχνίδι και η εμπειρία των παικτών ποικίλει σύμφωνα με την απόστασή τους από το διοργανωτή του παιχνιδιού αλλά και την ταχύτητα με την οποία οι ίδιοι μπορούν να στέλνουν μηνύματα στο δίκτυο. Τέλος, το περιβάλλον παιχνιδιού που αποτελείται από το επίπεδο, τα μοντέλα των αντικειμένων και τις διεπαφές των παικτών χρειάζεται να ανανεώνεται συνεχώς σε πραγματικό χρόνο ώστε να αποτυπώνει όλες τις ενέργειες των παικτών.[4][5][6]

1.3 Στόχος Εργασίας και Διάρθρωση Ενοτήτων

Σε αυτή την εργασία θα εστιάσουμε στη σχεδίαση και την ανάπτυξη ενός διαδικτυακού ηλεκτρονικού παιχνιδιού της κατηγορίας first person shooter σύμφωνα με τα πρότυπα των σύγχρονων client-server τρισδιάστατων παιχνιδιών με στόχο την ανάλυση των εργαλείων και των απαραίτητων προγραμματιστικών τεχνικών που χρειάζονται για την εφαρμογή και εκτέλεση των βασικών μηχανισμών ενός παιχνιδιού αυτής της κατηγορίας. Η ανάπτυξη του παιχνιδιού γίνεται με τη χρήση της πλατφόρμας Unity 5.5.1 και η εφαρμογή διαδικτυακών χαρακτηριστικών γίνεται με χρήση του Unity Networking, ενός συνόλου προγραμματιστικών εργαλείων για διαδικτυακά παιχνίδια που είναι μέρος της υπηρεσίας Unity Multiplayer. Ο έλεγχος των μηχανισμών του παιχνιδιού στο δίκτυο γίνεται σε τοπικό οικιακό δίκτυο διότι με αυτό τον τρόπο εκμηδενίζονται οι καθυστερήσεις επικοινωνίας οι οποίες θα μπορούσαν να αλλοιώσουν το αποτέλεσμα σε περίπτωση που ο εξυπηρετητής ο οποίος φιλοξενεί το παιχνίδι βρίσκεται σε μεγάλη απόσταση από τα προγράμματα-πελάτη. Επίσης, αυτός ο τρόπος ελέγχου ενδείκνυται από τη Unity ως προσομοίωση των διαδικτυακών λειτουργιών διότι η διεπαφή που χρησιμοποιεί το Unity Networking επιτρέπει στους προγραμματιστές να αναπτύξουν ένα διαδικτυακό παιχνίδι ανεξαρτήτως της τοπολογίας λόγω της ενοποίησης των δικτυακών λειτουργιών στις κλάσεις της διεπαφής .Η υπηρεσία Unity Multiplayer αποτελεί συνδρομητική υπηρεσία και απευθύνεται κυρίως σε παιχνίδια τα οποία βρίσκονται σε Early Access και Beta περιόδους κυκλοφορίας , επομένως με τη χρήση των προσομοιώσεων γίνεται

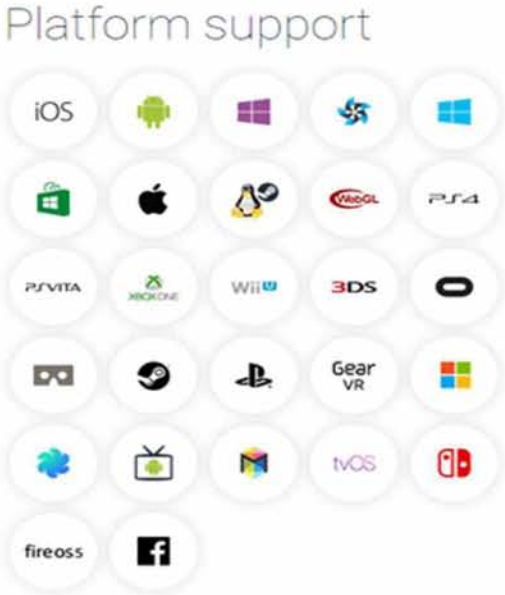
αξιολόγηση της λειτουργίας του παιχνιδιού ενώ παράλληλα εκμηδενίζεται το κόστος καθώς αυτό το παιχνίδι είναι αποτέλεσμα εργασίας και όχι εμπορικό προϊόν.

Στις επόμενες ενότητες της εργασίας γίνεται αναλυτική παρουσίαση των δομικών στοιχείων και των αρχών λειτουργίας της πλατφόρμας ανάπτυξης παιχνιδιών Unity 5 καθώς και των πληροφοριών που σχετίζονται με το Unity Networking. Η διερεύνηση αυτού του υποβάθρου είναι σημαντική για την βαθύτερη κατανόηση της διαδικασίας ανάπτυξης του παιχνιδιού και απευθύνεται τόσο σε αναγνώστες χωρίς εμπειρία με την πλατφόρμα όσο και σε αυτούς που έχουν προγραμματίσει σε Unity στο παρελθόν και ίσως δεν γνωρίζουν τις νέες δυνατότητες που προσφέρει η πλατφόρμα στην τρέχουσα έκδοσή της (version 5.5.1) με την ενσωμάτωση του High Level API. Ακολούθως, παρουσιάζονται η διαδικασία σχεδίασης και ανάπτυξης του first person shooter που έχουμε ονομάσει Project Bullet:Versus και συνοδεύεται από screenshots αλλά και επεξήγηση του κώδικα της υλοποίησης. Τέλος, παρατίθενται ενότητες στις οποίες γίνεται συζήτηση των αποτελεσμάτων, αναφέρονται προτάσεις για μελλοντική εξέλιξη της εργασίας και παρουσιάζονται τα συμπεράσματα που απορρέουν από αυτή την προγραμματιστική διαδικασία.

2. Επισκόπηση Πλατφόρμας Unity

2.1 Τί είναι η Unity;

Η Unity είναι μία πλατφόρμα ανάπτυξης παιχνιδιών η οποία αναπτύχθηκε από την εταιρεία Unity Technologies το 2005. Αρχικά η πλατφόρμα Unity υποστήριζε μόνο το OS X της Apple αλλά έκτοτε έχουν γίνει πολλές αλλαγές που έχουν οδηγήσει στην υποστήριξη σχεδόν όλων των λειτουργικών συστημάτων και όλων των συστημάτων εκτέλεσης παιχνιδιών (Εικόνα 4). Η έκδοση της πλατφόρμας που χρησιμοποιούμε είναι η 5.5.1 η οποία κυκλοφόρησε στις 24 Ιανουαρίου 2017.

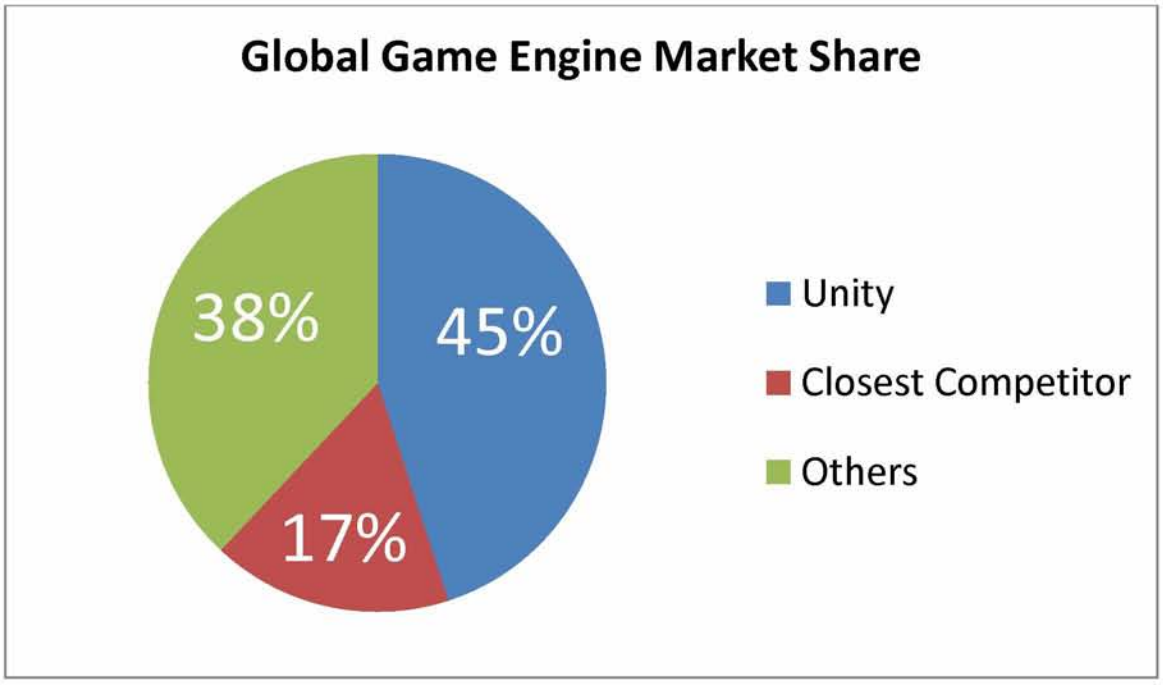


Εικόνα 4: Υποστηριζόμενα συστήματα της Unity

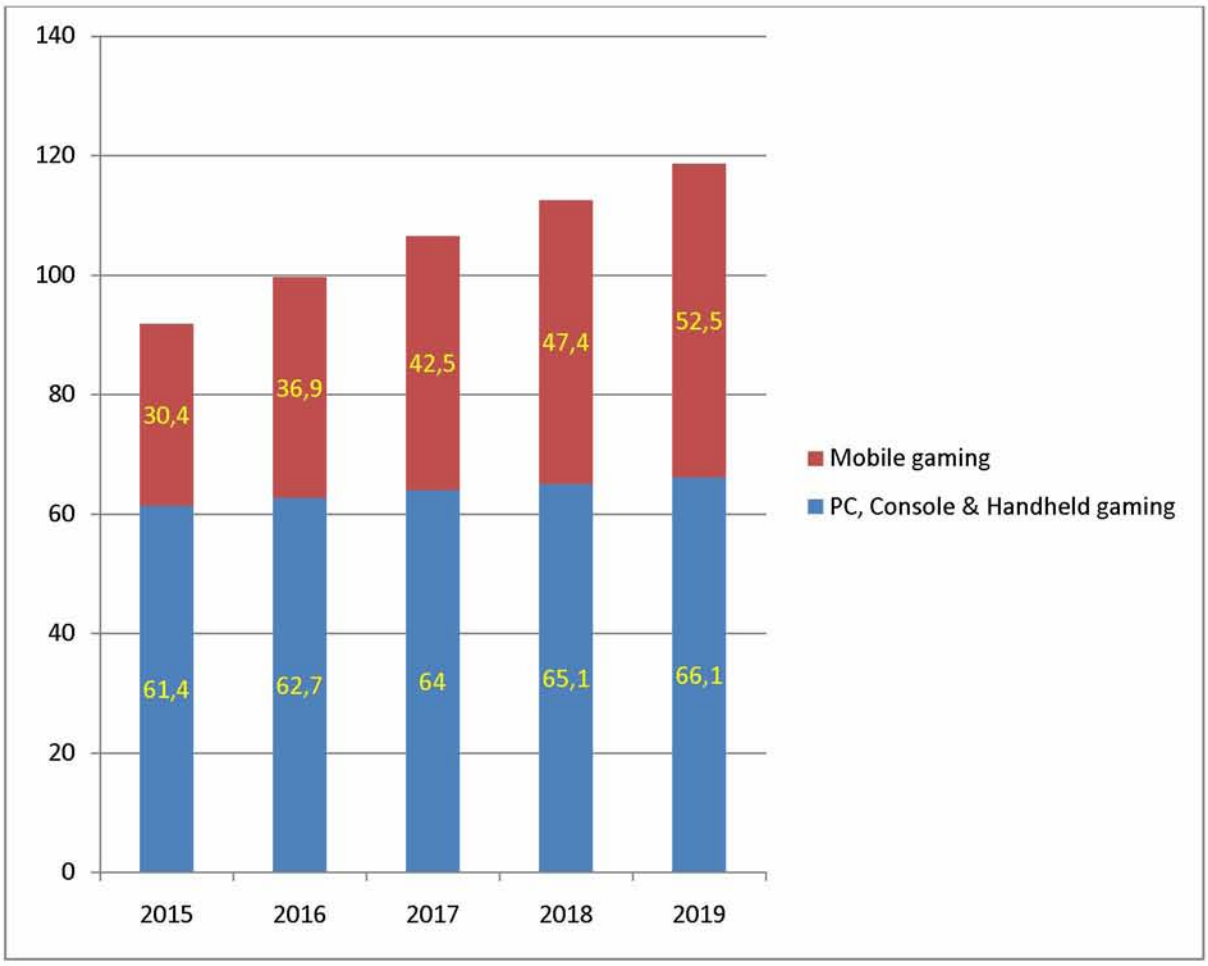
Ένα σημαντικό χαρακτηριστικό της πλατφόρμας Unity είναι η ικανότητα ανάπτυξης παιχνιδιών για πολλά συστήματα ταυτόχρονα, δίνοντας τη δυνατότητα στους προγραμματιστές να μεταφέρουν το παιχνίδι που αναπτύσσουν από ένα σύστημα σε ένα άλλο με ευκολία. Αυτή η δυνατότητα παρέχεται λόγω της ενσωμάτωσης διεπαφών

προγραμματισμού εφαρμογών συμβατών με κάθε σύστημα. Ειδικότερα, για συστήματα Android και iOS χρησιμοποιείται το OpenGL ES API, για Macintosh και Linux το OpenGL ενώ για Windows χρησιμοποιούνται εκτός του OpenGL οι διεπαφές Direct3D και Vulkan. Η Unity έχει γραφθεί σε C και C++ και ο προγραμματισμός σε αυτή την πλατφόρμα γίνεται κυρίως με C# και Javascript. Η κοινότητα της πλατφόρμας είναι ενεργή και βοηθά νέους προγραμματιστές με εργαλεία και προγραμματιστικές συμβουλές στα forums.

Αξίζει επίσης να σημειωθεί ότι η πλατφόρμα Unity έχει μία εξέχουσα θέση στη σύγχρονη βιομηχανία ανάπτυξης παιχνιδιών καθώς κατέχει ένα μεγάλο ποσοστό της αγοράς (Εικόνα 5) και σε έρευνα που διεξήχθη το 2014 κατέκτησε την πρώτη θέση ως η πιο δημοφιλής πλατφόρμα ανάπτυξης παιχνιδιών στο Ηνωμένο Βασίλειο με ποσοστό 62%. Σήμερα, η Unity προσεγγίζει κοινό 770 εκατομμυρίων παικτών συμβάλλοντας στην αύξηση των εσόδων της βιομηχανίας ηλεκτρονικών παιχνιδιών, σύμφωνα με εκτιμήσεις αναλυτών του Gartner (Εικόνα 6).Η πλατφόρμα κυκλοφορεί δωρεάν σε έκδοση Personal αλλά υπάρχουν και οι εκδόσεις Plus, Pro και Enterprise οι οποίες απευθύνονται σε επαγγελματική χρήση προσφέροντας δυνατότητες πρόσβασης στον πηγαίο κώδικα της Unity και αναφοράς επιδόσεων.[7]



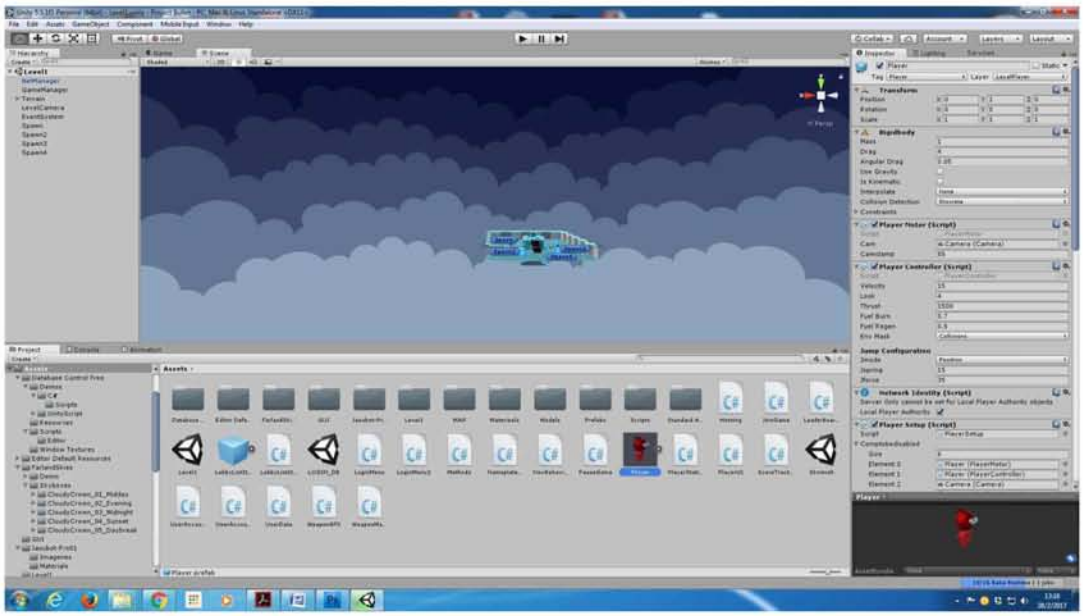
Εικόνα 5: Μερίδιο αγοράς Unity σε σύγκριση με τους ανταγωνιστές της.



Εικόνα 6 : Εκτιμώμενη πορεία των κερδών της βιομηχανίας ηλεκτρονικών παιχνιδιών εκφρασμένη σε δισεκατομμύρια US$ παγκοσμίως, κατά τα έτη 2015-2019 (Gartner 2013)

2.2 Το περιβάλλον σχεδίασης της Unity

Η διεπαφή χρήστη που χρησιμοποιεί η πλατφόρμα Unity για την παρουσίαση , τον έλεγχο και τη διαμόρφωση αντικειμένων παιχνιδιού χωρίζεται σε μία σειρά όψεων από τις οποίες ο προγραμματιστής μπορεί σε πραγματικό χρόνο να διαχειριστεί το παιχνίδι χωρίς να πραγματοποιήσει δομικές αλλαγές στον κώδικα που αναπτύσσει, έχοντας παράλληλα την ικανότητα να αλλάξει τις τιμές των μεταβλητών που εμπλέκονται στο παιχνίδι ώστε να ελέγξει τη συμπεριφορά συγκεκριμένων μηχανισμών. Οι πιο σημαντικές όψεις είναι οι όψεις σκηνής, παιχνιδιού, ιεραρχίας, εργασίας και παρατήρησης. Το περιβάλλον σχεδίασης φαίνεται συγκεντρωτικά στην Εικόνα 7.[8]



Εικόνα 7: Το περιβάλλον σχεδίασης της Unity

2.2.1 Γραμμή Εργαλείων

Η γραμμή εργαλείων συμβάλλει στην εφαρμογή μετασχηματισμών στη σκηνή που σχεδιάζουμε, μας επιτρέπει να ελέγχουμε τη ροή του παιχνιδιού σε πραγματικό χρόνο καθώς και να διαχειριστούμε το λογαριασμό μας στη Unity ώστε να αναπτύξουμε συνεργασίες είτε να μεταφέρουμε υλικό από και προς τη σκηνή μας με χρήση νεφοϋπολογιστικής.[9]

Εικόνα 8: Επισκόπηση της γραμμής εργαλείων

(i) (ii) (iii) (iv) (v)

Εικόνα 9 : i) Εργαλεία Μετασχηματισμών αντικειμένων (Move Camera / Move Object / Rotate / Scale / Rectangular Transform) ii) Εργαλεία ρύθμισης συστήματος συντεταγμένων iii) Έλεγχος ροής παιχνιδιού (Play / Pause / Step) iv) Επικαλυπτόμενη Σχεδίαση v) Επιλογή Διάταξης

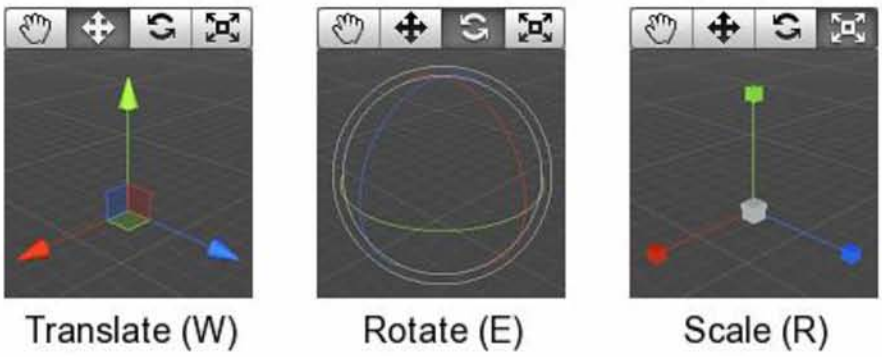
2.2.2 Όψη Σκηνής

Η όψη σκηνής (Εικόνα 10) αποτελεί το πιο διαδραστικό κομμάτι του περιβάλλοντος σχεδίασης καθώς σε αυτή την όψη εμφανίζεται το σύνολο των μετασχηματισμών σε

αντικείμενα και ο προγραμματιστής μπορεί χειροκίνητα να περιηγηθεί στη σκηνή που έχει σχεδιάσει. Η περιήγηση της σκηνής μπορεί να γίνει με πληθώρα τεχνικών προοπτικής ή ορθογραφικής προβολής καθώς η πλατφόρμα Unity υποστηρίζει 2D και 3D προβολές για ανάπτυξη δισδιάστατων και τρισδιάστατων παιχνιδιών αντίστοιχα.

Εικόνα 10: Όψη Σκηνής

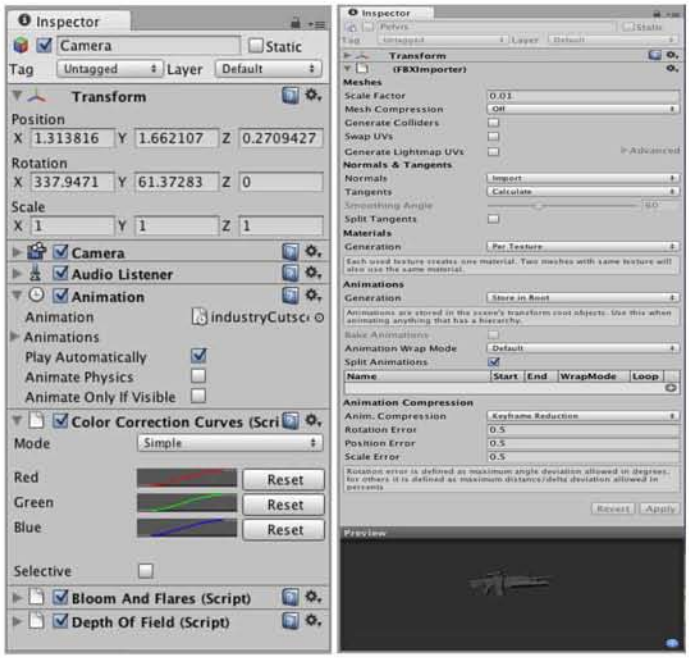
Η επιλογή αντικειμένων γίνεται εύκολα με κλικ εντός του πλαισίου της όψης. Με την επιλογή του αντικειμένου στην όψη αυτή, ενεργοποιούνται τα στιγμιότυπά του στην όψη παρατήρησης αλλά και στην όψη ιεραρχίας προβάλλοντας πληροφορίες σχετικά με τις αλλαγές που έχουν εφαρμοστεί στο αντικείμενο. Ειδικότερα, για τους μετασχηματισμούς τρισδιάστατου αντικειμένου που στην επιστήμη γραφικών υπολογιστών υλοποιούνται με χρήση πινάκων, ο σχεδιαστής του ηλεκτρονικού παιχνιδιού καλείται να επιλέξει το κουμπί που εμπεριέχει αυτό το μετασχηματισμό και στη συνέχεια να διαχειριστεί το σύστημα συντεταγμένων εντός του πλαισίου της όψης σκηνής (Εικόνα 11).[10]



Εικόνα 11: Βασικοί Μετασχηματισμοί Αντικειμένου στην Όψη Σκηνής

2.2.3 Όψη Παρατήρησης

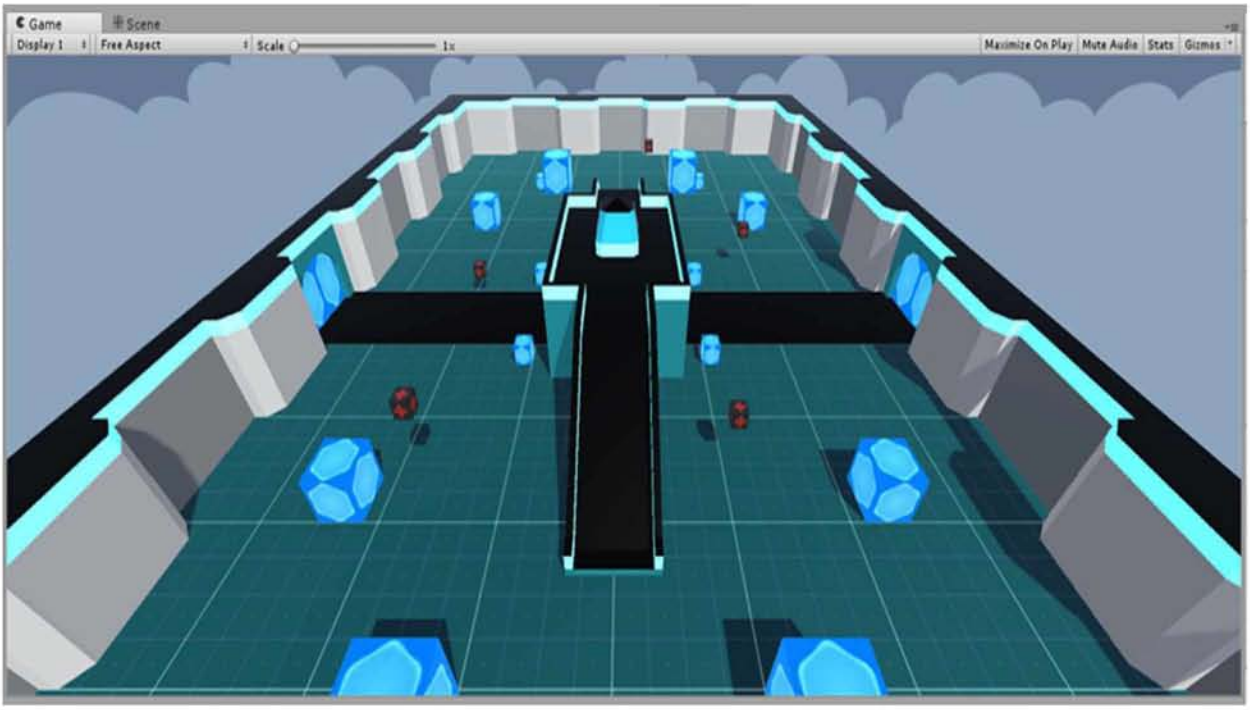
Η όψη παρατήρησης δίνει τη δυνατότητα προβολής των τιμών των μεταβλητών που διαχειρίζεται ο κώδικάς μας όπως επίσης και του συστήματος συντεταγμένων του επιλεγμένου αντικειμένου. Μέσα από αυτή την όψη μπορούμε να ορίσουμε και να αναθέσουμε συμπεριφορές και δομικά στοιχεία αντικειμένων τα οποία πρώτα έχουμε καθορίσει σε ένα C# ή Javascript αρχείο κώδικα. Οι μεταβλητές οι οποίες έχουν τεθεί ως public είτε ως SerializedField είναι απευθείας ορατές από αυτή την όψη ενώ για τις αυστηρά private μεταβλητές χρειάζεται να θέσουμε την παρατήρηση σε Debug Mode. Επιπροσθέτως, μπορούμε να διαφοροποιήσουμε ένα αντικείμενο από άλλα του ίδιου είδους προσθέτοντας σε αυτό ετικέτες με τις οποίες μπορούμε επίσης να αναφερθούμε στο αντικείμενο μέσα στον κώδικα. Ακόμη, στην όψη παρατήρησης τα σχόλια που προσθέτουμε στον κώδικά μπορούν να γίνουν ορατά με τη λειτουργία Tooltip ενώ πάντοτε μπορούμε να αποκρύψουμε πληροφορίες public μεταβλητών με τη χρήση της λειτουργίας HideInspector. Τέλος, στο κάτω μέρος της όψης παρατήρησης βλέπουμε μία προεπισκόπηση του αντικειμένου που παρατηρούμε μεμονωμένα, χωρίς την επίδραση των μετασχηματισμών σκηνής. Ενδεικτικά στιγμιότυπα της όψης παρατήρησης φαίνονται στην Εικόνα 12. [11]



Εικόνα 12: Στιγμιότυπα όψης παρατήρησης

2.2.4 Όψη Παιχνιδιού

Η όψη παιχνιδιού αποτελεί ένα χρήσιμο τρόπο ελέγχου του παιχνιδιού πριν αυτό εξαχθεί από τη Unity.Σε αυτή την όψη το παιχνίδι μπορεί να εκτελεστεί στο σύνολό του είτε με αφετηρία κάποια συγκεκριμένη σκηνή, δίνοντας μεγαλύτερη ευελιξία από αυτή που προσφέρει το τελικό εκτελέσιμο αρχείο. Η κάμερα η οποία είναι ενεργή σε αυτή την όψη είναι η κύρια κάμερα του παιχνιδιού σε αντίθεση με μία εξωτερική νέα κάμερα που ενεργοποιείται στην όψη σκηνής. Τα κουμπιά της εργαλειοθήκης επενεργούν στην όψη παιχνιδιού με παρόμοιο τρόπο με την όψη σκηνής , ενώ τα αντικείμενα και τα δομικά στοιχεία τους εμφανίζονται στις όψεις παρατήρησης και ιεραρχίας καθώς το παιχνίδι εκτελείται σε πραγματικό χρόνο. Ο σχεδιαστής παιχνιδιού καλείται να πραγματοποιήσει αλλαγές στα αντικείμενα αυτών των όψεων αλλά όσο η όψη παιχνιδιού παραμένει ενεργή οι αλλαγές αυτές δεν αποθηκεύονται κατά τη μετάβαση στην όψη σκηνής. Τέλος, ο σχεδιαστής θα μπορούσε να προσαρμόσει την ανάλυση παραθύρου και να εφαρμόσει ρυθμίσεις που επηρεάζουν την οθόνη και το παράθυρο του τελικού εξαγόμενου παιχνιδιού. Ένα στιγμιότυπο της όψης παιχνιδιού φαίνεται στην Εικόνα 13.[12]



Εικόνα 13: Στιγμιότυπο Όψης Παιχνιδιού

2.2.5 Όψη Ιεραρχίας

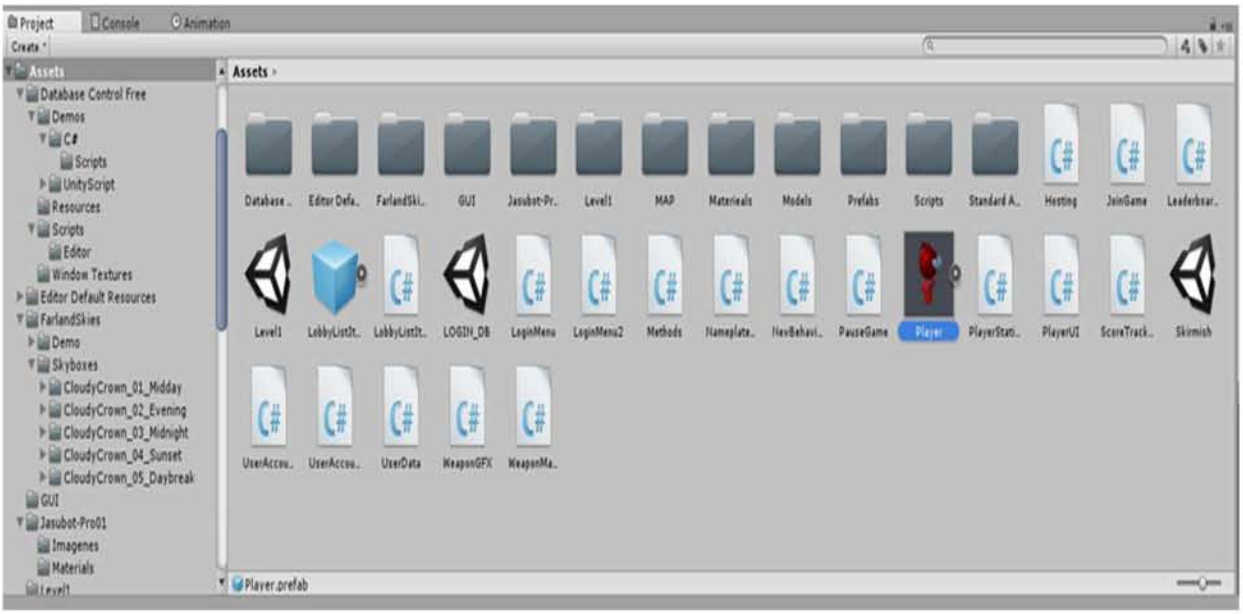
Στην όψη ιεραρχίας περιέχεται μία λίστα με όλα τα αντικείμενα που συνυπάρχουν σε μία σκηνή και δίνεται η δυνατότητα ενεργοποίησης είτε απενεργοποίησής τους όπως επίσης και η δημιουργία νέων αντικειμένων. Ο σχεδιαστής παιχνιδιού μπορεί να καθορίσει τη σειρά με την οποία σχεδιάζονται τα αντικείμενα στη σκηνή, ένα χρήσιμο εργαλείο σε περίπτωση επικαλυπτόμενης σχεδίασης. Επιπροσθέτως, υπάρχει η δυνατότητα δημιουργίας εμφωλευμένων αντικειμένων, όπου το γονικό αντικείμενο μπορεί να έχει ένα σύνολο συμπεριφορών και μετασχηματισμών το οποίο κληρονομούν τα αντικείμενα-παιδιά. Βέβαια, ο σχεδιαστής μπορεί να αλλάξει τις συμπεριφορές των αντικειμένων παιδιών χειροκίνητα είτε μέσω κώδικα, χωρίς να επηρεάσει το γονικό αντικείμενο, με αποτέλεσμα να δημιουργούνται ομάδες αντικειμένων οι οποίες εύκολα προσαρμόζονται στο παιχνίδι. Ένα στιγμιότυπο της όψης ιεραρχίας φαίνεται στην Εικόνα 14.[13]

Εικόνα 14: Στιγμιότυπο όψης ιεραρχίας

2.2.6 Περιηγητής Περιεχομένων Εργασίας

Στον περιηγητή περιεχομένων βρίσκονται όλα τα αρχεία που σχετίζονται με μία συγκεκριμένη εργασία. Η ιεράρχηση των αρχείων γίνεται με ένα σύστημα φακέλων και δίνεται η δυνατότητα αναζήτησης αρχείων έτσι ώστε οι σχεδιαστές και προγραμματιστές παιχνιδιών να διευκολύνονται σε μεγαλύτερες και πιο πολύπλοκες εργασίες. Η εισαγωγή

αρχείου σε μία εργασία είναι πολύ απλή καθώς γίνεται εφαρμογή ενός drag-and-drop συστήματος. Με την εισαγωγή ενός νέου αρχείου γίνεται αυτόματη αναγνώριση του format και των ιδιοτήτων του και το αρχείο μπορεί να προσαρτηθεί στη σκηνή μετά από κατάλληλες ρυθμίσεις στην όψη παρατήρησης. Ο σχεδιαστής δύναται επίσης να ανοίξει και να τροποποιήσει το αρχείο απευθείας από το περιβάλλον σχεδίασης ανοίγοντας ένα από τα εγκατεστημένα προγράμματα, όπως το Visual Studio αν πρόκειται για αρχείο κώδικα είτε το Photoshop αν πρόκειται για αρχείο εικόνας. Ένα στιγμιότυπο του περιηγητή περιεχομένου φαίνεται στην Εικόνα 15.[14]

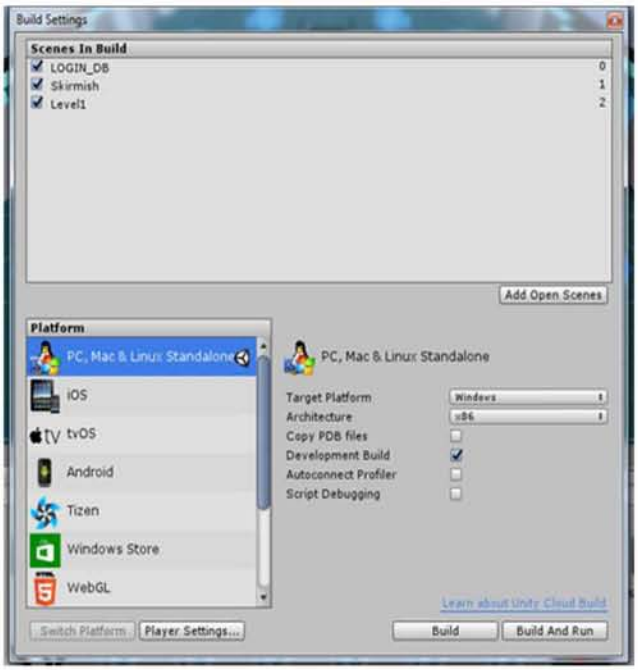


Εικόνα 15: Προεπισκόπηση περιηγητή περιεχομένου εργασίας

2.2.7 Ρυθμίσεις Εξαγωγής Παιχνιδιού

Όταν το παιχνίδι βρίσκεται στα τελευταία στάδια υλοποίησής του, χρειάζεται ο έλεγχος λειτουργίας του έξω από το περιβάλλον σχεδίασης της Unity έτσι ώστε το παιχνίδι να είναι εύκολο να διαμοιραστεί. Η Unity διαθέτει ένα σύνολο εντολών που βοηθούν στην ιεράρχηση των σκηνών αλλά και στην επιλογή του συστήματος στο οποίο θα κυκλοφορήσει το παιχνίδι. Επίσης, παρέχεται και μία λίστα επιλογών που ορίζει τις διαθέσιμες αναλύσεις παραθύρου αλλά και την ποιότητα των γραφικών. Τέλος, ο προγραμματιστής μπορεί να καθορίσει εκτός από το λειτουργικό σύστημα, την αρχιτεκτονική των υπολογιστών που είναι κατάλληλη για

το παιχνίδι αλλά και να συμπεριλάβει εργαλεία αποσφαλμάτωσης τα οποία θα μπορούν σε πραγματικό χρόνο να ανιχνεύουν σφάλματα χωρίς να χρειάζεται συνεχώς πρόσβαση στο τερματικό της Unity. Ένα στιγμιότυπο του πλαισίου ρυθμίσεων εξαγωγής παιχνιδιού φαίνεται στην Εικόνα 16.[15]



Εικόνα 16 : Πλαίσιο ρυθμίσεων Εξαγωγής Παιχνιδιού

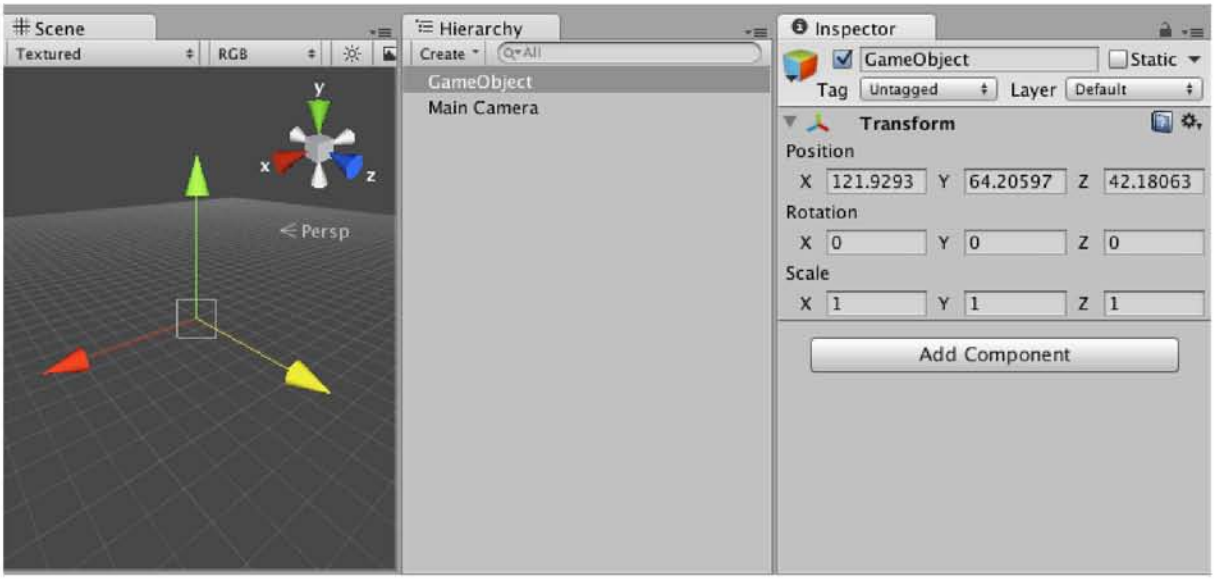
2.3 Αρχές Λειτουργίας της Unity

Στην προηγούμενη ενότητα έγινε ανάλυση του περιβάλλοντος σχεδίασης Unity όμως επιτακτική είναι η ανάγκη να αναλυθούν τα στοιχεία τα οποία αποτελούν ένα παιχνίδι της Unity έτσι ώστε να υπάρξει βαθύτερη κατανόηση της διαδικασίας κατασκευής ενός ηλεκτρονικού παιχνιδιού. Η πλατφόρμα Unity χρησιμοποιεί ένα μοντέλο δομικών στοιχείων και συμπεριφορών το οποίο ουσιαστικά κατακερματίζει κάθε αντικείμενο της σκηνής σε δομικά στοιχεία των οποίων η συμπεριφορά καθορίζεται πλήρως από τον προγραμματιστή με κώδικα. Επομένως, είναι χρήσιμο να γνωρίζουμε τους βασικούς τρόπους με τους οποίους μπορούμε να αλληλεπιδράσουμε με τα αντικείμενα που συμπεριλαμβάνουμε στο παιχνίδι μας.

2.3.1 Αντικείμενο Παιχνιδιού

Το αντικείμενο παιχνιδιού (GameObject) είναι από τα πιο σημαντικά στοιχεία της Unity καθώς αποτελεί το πρότυπο στο οποίο θα προσαρτηθούν δομικά στοιχεία (Components) και θα κωδικοποιηθούν συμπεριφορές (Behaviours). Η δημιουργία αντικειμένων γίνεται είτε από την όψη ιεραρχίας είτε από τον περιηγητή περιεχομένων και ένα αρχικά κενό πρότυπο αντικειμένου περιέχει μόνο το δομικό στοιχείο μετασχηματισμών έτσι ώστε να είναι έτοιμο να προσαρτηθεί στη σκηνή(Εικόνα 17). Κάθε αντικείμενο παιχνιδιού μπορεί να λειτουργεί ανεξάρτητα είτε μαζί με άλλα αντικείμενα με τη χρήση επιπρόσθετων δομικών στοιχείων.

Τα αντικείμενα παιχνιδιού διαφοροποιούνται από το όνομά τους και από μία ετικέτα η οποία δείχνει την κατηγορία του αντικειμένου που θέλουμε να προσδιορίσουμε. Επίσης, στην περίπτωση που επιθυμούμε να σχεδιάσουμε αντικείμενα στις ίδιες συντεταγμένες της σκηνής, ορίζουμε ένα επίπεδο επικάλυψης στο οποίο αναθέτουμε κάθε αντικείμενο. Τέλος, αντικείμενα τα οποία θέλουμε να επαναχρησιμοποιήσουμε με το σύνολο των συστατικών τους κατατάσσονται σε μία ειδική κατηγορία των προκατασκευασμένων αντικειμένων τα οποία η Unity διαχειρίζεται με διαφορετικό τρόπο και τα οποία θα αναλύσουμε σε επόμενη υποπαράγραφο.[16]

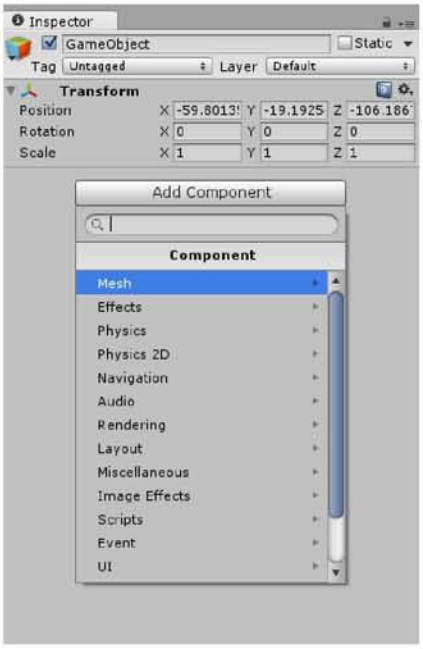


Εικόνα 17: Δημιουργία νέου αντικειμένου παιχνιδιού.

2.3.2 Δομικά Στοιχεία Αντικειμένων

Τα δομικά στοιχεία (Components) αποτελούν επιμέρους κομμάτια που προσαρτώνται σε αντικείμενα ώστε να τους δώσουν επιπλέον ικανότητες με τη χρήση μεθόδων. Οι μεταβλητές των δομικών στοιχείων μπορούν να ελεγχθούν πλήρως από την όψη παρατήρησης αλλά και

να αλλάξουν προσωρινά κατά την όψη παιχνιδιού ώστε ο προγραμματιστής να μπορεί απευθείας να δει πως μεταφράζονται αυτές οι μεταβλητές. Χρειάζεται να προσθέσουμε ότι τα δομικά στοιχεία ανήκουν σε δύο κατηγορίες. Η πρώτη κατηγορία αναφέρεται στα στοιχεία που αποτελούνται μόνο από ήδη υλοποιημένες μεθόδους της Unity και αυτά τα στοιχεία συνήθως αφορούν φυσικές αλληλεπιδράσεις ή αλληλεπιδράσεις πολυμέσων .Από την άλλη πλευρά, η δεύτερη κατηγορία αναφέρεται στα στοιχεία που καθορίζουν τους σύνθετους μηχανισμούς που θέλουμε να εμπεριέχει ένα ηλεκτρονικό παιχνίδι, όπως κίνηση ενός παίκτη στο επίπεδο. Η δεύτερη αυτή κατηγορία ελέγχεται αποκλειστικά από κώδικα που αναπτύσσει ο προγραμματιστής και αυτός ο κώδικας προσαρτάται στο επιλεγμένο αντικείμενο εκφράζοντας τη λογική του παιχνιδιού. Στην Εικόνα 18 βλέπουμε την επιλογή προεπιλεγμένων δομικών στοιχείων.[17]



Εικόνα 18: Προσάρτηση δομικού στοιχείου σε αντικείμενο.

2.3.3 Στοιχεία Προσάρτησης

Τα στοιχεία προσάρτησης (Assets) είναι μεμονωμένα αντικείμενα είτε σύνολα αντικειμένων τα οποία διαμοιράζονται και επαναχρησιμοποιούνται από την κοινότητα της Unity σε περισσότερες από μία εργασίες. Η δημιουργία στοιχείων προσάρτησης είναι μία πολύ απλή διαδικασία καθώς οι προγραμματιστές ή οι σχεδιαστές παιχνιδιών χρειάζεται να δημιουργήσουν πακέτα με τα αντικείμενα τα οποία θέλουν να συμπεριλάβουν και μετά να τα ανεβάσουν στο κατάστημα της Unity. Τα πιο δημοφιλή στοιχεία αφορούν αρχεία πολυμέσων καθώς τα περισσότερα από αυτά στοχεύουν στη διακόσμηση του κόσμου του παιχνιδιού.

Τέλος, κάθε πακέτο στοιχείων προσάρτησης χαρακτηρίζεται από αναφορές στον κατασκευαστή αλλά και από τις υποστηριζόμενες εκδόσεις της Unity καθώς οι συχνές αλλαγές στη διεπαφή προγραμματισμού της πλατφόρμας συχνά οδηγούν σε ασυμβατότητες πακέτων και συνεπώς σε μη λειτουργικές εργασίες.[18]

2.3.4 Προκατασκευασμένα Αντικείμενα

Μία ειδική κατηγορία αντικειμένων παιχνιδιού είναι τα προκατασκευασμένα αντικείμενα (Prefabs), τα οποία αποτελούν πρότυπα σχεδίασης πολλαπλών αντικειμένων της ίδιας κατηγορίας σε μία σκηνή. Αυτά τα πρότυπα αποθηκεύουν το σύνολο των μετασχηματισμών, το σύνολο των δομικών στοιχείων αλλά και όλες τις κωδικοποιημένες συμπεριφορές έτσι ώστε να επαναχρησιμοποιούνται με ευκολία. Κάθε αλλαγή στοιχείων επηρεάζει όλα τα στιγμιότυπα του προκατασκευασμένου αντικειμένου ενώ αλλαγές στην ιεραρχική δομή του αντικειμένου οδηγούν στην αλλοίωση του στιγμιότυπου το οποίο πλέον δεν χαρακτηρίζεται ως προκατασκευασμένο και μπορούμε να το επεξεργαστούμε αυτόνομα.

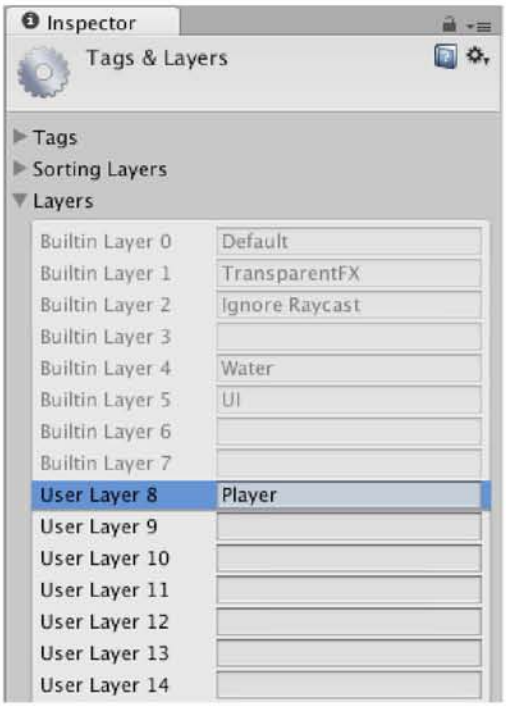
Η δημιουργία προκατασκευασμένων αντικειμένων [19] είναι μια απλή διαδικασία που απορρέει από την ομαδοποίηση των αντικειμένων στην όψη ιεραρχίας. Εφόσον τα αντικείμενα έχουν την επιθυμητή ομαδοποίηση ,με τη μέθοδο drag-and-drop αφαιρούμε τα αντικείμενα από την όψη ιεραρχίας και τα προσθέτουμε στον περιηγητή περιεχομένων. Κάθε προκατασκευασμένο αντικείμενο φιλοξενείται στον περιηγητή περιεχομένων και η μεταφορά ενός στιγμιότυπού του στη σκηνή γίνεται επίσης με drag-and-drop. Επιπροσθέτως αν επιθυμούμε να προσδιορίσουμε πλήρως την τοποθέτηση του στιγμιότυπου καθορίζοντας θέση, περιστροφή και γονικό αντικείμενο στην ιεραρχία, η Unity μας δίνει τη δυνατότητα να καλέσουμε τη μέθοδο Instantiate η οποία ορίζεται ως εξής :

public static Object **Instantiate**(Object **original** , Vector3 **position** ,Quaternion **rotation**, Transform **parent**);

2.3.5 Επικαλυπτόμενη Σχεδίαση

Μία μεγάλη πρόκληση που αντιμετωπίζουν οι προγραμματιστές κατά την ανάπτυξη παιχνιδιών πολλών κινούμενων καμερών είναι η ταξινόμηση των αντικειμένων έτσι ώστε αυτά να σχεδιάζονται σωστά στη σκηνή. Σε μία σκηνή με πολλά αντικείμενα, η όψη ιεραρχίας δεν μπορεί να εγγυηθεί ότι η αντιστοίχηση του φωτός αλλά και των συγκρούσεων γίνεται σωστά όταν τα αντικείμενα έχουν ίδιες είτε μερικώς επικαλυπτόμενες συντεταγμένες.

Η πλατφόρμα Unity δίνει λύσεις σε αυτό το πρόβλημα, παρέχοντας τη δυνατότητα καθορισμού επιπέδων σχεδίασης (Layers). Κάθε επίπεδο σχεδίασης αντιμετωπίζεται αυτόνομα από κάθε κάμερα στον κόσμο παιχνιδιού, και η προτεραιότητά του δίνεται από τη θέση του στη λίστα επιπέδων που είναι ορατή από την όψη παρατήρησης. Κατά την επιλογή ενός αντικειμένου, η αντιστοίχηση του σε ένα επίπεδο σχεδίασης γίνεται άμεσα από την όψη παρατήρησης. Στην Εικόνα 19 βλέπουμε τη λίστα επιπέδων σχεδίασης που περιλαμβάνει τα προκαθορισμένα επίπεδα της Unity αλλά και τα επίπεδα που έχει δημιουργήσει ο προγραμματιστής.[20]

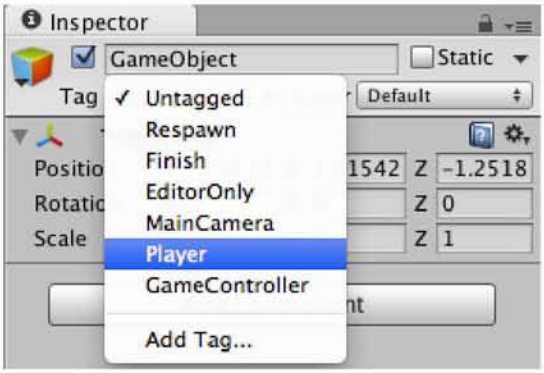


Εικόνα 19: Λίστα επιπέδων σχεδίασης.

2.3.6 Ετικέτες και Κατηγοριοποίηση Αντικειμένων

Οι ετικέτες (Tags) είναι λέξεις-κλειδιά οι οποίες κατηγοριοποιούν ένα αντικείμενο είτε ένα στιγμιότυπο προκατασκευασμένου αντικειμένου με σκοπό να το ομαδοποιήσουν με άλλα αντικείμενα που έχουν παρόμοια λειτουργικότητα. Η χρήση ετικετών δεν είναι τόσο χρήσιμη κατά την όψη σκηνής αλλά αποτελεί ένα πολύ χρήσιμο εργαλείο για τους προγραμματιστές διότι με τις ετικέτες έχουν πρόσβαση σε ένα κοινό πεδίο αντικειμένων στο οποίο μπορούν με κώδικα να καθορίσουν συμπεριφορές και αλληλεπιδράσεις χωρίς να αλλάξουν την ιεραρχία σκηνής δηλαδή χωρίς να είναι απαραίτητο να δημιουργήσουν ένα νέο γονικό αντικείμενο για κάθε αντικείμενο που είναι μέλος ομάδας.[21] Για παράδειγμα, η Unity έχει κάποιες προκαθορισμένες ετικέτες όπως Player οι οποίες θα μπορούσαν να χρησιμοποιηθούν ως εξής:

Object **obj.tag** = "Player"



Εικόνα 20: Ανάθεση ετικετών σε αντικείμενα

2.3.7 Διαχείριση πολλαπλών Σκηνών

Για τη δημιουργία ενός ηλεκτρονικού παιχνιδιού χρειάζεται να γίνεται παράλληλη φόρτωση σκηνών και σε άλλες περιπτώσεις πολλές εναλλαγές μεταξύ των σκηνών. Οι σκηνές οι οποίες φορτώνονται στην εργασία τοποθετούνται σε ειδικό παράρτημα της όψης ιεραρχίας, στο διαχωριστή σκηνών (Scene Divider), που δείχνει τη διάρθρωση της κάθε σκηνής και παρέχει τις δυνατότητες που φαίνονται στον παρακάτω πίνακα.

Λειτουργία Scene Divider	Επεξήγηση Λειτουργίας
Set Active Scene	Ενεργοποίηση σκηνής. Σε αυτή γίνεται δημιουργία νέων αντικειμένων. Πρέπει πάντα να υπάρχει μία σκηνή ενεργή
Save Scene	Αποθήκευση αλλαγών μόνο στην ενεργή σκηνή
Save Scene As	Αποθήκευση της σκηνής ως νέο στοιχείο προσάρτησης (Scene Asset)
Save All	Μαζική αποθήκευση αλλαγών σε όλες τις σκηνές.
Unload Scene	Η σκηνή απενεργοποιείται αλλά δεν αφαιρείται από την όψη ιεραρχίας.
Remove Scene	Αφαίρεση της σκηνής από την όψη ιεραρχίας
Select Scene Asset	Επιλογή στοιχείου προσάρτησης σκηνής από τον περιηγητή εργασίας.

GameObject	Δημιουργία αντικειμένων στην επιλεγμένη σκηνή.

Πίνακας 1 **:** Λίστα χαρακτηριστικών διαχωριστή σκηνών

Αξίζει να σημειωθεί ότι με τη μετάβαση από μία σκηνή στην επόμενη, όλα τα αντικείμενα παιχνιδιού καταστρέφονται.[22] Αν οι σχεδιαστές παιχνιδιών επιθυμούν να διατηρήσουν ορισμένα αντικείμενα στις θέσεις που αυτά κατασκευάστηκαν αρχικά, κατά τη φόρτωση της νέας σκηνής αρκεί να καλέσουν τη μέθοδο DontDestroyOnLoad ως εξής:

DontDestroyOnLoad(transform.gameObject);

2.4 Προγραμματισμός παιχνιδιών με C#

Στην πλατφόρμα Unity ο προγραμματισμός παιχνιδιών γίνεται σύμφωνα με τα πρότυπα του αντικειμενοστραφούς προγραμματισμού. Ιδιαίτερη έμφαση δίνεται στον καθορισμό συμπεριφορών των αντικείμενων με κώδικα ο οποίος τρέχει ανεξάρτητα από το πρόγραμμα του παιχνιδιού. Συνεπώς γλώσσες scripting όπως C#, JavaScript και Boo χρησιμοποιούνται για την ανάπτυξη ευέλικτων Unity scripts. Σε αυτή την ενότητα αλλά και στο υπόλοιπο της εργασίας θα επικεντρωθούμε στον προγραμματισμό σε C# λόγω μεγαλύτερης εξοικείωσης με τη γλώσσα γενικού-σκοπού C. Βέβαια, για τη δημιουργία ενός παιχνιδιού μπορούν να συνυπάρχουν scripts και από τις τρείς υποστηριζόμενες γλώσσες προγραμματισμού. Για την επεξεργασία των scripts η Unity υποστηρίζει αλληλεπιδράσεις με το Visual Studio, το MonoDevelop αλλά και πολλά άλλα περιβάλλοντα ανάπτυξης προσφέροντας άμεση επικοινωνία και αποσφαλμάτωση του κώδικα από το τερματικό της .

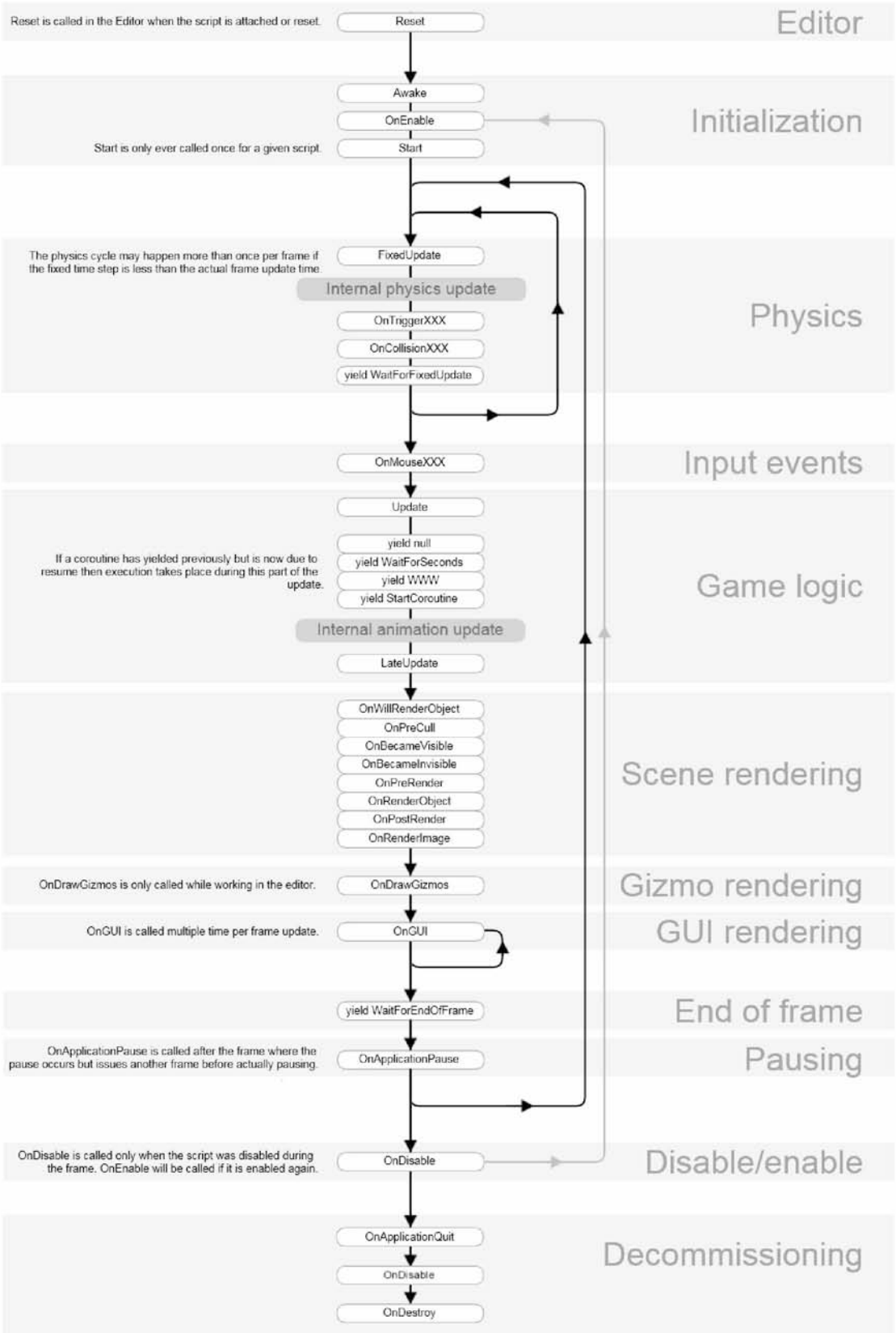
Στην επόμενη εικόνα φαίνεται η δομή ενός απλού C# Script που κωδικοποιεί μία συμπεριφορά αντικειμένου. Η μέθοδος Start() καθορίζει την αρχικοποίηση της συμπεριφοράς του αντικειμένου, συγκεκριμένα αποτελεί τον κώδικα ο οποίος εκτελείται στο πρώτο στιγμιότυπο που σχεδιάζεται η σκηνή. Η μέθοδος Update() καθορίζει τις ενέργειες που θα συμβούν κατά την ανανέωση του στιγμιότυπου σκηνής και αναφέρεται κυρίως σε πράξεις που απαιτούν είσοδο χρήστη.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

Εικόνα 21: Πρότυπο C# Script

Βέβαια, η δομή ενός C# script είναι πιο πολύπλοκη διότι κάθε συμπεριφορά χρειάζεται να κωδικοποιηθεί σύμφωνα με πολλές παραμέτρους εκτέλεσης. Επομένως είναι αναγκαίο να παρουσιάσουμε το πλήρες πλαίσιο ιεραρχίας εκτέλεσης συναρτήσεων μέσα σε ένα C# script.

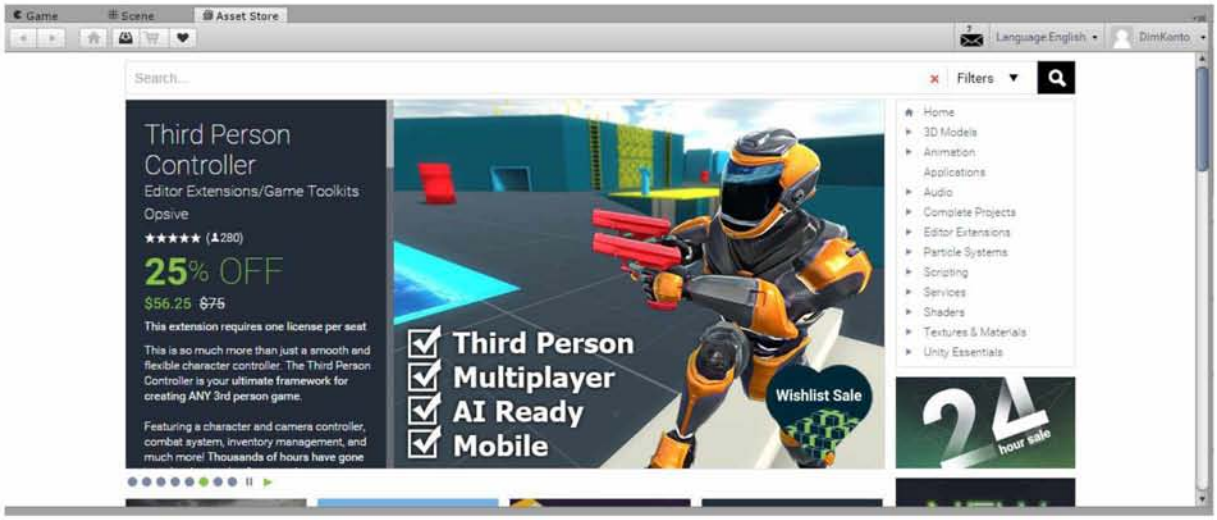
Αρχικά γίνεται εκτέλεση των μεθόδων που αφορούν αρχικοποιήσεις και εκκαθάριση του περιβάλλοντος σχεδίασης. Ακολούθως, γίνεται εκτέλεση μεθόδων που αφορούν εφαρμογή νόμων φυσικής στη σκηνή, όπως εφαρμογή συγκρούσεων. Οι μέθοδοι φυσικής δε χρειάζεται να εκτελούνται μόνο σε ένα στιγμιότυπο αλλά μπορούν να έχουν συνεχή εκτέλεση στη σκηνή. Οι επόμενες μέθοδοι αφορούν την είσοδο χρήστη όπως και την απόδοση σκηνής. Αυτές οι μέθοδοι είναι συνδεδεμένες με το χρόνο εκτέλεσης αλλά και με το χρόνο με τον οποίο συμβαίνουν ορισμένα γεγονότα στη σκηνή. Τέλος έχουμε δύο κατηγορίες μεθόδων που σχετίζονται με τη διατήρηση κατάστασης σκηνής αλλά και με την απελευθέρωση και καταστροφή σκηνής. Οι μέθοδοι διατήρησης οδηγούν σε εκ νέου εκτέλεση συναρτήσεων υψηλότερου ιεραρχικού επιπέδου ενώ οι μέθοδοι απελευθέρωσης και καταστροφής οδηγούν στην αφαίρεση αντικειμένων από τη σκηνή και στον τερματισμό της εφαρμογής.[23]



Εικόνα 22: Pipeline Μεθόδων σε ένα C# script της Unity

2.5 Κατάστημα στοιχείων παιχνιδιού

Ένα από τα πιο ισχυρά εργαλεία που έχουν στη διάθεσή τους οι προγραμματιστές παιχνιδιών είναι η πρόσβαση στο κατάστημα της Unity. Σε αυτό το κατάστημα γίνεται μεταφόρτωση αντικειμένων αλλά και επιπλέον εργαλείων από πολλούς καλλιτέχνες και προγραμματιστές έτσι ώστε κάθε παιχνίδι να μπορεί να επεκταθεί και να γίνεται πιο ελκυστικό με τις προσθήκες αυτών των δημιουργιών. Η κοινότητα της Unity έχει ένα πολύ σημαντικό ρόλο στη διαμόρφωση του καταστήματος καθώς η αλληλεπίδραση με το κατάστημα συνδέεται με το λογαριασμό χρήστη Unity συμβάλλοντας στην αξιολόγηση του περιεχομένου του .Το κατάστημα περιέχει δωρεάν αντικείμενα παιχνιδιού όπως και αντικείμενα επί πληρωμή, πληροφορώντας τους σχεδιαστές παιχνιδιών για τις υποστηριζόμενες εκδόσεις Unity όπως και εκτενείς περιγραφές του περιεχομένου. Τέλος, εκτός από έτοιμα αντικείμενα μέσα στο κατάστημα φιλοξενούνται επίσης οδηγοί και πρότυπα έτσι ώστε να βοηθηθούν νέοι δημιουργοί.[24]



Εικόνα 23: Επισκόπηση καταστήματος Unity μέσα από το περιβάλλον σχεδίασης

3. Επισκόπηση του Unity Networking

Η υλοποίηση διαδικτυακού παιχνιδιού πολλών παικτών είναι μία διαδικασία η οποία από τη φύση της είναι πολύπλοκη. Συχνά, υπάρχουν αρκετά προβλήματα και δυσκολίες τα οποία σχετίζονται με τον συγχρονισμό και την επικοινωνία των παικτών ,όταν αυτοί συνδέονται από απομακρυσμένες τοποθεσίες. Σε αυτή την ενότητα θα παρουσιάσουμε τη λύση που προσφέρει η Unity για τη δημιουργία διαδικτυακών παιχνιδιών, το Unity Networking. Η εισαγωγή στις βασικές αρχές λειτουργίας του Unity Networking, του High Level API καθώς και της οργάνωσής τους σε θεμελιώδεις κλάσεις και μεθόδους είναι πολύ σημαντική για την κατανόηση των δικτυακών χαρακτηριστικών που εφαρμόζουμε στο παιχνίδι που αναπτύσσουμε.

3.1 Τί είναι και τί προσφέρει το Unity Networking

Το Unity Networking είναι ένα σύνολο ενσωματωμένων δικτυακών εργαλείων το οποίο εμφανίστηκε κατά την έκδοση 5.1 και σήμερα βρίσκεται σε μία σταθερή κατάσταση ,με ελάχιστα σφάλματα, στην έκδοση 5.5.1 της Unity. Το Unity Networking προσφέρει πρόσβαση σε δύο διεπαφές προγραμματισμού δικτυακών εφαρμογών, το High Level API και το Transport Layer API, οι οποίες παρέχουν διαφορετικές λύσεις υλοποίησης παιχνιδιού πολλών παικτών.

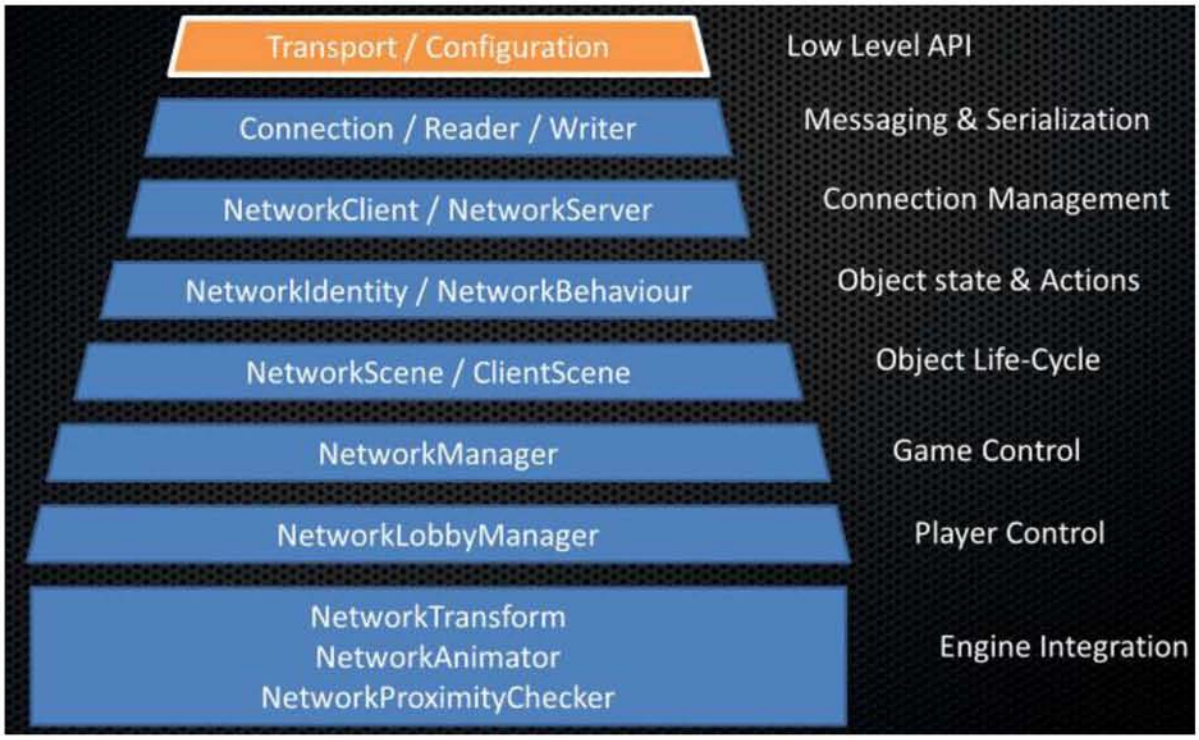
Το High Level API προσφέρει μεγάλη ευελιξία υλοποίησης δικτυακών χαρακτηριστικών στοχεύοντας σε εύκολο παιχνίδι μεταξύ απομακρυσμένων χρηστών. Από την άλλη πλευρά , το Transport Layer API είναι μία διεπαφή χαμηλού επιπέδου η οποία δίνει τη δυνατότητα στον προγραμματισμό παιχνιδιού για μία προσαρμοσμένη τοπολογία δικτύου στην οποία η επικοινωνία γίνεται μέσω του πρωτοκόλλου UDP είτε μέσω WebSockets για παιχνίδια που υποστηρίζουν WebGL. Σε αυτή την εργασία θα επικεντρωθούμε στο High Level API, το

οποίο θα αναλύσουμε σε επόμενη υποενότητα, διότι με τη χρήση αυτού η διεξαγωγή πειραμάτων γίνεται πολύ πιο εύκολη καθώς ο κώδικας που αναλαμβάνει τη δικτύωση είναι πανομοιότυπος για προγραμματισμό σε τοπικό αλλά και σε απομακρυσμένο δίκτυο χωρίς να δημιουργεί επιπλοκές στην εξαγωγή συμπερασμάτων.[25]

3.2 Το High Level API

Το High Level API (HLAPI) αποτελεί ένα ολοκληρωμένο σύστημα εφαρμογής δικτυακών δυνατοτήτων σε ηλεκτρονικά παιχνίδια το οποίο έχει υλοποιηθεί πάνω από το Transport Layer API (TLAPI) για επικοινωνία σε πραγματικό χρόνο. Σε αντίθεση με το TLAPI, το HLAPI δε μπορεί να προσαρμοστεί σε οποιαδήποτε τοπολογία λόγω της ανώτερης θέσης του στην ιεραρχία αλλά ακολουθεί ένα σύστημα προστακτικών εξυπηρετητών (authoritative server system). Σύμφωνα με αυτό το σύστημα, ένας εξυπηρετητής περιέχει το κυρίως σύνολο των δεδομένων, το οποίο ανανεώνεται με κάθε αλλαγή που πραγματοποιούν τα προγράμματα-πελάτες. Αυτή η πρακτική διαφέρει από τη μερική αποθήκευση των δεδομένων που αναζητούν οι χρήστες σε cache η οποία συνηθίζεται σε μη προστακτικά συστήματα. Επιπροσθέτως, το HLAPI επιτρέπει σε ένα χρήστη να αναλαμβάνει ταυτόχρονα το ρόλο του πελάτη και του εξυπηρετητή έτσι ώστε να μη χρειάζεται να υπάρχει ένας κεντρικός εξυπηρετητής για όλες τις ανάγκες του δικτύου. Η ενσωμάτωση του HLAPI στις διαδικτυακές υπηρεσίες της Unity επιτρέπει παιχνίδι πολλαπλών παικτών με χρήση άμεσων συνδέσεων χωρίς να αυξάνει το επίπεδο πολυπλοκότητας για τους προγραμματιστές.

Το High Level API βρίσκεται στο χώρο ονομάτων UnityEngine.Networking έτσι ώστε να μπορεί εύκολα να επαναχρησιμοποιείται σε πολλές εργασίες παρέχοντας διαχείριση μηνυμάτων, αποδοτική μετατροπή αντικειμένων σε bytes, διαχείριση κατανεμημένων αντικειμένων, συγχρονισμό καταστάσεων παιχνιδιού αλλά και διαχείριση αρχιτεκτονικής του δικτύου μέσα από κλάσεις που απευθύνονται αποκλειστικά σε εξυπηρετητή είτε σε προγράμμα-πελάτη. Η διεπαφή αυτή χωρίζεται σε επίπεδα τα οποία χρησιμεύουν σε ένα ενιαίο έλεγχο της επικοινωνίας των παικτών στο δίκτυο αλλά και της αλληλεπίδρασης με το περιβάλλον της πλατφόρμας. Ειδικότερα, παρουσιάζουμε στην Εικόνα 24 την ιεραρχική δομή των επιπέδων που αποτελούν το High Level API.[26]



Εικόνα 24: Ιεραρχική δομή επιπέδων του High Level API

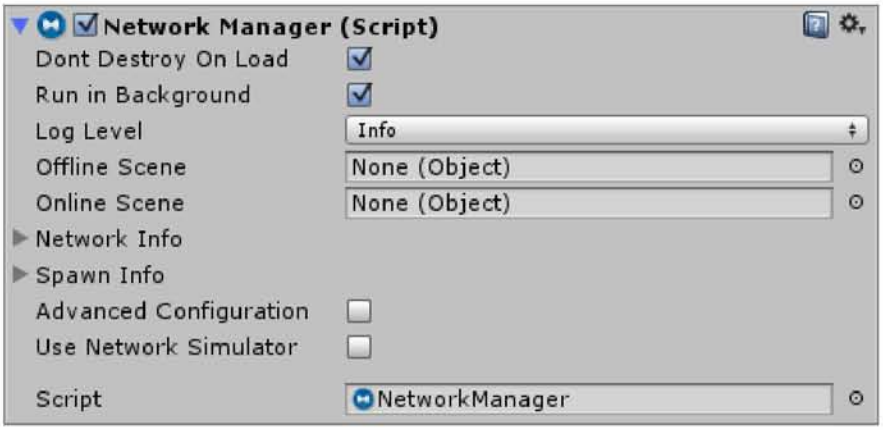
3.3 Επισκόπηση του Network Manager

Το στοιχείο αντικειμένου Network Manager είναι το δομικό μέρος δικτύωσης ενός παιχνιδιού και είναι πλήρως προγραμματισμένο μέσω του High Level API έτσι ώστε οι προγραμματιστές να έχουν εύκολα πρόσβαση σε αυτό, προσαρμόζοντάς το στις ανάγκες του παιχνιδιού τους και αλλάζοντας τις παραμέτρους του με κώδικα. Στον Πίνακα 2 παρουσιάζουμε τα πιο σημαντικά χαρακτηριστικά του Network Manager.

Χαρακτηριστικά Network Manager	**Λειτουργία**
1) Διαχείριση καταστάσεων παιχνιδιού	Εξυπηρετητής/Πελάτης/Διοργανωτής
2) Διαχείριση δημιουργίας αντικειμένων	Δημιουργία: Τυχαία/Round Robin
3) Διαχείριση σκηνής	Offline/Online
4) Προβολή πληροφοριών αποσφαλαμάτωσης	Κατάσταση Συνδέσεων
5)Διοργάνωση δικτυακών παιχνιδιών	Create/Host/Join/Disconnect
6) Επαναπρογραμματισμός λειτουργιών αντικειμένων	Προσθήκη και προσαρμογή παίκτη

Πίνακας 2:Λίστα χαρακτηριστικών Network Manager

Η δημιουργία μίας σκηνής η οποία έχει δικτυακές ικανότητες προϋποθέτει την ύπαρξη ενός αντικειμένου το οποίο έχει το δομικό στοιχείο του Network Manager. Η πρόσβαση στον Network Manager για μερικές από τις πιο απλές λειτουργίες του γίνεται από την όψη παρατήρησης όπου μπορούμε να δούμε κάποιες από τις public μεταβλητές του όπως και κάποιες από τις μεθόδους του. Στην Εικόνα 25 παρουσιάζουμε το δομικό στοιχείο του Network Manager.



Εικόνα 25: Network Manager στην όψη παρατήρησης.

Η αλληλεπίδραση με τον Network Manager δε σταματά στην όψη παρατήρησης διότι αυτό το στοιχείο είναι ορατό και από το παιχνίδι καθώς παρέχει μία διεπαφή χρήστη για τη δημιουργία τοπικού παιχνιδιού όπως και για τη διοργάνωση διαδικτυακού παιχνιδιού μέσω του Match Maker (Εικόνα 26). Αυτή η διεπαφή ωστόσο δεν είναι λειτουργική για το χρήστη ,συνεπώς σχεδόν σε όλους τους τύπους παιχνιδιού χρειάζεται επαναπρογραμματισμός του στοιχείου Network Manager. Σε αυτή την εργασία το ενδιαφέρον μας επικεντρώνεται στη διοργάνωση διαδικτυακών παιχνιδιών, επομένως θα προγραμματίσουμε με βάση τον Match Maker και όχι τον LAN Host και LAN Client δημιουργώντας ένα νέο σύστημα διοργάνωσης.[27]

(i) (ii)

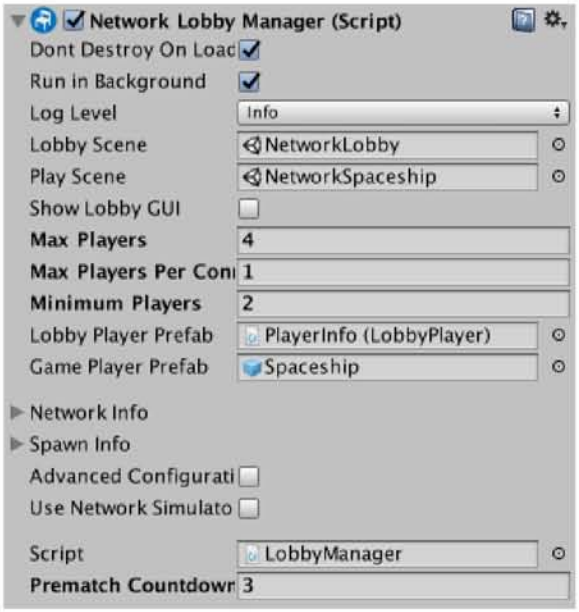
Εικόνα 26:Διεπαφή χρήστη για Network Manager(i) και το μενού του Match Maker(ii)

3.4 Το δομικό στοιχείο Network Lobby Manager

Ένα πολύ σημαντικό στοιχείο του Unity Networking είναι ο διαχειριστής διοργάνωσης παιχνιδιών (Network Lobby Manager) ο οποίος αναλαμβάνει να καθορίσει τις παραμέτρους παιχνιδιού πριν την είσοδο ενός νέου παίκτη στη σκηνή. Στις αρμοδιότητες αυτού του στοιχείου συμπεριλαμβάνονται η αυτόματη εκκίνηση όταν όλοι οι παίκτες είναι έτοιμοι, ο καθορισμός ενός μεγίστου αριθμού παικτών, η δυνατότητα πρόσβασης περισσότερων παικτών από ένα πρόγραμμα-πελάτη αλλά και η δημιουργία μενού επιλογών που αφορούν το παιχνίδι.

Ο Network Lobby Manager χρειάζεται δύο τύπους αντικειμένων έτσι ώστε να λειτουργήσει σωστά, το αντικείμενο Lobby Player και το αντικείμενο Game Player. Το αντικείμενο Lobby Player ανήκει σε κάθε παίκτη που συνδέεται στο παιχνίδι και παραμένει ενεργό μέχρι την αποσύνδεση του παίκτη. Τα στιγμιότυπα του Lobby Player είναι ξεχωριστά για κάθε παίκτη και αναλαμβάνουν την εκτέλεση εντολών πριν το παιχνίδι. Από την άλλη πλευρά, το αντικείμενο Game Player αναλαμβάνει την εκτέλεση εντολών κατά την εξέλιξη του παιχνιδιού και καταστρέφεται όταν ένας παίκτης συνδέεται ξανά στο παιχνίδι.

Συνολικά, το στοιχείο Network Lobby Manager είναι μία πολύ καλή προσθήκη στο Unity Networking και αυτοματοποιεί τις διαδικασίες που χρειάζονται να εκτελεστούν για να ξεκινήσει και να τελειώσει ένα διαδικτυακό παιχνίδι. Ωστόσο, στην παρούσα έκδοσή του για την Unity 5.5 είναι ένα αρκετά περιορισμένο εργαλείο καθώς η συμπεριφορά του δεν είναι πάντοτε η επιθυμητή συμπεριφορά που οι περισσότεροι σχεδιαστές παιχνιδιών θα ήθελαν να υλοποιήσουν. Επομένως, το στοιχείο Network Lobby Manager δίνει ενδείξεις προς τη σωστή κατεύθυνση αλλά η απλότητά του ωθεί τους προγραμματιστές σε ανάπτυξη διαφορετικών διαχειριστών διοργάνωσης που βασίζονται σε αυτό το στοιχείο. Στην Εικόνα 27, παρουσιάζεται το στοιχείο του διαχειριστή διοργάνωσης παιχνιδιού με τις βασικές λειτουργίες του.[28]

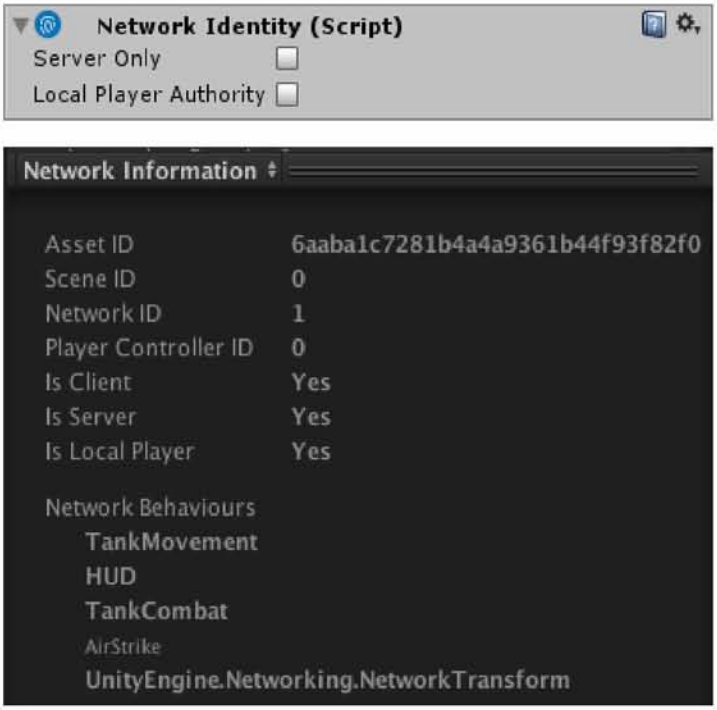


Εικόνα 27: Στοιχείο Network Lobby Manager

3.5 Στοιχεία Δικτύωσης Αντικειμένων

Η κατασκευή αντικειμένων παιχνιδιού όπως έχει περιγραφεί μέχρι τώρα αποτελεί απλή διαδικασία. Ωστόσο, με την ενσωμάτωση δικτυακών δυνατοτήτων η διαδικασία γίνεται πιο πολύπλοκη καθώς αρκετά δομικά στοιχεία δικτύωσης χρειάζεται να προσαρτηθούν στα αντικείμενα έτσι ώστε αυτά να συμπεριφέρονται σωστά στο περιβάλλον του δικτύου. Σε αυτή την ενότητα παρουσιάζουμε τα βασικότερα στοιχεία τα οποία αποτελούν ξεχωριστές κλάσεις εντός του Unity Networking.

Το πιο βασικό στοιχείο δικτύωσης είναι το Network Identity (Εικόνα 28) καθώς δίνει ένα ξεχωριστό αναγνωριστικό στοιχείο σε ένα αντικείμενο και καθορίζει την εξουσιοδότηση του αντικειμένου είτε αποκλειστικά στον εξυπηρετητή, είτε μόνο στο στιγμιότυπο του τοπικού παίκτη που κατέχει το αντικείμενο. Συμπληρωματικά, το στοιχείο Network Start Position καθορίζει την αρχική θέση και περιστροφή ενός δικτυακού αντικειμένου ενώ το στοιχείο Network Transform συγχρονίζει τις κινήσεις ενός αντικειμένου σύμφωνα με την εξουσιοδότηση που έχει τεθεί με το Network Identity ώστε η κίνηση να μεταδίδεται σωστά στους υπόλοιπους παίκτες.



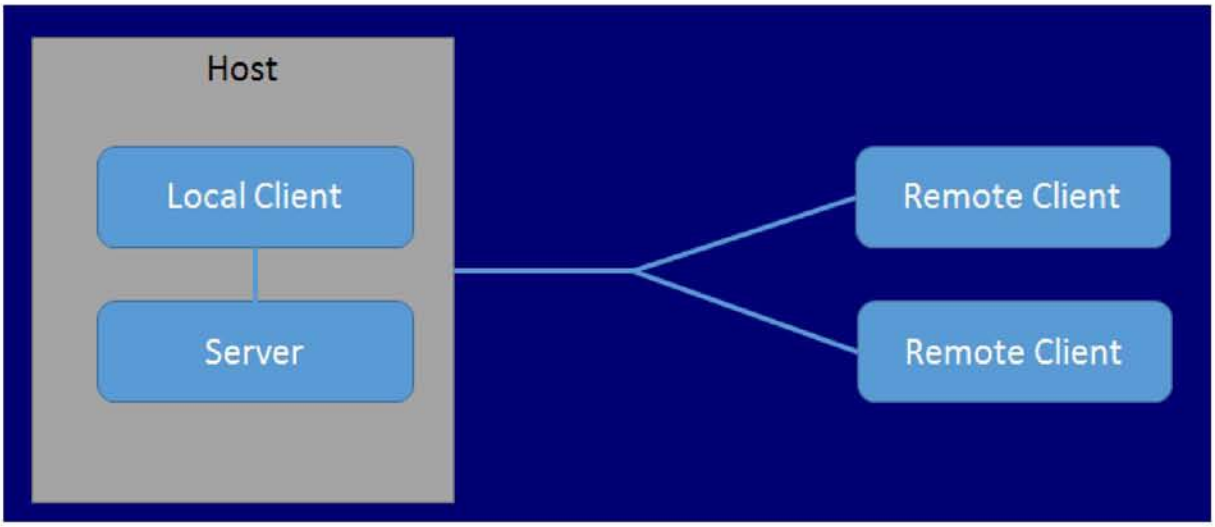
Εικόνα 28: Network Identity και δικτυακές πληροφορίες αντικειμένου παιχνιδιού

Εκτός από τις βασικές λειτουργίες της δημιουργίας και κίνησης αντικειμένου, δίνεται η δυνατότητα δικτύωσης πιο πολύπλοκων συμπεριφορών όπως φυσικών αλληλεπιδράσεων και σκηνοθετημένων κινήσεων. Το στοιχείο Network Proximity Checker ελέγχει φυσικές αλληλεπιδράσεις όπως συγκρούσεις, σύμφωνα με την απόσταση των αντικειμένων. Επίσης, το στοιχείο Network Animator μπορεί να καταγράψει σύνολα κινήσεων τα οποία χρειάζονται να μεταδοθούν επαναληπτικά σε συγκεκριμένα χρονικά διαστήματα.

Τέλος, χρειάζεται να γίνει αναφορά στα στοιχεία που αναλαμβάνουν την επικοινωνία και μετάδοση πληροφοριών μεταξύ των αντικειμένων. Αρχικά, το στοιχείο Network Discovery αναλαμβάνει να βρει στιγμιότυπα της εφαρμογής του παιχνιδιού στο δίκτυο και στη συνέχεια τα στοιχεία Network Reader και Network Writer χρησιμοποιούνται για τη σειριοποίηση των αντικειμένων σε bytes και τη μεταφορά τους ως μηνύματα. Εδώ ολοκληρώνεται η επισκόπηση των δικτυακών στοιχείων αντικειμένων που δείχνουν πως η συμβατική αλληλεπίδραση ενός offline παιχνιδιού μπορεί σχετικά εύκολα να μετατραπεί στην αντίστοιχη online εκδοχή της.[29]

3.6 Καθορισμός ρόλων και αρμοδιοτήτων

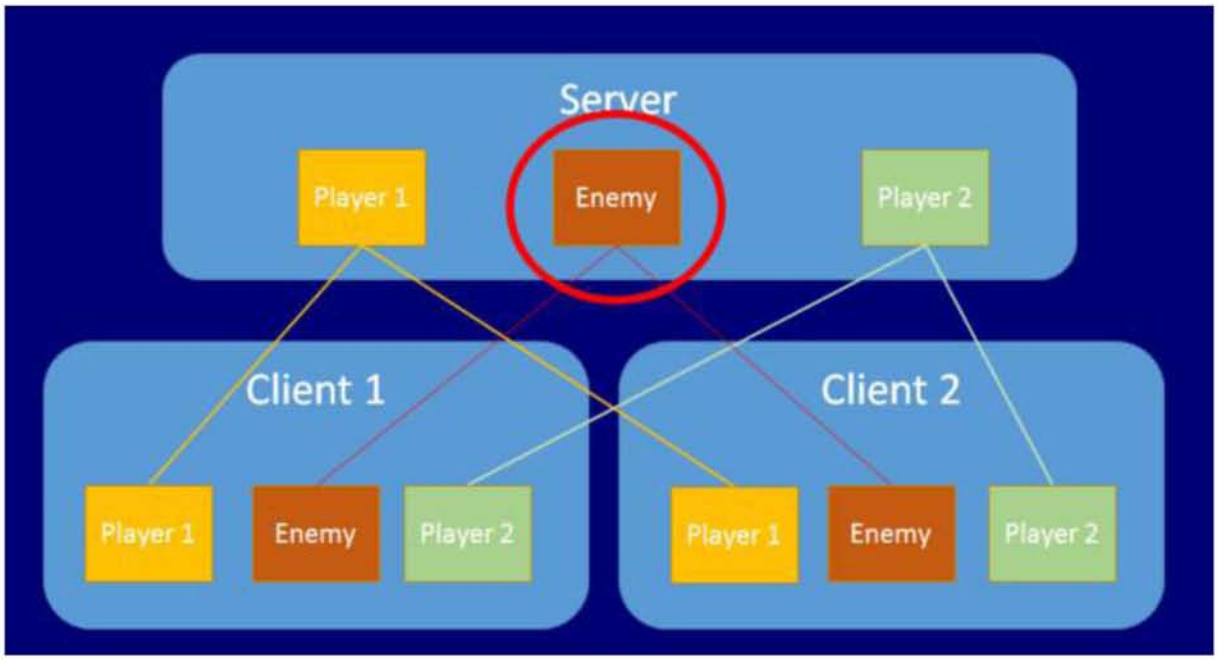
Το στοιχείο Network Identity μας βοήθησε να κατανοήσουμε ότι τα αντικείμενα και η χρήση τους χρειάζεται να κατηγοριοποιηθεί και να διαμοιρασθεί μέσα στο δίκτυο σε χρήστες διαφορετικού τύπου. Σε αυτή την ενότητα θα αναλύσουμε τους ρόλους και τις αρμοδιότητες που έχουν οι διαφορετικοί τύποι χρηστών ενός διαδικτυακού παιχνιδιού έτσι ώστε να διαφανεί η ιεραρχική δομή τους. Η βασική ιδέα του Unity Networking στηρίζεται στην αντιμετώπιση του εξυπηρετητή ως τοπικού προγράμματος-πελάτη. Η συνύπαρξη αυτών των δύο οντοτήτων στην ίδια διεργασία δημιουργούν την οντότητα Host (Εικόνα 29).Με αυτό τον τρόπο το τοπικό πρόγραμμα-πελάτη που εκτελείται στον εξυπηρετητή μπορεί να επικοινωνεί μαζί του με άμεσες κλήσεις μεθόδων και ουρές μηνυμάτων καθώς αυτές οι οντότητες βρίσκονται στο ίδιο δίκτυο. Η οντότητα Host αναλαμβάνει τη δημιουργία αντικειμένων στο δίκτυο.



Εικόνα 29: Ο ενοποιημένος ρόλος του εξυπηρετητή-τοπικού χρήστη

Μία εξίσου βασική έννοια προς ανάλυση είναι η έννοια του τοπικού παίκτη. Το Unity Networking επιτρέπει δρομολόγηση εντολών οι οποίες επηρεάζουν αντικείμενα τα οποία ανήκουν αποκλειστικά στον παίκτη που κατέχει αυτά τα αντικείμενα. Επομένως, κάθε παίκτης διαχειρίζεται την προβολή των αντικειμένων που ανήκουν σε αυτόν σε όλα τα στιγμιότυπά του μέσα στο δίκτυο ανεξαρτήτως αν αυτά τα στιγμιότυπα προβάλλονται στον εξυπηρετητή ή σε άλλους παίκτες. Αντίθετα με αυτή τη στρατηγική, τα αντικείμενα τα οποία δεν ελέγχονται από παίκτες όπως τα αντικείμενα που συμβολίζουν αντιπάλους συνήθως ελέγχονται και λαμβάνουν εντολές αποκλειστικά από τον εξυπηρετητή διότι για αυτά τα αντικείμενα δεν γίνεται αντιστοίχιση με κάποιο πρόγραμμα-πελάτη όπως γίνεται με τους

παίκτες (Εικόνα 30). Αξίζει ακόμη να αναφερθεί ότι στις τελευταίες εκδόσεις της Unity δίνεται η δυνατότητα και στα προγράμματα-πελάτη να καθορίζουν εντολές για αντικείμενα που δεν ελέγχονται από τον παίκτη όμως αυτή η πρακτική ίσως δυσχεραίνει την εμπειρία παιχνιδιού σε μεγάλες αποστάσεις καθώς οι μέσοι παίκτες δε συνηθίζεται να έχουν μεγάλες ταχύτητες μετάδοσης και ο επιπλέον φόρτος πιθανότατα θα οδηγούσε σε επιπλέον καθυστερήσεις ανανέωσης κατάστασης ώστε να αντικατοπτρίζονται οι αλλαγές στα αντικείμενα. [30]



Εικόνα 30: Ιεραρχικό σχήμα καθορισμού αρμοδιοτήτων

3.7 Συγχρονισμός Καταστάσεων Παιχνιδιού

Η ροή ενός διαδικτυακού παιχνιδιού συμβάλλει σημαντικά στην εμπειρία που αποκομίζουν οι παίκτες. Η ομαλότητα αυτής της ροής προϋποθέτει συνεχή συγχρονισμό των καταστάσεων του παιχνιδιού από τον εξυπηρετητή προς τους απομακρυσμένους χρήστες. Καθώς η οντότητα του τοπικού παίκτη συμπίπτει με αυτή του εξυπηρετητή, δε χρειάζεται πλεονάζουσα πληροφορία να συγχρονιστεί στον τοπικό παίκτη. Ωστόσο, κάθε μεταβλητή η οποία πρέπει να συγχρονίζεται εντός του δικτύου χρειάζεται το αναγνωριστικό SyncVar το οποίο είναι μέλος της κλάσης Network Behaviour. Στο επόμενο script παρουσιάζεται ενδεικτικό παράδειγμα χρήσης του SyncVar για συγχρονισμό ζωής παίκτη.

```
class Player : NetworkBehaviour
{

    [SyncVar]
    int health;

    public void TakeDamage(int amount)
    {
        if (!isServer)
            return;

        health -= amount;
    }
}
```

Εικόνα 31 : Script συγχρονισμού μεταβλητής

Το προηγούμενο παράδειγμα περιορίστηκε στο συγχρονισμό μίας μόνο μεταβλητής αλλά υπάρχουν και λύσεις για πιο πολύπλοκο συγχρονισμό ,οι οποίες όμως δεν χρησιμοποιούνται ευρύτατα. Για τον μαζικό συγχρονισμό μεταβλητών ενός τύπου χρησιμοποιούνται λίστες οι οποίες ονομάζονται SyncLists ενώ για νέους σύνθετους τύπους δεδομένων χρησιμοποιούνται SyncListStructs. Επίσης για πιο προχωρημένο συγχρονισμό δεδομένων μπορούν να χρησιμοποιηθούν οι συναρτήσεις OnSerialize και OnDeserialize οι οποίες τροφοδοτούνται από τα στοιχεία Network Writer και Network Reader αντίστοιχα ώστε να μεταφέρουν πακέτα πληροφορίας τα οποία εμπεριέχουν ευπροσάρμοστες μάσκες bit ελέγχου για την ανίχνευση αλλαγών.[31]

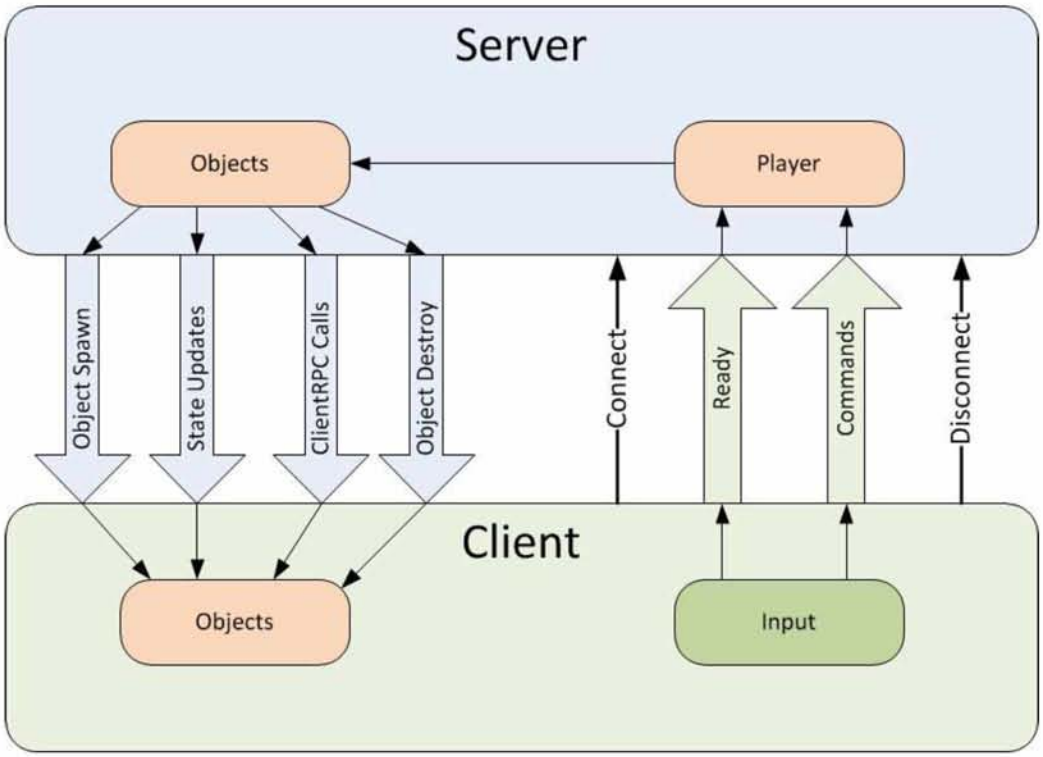
```
public virtual bool OnSerialize(NetworkWriter writer, bool initialState);
public virtual void OnDeSerialize(NetworkReader reader, bool initialState);
```

Εικόνα 32: Ορισμοί συναρτήσεων συγχρονισμού με μάσκες bit ελέγχου.

3.8 Δομή Εκτέλεσης Απομακρυσμένων Ενεργειών

Ένα διαδικτυακό παιχνίδι χρειάζεται μία δομή εκτέλεσης εντολών έτσι ώστε οι μέθοδοι οι οποίοι αναπτύσσονται εντός των scripts να εκτελούνται από τα κατάλληλα μέλη του δικτύου. Η δομή που χρησιμοποιείται από το Unity Networking αποτελείται από απομακρυσμένες ενέργειες οι οποίες ονομάζονται Remote Procedure Calls (RPCs).Αυτές οι ενέργειες χωρίζονται σε δύο υποσύνολα, τις εντολές εξυπηρετητή (Commands) και τις εντολές

προγράμματος-πελάτη (ClientRpc). Στο παρακάτω διάγραμμα φαίνονται τα είδη και οι κατευθύνσεις των εντολών.



Εικόνα 33: Διάγραμμα ροής απομακρυσμένων ενεργειών

Οι εντολές εξυπηρετητή είναι μέθοδοι οι οποίες καλούνται από τα προγράμματα-πελάτη αλλά εκτελούνται στον εξυπηρετητή. Η αποστολή των εντολών αυτών γίνεται από αντικείμενα παίκτη στο πρόγραμμα-πελάτη προς τα αντίστοιχα στιγμιότυπα αντικειμένων παίκτη στον εξυπηρετητή. Σε αυτή την επικοινωνία ισχύει η έννοια της εξουσιοδότησης που αναφέραμε σε προηγούμενη ενότητα, επομένως οι αλλαγές εκτελούνται μόνο στα αντικείμενα τα οποία ανήκουν σε ένα συγκεκριμένο παίκτη έτσι ώστε να διατηρηθεί η ασφάλεια ανάμεσα στους παίκτες. Για τη σήμανση μίας μεθόδου ως εντολής εξυπηρετητή χρειάζεται να έχουμε γράψει το αναγνωριστικό Command πριν τον ορισμό της μεθόδου και να έχουμε συμπεριλάβει το πρόθημα Cmd στο όνομα της μεθόδου.

Από την άλλη πλευρά, οι εντολές προγράμματος-πελάτη καλούνται από στιγμιότυπα αντικειμένων στον εξυπηρετητή και εκτελούνται σε προγράμματα-πελάτη. Σε αυτό το σύνολο των εντολών δεν χρειάζεται κάποια επιπλέον εξουσιοδότηση και δεν υπάρχουν προβλήματα ασφάλειας κώδικα καθώς ο εξυπηρετητής ευθύνεται για τη συμπεριφορά των αντικειμένων μέσω του στοιχείου Network Identity. Επομένως, αυτές οι εντολές εκφράζουν πλήρως την έννοια της ύπαρξης συστήματος προστακτικών εξυπηρετητών που παρουσιάσαμε κατά την

ανάλυση του High Level API. Για τη σήμανση μίας μεθόδου ως εντολής προγράμματος-πελάτη χρειάζεται να έχουμε γράψει το αναγνωριστικό ClientRpc πριν τον ορισμό της μεθόδου και να έχουμε συμπεριλάβει το πρόθημα Rpc στο όνομα της μεθόδου.[32]

3.9 Διαδικτυακές Υπηρεσίες

Το Unity Networking βοηθά νέους προγραμματιστές στην ανάπτυξη διαδικτυακών παιχνιδιών δίνοντάς τους πρόσβαση στην υπηρεσία Multiplayer. Μέσα από αυτή την υπηρεσία οι προγραμματιστές χρειάζεται να καταχωρήσουν το παιχνίδι τους λαμβάνοντας άδεια χρήσης του Match Maker στον κώδικά τους. Μέσα από τη χρήση της υπηρεσίας Multiplayer οι παίκτες έχουν τη δυνατότητα να συνδεθούν μέσω διαδικτύου χωρίς να χρειάζονται μία δημόσια διεύθυνση IP . Επίσης, οι παίκτες μπορούν να οργανώνουν παιχνίδια, να συνδέονται και να αποσυνδέονται με ευκολία καθώς ο φόρτος του δικτύου περνά μέσα από τον εξυπηρετητή αναμετάδοσης της Unity με χρήση νεφοϋπολογιστικής χωρίς να μεταδίδεται απευθείας στα προγράμματα-πελάτη. Με αυτό τον τρόπο παρακάμπτονται προβλήματα που σχετίζονται με firewall και NAT έτσι ώστε οι παίκτες να μπορούν να συνδεθούν από οποιοδήποτε δίκτυο.

Η διοργάνωση παιχνιδιού απλοποιείται αρκετά μέσω κώδικα καθώς παρέχεται πρόσβαση σε ένα σύνολο μεθόδων που ανήκουν στο χώρο ονομάτων UnityEngine.Networking.Match. Ωστόσο, η πρόσβαση στον εξυπηρετητή αναμετάδοσης δεν παρέχεται δωρεάν αλλά αποτελεί συνδρομητική υπηρεσία η οποία καθορίζει τις χρεώσεις σύμφωνα με τον όγκο των δεδομένων που μεταδίδονται και τον αριθμό των παικτών. Βέβαια, η Unity δίνει την ευκαιρία παράκαμψης του εξυπηρετητή αναμετάδοσης και δημιουργίας απευθείας συνδέσεων χωρίς αλλαγές στον κώδικα για όσους θέλουν να χρησιμοποιήσουν τον Match Maker δωρεάν όταν πρόκειται για παιχνίδια τα οποία δεν έχουν κερδοσκοπικό χαρακτήρα. Τη διαδικασία καταχώρησης και οργάνωσης του διαδικτυακού παιχνιδιού επιλέγουμε να την παρουσιάσουμε στην τέταρτη ενότητα όπου θα δείξουμε την εφαρμογή της υπηρεσίας στην δωρεάν εκδοχή της στο παιχνίδι που αναπτύσσουμε.[33]

4. Σχεδίαση και Ανάπτυξη Project Bullet Versus

Σε αυτή την ενότητα παρουσιάζονται αναλυτικά τα στάδια της διαδικασίας σχεδίασης και ανάπτυξης του διαδικτυακού First Person Shooter το οποίο ονομάζουμε Project Bullet Versus. Στις υποενότητες που ακολουθούν γίνεται παρουσίαση της ροής σκέψης που μας οδήγησε στην ανάπτυξη του συγκεκριμένου παιχνιδιού, εκτενής επεξήγηση των εργαλείων που χρησιμοποιήθηκαν όπως επίσης και ανάλυση του κώδικα του παιχνιδιού. Ακόμη, γίνονται αναφορές στις σχεδιαστικές επιλογές οι οποίες οδήγησαν στις συμπεριφορές αντικειμένων εντός του παιχνιδιού καθώς και στην καλλιέργεια του γενικότερου ύφους του.

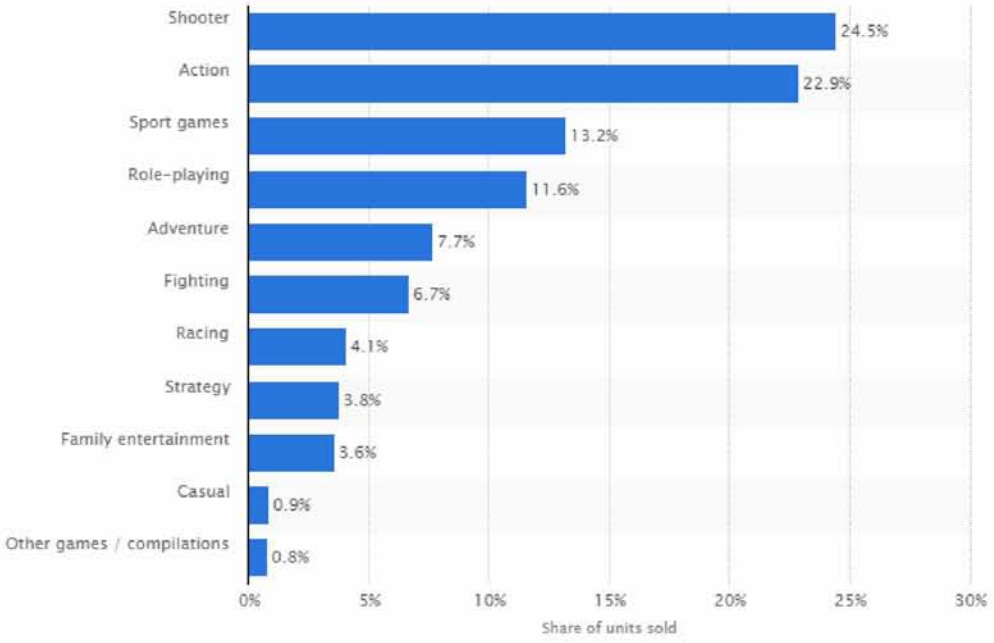
4.1 Η επιλογή του παιχνιδιού

4.1.1 Επιλογή είδους παιχνιδιού

Όπως αναφέραμε και στην εισαγωγή, η πρόοδος των διαδικτυακών παιχνιδιών ωφέλησε ιδιαίτερα την κατηγορία των παιχνιδιών First Person Shooter. Ωστόσο, κάθε προγραμματιστής οφείλει να διερευνήσει τις τάσεις της αγοράς στο σύνολό τους πριν την υλοποίηση του παιχνιδιού. Επομένως, το πρώτο βήμα είναι η εξέταση στατιστικών στοιχείων τα οποία θα οδηγήσουν τη ροή σκέψης μας και θα καθορίσουν την κατεύθυνση του πρωτοτύπου.

Έρευνες δείχνουν ότι κάποιες κατηγορίες παιχνιδιών είναι αρκετά πιο δημοφιλείς από άλλες. Ειδικότερα, έρευνα [34] που διεξήχθη το 2015 στις Ηνωμένες Πολιτείες Αμερικής (Εικόνα 34) έδειξε ότι παιχνίδια των κατηγοριών Shooter και Action είναι αρκετά πιο δημοφιλή από παιχνίδια άλλων κατηγοριών, επιβεβαιώνοντας τη συμβολή των First Person Shooters στις δύο μεγαλύτερες κατηγορίες. Επιπροσθέτως, αυτές οι κατηγορίες λαμβάνουν τις πρώτες

θέσεις σε πολλές ιστοσελίδες αφιερωμένες στην αξιολόγηση παιχνιδιών. Λαμβάνοντας υπόψη αυτά τα στατιστικά στοιχεία όπως επίσης και την προσωπική εμπειρία ενασχόλησης με παιχνίδια κατηγορίας Shooter θεωρήσαμε ότι η ανάπτυξη παιχνιδιού αυτής της κατηγορίας θα οδηγούσε σε αρκετές προγραμματιστικές προκλήσεις οι οποίες θα συνδύαζαν και θα επέκτειναν γνώσεις γραφικών υπολογιστών, δικτύων και βάσεων δεδομένων.



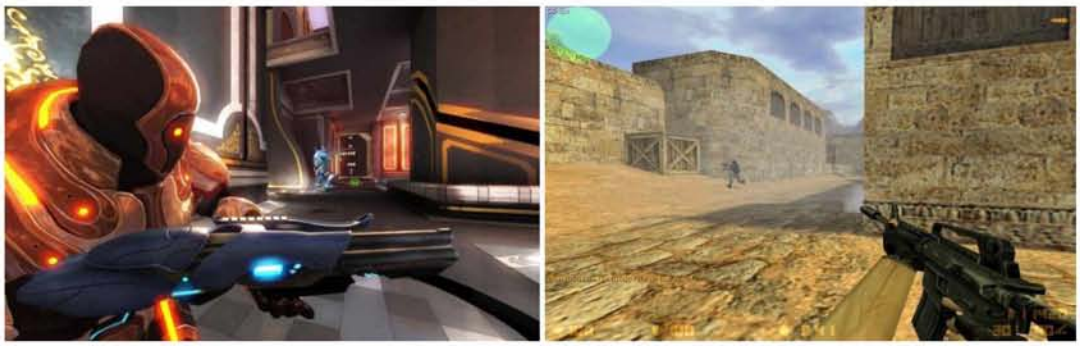
Εικόνα 34: Στατιστικά πωλήσεων ηλεκτρονικών παιχνιδιών ανά κατηγορία στις Ηνωμένες Πολιτείες Αμερικής το 2015

4.1.2 Επιρροές και Θέμα Παιχνιδιού

Τα διαδικτυακά παιχνίδια της κατηγορίας First Person Shooter εμφανίζουν μεγάλη ποικιλία στα χαρακτηριστικά τους καθώς απευθύνονται σε διαφορετικό κοινό, προσφέροντας διαφορετικές εμπειρίες. Παραδείγματα αυτής της ποικιλομορφίας αποτελούν το παιχνίδι Planetside [35], το οποίο βασίζεται σε μάχες ομάδων με σκοπό την κυριαρχία ευρύτερων περιοχών του παιχνιδιού και το παιχνίδι Destiny[36], το οποίο στηρίζεται στην εφαρμογή κοινωνικών χαρακτηριστικών και στην ατομική εξέλιξη των χαρακτήρων. Επομένως, οι κατευθύνσεις στις οποίες θα μπορούσε να κινηθεί η ανάπτυξη του Project Bullet Versus είναι πολυάριθμες και η επιλογή τους οδηγεί σε εφαρμογή διαφορετικών προγραμματιστικών τεχνικών.

Το Project Bullet Versus συνδυάζει τον ανταγωνιστικό χαρακτήρα και το γρήγορο ρυθμό παιχνιδιού των First Person Shooters σε ένα κλειστό, μικρού μεγέθους επίπεδο κατά τα πρότυπα των παιχνιδιών Nexuiz[37] και Counter Strike 1.6[38] (Εικόνα 35). Το παιχνίδι βασίζεται αποκλειστικά σε αγώνες μεταξύ παικτών στο δίκτυο και στοχεύει στην όξυνση των αντανακλαστικών καθώς και στη βελτίωση της ικανότητας των παικτών να στοχεύουν. Το θέμα που ακολουθεί το παιχνίδι βασίζεται σε ένα πρόγραμμα προσομοίωσης στο οποίο μία σειρά ρομπότ, η Jasubot-01, εμπλέκεται σε καταστάσεις μάχης με σκοπό την ανάδειξη του καλύτερου πρωτοτύπου Jasubot. Οι παίκτες καλούνται να ελέγξουν ένα από αυτά τα ρομπότ με στόχο να καταστρέψουν τα αντίπαλα ρομπότ σε αγώνες χωρίς χρονικό όριο, κερδίζοντας τίτλους οι οποίοι αντικατοπτρίζουν την απόδοσή τους στο παιχνίδι. Η σχεδίαση του παιχνιδιού είναι άρρηκτα συνδεδεμένη με το θέμα του καθώς όλα τα αντικείμενα τα οποία απαρτίζουν το παιχνίδι παραπέμπουν σε μία μελλοντική εποχή.

Η λειτουργία παιχνιδιού που επιθυμούμε να αναπτύξουμε ονομάζεται Asymmetric Free-For-All και αποτελεί τροποποιημένη μορφή της λειτουργίας Free-For-All που συναντάται σε πολλά First Person Shooters, κατά την οποία οι παίκτες περιορίζονται σε ένα χρονικό περιθώριο και σε ένα μοναδικό επίπεδο για να επιτύχουν τους ατομικούς στόχους που τους έχουν ανατεθεί. Στο Asymmetric Free-For-All οι παίκτες μπορούν να διοργανώνουν αγώνες και να μεταβαίνουν σε διαφορετικά επίπεδα χωρίς χρονικό περιορισμό με σκοπό να βελτιώσουν τα προσωπικά τους στατιστικά. Τα στατιστικά παίκτη δείχνουν την απόδοση του παίκτη στα πλαίσια του παιχνιδιού αλλά δεν καθορίζουν την προσωρινή νίκη και ήττα. Αντιθέτως, οι παίκτες προσπαθούν συνεχώς να βελτιώσουν τα στατιστικά τους τα οποία αντικατοπτρίζουν τη συνολική πορεία τους στο παιχνίδι.



(i) (ii)

Εικόνα 35: Επιρροές στη σχεδίαση του Project Bullet Versus i) Nexuiz , ii)Counter Strike 1.6

4.2 Επιλογή συστήματος ανάπτυξης

Όπως αναφέραμε και στην εισαγωγή, η πλατφόρμα Unity είναι αρκετά ευέλικτη διότι υποστηρίζει τα περισσότερα συστήματα που είναι ικανά να εκτελέσουν εφαρμογές παιχνιδιών. Σε αυτή την ενότητα χρειάζεται να εξηγήσουμε τους λόγους για τους οποίους επιλέξαμε να αναπτύξουμε το παιχνίδι σε υπολογιστές λειτουργικού συστήματος Windows και όχι σε κάποιο άλλο σύστημα όπως για παράδειγμα Playstation 4 ή κινητό με λειτουργικό Android.

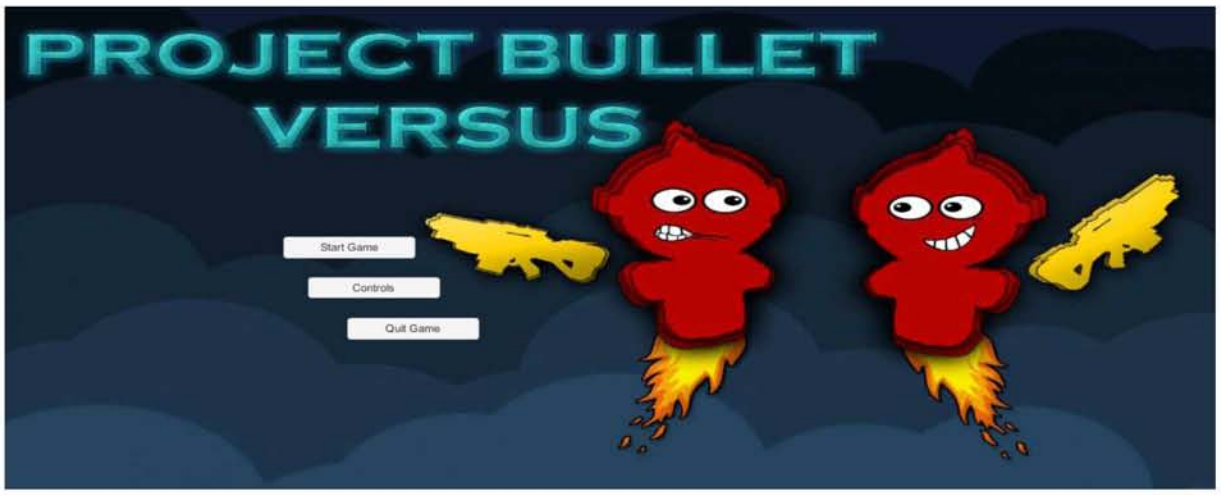
Ο πρώτος βασικός λόγος για την επιλογή του συστήματος υπολογιστή Windows είναι η συσχέτιση του είδους του παιχνιδιού με το υλικό των συστημάτων. Σε ένα First Person Shooter, η ακρίβεια στόχευσης έχει πρωταρχικό ρόλο στην εμπειρία του παιχνιδιού και η χρήση ποντικιού στον υπολογιστή για τον έλεγχο της κάμερας είναι αρκετά πιο ακριβής σε σχέση με την πλειοψηφία των χειριστηρίων και των οθονών αφής άλλων συσκευών. Επιπροσθέτως, οι υπολογιστές επιδέχονται συχνές βελτιώσεις υλικού οι οποίες οδηγούν στην αποδοτικότερη εκτέλεση του παιχνιδιού σε μεγαλύτερες αναλύσεις οθόνης με καλύτερη ποιότητα γραφικών. Τέλος, αξίζει να σημειωθεί ότι υπάρχουν πολύ περισσότερες διαθέσιμες λύσεις δικτύωσης αλλά και τροποποίησης του παιχνιδιού με τη χρήση mod tools, όταν αυτό εκτελείται σε ένα υπολογιστή, συμβάλλοντας στην εξέλιξή του.

4.3 Μενού Έναρξης

4.3.1 Δημιουργία Καμβά

Το μενού έναρξης είναι το πρώτο κομμάτι της διεπαφής χρήστη και στόχος του είναι να προσφέρει στον παίκτη τις βασικές πληροφορίες που χρειάζεται πριν συνδεθεί στο λογαριασμό του. Επιλέγουμε να δημιουργήσουμε μία δισδιάστατη σκηνή η οποία θα περιέχει ένα φόντο σχεδιασμένο με τη χρήση Photoshop, τρία κουμπιά τα οποία θα επιτελούν τις βασικές λειτουργίες του μενού και ένα αντικείμενο ήχου που θα ευθύνεται για την αναπαραγωγή μουσικής στα ηχεία των παικτών. Στις επόμενες Εικόνες παρουσιάζουμε την ιεραρχία του καμβά και μία προεπισκόπηση του μενού.

Εικόνα 36: Ιεραρχία Σχεδίασης Αντικειμένων Καμβά



Εικόνα 37: Προεπισκόπηση Μενού Έναρξης

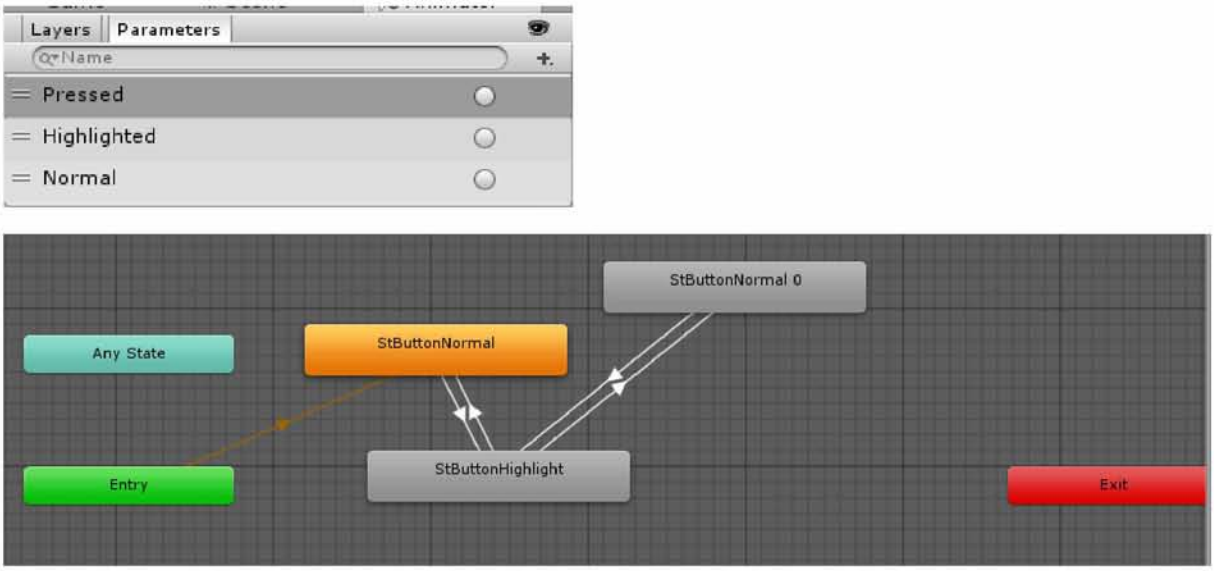
4.3.2 Λειτουργίες του μενού

Το μενού λειτουργεί με τη χρήση των κουμπιών Start Game, Controls και Quit Game. Αυτά τα κουμπιά ελέγχονται μέσω του αντικειμένου Button Manager το οποίο εκτελεί το script MenuButtons που γράψαμε.

```
public class MenuButtons : MonoBehaviour {

    public void LoadALevel(string newLevel)
    {
        SceneManager.LoadScene(newLevel);
    }

    public void QuitGame()
    {
        Application.Quit();
    }

}
```

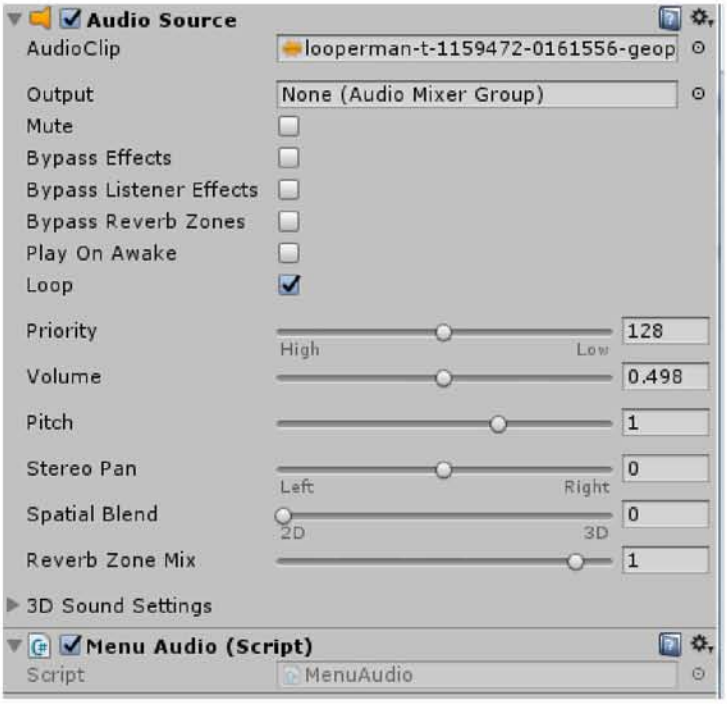
Εικόνα 38: Script βασικών λειτουργιών μενού

Τα κουμπιά Start Game και Controls φορτώνουν νέες σκηνές στην εφαρμογή ,επομένως εκτελούν τη μέθοδο LoadALevel (string newLevel) για να μεταβούν στη σκηνή της οποίας το όνομα αποθηκεύεται στο string newLevel. Το κουμπί Quit Game ευθύνεται για τον τερματισμό της εφαρμογής και εκτελεί με το πάτημά του τη μέθοδο QuitGame(). Βέβαια, επιθυμούμε το αρχικό μας μενού να είναι πιο δυναμικό, επομένως προσθέτουμε animations στα κουμπιά, αυξομειώνοντας το μέγεθός τους σύμφωνα με παραμέτρους καταστάσεων. Η δημιουργία animations στην πλατφόρμα Unity γίνεται μέσω του αντικειμένου Animator και κάθε animation αποτελεί ένα διάγραμμα πεπερασμένων καταστάσεων. Στην επόμενη Εικόνα παρουσιάζουμε το διάγραμμα που δημιουργήσαμε καθώς και τις παραμέτρους των καταστάσεων που ορίσαμε.



Εικόνα 39: Διάγραμμα καταστάσεων και παραμέτρων των animations κουμπιών.

Η μουσική του μενού ελέγχεται από το αντικείμενο Menu Audio και το αντίστοιχο MenuAudio script το οποίο επιδρά στο στοιχείο Audio Source.



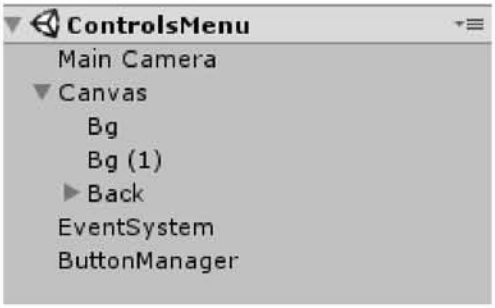
Εικόνα 40: Αντικείμενο Audio Source

```
[RequireComponent(typeof(AudioSource))]
public class MenuAudio : MonoBehaviour {

    // Use this for initialization
    void Start () {
        AudioSource menuAudio = GetComponent<AudioSource>();
        menuAudio.Play();
    }


}
```

Εικόνα 41: Script ήχου μενού

4.3.3 Σκηνή Χειρισμού

Ο παίκτης έχει πρόσβαση στη σκηνή χειρισμού με το πάτημα του κουμπιού Controls και από εκεί πληροφορείται για τα πλήκτρα με τα οποία θα μπορεί να αλληλεπιδράσει με το παιχνίδι. Η σκηνή χειρισμού περιέχει ένα νέο καμβά με ένα δισδιάστατο φόντο που δημιουργήσαμε με τη χρήση του Photoshop. Επίσης, αυτή η σκηνή εκτελεί ξανά τη μέθοδο LoadALevel (string newLevel) διότι σε αυτή δημιουργούμε ένα κουμπί Back για εύκολη μετάβαση πίσω στο μενού έναρξης. Στις επόμενες Εικόνες παρουσιάζουμε την ιεραρχία της σκηνής όπως και την προεπισκόπησή της.



Εικόνα 42: Ιεραρχία Σκηνής Χειρισμού

Εικόνα 43: Προεπισκόπηση σκηνής χειρισμού

4.4 Λογαριασμός Παίκτη

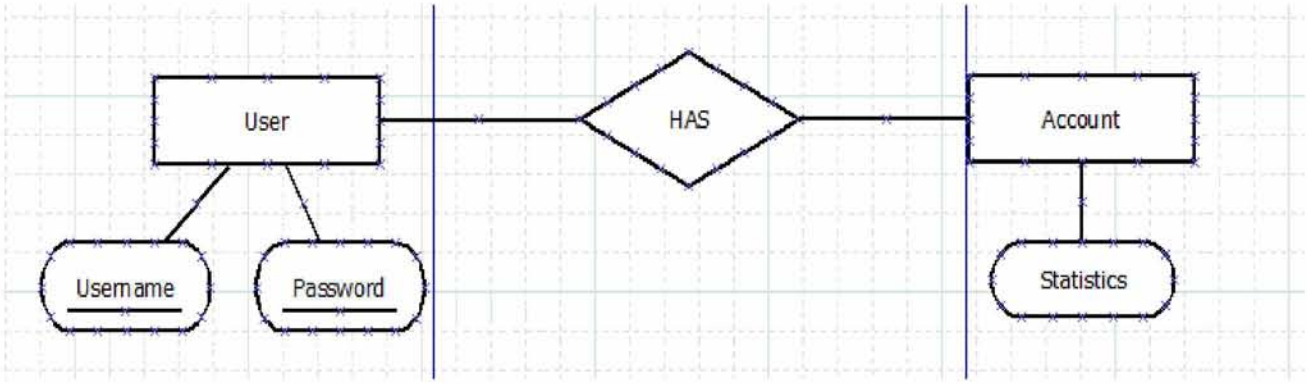
4.4.1 Σχεσιακό Σχήμα και Λειτουργία της Βάσης Δεδομένων

Κάθε παίκτης για να ξεκινήσει το παιχνίδι χρειάζεται να εγγραφεί είτε να συνδεθεί στο λογαριασμό του έτσι ώστε να μπορούν τα στοιχεία του όπως και οι πληροφορίες του παιχνιδιού του να αποθηκεύονται στο διαδίκτυο. Το εργαλείο που επιλέγουμε να χρησιμοποιήσουμε είναι το Database Control [39] το οποίο αναπτύχθηκε από τη Unity Solution Studios. Αυτό το εργαλείο δίνει τη δυνατότητα στον προγραμματιστή να δημιουργήσει διαδικτυακές βάσεις δεδομένων οι οποίες υποστηρίζουν έναν απεριόριστο αριθμό χρηστών σε ενεργούς διαδικτυακούς εξυπηρετητές μέσα από τη διεπαφή του. Συνεπώς, με το Database Control χρειαζόμαστε μόνο γνώσεις προγραμματισμού C# και χειρισμό του Unity Editor για να διαμορφώσουμε τη βάση δεδομένων μας.

Το σχεσιακό σχήμα της βάσης δεδομένων επιθυμούμε να χαρακτηρίζεται από απλότητα έτσι ώστε η εισαγωγή και η ανάκτηση της πληροφορίας να γίνεται γρήγορα μέσω χειρισμού συμβολοσειρών. Ειδικότερα η βάση μας αποτελείται από την οντότητα User με

χαρακτηριστικά το username και το password τα οποία αποτελούν πρωτεύοντα κλειδιά, τη σχέση HAS η οποία είναι ένα προς ένα και την οντότητα Account με χαρακτηριστικό τα Statistics τα οποία αποθηκεύονται σε μία συμβολοσειρά. Στην Εικόνα 44 παρουσιάζουμε το σχεσιακό σχήμα της βάσης δεδομένων το οποίο κατασκευάσαμε με το πρόγραμμα DIA. Οι χρήστες χρειάζονται σύνδεση στο διαδίκτυο ώστε να έχουν πρόσβαση στον λογαριασμό τους.

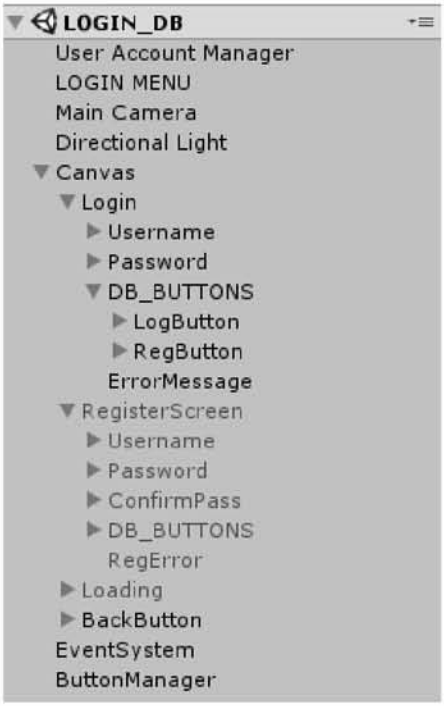
Το Database Control εμπεριέχει παραδείγματα σε C# και Javascript με ενδεικτική διάρθρωση βάσεων δεδομένων τα οποία περιλαμβάνουν βασικές μεθόδους επικοινωνίας με τη βάση. Στις επόμενες ενότητες θα προσαρμόσουμε αυτά τα scripts στις ανάγκες της βάσης δεδομένων μας χωρίζοντάς τα σε επιμέρους scripts διαχείρισης της σκηνής σύνδεσης και εγγραφής όπως και διαχείρισης των πληροφοριών λογαριασμού χρήστη.



Εικόνα 44: Σχεσιακό σχήμα βάσης δεδομένων παιχνιδιού

4.4.2 Σκηνή Σύνδεσης και Εγγραφής Παίκτη

Η σκηνή σύνδεσης και εγγραφής δίνει πρόσβαση στους υπάρχοντες λογαριασμούς παικτών και επιτρέπει σε νέους παίκτες να δημιουργήσουν ένα λογαριασμό ώστε η βάση δεδομένων να αποθηκεύει χρήσιμα στατιστικά σχετικά με την πρόοδο του παίκτη. Για να οργανώσουμε αυτή τη σκηνή ορίζουμε το Login script το οποίο περιέχει μεθόδους του Database Control οι οποίες εφαρμόζουν τις ενέργειες σύνδεσης και εγγραφής αλλά και μεθόδους οι οποίες ρυθμίζουν την αλληλεπίδραση του χρήστη με τα κουμπιά και τα πεδία συμπλήρωσης της σκηνής.



Εικόνα 45: Ιεραρχική Διάρθρωση Σκηνής Εγγραφής και Σύνδεσης

Ο κώδικας του Login script βρίσκεται στην παράγραφο 6.1. Σε αυτή την ενότητα εξηγούμε τη λειτουργία των μεθόδων που αναπτύσσονται στο Login Menu script.

```
void ResetAllUIElements() :
```

Με αυτή τη μέθοδο όλα τα πεδία προς συμπλήρωση αρχικοποιούνται στην κενή συμβολοσειρά. Η μέθοδος αυτή καλείται κατά τη μετάβαση στο μενού σύνδεσης όπως επίσης και κατά τη σύνδεση και αποσύνδεση παικτών έτσι ώστε τα πεδία συμπλήρωσης να παραμένουν κενά.

```
IEnumerator LoginUser(string playerUsername,string playerPassword) :
```

Με αυτή τη μέθοδο γίνεται αποστολή αίτησης στη βάση δεδομένων έτσι ώστε να συνδεθεί ο παίκτης με όνομα playerUsername και κωδικό playerPassword. Από αυτή τη συνάρτηση μεταφέρεται η πληροφορία στο διαχειριστή λογαριασμών ώστε να δοθεί πρόσβαση σε αυτό το λογαριασμό.

```
IEnumerator RegisterUser(string playerUsername , string playerPassword):
```

Με αυτή τη μέθοδο γίνεται αποστολή αίτησης στη βάση δεδομένων για καταχώρηση ενός νέου παίκτη με όνομα playerUsername και κωδικό playerPassword. Ο παίκτης καλείται να καταχωρήσει δύο φορές τον κωδικό του για επιβεβαίωση. Μετά από την επιτυχή εγγραφή του ο παίκτης έχει πρόσβαση στο λογαριασμό του μέσω του διαχειριστή λογαριασμών.

```
public void Login_LoginButtonPressed():
```

Η μέθοδος αυτή ενεργοποιείται με το πάτημα του κουμπιού Login στο πρώτο αντικείμενο του καμβά. Σε αυτή τη μέθοδο γίνεται έλεγχος εγκυρότητας στοιχείων του παίκτη. Αν ο έλεγχος αληθεύει τότε γίνεται μετάβαση στην επόμενη σκηνή, ενώ αν ο έλεγχος αποτύχει επιστρέφεται μήνυμα σφάλματος.

```
public void Login_RegisterButtonPressed():
```

Η μέθοδος αυτή ενεργοποιείται με το πάτημα του κουμπιού Register στο πρώτο αντικείμενο του καμβά. Ως αποτέλεσμα εκτέλεσης αυτής της μεθόδου έχουμε τη μετάβαση στο αντικείμενο RegisterScreen όπου εκεί ένας νέος παίκτης μπορεί να δημιουργήσει το λογαριασμό του.

```
public void Register_RegisterButtonPressed():
```

Αυτή η μέθοδος ελέγχει και επιβεβαιώνει την καταχώρηση ενός νέου παίκτη και τον συνδέει απευθείας στο παιχνίδι. Αν οι χαρακτήρες εισαγωγής των στοιχείων παίκτη δεν είναι έγκυροι τότε επιστρέφεται μήνυμα σφάλματος.

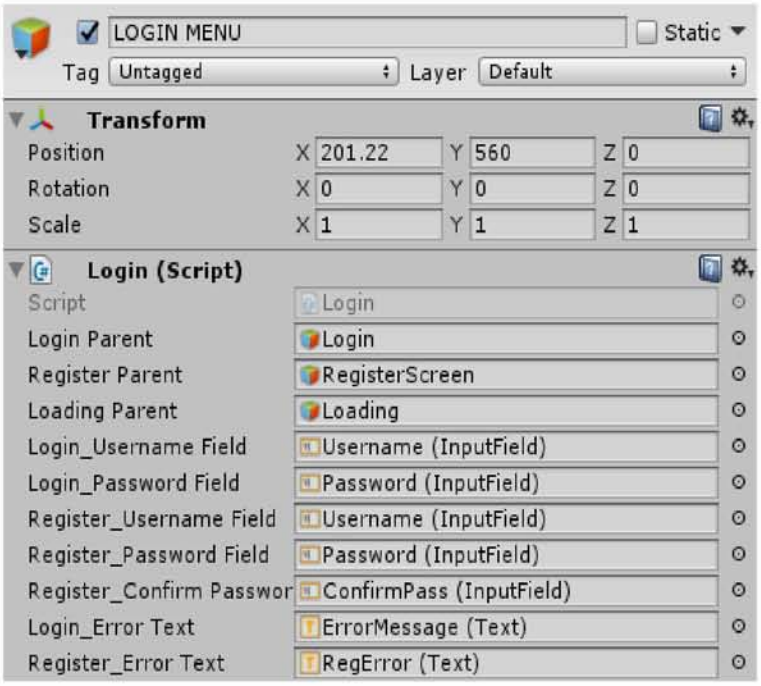
```
public void Register_BackButtonPressed():
```

Με την εκτέλεση της μεθόδου επιστρέφουμε στο μενού Login στην κορυφή της ιεραρχίας καμβά.

```
public void LoggedIn_LogoutButtonPressed():
```

Αυτή η μέθοδος μπορεί να εκτελεσθεί εφόσον ο παίκτης έχει συνδεθεί και έχει μεταφερθεί στην επόμενη σκηνή. Η εκτέλεση της μεθόδου οδηγεί στην αποσύνδεση του παίκτη από το παιχνίδι και στην έξοδο από το λογαριασμό του.

Στις επόμενες εικόνες παρουσιάζουμε το αντικείμενο Login Menu με τα πεδία τα οποία ελέγχει και παραθέτουμε μία προεπισκόπηση της σκηνής σύνδεσης.



Εικόνα 46: Αντικείμενο Login Menu και οι συνδέσεις του script με τα αντικείμενα της ιεραρχίας

Εικόνα 47: Επισκόπηση σκηνής σύνδεσης και εγγραφής

4.4.3 Διαχείριση Πληροφοριών Λογαριασμού

Σε αυτή την ενότητα ορίζουμε το script Account Management το οποίο περιέχει τροποποιημένες μεθόδους του Database Control που διαχειρίζονται τη λήψη και αποστολή πληροφοριών ώστε να πραγματοποιείται αποδοτική επεξεργασία των στοιχείων του παίκτη με χειρισμό συμβολοσειρών. Επιπροσθέτως, το script Account Management επαναχρησιμοποιείται και σε άλλες σκηνές για τον προσδιορισμό της εξέλιξης του παίκτη.

Ο κώδικας του Account Management script παρουσιάζεται στην παράγραφο 6.2. Είναι βέβαια αναγκαίο να γίνει επεξήγηση των λειτουργιών των μεθόδων στο παιχνίδι που αναπτύσσουμε. Οι μέθοδοι αυτού του script λειτουργούν σε συνεργασία με τις μεθόδους του Login script ωστόσο τις έχουμε διαφοροποιήσει λόγω του εκτεταμένου ρόλου του Account Management σε επόμενες σκηνές.

```
public void LogOut():
```

Η μέθοδος ελέγχει την κατάσταση εγγραφής παίκτη και αν ο παίκτης είναι αποσυνδεδεμένος τότε γίνεται εκ νέου φόρτωση του Login Menu.

```
public void LogIn(string username, string password):
```

Η μέθοδος ελέγχει αν ο χρήστης είναι συνδεδεμένος και εκτελεί φόρτωση της επόμενης σκηνής.

```
public void SendData(string data)
IEnumerator ActivateQuery(string username, string password, string data):
```

Με τις δύο παραπάνω μεθόδους γίνεται αποστολή αίτησης προς τη βάση δεδομένων για την αποστολή πληροφοριών παίκτη με τη μορφή συμβολοσειράς. Η αποστολή αυτής της αίτησης γίνεται αποδοτικά μέσω εκτέλεσης coroutine με αποτέλεσμα ο κώδικας να έχει μεγαλύτερη ευελιξία στο χρόνο εκτέλεσής του.

```
public void GetData(OnDataReceivedCallback onDataReceived)
IEnumerator GetQueryResult(string username, string password, OnDataReceivedCallback
onDataReceived):
```

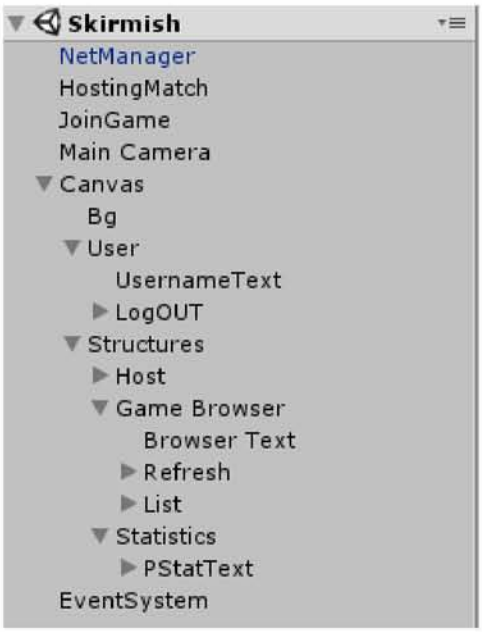
Παρόμοια με τις δύο προηγούμενες μεθόδους ,αυτές οι μέθοδοι αναλαμβάνουν τη λήψη πληροφοριών και στη συνέχεια καλούν τη μέθοδο onDataReceived για επεξεργασία αυτών των πληροφοριών σε άλλη σκηνή. Ακριβώς όπως και πριν γίνεται η χρήση coroutine για πιο ευέλικτη εκτέλεση του κώδικα.

4.5 Διοργάνωση Διαδικτυακών Παιχνιδιών

4.5.1 Οργάνωση Παιχνιδιού

Στις προηγούμενες ενότητες αναλύσαμε το αντικείμενο Network Manager και παρουσιάσαμε το εργαλείο Match Maker για τη διοργάνωση διαδικτυακών παιχνιδιών. Σε αυτή την ενότητα παρουσιάζουμε την εφαρμογή αυτών των εργαλείων για τη σταδιακή δημιουργία της σκηνής Skirmish η οποία θα επιτρέπει στους παίκτες να οργανώσουν παιχνίδια, να συνδεθούν σε παιχνίδια που βρίσκονται σε εξέλιξη και να παρακολουθούν την πρόοδό τους στο παιχνίδι.

Στην Εικόνα 48 παρουσιάζουμε την ιεραρχία των αντικειμένων που αποτελούν τη σκηνή Skirmish.



Εικόνα 48: Ιεραρχική δομή της σκηνής Skirmish

Αρχικά, είναι απαραίτητη η υλοποίηση του κώδικα που θα προσαρτηθεί στο αντικείμενο HostingMatch το οποίο θα αλληλεπιδρά με το χρήστη μέσω της δομής Host στην ιεραρχία του καμβά. Το Hosting script το οποίο παρατίθεται στην παράγραφο 6.3 υλοποιεί τη λειτουργία οργάνωσης παιχνιδιού. Καθώς επιθυμούμε οι αγώνες να είναι μικρής κλίμακας και απεριόριστης διάρκειας, επιλέγουμε κάθε παιχνίδι που διοργανώνεται να υποστηρίζει τέσσερις παίκτες το πολύ έτσι ώστε η εμπειρία κάθε παίκτη στο επίπεδο να είναι ισορροπημένη. Επιπροσθέτως, το παιχνίδι επιτρέπει στο διοργανωτή να επιλέξει ένα online επίπεδο στο οποίο θα συνδέονται οι υπόλοιποι παίκτες. Συνεπώς, με την υποστήριξη περισσότερων online επιπέδων οι παίκτες μπορούν να συμμετέχουν σε αγώνες σε περισσότερα περιβάλλοντα.

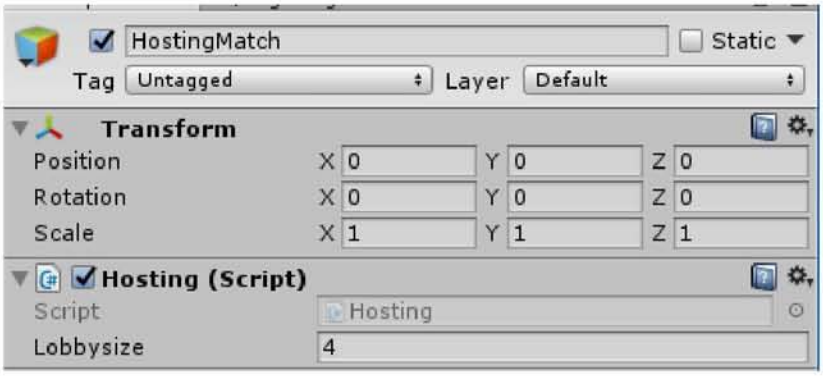
Εντός του Hosting script αρχικοποιούμε τη μεταβλητή singleton του Network Manager η οποία θα καθορίσει την κατάσταση του δικτύου και ενεργοποιούμε το εργαλείο του Match Maker κατά τη φόρτωση της σκηνής. Επιπροσθέτως, υλοποιούμε τις παρακάτω μεθόδους.

```
public void SetName(string thename):
```

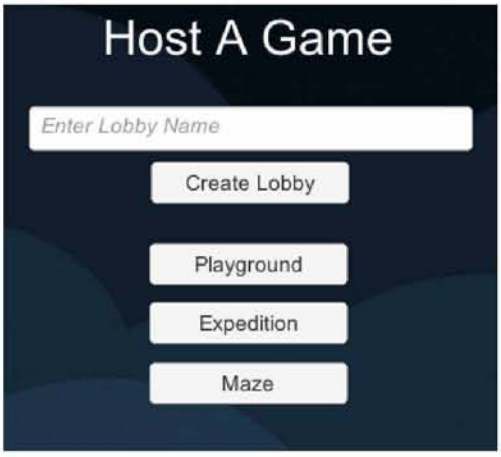
Η μέθοδος SetName καθορίζει το όνομα του παιχνιδιού που πρόκειται να διοργανωθεί έτσι ώστε αυτό να γίνεται ευδιάκριτο από τον περιηγητή παιχνιδιών που θα αναπτύξουμε σε επόμενη ενότητα.

```
public void CreateLobby():
```

Η εκτέλεση της μεθόδου καλεί τη λειτουργία Create Match της διεπαφής δικτύωσης της Unity με σκοπό τη δημιουργία ενός αγώνα τεσσάρων παικτών. Οι παράμετροι της λειτουργίας Create Match στοχεύουν στη μείωση των αγώνων που είναι διαθέσιμοι σύμφωνα με τις δεξιότητες των παικτών και τις διευθύνσεις των προγραμμάτων-πελατών. Επειδή επιθυμούμε οι παίκτες να έχουν τη δυνατότητα να προβάλλουν όλους τους εξελισσόμενους αγώνες, θέτουμε τις παραμέτρους στις προκαθορισμένες τιμές τους.



Εικόνα 49: Το αντικείμενο HostingMatch



Εικόνα 50: Η λειτουργία της διοργάνωσης παιχνιδιού

4.5.2 Περιηγητής Ενεργών Παιχνιδιών

Όπως έχουμε αναφέρει και σε προηγούμενη ενότητα, η δυνατότητα σύνδεσης σε ένα παιχνίδι που βρίσκεται σε εξέλιξη είναι εξίσου σημαντική με τη δυνατότητα διοργάνωσης νέου παιχνιδιού. Επιθυμούμε μέσα από τη σκηνή Skirmish να παρέχουμε πρόσβαση σε ένα περιηγητή ο οποίος θα παρουσιάζει μία λίστα ενεργών παιχνιδιών συμπεριλαμβάνοντας τον αριθμό των παικτών που βρίσκονται ήδη εντός του παιχνιδιού. Η δημιουργία του περιηγητή αποτελείται από το αντικείμενο JoinGame στο οποίο έχουμε προσαρτήσει το αντίστοιχο

Joining script (6.4) .Επίσης, είναι αναγκαία και η δημιουργία της κυλιόμενης λίστας που έχουμε ονομάσει Game Browser η οποία αλληλεπιδρά με τον παίκτη παρουσιάζοντας τα παιχνίδια που βρίσκονται σε εξέλιξη. Κάθε παιχνίδι που βρίσκεται σε εξέλιξη έχει οριστεί ως προκατασκευασμένο αντικείμενο και η συμπεριφορά του εντός της κυλιόμενης λίστας καθορίζεται από το LobbyListItem script (6.5).Στη συνέχεια επεξηγούμε τη λειτουργία των μεθόδων που υλοποιούμε εντός αυτών των scripts.

Joining Script:

```
public void RefreshLobbies():
```

Η μέθοδος αυτή ανανεώνει τη λίστα των παιχνιδιών στον περιηγητή εμφανίζοντας νέα παιχνίδια τα οποία έχουν δημιουργηθεί. Η κλήση της μεθόδου γίνεται από το κουμπί Refresh.

```
public void Lobbies(bool success, string extendedInfo, List<MatchInfoSnapshot> matches):
```

Η μέθοδος Lobbies φροντίζει να προσδιορίσει όλες τις ιδιότητες των προκατασκευασμένων αντικειμένων ενεργού παιχνιδιού πριν αυτά εμφανιστούν στην κυλιόμενη λίστα.

```
void ClearLobbyList():
```

Με την εκτέλεση αυτής της μεθόδου γίνεται καταστροφή ενός προκατασκευασμένου αντικειμένου παιχνιδιού αφαιρώντας το από τη λίστα καθώς το παιχνίδι δεν είναι πλέον προσβάσιμο. Αξίζει να σημειωθεί ότι η ανανέωση της λίστας σε αυτή την έκδοση του Unity Networking δεν αποκρίνεται άμεσα σε αλλαγές .Επομένως χρειάζεται ειδικός χειρισμός έτσι ώστε να αποτρέψουμε τους παίκτες από τη σύνδεση σε μη ενεργά παιχνίδια.

```
public void JoinLobby(MatchInfoSnapshot mymatch):
```

Η μέθοδος JoinLobby καθορίζει τις παραμέτρους της μεταβλητής τύπου MatchInfoSnapshot ώστε να έχει όλες τις απαραίτητες πληροφορίες για να πραγματοποιήσει αίτηση σύνδεσης στον Unity Match Maker. Σε αυτή τη μέθοδο γίνεται χρήση του εργαλείου JoinMatch του Match Maker με τις προκαθορισμένες τιμές του για την πραγματοποίηση της σύνδεσης.

```
IEnumerator WaitForJoin():
```

Με τη μέθοδο WaitForJoin ελέγχουμε αν το παιχνίδι που παρουσιάζεται στη λίστα του περιηγητή είναι διαθέσιμο για σύνδεση καθώς παρουσιάζουμε στον παίκτη μία αντίστροφη μέτρηση. Αν η αντίστροφη μέτρηση λήξει και ο παίκτης δεν έχει συνδεθεί τότε το παιχνίδι δεν ήταν διαθέσιμο και η ενέργεια της μεθόδου ClearLobbyList θα έχει πλέον αντικατοπτριστεί στη λίστα με αποτέλεσμα να διορθωθούν οι καθυστερήσεις του Unity Networking σε αυτό τον τομέα. Επίσης, αυτή η μέθοδος χρησιμεύει ως ουρά αναμονής στην περίπτωση που ένα ενεργό παιχνίδι έχει το μέγιστο αριθμό παικτών και ένας παίκτης

αποφασίσει να αποσυνδεθεί το ίδιο χρονικό διάστημα που ένας παίκτης αποφασίσει να συνδεθεί.

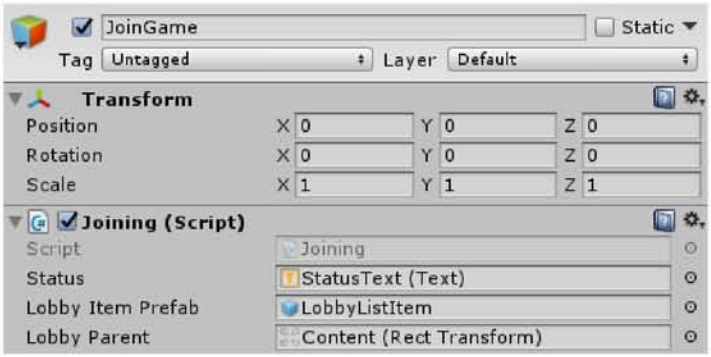
Lobby List Item script:

```
public void Setup(MatchInfoSnapshot mymatch,JoinLobbyDelegate myjoinLobbyCallback):
```

Η μέθοδος Setup στοχεύει στη ρύθμιση του αντικειμένου της κυλιόμενης λίστας με τον τίτλο και τον αριθμό παικτών έτσι ώστε να ανανεώνεται ανάλογα με το πιο πρόσφατο στιγμιότυπο του παιχνιδιού.

```
public void JoinLobby ():
```

Η μέθοδος εκτελείται με την επιλογή του προκατασκευασμένου αντικειμένου καλώντας την αντίστοιχη συνάρτηση σύνδεσης του Joining script με τις πληροφορίες στιγμιότυπου παιχνιδιού που είχαν καθοριστεί κατά τη δημιουργία του.



Εικόνα 51: Επισκόπηση δομικού αντικειμένου JoinGame

Εικόνα 52: Επισκόπηση περιηγητή ενεργών παιχνιδιών

4.5.3 Αποσύνδεση Παίκτη

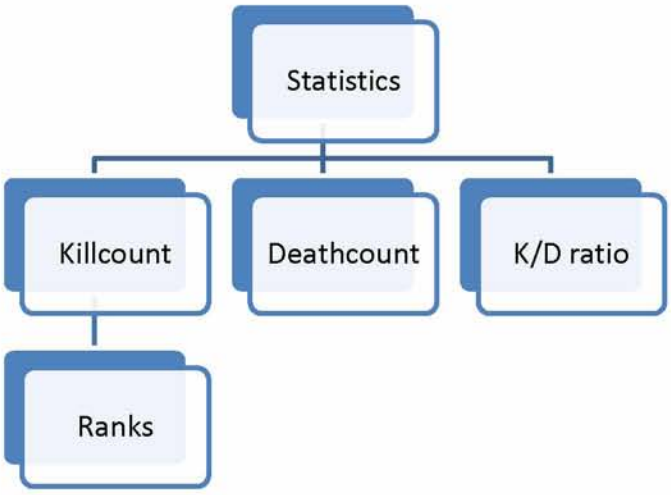
Η σκηνή Skirmish δίνει τη δυνατότητα προβολής του λογαριασμού παίκτη όπως επίσης και την επιλογή αποσύνδεσης διότι ο διαχειριστής λογαριασμών που ορίσαμε σε προηγούμενη ενότητα δεν καταστρέφεται με τη φόρτωση της σκηνής Skirmish. Ειδικότερα, έχουμε υλοποιήσει το script User Account για να ελέγχει την κατάσταση του λογαριασμού παίκτη στη νέα σκηνή και να επικοινωνεί με το Account Management όταν ένας παίκτης επιθυμεί να αποσυνδεθεί.

Το script User Account που παρατίθεται στην παράγραφο 6.6 είναι ένα απλοϊκό script το οποίο αρχικοποιεί το στιγμιότυπο του User Account Manager στη νέα σκηνή και προβάλλει το όνομα χρήστη ως αναγνωριστικό σύνδεσης. Επίσης, υλοποιείται η μέθοδος αποσύνδεσης η οποία χρειάζεται να προσαρτηθεί στο αντικείμενο logOUT της ιεραρχίας έτσι ώστε το κουμπί με όνομα Sign Out να εκτελεί τη μέθοδο LogOut την οποία έχουμε επεξηγήσει στο Account Management. Ως αποτέλεσμα της εφαρμογής αυτού του script έχουμε το μέρος αποσύνδεσης παίκτη το οποίο είναι πάντοτε προσανατολισμένο στο κάτω δεξιά μέρος του παραθύρου (Εικόνα 53).

Εικόνα 53: Προβολή ενεργού παίκτη και κουμπί αποσύνδεσης

4.5.4 Σύστημα Εξέλιξης Παίκτη

Η πρόοδος όλων των παικτών αποθηκεύεται στη βάση δεδομένων με τη μορφή συμβολοσειράς η οποία διαχωρίζει τα στατιστικά με την προσθήκη ενός ειδικού χαρακτήρα. Συγκεκριμένα, σε κάθε παιχνίδι οι παίκτες προσπαθούν να καταστρέψουν τα ρομπότ των αντιπάλων τους. Επομένως, γίνεται καταμέτρηση των καταστροφών των αντιπάλων αλλά και του ίδιου του παίκτη σε πραγματικό χρόνο και ανανεώνονται τα στατιστικά παίκτη στη βάση δεδομένων. Όλοι οι παίκτες έχουν ως στόχο να μεγιστοποιήσουν τις καταστροφές των αντιπάλων τους ώστε να διατηρήσουν μία καλή αναλογία καταστροφών και να κατακτήσουν νέους τίτλους που θα αντικατοπτρίζουν την απόδοσή τους.



Εικόνα 54: Ιεραρχία στατιστικών παίκτη.

Για την επεξεργασία των στατιστικών παίκτη υλοποιούμε το script User Data το οποίο περιέχει μεθόδους που αναλαμβάνουν κατάτμηση και μετατροπή συμβολοσειρών σε ακεραίους και αντίστροφα. Ο κώδικας του script παρατίθεται στην παράγραφο 6.7 και στη συνέχεια επεξηγούμε τη λειτουργία των μεθόδων του.

`public static int KillDataConversion(string data):`
Η μέθοδος λαμβάνει και εξάγει τις καταστροφές αντιπάλων σε ακέραια τιμή.

`public static int DeathsDataConversion(string data):`
Η μέθοδος λαμβάνει και εξάγει τις καταστροφές παίκτη σε ακέραια τιμή.

`public static string ValuesToString(int killcount, int deathcount):`
Η εκτέλεση της μεθόδου μετατρέπει πληροφορίες στατιστικών παίκτη σε συμβολοσειρά χωρισμένη με κατάλληλο ειδικό χαρακτήρα ώστε να ανανεωθεί στη βάση δεδομένων.

`private static string StringToValue(string data, string symbol):`
Η μέθοδος αυτή εκτελεί την αντίστροφη διαδικασία της προηγούμενης μεθόδου κάνοντας κατάτμηση της συμβολοσειράς και ανίχνευση των πεδίων στατιστικών παίκτη.

Ο συγχρονισμός και η ανάκτηση των στατιστικών κατά την εξέλιξη του παιχνιδιού επιτυγχάνεται με την υλοποίηση του script Score Tracking.Ο κώδικας του script παρουσιάζεται στην παράγραφο 6.8 και είναι αναγκαίο να παρουσιάσουμε σε αυτή την ενότητα τη λειτουργία των μεθόδων του.

`IEnumerator ScoreTracker ():`
Η μέθοδος ScoreTracker πραγματοποιεί έλεγχο ενημέρωσης δεδομένων κάθε δέκα δευτερόλεπτα έτσι ώστε να καταχωρείται ένα πρόσφατο στιγμιότυπο δεδομένων στη βάση.

`void Tracking():`
Η εκτέλεση αυτής της μεθόδου δημιουργεί επικοινωνία με την κλάση Account Management για ενημέρωση των πληροφοριών παίκτη και καλείται εντός της προηγούμενης μεθόδου.

`void ReceiveHandle(string data):`

Η πρόσβαση και η ενημέρωση των τιμών που θα αποστείλει η μέθοδος Tracking γίνεται μέσω της μεθόδου ReceiveHandle η οποία επικοινωνεί άμεσα με το αντικείμενο παίκτη ώστε να λάβει τις τιμές των πρόσφατων στατιστικών στοιχείων που αφορούν το τρέχον παιχνίδι.

Ωστόσο, οι λειτουργίες των παραπάνω scripts εκτελούνται στο παρασκήνιο και ο παίκτης μέχρι τώρα δεν έχει γνώση της καταγραφής της προόδου. Επομένως ,υλοποιούμε το script Player Statistics το οποίο αναλαμβάνει την προβολή των πληροφοριών του παίκτη στη σκηνή Skirmish εφόσον έχει ήδη αρχικοποιήσει επικοινωνία με την κλάση Account Management. Ο πλήρης κώδικας του script βρίσκεται στην παράγραφο 6.9 και σε αυτό το κομμάτι της εργασίας παρουσιάζουμε τις μεθόδους του.

```
void onDataRecieved (string data):
```

Η μέθοδος onDataRecieved είναι η δομική μέθοδος προβολής πληροφοριών. Αναφορά σε αυτή τη μέθοδο γίνεται και εντός του script Account Management ώστε να πραγματοποιηθεί αίτηση αποστολής δεδομένων προς συμπλήρωση του ορίσματος και εκτέλεση αυτής της μεθόδου. Επίσης, σε αυτή τη μέθοδο καθορίζονται οι παρακάτω τίτλοι προόδου οι οποίοι αποκτώνται μετά από ένα συγκεκριμένο αριθμό καταστροφών αντιπάλων.

Τίτλοι προόδου:

- Rookie Rank (Απόκτηση με 0 καταστροφές)
- Obsidian Rank (Απόκτηση με 5 καταστροφές)
- Onyx Rank (Απόκτηση με 10 καταστροφές)
- Pearl Rank (Απόκτηση με 20 καταστροφές)
- Pyrite Rank (Απόκτηση με 30 καταστροφές)
- Ruby Rank (Απόκτηση με 40 καταστροφές)
- Sapphire Rank (Απόκτηση με 50 καταστροφές)
- Topaz Rank (Απόκτηση με 60 καταστροφές)
- Zircon Rank (Απόκτηση με 70 καταστροφές)

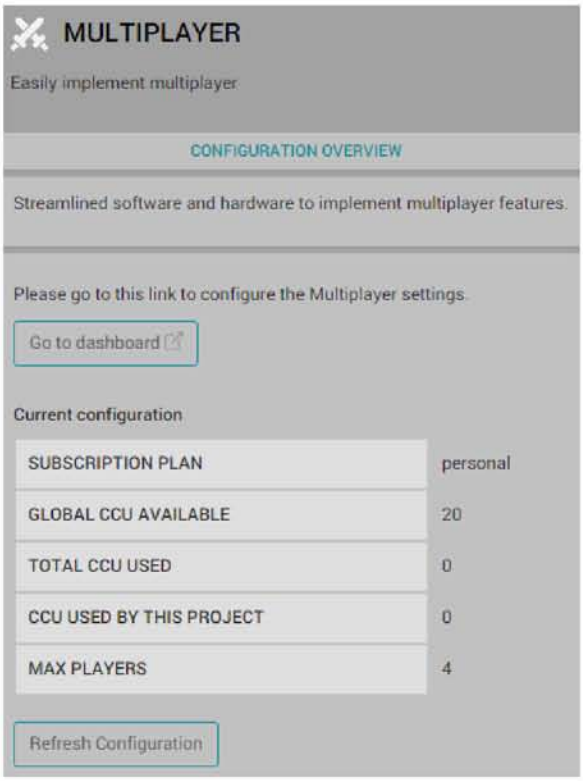
Εικόνα 55: Επισκόπηση δομής Player Statistics στη σκηνή Skirmish

4.5.5 Καθορισμός Διαδικτυακών Υπηρεσιών Unity

Το παιχνίδι που αναπτύσσουμε υποστηρίζει διαδικτυακούς αγώνες με άλλους παίκτες λόγω της χρήσης του Unity Networking. Ωστόσο, το παιχνίδι που προγραμματίζουμε, όπως και κάθε διαδικτυακό παιχνίδι της Unity, κατατάσσεται στο επίπεδο εφαρμογής του μοντέλου OSI και δεν διαθέτει μηχανισμούς οι οποίοι χειρίζονται περιορισμούς που σχετίζονται με την ύπαρξη δρομολογητών και πρωτοκόλλου NAT. Όπως αναφέραμε και σε προηγούμενη ενότητα, η Unity προσφέρει την υπηρεσία Multiplayer η οποία φιλοξενεί εφαρμογές στους εξυπηρετητές της πλατφόρμας με συνδρομητικό κόστος. Όπως γίνεται σαφές και από το Unity Networking, η υπηρεσία Multiplayer αλλάζει μόνο τα περιεχόμενα του αντικειμένου Network Manager singleton για να εκφράσει τη νέα τοπολογία δικτύου. Με αυτό τον τρόπο η πλατφόρμα επιτρέπει στους προγραμματιστές να ελέγχουν δωρεάν τις διαδικτυακές λειτουργίες του παιχνιδιού τους στο δίκτυό τους καθώς η διεπαφή δικτύου είναι ενοποιημένη και δεν χρειάζεται αλλαγές κώδικα ώστε να λειτουργήσει σε άλλη τοπολογία.

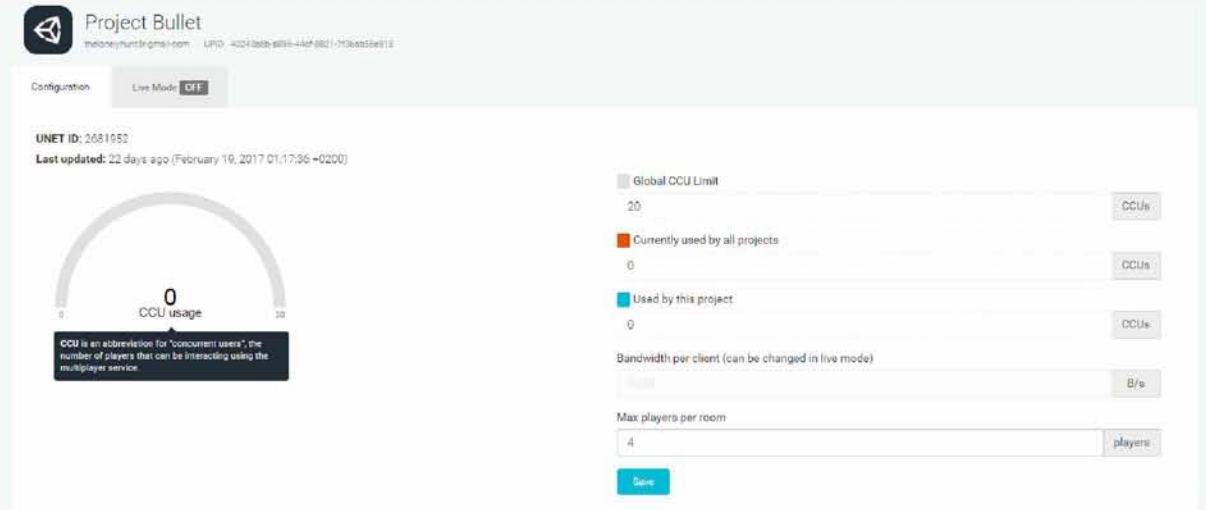
Σε αυτή την ενότητα θα παρουσιάσουμε τη διαδικασία με την οποία θα μπορούσαμε να φιλοξενήσουμε το διαδικτυακό μας παιχνίδι στους εξυπηρετητές Unity, εξερευνώντας τη διαδικασία κυκλοφορίας του παιχνιδιού σε έκδοση beta ή Early Access καθώς και αυτό αποτελεί βήμα της διαδικασίας ανάπτυξης ενός παιχνιδιού. Βέβαια, όλοι οι έλεγχοι λειτουργίας του παιχνιδιού γίνονται με τη δωρεάν εναλλακτική της Unity καθώς αυτό το παιχνίδι αποτελεί μέρος εργασίας και όχι εμπορικό προϊόν.

Αρχικά χρειάζεται να καταχωρήσουμε την εργασία μας στο σύστημα της υπηρεσίας Multiplayer έτσι ώστε να επιβεβαιώσουμε τη χρήση του εργαλείου Match Maker και να λάβουμε το ειδικό αναγνωριστικό Unity που συνδέεται με την εργασία μας.



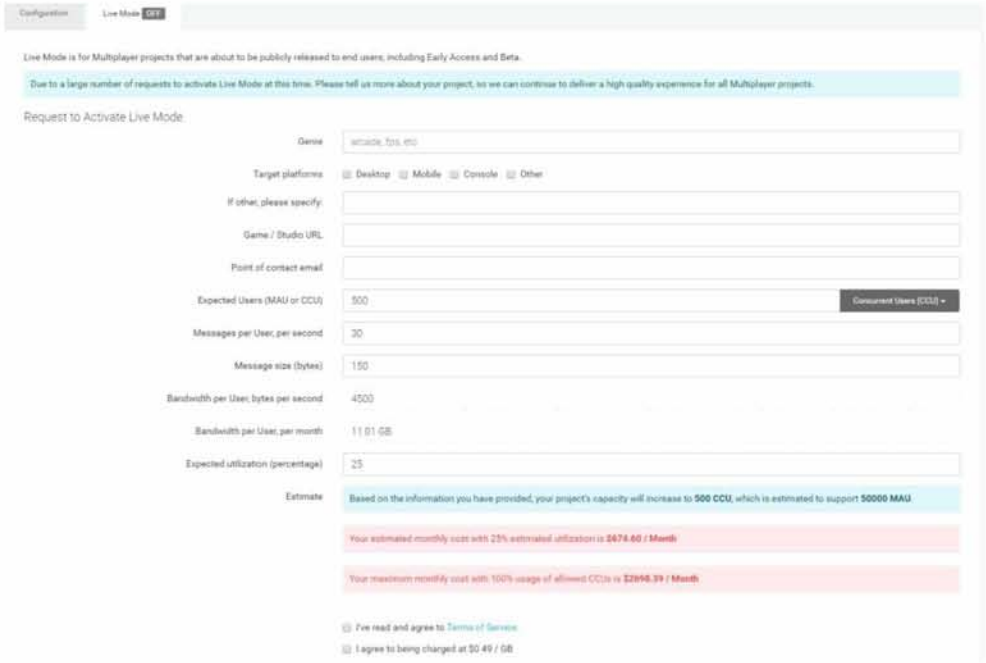
Εικόνα 56: Επισκόπηση Υπηρεσίας Multiplayer

Στη συνέχεια είναι απαραίτητο να επισκεφθούμε τη σελίδα Unity Developer όπου μπορούμε να καθορίσουμε τον αριθμό παικτών του επιθυμούμε να υποστηρίζουμε ανά ενεργό παιχνίδι καθώς και μία εκτίμηση του συνολικού αριθμού παικτών που θα είναι συνδεδεμένοι ταυτόχρονα. Αυτή η υπηρεσία προσαρμόζει τις χρεώσεις του συνδρομητικού προγράμματος δεδομένου του μέγιστου αριθμού ταυτόχρονα συνδεδεμένων παικτών και του όγκου των μηνυμάτων που μπορούν να μεταφερθούν για την εξυπηρέτηση αυτών των παικτών.[40]



Εικόνα 57: Σελίδα Unity Developer για το παιχνίδι μας , Project Bullet

Τελικό βήμα σε αυτή τη διαδικασία είναι η ενεργοποίηση της λειτουργίας Live η οποία θα ενημερώσει την πλατφόρμα Unity ώστε κάθε στιγμή που θα εκτελείται ένα στιγμιότυπο του παιχνιδιού μας, το αντικείμενο NetworkManager.singleton θα δρομολογείται μέσω των εξυπηρετητών Unity. Η καρτέλα Live παρουσιάζει όλα τα απαραίτητα πεδία προς συμπλήρωση καθώς και την εκτίμηση του κόστους της συνδρομητικής υπηρεσίας.



Εικόνα 58: Ρύθμιση Live και εκτίμηση κόστους υπηρεσίας

4.6 Δημιουργία Παίκτη

4.6.1 Χειρισμός Παίκτη

Η δημιουργία παικτών στο επίπεδο του παιχνιδιού είναι μία πιο πολύπλοκη διαδικασία καθώς χρειάζεται να ρυθμιστούν αρκετοί παράγοντες οι οποίο αφορούν την κάμερα από την οποία ο παίκτης θα βλέπει το επίπεδο καθώς και παράγοντες που σχετίζονται με τη θέση του μοντέλου του παίκτη στο χώρο και τις αλληλεπιδράσεις του με άλλα αντικείμενα. Επιπλέον πολυπλοκότητα προστίθεται στην ανάπτυξη των χειρισμών παίκτη όταν αυτός πρόκειται να συμμετάσχει σε ένα διαδικτυακό παιχνίδι καθώς πρέπει να διασφαλίσουμε ότι όλες οι κινήσεις του συγχρονίζονται ομαλά στο δίκτυο. Επομένως, σε αυτή την ενότητα αναπτύσσουμε δύο κλάσεις κώδικα, τις κλάσεις Player Movement και Player Controls επεξηγώντας τη λειτουργία τους. Επίσης, παρουσιάζουμε τα αντικείμενα που χρειάζεται να προσαρτηθούν στο αντικείμενο του παίκτη ώστε ο χειρισμός του να είναι πλήρης. Τέλος, παρουσιάζουμε το μοντέλο παίκτη και την πορεία που οδήγησε στην επιλογή του.

Το script Player Movement (6.10) επικοινωνεί με το περιβάλλον σχεδίασης της Unity για εκχώρηση τιμών που αφορούν την κίνηση του αντικειμένου παίκτη, την περιστροφή του γύρω από τον άξονα Y και την ώθηση που δίνεται στον παίκτη ώστε να μπορεί να πραγματοποιήσει πτήσεις με τη μορφή ομαλών αλμάτων πάνω από αντικείμενα. Στις επόμενες ενότητες θα δούμε ότι το επίπεδο και ο τρόπος παιχνιδιού ενθαρρύνει την πραγματοποίηση αυτών των αλμάτων. Παρακάτω παραθέτουμε επεξήγηση της λειτουργίας των μεθόδων του script καθώς και σχηματική απεικόνιση των στοιχείων που χρειάζεται να προσαρτηθούν στον παίκτη στην όψη παρατήρησης.

```
public void Move(Vector3 myspeed):
```

Με τη μέθοδο Move ανανεώνουμε την τρέχουσα ταχύτητα κίνησης με αυτή που έχει οριστεί από το περιβάλλον ώστε να μοντελοποιηθεί η κίνηση στο script Player Controls.

```
public void Rotate(Vector3 myrotation):
```

Με τη μέθοδο Rotate δίνεται η εντολή περιστροφής μοντέλου παίκτη σύμφωνα με κατάλληλο τρισδιάστατο διάνυσμα που δίνεται ως όρισμα.

```
public void RotateCam(float mycamrotX):
```

Με τη μέθοδο RotateCam εκχωρείται η τιμή των μοιρών κατά τις οποίες μπορεί να περιστρέφεται η κάμερα του παίκτη κατά τη μοντελοποίηση της σε επόμενη μέθοδο.

```
public void GiveThrust (Vector3 mythrust):
```

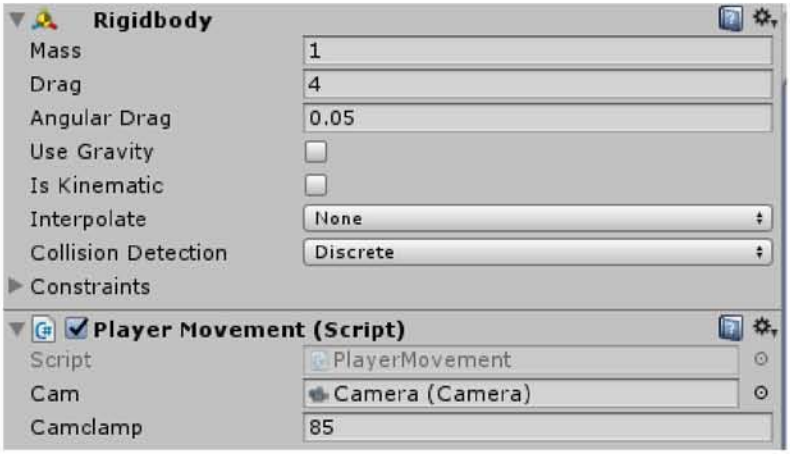
Με την εκτέλεση αυτής της μεθόδου καθορίζουμε το ποσό της ώθησης που θα ασκηθεί στο μοντέλο του παίκτη κατά την πραγματοποίηση άλματος.

```
void DoMoves():
```

Η εκτέλεση της μεθόδου DoMoves μοντελοποιεί την κίνηση και την ώθηση του μοντέλου παίκτη χρησιμοποιώντας τη μηχανή προσομοίωσης αλληλεπιδράσεων φυσικής. Με την εφαρμογή αυτών των κινήσεων στο δομικό στοιχείο Rigidbody του μοντέλου παίκτη δημιουργούμε πιο ρεαλιστικές αντιδράσεις στις συγκρούσεις λόγω ύπαρξης μάζας.

```
void DoRotation():
```

Η μέθοδος DoRotation εφαρμόζει την περιστροφή του παίκτη σύμφωνα με τη μη-αντιμεταθετική επέκταση της θεωρίας των μιγαδικών αριθμών, τα τετραδόνια. Ειδικότερα, με τη χρήση γωνιών Euler σε τετραδόνια δημιουργείται ένα διάνυσμα τριών συντεταγμένων στο οποίο κάθε συντεταγμένη αναπαριστά τις μοίρες της γωνίας περιστροφής προς κάθε άξονα (x,y,z). Για να αποφύγουμε την ανεξέλεγκτη περιστροφή γύρω από τον άξονα X θέτουμε ως όριο το διάστημα (-85,85) μοιρών.

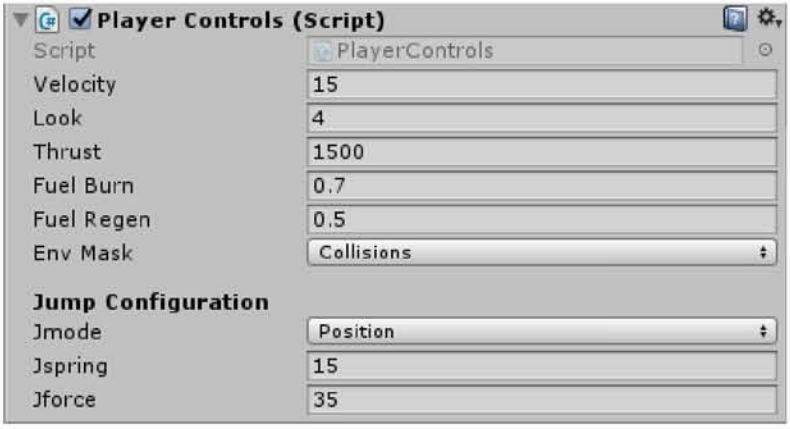


Εικόνα 59: Στοιχεία στερεού σώματος και Player Movement

Συνεχίζοντας την ανάλυση του χειρισμού παίκτη είναι αναγκαία η ανάπτυξη του script Player Controls (6.11) το οποίο θέτει τις τιμές που καθορίζουν την ταχύτητα περιστροφής του παίκτη, την ώθηση καθώς και την οικονομία καυσίμου της μηχανής που παράγει αυτή την ώθηση. Επίσης, σε αυτό το script μοντελοποιούμε τη φυσική των αλμάτων με τη χρήση του στοιχείου ρυθμιζόμενης άρθρωσης σε συνδυασμό με ανίχνευση ακτινών για τη σύγκρουση με το έδαφος και ελέγχουμε την είσοδο ποντικιού του χρήστη για να προσδιορίσουμε κατεύθυνση κίνησης και περιστροφής. Όλες οι παραπάνω ενέργειες χρειάζεται να ανανεώνονται συνεχώς και τοποθετούνται εντός της δομικής μεθόδου Update.

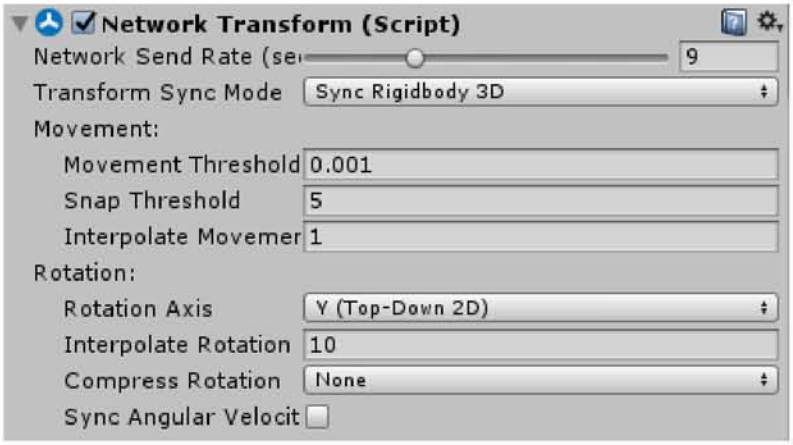
Η χρήση του Player Controls σε συνδυασμό με τις δομικές μεθόδους καθορισμού κουμπιών της Unity οδηγεί στις εξής κινήσεις όταν η κάμερα του παιχνιδιού είναι ενσωματωμένη στο αντικείμενο παίκτη:

- W ή Forward Arrow Key : Μετακίνηση προς την κατεύθυνση που δείχνει η κάμερα.
- A ή Left Arrow Key : Μετακίνηση παίκτη αριστερά.
- S ή Back Arrow Key : Μετακίνηση παίκτη προς την αντίθετη κατεύθυνση που δείχνει η κάμερα.
- D ή Right Arrow Key : Μετακίνηση παίκτη δεξιά.
- Spacebar : Εκτέλεση ομαλού άλματος ώθησης.
- Κίνηση στον άξονα X (2-D) ποντικού : Περιστροφή κάμερας γύρω από τον άξονα Y.
- Κίνηση στον άξονα Y (2-D) ποντικιού : Περιστροφή κάμερας γύρω από τον άξονα X.



Εικόνα 60: Στοιχείο Player Controls

Ένα ακόμη απλό χαρακτηριστικό των περισσότερων First Person Shooters είναι η δυνατότητα κλειδώματος του κέρσορα ώστε να μην εμφανίζεται εντός του παραθύρου με αποτέλεσμα ο παίκτης να χρησιμοποιεί τη διεπαφή του παιχνιδιού. Αυτό το χαρακτηριστικό το συμπεριλαμβάνουμε εντός της μεθόδου Update.Ωστόσο, ο χειρισμός όλων των κινήσεων που αναπτύσσονται πρέπει να μπορεί να συγχρονιστεί στο δίκτυο. Για την επίτευξη του ομαλού συγχρονισμού χρειάζεται η προσάρτηση του στοιχείου Network Transform στο αντικείμενο παίκτη ρυθμίζοντας την παρεμβολή των κινήσεων και των περιστροφών τρισδιάστατου στερεού σώματος.



Εικόνα 61: Στοιχείο Network Transform

4.6.2 Ρυθμίσεις Παίκτη

Σε αυτή την ενότητα παρουσιάζουμε τις βασικές ρυθμίσεις κατά την είσοδο του παίκτη στη σκηνή παιχνιδιού διότι το αντικείμενο παίκτη έχει δικτυακά χαρακτηριστικά και χρειάζεται να επικοινωνεί με άλλες κλάσεις για ανταλλαγή πληροφοριών. Επίσης, το αντικείμενο παίκτη σχεδιάζεται μετά τη σχεδίαση του περιβάλλοντος παιχνιδιού στην ιεραρχία, επομένως χρειάζεται να ρυθμίσουμε τα επίπεδα σχεδίασης ώστε να μην εμφανίζονται περίεργες συμπεριφορές όπως προβολή συνένωσης της επιφάνειας παιχνιδιού με το μοντέλο παίκτη. Ο κώδικας που περιέχει το σύνολο των ρυθμίσεων βρίσκεται στο Setting Player script της παραγράφου 6.12.

Αρχικά, χρειάζεται να ορίσουμε και να ενεργοποιήσουμε τη διεπαφή παίκτη η οποία θα παρέχει χρήσιμες πληροφορίες στον παίκτη κατά τη διάρκεια του παιχνιδιού σε ένα δισδιάστατο καμβά. Επίσης, είναι αναγκαίο να τοποθετήσουμε τον παίκτη σε ένα ξεχωριστό επίπεδο σχεδίασης με ξεχωριστή ετικέτα ώστε να μη δημιουργείται το φαινόμενο της αποκοπής του μοντέλου του παίκτη κατά την επαφή του με άλλα αντικείμενα. Οι ρυθμίσεις επικαλυπτόμενης σχεδίασης βοηθούν στην εξάλειψη αυτών των φαινομένων σε προβολή πρώτου προσώπου αλλά και τρίτου προσώπου. Τέλος, χρειάζεται να δημιουργηθεί μία λίστα ελέγχου των στοιχείων που σχετίζονται με τον παίκτη διότι αρκετά στοιχεία χρειάζεται να είναι απενεργοποιημένα κατά τη σύνδεση του παίκτη είτε απενεργοποιούνται σύμφωνα με ένα γεγονός, όπως για παράδειγμα την καταστροφή του παίκτη. Στη συνέχεια παρουσιάζουμε τις μεθόδους που συγκεντρώνουν το σύνολο των ρυθμίσεων που εφαρμόζουμε στους παίκτες στο Setting Player script.

```
void CmdSetUname (string pID,string uname ):
```

Η εκτέλεση της μεθόδου CmdSetUname αρχικοποιεί τις πληροφορίες παίκτη όπως αυτές ανακτώνται από τη βάση δεδομένων στον εξυπηρετητή.

```
void SetLayerRecursively (GameObject obj , int nlayer):
```

Με αυτή τη μέθοδο αναθέτουμε αναδρομικά όλα τα αντικείμενα της ιεραρχίας του μοντέλου παίκτη σε ένα άλλο επίπεδο σχεδίασης για αποφυγή των φαινομένων επικάλυψης και αποκοπής.

```
public override void OnStartClient():
```

Η μέθοδος OnStartClient καταχωρεί τον παίκτη ως δικτυακό αντικείμενο και του δίνει όλες τις ιδιότητες που παρέχονται από το script διαχειριστή παιχνιδιού το οποίο θα αναλύσουμε σε επόμενη ενότητα που αφορά το περιβάλλον παιχνιδιού .

```
void AssignRemLayer():
```

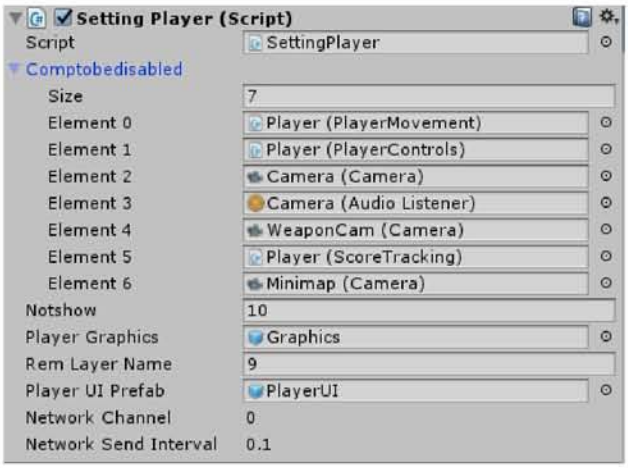
Σε αυτή τη μέθοδο γίνεται καθορισμός του επιπέδου σχεδίασης με βάση τον ακέραιο αριθμό που το αντιπροσωπεύει.

```
void DisableComponents ():
```

Η μέθοδος χειρίζεται μία λίστα με αντικείμενα τα οποία απενεργοποιούνται για την αποφυγή σφαλμάτων σε δικτυακό περιβάλλον πολλών παικτών. Για παράδειγμα, κάθε παίκτης έχει τη δική του κάμερα η οποία είναι εφοδιασμένη με ένα στοιχείο σήμανσης ήχων. Η Unity λειτουργεί σωστά με την ύπαρξη μόνο ενός στοιχείου σήμανσης ήχων ανά σκηνή, επομένως η σύνδεση πολλών θα ήταν προβληματική χωρίς την απενεργοποίηση αυτού του στοιχείου.

```
void OnDisable():
```

Η δομική μέθοδος OnDisable ελέγχει την έξοδο του παίκτη από τη σκηνή παιχνιδιού και άμεσα αφαιρεί τα στοιχεία του από το αντικείμενο διαχειριστή παιχνιδιού.



Εικόνα 62: Στοιχείο Setting Player

4.6.3 Ενέργειες Παίκτη

Σε αυτή την ενότητα παρουσιάζουμε τους βασικούς μηχανισμούς παιχνιδιού οι οποίοι εκτελούνται από το αντικείμενο παίκτη αλλά και τις επιπτώσεις αυτών των μηχανισμών στη δικτυακή σκηνή παιχνιδιού. Καθώς ο βασικός στόχος κάθε παίκτη είναι η καταστροφή των ρομπότ των αντιπάλων του στο αντικείμενο παίκτη πρέπει να προσαρτηθούν πληροφορίες που σχετίζονται με τον αριθμό των καταστροφών που έχουν προκληθεί ώστε να ανανεώνονται σωστά οι λογαριασμοί των παικτών. Επίσης, χρειάζεται να κωδικοποιηθούν οι συμπεριφορές τραυματισμού από αντιπάλους και καταστροφής με την εφαρμογή εφέ σωματιδίων και τη διαχείριση μεταβλητών που καθορίζουν το ποσό τραυματισμού πριν την καταστροφή. Ακολούθως, οι παίκτες οι οποίοι έχουν καταστραφεί και επιθυμούν να συνεχίσουν το παιχνίδι πρέπει να επανατοποθετηθούν στη σκηνή με τη χρήση ειδικών εφέ. Οι παραπάνω ενέργειες αν και εφαρμόζονται στο αντικείμενο τοπικού παίκτη ,πρέπει να είναι ενιαίες για όλους τους συνδεδεμένους παίκτες, επομένως οι μεταβλητές και οι μέθοδοι που αναλαμβάνουν την εκτέλεσή τους χρειάζεται να φέρουν τα κατάλληλα αναγνωριστικά δικτύου ώστε το Unity Networking να διακρίνει την εξουσιοδότηση αυτών των συμπεριφορών στο δίκτυο.

Οι συμπεριφορές που αναφέραμε στην προηγούμενη παράγραφο παρουσιάζονται στο Player script (6.13) και σε αυτό το σημείο επεξηγούμε το ρόλο και τη λειτουργία των μεταβλητών και των μεθόδων που κωδικοποιούν αυτές τις συμπεριφορές.

```
private int currentHealth:
```

Η μεταβλητή currentHealth καθορίζει την τρέχουσα αντοχή σε τραυματισμούς του ρομπότ που ελέγχει ο παίκτης. Η μέγιστη τιμή της έχει καθοριστεί στους 100 πόντους. Η μεταβλητή πρέπει να συγχρονίζεται στο δίκτυο ,επομένως χαρακτηρίζεται ως SyncVar.

```
private bool dead:
```

Η μεταβλητή dead επίσης καθορίζεται ως SyncVar διότι χρειάζεται να πληροφορήσει όλους τους παίκτες ότι ένας παίκτης έχει καταστραφεί.

```
public string uname:
```

Η μεταβλητή uname διαχειρίζεται το όνομα του παίκτη εντός της διεπαφής παίκτη στη σκηνή παιχνιδιού και επειδή όλοι οι παίκτες πρέπει να γνωρίζουν τα ονόματα των αντιπάλων τους η μεταβλητή έχει οριστεί ως SyncVar.

```
public float GetHealth():
```

Η μέθοδος υπολογίζει την τρέχουσα αντοχή του παίκτη σε σχέση με τη μέγιστη τιμή της.

```
public bool Death:
```

Με τη μέθοδο Death εκτελείται η αυτόματη ιδιότητα ανάθεσης και επιστροφής κατά τη μεταγλώττιση του προγράμματος ώστε να επιστρέφεται η τρέχουσα κατάσταση παίκτη.

```
public void SetupPlayer ():
```

Γίνεται αρχικοποίηση ενός παίκτη και η αρμοδιότητα αυτής της αρχικοποίησης κατευθύνεται προς τον εξυπηρετητή.

```
private void CmdNewPlayer():
```

Ο εξυπηρετητής ενημερώνεται για την αρχικοποίηση ενός παίκτη και στέλνει εντολή προς όλα τα στιγμιότυπα προγραμμάτων-πελάτη ώστε η αρχικοποίηση να εκτελεστεί σε αυτά.

```
private void RpcSetupPlayer():
```

Η εντολή αρχικοποίησης η οποία εκτελείται σε όλους τους παίκτες στο δίκτυο ώστε να έχει συγχρονιστεί σωστά. Αυτή η μέθοδος καλεί τη μέθοδο Initialize( ) η οποία θα εφαρμόσει τις αρχικοποιήσεις.

```
public void RpcDealDmg (int howmuch,string enemyPlayer):
```

Η μέθοδος RpcDealDmg εκτελείται σε όλους τους παίκτες και καθορίζει την προέλευση και το ποσό του τραυματισμού ενός παίκτη. Αυτές οι πληροφορίες χρειάζεται να είναι γνωστές από όλους τους παίκτες διότι στο ανταγωνιστικό περιβάλλον παιχνιδιού οι παίκτες πρέπει να δώσουν προτεραιότητα στην καταστροφή αντιπάλων με τη μικρότερη τρέχουσα αντοχή.

```
private void Die (string enemyPlayer):
```

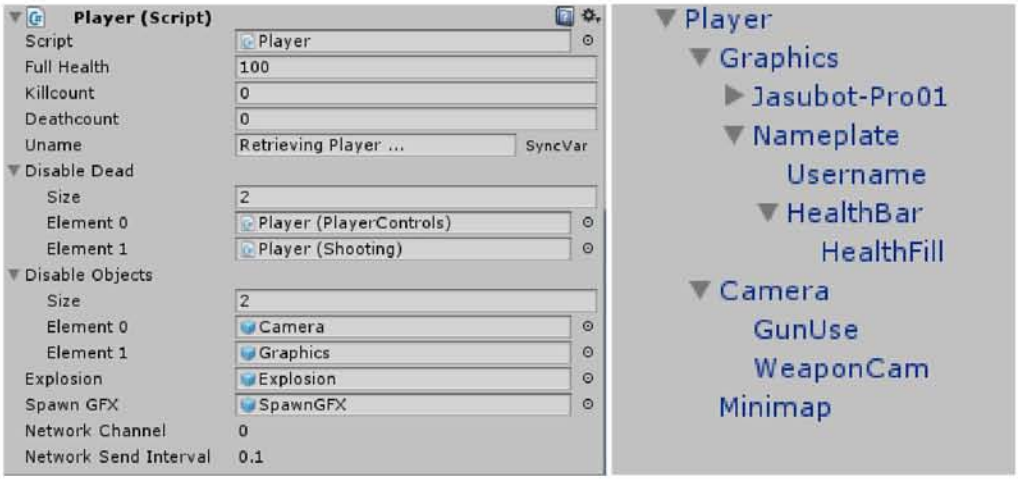
Η μέθοδος Die ορίζει την καταστροφή του παίκτη και δέχεται ως όρισμα τον αντίπαλο προέλευσης. Με την καταστροφή ενός παίκτη επιβραβεύουμε τον αντίπαλο αυξάνοντας τη μεταβλητή killcount και τιμωρούμε τον παίκτη που καταστράφηκε αυξάνοντας τη μεταβλητή deathcount. Επίσης, ο παίκτης ο οποίος καταστρέφεται απενεργοποιείται προσωρινά, η κάμερα πρώτου προσώπου αλλάζει ώστε να φαίνεται μόνο το επίπεδο της σκηνής και προβάλλεται ένα ειδικό εφέ καταστροφής στους άλλους παίκτες.

```
private IEnumerator Respawn ():
```

Η μέθοδος Respawn αναλαμβάνει την επανένταξη του παίκτη μετά από μία μικρή καθυστέρηση. Η αρχικοποίηση του παίκτη γίνεται εκ νέου ώστε να μπορεί κανονικά να συνεχίσει το παιχνίδι.

```
public void Initialize ():
```

Η μέθοδος Initialize αρχικοποιεί τους πόντους ζωής και τα γραφικά του παίκτη προβάλλοντας ένα ειδικό εφέ σωματιδίων κατά την είσοδο του παίκτη στο επίπεδο.



Εικόνα 63: Στοιχείο Player και δομή ιεραρχίας παίκτη

4.7 Δημιουργία Όπλου

4.7.1 Διαχείριση Όπλου

Καθώς το παιχνίδι που αναπτύσσουμε ανήκει στην κατηγορία first person shooter, είναι αναγκαίο να δημιουργήσουμε το περιβάλλον ελέγχου των όπλων που θα χειρίζεται ο παίκτης ώστε στη συνέχεια να προσδιορίσουμε τις λειτουργίες τους και να δημιουργήσουμε προκατασκευασμένα αντικείμενα που θα τα αντιπροσωπεύουν. Ξεκινώντας με το script Weapon Handling (6.14) επιθυμούμε να ορίσουμε συμπεριφορές οι οποίες εφαρμόζονται σε οποιοδήποτε όπλο με σκοπό να δημιουργήσουμε ένα πλαίσιο στο οποίο η προσθήκη επιπλέον όπλων στο μέλλον θα αποτελεί απλή ανακύκλωση κώδικα. Σε αυτό το script ,οι βασικοί μηχανισμοί οι οποίοι ελέγχονται σχετίζονται με την τοποθέτηση του όπλου στη σκηνή παιχνιδιού, τη διαχείριση του προκατασκευασμένου αντικειμένου όπλου και το γέμισμα του όπλου. Ειδικότερα, σε αυτή την ενότητα παρουσιάζουμε αναλυτικά τις μεθόδους του Weapon Handling script.

```
public Weapon GetWeapon ():
```

Η μέθοδος αναλαμβάνει την ανάκτηση αντικειμένου τύπου weapon όπου εμπεριέχονται όλες οι πληροφορίες χειρισμού του όπλου.

```
public WeaponGFX GetWeaponGFX ():
```

Η μέθοδος ανακτά τα γραφικά και τα εφέ σωματιδίων που εφαρμόζονται στο αντικείμενο του όπλου.

```
void Equip (Weapon myweapon):
```

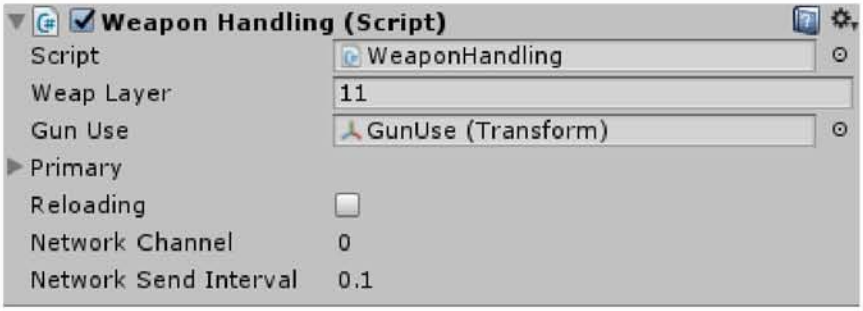
Η μέθοδος Equip τοποθετεί το επιλεγμένο όπλο κοντά στο μοντέλο του παίκτη αναθέτοντας τη δική του κάμερα και αρχικοποιώντας το ως προκατασκευασμένο αντικείμενο. Επίσης, γίνεται αναδρομική ανάθεση των γραφικών του όπλου σε ένα νέο επίπεδο σχεδίασης ώστε να είναι ορατό από όλους τους παίκτες και να μην εμφανίζεται το φαινόμενο της αποκοπής.

```
public void Reload ():
```

Η εκτέλεση της μεθόδου Reload ενεργοποιεί το γέμισμα του όπλου το οποίο γίνεται αποδοτικά μέσα από την εκτέλεση της coroutine ReloadDelay.

```
private IEnumerator ReloadDelay ():
```

Η μέθοδος ReloadDelay εφαρμόζει το γέμισμα του όπλου το οποίο εκτελείται επιβραδύνοντας τη μέθοδο κατά ένα σταθερό χρονικό διάστημα το οποίο καθορίζεται αποκλειστικά από κάθε όπλο.



Εικόνα 64: Αντικείμενο Διαχειριστή όπλου

4.7.2 Ρυθμίσεις και Ενέργειες Όπλου

Εφόσον το πλαίσιο διαχείρισης όπλου έχει δημιουργηθεί, μεταβαίνουμε στη δημιουργία του όπλου που θα χρησιμοποιεί ο παίκτης. Η διαδικασία δημιουργίας του όπλου χωρίζεται σε δύο στάδια, τη ρύθμιση των παραμέτρων απόδοσης του όπλου και την κωδικοποίηση της χρήσης του. Σε αυτή την ενότητα αναλύουμε τον κώδικα που βρίσκεται στα scripts Weapon , WeaponGFX και Shooting (6.15-6.17) ενώ η παρουσίαση του μοντέλου του όπλου θα γίνει σε επόμενη ενότητα όπου θα αναλύουμε όλα τα στοιχεία σχεδίασης του περιβάλλοντος παιχνιδιού.

Η ρεαλιστική απεικόνιση των ικανοτήτων του όπλου είναι πολύ σημαντική διότι οι παίκτες χρειάζεται να λάβουν υπόψη πολλές παραμέτρους οι οποίες δε σχετίζονται μόνο με την ακριβή στόχευση. Συγκεκριμένα, στον επόμενο πίνακα παραθέτουμε τις παραμέτρους που

χαρακτηρίζουν το όπλο μέσω public μεταβλητών ώστε η αλλαγή των τιμών τους να γίνεται εύκολα από το περιβάλλον σχεδίασης της Unity στην περίπτωση που θέλουμε να εξισορροπήσουμε γρήγορα κάποιες συμπεριφορές ώστε να είναι πιο δίκαιες για τον παίκτη.

Μεταβλητή Παραμέτρου	Περιγραφή Παραμέτρου
name	Αναγνωριστικό όνομα όπλου
dmg	Ποσό ζημίας ανά σφαίρα που προκαλεί το όπλο.
range	Μέγιστη απόσταση στην οποία το όπλο είναι αποτελεσματικό απέναντι σε αντίπαλους παίκτες.
fireRate	Ο ρυθμός με τον οποίο το όπλο πυροβολεί.
magCapacity	Ο αριθμός των φορών που μπορεί να πυροβολήσει το όπλο χωρίς να χρειαστεί γέμισμα.
reloadSpeed	Η ταχύτητα γεμίσματος του όπλου.

Πίνακας 3: Παραμετροποίηση αντικειμένου όπλου στο Weapon script

Εκτός από τις παραμέτρους όπλου, στο script WeaponGFX ορίζουμε δύο είδη ειδικών εφέ τα οποία ενεργοποιούνται όταν ο παίκτης χρησιμοποιεί το όπλο. Το πρώτο ειδικό εφέ χρησιμοποιείται για να δείξει ότι ο παίκτης έχει πυροβολήσει και το δεύτερο ειδικό εφέ χρησιμοποιείται για να δείξει το στόχο τον οποίο έχει πετύχει ο παίκτης. Οι παράμετροι όπλου όπως και τα ειδικά εφέ συνδυάζονται στο script Shooting (6.17) το οποίο αναλύουμε παρακάτω.

Στο script Shooting αναπτύσσονται οι εξής μέθοδοι:

```
void CmdFire():
```

Η μέθοδος CmdFire εκτελεί μία εντολή στον εξυπηρετητή ο οποίος καθορίζει την εκτέλεση συναρτήσεων ειδικών εφέ στα προγράμματα-πελάτη.

```
void RpcMuzzleFlash():
```

Η μέθοδος εκτελείται σε όλα τα προγράμματα πελάτη προβάλλοντας ένα ειδικό εφέ σωματιδίων και αναπαράγοντας τον ήχο του όπλου.

```
void CmdOnShot(Vector3 ShotPos, Vector3 Nsurface):
```

Αυτή η μέθοδος εκτελεί μία εντολή στον εξυπηρετητή ο οποίος καθορίζει την εκτέλεση του εφέ που σχετίζεται με την πρόσκρουση σφαίρας σε επιφάνεια στα προγράμματα-πελάτη.

```
void RpcImpactEffect(Vector3 ShotPos, Vector3 Nsurface):
```

Η εκτέλεση της μεθόδου RpcImpactEffect εκτελεί το ειδικό εφέ πρόσκρουσης σε επιφάνεια εφόσον η μέθοδος τροφοδοτηθεί με τη θέση πρόσκρουσης και το ορθοκανονικό διάνυσμα της επιφάνειας πρόσκρουσης ώστε το εφέ να αποδίδεται ρεαλιστικά.

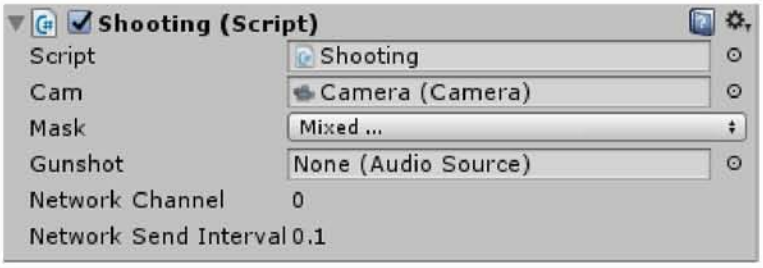
```
void Fire():
```

Η μέθοδος Fire είναι η βασικότερη μέθοδος του script Shooting η οποία εκτελείται αποκλειστικά στο πρόγραμμα-πελάτη που έχει ένα στιγμιότυπο του παιχνιδιού. Αυτή η μέθοδος καλείται με το πάτημα του αριστερού πλήκτρου του ποντικιού είτε μεμονωμένα είτε παρατεταμένα δίνοντας δύο εναλλακτικούς τρόπους χρήσης του όπλου, την ημιαυτόματη και την αυτόματη ρίψη πυρών . Επίσης, με αυτή τη μέθοδο γίνεται ανίχνευση στόχων με την τεχνική της ρίψης ακτινών με αρχή την κάμερα του όπλου και προορισμό το σημείο της επιφάνειας που βρίσκεται στο κέντρο της κάμερας. Τέλος, η μέθοδος Fire καθορίζει τις εντολές εφέ που θα σταλούν στον εξυπηρετητή και εκτελεί ελέγχους γεμίσματος όπλου οι οποίοι μπορούν να πραγματοποιηθούν και χειροκίνητα με το πάτημα του πλήκτρου "R".

```
void CmdPlayerShot (string playerID, int mydmg,string enemyPlayer):
```

Με την εκτέλεση της μεθόδου CmdPlayerShot γίνεται απόδοση του ποσού τραυματισμού στον παίκτη που έχουν πετύχει τα πυρά του όπλου. Ο εξυπηρετητής καθορίζει τον παίκτη στον οποίο θα αποδώσει το ποσό ζημίας ,επομένως η μέθοδος έχει τη μορφή εντολής.

Εικόνα 65: Στοιχείο ειδικών εφέ όπλου



Εικόνα 66: Στοιχείο Shooting

4.8 Περιβάλλον Παιχνιδιού

4.8.1 Επίπεδο και Μοντέλα

Στις προηγούμενες ενότητες επικεντρωθήκαμε στην ανάλυση του κώδικα που πραγματοποιεί τους μηχανισμούς του παιχνιδιού χωρίς να κάνουμε αναφορά σε συγκεκριμένα μοντέλα και γραφικά. Είναι ωστόσο αναγκαίο να παρουσιάσουμε όλα τα αντικείμενα γραφικών τα οποία απαρτίζουν τη σκηνή καθώς και τα αντικείμενα καμβά που είναι ορατά μόνο από την κάμερα παίκτη ώστε να ενισχύσουμε το θέμα που ακολουθεί το παιχνίδι και να δείξουμε ορισμένες σχεδιαστικές επιλογές που βοηθούν στην οπτικοποίηση αυτού του θέματος. Αρχικά, παρουσιάζουμε την ιεραρχία δομής που ακολουθούμε για τη σχεδίαση κάθε επιπέδου στην Εικόνα 66, η οποία περιλαμβάνει τα αντικείμενα διαχείρισης δικτύου και παιχνιδιού όπως επίσης και αντικείμενα που φωτίζουν και διακοσμούν το επίπεδο ως αντικείμενα προσαρτημένα στο αντικείμενο Terrain. Επιπροσθέτως, έχουμε ορίσει την κάμερα που εποπτεύει όλο το επίπεδο καθώς και τις θέσεις στις οποίες θα αρχικοποιούνται οι παίκτες.

Εικόνα 67: Ιεραρχία δομής κάθε επιπέδου παιχνιδιού

Καθώς το παιχνίδι διαδραματίζεται στο μέλλον και το θέμα του εμπλέκει μάχες μεταξύ ρομπότ σε ένα περιβάλλον προσομοίωσης ,επιθυμούμε οι παίκτες να έχουν την αίσθηση μίας σύγχρονης εποχής κατά την αλληλεπίδρασή τους με το παιχνίδι. Επομένως, δημιουργούμε υλικά αντικειμένων τα οποία εμπεριέχουν textures και shaders για την απόδοση υφής και σκίασης που ταιριάζει με το θέμα παιχνιδιού που επιθυμούμε να επιτύχουμε.

Με τη χρήση του περιβάλλοντος σχεδίασης της Unity και του συνόλου των εργαλείων κατασκευής πρωτοτύπων, σχεδιάζουμε τρία επίπεδα τα οποία αναδεικνύουν διαφορετικούς συνδυασμούς των τεχνικών δημιουργίας επιπέδων. Το πρώτο επίπεδο ονομάζεται Playground και στοχεύει να αναδείξει το συγχρονισμό κινήσεων μη ελεγχόμενων αντικειμένων στο δίκτυο με τη χρήση του δομικού στοιχείου Network Animator όπως επίσης και των ειδικών

εφέ σωματιδίων που χρησιμοποιούνται στις πλατφόρμες και στους πυλώνες του επιπέδου. Το δεύτερο επίπεδο που σχεδιάζουμε ονομάζεται Expedition και σε αυτό χρησιμοποιούμε την υψομετρική αντιστοίχιση για τη δημιουργία πολύπλοκων επιφανειών που προσομοιώνουν μη ομαλό έδαφος. Επίσης, σε αυτό το επίπεδο χρησιμοποιούμε συνδυασμό αντικειμένων υψηλού και χαμηλού αριθμού πολυγώνων με εφαρμογή στοιχείων απόδοσης υφής υψηλής ανάλυσης. Το τρίτο και τελευταίο επίπεδο ονομάζεται Maze και έχει κατασκευαστεί αποκλειστικά από εργαλεία πρωτοτύπων διατηρώντας συμμετρική μορφή. Σε αυτό το επίπεδο αναδεικνύονται τεχνικές φωτισμού καθώς εισάγουμε υλικά που προσομοιώνουν το γυαλί. Ειδικότερα, συνδυάζουμε τις κατευθυνόμενες ακτίνες φωτός με ένα σύστημα αντανακλάσεων το οποίο τοποθετείται στο κέντρο του επιπέδου. Αξίζει ακόμη να επισημάνουμε ότι οι κατευθυνόμενες ακτίνες δεν αποτελούν τη μοναδική πηγή φωτός καθώς χρησιμοποιούμε υλικά εκπομπής που φωτίζουν έμμεσα ένα μέρος της σκηνής.

(i) (ii) (iii)

Εικόνα 68: Κάτοψη επιπέδων i)Playground , ii)Expedition , iii)Maze.

Συνεχίζοντας την παρουσίαση των αντικειμένων του παιχνιδιού είναι αναγκαίο να παρουσιάσουμε το μοντέλο του παίκτη όπως και το μοντέλο όπλου τα οποία παρέχονται δωρεάν από καλλιτέχνες στο διαδίκτυο και είναι κατασκευασμένα στο πρόγραμμα Blender.Η τροποποίηση και προσαρμογή τους στο θέμα του παιχνιδιού επιτάχυνε αρκετά τη διαδικασία σχεδίασής του.[41][42]

(i) (ii)

Εικόνα 69: i) Μοντέλο Παίκτη ii) Μοντέλο Όπλου

Τέλος χρειάζεται να αναφερθούμε στα σημεία αρχικοποίησης και επανατοποθέτησης του παίκτη διότι η τοποθέτησή τους είναι σημαντική για τη ροή του παιχνιδιού. Έχουμε επιλέξει να δημιουργήσουμε 4 σημεία τα οποία έχουν αρκετή απόσταση μεταξύ τους ώστε οι παίκτες θα χρειαστεί να κινηθούν για να συναντηθούν. Επίσης, με την εφαρμογή του αλγορίθμου Round Robin διασφαλίζουμε ότι οι παίκτες θα εμφανίζονται σε αυτά τα σημεία κυκλικά χωρίς να υπάρχει επικάλυψη, μειώνοντας την πιθανότητα αρχικοποίησης δύο παικτών στο ίδιο σημείο.

4.8.2 Heads Up Display

Καθώς έχουμε παρουσιάσει τα αντικείμενα που απαρτίζουν τη σκηνή, είναι αναγκαίο να αναφερθούμε και στα αντικείμενα τα οποία είναι προσαρτημένα στην κάμερα πρώτου προσώπου του παίκτη. Πρόκειται για αντικείμενα δισδιάστατου καμβά τα οποία ενεργοποιούνται και απενεργοποιούνται καταλλήλως ώστε να προβάλλουν χρήσιμες πληροφορίες κατά τη διάρκεια του παιχνιδιού. Σε αυτή την ενότητα, θα αναλύσουμε το script UI Manager (6.18) το οποίο ευθύνεται για την οργάνωση όλων αυτών των αντικειμένων και θα γίνουν σύντομες αναφορές στα χαρακτηριστικά κάθε στοιχείου ξεχωριστά. Στον επόμενο πίνακα παραθέτουμε το σύνολο των αντικειμένων που αποτελούν το Heads Up Display με την περιγραφή της λειτουργίας τους.

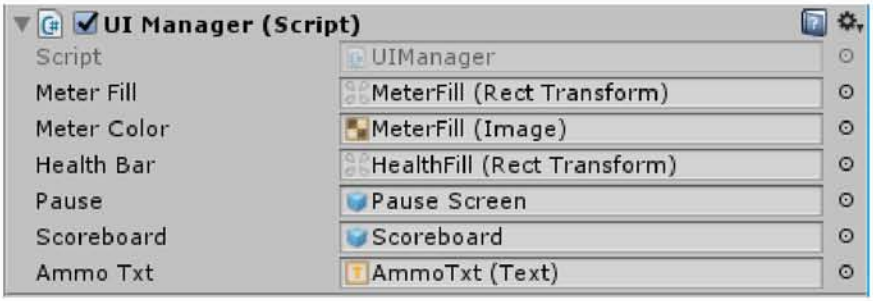
Αντικείμενο Heads Up Display	**Λειτουργία Αντικειμένου**
Health Bar	Προβολή σε πραγματικό χρόνο της τρέχουσας αντοχής σε ζημία.

Fuel Meter	Προβολή σε πραγματικό χρόνο της οικονομίας καυσίμου του παίκτη για πραγματοποίηση ομαλών αλμάτων.
Ammo Count	Ανανέωση του αριθμού σφαιρών που διαθέτει το όπλο.
Minimap	Top Down προβολή κίνησης του παίκτη.
Scoreboard	Προβολή τρέχουσας απόδοσης παικτών.(6.20, 6.21)
Killfeed	Προβολή συμβάντων καταστροφής παικτών με τη μορφή ειδοποιήσεων.(6.22, 6.23)
Pause Screen	Παύση του παιχνιδιού και δυνατότητα εξόδου.(6.24)
Nameplate	Προβολή ονομάτων άλλων παικτών εντός του παιχνιδιού.(6.19)

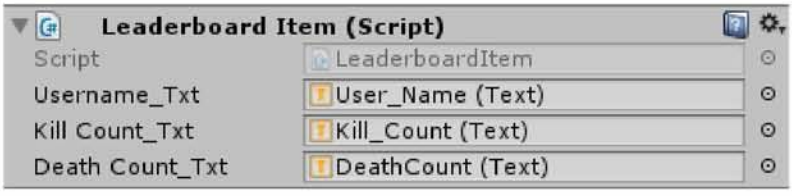
Πίνακας 4: Αντικείμενα του Heads Up Display

Κάθε αντικείμενο του Heads Up Display έχει διαφορετική συμπεριφορά η οποία είτε κωδικοποιείται αποκλειστικά εντός του UI Manager script είτε επικοινωνεί με άλλα scripts για την ανάκτηση πληροφοριών με τη μορφή κειμένου. Η διαδικασία σχεδίασης αυτών των αντικειμένων ξεκινά με τη σχεδίαση του Health Bar και του Fuel Meter, δύο πανομοιότυπων παραλληλογράμμων των οποίων οι διαστάσεις μεταβάλλονται με τη μεταβολή των τιμών της τρέχουσας αντοχής και του τρέχοντος καυσίμου αντίστοιχα. Το χρώμα του αντικειμένου Health Bar παραμένει σταθερό ενώ το χρώμα του Fuel Meter αλλάζει από πράσινο σε κόκκινο ομαλά με χρήση γραμμικής παρεμβολής ώστε να δώσει την αίσθηση αυξομείωσης των καυσίμων. Επίσης, το αντικείμενο Ammo Count συνδέεται με τα script που αφορούν το όπλο του παίκτη και ουσιαστικά αποτελεί ένα μετρητή που ενεργοποιείται με το αριστερό πλήκτρο του ποντικιού. Εξίσου σημαντική είναι και η προσθήκη του αντικειμένου Minimap το οποίο είναι μία κάμερα που ακολουθεί την κίνηση του παίκτη και βοηθά την πλοήγησή του στο επίπεδο. Το αντικείμενο Minimap χρησιμεύει ιδιαίτερα σε επίπεδα ανοιχτού χώρου καθώς δείχνει αντικείμενα τα οποία δε διακρίνονται εύκολα από την κάμερα πρώτου προσώπου ενώ σε κλειστούς χώρους ο παίκτης δε θα είχε μεγάλο όφελος από αυτή την top-down κάμερα.

Προχωρώντας στα αντικείμενα που δεν προβάλλουν μόνο τις τοπικές πληροφορίες κατάστασης του παίκτη , σχεδιάζουμε το αντικείμενο Scoreboard το οποίο προσαρτά ένα σύνολο μικρότερων προκατασκευασμένων αντικειμένων κειμένου ,έχοντας τη δυνατότητα να προβάλει το ιστορικό των καταστροφών στο τρέχον παιχνίδι εύκολα με τη χρήση του πλήκτρου Tab. Στη συνεχεία, γίνεται επέκταση της ιδέας προβολής των καταστροφών με τη σχεδίαση του αντικειμένου Killfeed το οποίο προβάλλει σύντομα την κατάσταση καταστροφής παίκτη χρησιμοποιώντας ετικέτες HTML για να μορφοποιήσει το κείμενο. Το κείμενο του Killfeed λαμβάνει μόνιμα χώρο στον καμβά του UI Manager καθώς τα μηνύματα εξαφανίζονται εντός λίγων δευτερολέπτων. Ακολούθως, χρειάζεται να σχεδιαστεί η οθόνη παύσης η οποία ενεργοποιείται με το πλήκτρο F1 και κατά τη χρήση της ο παίκτης δεν ελέγχει πλέον το μοντέλο ρομπότ και το όπλο αλλά μπορεί είτε να αποσυνδεθεί είτε να επιστρέψει στο παιχνίδι . Τέλος, η σχεδίαση του αντικειμένου Nameplate μέσω του script Nameplate Manager δίνει τη δυνατότητα προβολής του ονόματος αλλά και του Health Bar των παικτών άμεσα πάνω από το μοντέλο του κάθε παίκτη χρησιμοποιώντας ένα μετασχηματισμό που στρέφει την κάμερα στο σημείο που κοιτάει ο παίκτης αυξάνοντας την ορατότητά του. Στην επόμενη εικόνα δείχνουμε τη σύνδεση αυτών των αντικειμένων στο στοιχείο UI Manager του περιβάλλοντος σχεδίασης καθώς και το στοιχείο που προβάλλει την τρέχουσα απόδοση των παικτών.



Εικόνα 70: Δομικό στοιχείο UI Manager



Εικόνα 71: Στοιχείο προβολής τρέχουσας απόδοσης παικτών

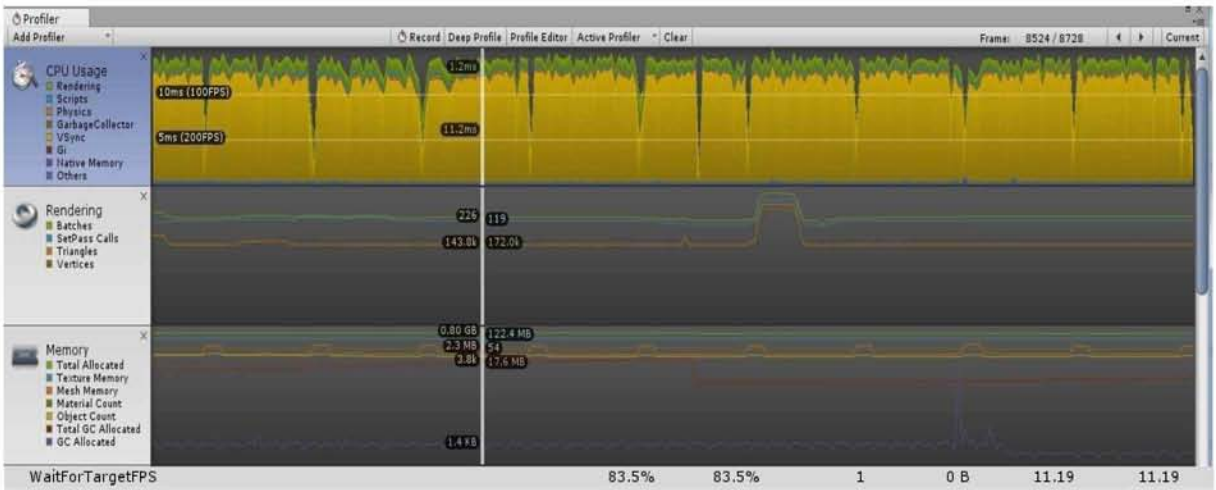
5. Αποτίμηση Εργασίας

5.1 Μελέτη Απόδοσης Παιχνιδιού

Σε αυτή την ενότητα θα εξετάσουμε την απόδοση του παιχνιδιού που αναπτύξαμε χρησιμοποιώντας το εργαλείο Unity Profiler για να καταγράψουμε σε πραγματικό χρόνο μερικές σημαντικές μετρικές συστήματος. Επεξηγώντας τα αποτελέσματα των γραφικών παραστάσεων που θα εξάγει ο Profiler, θα αποκτήσουμε σημαντικές πληροφορίες για τη χρήση του επεξεργαστή και της μνήμης στο σταθερό μας υπολογιστή ώστε να γίνουν εμφανείς οι απαιτήσεις του παιχνιδιού .Επίσης, οι πληροφορίες χρήσης πόρων συστήματος θα μας βοηθήσουν να βελτιώσουμε ή να διατηρήσουμε την απόδοση του παιχνιδιού υψηλή στο μέλλον κατά την επέκτασή του με περισσότερο περιεχόμενο.

Ο έλεγχος του παιχνιδιού έγινε με ανάλυση 1280x720 (native) και ρύθμιση γραφικών "Fantastic" σε υπολογιστή με λειτουργικό σύστημα Windows 7 Home Premium 64-bit με τα εξής χαρακτηριστικά υλικού:

- Επεξεργαστής: Intel(R) Core i7-2600 συχνότητας 3.40GHz υπερχρονισμένος σε 3.70GHz
- Μνήμη RAM: 8GB
- Κάρτα Γραφικών: AMD Radeon HD 6850 2GB



WaitForTargetFPS	83.5%	83.5%	1	0 B	11.19	11.19
▶ Camera.Render	9.3%	0.9%	3	0 B	1.24	0.12
Overhead	1.1%	1.1%	1	0 B	0.16	0.16
▶ Physics.Processing	1.0%	0.4%	1	0 B	0.14	0.05
▶ BehaviourUpdate	0.8%	0.0%	1	30 B	0.11	0.01
▶ FixedBehaviourUpdate	0.8%	0.0%	1	336 B	0.11	0.01
▶ Monobehaviour.OnMouse_	0.4%	0.0%	1	0.6 KB	0.06	0.00
Physics.Simulate	0.3%	0.3%	1	0 B	0.04	0.04
▶ GUI.Repaint	0.3%	0.1%	1	384 B	0.04	0.01
Canvas.BuildBatch	0.2%	0.2%	2	0 B	0.04	0.04
▶ Physics.UpdateBodies	0.2%	0.2%	1	0 B	0.03	0.02
Profiler.FinalizeAndSendFrame	0.1%	0.1%	1	0 B	0.02	0.02
▶ Canvas.RenderOverlays	0.1%	0.0%	1	0 B	0.02	0.00
GameView.GetMainGameViewTargetSize()	0.1%	0.1%	1	24 B	0.02	0.02
Physics.FetchResults	0.1%	0.1%	1	0 B	0.01	0.01
NetworkIdentity.UNetStaticUpdate()	0.1%	0.1%	1	24 B	0.01	0.01
ParticleSystem.Update	0.1%	0.1%	1	0 B	0.01	0.01

Εικόνα 72: Benchmark Παιχνιδιού Project Bullet: Versus

Σύμφωνα με τα αποτελέσματα των μετρήσεων παρατηρούμε ότι το παιχνίδι αποδίδει πολύ καλά στις μέγιστες ρυθμίσεις ποιότητας, δεδομένου του υλικού στο οποίο το εκτελέσαμε. Ειδικότερα, παρατηρούμε ότι ο επεξεργαστής χρειάζεται κατά μέσο όρο 13,4 ms για να σχεδιάσει ένα frame (στιγμιότυπο) του παιχνιδιού με αποτέλεσμα να επιτυγχάνουμε 74,6 frames κάθε δευτερόλεπτο. Ωστόσο, αξίζει να παρατηρήσουμε ότι ο περισσότερος χρόνος και οι περισσότεροι πόροι χρησιμοποιούνται από τη λειτουργία του Vertical Synchronization η οποία ευθύνεται για το συγχρονισμό της σχεδίασης των frames με τη συχνότητα ανανέωσης της οθόνης του υπολογιστή. Επομένως, το Project Bullet: Versus θα μπορούσε να έχει μεγαλύτερη απόδοση χωρίς αυτή τη λειτουργία δημιουργώντας όμως μία λιγότερο ομαλή σχεδίαση της σκηνής σε κάθε οθόνη. Επιπροσθέτως, παρατηρούμε ότι οι μηχανισμοί παιχνιδιού, η διαχείριση των αντικειμένων της σκηνής και η δικτύωση δεν επιβαρύνουν την απόδοση καθώς λιγότερο από 1% του χρόνου σχεδίασης αφιερώνεται σε κάθε ένα από αυτά τα χαρακτηριστικά.

Ένα άλλο εξίσου ενθαρρυντικό αποτέλεσμα έχουμε και από τις μετρήσεις της χρήσης μνήμης καθώς χρειάζεται ανάθεση 800MB συνολικά λόγω της σχεδίασης απλών επιπέδων τα οποία είναι μικρά σε κλίμακα και δεν εμπεριέχουν πολλά ξεχωριστά αντικείμενα. Αυτό το αποτέλεσμα σε συσχέτιση με τις ενδείξεις του επεξεργαστή μας δείχνει ότι το παιχνίδι που

αναπτύξαμε μπορεί να εκτελεστεί στους περισσότερους υπολογιστές είτε τύπου desktop είτε laptop με αξιοπρεπή απόδοση και μας επιτρέπει να μεταφέρουμε το παιχνίδι και σε φορητές πλατφόρμες τύπου tablet μετά από κάποιες μελλοντικές ρυθμίσεις που θα στοχεύουν στην εναρμόνιση του παιχνιδιού με τους φορητούς επεξεργαστές.

5.2 Μελλοντικές Βελτιώσεις και Προσθήκες

Σε αυτή την εργασία εστιάσαμε στην μελέτη της πλατφόρμας Unity και των εργαλείων δικτύου της διεπαφής Unity Networking ώστε ο συνδυασμός αυτών να οδηγήσει στην σχεδίαση και την ανάπτυξη ενός διαδικτυακού First Person Shooter.Οι μηχανισμοί παιχνιδιού που παρουσιάζονται είναι οι απαραίτητοι μηχανισμοί που χαρακτηρίζουν το είδος του παιχνιδιού. Ο αρχικός μας στόχος ήταν να αναπτύξουμε ένα απλό παιχνίδι ώστε να υπάρξει εξοικείωση με τα εργαλεία που χρησιμοποιούν πολυάριθμες ομάδες προγραμματιστών στη δημιουργία των δικών τους ηλεκτρονικών παιχνιδιών. Βέβαια, η μελλοντική ενασχόληση με αυτά τα εργαλεία μπορεί να οδηγήσει σε αλλαγές και προσθήκες οι οποίες θα βελτιώνουν την ποιότητα του παιχνιδιού αυξάνοντας τις διαθέσιμες λειτουργίες του και επιτρέποντάς του να συναγωνιστεί μερικούς από τους πιο δημοφιλείς τίτλους της κατηγορίας.

Ένα βασικό στοιχείο το οποίο θα μπορούσε να βελτιωθεί στο μέλλον είναι τα μοντέλα των παικτών και των όπλων. Το παιχνίδι που παρουσιάσαμε βασίζεται σε τροποποιημένα μοντέλα τα οποία διατίθενται δωρεάν στο διαδίκτυο, επομένως η ποιότητα και η λεπτομέρεια αυτών των μοντέλων δεν συμβαδίζει με τα πιο σύγχρονα First Person Shooters των μεγαλύτερων εταιρειών. Με τη βελτίωση των μοντέλων μπορεί να επιτευχθεί ένα πιο ελκυστικό αποτέλεσμα το οποίο θα μας δώσει τη δυνατότητα να σκηνογραφήσουμε επιπλέον κινήσεις οι οποίες θα χρησιμοποιούν περισσότερα εργαλεία της μηχανής φυσικής της Unity.

Εκτός από τις αλλαγές που σχετίζονται με τα γραφικά του παιχνιδιού , θα μπορούσαμε να προσθέσουμε κάποιους μηχανισμού παιχνιδιού με σκοπό να επιταχύνουμε το παιχνίδι δίνοντας στον παίκτη περισσότερους μικρότερους στόχους προς επίτευξη κατά τη διάρκεια του παιχνιδιού. Ειδικότερα, θα ήταν εφικτό να προσθέσουμε ένα σύστημα αποστολών που θα εξελίσσεται κατά τη διάρκεια του διαδικτυακού παιχνιδιού και θα απαιτεί από τους παίκτες να πραγματοποιήσουν συγκεκριμένες ενέργειες. Με τη χρήση αυτού του συστήματος το παιχνίδι θα μπορούσε να επιβραβεύει περισσότερο τους παίκτες με ξεχωριστά αντικείμενα τα οποία θα αποκτούσαν μέσα από ένα κατάστημα προγραμματισμένο εντός του παιχνιδιού όπου η πρόοδος στο παιχνίδι θα μπορούσε να μεταφραστεί σε ένα μέσο συναλλαγής για την αγορά και την ανταλλαγή αυτών των αντικειμένων . Τέλος, χρειάζεται να αναφέρουμε ότι

στις παραπάνω προσθήκες δεν συγκαταλέγονται οι προσθήκες επιπέδων και όπλων καθώς το παιχνίδι που αναπτύξαμε σε αυτή την εργασία υποστηρίζει ήδη αυτές τις επεκτάσεις καθιστώντας τις προσθήκες αυτού του περιεχομένου ως απλές διαδικασίες ανακύκλωσης κώδικα.

5.3 Συμπεράσματα

Συμπερασματικά, η εκπόνηση αυτής της εργασίας συνέβαλε στην απόκτηση πολλών χρήσιμων γνώσεων που σχετίζονται με τη βιομηχανία ανάπτυξης ηλεκτρονικών παιχνιδιών. Η ενασχόληση με τα εργαλεία της πλατφόρμας Unity βοήθησε στην κατανόηση σημαντικών εννοιών και στην απόκτηση βασικών δεξιοτήτων που αφορούν την ανάπτυξη τρισδιάστατων ηλεκτρονικών παιχνιδιών. Βέβαια, πολύ σημαντική ήταν και η συμβολή της κοινότητας Unity η οποία διαθέτει ένα μεγάλο όγκο διδακτικού υλικού με πολλές αναφορές σε προγραμματιστικές πρακτικές και τεχνικές σχεδίασης που μπορούν να χρησιμοποιηθούν για την ανάπτυξη κάθε κατηγορίας ηλεκτρονικού παιχνιδιού. Η αλληλεπίδραση με αυτή την κοινότητα και η μελέτη του υλικού[43][44][45] που προσφέρει βοήθησε σημαντικά στην εξοικείωση με την πλατφόρμα και την αξιοποίηση διαθέσιμων πόρων για τις ανάγκες του παιχνιδιού που αναπτύσσουμε. Η εκπόνηση της εργασίας πραγματοποιήθηκε μεθοδικά χωρίς ιδιαίτερα προβλήματα αναδεικνύοντας την εφαρμογή ενός συνδυασμού γνώσεων από μαθήματα γραφικών υπολογιστών , βάσεων δεδομένων και δικτύων στην πλατφόρμα Unity.

6.ΠΑΡΑΡΤΗΜΑΤΑ

6.1 Login Script

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

public class Login : MonoBehaviour
{

    //Enabling and disabling GameObjects in the Login Menu UI
    public GameObject loginParent;
    public GameObject registerParent;

    public GameObject loadingParent;

    //Input field objects in the Login Menu scene
    public InputField Login_UsernameField;
    public InputField Login_PasswordField;
    public InputField Register_UsernameField;
    public InputField Register_PasswordField;
    public InputField Register_ConfirmPasswordField;


    //Error Message Textboxes
    public Text Login_ErrorText;
    public Text Register_ErrorText;


    //Reset UI at start
    void Awake()
    {
        ResetAllUIElements();
    }

    //Leaves all fields blank
    void ResetAllUIElements()
    {

        Login_UsernameField.text = "";
        Login_PasswordField.text = "";
        Register_UsernameField.text = "";
        Register_PasswordField.text = "";
        Register_ConfirmPasswordField.text = "";

        Login_ErrorText.text = "";
        Register_ErrorText.text = "";

    }

    //Sending a request to the Database Control server to log a player in the game
    //given the username and password.
    IEnumerator LoginUser(string playerUsername,string playerPassword)
    {
        IEnumerator loginRequest = DCF.Login(playerUsername, playerPassword);
        while (loginRequest.MoveNext())
        {
```

```
            yield return loginRequest.Current;
        }
        string response = loginRequest.Current as string;

        //if the user is verified then log him in the database
        if (response == "Success")
        {

            ResetAllUIElements();
            AccountManagement.singleton.LogIn(playerUsername,playerPassword);
            loadingParent.gameObject.SetActive(false);

        }
        else
        {
            //if user is not verified go back to the login UI
            loadingParent.gameObject.SetActive(false);
            loginParent.gameObject.SetActive(true);
            if (response == "UserError")
            {
                //Wrong Username Error
                Login_ErrorText.text = "Unable to find username";
            }
            else
            {
                if (response == "PassError")
                {
                    //Wrong Password Error
                    Login_ErrorText.text = "Incorrect Password";
                }
                else
                {
                    //General Error Message
                    Login_ErrorText.text = "An error occured.Try again.";
                }
            }
        }
    }
    IEnumerator RegisterUser(string playerUsername , string playerPassword)
    {
        //Sending a Register Request and initializing user info for player statistics
        IEnumerator registerRequest = DCF.RegisterUser(playerUsername, playerPassword,
"[KillCount]0/[DeathCount]0");
        while (registerRequest.MoveNext())
        {
            yield return registerRequest.Current;
        }
        string response = registerRequest.Current as string;

        //if request was successful proceed by logging in the registered player
        if (response == "Success")
        {

            ResetAllUIElements();
            loadingParent.gameObject.SetActive(false);
            AccountManagement.singleton.LogIn(playerUsername, playerPassword);

        }
        else
        {
            //if an error occurs go back to the UI
            loadingParent.gameObject.SetActive(false);
```

```
            registerParent.gameObject.SetActive(true);
            if (response == "UserError")
            {
                //Error notifying that the username is taken
                Register_ErrorText.text = "Error: Username Already Taken";
            }
            else
            {
                //Generic register error text
                Login_ErrorText.text = "An error occured.Try again.";
            }
        }
    }

    //Action when Login button is pressed
    public void Login_LoginButtonPressed()
    {

        string    playerUsername = Login_UsernameField.text;
        string    playerPassword = Login_PasswordField.text;

        //Password Length Check
        if (playerUsername.Length > 3)
        {
            if (playerPassword.Length > 5)
            {

                loginParent.gameObject.SetActive(false);
                loadingParent.gameObject.SetActive(true);
                StartCoroutine(LoginUser(playerUsername,playerPassword));
            }
            else
            {
                //Error that the password is too short
                Login_ErrorText.text = "Error: Password is too short";
            }
        }
        else
        {
            //Error that username is too short
            Login_ErrorText.text = "Error: Username is too short";
        }
    }

    //Actions activated when the register button in the login scene is pressed
    public void Login_RegisterButtonPressed()
    {

        ResetAllUIElements();
        loginParent.gameObject.SetActive(false);
        registerParent.gameObject.SetActive(true);
    }

    //Actions when the Register button on the register screen is pressed
    public void Register_RegisterButtonPressed()
    {

        string   playerUsername = Register_UsernameField.text;
        string   playerPassword = Register_PasswordField.text;
        string confirmedPassword = Register_ConfirmPasswordField.text;

        //Check the password length and be sure to cross-validate it.
```

```
        if (playerUsername.Length > 3)
        {
            if (playerPassword.Length > 5)
            {

                if (playerPassword == confirmedPassword)
                {
                    //If password is ok register the user
                    registerParent.gameObject.SetActive(false);
                    loadingParent.gameObject.SetActive(true);
                    StartCoroutine(RegisterUser(playerUsername, playerPassword));
                }
                else
                {
                    //If cross-validity check fails,display this.
                    Register_ErrorText.text = "Error: Password's don't Match";
                }
            }
            else
            {
                //Error when password is too short
                Register_ErrorText.text = "Error: Password too Short";
            }
        }
        else
        {
            //Error when username is too short
            Register_ErrorText.text = "Error: Username too Short";
        }
    }

    //Going back to the UI when pressing the back button
    public void Register_BackButtonPressed() {

        ResetAllUIElements();
        loginParent.gameObject.SetActive(true);
        registerParent.gameObject.SetActive(false);
    }


    //Logging out the user properly by also adjusting the parameters in the
AccountManagement instance.
    public void LoggedIn_LogoutButtonPressed() {

        ResetAllUIElements();

        AccountManagement.singleton.LogOut();

        loginParent.gameObject.SetActive(true);


    }
}
```

6.2 Account Management Script

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

```
using UnityEngine.SceneManagement;

//Class responsible for logging in and out players
//and transfering data to the database
public class AccountManagement : MonoBehaviour {

    public static AccountManagement singleton;

    //Account Management runs on every scene until the player is signed out.
    void Awake()
    {
        if (singleton != null)
        {
            Destroy(gameObject);
            return;
        }

        singleton = this;
        DontDestroyOnLoad(this);
    }

    public static string playerUsername { get; protected set; }
    private static string playerPassword = "";

    public static bool ActiveUser { get; protected set; }


    //Specifing scene transitions
    public string loggedInSceneName = "Skirmish";
    public string loggedOutSceneName = "LOGIN_DB";

    //Delegate with the method that handles the data as a variable
    public delegate void OnDataReceivedCallback(string data);

    //Logging out an Active User
    public void LogOut()
    {
        playerUsername = "";
        playerPassword = "";

        ActiveUser = false;

        Debug.Log("User logged out!");

        SceneManager.LoadScene(loggedOutSceneName);
    }

    //Logging in an incoming user
    public void LogIn(string username, string password)
    {
        playerUsername = username;
        playerPassword = password;

        ActiveUser = true;

        Debug.Log("Logged in as " + username);

        SceneManager.LoadScene(loggedInSceneName);
    }

    public void SendData(string data)
    {
```

```
        //Sending a request to the database in order to transfer player data
        if (ActiveUser)
        {
            //ready to send request
            StartCoroutine(ActivateQuery(playerUsername, playerPassword, data));
        }
    }

    //Query sending the data passed as an arguement
    IEnumerator ActivateQuery(string username, string password, string data)
    {
        IEnumerator toBeSent = DCF.SetUserData(username, password, data);
        while (toBeSent.MoveNext())
        {
            yield return toBeSent.Current;
        }
        string sentResult = toBeSent.Current as string;
        if (sentResult == "ContainsUnsupportedSymbol")
        {
            //Unsupported symbol '-' error handling
            Debug.Log("Error Uploading Data to the Database");
        }
        if (sentResult == "Error")
        {
            Debug.Log("Unsupported Symbol '-' Detected");
        }
    }

    public void GetData(OnDataReceivedCallback onDataReceived)
    {
        //A request to receive player data for an active user is made

        if (ActiveUser)
        {
            //sending a request as a coroutine
            StartCoroutine(GetQueryResult(playerUsername,              playerPassword,
onDataReceived));
        }
    }

    //Method that handles the result of the query
    IEnumerator      GetQueryResult(string      username,      string      password,
OnDataReceivedCallback onDataReceived)
    {
        string data = "ERROR";

        IEnumerator toBeRetrieved = DCF.GetUserData(username, password);
        while (toBeRetrieved.MoveNext())
        {
            yield return toBeRetrieved.Current;
        }
        string retrievedResult = toBeRetrieved.Current as string;

        //Error handling for retrieval failure and unsupported symbols
        if (retrievedResult == "Error")
        {

            Debug.Log("Error Getting Data from the Database");
        }
        else
        {
            if (retrievedResult == "ContainsUnsupportedSymbol")
```

```
            {

                Debug.Log("Unsupported Symbol '-' Detected");
            }
            else
            {
                //Data received in returned.text variable
                string DataRecieved = retrievedResult;
                data = DataRecieved;
            }
        }

        if (onDataReceived != null)
        {
            onDataReceived.Invoke(data);
        }
    }


}
```

6.3 Hosting Script

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

//Class responsible for initiating a match
//Functions:
//1)Choose a level to play
//2)Setting the room name and limit(fixed)
//3)Create a joinable room with Unity Networking
public class Hosting : MonoBehaviour {

    //Fixed player limit (4) due to UNet CCU restrictions
    [SerializeField]
    private uint lobbysize = 4;

    private string lobbyName;

    private NetworkManager nManager;

    public int MapSelector;

    //Safety check for a valid Network Manager instance.
    //Automatically start the matchmaker because the game
    //has to be online.
    //Initialize level selection.
    void Start()
    {
        nManager = NetworkManager.singleton;
        if(nManager.matchMaker==null)
        {
            nManager.StartMatchMaker();
        }

        MapSelector = 1;
    }
```

```
    //Adjusting the functionality of the hosting input
    //field and map selector buttons
    public void SetName(string thename)
    {
        lobbyName = thename;
    }

    public void SetMap1()
    {
        MapSelector = 1;
    }

    public void SetMap2()
    {
        MapSelector = 2;
    }

    public void SetMap3()
    {
        MapSelector = 3;
    }

    //The network manager object sets the online and the
    //offline scene explicitly.We interchange with MapSelector
    //by overriding that functionality manually.
    public void CreateLobby()
    {
        if(lobbyName !="" && lobbyName!=null)
        {
            Debug.Log("Creating Lobby: " + lobbyName +"for " + lobbysize + "
players.");
            if (MapSelector == 1)
            {
                nManager.onlineScene = "Level1";
            }
            if(MapSelector == 2)
            {
                nManager.onlineScene = "Level2";
            }
            if (MapSelector == 3)
            {
                nManager.onlineScene = "Level3";
            }
            //Establish connection with the default Unity Networking vallues.
            //The level will be joinable for connected clients as long as the
            //player limit is not exceeded.No elo requirements are set.
            nManager.matchMaker.CreateMatch(lobbyName,         lobbysize,         true,
"","","",0,0,nManager.OnMatchCreate);
        }
    }

}
```

6.4 : Joining Script

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

```
using UnityEngine.Networking.Match;
using System.Collections;

//Class responsible for handling connections
//to host-generated lobbies so that up to 3 more
//clients can connect.
public class Joining : MonoBehaviour
{
    //Each lobby is displayed in a list on the game browser
    List<GameObject> lobbyList = new List<GameObject>();

    [SerializeField]
    private Text status;

    [SerializeField]
    private GameObject LobbyItemPrefab;

    [SerializeField]
    private Transform LobbyParent;

    private NetworkManager networkManager;

    //On start of the client , MatchMaker starts
    //and the lobby list is refreshed to display current lobbies.
    void Start()
    {
        networkManager = NetworkManager.singleton;
        if (networkManager.matchMaker == null)
        {
            networkManager.StartMatchMaker();
        }

        RefreshLobbies();

    }

    //Calls ListMatches to display active games
    public void RefreshLobbies()
    {
        ClearLobbyList();
        if(networkManager.matchMaker ==null)
        {
            networkManager.StartMatchMaker();
        }
        networkManager.matchMaker.ListMatches(0, 20, "", true, 0, 0, Lobbies);
        status.text = "Loading Lobbies...";
    }

    //Handling the event of finding a lobby by setting up a join callback
    //Handling the event of not finding a lobby by updating the status text
    public void Lobbies(bool success, string extendedInfo, List<MatchInfoSnapshot>
matches)
    {
        status.text = "";

        if (!success || matches == null)
        {
            status.text = "Could not find a lobby";
            return;
        }
```

```
        //Iteratively add a lobby item prefab to the list
        foreach (MatchInfoSnapshot match in matches)
        {
            GameObject matchItem = Instantiate(LobbyItemPrefab);
            matchItem.transform.SetParent(LobbyParent);

            //Prefab follows parent
            matchItem.transform.localScale = LobbyParent.localScale;
            LobbyListItem mylobbyItem = matchItem.GetComponent<LobbyListItem>();
            if (mylobbyItem != null)
            {
                mylobbyItem.Setup(match,JoinLobby);
            }



            lobbyList.Add(matchItem);

        }

        if (lobbyList.Count == 0)
        {
            status.text = "No lobbies available";
        }

    }

    //Clears the game browser from lobbies
    void ClearLobbyList()
    {
        for (int i = 0; i < lobbyList.Count; i++)
        {
            Destroy(lobbyList[i]);
        }
        lobbyList.Clear();
    }

    //Establishing a direct connection to the host through the matchmaker.
    //Adjusting the join delay to connect to active games
      public void JoinLobby(MatchInfoSnapshot mymatch)
      {
            Debug.Log("Joining Game " + mymatch.name);

            networkManager.matchMaker.JoinMatch(mymatch.networkId,        "","","",0,0,
networkManager.OnMatchJoined);

            StartCoroutine(WaitForJoin());

    }

    //Handling failure to join inactive games that are still displayed on the game
browser
    //by initiating a countdown and timing out the connection.
    IEnumerator WaitForJoin()
    {
        ClearLobbyList();

        int countdown = 10;
        while (countdown > 0)
        {
            status.text = "JOINING... (" + countdown + ")";
```

```
            yield return new WaitForSeconds(1);

            countdown--;
        }

        // On connection failure
        status.text = "Failed to connect.";
        yield return new WaitForSeconds(1);

        //Update the information on the network manager and drop the connection to the
inactive lobby
        MatchInfo matchInfo = networkManager.matchInfo;
        if (matchInfo != null)
        {
            networkManager.matchMaker.DropConnection(matchInfo.networkId,
matchInfo.nodeId, 0, networkManager.OnDropConnection);
            networkManager.StopHost();
        }

        RefreshLobbies();

    }

}
```

6.5 Lobby List Item Script

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking.Match;

//Class responsible for setting the properties of the Lobby
//prefabricated item and enabling the ability to join.
public class LobbyListItem : MonoBehaviour {

    //Setting up delegate that makes the lobby joinable
    public delegate void JoinLobbyDelegate (MatchInfoSnapshot mymatch );
    private JoinLobbyDelegate joinLobbyCallback;

    [SerializeField]
    private Text LobbyText;

    private MatchInfoSnapshot match;

    //Setting lobby information and initializing callback.
    public void Setup(MatchInfoSnapshot mymatch,JoinLobbyDelegate myjoinLobbyCallback)
    {
        match = mymatch;
        joinLobbyCallback = myjoinLobbyCallback;

        LobbyText.text = match.name + " (" + match.currentSize + "/" + match.maxSize +
")";
    }

    //Invoking the delegate to trigger the join event and connect to the game

    public void JoinLobby ()
    {
        joinLobbyCallback.Invoke(match);
    }
```

```
}
```

6.6 User Account Script

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

//Simple Class displaying the username on the UI
//and signing out the player when pressing the
//sign out button of the Skirmish scene
public class UserAccount : MonoBehaviour {

    public Text userText;

    void Start()
    {
        if (AccountManagement.ActiveUser == true)
        {
            userText.text = AccountManagement.playerUsername;
        }
    }

    public void SignOutButton()
    {
        if (AccountManagement.ActiveUser == true)
        {
            AccountManagement.singleton.LogOut();
        }
    }
}
```

6.7 User Data Script

```
using UnityEngine;
using System;

//Class responsible for conversion of player statistics
//Player statistics are stored as a string in Database Control
//but they are handled as integers in the game.
//A simple parser is also used to translate that conversion over to the database
public class UserData : MonoBehaviour {

    private static string KILL_TAG = "[KillCount]";
    private static string DEATH_TAG = "[DeathCount]";

   // Data conversion methods
    public static int KillDataConversion(string data)
    {
        return int.Parse (StringToValue(data, KILL_TAG));
    }

    public static int DeathsDataConversion(string data)
    {
```

```
        return int.Parse(StringToValue(data, DEATH_TAG));
    }

    public static string ValuesToString(int killcount, int deathcount)
    {
        return KILL_TAG + killcount + "/" + DEATH_TAG + deathcount;
    }

    // '/' Delimited parser
    private static string StringToValue(string data, string symbol)
    {
        string[] parts = data.Split('/');

        foreach (string part in parts)
        {
            if (part.StartsWith(symbol))
            {
                return part.Substring(symbol.Length);

            }
        }

        Debug.LogError(symbol + "not found in" + data);
        return "";

    }

}
```

6.8 Score Tracking Script

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

//Class dedicated to keeping track of the players killcount and deathcount in real-
time
[RequireComponent(typeof(Player))]
public class ScoreTracking : MonoBehaviour {

    int killsLatest = 0;
    int deathsLatest = 0;

    Player player;

       // Starting the tracking coroutine from
    //the beginning of the match
       void Start ()
    {
        player = GetComponent<Player>();
        StartCoroutine(ScoreTracker());

       }

    void OnDestroy()
    {
        if (player != null)
        {
            Tracking();
        }
```

```
    }

    //Method that efficiently tracks the statistics without stopping
    IEnumerator ScoreTracker ()
    {

        while (true)
        {

            yield return new WaitForSeconds(10f);
            Tracking();
        }


    }

    //Communicate with the User Account Manager
    void Tracking()
    {
        if (AccountManagement.ActiveUser == true)
        {
            AccountManagement.singleton.GetData(ReceiveHandle);
        }
    }

    //Do the necessary data calculations to update the current and the
    //all time statistics
    void ReceiveHandle(string data)
    {

        if (player.killcount <= killsLatest && player.deathcount <= deathsLatest)
        {

            return;
        }

        int killsThisGame = player.killcount - killsLatest;

        int deathsThisGame = player.deathcount - deathsLatest;



        int killcount = UserData.KillDataConversion(data);

        int deathcount = UserData.DeathsDataConversion(data);

        int updatedKillcount = killsThisGame + killcount;
        int updatedDeathcount = deathsThisGame + deathcount;

        string      updatedData      =      UserData.ValuesToString(updatedKillcount,
updatedDeathcount);

        Debug.Log("Tracking" + updatedData);

        killsLatest = player.killcount;
        deathsLatest = player.deathcount;

        AccountManagement.singleton.SendData(updatedData);

    }
```

```
}
```

6.9 Player Statistics Script

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

//Class that displays the entirety of players stats in a
//"Combat Record" format.
public class PlayerStatistics : MonoBehaviour {

    public Text deathCount;

    public Text killCount;

    void Start()
    {
        if (AccountManagement.ActiveUser == true)
        {
            AccountManagement.singleton.GetData(onDataRecieved);
        }
    }

    //Ranking system and kill/death ratio displayed together with
    //killcount and deathcount
    void onDataRecieved (string data)
    {
        string rank = "Rookie";
        int kills, deaths;
        float kdratio;
        if(killCount == null || deathCount == null)
        {
            return;
        }

        kills = UserData.KillDataConversion(data);
        deaths = UserData.DeathsDataConversion(data);

        if( kills>=5 && kills < 10)
        {
            rank = "Obsidian";
        }
        if (kills>=10 && kills < 20)
        {
            rank = "Onyx";
        }
        if (kills >= 20 && kills < 30)
        {
            rank = "Pearl";
        }
        if (kills >= 30 && kills < 40)
        {
            rank = "Pyrite";
        }
        if (kills >= 40 && kills < 50)
        {
            rank = "Ruby";
```

```
        }
        if (kills >= 50 && kills < 60)
        {
            rank = "Sapphire";
        }
        if (kills >= 60 && kills < 70)
        {
            rank = "Topaz";
        }
        if (kills >= 70 && kills < 80)
        {
            rank = "Zircon";
        }
        if (kills >80)
        {
            rank = "Zircon";
        }
        //Leaving the ranking system open-ended for future improvements

        if (deaths == 0)
        {
            kdratio = (float)kills;
        }
        else
        {
            kdratio = (float)kills / deaths;
        }

        //Display statistics
        killCount.text = kills.ToString() + " kills" + "\r\n" + "Rank: " + rank +
"\r\n" + "K/D Ratio: " + kdratio;
        deathCount.text = deaths.ToString() + " deaths";
        Debug.Log(data);
    }
}
```

6.10 Player Movement Script

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

//Game Logic class that models player movements by setting the rotation and
transformation properties.
//Adjusting the First Person Camera ,making sure everything displays correctly.
[RequireComponent(typeof(Rigidbody))]
public class PlayerMovement : MonoBehaviour {

    [SerializeField]
    private Camera cam;

    private Vector3 speed = Vector3.zero;
    private Vector3 thrust = Vector3.zero;
    private Vector3 rotation = Vector3.zero;
    private float camRotX = 0f;

    private float currentRotX = 0f;

    [SerializeField]
```

```
private float camclamp = 85f;

private Rigidbody rigid;

void Start()
{
    rigid = GetComponent<Rigidbody>();

}
//Pass Movement
public void Move(Vector3 myspeed)
{
    speed = myspeed;
}
//Turn Player

public void Rotate(Vector3 myrotation)
{
    rotation = myrotation;
}
//Turn Camera
public void RotateCam(float mycamrotX)
{
    camRotX = mycamrotX;
}
//Give Thrust to player
public void GiveThrust (Vector3 mythrust)
{
    thrust = mythrust;
}

//Runs at fixed time
void FixedUpdate ()
{
    DoMoves();
    DoRotation();
}

//Movements Performed
void DoMoves()
{
    if (speed !=Vector3.zero)
    {
        rigid.MovePosition(rigid.position + speed * Time.fixedDeltaTime);
    }

    if (thrust != Vector3.zero)
    {
        rigid.AddForce(thrust * Time.fixedDeltaTime, ForceMode.Acceleration);
    }
}

//Rotation Performed
void DoRotation ()
{
    rigid.MoveRotation(rigid.rotation * Quaternion.Euler(rotation));
    if (cam!=null)
    {

        currentRotX -= camRotX;
        currentRotX = Mathf.Clamp(currentRotX, -camclamp, camclamp);
```

```
            cam.transform.localEulerAngles = new Vector3(currentRotX, 0f, 0f);
        }
    }
}
```

6.11 Player Controls Script

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

//Game Logic Class that sets the controls for each player
//Class functionality
//1) Movement Speed
//2)Look Sensitivity
//3)Player Jumping with Physics Engine
//4)Cursor Locking

[RequireComponent(typeof(ConfigurableJoint))]

[RequireComponent(typeof(PlayerMovement))]
public class PlayerControls : MonoBehaviour {

    //Setting up default player parameters that can be tweaked in the editor
    [SerializeField]
    private float velocity = 6f;

    [SerializeField]
    private float look= 3.5f;

    [SerializeField]
    private float thrust = 900f;

    [SerializeField]
    private float FuelBurn = 0.8f;

    [SerializeField]
    private float FuelRegen = 0.3f;

    private float FuelAmount = 1f;

    public float GetFuel()
    {
        return FuelAmount;
    }

    [SerializeField]
    private LayerMask envMask;

    //Former obsolete usage of the mode used only to add smoothness in the later
versions of Unity
    [Header("Jump Configuration")]
    [SerializeField]
    private JointDriveMode jmode = JointDriveMode.Position;
    [SerializeField]
    private float jspring = 20f;
    [SerializeField]
    private float jforce = 35f;
```

```
private ConfigurableJoint cjoint;
private PlayerMovement motor;

void Start()
{
    motor = GetComponent<PlayerMovement>();
    cjoint = GetComponent<ConfigurableJoint>();
    JumpSettings(jspring);
}

void Update()
{
    //Toggling Cursor Locking depending on the state of the game (paused/running)

    if (PauseGame.freeze == true)
    {
        if(Cursor.lockState!= CursorLockMode.None)
        {
            Cursor.lockState = CursorLockMode.None;
        }

        motor.Move(Vector3.zero);
        motor.Rotate(Vector3.zero);
        motor.RotateCam(0f);
        return;
    }

    if (Cursor.lockState != CursorLockMode.Locked)
    {
        Cursor.lockState = CursorLockMode.Locked;
    }

    //Spring Joint On Object Jumps
    RaycastHit thehit;
    if (Physics.Raycast (transform.position,Vector3.down,out thehit,100f,envMask))
    {
        cjoint.targetPosition = new Vector3(0f,-thehit.point.y, 0f);
    }else
    {
        cjoint.targetPosition = new Vector3(0f, 0f, 0f);
    }

    //Movement Vectorization
    float XaxisMov = Input.GetAxisRaw("Horizontal");
    float ZaxisMov = Input.GetAxisRaw("Vertical");

    Vector3 HorMov = transform.right * XaxisMov;
    Vector3 VerMov = transform.forward * ZaxisMov;


    Vector3 speed = (HorMov + VerMov).normalized * velocity;

    motor.Move(speed);

    //Find Rotation
    float yRot = Input.GetAxisRaw("Mouse X");

    Vector3 rot = new Vector3(0f, yRot, 0f) * look;

    //Apply Rotation
    motor.Rotate(rot);
```

```
        //Find Rotation of Camera
        float xRot = Input.GetAxisRaw("Mouse Y");

        float camrotX = xRot * look;

        //Apply Camera Rotation
        motor.RotateCam(camrotX);

        //Model Thust
        Vector3 mythrust = Vector3.zero;

        if (Input.GetButton("Jump") && FuelAmount > 0f)
        {
            FuelAmount = FuelAmount - FuelBurn * Time.deltaTime;

            if(FuelAmount >= 0.01f)
            {
                mythrust = Vector3.up * thrust;
                JumpSettings(0f);
            }

        } else
        {

            FuelAmount = FuelAmount + FuelRegen * Time.deltaTime;

            JumpSettings(jspring);
        }

        FuelAmount = Mathf.Clamp(FuelAmount, 0f, 1f);

        //Apply thrust
        motor.GiveThrust(mythrust);
    }

    //Setting the configurable joint for smooth movement
    private void JumpSettings (float myspring)
    {
        cjoint.yDrive = new JointDrive { mode = jmode, positionSpring = myspring,
maximumForce = jforce };
    }

}
```

6.12 Setting Player Script

```
using UnityEngine;
using UnityEngine.Networking;

//Game Logic Class responsible for managing the visible
//player components in the scene and how the player is drawn.
//Further denoting the registration and the un-registration of
//the player in the scene.

[RequireComponent(typeof(Player))]
[RequireComponent(typeof(PlayerControls))]
public class SettingPlayer : NetworkBehaviour {
```

```
//Serializing the fields and the layers to be edited.
[SerializeField]
Behaviour[] comptobedisabled;

[SerializeField]
int notshow = 10;

[SerializeField]
GameObject playerGraphics;

[SerializeField]
int remLayerName = 9;

[SerializeField]
GameObject playerUIPrefab;

[HideInInspector]
public GameObject playerUISingleton;

//Checking for local player authority and drawing all the elements associated
//with the local player.
void Start()
{
    if (!isLocalPlayer)
    {
        DisableComponents();
        AssignRemLayer();
    }
    else
    {

        SetLayerRecursively(playerGraphics, notshow);

        //Create Player UI
        playerUISingleton = Instantiate(playerUIPrefab);
        playerUISingleton.name = playerUIPrefab.name;

        //Settings
        UIManager ui = playerUISingleton.GetComponent<UIManager>();
        if (ui==null)
        {
            Debug.LogError("Could not get UI Component from prefab.");
        }
        ui.SetPlayer(GetComponent<Player>());

        GetComponent<Player>().SetupPlayer();

        string theusername = "Retrieving Player...";
        if(AccountManagement.ActiveUser == true)
        {
            theusername = AccountManagement.playerUsername;
        }else
        {
            theusername = transform.name;
        }

        CmdSetUname(transform.name,theusername);

    }


}
```

```
//Server spreads the information that player has successfully joined the game
[Command]
void CmdSetUname (string pID,string uname )
{
    Player player = GManager.FindPlayer(pID);
    if (player != null)
    {
        Debug.Log(uname + "joinded game.");
        player.uname = uname;
    }
}

//Recursive assignment to a different layer for some of the player's gameObjects
//(i.e weapon)
void SetLayerRecursively (GameObject obj , int nlayer)
{
    obj.layer = nlayer;

    foreach (Transform child in obj.transform)
    {
        SetLayerRecursively(child.gameObject, nlayer);
    }

}

//Registering the new client player on the game manager
//while saving his information.
public override void OnStartClient()
{
    base.OnStartClient();
    string thenetID = GetComponent<NetworkIdentity>().netId.ToString();
    Player theplayer = GetComponent<Player>();
    GManager.RegisterPlayer(thenetID, theplayer);
}



void AssignRemLayer()
{
    gameObject.layer = remLayerName;
}

void DisableComponents ()
{
    for (int i = 0; i < comptobedisabled.Length; i++)
    {
        comptobedisabled[i].enabled = false;

    }
}


//When Object is destroyed Re-Enable Process
void OnDisable()
{
    Destroy(playerUISingleton);


    if (isLocalPlayer)
    {
        GManager.singleton.SetSceneCam(true);
```

```
        }
        //UnRegister Players
        GManager.UnRegisterPlayer(transform.name);
    }
}
```

6.13 Player Script

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

//Main Game Logic Class that denotes actions that happen to the player.
//Player executes the following actions in the course of the game:
//1)Receiving Damage
//2)Player Death
//3)Respawn
//4)Enabling/Disabling other GameObjects

[RequireComponent(typeof(SettingPlayer))]
public class Player : NetworkBehaviour {

    [SerializeField]
    private int FullHealth = 100;

    [SyncVar]
    private int currentHealth;

    [SyncVar]
    private bool dead = false;

    // [SyncVar]
    public int killcount;
    public int deathcount;

    [SyncVar]
    public string uname = "Retrieving Player ...";

    //Player has 100 Health that is synced through the network

    public float GetHealth()
    {
        return (float)currentHealth / FullHealth;
    }

    //Getter-Setter indicating Player Death
    public bool Death
    {
        get { return dead; }

        protected set { dead = value; }
    }

    [SerializeField]
    private Behaviour[] disableDead;
    private bool[] previouslyEnabled;

    [SerializeField]
    private GameObject[] disableObjects;
```

```
[SerializeField]
private GameObject explosion;

[SerializeField]
private GameObject spawnGFX;

private bool FirstSetup=true;

//Executing first player initialization
public void SetupPlayer ()
{

    if (isLocalPlayer)
    {
        GManager.singleton.SetSceneCam(false);
        GetComponent<SettingPlayer>().playerUISingleton.SetActive(true);

    }


    CmdNewPlayer();
}

//Server commands clients to setup their player
[Command]
private void CmdNewPlayer()
{
    RpcSetupPlayer();
}

//Clients Initialize their player and enable their associated gameObjects
[ClientRpc]
private void RpcSetupPlayer()
{
    if (FirstSetup)
    {
        previouslyEnabled = new bool[disableDead.Length];
        for (int i = 0; i < previouslyEnabled.Length; i++)
        {
            previouslyEnabled[i] = disableDead[i].enabled;
        }

        FirstSetup = false;
    }



    Initialize();
}



//Locally Take Damage RPC Call to all clients
[ClientRpc]
public void RpcDealDmg (int howmuch,string enemyPlayer)
{
    if (Death)
    {
        return;
    }
```

```
        currentHealth = currentHealth - howmuch;

        Debug.Log(transform.name + " has " + currentHealth + " health. ");

        if (currentHealth<=0)
        {

            Die(enemyPlayer);
        }
    }

    //Event of player death and settling the score
    private void Die (string enemyPlayer)
    {
        Death = true;

        Player enemy = GManager.FindPlayer(enemyPlayer);

        if (enemy !=null)
        {

            enemy.killcount = enemy.killcount + 1;
            GManager.singleton.whenkilled.Invoke(uname, enemy.uname);
        }



        deathcount = deathcount + 1;

        //Remove player object properties
        for (int i = 0; i < disableDead.Length; i++)
        {
            disableDead[i].enabled = false;
        }

        //Disable Player GFX upon death
        for (int i = 0; i < disableObjects.Length; i++)
        {
            disableObjects[i].SetActive(false);
        }

        Collider col = GetComponent<Collider>();
        if (col != null)
        {
            col.enabled = false;
        }

        //Explosion upon death
        GameObject deathGFX = (GameObject)Instantiate(explosion, transform.position,
Quaternion.identity);
        Destroy(deathGFX,3f);

        //Camera upon death
        if (isLocalPlayer)
        {
            GManager.singleton.SetSceneCam(true);
            GetComponent<SettingPlayer>().playerUISingleton.SetActive(false);
        }

        Debug.Log(transform.name + " died !");

        //Respawn Player with Enum
```

```
        StartCoroutine(Respawn());
    }

    //Method that respawns the player on the specified spawn location
    //after a small spawn delay
    private IEnumerator Respawn ()
    {
        yield return new WaitForSeconds(GManager.singleton.matchSet.respawnDelay);

        Transform spawnpoint = NetworkManager.singleton.GetStartPosition();
        transform.position = spawnpoint.position;
        transform.rotation = spawnpoint.rotation;

        yield return new WaitForSeconds(0.1f);

        SetupPlayer();

        Debug.Log(transform.name + " respawned.");
    }

    //Initializing the player
    public void Initialize ()
    {
        Death = false;
        currentHealth = FullHealth;
        //Set Components
        for (int i = 0; i < disableDead.Length; i++)
        {
            disableDead[i].enabled = previouslyEnabled[i];
        }

        //Set Player GFX
        for (int i = 0; i < disableObjects.Length; i++)
        {
            disableObjects[i].SetActive(true);
        }

        Collider col = GetComponent<Collider>();
        if (col!=null)
        {
            col.enabled = true;
        }


        //Spawn Graphics
        GameObject       spawngraphics       =       (GameObject)Instantiate(spawnGFX,
transform.position, Quaternion.identity);
        Destroy(spawngraphics, 3f);

    }



}
```

6.14 Weapon Handling Script

```
using UnityEngine;
using UnityEngine.Networking;
```

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

//Blueprint class that spawns the weapon on the player
//and manages ammunition and reloading events.
//Class can be used as for future weapon additions.
public class WeaponHandling : NetworkBehaviour {

    [SerializeField]
    private int weapLayer = 11;

    [SerializeField]
    private Transform GunUse;

    [SerializeField]
    private Weapon Primary;

    private Weapon currWeap;
    private WeaponGFX currGFX;

    public bool reloading = false;

    //On Start spawn weapon on the player
    void Start()
    {
        Equip(Primary);
    }

    //Get weapon information and effects
    public Weapon GetWeapon ()
    {
        return currWeap;
    }

    public WeaponGFX GetWeaponGFX ()
    {
        return currGFX;
    }

    //Instantiate the prefabricated model to each player
    //Managing the drawing layer
    void Equip (Weapon myweapon)
    {
        currWeap = myweapon;

        GameObject      WeaponInstance      =      (GameObject)Instantiate(myweapon.gfx,
GunUse.position, GunUse.rotation);
        WeaponInstance.transform.SetParent(GunUse);

        currGFX = WeaponInstance.GetComponent<WeaponGFX>();
        if(currGFX==null)
        {
            Debug.LogError("Weapon Graphics not found : " + WeaponInstance.name);
        }

        if (isLocalPlayer)
        {
            Methods.SetLayerRecursively(WeaponInstance, weapLayer);
        }
    }

    //Reload Event
```

```
    public void Reload ()
    {
        if (reloading)
        {
            return;
        }
        StartCoroutine(ReloadDelay());
    }

    //Efficient reload delay
    private IEnumerator ReloadDelay ()
    {
        Debug.Log("Reloading Weapon");
        reloading = true;

        yield return new WaitForSeconds(currWeap.reloadSpeed);

        currWeap.currbullets = currWeap.magCapacity;

        reloading = false;
    }

}
```

6.15 Weapon Script

```
using UnityEngine;

//Class that models the weapon and its parameters
//for later use in the editor
[System.Serializable]
public class Weapon  {

    public string name = "Peacekeeper";

    //Damage the weapon deals per bullet
    public int dmg = 20;
    //Maximum distance the weapon can inflict damage to other players
    public float range = 100f;

    //shooting the weapon in sem-auto or full-auto
    public float fireRate = 0f;

    //maximum amount of bullets before reloading
    public int magCapacity = 20;

    //time needed to reload
    public float reloadSpeed = 1f;

    [HideInInspector]
    public int currbullets;

    //Weapon Model
    public GameObject gfx;

    //Weapon Constructor
    public Weapon()
    {
        currbullets = magCapacity;
```

```
    }

}
```

6.16 Weapon GFX Script

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

//Small class assigning the muzzle and the impact effects to the weapon
public class WeaponGFX : MonoBehaviour {

    //Effects are modified particle systems
    public ParticleSystem muzzleFlash;
    public GameObject ImpactGFX;
}
```

6.17 Shooting Script

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

//Game Logic class that models the events of shooting a weapon
//and manages graphics,and the consequences of shooting and reloading.

[RequireComponent(typeof(WeaponHandling))]
[RequireComponent(typeof(AudioSource))]
public class Shooting : NetworkBehaviour {

    private const string PLAYER_TAG = "Player";

    [SerializeField]
    private Camera cam;

    [SerializeField]
    private LayerMask mask;


    private WeaponHandling weaponManager;
    private Weapon currWeap;

    public AudioSource gunshot;

    //Looking for the fist person camera and getting the
    //associated information for the weapon.
    void Start()
    {
        if (cam == null)
        {
            Debug.LogError("Shooting : No camera referenced!");
            this.enabled = false;
        }
```

```
    weaponManager = GetComponent<WeaponHandling>();

    gunshot = GetComponent<AudioSource>();



}

//Player fires with left mouse button
//Player reloads with R button
void Update()
{
    currWeap = weaponManager.GetWeapon();

    if (currWeap.currbullets < currWeap.magCapacity)
    {
        if (Input.GetKeyDown(KeyCode.R))
        {
            weaponManager.Reload();
            return;
        }
    }

    if (PauseGame.freeze == true)
    {
        return;
    }

    if (currWeap.fireRate == 0f)
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Fire();


        }
    }
    else
    {
        if (Input.GetButtonDown("Fire1"))
        {
            InvokeRepeating("Fire", 0f, 1f / currWeap.fireRate);
        }
        else if (Input.GetButtonUp("Fire1"))
        {
            CancelInvoke("Fire");
        }
    }

}

//Server commands clients to show the muzzle effects
[Command]
void CmdFire()
{
    RpcMuzzleFlash();



}
```

```
    //Each clients displays the muzzle effects and gun sound.
    [ClientRpc]
    void RpcMuzzleFlash()
    {
        gunshot.Play();
        weaponManager.GetWeaponGFX().muzzleFlash.Play();
    }

    //Server commands clients to show the effects of bullet impact
    [Command]
    void CmdOnShot(Vector3 ShotPos, Vector3 Nsurface)
    {
        RpcImpactEffect(ShotPos, Nsurface);
    }

    //Each client displays the appropriate impact effect
    [ClientRpc]
    void RpcImpactEffect(Vector3 ShotPos, Vector3 Nsurface)
    {

        GameObject                                shotGFX                                =
(GameObject)Instantiate(weaponManager.GetWeaponGFX().ImpactGFX,                ShotPos,
Quaternion.LookRotation(Nsurface));
        Destroy(shotGFX, 2f);
    }

    //Client-side raycasting to determine shootout targets
    //through their colliders
    [Client]
    void Fire()
    {


        if (!isLocalPlayer || weaponManager.reloading)
        {
            return;
        }

        if(currWeap.currbullets <=0 )
        {
            //Debug.Log("OutOfBullets");
            weaponManager.Reload();

            return;
        }

        currWeap.currbullets--;

        Debug.Log("Remaining bullets: " + currWeap.currbullets);

        //Server determines what clients see when shooting
        CmdFire();

        RaycastHit hit;
        if            (Physics.Raycast(cam.transform.position,cam.transform.forward,out
hit,currWeap.range, mask))
        {

            if (hit.collider.tag == PLAYER_TAG)
            {
                CmdPlayerShot(hit.collider.name ,currWeap.dmg,transform.name);
```

```
            }
            //Server determines the effect that clients see when we hit something
            CmdOnShot(hit.point, hit.normal);

        }

        if (currWeap.currbullets <= 0)
        {
            weaponManager.Reload();
        }
    }

    //Server commands clients to display the event of damage dealt to the player being
shot
    [Command]
    void CmdPlayerShot (string playerID, int mydmg,string enemyPlayer)
    {
        Debug.Log(playerID + " was shot !");

        Player myplayer = GManager.FindPlayer(playerID);

        myplayer.RpcDealDmg(mydmg,enemyPlayer);

    }

}
```

6.18 UI Manager Script

```
using UnityEngine;
using UnityEngine.UI;

//Class responsible for displaying all of
//the UI elements associated with the player

public class UIManager : MonoBehaviour {

    //Serializing fields to associated media in the editor
    [SerializeField]
    RectTransform MeterFill;

    [SerializeField]
    Image MeterColor;

    [SerializeField]
    RectTransform HealthBar;

    [SerializeField]
    GameObject pause;

    [SerializeField]
    GameObject Scoreboard;

    [SerializeField]
    Text AmmoTxt;

    private WeaponHandling wManager;
    private Player player;
```

```
private PlayerControls controller;

//Get the components needed to connect to the UI
public void SetPlayer(Player myplayer)
{
    player = myplayer;
    controller = player.GetComponent<PlayerControls>();
    wManager = player.GetComponent<WeaponHandling>();
}

void Start()
{
    PauseGame.freeze = false;
    MeterColor.color = Color.green;

}

//F1 toggles the pause screen
//Tab shows the scoreboard and the connected players
void Update()
{


    SetFuel(controller.GetFuel());

    SetHealth(player.GetHealth());

    UpdateAmmo(wManager.GetWeapon().currbullets);

    if (Input.GetKeyDown(KeyCode.F1))
    {
        PauseToggle();
        //Debug.Log("Escape was pressed");
    }

    if (Input.GetKeyDown(KeyCode.Tab))
    {
        Scoreboard.SetActive(true);
    }
    else if (Input.GetKeyUp(KeyCode.Tab))
    {
        Scoreboard.SetActive(false);
    }

}

//Toggling the Pause gameObjects
public void PauseToggle()
{
    pause.SetActive(!pause.activeSelf);
    PauseGame.freeze = pause.activeSelf;
}

//Linear interpolation to recolor the fuel meter
void SetFuel (float amount)
{
    MeterFill.localScale = new Vector3(1f, amount, 1f);
    MeterColor.color = new Color(1f - amount,amount, 0f);
}

//Resizing the health meter
void SetHealth(float health)
```

```
    {
        HealthBar.localScale = new Vector3(1f, health, 1f);
    }

    //Updating the ammount of bullets remaining.
    void UpdateAmmo (int ammo)
    {
        AmmoTxt.text = ammo.ToString();
    }

}
```

6.19 Nameplate Manager Script

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

//Class responsible for displaying the Nameplate
//UI element for each player.The following items are displayed:
//1)Player Username
//2)Player Health
public class NameplateManager : MonoBehaviour {

    [SerializeField]
    private Text UserTxt;

    [SerializeField]
    private Player myplayer;

    [SerializeField]
    private RectTransform Healthbar;

       // Player Health is also updated on the nameplate to make it visible to other
players
    // The position of the nameplate adjusts according to where the player is looking.
       void Update () {
        Camera cam = Camera.main;

        if (cam != null)
        {
            UserTxt.text = myplayer.uname;

            Healthbar.localScale = new Vector3(myplayer.GetHealth(), 1f, 1f);

            transform.LookAt(transform.position     +     cam.transform.rotation     *
Vector3.forward,
                cam.transform.rotation * Vector3.up);
        }
    }
}
```

6.20 Game Leaderboards Script

```
using UnityEngine;
using System.Collections;

//This class displays the score of each player
```

```
//during the game on their UI.
public class GameLeaderboards : MonoBehaviour {

    //Displays the prefab of the in-game leaderboards
    [SerializeField]
    GameObject ScoreItem;

    //Denotes the position of the score items
    [SerializeField]
    Transform pList;


       void OnEnable ()
    {
        //Retrieve player list
        Player[] players = GManager.FindAllPlayers();

        foreach (Player player in players)
        {
            //Instantiate the prefabs for every active player in the match
            GameObject scoreObj = (GameObject) Instantiate(ScoreItem, pList);

            //ScoreItem Follows List Transform
            scoreObj.transform.localScale = pList.localScale;

            LeaderboardItem item =scoreObj.GetComponent<LeaderboardItem>();

            if (item != null)
            {
                //initiate the setup with the player username , the amount of kills
and deaths in the prefab
                item.ScoreSetup(player.uname, player.killcount, player.deathcount);
            }



        }

    }

    void OnDisable ()
    {
        foreach(Transform child in pList)
        {
            //Each prefab is placed in a list formation, once a player leaves the
prefab is destroyed
            Destroy(child.gameObject);
        }
    }
}
```

6.21 Leaderboard Item Script

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

//Class responsible for setting up the in-game leaderboard
```

```
//prefab item with the current score.
public class LeaderboardItem : MonoBehaviour {

    [SerializeField]
    Text Username_Txt;

    [SerializeField]
    Text KillCount_Txt;

    [SerializeField]
    Text DeathCount_Txt;

    //Update textboxes to reflect current score
    public void ScoreSetup(string username , int killcount , int deathcount)
    {
        Username_Txt.text = username;
        KillCount_Txt.text = "Killcount: " + killcount;
        DeathCount_Txt.text = "Deathcount: " + deathcount;


    }
}
```

6.22 KillFeed Manager Script

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

//This class handles the functionality of the Killfeed UI element
//to display the death of one player by another
public class KillFeedManager : MonoBehaviour {

    [SerializeField]
    GameObject FeedDisplay;

       // Event initialized on the game manager
    // showing that another player died.
       void Start () {
        GManager.singleton.whenkilled += Killed;
       }

    //Method responsible of creating an instance of the killfeed prefab item
    //given the names of the two players that were fighting.
    //The feed disappears after 5 seconds to avoid cluttering the UI.
       public void Killed (string player , string enemy)
    {
        GameObject feed = (GameObject)Instantiate(FeedDisplay, this.transform);

        //Feed follows localscale
        feed.transform.localScale = this.transform.localScale;

        feed.GetComponent<KillfeedPrefabItem>().SetupFeed(player, enemy);

        Destroy(feed,5f);

    }
}
```

6.23 Killfeed Prefab Script

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

//Class connected to the prefabricated Killfeed item
//responsible of placing it properly on the player UI.
public class KillfeedPrefabItem : MonoBehaviour {

    //Serializable Textbox
    [SerializeField]
    Text KillFeedDisplay;

    //Feed setup is done with HTML tags to customize the text.
    public void SetupFeed (string player ,string enemy )
    {
        KillFeedDisplay.text = "<color=#00ff00ff>" + enemy + "</color>" + " Killed " +
"<color=#ffff00ff>" + player + "</color>";
    }
}
```

6.24 Game Manager Script

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

//This class is responsible for the following functions
//1)Game and Level Camera Setup
//2)Player Detection
//3)Player Registration
public class GManager : MonoBehaviour {

    //The running instance of the game manager
    public static GManager singleton;

    //reference to a Settings object
    public Settings matchSet;

    //main level camera
    [SerializeField]
    private GameObject sceneCam;

    //Using delegate to pass an entire method as an arguement to other methods.
    public delegate void WhenKilled(string player, string enemy);

    public WhenKilled whenkilled;

    //Error checking of the running game manager instance when scene loads.
    void Awake()
    {
        if (singleton != null)
        {
            Debug.LogError("Another Instance of Game Manager has been already set!");
        }else
        {
            singleton = this;
```

```
        }

    }

    //Enabling the main camera of the current level
    public void  SetSceneCam(bool enabler)
    {
        if (sceneCam == null)
        {
            return;
        }

        sceneCam.SetActive(enabler);
    }

    #region Player Registration
    //Assign each player to a unique categorization ID and
    //create a dictionary of all the Player objects with their network ID

    private const string PLAYER_PRESET_ID = "Player ";

    private  static  Dictionary<string,  Player>  players  =  new  Dictionary<string,
Player>();

    //Each time a player joins the game , register that player to the dictionary
    public static void RegisterPlayer (string mynetID,Player myplayer)
    {

        string myplayerID = PLAYER_PRESET_ID + mynetID;
        players.Add(myplayerID, myplayer);

        myplayer.transform.name = myplayerID;
    }

    //When a player leaves the game , he is removed from the dictionary
    public static void UnRegisterPlayer (string myplayerID)
    {
        players.Remove(myplayerID);
    }

    //Detect a player in the dictionary
    public static Player FindPlayer (string myplayerID)
    {
        return players[myplayerID];
    }

    //Return all the information retrieved from the players
    public static Player[] FindAllPlayers()
    {
        return players.Values.ToArray();
    }



    #endregion


}
```

6.25 Pause Game Script

```
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Networking.Match;

//Class responsible for stoppoing and pausing a network game
public class PauseGame : MonoBehaviour {

    //Boolean parameter indicating that the game is paused
    public static bool freeze = false;

    private NetworkManager nManager;

    void Start()
    {
        nManager = NetworkManager.singleton;
    }

    //Method responsible for disconnecting players that wish to leave the game
    //If Host leaves the game the lobby is disbanded.
    public void LeaveLobby()
    {
        MatchInfo matchParameters = nManager.matchInfo;
        nManager.matchMaker.DropConnection(matchParameters.networkId,
matchParameters.nodeId, 0, nManager.OnDropConnection);
        nManager.StopHost();
    }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1]"Multiplayer video game", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Multiplayer_video_game. [Accessed: 30- Mar- 2017].

[2]"Byte Magazine Volume 05 Number 12 - Adventure", Archive.org, 2017. [Online]. Available: https://archive.org/stream/byte-magazine-1980-12/1980_12_BYTE_05-12_Adventure#page/n309/mode/2up. [Accessed: 30- Mar- 2017].

[3]"'Battlefield 1' Is Good, but 'Overwatch' Is Still the Only Online Shooter I Need", Vice, 2017. [Online]. Available: https://www.vice.com/en_us/article/battlefield-1-is-good-but-overwatch-is-still-the-only-online-shooter-i-need-100-blizzard-ea-dice. [Accessed: 30- Mar- 2017].

[4]"First-person shooter", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/First-person_shooter. [Accessed: 30- Mar- 2017].

[5]"Artificial Life and Puzzle Games", Wps.prenhall.com, 2017. [Online]. Available: http://wps.prenhall.com/bp_gamedev_1/54/14053/3597646.cw/index.html. [Accessed: 30- Mar- 2017].

[6]"The Art Of FPS Multiplayer Design", Web.archive.org, 2017. [Online]. Available: https://web.archive.org/web/20080525134030/http://www.gameinformer.com/News/Story/200803/N08.0305.1634.21173.htm?Page=2. [Accessed: 30- Mar- 2017].

[7]"Unity - Fast Facts", Unity, 2017. [Online]. Available: https://unity3d.com/public-relations. [Accessed: 08- Apr- 2017].

[8]"Unity - Unity - Editor", Unity, 2017. [Online]. Available: https://unity3d.com/unity/editor. [Accessed: 08- Apr- 2017].

[9]U. Technologies, "Unity - Manual: The Toolbar", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/Toolbar.html. [Accessed: 08- Apr- 2017].

[10]"Unity - The Scene View", Unity, 2017. [Online]. Available: https://unity3d.com/learn/tutorials/topics/interface-essentials/scene-view.[Accessed: 08- Apr- 2017].

[11]U. Technologies, "Unity - Manual: Inspector", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/462/Documentation/Manual/Inspector.html. [Accessed: 08- Apr- 2017].

[12]U. Technologies, "Unity - Manual: The Game view", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/GameView.html. [Accessed: 08- Apr- 2017].

[13]U. Technologies, "Unity - Manual: The Hierarchy window", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/Hierarchy.html. [Accessed: 08- Apr- 2017].

[14]U. Technologies, "Unity - Manual: The Project Window", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/ProjectView.html. [Accessed: 08- Apr- 2017].

[15]U. Technologies, "Unity - Manual: Build Settings", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/BuildSettings.html. [Accessed: 08- Apr- 2017].

[16]U. Technologies, "Unity - Scripting API: GameObject", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/ScriptReference/GameObject.html. [Accessed: 08- Apr- 2017].

[17]U. Technologies, "Unity - Scripting API: Component", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/ScriptReference/Component.html. [Accessed: 08- Apr- 2017].

[18]U. Technologies, "Unity - Manual: Importing Assets", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/ImportingAssets.html. [Accessed: 08- Apr- 2017].

[19]"Unity - Prefabs - Concept & Usage", Unity, 2017. [Online]. Available: https://unity3d.com/learn/tutorials/topics/interface-essentials/prefabs-concept-usage. [Accessed: 08- Apr- 2017].

[20]U. Technologies, "Unity - Manual: Layers", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/Layers.html. [Accessed: 08- Apr- 2017].

[21]U. Technologies, "Unity - Manual: Tags", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/Tags.html. [Accessed: 08- Apr- 2017].

[22]U. Technologies, "Unity - Scripting API: SceneManager", Docs.unity3d.com, 2017. [Online].Available:https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html. [Accessed: 08- Apr- 2017].

[23] U. Technologies, "Unity - Manual: Execution Order of Event Functions", Docs.unity3d.com,2017.[Online].Available:https://docs.unity3d.com/Manual/ExecutionOrder.html. [Accessed: 08- Apr- 2017].

[24]"Asset Store", Assetstore.unity3d.com, 2017. [Online]. Available: https://www.assetstore.unity3d.com/en/. [Accessed: 08- Apr- 2017].

[25]U. Technologies, "Unity - Manual: Multiplayer and Networking", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/UNet.html. [Accessed: 08- Apr- 2017].

[26]U. Technologies, "Unity - Manual: The High Level API", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/UNetUsingHLAPI.html. [Accessed: 08- Apr- 2017].

[27]U. Technologies, "Unity - Scripting API: NetworkManager", Docs.unity3d.com, 2017. [Online].Available:https://docs.unity3d.com/ScriptReference/Networking.NetworkManager.html. [Accessed: 08- Apr- 2017].

[28]U. Technologies, "Unity - Manual: Network Lobby Manager", Docs.unity3d.com, 2017. [Online].Available:https://docs.unity3d.com/Manual/class-NetworkLobbyManager.html. [Accessed: 08- Apr- 2017].

[29]U. Technologies, "Unity - Manual: NetworkIdentity", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/class-NetworkIdentity.html. [Accessed: 08- Apr- 2017].

[30]U. Technologies, "Unity - Manual: Network System Concepts", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/UNetConcepts.html. [Accessed: 08- Apr- 2017].

[31]U. Technologies, "Unity - Manual: State Synchronization", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/UNetStateSync.html. [Accessed: 08- Apr- 2017].

[32]U. Technologies, "Unity - Manual: Remote Actions", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/UNetActions.html. [Accessed: 08- Apr- 2017].

[33]"Unity - Services - Multiplayer", Unity, 2017. [Online]. Available: https://unity3d.com/services/multiplayer. [Accessed: 08- Apr- 2017].

[34]G. 2015, "U.S. most popular video game genres 2015 | Statista", Statista, 2017. [Online]. Available: https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre/. [Accessed: 08- Apr- 2017].

[35]"PlanetSide 2 - Home", PlanetSide 2, 2017. [Online]. Available: https://www.planetside2.com/home. [Accessed: 08- Apr- 2017].

[36]"Destiny the Game | Home", Destinythegame.com, 2017. [Online]. Available: https://www.destinythegame.com/. [Accessed: 08- Apr- 2017].

[37]"Nexuiz", Nexuiz.com, 2017. [Online]. Available: http://www.nexuiz.com/. [Accessed: 08- Apr- 2017].

[38]"Counter-Strike on Steam", Store.steampowered.com, 2017. [Online]. Available: http://store.steampowered.com/app/10/. [Accessed: 08- Apr- 2017].

[39]"Database Control", Solution Studios, 2017. [Online]. Available: http://unitysolutionstudios.weebly.com/database-control.html. [Accessed: 08- Apr- 2017].

[40]U. Technologies, "Unity - Manual: Setting up Unity Multiplayer", Docs.unity3d.com, 2017. [Online]. Available: https://docs.unity3d.com/Manual/UnityMultiplayerSettingUp.html. [Accessed: 08- Apr- 2017].

[41]"Robot Jasubot-PRO01 - 3d model - .3ds, .obj, .c4d, .mb, .fbx", Tf3dm.com, 2017. [Online]. Available: http://tf3dm.com/3d-model/jasubot-pro01-20918.html. [Accessed: 08- Apr- 2017].

[42]"Weapons 3D Models - Free 3D Weapons download", Tf3dm.com, 2017. [Online]. Available: http://tf3dm.com/3d-models/weapons. [Accessed: 08- Apr- 2017].

[43]"Unity - Merry Fragmas: Multiplayer FPS, Part 1", Unity, 2017. [Online]. Available: https://unity3d.com/learn/tutorials/topics/asset-store/merry-fragmas-multiplayer-fps-part-1. [Accessed: 08- Apr- 2017].

[44]"Unity - Merry Fragmas 2.0 - Multiplayer FPS", Unity, 2017. [Online]. Available: https://unity3d.com/learn/tutorials/modules/intermediate/live-training-archive/fragmas-2-multiplayer-fps. [Accessed: 08- Apr- 2017].

[45]"noobtuts - Unity First Person Shooter (FPS) Game", Noobtuts.com, 2017. [Online]. Available: https://noobtuts.com/unity/first-person-shooter-game. [Accessed: 08- Apr- 2017].