ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση fighting παιχνιδιού σε Unity με έμφαση 2D Pixel Art Graphics και Animations

---

Ονοματεπώνυμο Φοιτητή

Αλεξανδρόπουλος Αποστόλης

Επιβλέπων καθηγητής: Σωτήρης Χριστοδούλου

Περιεχόμενα

Ευχαριστίες

Ως ένδειξη ευγνωμοσύνης για την βοήθεια του, θα ήθελα να ευχαριστήσω τον κο Σωτήρη Χριστοδούλου που με εμπιστεύθηκε στην δημιουργία τους θέματος της πτυχιακής εργασίας. Θα ήθελα να ευχαριστήσω την οικογένεια μου για την ψυχολογική υποστήριξη που μου παρείχε.

Περίληψη

Η παρούσα πτυχιακή εργασία ασχολείται με την ανάπτυξη 2D Fighting παιχνιδιού με τέχνη και Animation δημιουργημένα από Pixel sprites, χρησιμοποιώντας την μηχανή Unity και ως γλώσσα προγραμματισμού την C#. Σκοπός της εργασίας είναι η δημιουργία ενός fighting παιχνιδιού με Pixel sprites, ο διαδικτυακός προγραμματισμός και η παρουσίαση της μηχανής “Unity” καθώς και η δημιουργία του 2D παιχνιδιού από το μηδέν. Γίνεται αναφορά για το περιβάλλον της, τις δυνατότητες και τις αδυναμίες της, καθώς και την βοήθεια που παρέχει. Τέλος παρουσιάζονται επιπλέον εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη του παιχνιδιού.

Λέξεις κλειδιά: 2D, Unity, Fighting, Pixel, Sprites

Abstract

The aim of this thesis was to develop a 2D Fighting game with art and animations created with the help of pixel sprites, using the Unity engine and C# as the programming language. The purpose of this work is to create a fighting game with pixel sprites, online programming and to make a presentation of the “Unity” game engine as well to create this 2D game from scratch. Reference is made to its environment, its strengths, and weaknesses, as well as the assistance it provides. Finally, additional tools for the development of the game are presented.

Keywords: 2D, Unity, Fighting, Pixel, Sprites

1 Εισαγωγή

Όσον αφορά τα βιντεοπαιχνίδια έχει γίνει αλματώδης εξέλιξη τα τελευταία χρόνια τόσο στο κομμάτι του Animation, όσο και στο κομμάτι των γραφικών και του χειρισμού.

Συγκεκριμένα ένας παράγοντας που καθιστούσε την ανάπτυξή των πρώτων βιντεοπαιχνιδιών ιδιαίτερα δύσκολη είναι ότι για τη δημιουργία τους γινόταν χρήση τέχνης και Animation σε Pixel, δηλαδή τα μοντέλα και το περιβάλλον τους ήταν ζωγραφισμένα σε κουτάκια. Βέβαια η μεγαλύτερη δυσχέρεια στην δημιουργία τους ήταν ότι εκείνες τις εποχές δεν υπήρχαν διαθέσιμες πολλές μηχανές ανάπτυξης βιντεοπαιχνιδιών.

Σήμερα χάρη στην πρόοδο της τεχνολογίας και την ύπαρξη πληθώρας μηχανών ανάπτυξης η δημιουργία των βιντεοπαιχνιδιών καθίσταται ευκολότερη τόσο στο κομμάτι της τέχνης με Pixel ή models, όσο και στο κομμάτι του προγραμματισμού. Αυτό έχει ως αποτέλεσμα να δίνεται ανεμπόδιστα η δυνατότητα δημιουργίας ενός βιντεοπαιχνιδιού από το μηδέν.

1.1 Unity game engine

Το Unity game engine είναι ένα εργαλείο παραγωγής παιχνιδιών που κυκλοφόρησε το 2005 και είχε ως σκοπό η πρόσβαση σε αυτό να είναι εφικτή από όλοι την κοινότητα των προγραμματιστών, ώστε να γίνει δυνατό το «άνοιγμα» της αγοράς του game development. Το Unity υποστηρίζει την δημιουργία τόσο 2D όσο και 3D βιντεοπαιχνιδιών.

Η κυκλοφορία του Unity έγινε αρχικά για MAC OS X και αργότερα προστέθηκε η δυνατότητα υποστήριξης και από άλλες πλατφόρμες όπως για παράδειγμα Windows, Linux, Android, iOS. Το 2006 το εν λόγω εργαλείο αναδείχθηκε δεύτερο στην κατηγορία Best Use of Mac OS X Graphics στα Apple Design Awards της Apple Inc.

Το 2007 προστέθηκε στο Unity πληθώρα δυνατοτήτων για την δημιουργία βιντεοπαιχνιδιών όπως δυναμικές σκιές σε πραγματικό χρόνο, κατεύθυνση του φωτός, βελτιστοποιήσεις στο περιβάλλον ενός 3D βιντεοπαιχνιδιού κ.α. Αργότερα κατέστη σε αυτό δυνατή η δημιουργία βιντεοπαιχνιδιών για πολλούς παίκτες καθώς επίσης εισήχθη η λειτουργία drag and drop, δηλαδή η δυνατότητα δημιουργίας ενός βιντεοπαιχνιδιού με κομμάτια έτοιμου κώδικα.

Τέλος από το 2018 μέχρι και σήμερα το Unity έχει χρησιμοποιηθεί για την δημιουργία περίπου των μισών βιντεοπαιχνιδιών για κινητά που κυκλοφόρησαν στην αγορά καθώς και για την δημιουργία πάνω απ’ των μισών βιντεοπαιχνιδιών επαυξημένης πραγματικότητας και εικονικής πραγματικότητας. Παραδείγματα βιντεοπαιχνιδιών που δημιουργήθηκαν με το Unity είναι το Cuphead, Hearthstone, Pokémon GO, RUST και άλλα πολλά.

Εικόνα 1: Το Logo του unity

1.2 Επιλογή και είδη Τέχνης

Υπάρχει πλήθος επιλογών όσον αφορά το είδος της τέχνης που μπορεί να επιλεχθεί για τον σχεδιασμό ενός βιντεοπαιχνιδιού. Στην παρούσα εργασία, ανάμεσα στις διαθέσιμες επιλογές Pixel,Vector και Raster που υποστηρίζει ένα 2D βιντεοπαιχνίδι, προτιμήθηκε η τέχνη Pixel. Η προαναφερθείσα επιλογή έγινε κυρίως λόγω του retro χαρακτήρα τον οποίον ο υποφαινόμενος ήθελε να δώσει στο παιχνίδι, καθώς η επιδίωξή του ήταν η ανάπτυξη ενός Retro Emulation βιντεοπαιχνιδιού. Ο όρος Retro Emulation χρησιμοποιείται για σύγχρονα παιχνίδια που έχουν, όμως, σχεδιαστικά χαρακτηριστικά παρόμοια με εκείνα των παλαιότερων γενεών, κάποια από τα οποία έχουν αφήσει εποχή στην ιστορία των βιντεοπαιχνιδιών.

Παρακάτω θα αναλυθεί η κάθε τέχνη ξεχωριστά καθώς και τα πλεονεκτήματα και μειονεκτήματα αυτής.

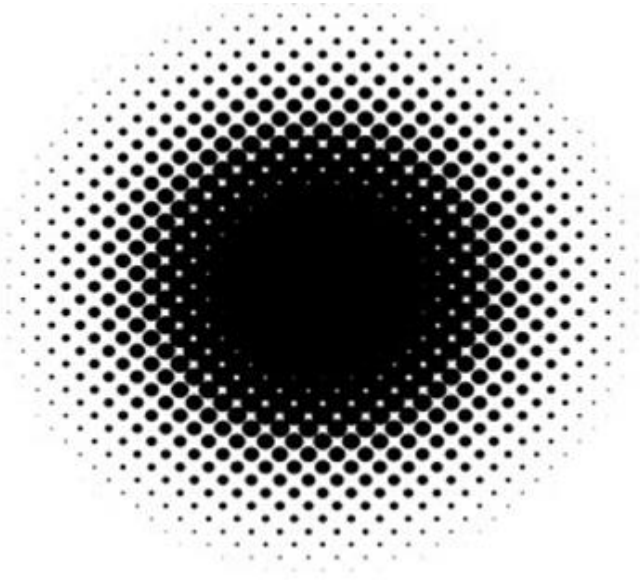
1.2.1 Vector τέχνη

Η Vector τέχνη ή αλλιώς η διανυσματική τέχνη, αποθηκεύτε ως μια σειρά γραμμών και σχημάτων. Ας δούμε αναλυτικά με ένα παράδειγμα τον τρόπο λειτουργίας της συγκεκριμένης τέχνης με την βοήθεια ενός προγράμματος επεξεργασίας εικόνας: Ο σχεδιασμός ενός κύκλου σε διανυσματική μορφή δεν απαιτεί την παρακολουθήσει κάθε pixel που σχηματίζει τον κύκλο. Αντίθετα στην διανυσματική τέχνη αναγνωρίζεται μόνο που βρίσκεται το κέντρο του κύκλου, πόσο μεγάλη είναι η ακτίνα του και ποιο είναι το χρώμα του. Κατά αυτόν τον τρόπο σε ένα βιντεοπαιχνίδι που διαβάζει διανυσματική τέχνη για την δημιουργία κινουμένων γραφικών αρκούν οι πληροφορίες σχετικά με τα διανυσματικά χαρακτηριστικά καθώς και το χρώμα των σχημάτων τα οποία μετατρέπονται σε εικονοστοιχεία εν κινήσει.

Εικόνα 2: Κύκλος με διανυσματική τέχνη

1.2.2 Raster τέχνη

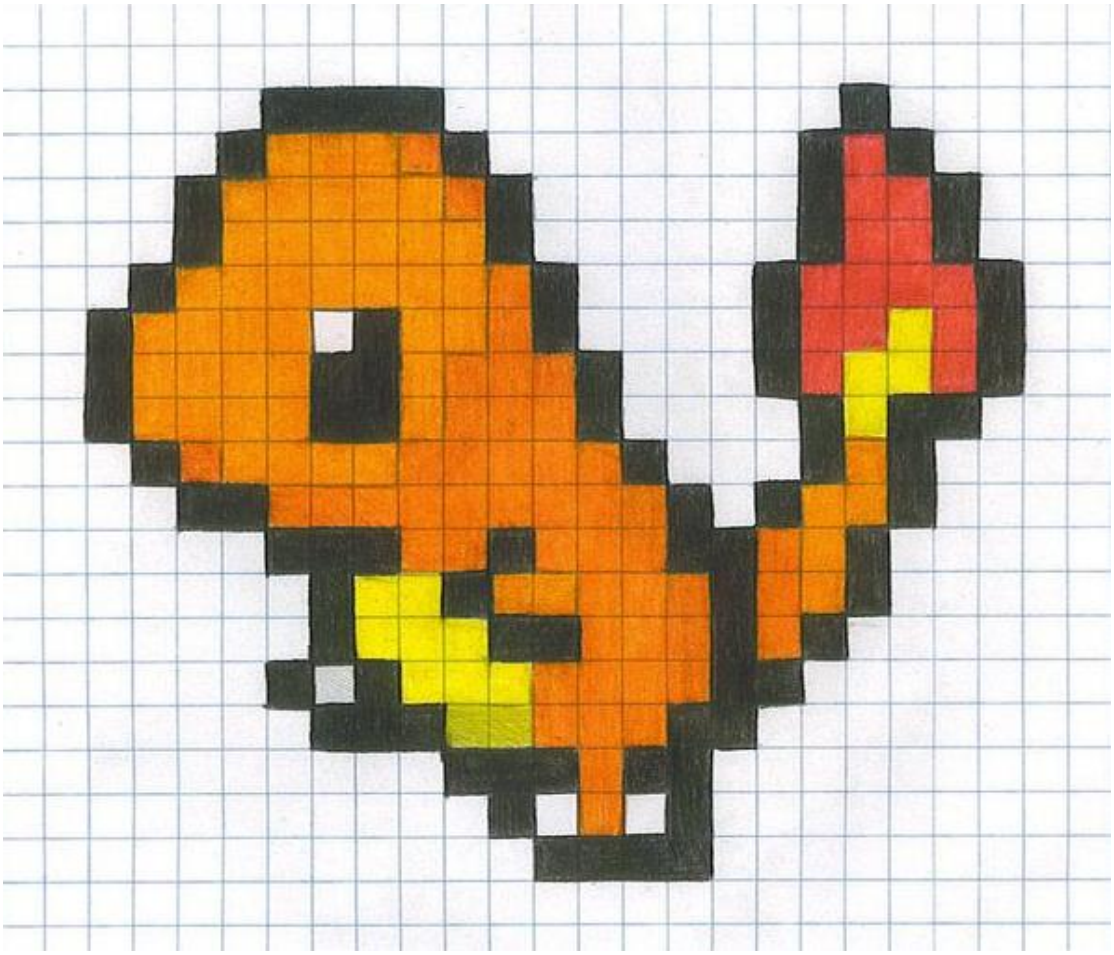
Η Raster τέχνη αποθηκεύτε ως μια σειρά τιμών και χρωμάτων σε ένα πλέγμα, δηλαδή ως μια συλλογή από pixel. Για παράδειγμα σε ένα πρόγραμμα επεξεργασίας εικόνας η επιλογή τέχνης Raster για την δημιουργία ενός κύκλου θα είχε το εξής αποτέλεσμα: Η εικόνα του κύκλου θα αποθηκεύονταν με τέτοιον τρόπο ώστε να φαίνεται κάθε μεμονωμένο pixel του σχήματός του κύκλου.



Εικόνα 3: Κύκλος με Raster τέχνη

1.2.3 Pixel τέχνη

Η Pixel τέχνη είναι ένα υποσύνολο της τέχνης raster. Σε αυτήν την τέχνη δημιουργούνται πολύ μικρές σε ανάλυση εικόνες με συνέπεια για την αποθήκευσή τους να χρησιμοποιούνται ελάχιστα pixel. Πολλοί άνθρωποι γνωρίζουν την pixel τέχνη από παλιά βιντεοπαιχνίδια στα οποία ήταν απαραίτητη η χρήση μικροσκοπικών σε ανάλυση εικόνων λόγω των ιδιαίτερα περιορισμένων δυνατοτήτων των παλαιών παιχνιδομηχανών.



Εικόνα 4: Χαρακτήρας βιντεοπαιχνιδιού με pixel τέχνη

1.3 Τα πλεονεκτήματα των ειδών τέχνης

Πλεονεκτήματα:

Raster τέχνη:

- Η raster τέχνη δίνει μεγάλη ελευθέρια. Συγκεκριμένα με την εν λόγω τέχνη μπορούμε να σχεδιάσουμε σχεδόν ότι εικόνα επιθυμούμε χωρίς να περιοριζόμαστε σε στυλ χειρός.
- Το σχέδιο τέχνης raster είναι φυσικό, δηλαδή για το σχεδιασμό της χρησιμοποιούνται οι ιδίες τεχνικές με τα σχέδια με μολυβί καθώς και στυλό ή μελάνι. Αυτό έχει ως αποτέλεσμα οι καλλιτέχνες να είναι ιδιαίτερα εξοικειωμένοι με το σχεδιασμό της τέχνης raster.

Vector τέχνη:

- Η vector τέχνη καθιστά εύκολη την επίτευξη μιας καθαρής εμφάνισης, η οποία για να πραγματοποιηθεί με το χέρι θα απαιτούσε πολύ μεγαλύτερη προσπάθεια.
- Τα αρχεία τα οποία είναι αποθηκευμένα σε vector τέχνη είναι ιδιαίτερα μικρά σε μέγεθος, ως εκ τούτου δεν χρειάζεται να ανησυχούμε για το χώρο του σκληρού μας δίσκου ή την μεταφορά των αποθηκευμένων αρχείων σε αλλά άτομα.
- Η vector τέχνη μπορεί να κλιμακωθεί απείρως.

Pixel τέχνη:

- Η ανάλυση και ερμηνεία της pixel τέχνης είναι εύκολο να επιτευχθεί μετά την δημιουργία της. Συγκεκριμένα σε ένα βιντεοπαιχνίδι αυτή η μινιμαλιστική εκδοχή της εικόνας βοηθά τους παίκτες στο καλύτερο χειρισμό. Το παραπάνω συμβαίνει καθώς οι αλλαγές στο γραφικό περιβάλλον του παιχνιδιού γίνονται ευκολότερα διακριτές σε σχέση με τις ενέργειες χειρισμού που πρέπει να γίνουν από τον παίκτη. Με άλλα λόγια η pixel τέχνη είναι μια αρκετά ικανοποιητική λύση για τα βιντεοπαιχνίδια, στα οποία οι παίκτες καλούνται να ερμηνεύσουν κινούμενες εικόνες πολύ γρηγορά και συχνά.
- Η pixel τέχνη καταλαμβάνει ελάχιστο χώρο, γεγονός που την καθιστά ιδανική για την ανάπτυξη βιντεοπαιχνιδιών σε κινητές συσκευές.

1.4 Τα μειονεκτήματα των ειδών τέχνης

Μειονεκτήματα:

Raster τέχνη:

- Η raster τέχνη καταλαμβάνει ιδιαίτερα αυξημένο χώρο. Αυτό είναι σπάνια ζήτημα από την πλευρά της παραγωγής σήμερα, δεδομένης της τεράστιας χωρητικότητας των σημερινών σκληρών δίσκων. Όμως, από την πλευρά του χρήστη αυτό μπορεί να είναι ένα τεράστιο πρόβλημα. Παραδείγματος χάριν σε ένα βιντεοπαιχνίδι για κινητά ή για σε ένα πρόγραμμα περιήγησης, όλες εκείνες οι εικόνες θα πρέπει να χωρέσουν σε μια ιδιαίτερα μικρή ποσότητα μνήμης, η οποία είναι εύκολο να εξαντληθεί
- Η raster τέχνη συνήθως δεν μας επιτρέπει να χρησιμοποιήσουμε στα βιντεοπαιχνίδια ένα απλό στυλ, διότι η τέχνη αυτή είναι πολύπλοκη και λεπτομερής.
- Η raster τέχνη δεν μπορεί να κλιμακωθεί πέρα της αρχικής της ανάλυσης την οποία έχουμε ορίσει.

Vector τέχνη:

- Η vector τέχνη δεν συνιστάται για την ανάπτυξη μικρών λεπτομερειών
- Επίσης μπορεί να γίνει ιδιαίτερα περιπλοκή και χαοτική.

Pixel τέχνη:

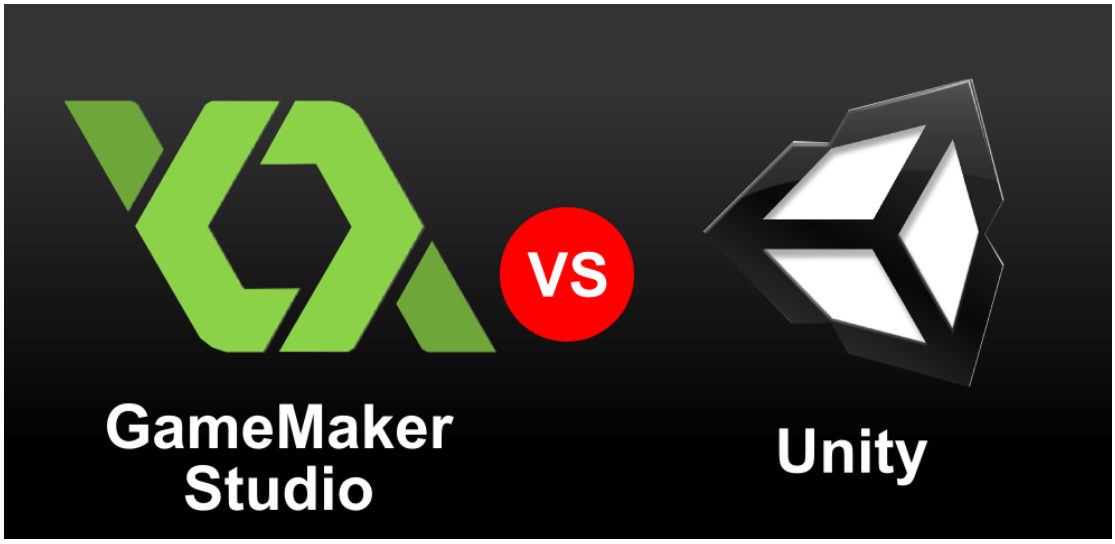
- Στην pixel τέχνη μερικές φορές τα εικονοστοιχεία μπορεί να είναι πιο περιοριστικά από ότι επιθυμούμε. Αυτό καθιστά εξαιρετικά δύσκολη την ενσωματώσει λεπτομερειών, όπως για παράδειγμα τις έκφρασης του προσώπου ενός χαρακτήρα
- Η pixel τέχνη δεν είναι ευέλικτη καθώς για να αλλάξει επί παραδείγματι το χρώμα ενός χαρακτήρα απαιτείται η αλλαγή χρώματος σε κάθε κινούμενο σχέδιο του χαρακτήρα.

1.5 Επιλογή Unity

Στον χώρο της ανάπτυξης βιντεοπαιχνιδιών υπάρχει πληθώρα επιλογών σήμερα, όπως για παράδειγμα το Godot, GameMaker Studio, Stencyl, Construct2, Unreal Engine και άλλα πολλά. Ο σκοπός της παρούσας εργασίας ήταν η δημιουργία ενός βιντεοπαιχνιδιού δύο διαστάσεων και για αυτόν τον σκοπό οι καταλληλότερες μηχανές ανάπτυξης θεωρήσαμε ότι είναι το GameMaker Studio και το Unity. Το GameMaker Studio ειδικεύεται σε βιντεοπαιχνίδια δύο διαστάσεων, ενώ στο Unity χρησιμοποιούνται γλώσσες προγραμματισμού όπως η C# με την οποία υπήρχε μεγαλύτερη εξοικείωση του υποφαινόμενου. Αντίθετα οι υπόλοιπες μηχανές ανάπτυξης προορίζονται είτε για αρχάριους προγραμματιστές, έχοντας επιλογές τύπου σκρατς, οι οποίες δεν καλύπτουν της ανάγκες της παρούσας εργασίας, είτε δεν υπάρχει για αυτές αρκετό διαθέσιμο υλικό εκμάθησης όπως για παράδειγμα forum, tutorials κ.λπ.

Μετά από την καταμέτρηση των μειονεκτημάτων και των πλεονεκτημάτων που έχουν αυτές οι δύο μηχανές επιλέχθηκε το Unity ως η καταλληλότερη μηχανή για την ανάπτυξη της εν λόγω εργασίας καθώς :

- Υποστηρίζει 25 πλατφόρμες και μέσα σε αυτές είναι Xbox One, Nintendo Switch, PlayStation 4, PlayStation Vita, Windows, Linux, Mac OS, Android και άλλες.
- Η γλώσσα προγραμματισμού που χρησιμοποιεί έχει εξαιρετικά πολλές δυνατότητες.
- Υποστηρίζει το drag and drop στοιχείων με πολύ καλή οργάνωση
- Έχει store στο οποίο μπορεί εύκολα κάποιος να πάρει έτοιμα και δωρεάν templates για την δημιουργία του βιντεοπαιχνιδιού του.
- Έχει τεράστια κοινότητα και ιδιαίτερα υποστηρικτική.
- Είναι δωρεάν.



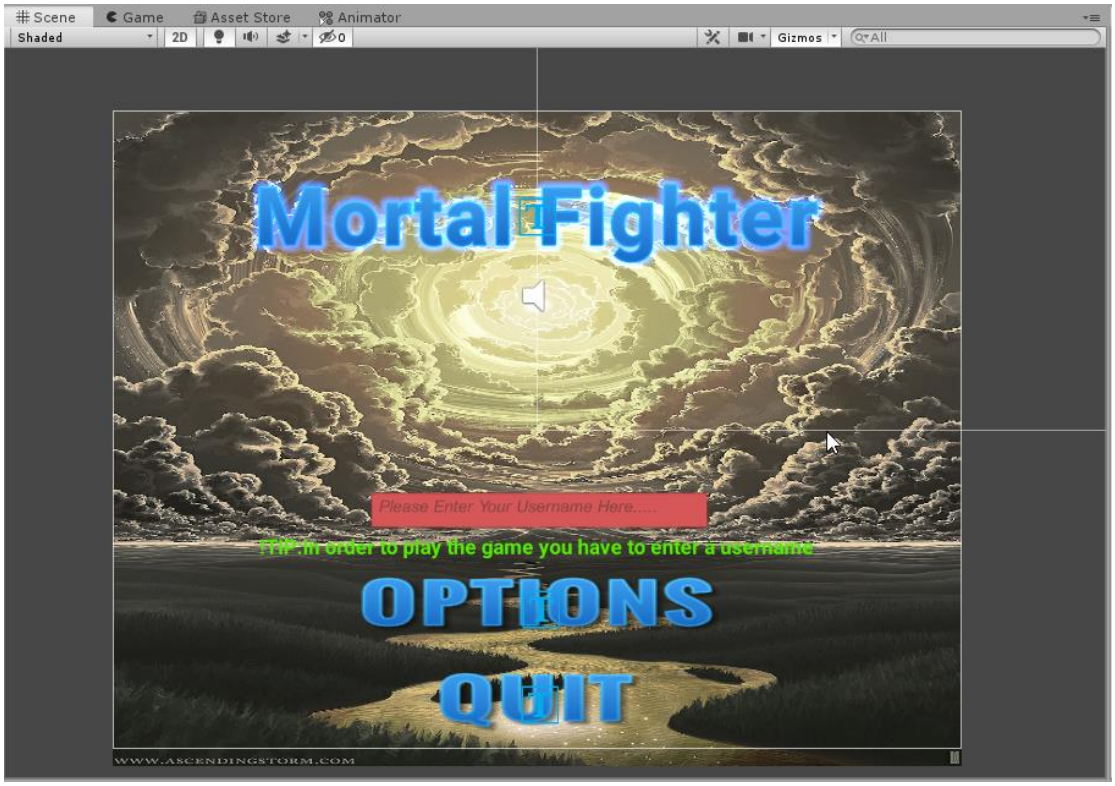
2 Το περιβάλλον του Unity

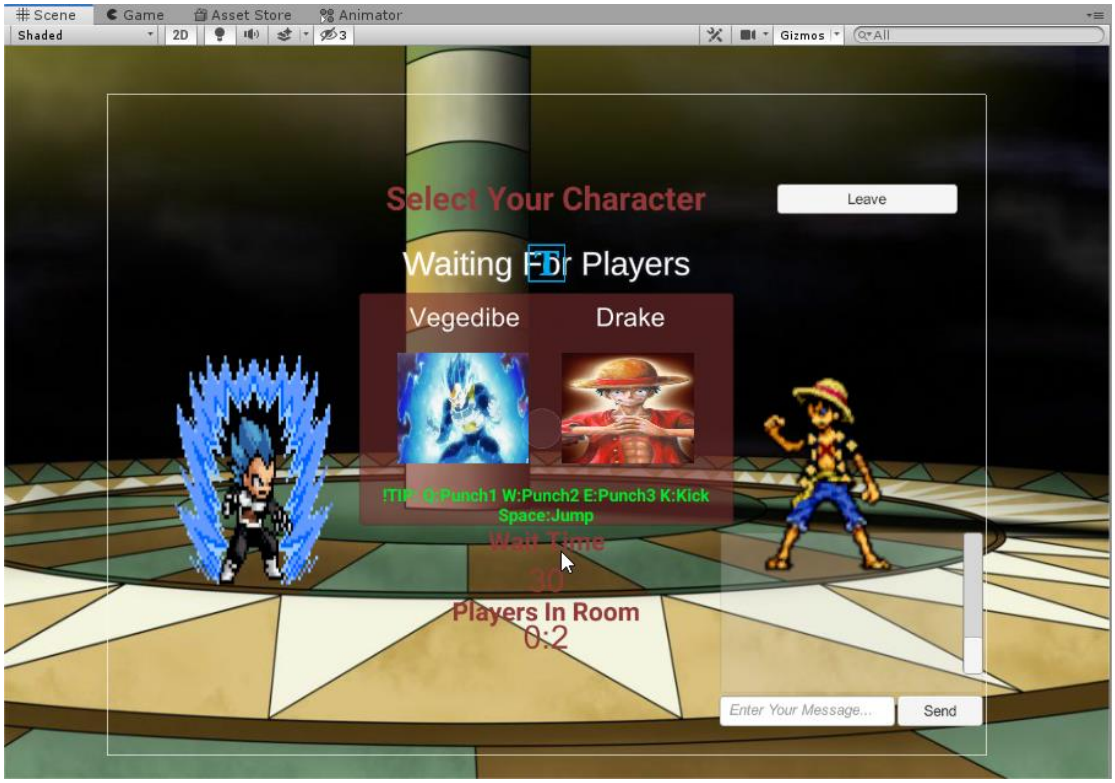
Το Unity περιέχει έναν επεξεργαστή (Editor) ο οποίος είναι το σημαντικότερο κομμάτι της μηχανής. Ο Επεξεργαστής, λοιπόν, περιέχει όλα τα assets του βιντεοπαιχνιδιού, δηλαδή το περιβάλλον, το έδαφος, τους χαρακτήρες κ.λπ.. Μέσα στον επεξεργαστή γίνεται η κατασκευή των επιπέδων(levels) που θέλουμε να έχει το βιντεοπαιχνίδι, χρησιμοποιώντας ήχους, φωτισμούς και άλλα. Η ανάπτυξη των στοιχείων του βιντεοπαιχνιδιού γίνεται μέσω του κώδικα ο οποίος είναι γραμμένος σε scripts σε συνδυασμό με τον επεξεργαστή. Τα scripts γράφονται σε C# χρησιμοποιώντας τον IDE του Visual Studio και διαφόρους άλλους IDE. Βέβαια, ο πιο διάσημος IDE του Unity είναι το Visual Studio. Ο Unity Editor φυσικά αποτελείται από πληθώρα χαρακτηριστικών. Επιγραμματικά τα κυριότερα από αυτά είναι το Scene, Game, Assets/Project, Inspector, Animator - Animation, Hierarchy, Debug Console και το Asset Store τα οποία θα αναλυθούν παρακάτω.

2.1 Κύρια χαρακτηριστικά Unity

2.1.1 Scene

Στο παράθυρο του scene γίνεται η τοποθέτηση των στοιχείων του βιντεοπαιχνιδιού που θέλουμε να χρησιμοποιήσουμε και να αλληλοεπιδράσουμε με αυτά όπως για παράδειγμα χαρακτήρες, δάπεδο κ.λπ.. Μπορούμε επίσης να ορίσουμε την οπτική γωνιά του βιντεοπαιχνιδιού και την απόσταση της κάμερας. Σε ένα βιντεοπαιχνίδι περιέχονται πολλές σκηνές τις οποίες τις ονομάζουμε και επίπεδα(levels). Παρακάτω θα δούμε κάποιες από τις σκηνές που περιέχει το παρόν βιντεοπαιχνίδι.





Εικόνα 5: Σκηνή 2

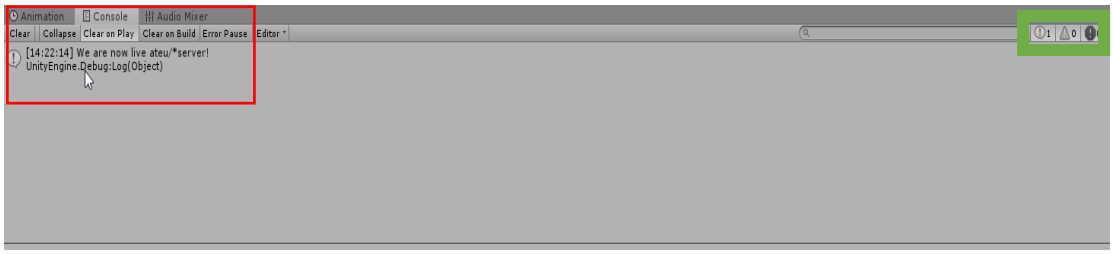
2.1.2 Game

Εδώ μπορεί να ελέγξει και να αλληλοεπιδράσει ο προγραμματιστής με το βιντεοπαιχνίδι σε πραγματικό χρόνο. Γεγονός που είναι ιδιαίτερα χρήσιμο καθώς καθίσταται δυνατός ο έλεγχος τυχών bugs καθώς επίσης και η διόρθωσή τους. Αυτό γίνεται ενώ βρισκόμαστε στον επεξεργαστή, χωρίς δηλαδή να χρειάζεται να γίνει build για να το δοκιμάσουμε την λειτουργία του βιντεοπαιχνιδιού. Για την εκκίνηση του παιχνιδιού απλά πατάμε το Play Button

Με το πάτημα του Play Button ο επεξεργαστής μας μεταφέρει στο Game Preview για να ξεκινήσει η αλληλεπίδραση όπως φαίνεται παρακάτω.

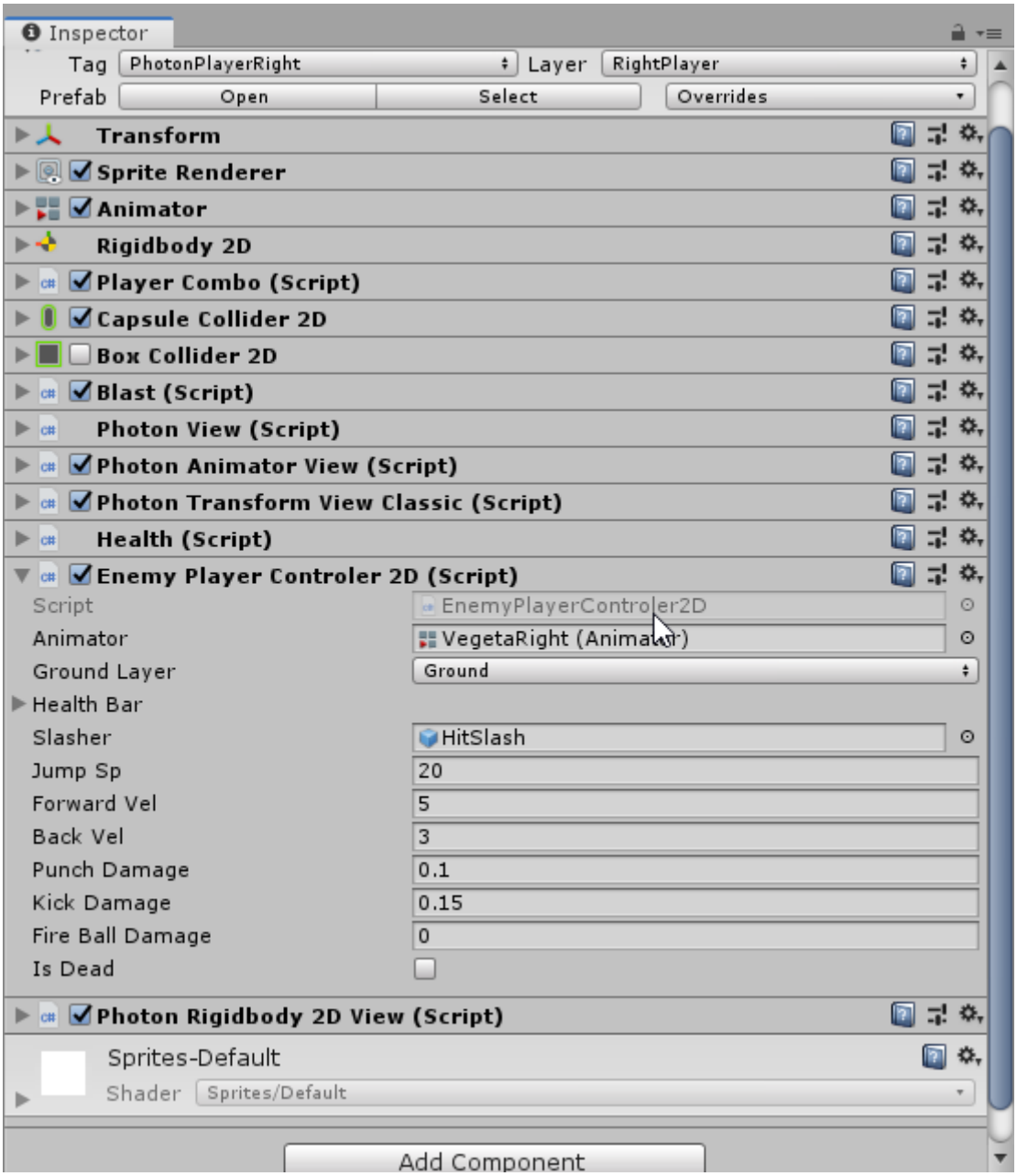
2.1.3 Debug Console

Στην παρακάτω κονσόλα μπορούμε να δούμε τα warnings και τα errors που εμφανίζονται στην περίπτωση κάποιας δυσλειτουργίας στον κώδικα καθώς και τα Debug.log του κώδικα. Η εμφάνιση γίνεται όπως φαίνεται στο κόκκινο περίγραμμα. Επίσης μπορούμε να δούμε και τον αριθμό των warnings και errors που μπορεί να υπάρχουν όπως φαίνεται στο πράσινο περίγραμμα.



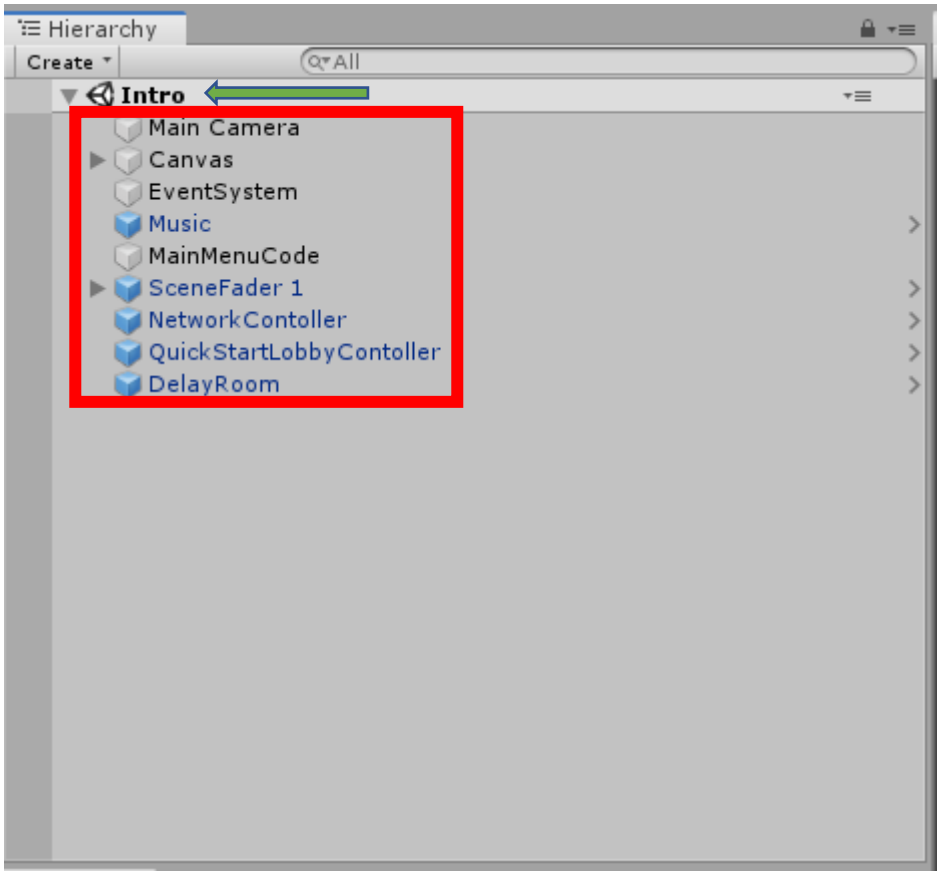
2.1.4 Inspector

Το παράθυρο inspector δείχνει όλα τα στοιχεία του αντικειμένου που έχουμε επιλέξει. Συγκεκριμένα εμφανίζει όλες τις λεπτομέρειες και πληροφορίες του επιλεγμένου Game Object συμπεριλαμβανομένων όλων των μεταβλητών που έχουμε βάλει να φαίνονται Public μέσα στα Scripts του αντικειμένου. Επίσης μας επιτρέπει να εισάγουμε και να τροποποιήσουμε τις τιμές των μεταβλητών του Game Object που έχουμε επιλέξει.



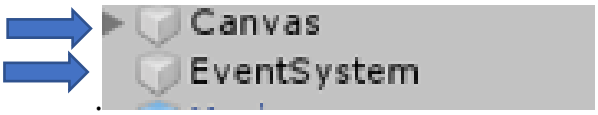
2.1.5 Hierarchy

Το Hierarchy όπως υποδηλώνει και το όνομα του αποτελεί μια ιεραρχία αντικειμένων που υπάρχουν σε μια σκηνή στην οποία εμείς ορίζουμε την σειρά των Game Objects. Αρχικά τα Game Objects από αυτόματη επιλογή ο επεξεργαστής τα βάζει με την σειρά που εισήχθησαν. Το παράθυρο της ιεραρχίας, λοιπόν, μας δίνει την δυνατότητα να τους αλλάξουμε θέσεις, να προσθέσουμε καινούργια και να αφαιρέσουμε ότι δεν χρειαζόμαστε. Έτσι το παράθυρο της ιεραρχίας περιέχει όλα τα αντικείμενα με την σειρά που τα έχουμε εισάγει εμείς. Κάθε φορά που προσθέτουμε ένα Game Object στην ιεραρχία αυτό εμφανίζεται αμέσως στην οθόνη της σκηνής του επεξεργαστή. Στην παρακάτω οθόνη με το πράσινο βελάκι βλέπουμε την ονομασία της σκηνής στην οποία είμαστε και στο κόκκινο κουτάκι την ιεραρχία των αντικειμένων που υπάρχουν στην συγκεκριμένη σκηνή.

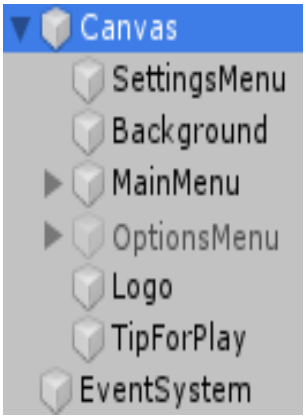


Εικόνα 6: Παράθυρο ιεραρχίας.

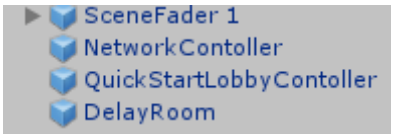
Επίσης στην εικόνα 4 παρατηρούμε ότι υπάρχουν κάποια Game Objects τα οποία έχουν και ένα βελάκι διπλά τους. Όπως φαίνεται παρακάτω.



Αν πατήσουμε πάνω στο Canva θα παρατηρήσουμε τα εμφωλευμένα Game Object που έχουμε περάσει ως child στον Canva με drag and drop. Επίσης παρατηρούμε ότι μπορούμε να έχουμε και άλλα εμφωλευμένα Game Objects όπως για παράδειγμα μέσα στον Canva έχουμε ως child το Options Button που έχει και αυτό child to Options Game object

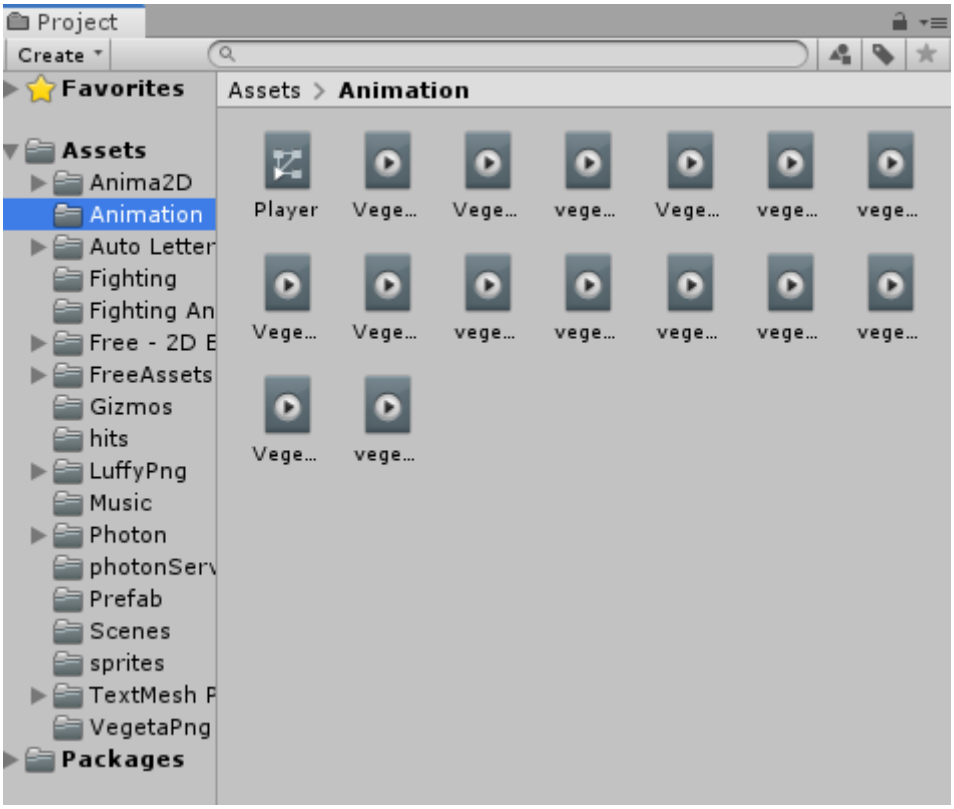


Υπάρχουν και κάποια Game Objects όπως φαίνεται παρακάτω τα οποία είναι μπλε. Αυτά ονομάζονται prefabs. Στην συνέχεια θα εξηγηθεί τι ονομάζουμε με τον όρο prefabs.



2.1.6 Assets/Project

Στο φάκελο assets βρίσκονται όλα τα αρχεία του βιντεοπαιχνιδιού τα οποία θέλουμε να επεξεργαστούμε. Από τον συγκεκριμένο φάκελο μπορούμε να πάρουμε οτιδήποτε χρειαζόμαστε και να το εισάγουμε στην ιεραρχία ώστε να μπορέσουμε να το επεξεργαστούμε περεταίρω . Επίσης μπορούμε να εισάγουμε textures, materials, animation, γενικά δηλαδή όλα τα assets τα οποία μπορεί να χρειαστούμε. Εδώ θα βρούμε επίσης και τον κώδικα που έχουμε γράψει(δηλαδή τα scripts).



Εικόνα 7: Project assets

Στην προηγούμενη εικόνα μπορούμε να παρατηρήσουμε ένα φάκελο ο οποίος ονομάζεται Prefab. Μέσα σε αυτόν τον φάκελο υπάρχουν όλα τα Game Objects τα οποία έχουμε φτιάξει στην ιεραρχία και έχουμε κάνει drag and drop στον Prefab φάκελο. Αυτά τα Game Objects μπορούμε πλέον να τα πάρουμε από τον συγκεκριμένο φάκελο και να τα επαναχρησιμοποιήσουμε όσες φορές θέλουμε στην σκηνή που είμαστε ή και σε άλλες σκηνές.

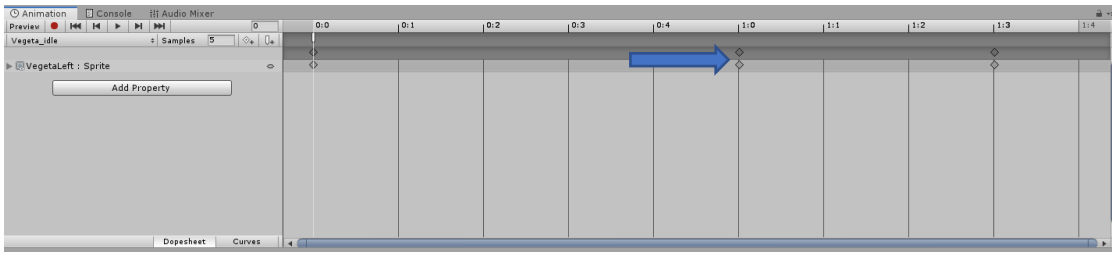
Εικόνα 8: Prefab αρχείο με prefab Game Objects

2.1.7 Animator – Animation

Στο εν λόγω παράθυρο μπορούμε να δώσουμε κίνηση στα στοιχεία του βιντεοπαιχνιδιού. Συγκεκριμένα στο παράθυρο του Animation μπορούμε να εισάγουμε τα sprites μας, δηλαδή τις φωτογραφίες που έχουμε κόψει καρέ - καρέ και να δημιουργήσουμε κίνηση όπως φαίνεται στην παρακάτω εικόνα.

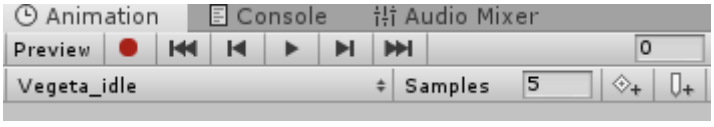
Εικόνα 9: Κίνηση χαρακτήρα καρέ – καρέ.

Στο παραπάνω παράδειγμα επιθυμούμε μέσα στο βιντεοπαιχνίδι να δημιουργηθεί η αίσθηση της κίνησης , δηλαδή η αίσθηση ότι ο χαρακτήρας περπατάει είτε μπροστά είτε πίσω. Ας σημειωθεί ότι η τέχνη που χρησιμοποιείτε εδώ είναι pixel και ο χαρακτήρας είναι σχεδιασμένος σε «τετραγωνάκια». Για να δημιουργήσουμε αυτήν την αίσθηση της κίνησης, λοιπόν, πρέπει στο παράθυρο του animation να εισάγουμε με drag and drop τις εικόνες με την σειρά την οποία θέλουμε.



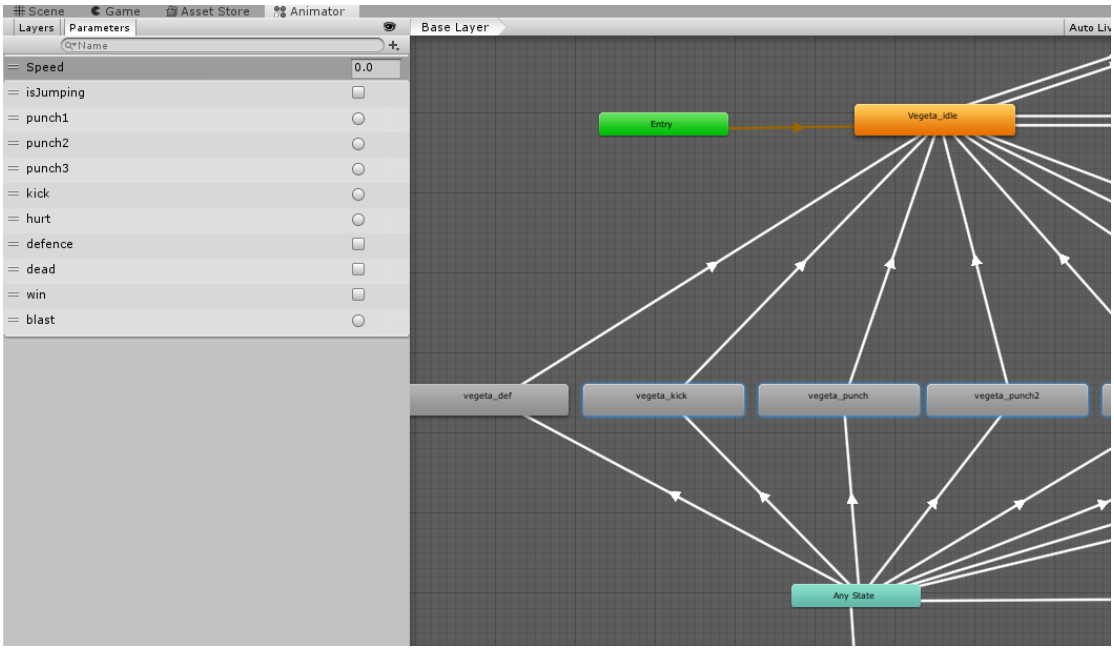
Εικόνα 10: Παράθυρο animation.

Οι ρόμβοι που φαίνονται στην εικόνα αποτελούν τα sprites που έχουν εισαχθεί. Στο συγκεκριμένο παράδειγμα θέλουμε να κινείται ελαφρά ο χαρακτήρας ακόμη και στην περίπτωση που είναι αδρανής, ώστε να δημιουργείται η ψευδαίσθηση ότι είναι «ζωντανός». Στην παρακάτω εικόνα φαίνεται η ονομασία του συγκεκριμένου Animation και του Samples που δηλώνει τα fps του χαρακτήρα, δηλαδή τον αριθμό των εικόνων(καρέ) τα οποία βλέπουμε στην οθόνη μας ανά δευτερόλεπτο. Κατά αυτόν τον τρόπο επιτυγχάνεται η κίνηση.



Επίσης πατώντας το πλήκτρο που φαίνεται στην παρακάτω εικόνα μπορούμε να «σηκώσουμε» ένα συμβάν(event) μέσα σε ένα συγκεκριμένο καρέ. Παραδείγματος χάριν είναι δυνατό να δημιουργήσουμε μέσα στον κώδικα ένα συμβάν όπου ο χαρακτήρας πετάει projectiles(βλήματα). Συνεπώς το συμβάν πρέπει να «σηκωθεί» μόνο όταν ο χαρακτήρας βρίσκεται στο καρέ στο οποίο σηκώνει το χέρι του. Ας τονίσουμε λοιπόν ότι σε εκείνο το καρέ αποκλειστικά απαιτείται η εισαγωγή του συμβάντος.

Στο παράθυρο του Animator μπορούμε να δηλώσουμε τις διάφορες καταστάσεις, δηλαδή την κατάσταση ακινησίας, κίνησης κ.λπ.

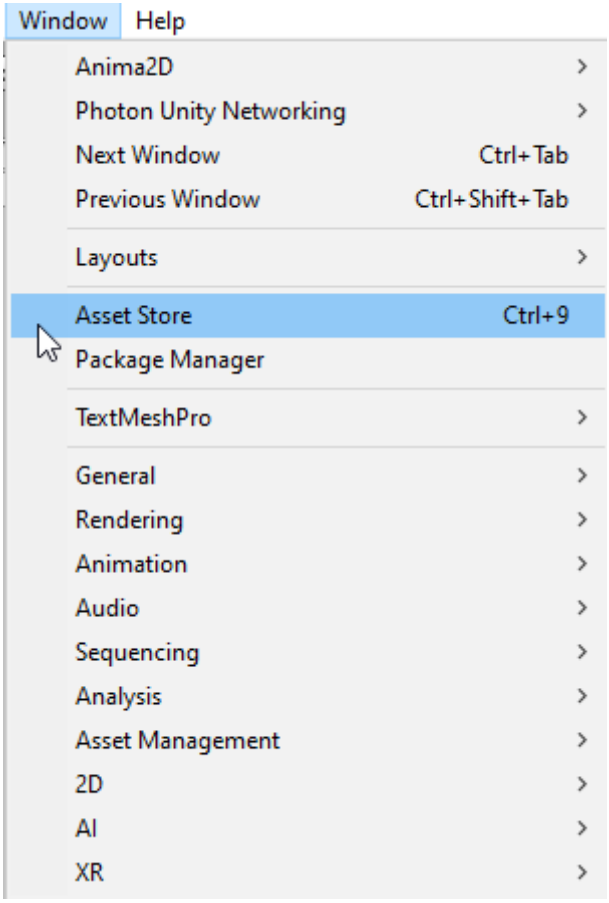


Εικόνα 11: Παράθυρο Animator με κάποιες από τις καταστάσεις.

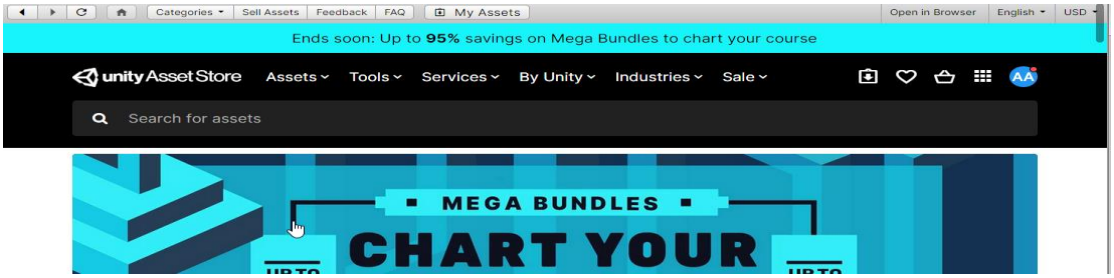
Στις καταστάσεις που φαίνονται στην εικόνα 9 μπορούμε να εισάγουμε και παραμέτρους οι οποίες προσδιορίζουν πότε θέλουμε μια κατάσταση να ξεκινάει ή να σταματήσει.

2.1.8 Asset Store

Το ηλεκτρονικό κατάστημα του Unity το οποίο ονομάζετε Asset Store διαθέτει πλήθος από χρήσιμα assets τα οποία έχουν στόχο να καταστήσουν την δημιουργία των βιντεοπαιχνιδιών ιδιαίτερα ευκολότερη όσον αφορά το γραφικό κομμάτι. Το κατάστημα περιέχει χιλιάδες δωρεάν assets τα οποία είναι εξαιρετικά, καθώς επίσης περιέχει και assets που είναι ιδιαίτερα οικονομικά. Το asset store βρίσκεται στην εργαλειοθήκη του επεξεργαστή του Unity, δηλαδή βρίσκεται στο window της παρακάτω εικόνας.



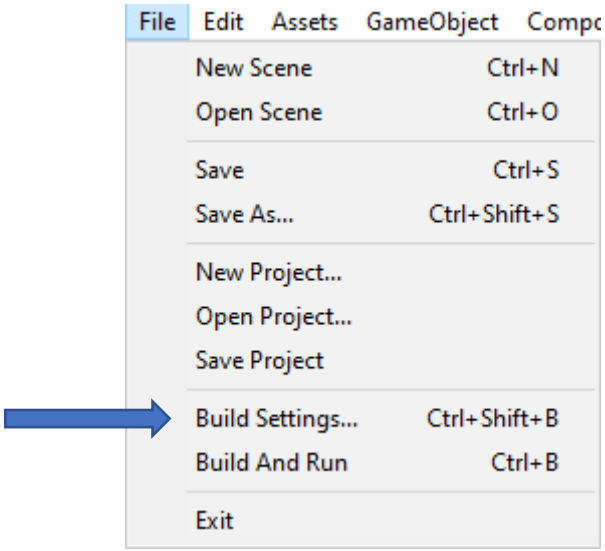
Επίσης το Unity asset store είναι διαθέσιμο και στην επίσημη ιστοσελίδα του Unity. Σε αυτήν περιέχει αρκετά περισσότερα assets από ότι στον επεξεργαστή του Unity. Unity asset συνήθως αποτελεί οποιοδήποτε αρχείο έχει δημιουργηθεί εκτός του Unity, όπως για παράδειγμα ένα μοντέλο 3D, ένα αρχείο ήχου, μια εικόνα ή οποιοδήποτε αλλού τύπου αρχείο το οποίο υποστηρίζεται από το Unity. Υπάρχουν επίσης ορισμένοι τύποι assets που μπορούμε να δημιουργήσουμε εντός του Unity.



2.2 Δημιουργία αρχείου παιχνιδιού

Για την δημιουργία του αρχείου θα χρειαστεί να κάνουμε κάποιες ρυθμίσεις πρώτα. Αρχικά θα πρέπει να επιλέξουμε ανάμεσα στις επιλογές που υπάρχουν στον επεξεργαστής του Unity σχετικά με την επιθυμητή πλατφόρμα μέσα στην οποία θα παίξει το εκτελέσιμο. Τέτοιες πλατφόρμες για παράδειγμα είναι τα Windows, Mac, Xbox, PlayStation κ.λπ.

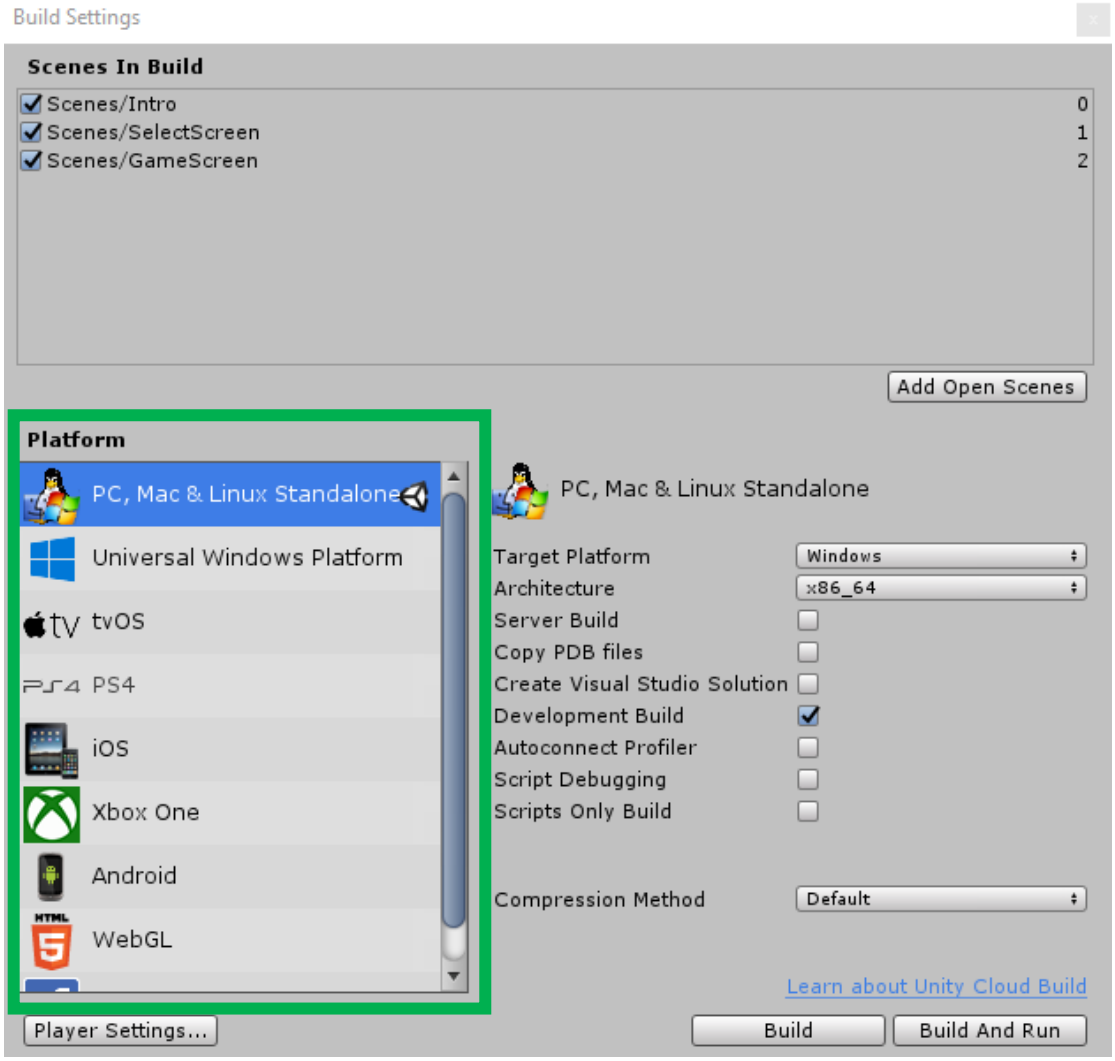
Βήμα 1ο) Πατάμε στην επιλογή file/φάκελος μετά Build Settings



Επιπλέον μπορούμε να διαλέξουμε την επιλογή του Build And Run για να τρέξει το εκτελέσιμο αμέσως. Πρώτα όμως θα πρέπει να φτιάξουμε τα Build Settings.

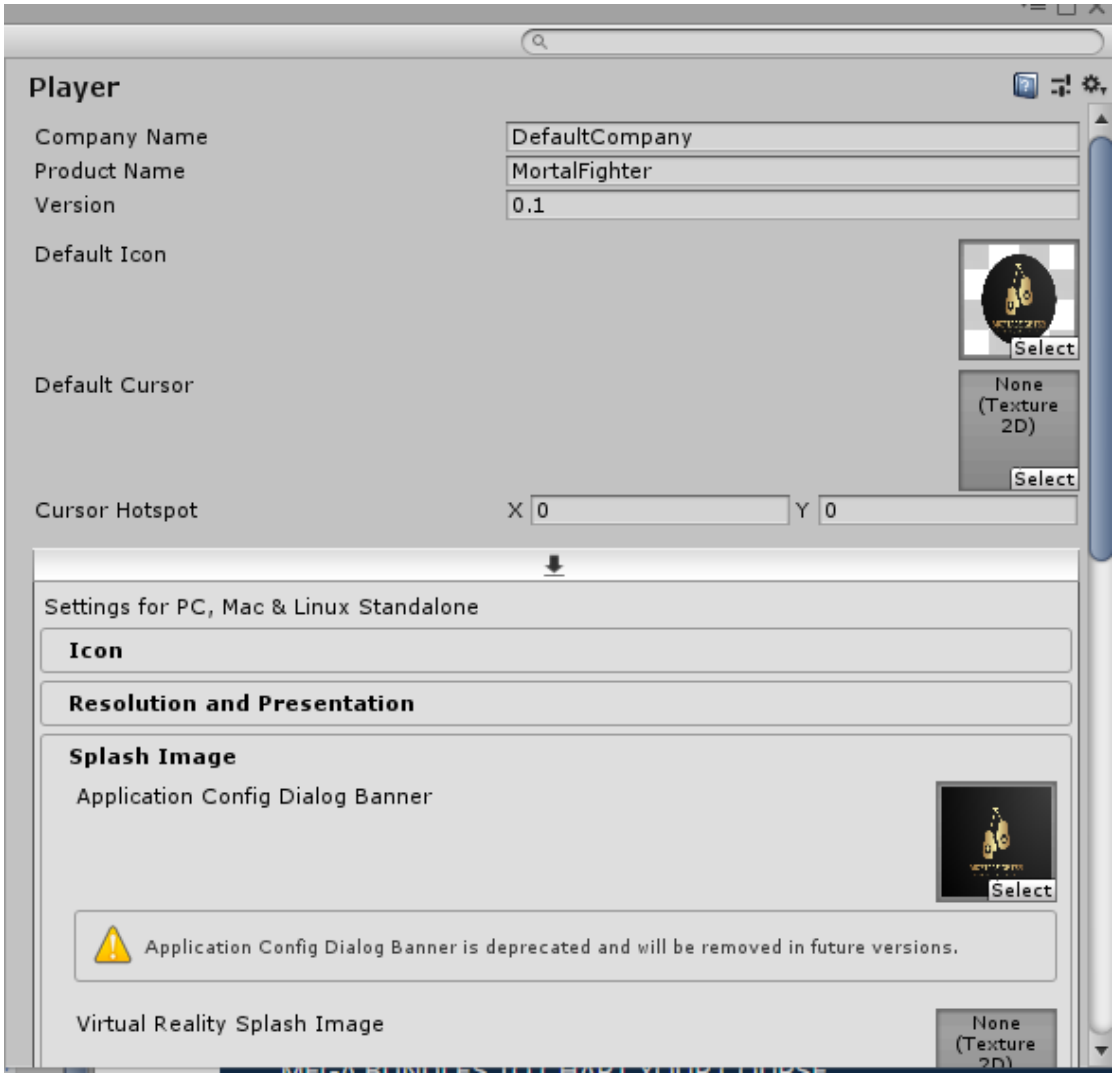
(Σημείωση: Για το παρόν project δεν χρησιμοποιήθηκαν παραπάνω από τρείς σκηνές καθώς είναι βιντεοπαιχνίδι μάχης).

Βήμα 2ο) Διαλέγουμε την πλατφόρμα στην οποία θέλουμε να παίξει το εκτελέσιμο βιντεοπαιχνίδι.



Εικόνα 12: Παράθυρο Build Settings.

Βήμα 3ο) Πατάμε στην επιλογή Player Settings ώστε να μας εμφανιστεί ένα νέο παράθυρο στο οποίο έχουμε την δυνατότητα να εισάγουμε το όνομα του βιντεοπαιχνιδιού καθώς και την εταιρεία παραγωγής του, εφόσον αυτή υπάρχει. Επίσης μπορούμε να εισάγουμε και το εικονίδιο του εκτελέσιμου όπως φαίνεται στην εικόνα 10.



Εικόνα 13: Παράθυρο Player Settings.

3 Βοηθητικά εργαλεία που χρησιμοποιήθηκαν

3.1 Εργαλείο επεξεργασίας εικόνας (GIMP)

Για την δημιουργία κάποιων γραφικών στοιχείων του βιντεοπαιχνιδιού έγινε χρήση του εργαλείου επεξεργασίας εικόνας GIMP. Το Gimp είναι ένα πρόγραμμα επεξεργασίας εικόνας πολλαπλών πλατφορμών διαθέσιμο και για GNU/Linux, OS X, Windows καθώς και για άλλα πολλά λειτουργικά συστήματα. Αποτελεί δωρεάν λογισμικό, ένα open source. Πράγμα το οποίο σημαίνει ότι μπορούμε να επεξεργαστούμε τον κώδικα του και να διανείμουμε τις τυχόν αλλαγές σε αυτόν. Επίσης το GIMP παρέχει εξαιρετικά εξελιγμένα εργαλεία για την ολοκλήρωση οποιαδήποτε είδους γραφικής εργασίας επιθυμούμε να πραγματοποιήσουμε είτε η ιδιότητά μας είναι γραφίστας, φωτογράφος, εικονογράφος ή επιστήμονας. Μπορούμε να βελτιώσουμε περαιτέρω την παραγωγικότητα μας με το GIMP χάρη στις πολλές επιλογές προσαρμογής, καθώς επίσης και στα προσθετά τρίτων(Third Party Plugins) που παρέχει.

Εικόνα 14: GIMP Logo.

Εικόνα 15: Δημιουργία Splash image του βιντεοπαιχνιδιού.

3.2 Διαδικτυακός προγραμματισμός

Το εργαλείο που χρησιμοποιήθηκε ώστε το παιχνίδι να είναι διαδικτυακό (online) ονομάζετε Photon. Με τον όρο διαδικτυακό βιντεοπαιχνίδι μάχης, όπως αυτό της παρούσας εργασίας, εννοούμε ένα παιχνίδι στο οποίο μπορούν οι διάφοροι παίκτες να παίξουν απομακρυσμένα ως αντίπαλοι. Επίσης το εν λόγω βιντεοπαιχνίδι διαθέτει και chat ώστε να είναι δυνατή η συνομιλία με τον αντίπαλο πριν διεξαχθεί η μάχη. Για την δημιουργία του chat χρησιμοποιήθηκε μια υποκατηγορία του photon η οποία ονομάζετε photon chat. Τα δυο προαναφερθέντα εργαλεία θα αναλυθούν παρακάτω.

3.2.1 Photon

Το Photon Unity Networking(PUN) είναι ένα πακέτο του Unity που υποστηρίζει την ύπαρξη πολλών απομακρυσμένων παικτών στα βιντεοπαιχνίδια, δηλαδή είναι μια πλατφόρμα που μας προσφέρει πολλές δυνατότητες για διαδικτυακό προγραμματισμό. Συγκεκριμένα παρέχει το δικό του cloud και server, έτσι ώστε να μπορεί ο προγραμματιστής να ασχοληθεί με την παραγωγή του βιντεοπαιχνιδιού και μόνον. Αρχικά δημιουργήθηκε δίλημμα σχετικά με τη χρησιμοποίηση του εργαλείου που παρείχε το ίδιο το Unity,δηλαδή το εργαλείο Unity Multiplayer, στη θέση του Photon. Στη συνέχεια, όμως, κρίθηκε το Photon καταλληλότερο. Η συγκεκριμένη απόφαση πάρθηκε καθώς το άλλο υποψήφιο εργαλείο είναι υπό κατάργηση και ενδέχεται να αλλάξει αρκετά σύντομα. Το Photon επίσης αποτελεί ένα ολοκληρωμένο πρόγραμμα στην κατηγορία του παρέχοντας όλα τα εργαλεία που χρειάζονται για να καταστεί δυνατή η ανάπτυξη ενός διαδικτυακού βιντεοπαιχνιδιού. Τα σημαντικότερα εργαλεία του Photon που χρησιμοποιήθηκαν είναι τα εξής:

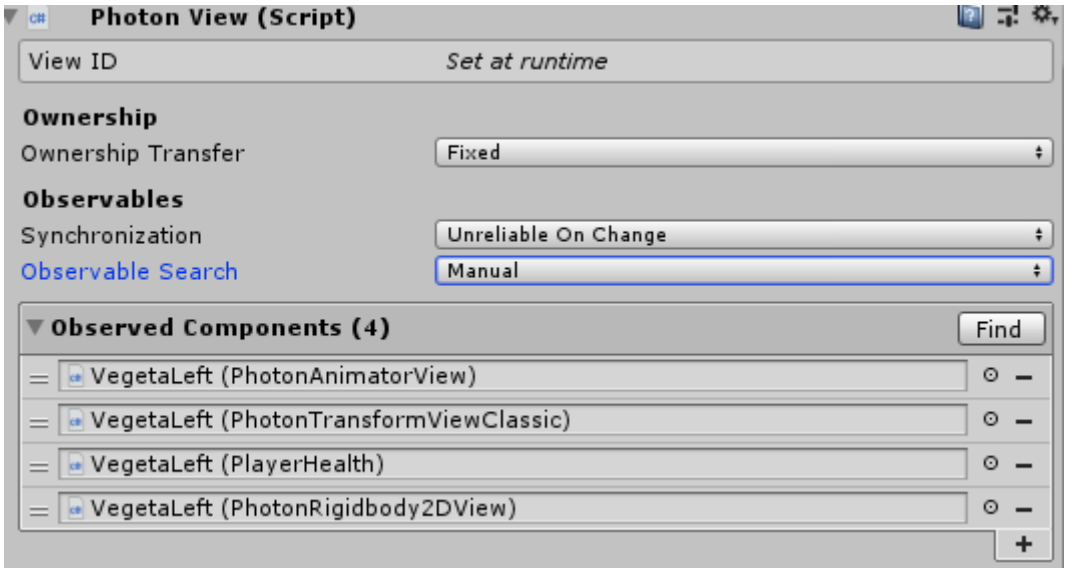
- Photon View
- Photon RigidBody2D View
- Photon Transform View Classic (Υπάρχει και το Photon Transform View που είναι νέα έκδοση για αυτό δίνεται έμφαση στο classic)
- Photon Animator View

Παρακάτω θα γίνει η ανάλυση για καθένα εργαλείο που αναφέρθηκε.

Εικόνα 16: Photon And Unity Logos

Photon View

Από όλα τα εργαλεία που χρησιμοποιήθηκαν το Photon View είναι το σημαντικότερο, καθώς μέσω αυτού του εργαλείου γίνεται η εμφάνιση των Game Objects σε όλους τους παίκτες συγχρονισμένα. Με άλλα λόγια τοποθετώντας το συγκεκριμένο component σε ένα Game Object μπορούμε να κάνουμε ορατή την κίνηση των χαρακτήρων, τα physics του βιντεοπαιχνιδιού ή οποιοδήποτε άλλο αντικείμενο θέλουμε να γίνει ορατό σε όλους τους παίκτες την ίδια στιγμή. Στην παρακάτω εικόνα φαίνεται η τοποθέτηση ενός Photon View σε ένα από τους χαρακτήρες του βιντεοπαιχνιδιού.

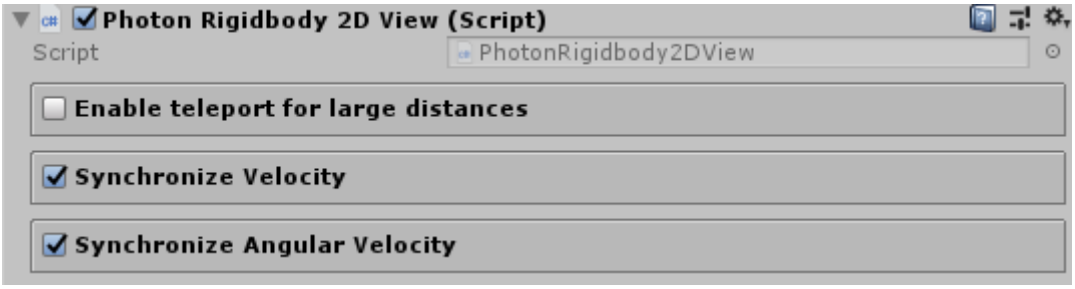


Παρατηρούμε επίσης ότι έχουμε τοποθετήσει στο παράθυρο του observed components τα Photon Animator View, Photon Transform View Classic, Player Health, Photon Rigid body2D View. Με αυτόν τον τρόπο τα συγκεκριμένα στοιχεία γίνονται ορατά και στους άλλους παίκτες. Παρακάτω θα αναλυθούν καθένα από τα στοιχεία.

Το Player Health είναι ένα script στην κλάση του οποίου έχει εισαχθεί το IPunObservable. Έτσι το συγκεκριμένο script καθίσταται ορατό σε όλους. Η λειτουργία του Player Health θα αναλυθεί περεταίρω παρακάτω.

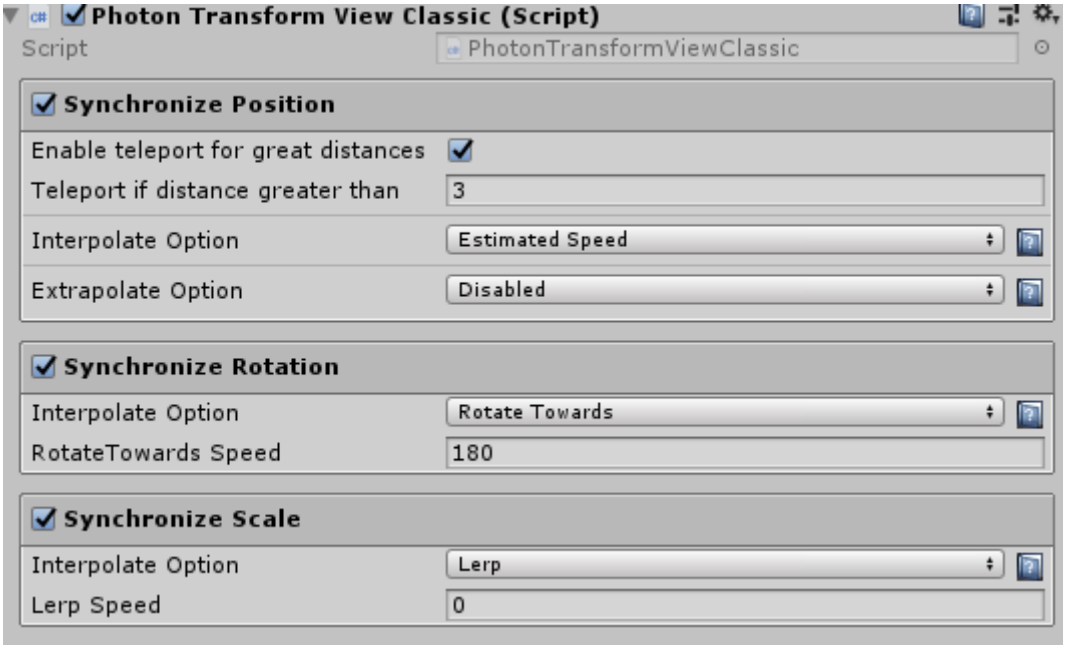
Photon Rigidbody2D View

Το Photon Rigid Body2D παρακολουθεί το Rigid Body του αντικειμένου. Κατά αυτόν τον τρόπο μπορούμε να συγχρονίσουμε σε όλους του παίκτες την ταχύτητα ενός αντικειμένου. Παραδείγματος χάριν μπορεί να γίνει ορατή η κίνηση ενός χαρακτήρα του βιντεοπαιχνιδιού.



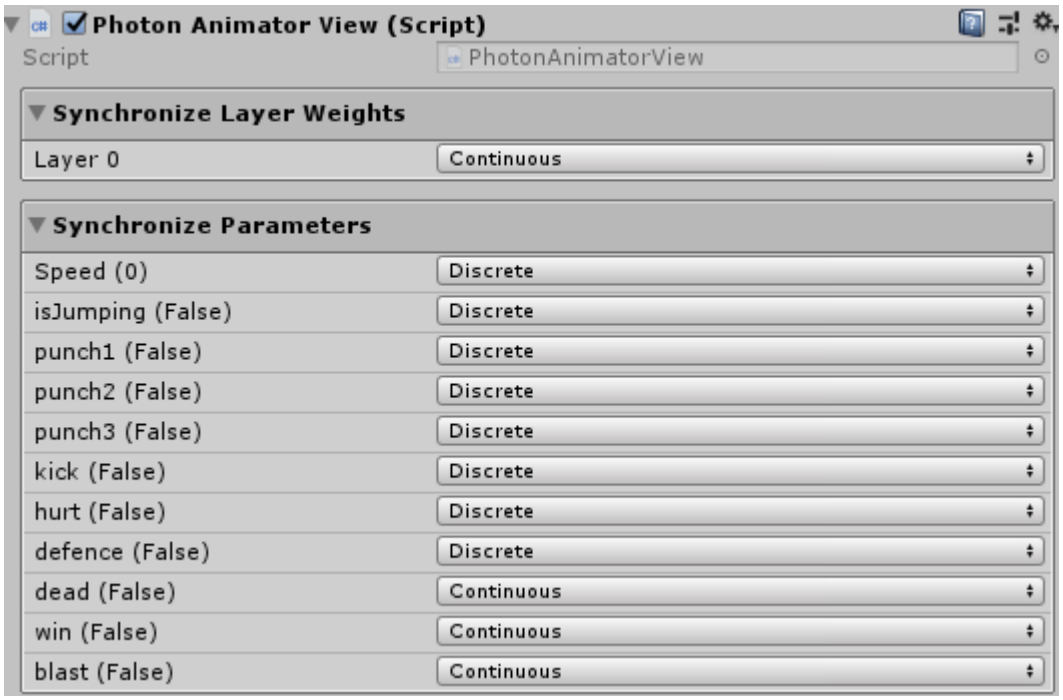
Photon Transform View Classic

Το Photon transform view classic παρακολουθεί όλες τις αλλαγές που μπορεί να προκύψουν στα διάφορα αντικείμενα μέσα στο βιντεοπαιχνίδι, δηλαδή οποιαδήποτε περιστροφή, κλιμάκωση ή μετακίνηση τους.



Photon Animator View

Το Photon Animator View χρησιμοποιήθηκε για την παρακολούθηση οποιουδήποτε animation καθώς και των παραμέτρων που έχουν τα animation .



Το προγραμματιστικό κομμάτι της επικοινωνίας έγινε με

- RPCs
- Custom Properties

Παρακάτω γίνεται μια σύντομη αναφορά σχετικά με το RPC και Custom Properties.

Photon RPC

Το RPC ή αλλιώς Remote Procedure Calls αποτελεί μια μέθοδο που καλείτε για να ενημερώσει όλους τους απομακρυσμένους clients οι οποίοι βρίσκονται στο ίδιο δωμάτιο. Ένα Photon RPC μπορεί να κληθεί μόνο από αντικείμενα που έχουν Photon View. Επίσης μπορούμε να επιλέξουμε σε ποιους client θέλουμε να στείλουμε το RPC, με την εντολή RpcTarget όπως φαίνεται στην παρακάτω εικόνα. Γενικά υπάρχει πλήθος επιλογών παραδείγματος χάριν μια από αυτές είναι να στείλουμε το RPC σε όλους τους clients δηλαδή και σε αυτούς που θα μπουν αργότερα στο βιντεοπαιχνίδι. Ειδικά με την εντολή AllBufferd, χωρίς την λέξη Buffered γίνεται η κλήση των clients που βρίσκονται αυτήν την στιγμή στο βιντεοπαιχνίδι.

```
if (GameController.gameController.isGameFinished && GameController.gameController.isPlayerWin)
{
    animator.SetBool("win", true);
    PV.RPC("RPC_WinnersName", RpcTarget.AllBuffered);
    GameController.gameController.AddWinLeft(PhotonNetwork.PlayerList[0], 1);
    return;
}
```

Εικόνα 17: Παράδειγμα RPC από το project.

```
[PunRPC]
0 references
void RPC_WinnersName(){
    GameController.gameController.WinnersName.text = PlayerName.text;
}
```

Εικόνα 18: Η μέθοδος του RPC

Custom Properties

Τα Custom Properties χρησιμοποιήθηκαν ώστε να μπορούμε να κρατάμε τα wins του κάθε παίκτη διαδικτυακά καθώς και να τα διαχειριζόμαστε. Για παράδειγμα όταν ο Player1 μηδενίσει την ζωή του Player2 στη μάχη θα πρέπει να προσθέσουμε μία νίκη (win) υπέρ του Player1 και παράλληλα αυτή η επιπλέον νίκη θα πρέπει να είναι ορατή ως πληροφορία για κάθε χρήστη που επιθυμεί να δει τα στατιστικά Τέτοιες μικρές αλλαγές μπορούν να επιτευχθούν χάρη στην Custom Properties.

3.2.2 Photon Chat

Το Photon Chat είναι ένα εργαλείο διαδικτυακής συνομιλίας. Μπορεί κάθε παίκτης που έχει συνδεθεί στο παιχνίδι δίνοντας ένα όνομα να επικοινωνήσει με οποιονδήποτε άλλον παίχτη βρίσκεται στο ίδιο δωμάτιο. Για να λειτουργήσει το Photon Chat χρειαζόμαστε κάποιες μεθόδους που μας δίνει το Photon. Η αναπαραγωγή των όλων των μεθόδων του Photon Chat γίνεται όταν εισάγουμε το interface IChatClientListener στον κώδικα διπλά από την MonoBehaviour κλάση. Παρακάτω θα αναλύσουμε κάποιες από τις αυτόματες μεθόδους που χρησιμοποιήσαμε για να γίνει δυνατή η επικοινωνία μεταξύ των παικτών.

- OnConnected
- OnSubscribed
- OnGetMessages
- OnPrivateMessage

OnConnected

Χρησιμοποιώντας την μέθοδο OnConnected γίνεται δυνατή η χρήση του Chat από τους χρήστες του δωμάτιου. Στην μέθοδο αυτή στο παρακάτω παράδειγμα εισάγουμε και κάποιες εντολές όπως την Subscribe και την SetOnlineStatus. Με την εντολή Subscribe έχουμε την δυνατότητα να εγγράψουμε τον χρήστη σε οπουδήποτε κανάλι

του chat θέλουμε το οποίο είναι δημόσιο και το έχουμε δημιουργήσει στον κώδικα. Επίσης με την εντολή SetOnlineStatus εισάγουμε τον χρήστη την διαδικτυακή κατάσταση στην οποία βρίσκεται, δηλαδή ενεργός ή ανενεργός.

```
3 references
public void OnConnected()
{
    Debug.Log("OnConnected");
    chatClient.Subscribe("gamechannel");
    chatClient.SetOnlineStatus(ChatUserStatus.Online);
}
```

OnSubscribed

Η μέθοδος OnSubscribed καλείται όταν ένας χρήστης εγγράφετε στο δημόσιο κανάλι που είναι ενεργό. Συνήθως στην OnSubscribed χρησιμοποιούμε την εντολή Publish Message για να γνωστοποιήσουμε στον χρήστη ότι συνδέθηκε στο κανάλι.

```
3 references
public void OnSubscribed(string[] channels, bool[] results)
{
    if (channels.Length == 1 && results[0] && channels[0].Equals(channelName, StringComparison.OrdinalIgnoreCase))
    {
        foreach (string channel in channels)
        {
            chatClient.PublishMessage(channel, "says 'hi'."); // you don't HAVE to send a msg on join but you could.
        }
        Debug.Log("Subscribed");
    }
}
```

OnGetMessages

Με την μέθοδο OnGetMessages μπορούμε να αποθηκεύσουμε όλα τα μηνύματα που έχουν σταλθεί στο κανάλι στο οποίο είναι συνδεδεμένος ο χρήστης. Έτσι όταν ένας νέος χρήστης συνδεθεί στο chat μπορεί να δει τα μηνύματα και τις συνομιλίες που έχουν προηγηθεί στο συγκεκριμένο κανάλι εφόσον αυτό είναι δημόσιο.

```
3 references
public void OnGetMessages(string channelName, string[] senders, object[] messages)
{
    for (int i = 0; i < senders.Length; i++)
    {
        msgArea.text += senders[i] + " : " + messages[i] +"\n";
    }
}
```

OnPrivateMessage

Στην περίπτωση ενός χρήστης που θελήσει να στείλει ένα μήνυμα το οποίο να μην είναι ορατό στο δημόσιο κανάλι, πρέπει να χρησιμοποιηθεί η OnPrivateMessage. Με αυτόν τον τρόπο μπορούμε να καταγράψουμε τον πομπό και το μήνυμα που θέλει να στείλει καθώς επίσης και τον δεκτή που θα παραλάβει το μνήμα. Για την αποστολή

ενός προσωπικού μηνύματος θα πρέπει ο αποστολέας να γνωρίζει το όνομα του αποδέκτη όπως ο τελευταίος το έχει δηλώσει κατά την εγγραφή του στο κανάλι.

```
public class MyChatListner : IChatClientListener
{
    public void OnPrivateMessage( string sender, object message, string channelName )
    {
       Console.WriteLine( "OnPrivateMessage: {0} ({1}) > {2}", channelName, sender, message );
       // All private messages are automatically cached in `ChatClient.PrivateChannels`, so you don't
       // A channel name is applied as key for `PrivateChannels`.
       // Get a (remote) user's channel name with `ChatClient.GetPrivateChannelNameByUser(name)`.
       // e.g. To get and show all messages of a private channel:
       // ChatChannel ch = this.chatClient.PrivateChannels[ channelName ];
       // foreach ( object msg in ch.Messages )
       // {
       //     Console.WriteLine( msg );
       // }
    }
```

(Σημείωση: Τα μηνύματα για όσο είναι ενεργό ένα κανάλι αποθηκεύονται στο cloud που παρέχει το Photon)

4 Ανάπτυξη παιχνιδιού

Το παρόν βιντεοπαιχνίδι είναι διαδικτυακό καθώς θεωρήθηκε ότι δεν υπάρχει η ανάγκη για την δημιουργία ενός παιχνιδιού που θα λειτουργεί μόνο τοπικά. Άλλωστε η ανάπτυξη ενός τοπικού παιχνιδιού αποτελεί ένα θέμα το οποίο έχει καλυφθεί πολλές φορές στο παρελθόν. Παρακάτω θα αναλυθεί διεξοδικά ο τρόπος δημιουργίας του παιχνιδιού. Αρχικά θα αναλυθεί ο κώδικας με τις κλάσης που χρησιμοποιήθηκαν ώστε να συνδεθεί το παιχνίδι διαδικτυακά, καθώς επίσης και τα προβλήματα που προέκυψαν και πως ξεπεράστηκαν. Έπειτα θα περιγραφεί το κομμάτι του κώδικα και των κλάσεων που χρησιμοποιήθηκαν για την κίνηση των παικτών, την μάχη κ.λπ.

(Σημείωση: Ο κώδικας δεν είναι τέλειος και υπάρχουν αρκετές μεταβλητές οι οποίες μπορεί να μην έχουν λόγο ύπαρξης καθώς δεν χρειάστηκαν εν τέλει).

4.1 Διαχείριση και δημιουργία διαδικτυακού παιχνιδιού

Για την σύνδεση του Photon με το Unity στο κομμάτι του προγραμματισμού έπρεπε να δημιουργηθούν κάποιες κλάσης, όπως παρουσιάζονται παρακάτω.

- NetworkController.cs
- QuickStartLobbyController.cs
- DelayRoom.cs
- DelayStartWaitingRoomController.cs

NetworkController.cs

Για να επιτευχθεί η σύνδεση με τους server του Photon πρέπει να αλλάξουμε την κλάση από MonoBehaviour σε MonoBehaviourPunCallbacks. Κατά αυτόν τον τρόπο μπορούν οι Photon server να δημιουργήσουν τις απαραίτητες κλήσης μεθόδων που απαιτούνται από το script στον server. Έπειτα πρέπει να χρησιμοποιηθεί η εντολή του Photon που ονομάζετε ConnectUsingSettings. Με αυτήν την εντολή γίνεται η σύνδεση του βιντεοπαιχνιδιού μας με τον server του Photon διατηρώντας τις ρυθμίσεις που έχει από επιλογή το Photon. Έπειτα για να διαχειριστούμε την σύνδεση μας πρέπει να χρησιμοποιούμε την μέθοδο OnConnectedToMaster στην οποία έχουμε εισάγει την εντολή Join Lobby. Αυτό είναι απαραίτητο ώστε να μπορεί ο χρήστης να εισαχθεί σε Lobby μετά την είσοδο του στους server. Ταυτόχρονα σε αυτήν την κλάση διαχειριζόμαστε και το όνομα που θα εισαγάγει ο παίκτης, έτσι ώστε να μπορούμε αργότερα να περάσουμε στην κλάση το custom property του

παίκτη. Στην παρακάτω εικόνα φαίνεται η κλάση Network Controller.

```
using Photon.Pun;
using UnityEngine;
using UnityEngine.UI;

Unity Script | 0 references
public class networkContoller : MonoBehaviourPunCallbacks
{
    [SerializeField] private InputField UserNameInput;
    [SerializeField] private GameObject PlayButton;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        PhotonNetwork.ConnectUsingSettings(); //connect on server
    }

    13 references
    public override void OnConnectedToMaster() // on connect to server do something
    {
        if (!PhotonNetwork.InLobby)
        {
            PhotonNetwork.JoinLobby();
        }
        base.OnConnectedToMaster();
        Debug.Log("We are now live at" + PhotonNetwork.CloudRegion + "server!");
    }

    0 references
    public void ChangeUserNameInput() //players username on label
    {
        if(UserNameInput.text.Length >= 3)
        {
            PlayButton.SetActive(true);
        }
        else
        {
            PlayButton.SetActive(false);
        }
    }

    0 references
    public void SetUserName() //set players nickname on servers
    {
        PhotonNetwork.NickName = UserNameInput.text;
    }
}
```

QuickStartLobbyController.cs

Σε αυτήν την κλάση δημιουργούμε ένα δωμάτιο με την μέθοδο CreateRoom και ορίζουμε πόσοι παίκτες θέλουμε να μπουν στο δωμάτιο. Φυσικά από την στιγμή που το παιχνίδι είναι fighting game θέλουμε να συνδεθούν μόνο δυο παίκτες. Επίσης όπως φαίνεται στην παρακάτω εικόνα δημιουργούμε ένα serialized field(το Serialized field μας δίνει την δυνατότητα να επεξεργαστούμε την τιμή της μεταβλητής από τον επεξεργαστή του Unity. Αν για παράδειγμα θέλαμε να περάσουμε αυτήν την τιμή σε άλλες κλάσης θα ορίζαμε την μεταβλητή ως Public). Η δημιουργία του serialized field μας βοηθά ώστε να διαχειριστούμε από τον επεξεργαστή του Unity την τιμή της μεταβλητής roomSize. Επίσης καλύπτουμε και τα ενδεχόμενα πρώτον της μη σωστής δημιουργίας του δωματίου με την μέθοδο OnCreateRoomFailed και δεύτερον την

περίπτωση δημιουργίας κάποιου σφάλματος με την μέθοδο OnJoinRoomFailed κατά την είσοδο του χρήστη.

```
using Photon.Pun;
using Photon.Realtime;
using UnityEngine;

Unity Script | 0 references
public class QuickStartLobbyContoller : MonoBehaviourPunCallbacks
{
    [SerializeField]
    private GameObject playButton;
    [SerializeField]
    private int roomSize;

    13 references
    public override void OnConnectedToMaster() //sync scenes for both players
    {
        PhotonNetwork.AutomaticallySyncScene = true;


    }
    0 references
    public void QuickStart() //connect to room
    {
        PhotonNetwork.JoinRandomRoom();
        Debug.Log("START");
    }

    12 references
    public override void OnJoinRandomFailed(short returnCode, string message) // if player failed to join the room
    {
        Debug.Log("failed to join a room");
        CreateRoom();
    }
    2 references
    void CreateRoom() // create room
    {
        Debug.Log("Creating room now");
        int randomNubmer = Random.Range(0, 10000);
        RoomOptions roomsOps = new RoomOptions() { IsVisible = true, IsOpen = true, MaxPlayers = (byte)roomSize };
        PhotonNetwork.CreateRoom("Room" + randomNubmer, roomsOps);
        Debug.Log(randomNubmer);
    }
    11 references
    public override void OnCreateRoomFailed(short returnCode, string message) // if create room failed
    {
        Debug.Log("Failed to create room...try again");
        CreateRoom();
    }
}

```

DelayRoom.cs

Την κλάση DelayRoom την καλούμε έχοντας σκοπό όταν δημιουργείτε το δωμάτιο να μένει ανοιχτό μέχρι την σκηνή που θέλουμε εμείς. Διαφορετικά αν δεν υπήρχε η DelayRoom κατά την εκκίνηση του παιχνιδιού και της σύνδεσης των παιχτών δεν θα υπήρχε πλέον το δωμάτιο στην επόμενη σκηνή. Με αυτόν τον τρόπο λοιπόν μέσω της συγκεκριμένης συνάρτησης διατηρούμε το δωμάτιο ενεργό. Για να επιτευχθεί αυτό έγινε η κλάση singleton ώστε και οι δυο παίκτες να έχουν τον ίδιο κώδικα και με την βοήθεια της μέθοδος OnJoinedRoom μένει ανοιχτό το δωμάτιο μέχρι και την επόμενη σκηνή. Οι μέθοδοι OnEnable, OnDisable είναι του Photon. Η OnEnable καλείται όταν κάποιος παίκτης εισέλθει για πρώτη φορά στο παιχνίδι συνεπώς αποτελεί τον Master Client, δηλαδή είναι ο δημιουργός του δωμάτιου. Για τον επόμενο παίκτη που θα εισέλθει στο παιχνίδι, ο οποίος θα είναι ένας απλός client, η OnEnable δεν ενεργοποιείται. Και τέλος η OnDisable καλείται όταν σταματήσει ο κώδικα για

οποιοδήποτε λόγω να εκτελείται.

```
using Photon.Pun;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script | 6 references
public class DelayRoomContoller : MonoBehaviourPunCallbacks
{
    public static DelayRoomContoller room;
    [SerializeField]
    private int waitingRoomSceneIndex; // number scene until object destroy
    Unity Message | 0 references
    private void Awake() // Creating Singleton on Await
    {
        if(DelayRoomContoller.room == null)
        {
            DelayRoomContoller.room = this;
        }
        else
        {
            if (DelayRoomContoller.room != this)
            {
                Destroy(DelayRoomContoller.room.gameObject);
                DelayRoomContoller.room = this;
            }
        }
        DontDestroyOnLoad(this.gameObject);
    }
    Unity Message | 9 references
    public override void OnEnable() // enable script
    {
        PhotonNetwork.AddCallbackTarget(this);
    }

    Unity Message | 11 references
    public override void OnDisable() // disable script
    {
        PhotonNetwork.RemoveCallbackTarget(this);
    }

    19 references
    public override void OnJoinedRoom()
    {
        SceneManager.LoadScene(waitingRoomSceneIndex);
    }
}
```

DelayStartWaitingRoomController.cs

Αυτή η κλάση έχει βαρύνουσα σημασία για διαχείριση των παικτών στο παιχνίδι. Η παρούσα κλάση ενεργοποιείται όταν εισέρχεται ένας παίκτης στην δεύτερη σκηνή.(βλέπε εικόνα 5). Επίσης περιέχει κάποιες μεθόδους. Αρχικά εισάγουμε όλους του χρήστες με όλες τις πληροφορίες τους, δηλαδή το id, το nickname κ.λπ. σε μια λίστα, με την βοήθεια τις μεθόδου AddPlayerListing. Έπειτα με την OnPlayerLeftRoom διαχειριζόμαστε την περίπτωση που ο παίκτης φύγει από το δωμάτιο. Με την OnDisconected αν ένας παίκτης θελήσει να πατήσει το κουμπί leave(βλέπε εικόνα 5) ώστε να φύγει από το δωμάτιο τον επιστρέφουμε στην αρχική οθόνη με την βοήθεια της εντολής LoadLevel. Στην κλάση αυτή διαχειριζόμαστε την αρίθμηση των παικτών και την εκκίνηση του χρόνου όταν στο δωμάτιο εισέλθουν

δυο παίκτες. Στις παρακάτω εικόνες είναι η κλάση DelayStartWaitingRoomController.

```
using Photon.Pun;
using Photon.Realtime;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using UnityEngine;
using System.Collections.Generic;
using TMPro;
using System.Collections;

Unity Script | 0 references
public class DelayStartWaitingRoomContoller : MonoBehaviourPunCallbacks,IInRoomCallbacks
{

    private ExitGames.Client.Photon.Hashtable myCustomProperties = new ExitGames.Client.Photon.Hashtable();
    private PhotonView myPhotonView;
    //scene navigation indexes
    [SerializeField]
    private int multiplayerSceneIndex;
    [SerializeField]
    private int menuSceneIndex;
    //number of players in the room out of the total room size
    private int playerCount;
    private int roomSize;
    [SerializeField]
    private int minPlayersToStart;
    //text variables
    [SerializeField]
    private Text roomCountDisplay;
    [SerializeField]
    private Text timerToStartDisplay;
    //bool values for if the timer count down
    private bool readyToCountDown;
    private bool readyToStart;
    private bool startingGame;
    //countdown timer variables
    private float timerTostartGame;
    private float notFullGameTimer;
    private float fullGameTimer;
    //countdown timer reset variables
    [SerializeField]
    private float maxWaitTime;
```

```
    [SerializeField]
    private float maxFullGameWaitTime;
    [SerializeField]
    private Transform content;
    [SerializeField]
    private PlayerListing playerListing;
    [SerializeField]
    private TextMeshProUGUI versus;

    private List<PlayerListing> playerListings = new List<PlayerListing>();
    Unity Message | 0 references
    private void Awake()
    {
        GetPlayersOnRoom();
    }

    1 reference
    private void GetPlayersOnRoom()
    {
        foreach (KeyValuePair<int, Player> playerInfo in PhotonNetwork.CurrentRoom.Players)
        {
            AddPlayerListing(playerInfo.Value);
        }
    }
    2 references
    private void AddPlayerListing(Player player)
    {
        PlayerListing listing = Instantiate(playerListing, content);
        if (listing != null)
        {
            myCustomProperties["Wins"] = 0;
            player.SetCustomProperties(myCustomProperties);
            listing.SetPlayerInfo(player);
            playerListings.Add(listing);
        }
    }

```

```
74
75      // Start is called before the first frame update
        Unity Message | 0 references
76      void Start()
77      {
78          myPhotonView = GetComponent<PhotonView>();
79          fullGameTimer = maxFullGameWaitTime;
80          notFullGameTimer = maxWaitTime;
81          timerTostartGame = maxWaitTime;
82
83          PlayerCountUpdate();
84      }
85
        3 references
86      void PlayerCountUpdate()
87      {
88          playerCount = PhotonNetwork.PlayerList.Length;
89          roomSize = PhotonNetwork.CurrentRoom.MaxPlayers;
90          roomCountDisplay.text = playerCount + ":" + roomSize;
91
92          if(playerCount== roomSize)
93          {
94              readyToStart = true;
95          }
96          else if (playerCount >= minPlayersToStart)
97          {
98              readyToCountDown = true;
99          }
100         else
101         {
102             readyToCountDown = false;
103             readyToStart = false;
104         }
105     }
        10 references
```

```
        10 references
106     public override void OnPlayerEnteredRoom(Player newPlayer)
107     {
108         AddPlayerListing(newPlayer);
109         PlayerCountUpdate();
110
111         if (PhotonNetwork.IsMasterClient)
112         {
113             myPhotonView.RPC("RPC_SendTimer", RpcTarget.Others, timerTostartGame);
114         }
115     }
116     [PunRPC]
        0 references
117     private void RPC_SendTimer(float timeIn)
118     {
119         timerTostartGame = timeIn;
120         notFullGameTimer = timeIn;
121         if(timeIn < fullGameTimer)
122         {
123             fullGameTimer = timeIn;
124         }
125     }
126
        13 references
127     public override void OnPlayerLeftRoom(Player otherPlayer)
128     {
129         int index = playerListings.FindIndex(x => x.Player == otherPlayer);
130         if(index != -1)
131         {
132             Destroy(playerListings[index].gameObject);
133             playerListings.RemoveAt(index);
134         }
135         base.OnPlayerLeftRoom(otherPlayer);
136         PlayerCountUpdate();
137     }
138
139     // Update is called once per frame
        Unity Message | 0 references
140     private void Update()
141     {
142         waitingForMorePlayers();
143     }
```

```
        1 reference
144     void waitingForMorePlayers()
145     {
146         if (PhotonNetwork.IsConnected)
147         {
148             if (playerCount <= 1)
149             {
150                 ResetTimer();
151                 versus.text = "Waiting For Players";
152             }
153             if (readyToStart)
154             {
155                 fullGameTimer -= Time.deltaTime;
156                 timerTostartGame = fullGameTimer;
157                 versus.text = PhotonNetwork.PlayerList[0].NickName + " " + "Versus" + " " + PhotonNetwork.PlayerList[1].NickName;
158             }
159             else if (readyToCountDown)
160             {
161                 notFullGameTimer -= Time.deltaTime;
162                 timerTostartGame = notFullGameTimer;
163             }
164             string tempTimer = string.Format("{0:00}", timerTostartGame);
165             timerToStartDisplay.text = tempTimer;
166             if (timerTostartGame <= 0f)
167             {
168                 if (startingGame)
169                 {
170                     return;
171                 }
172                 StartGame();
173             }
174         }
175     }
        1 reference
176     void ResetTimer()
177     {
178         timerTostartGame = maxWaitTime;
179         notFullGameTimer = maxWaitTime;
180         fullGameTimer = maxFullGameWaitTime;
181     }
```

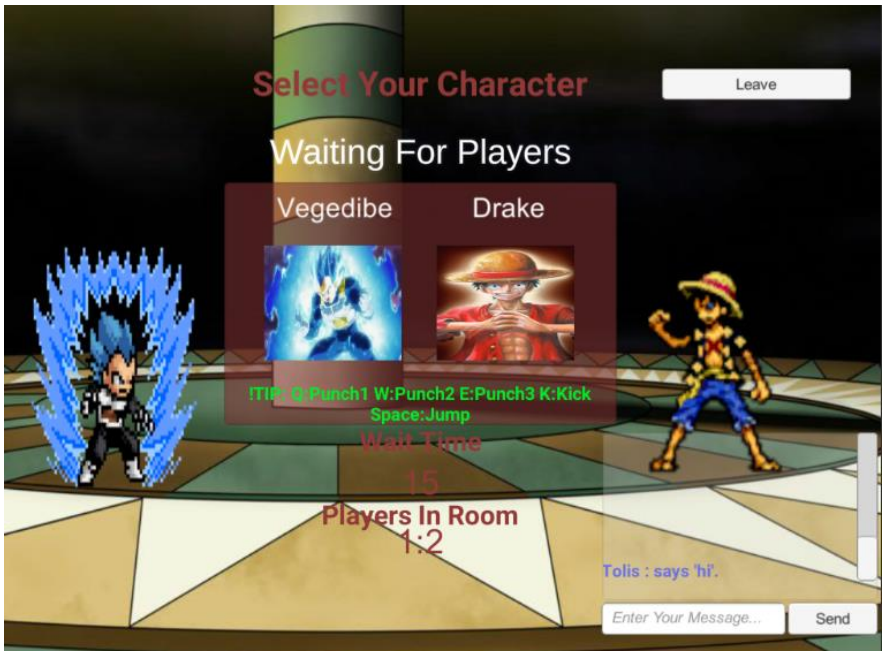
```
182     public void StartGame()
183     {
184         startingGame = true;
185         if (!PhotonNetwork.IsMasterClient)
186         {
187             return;
188         }
189         PhotonNetwork.CurrentRoom.IsOpen = false;
190         PhotonNetwork.LoadLevel(multiplayerSceneIndex);
191     }
192
            0 references
193     public void LeaveRoom()
194     {
195         PhotonNetwork.Disconnect();
196
197     }
            16 references
198     public override void OnDisconnected(DisconnectCause cause)
199     {
200         PhotonNetwork.LoadLevel(0);
201         base.OnDisconnected(cause);
202     }
203 }
```

4.2 Ανάλυση παιχνιδιού

Το παιχνίδι όπως προαναφέρθηκε αποτελείται από τρεις σκηνές. Στην πρώτη σκηνή ο παίκτης εισάγει το username που θέλει και μπορεί να πατήσει το κουμπί Play που του εμφανίζετε.

Έπειτα εισέρχεται στην δεύτερη σκηνή οπού περιμένει έναν δεύτερο παίκτη να συνδεθεί. Ο παίκτης μπορεί να επιλέξει έναν από τους δυο χαρακτήρες που υπάρχουν

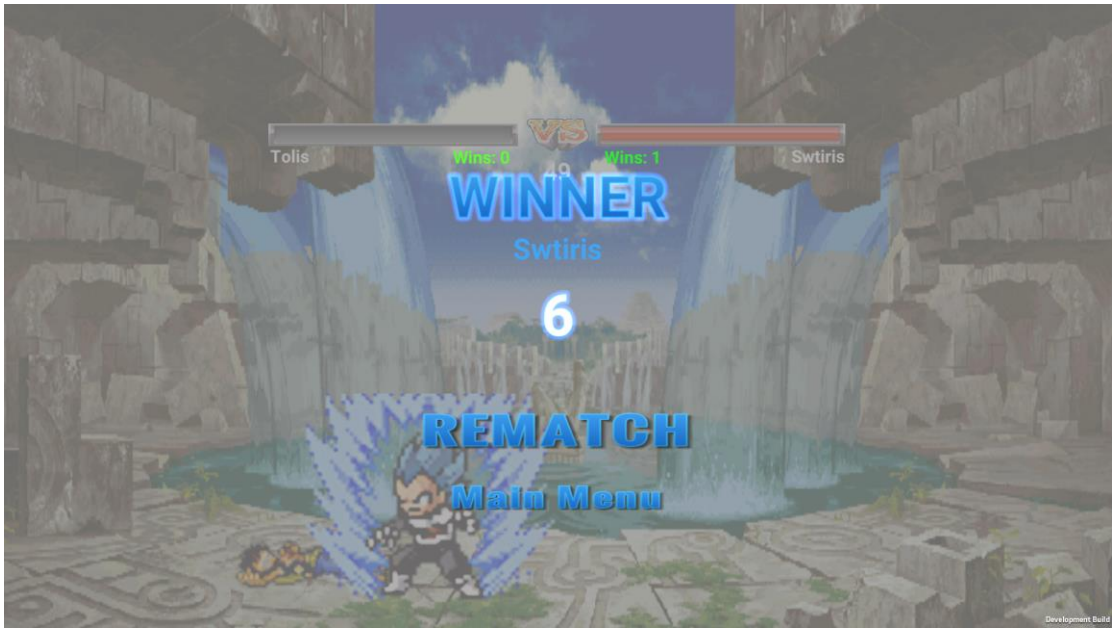
στο παιχνίδι. Όταν ο παίκτης επιλέξει τον χαρακτήρα που θέλει τότε το ανάλογο εικονίδιο πρασινίζει.



Όταν ο δεύτερος παίκτης συνδεθεί ξεκινάει ο χρόνος αντίστροφα. Στο μεσοδιάστημα να μπορούν οι παίχτες να επικοινωνήσουν μεταξύ τους μέσω του Photon chat όπως φαίνεται στην παρακάτω εικόνα στο πράσινο τετραγωνάκι.

Όταν ο χρόνος τελειώσει τότε και οι δυο παίκτες εισέρχονται στην τελευταία σκηνή οπού γίνεται η μονομαχία όπως φαίνεται στην παρακάτω εικόνα.

Στο τέλος της μονομαχίας εμφανίζετε το όνομα του νικητή. Έχουν και οι δυο παίκτες την επιλογή να μονομαχήσουν ξανά πατώντας το κουμπί Rematch ή να φύγουν. Οι παίκτες έχουν δέκα δευτερόλεπτα να πάρουν την απόφαση αυτή . Στην περίπτωση που δεν επιλέξουν τίποτα επιστρέφουν στην αρχική οθόνη του παιχνιδιού.



Game Controller

Ο Game Controller είναι η πιο σημαντική κλάση του παιχνιδιού. Μέσα από αυτήν την κλάση διαχειριζόμαστε τις νίκες των παικτών, το τέλος του παιχνιδιού, και τους χρόνους που υπάρχουν στην σκηνή της μάχης καθώς επίσης και την απόφαση των παικτών για επανάληψη τις μάχης ή την επιστροφή τους στην αρχική οθόνη. Παρακάτω θα γίνει αναλύσει του Game Controller.

Αρχικά θέλουμε ο Game Controller να είναι ένας και για τους δυο παίκτες, οπότε τον μετατρέπουμε σε singleton

```
Unity Message | 9 references
public override void OnEnable()
{
    if (GameController.gameController == null)
    {
        GameController.gameController = this;
    }

    BeginRematch(false);
    base.OnEnable();
}

Unity Message | 11 references
public override void OnDisable()
{
    Destroy(this.gameObject);
    base.OnDisable();
}
```

Όταν το παιχνίδι είναι έτοιμο να ξεκινήσει ο Game Controller επιλεγεί ένα τυχαίο χάρτη και αναζητάει τις νίκες των παιχτών. Στην αρχή οι παίκτες δεν έχουν νίκες οπότε περνάει την τιμή μηδέν.

```
void Start()
{
    StartingTimer = true;
    if (PhotonNetwork.IsMasterClient)
    {
        random = Random.Range(0, randomMap.Length);
        randomBackground(random); // set random background
    }
    won = true; // reset who won
    GetScoreLeft(PhotonNetwork.PlayerList[0]); //Get Left Players Score
    GetScoreRight(PhotonNetwork.PlayerList[1]); //Get Right Players Score
    isGameFinished = true; //reset game
    currentGameTimer = defaultGametimer; // set current time
    StopSound = true; //reset sound
}
```

Για να τελειώσει η μάχη πρέπει κάποιος από τους παρακάτω ελέγχους να ισχύσει. Οι έλεγχοι αυτοί γίνονται ανα δευτερόλεπτο καθώς η update τρέχει κάθε δευτερόλεπτο σε πραγματικό χρόνο. Στην εικόνα 19 ο κώδικας είναι για την περίπτωση που τελειώσει ο χρόνος. Με άλλα λόγια ορίζει ποιος θα είναι ο νικητής όταν τελειώσει ο χρόνος με βάση τους πόντους ζωής του κάθε παίχτη. Ενώ στην εικόνα 20 εμανίζεται κώδικας για την περίπτωση που κάποιος χαρακτήρας ηττηθεί , δηλαδή η ζωή του μηδενιστεί.

```
if(currentGameTimer <= 0)
{
    isGameFinished = true;
    if (PlayerHealth > EnemyHealth)
    {
        TimeToStartTextObj.SetActive(true);
        isEnemyDead = true;
        isPlayerWin = true;
        RematchMenu.SetActive(true);
    }
    else if (PlayerHealth < EnemyHealth)
    {
        TimeToStartTextObj.SetActive(true);
        isPlayerDead = true;
        isEnemyWin = true;
        RematchMenu.SetActive(true);
    }
    else if (PlayerHealth == EnemyHealth)
    {
        TimeToStartTextObj.SetActive(true);
        TextForDraw.text = "DRAW";
        isDraw = true;
        RematchMenu.SetActive(true);
    }
}
```

Εικονα 19: Προϋποθέσεις για το τέλος της μονομανίας στο τέλος του χρόνου.

```
else if (isEnemyDead)
{
    won = true;
    if (PlayerHealth >= 1f)
    {
        TimeToStartTextObj.SetActive(true);
        timerToStartText.text = "PERFECT!";
    }
    TimeToStartTextObj.SetActive(true);
    isGameFinished = true;
    isPlayerWin = true;
    RematchMenu.SetActive(true);
}
else if (isPlayerDead)
{
    won = true;
    if (EnemyHealth >= 1f)
    {
        TimeToStartTextObj.SetActive(true);
        timerToStartText.text = "PERFECT!";
    }
    TimeToStartTextObj.SetActive(true);
    isGameFinished = true;
    isEnemyWin = true;
    RematchMenu.SetActive(true);
}
```

Εικονα 20: Προϋποθέσεις για το τέλος της μονομανίας όταν ενας από τους δυο παικτες ηττηθει.

Η παρακάτω μέθοδος καλείτε για την επιλογή του τυχαίου χάρτη.

```
void randomBackground(int random)
{
    if(random == 0)
    {
        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "CyberpunkStreet"),new Vector3(-6.815f,3.12f,0),Quaternion.identity,0);
        if (PhotonNetwork.IsMasterClient)
        {
            base.GetComponent<PhotonView>().RPC("RPC_PlayBGM1", RpcTarget.AllBuffered);
        }
    }
    if (random == 1)
    {
        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "NightForest"), new Vector3(-7.202f, 1.1308f,0), Quaternion.identity, 0);
        if (PhotonNetwork.IsMasterClient)
        {
            base.GetComponent<PhotonView>().RPC("RPC_PlayBGM2", RpcTarget.AllBuffered);
        }
    }
    if (random == 2)
    {
        PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "Background"), new Vector3(-6.911f, 3.36f,0), Quaternion.identity, 0);
        if (PhotonNetwork.IsMasterClient)
        {
            base.GetComponent<PhotonView>().RPC("RPC_PlayBGM3", RpcTarget.AllBuffered);
        }
    }
}
```

Στην μέθοδο randomBackground επίσης γίνεται και ένα RPC για την μουσική που παίζετε κατά την διάρκεια της μάχης. Ο κάθε χάρτης έχει την δική του μουσική.

```
[PunRPC]
0 references
public void RPC_PlayBGM1()
{
    AudioController.audioController.StopAllSound();
    AudioController.audioController.PlayBGMSound("BGMForPunkStreet");
}
[PunRPC]
0 references
public void RPC_PlayBGM2()
{
    AudioController.audioController.StopAllSound();
    AudioController.audioController.PlayBGMSound("BGMForForest");

}
[PunRPC]
0 references
public void RPC_PlayBGM3()
{
    AudioController.audioController.StopAllSound();
    AudioController.audioController.PlayBGMSound("BGMForBackground");

}
```

Οι μέθοδοι GetScoreLeft και GetScoreRight ελέγχουν αν περιέχουν οι παίκτες Custom Properties με την ονομασία wins.

```
void GetScoreLeft(Player targetPlayer)
{
    if (targetPlayer.CustomProperties.ContainsKey("Wins"))
    {
        leftPlayerWins.text = "Wins:"+ " " + targetPlayer.CustomProperties["Wins"];
    }
}
2 references
void GetScoreRight(Player targetPlayer)
{
    if (targetPlayer.CustomProperties.ContainsKey("Wins"))
    {
        rightPlayerWins.text = "Wins:" +" " + targetPlayer.CustomProperties["Wins"];
    }
}
```

Οι παρακάτω μέθοδοι είναι για την προσθήκη της νίκης στον παίκτη που κατόρθωσε να την πραγματοποιήσει.

```
public void AddWinLeft(Player player, int wins)
{
    if (won)
    {
        int current = (int)player.CustomProperties["Wins"];
        current = current + wins;
        Hashtable Score = new Hashtable();
        Score["Wins"] = current;
        player.SetCustomProperties(Score);
        won = false;
    }

}
1 reference
public void AddWinRight(Player player, int wins)
{
    if (won)
    {
        int current = (int)player.CustomProperties["Wins"];
        current = current + wins;
        Hashtable Score = new Hashtable();
        Score["Wins"] = current;
        player.SetCustomProperties(Score);
        won = false;
    }
}
```

Συνδυασμός κινήσεων(Combo)

Ο κάθε χαρακτήρας μπορεί να κάνει ένα combo αρκεί ο παίκτης να πατήσει τα καταλληλά κουμπιά στον σωστό χρόνο. Αν δεν πατήσει τα σωστά κουμπιά τότε το combo σταματάει. Τα combo χρειάζονται ώστε να μπορέσει ο παίκτης να αφαιρέσει

περισσοτέρους πόντους ζωής μέσω αυτού παρά με ένα απλό χτύπημα. Κατά την διάρκεια του combo ο αντίπαλος παίκτης δεν μπορεί να αμυνθεί με αποτέλεσμα να χάνει αρκετούς πόντους ζωής. Κάθε φορά που ο παίκτης πατάει ένα από τα κουμπιά των μπουνιών είτε το κουμπί της κλωτσιάς τότε μπαίνει σε μια κατάσταση combo. Παρακάτω φαίνεται ο κώδικας με τις καταστάσεις των combo.

```
void comboAttack()
{
    if (GameController.gameController.isGameFinished)
        return;

    if (Input.GetKeyDown(KeyCode.Q))
    {
        if (CurrentComboState == ComboState.Punch3 || CurrentComboState == ComboState.kick || Input.GetKey("right") || Input.GetKey("left") || Input.GetKeyDown("space") || Input.GetKeyDown("down"))
        {
            return;
        }

        CurrentComboState++;
        activateTimeToReset = true;
        currentComboTimer = defaultComboTimer;

        if (CurrentComboState == ComboState.Punch1)
        {
            AudioController.audioController.PlaySound("Punch");
            animator.SetTrigger("punch1");
        }
    }
```

```
    if (Input.GetKeyDown(KeyCode.W))
    {
        if (GameController.gameController.isGameFinished)
            return;
        if (CurrentComboState == ComboState.Punch3 || CurrentComboState == ComboState.kick || Input.GetKey("right") || Input.GetKey("left") || Input.GetKeyDown("space") || Input.GetKeyDown("down"))
        {
            return;
        }
        if (CurrentComboState == ComboState.None ||
            CurrentComboState == ComboState.Punch1 ||
            CurrentComboState == ComboState.kick)
        {
            CurrentComboState = ComboState.Punch2;
        }
        activateTimeToReset = true;
        currentComboTimer = defaultComboTimer;

        if (CurrentComboState == ComboState.Punch2)
        {
            AudioController.audioController.PlaySound("Punch");
            animator.SetTrigger("punch2");
        }
    }
```

```
if (Input.GetKeyDown(KeyCode.E))
{
    if (GameController.gameController.isGameFinished)
        return;
    if (CurrentComboState == ComboState.Punch3 || CurrentComboState == ComboState.kick || Input.GetKey("right") || Input.GetKey("left") || Input.GetKeyDown("space") || Input.GetKeyDown("down"))
    {
        return;
    }
    if (CurrentComboState == ComboState.None ||
        CurrentComboState == ComboState.Punch1 ||
        CurrentComboState == ComboState.Punch2 ||
        CurrentComboState == ComboState.kick)
    {
        CurrentComboState = ComboState.Punch3;
    }
    activateTimeToReset = true;
    currentComboTimer = defaultComboTimer;

    if (CurrentComboState == ComboState.Punch3)
    {
        AudioController.audioController.PlaySound("Punch");
        animator.SetTrigger("punch3");
    }
}
```

```
if (Input.GetKeyDown(KeyCode.R))
{
    if (GameController.gameController.isGameFinished)
        return;
    if (CurrentComboState == ComboState.Punch3 || CurrentComboState == ComboState.kick || Input.GetKey("right") || Input.GetKey("left") || Input.GetKeyDown("space") || Input.GetKeyDown("down"))
    {
        return;
    }
    if(CurrentComboState == ComboState.None ||
        CurrentComboState == ComboState.Punch1 ||
        CurrentComboState == ComboState.Punch2 ||
        CurrentComboState == ComboState.Punch3)
    {
        CurrentComboState = ComboState.kick;
    }
    activateTimeToReset = true;
    currentComboTimer = defaultComboTimer;

    if(CurrentComboState == ComboState.kick)
    {
        AudioController.audioController.PlaySound("Kick");
        animator.SetTrigger("kick");
    }
}
```

Η παρακάτω μέθοδος είναι σχεδιασμένη ώστε όταν ο παίκτης πατήσει είτε κάποιο λάθος κουμπί είτε δεν το πατήσει στο σωστό χρόνο να γίνετε reset το combo,δηλαδή να μηδενίζετε η κατάσταση του combo που βρίσκετε και να ξεκινάει από την αρχή.

```
154     void ResetComboState()
155     {
156         if (activateTimeToReset)
157         {
158             currentComboTimer -= Time.deltaTime;
159 
160             if(currentComboTimer <= 0f)
161             {
162                 CurrentComboState = ComboState.None;
163                 activateTimeToReset = false;
164                 currentComboTimer = defaultComboTimer;
165             }
166         }
167     }
168 
```

Χαρακτήρες

Κίνηση χαρακτήρων

Οι χαρακτήρες μπορούν να επιτεθούν, να αμυνθούν, να κινηθούν, να κάνουν άλμα και διπλό άλμα

Ο κάθε χαρακτήρας έχει μια κλάση που ονομάζετε PlayerController και αυτή είναι η κλάση που ευθύνεται για την κίνηση των χαρακτήρων. Παρακάτω θα δούμε και θα αναλύσουμε την κλάση PlayerController.

Για να ξεκινήσει ο χαρακτήρας να κινείτε γίνετε έλεγχος με την βοήθεια της μεθόδου update, αν ο χαρακτήρας είναι νεκρός ή βρίσκετε στο πάτωμα της πίστας.

```
void Update()
{
    if (PV.IsMine && isGroundedCheck())
    {
        DeadChecker();
        rightSide = false;
        FacingToTarget();
        DoubleJump = true;
        basicMovementToRight();
    }
    if (!PV.IsMine)
    {
        return;
    }
```

Μέθοδος DeadCheker

Στην μέθοδο DeadCheker θέλαμε όταν η ζωή του παίκτη φτάσει στο μηδέν να γίνονται οι απαραίτητη έλεγχοι για το ποιος παίκτης νίκησε, αν ο παίκτης είναι νεκρός ή αν το παιχνίδι τέλειωσε με ισοπαλία.

```
void DeadChecker()
{
        //Check if is game finished and if this player won
    if (GameController.gameController.isGameFinished && GameController.gameController.isPlayerWin)
    {
        animator.SetBool("win", true);
        PV.RPC("RPC_WinnersName", RpcTarget.AllBuffered);
        GameController.gameController.AddWinLeft(PhotonNetwork.PlayerList[0], 1);
        return;
    }

    //Check if is game finished and if this player is dead
    if (GameController.gameController.isPlayerDead)
    {
        isDead = true;
        animator.SetBool("dead", true);
        animator.SetBool("win", false);
        GameController.gameController.isPlayerDead = isDead;

    }
    //Check if is game finished and its a draw
    if (GameController.gameController.isDraw)
    {
        GameController.gameController.isDraw = true;
    }
}
```

Μέθοδος isGroundedCheck

Με την μέθοδο isGroundedCheck θέλαμε να γίνεται ο έλεγχος για το αν ο παίκτης βρίσκετε στο πάτωμα έτσι ώστε να απενεργοποιούμε την κατάσταση του άλματος του animator.

```
private bool isGroundedCheck()
{
    float extraHeightText = 0.1f;
    RaycastHit2D raycastHit = Physics2D.BoxCast(capsule2d.bounds.center,
        capsule2d.bounds.size, 0f, Vector2.down, extraHeightText, GroundLayer);
    Color rayColor;
    if (raycastHit.collider != null)
    {
        animator.SetBool("isJumping", false);
        rayColor = Color.green;

    }
    else
    {
        animator.SetBool("isJumping", true);
        rayColor = Color.red;

    }
    return raycastHit.collider != null;
}
```

Μέθοδος basicMovementToRight Or basicMovementToLeft

Ο κάθε χαρακτήρας ανάλογα την πλευρά που βρίσκετε έχει την αντίστοιχη basic Movement δηλαδή ο χαρακτήρας που βρίσκετε στην αριστερά πλευρά έχει την μέθοδο basicMovementToLeft. Για να επιτευχθεί αυτό έχουμε δημιουργήσει τον χαρακτήρα δυο φορές, για παράδειγμα έχουμε τον Vegeta Left prefab και τον Vegeta Right prefab, οπότε αν ο παίκτης βρίσκεται στην δεξιά πλευρά έχει τον έλεγχο του χαρακτήρα Vegeta Left. Αυτό γίνεται καθώς θέλαμε να έχουμε καλύτερο έλεγχο των physics από οποιαδήποτε πλευρά και να βρίσκεται ο παίκτης. Επίσης οι κλάσης που χρησιμοποιούνται είναι παρόμοιες και για τους δυο χαρακτήρες. Η διαφορά βρίσκεται στις ονομασίες και στην κίνηση μπρος και πίσω. Ο παραπάνω έλεγχος επιτυγχάνετε με τις κλάσης PhotonPlayer, Avatarsetup. Με την κλάση Photon Player ορίζουμε για τον παίκτη και σε ποια πλευρά θα βρίσκεται, δηλαδή αν ο παίκτης είναι ο master client του δίνουμε την αριστερή πλευρά με την left Flag μεταβλητή . Αντίθετα στον client δίνουμε την δεξιά με την right Flag μεταβλητή. Στην κλάση Avatar setup δεχόμαστε το flag δηλαδή το left Flag ή το right Flag και δίνουμε στους παίκτες τους αντίστοιχους χαρακτήρες για την πλευρά στην οποία βρίσκονται.

```
void Start()
{
    PV = GetComponent<PhotonView>();
    if (PV.IsMine && PhotonNetwork.IsMasterClient)
    {
        Debug.Log("eimai o master client");
        Debug.Log(spawnPicker);
        myAvatar = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PlayerAvatar"),
            GameSetupContoller.GS.spawnPoints[0].position, GameSetupContoller.GS.spawnPoints[0].rotation, 0) as GameObject;
        leftFlag = true;
    }else if (!PV.IsMine)
    {
        return;
    }
    if (PV.IsMine && !PhotonNetwork.IsMasterClient)
        {
        Debug.Log("DEN eimai o master client");
        Debug.Log(spawnPicker);
        myAvatar = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PlayerAvatar"),
            GameSetupContoller.GS.spawnPoints[1].position, GameSetupContoller.GS.spawnPoints[1].rotation, 0) as GameObject;
        rightFlag = true;
    }
    else if (!PV.IsMine)
    {
        return;
    }
}
```

Εικόνα 21: Κλάση Photon Player

```
void Start()
{
    PV = GetComponent<PhotonView>();
    if (PV.IsMine)
    {
        if (PlayerInfo.PI.mySelectedCharacter == 0 && PhotonPlayer.PP.rightFlag==true)
        {
            Debug.Log("vegeta master client");
            myCharacter = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "VegetaRight"), transform.position, transform.rotation, 0);
        }
        else if (PlayerInfo.PI.mySelectedCharacter == 0 && PhotonPlayer.PP.leftFlag == true)
        {
            myCharacter = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "VegetaLeft"), transform.position, transform.rotation, 0);
        }
        else if (PlayerInfo.PI.mySelectedCharacter == 1 && PhotonPlayer.PP.leftFlag == true)
        {
            myCharacter = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "luffyLeft"), transform.position, transform.rotation, 0);
        }
        else if (PlayerInfo.PI.mySelectedCharacter == 1 && PhotonPlayer.PP.rightFlag == true)
        {
            myCharacter = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "luffyRight"), transform.position, transform.rotation, 0);
        }
    }
}
```

Εικόνα 22: Κλάση Avatar Setup

Αφού επεξηγήθηκε ο τρόπος που ορίζετε σε ποια πλευρά θα βρίσκετε ο κάθε παίκτης και ο έλεγχος του χαρακτήρα που θα έχει την ανάλογη την πλευρά, θα αναλύσουμε και την μέθοδο της κίνησης. Θέλαμε στον παίκτη να δώσουμε την δυνατότατά του dash δηλαδή να μπορεί ο παίκτης πατώντας δυο φορές το ίδιο πλήκτρο σε συγκεκριμένο χρόνο να επιταχύνετε για ένα χρονικό διάστημα. Αυτό το επιτυγχάνουμε χρησιμοποιώντας τις παρακάτω μεταβλητές που βρίσκονται στην κλάση του PlayerController.

```
public float ForwardVel = 5f; //speed for forward movement
public float BackVel = 3f; //speed for backward movement
private int RTotal = 0; //how many times pressed forward button
private float RtimeDelay = 0.0f; //time delay for forward button to be pressed twice
private int LTotal = 0; //how many times pressed backward button
private float LtimeDelay = 0.0f; //time delay for backward button to be pressed twice
private int xMovement = 0; //total speed
private float DashDuration = 0.0f; //time duration for dash
```

Με την μεταβλητή ForwardVel και BackVel δίνουμε την ταχύτητα του παίκτη ανάλογα με την κατεύθυνσή του dash. Με τις μεταβλητές RTotal και LTotal κρατάμε τις φορές που πάτησε ο χρήστης το ίδιο πλήκτρο για την κίνηση. Έπειτα με την RtimeDelay και με την LtimeDelay ελέγχουμε τον χρόνο που πάτησε το ίδιο πλήκτρο ο παίκτης, ώστε αν ο παίκτης αργήσει να ξαναπατήσει το κουμπί να μην ξεκινήσει το dash. Τέλος έχουμε την xMovement που είναι η συνολική ταχύτητα του παίκτη και την DashDuration με την οποία συμβολίζεται ο χρόνος στον οποίο ο παίκτης θα έχει την αυξημένη ταχύτητα. Οι παρακάτω εικόνες είναι κώδικας της basicMovementToRight

```
123     void basicMovementToRight()
124         {
125         if (GameController.gameController.isGameFinished)
126             return;
127         // Forward Movement
128         if (isDefence == false && !Input.GetKeyDown("down"))
129         {
130             if (Input.GetKey("right"))
131             {
132                 rgb2d.velocity = new Vector2(ForwardVel + xMovement, rgb2d.velocity.y);
133                 animator.SetFloat("Speed", ForwardVel);
134             }
135
136             if (Input.GetKeyDown("right"))
137             {
138                 RTotal = RTotal + 1;
139             }
140
141             if (Input.GetKeyUp("right"))
142             {
143                 xMovement = 0;
144                 rgb2d.velocity = new Vector2(0, 0);
145                 animator.SetFloat("Speed", 0);
146             }
147
```

```
148             if ((RTotal == 1) && (RtimeDelay < .5))
149                 RtimeDelay = RtimeDelay + Time.deltaTime;
150
151             if ((RTotal == 1) && (RtimeDelay >= .5))
152             {
153                 RtimeDelay = 0;
154                 RTotal = 0;
155             }
156
157             if ((RTotal == 2) && (RtimeDelay < .5))
158             {
159                 xMovement = 7;
160                 RTotal = 0;
161             }
162
163             if ((RTotal == 2) && (RtimeDelay >= .5))
164             {
165                 xMovement = 0;
166                 RTotal = 0;
167                 RtimeDelay = 0;
168             }
169
170             if (xMovement == 7)
171             {
172                 DashDuration = DashDuration + Time.deltaTime;
173             }
174
175             if (DashDuration > 1)
176             {
177                 xMovement = 0;
178                 DashDuration = 0;
179                 RTotal = 0;
180                 RtimeDelay = 0;
181             }
```

```
182             // Back Movement
183             if (Input.GetKey("left"))
184             {
185                 animator.SetFloat("Speed", -BackVel);
186                 rgb2d.velocity = new Vector2(-BackVel - xMovement, rgb2d.velocity.y);
187             }
188
189
190             if (Input.GetKeyDown("left"))
191             {
192
193                 LTotal = LTotal + 1;
194             }
195
196             if (Input.GetKeyUp("left"))
197             {
198                 xMovement = 0;
199                 rgb2d.velocity = new Vector2(0, 0);
200                 animator.SetFloat("Speed", 0);
201             }
```

```
202
203             if ((LTotal == 1) && (LtimeDelay < .5))
204                 LtimeDelay = LtimeDelay + Time.deltaTime;
205
206             if ((LTotal == 1) && (LtimeDelay >= .5))
207             {
208                 LtimeDelay = 0;
209                 LTotal = 0;
210             }
211
212             if ((LTotal == 2) && (LtimeDelay < .5))
213             {
214                 xMovement = 7;
215                 LTotal = 0;
216             }
217
218
219             if ((LTotal == 2) && (LtimeDelay >= .5))
220             {
221                 xMovement = 0;
222                 LTotal = 0;
223                 LtimeDelay = 0;
224             }
225
226             if (xMovement == 7)
227             {
228                 DashDuration = Time.deltaTime;
229             }
230
231             if (DashDuration > 0.5)
232             {
233                 xMovement = 0;
234                 DashDuration = 0;
235                 LTotal = 0;
236                 LtimeDelay = 0;
237             }
238         }
239     }
```

Ζωή

Ο κάθε παίκτης έχει πόντους ζωής και μια μπάρα οπού γίνετε η αναπαράσταση των πόντων ζωής του.

Ένας παίκτης για να χάσει πόντους ζωής, πρέπει ο αντίπαλος του να τον χτυπήσει με μια από τις μπουνιές, κλωτσιές ή με ένα combo το οποίο αναλύσαμε παραπάνω. Αυτό επιτυγχάνετε με colliders οπού λειτουργούν σαν triggers. Δηλαδή έχουμε δώσει στα χεριά και στα ποδιά των χαρακτήρων colliders οι οποίοι είναι αόρατοι και ενεργοποιούνται όταν ο παίκτης πατήσει το πλήκτρο της μπουνιάς ή της κλωτσιάς. Ο κώδικας των collider βρίσκεται στην κλάση του Player Controller οπού εκεί καλείται

και η κλάση τις ζωής του παίκτη και γίνεται η αφαίρεση των πόντων ζωής του. Επίσης όταν ο αντίπαλος παίκτης χτυπήσει τον παίκτη τότε ενεργοποιείται και ένα prefab το οποίο μας δείχνει ότι το χτύπημα πέτυχε.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (GameController.gameController.isGameFinished)
    {
        return;
    }

    if (collision.tag == "punchAttacker" && isDefence == false && !Input.GetKeyDown("down"))
    {
        Instantiate(slasher, new Vector3(transform.position.x, transform.position.y, 3f),
              Quaternion.identity);
        AudioController.audioController.PlaySound("Punch");
        if (PV.IsMine)
        {
            animator.SetTrigger("hurt");

            myHealth.ReduceHealthPlayer(punchDamage);
        }
    }
    if (collision.tag == "kickAttacker" && isDefence == false && !Input.GetKeyDown("down"))
    {
       Instantiate(slasher, new Vector3(transform.position.x, transform.position.y, 3f),
              Quaternion.identity);
        AudioController.audioController.PlaySound("Kick");
        if (PV.IsMine)
        {
            animator.SetTrigger("hurt");

            myHealth.ReduceHealthPlayer(kickDamage);
        }
    }
}
```

Εικόνα 22: Player Controller κλάση οπού γίνεται trigger ο collider όταν ο παίκτης χτυπηθεί.

Για την αφαίρεση των πόντων ζωής καλούνται δυο RPC στο script του PlayerHealth τα οποία ενημερώνουν όλους τους παίκτες στο δωμάτιο για την εναπομείνουσα ζωή του παίκτη. Επίσης γίνεται και ο έλεγχος για τον αν ο παίκτης είναι νεκρός στο παιχνίδι βάσει των πόντων ζωής του. Με την βοήθεια της μεθόδου του Photon την OnPhotonSerializeView μπορούμε να αναπαραστήσουμε και γραφικά την ζωή του παίκτη. Μας βοηθάει, δηλαδή, να στείλουμε στον αντίπαλο παίκτη τους πόντους ζωής που μας έχουν απομείνει και η μπάρα ζωής να μειώνετε αντίστοιχα. Για να χρησιμοποιήσουμε την μέθοδο OnPhotonSerializeView πρέπει να δώσουμε την IPunObservable στην κλάση έτσι ώστε να μπορεί το Photon View μας να την αναπαραστήσει. Οι παρακάτω εικόνες είναι ο κώδικας του PlayerHealth.

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

Unity Script | 3 references
public class PlayerHealth : MonoBehaviourPun,IPunObservable
{
    public static PlayerHealth health;
    public float PlayerHealthAmount;
    public bool isPlayerDead = false;
    public Image playerHealthBar;
    public GameObject PlayerCanvas;

    Unity Message | 0 references
    private void Awake()
    {
        playerHealthBar = GameObject.Find("Header/PlayerHealthBar/Mask/Content").GetComponent<Image>();
        if (photonView.IsMine)
        {
            GameController.gameController.PC2D = this.gameObject;
        }
    }
    12 references
    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.IsWriting)
        {
            stream.SendNext(playerHealthBar.fillAmount);
            stream.SendNext(isPlayerDead);
        }
        else if (stream.IsReading)
        {
            playerHealthBar.fillAmount = (float)stream.ReceiveNext();
            isPlayerDead = (bool)stream.ReceiveNext();
        }
    }
```

```
    private void CheckHealthRemaining()
    {
        playerHealthBar.fillAmount = PlayerHealthAmount;
        GameController.gameController.PlayerHealth = PlayerHealthAmount;
        if (photonView.IsMine && PlayerHealthAmount <= 0.05f)
        {
            this.GetComponent<PhotonView>().RPC("PlayerIsDead", RpcTarget.AllBuffered);
        }
    }

    [PunRPC]
    0 references
    private void PlayerIsDead()
    {
        GameController.gameController.isPlayerDead = true;
    }
    2 references
    public void ReduceHealthPlayer(float damage)
    {
        base.GetComponent<PhotonView>().RPC("ModifyHealthPlayer",RpcTarget.All,damage);
    }
```

```
56      [PunRPC]
        0 references
57      private void ModifyHealthPlayer(float damage)
58      {
59          if (photonView.IsMine)
60          {
61              PlayerHealthAmount -= damage;
62              playerHealthBar.fillAmount -= damage;
63          }
64          else
65          {
66              PlayerHealthAmount -= damage;
67              playerHealthBar.fillAmount -= damage;
68          }
69          CheckHealthRemaining();
70      }
71  }
72
```

4.3 Προβλήματα που δημιουργήθηκαν κατά την ανάπτυξη του παιχνιδιού

Παρακάτω θα γίνει μια σύντομη ανάλυση κάποιων σημαντικών προβλημάτων που δημιουργήθηκαν κατά ανάπτυξη του παιχνιδιού. Πρώτα θα αναφερθεί το πρόβλημα και έπειτα η λύση του.(Σημείωση: Δεν γίνεται πλήρης ανάλυση όλων των προβλημάτων που προέκυψαν μόνο των σημαντικότερων).

4.3.1 Δημιουργία παικτών – Client

Πρόβλημα

Για να δημιουργήσουμε ένα παίκτη όταν ξεκινάει το παιχνίδι πρέπει να φτιάξουμε prefab το οποίο θα είναι ο διαδικτυακός παίκτης. Στην δημιουργία του prefab προέκυψε το εξής πρόβλημα όταν δυο παίκτες συνδεόντουσαν στο παιχνίδι και επέλεγαν διαφορετικούς χαρακτήρες. Αυτό συνέβη λόγω του ότι και οι δυο παίκτες χρησιμοποιούσαν το ίδιο prefab όταν ξεκινούσε η μάχη. Με αποτέλεσμα και οι δυο παίκτες να έχουν τον ίδιο χαρακτήρα στην ιδιά πλευρά. Επίσης χανόταν ο συγχρονισμός.

Λύση

Το παραπάνω πρόβλημα επιλύθηκε ξεχωρίζοντας τους clients. Δηλαδή δημιουργήσαμε δυο if. Στην πρώτη if ελέγχαμε αν ο παίκτης που συνδέθηκε ήταν ο master client και του δίναμε κάποιες συγκεκριμένες πληροφορίες, όπως την πλευρά που θα βρίσκετε και τον αντίστοιχο χαρακτήρα. Η δεύτερη if έλεγχε αν ο client δεν είναι ο master client και του έδινε τις αντίστοιχες πληροφορίες. Αυτό μας βοήθησε στο να ξεχωρίσουμε τα prefab των παικτών.

4.3.2 Νίκες των παικτών

Πρόβλημα

Έπρεπε κάπως να κρατάμε τις νίκες των παικτών ως δική τους πληροφορία και έπειτα να τις εμφανίζουμε.

Λύση

Τα custom properties του Photon βοήθησαν στην επίλυση του συγκεκριμένου προβλήματος. Με την εντολή SetCustomProperties μπορέσαμε κάθε φορά που ο παίκτης έβγαινε νικητής να αυξάνονται οι νίκες του και να τις κρατάμε μέχρι ο ένας από τους δυο παίκτες να αποχωρίσει.

4.3.3 Συγχρονισμός ήχου

Πρόβλημα

Κατά την εισαγωγή του ήχου στο παιχνίδι δημιουργήθηκε ένα πρόβλημα το οποίο αφορούσε τον συγχρονισμό του. Με άλλα λόγια όταν τέλειωνε η μονομαχία και οι παίκτες συμφωνούσαν να μονομαχήσουν ξανά στον master client ο ήχος άλλαζε σε αντίθεση με τον απλό client που άκουγε δυο ήχους, τον ήχο του πρώτου χάρτη και εκείνο του νέου.

Λύση

Στην επίλυση του παραπάνω προβλήματος έπαιξαν πολύ σημαντικό ρολό τα RPC. Στον Game Controller δημιουργήσαμε κάποια RPC και τα καλέσαμε όταν άλλαζε ο χάρτης. Τα RPC αυτά σταματάνε όλους τους ήχους και ξεκινήσουν τους ήχους του καινούργιου χάρτη.

4.3.4 Ανίχνευση Game Object

Πρόβλημα

Στην ανάπτυξη του διαδικτυακού παιχνιδιού παρατηρήθηκε ένα πολύ σημαντικό πρόβλημα. Ένας χαρακτήρας για να έχει αλληλεπίδραση με ένα Game Object που βρίσκετε στην ιδιά σκηνή με αυτόν σε ένα παιχνίδι μη διαδικτυακό αρκεί να βρει το με την ονομασία του. Στο διαδικτυακό παιχνίδι αντίθετα αυτό δεν γίνετε καθώς δεν μπορούν να διαβαστούν τα Game Objects που δεν είναι στα components του χαρακτήρα.

Λύση

Το πρόβλημα αυτό λύθηκε χρησιμοποιώντας την εντολή find που παρέχει το Unity με την οποία μπορέσαμε να γράψουμε ολόκληρη την διεύθυνση της τοποθεσίας του Game Object ως παράμετρο. Έτσι κάθε φορά που παίκτης εισέρχεται στην σκηνή της μονομαχίας βρίσκει το Game Object που βρίσκετε στην συγκεκριμένη τοποθεσία και το παίρνει στην κατοχή του.

4.3.5 Συγχρονισμός μπάρας ζωής

Πρόβλημα

Το κύριο πρόβλημα που δημιουργήθηκε κατά την ανάπτυξη του script για τους πόντους ζωής ήταν στον συγχρονισμό των client καθώς όταν ένας παίκτης έχανε ζωή φαινόταν μόνο στον client του ίδιου. Με άλλα λόγια αρχικά το Fill amount της κάθε μπάρας δεσμευόταν στον παίκτη ο οποίος βρισκόταν στην ιδιά πλευρά με την μπάρα ζωής Για παράδειγμα αν ο παίκτης 1 χτυπήσει των παίκτη 2, ο παίκτης 2 χάνει ζωή και η μπάρα ζωής φαίνεται να μειώνετε στον παίκτη 2, ενώ ο παίκτης 1 την βλέπει γεμάτη.

Λύση

Για την επίλυση αυτού του προβλήματα χρειαστήκαμε την μέθοδο του Photon, OnPhotonSerializeView. Αυτή η μέθοδος είναι πολύ σημαντική για τον συγχρονισμό οποιουδήποτε Game Object έχουν οι παίκτες και θέλουμε να φαίνεται το ίδιο σε όλους τους client.. Οι εντολές που χρειαστήκαμε επίσης για να επιτευχθεί ο συγχρονισμός είναι η Stream.SendNext, η οποία βοηθάει να στείλουμε την πληροφορία στους άλλους client καθώς και την εντολή Stream.ReceiveNext , η οποία βοηθά να διαβάζουμε την πληροφορία που μας αποστέλλετε.

Επίλογος – Συμπεράσματα

Το Unity είναι μια ολοκληρωμένη και δωρεάν μηχανή ανάπτυξης παιχνιδιών που δίνει την δυνατότητα στους Game Developers να δημιουργήσουν και να αναπτύξουν ένα παιχνίδι από το μηδέν. Σε αυτήν την προσπάθεια βαρύνουσα σημασία έχουν και τα δωρεάν assets που παρέχει η συγκεκριμένη μηχανή . Επίσης το Unity έχει μια τεράστια και υποστηρικτική κοινότητα καθώς και πολλά tutorials τα οποία είναι ιδιαίτερα βοηθητικά. Στον διαδικτυακό προγραμματισμό το Unity υστερεί, αλλά χάρη στο πολύ χρήσιμο εργαλείο του Photon καθίσταται δυνατή η δημιουργία ενός διαδικτυακό παιχνίδι για να παίξουμε με τους φίλους μας. Το Photon εκτός από το Photon chat το οποίο αποδείχθηκε ιδιαίτερα χρήσιμο για την επικοινωνία των παικτών μέσα στο βιντεοπαιχνίδι, μας παρέχει και ένα άλλο εξίσου χρήσιμο εργαλείο, το οποίο ονομάζεται Photon voice chat και βοηθά στην επικοινωνία των παικτών με την χρήση μικροφώνου. Στην παρούσα πτυχιακή δεν υπήρχε η ανάγκη του Photon voice chat καθώς το βιντεοπαιχνίδι που αναπτύχθηκε αποτελεί παιχνίδι μάχης. Εν κατακλείδι η δημιουργία και η ανάπτυξη ενός διαδικτυακού βιντεοπαιχνιδιού μάχης υπήρξε μια εκπληκτική εμπειρία καθώς επίσης και η δημιουργία των animations με Pixel τέχνη.

Βιβλιογραφία

1. https://unity.com/
2. https://www.gimp.org/
3. https://www.photonengine.com/en-US/Chat
4. https://www.photonengine.com/pun
5. https://doc-api.photonengine.com/en/pun/v2/index.html
6. https://doc.photonengine.com/en-us/pun/current/demos-and-tutorials/pun-basics-tutorial/lobby
7. https://doc.photonengine.com/en-us/pun/current/demos-and-tutorials/pun-basics-tutorial/player-prefab
8. https://doc.photonengine.com/en-us/pun/current/demos-and-tutorials/pun-basics-tutorial/player-instantiation
9. https://doc.photonengine.com/en-us/pun/current/demos-and-tutorials/pun-basics-tutorial/player-networking
10. https://blog.deepworldgame.com/post/52235728843/game-art-tips-raster-vs-vector-vs-pixel-art
11. https://doc.photonengine.com/en-us/pun/current/gameplay/ownershipandcontrol
12. https://doc.photonengine.com/en-us/pun/current/gameplay/offlinemode
13. https://doc.photonengine.com/en-us/pun/current/gameplay/rpcsandraiseevent
14. https://doc.photonengine.com/en-us/pun/current/gameplay/synchronization-and-state
15. https://www.youtube.com/watch?v=on9nwbZngyw
16. https://www.youtube.com/watch?v=BRLWRSJ40UQ&list=PLOW2-VkIXKtUJDKECrv1TBnMLBljjvvxf&ab_channel=RizalNurulHudaRizalNurulHuda
17. https://www.youtube.com/watch?v=TyZDKQ2Dpjw&list=PLOW2-VkIXKtUJDKECrv1TBnMLBljjvvxf&index=4&ab_channel=RizalNurulHudaRizalNurulHuda
18. https://www.youtube.com/watch?v=RGb0_o691jo&t=401s&ab_channel=RizalNurulHudaRizalNurulHuda
19. https://www.youtube.com/watch?v=vbeAuIH-4WA&ab_channel=FirstGearGamesFirstGearGames
20. https://opengameart.org/
21. https://www.freepik.com/free-photos-vectors/game-background
22. https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro
23. Starting Photon in 5 Minutes | Photon Engine
24. https://www.youtube.com/watch?v=6I2aAwpfTh0&list=PLWeGoBm1YHVjY3vZ2qMFb95rAmyiVtBdz
25. https://www.youtube.com/watch?v=3kuVfUu6FPs&t=505s&ab_channel=InfoGamerInfoGamer