ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη 3D σκοπευτικού παιχνιδιού τρίτου προσώπου πολλαπλών παικτών τύπου Roguelite

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μετρεβέλι Νικολόζι

Επιβλέπων: Δρ. Μηνάς Δασυγένης

Επίκουρος Καθηγητής

Δρ. Κώστας Καρπούζης

Επίκουρος Καθηγητής

Εργαστήριο Ρομποτικής, Ενσωματωμένων και Ολοκληρωμένων Συστημάτων

ΚΟΖΑΝΗ ΟΚΤΩΒΡΙΟΣ 2023

HELLENIC DEMOCRACY
UNIVERSITY OF WESTERN MACEDONIA
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL
& COMPUTER ENGINEERING
ece.uowm.gr

Development of a 3D Multiplayer Third Person Roguelite Shooter game

THESIS

Metreveli Nikolozi

SUPERVISOR: Dr. Minas Dasygenis

Assistant Professor

Dr. Kostas Karpouzis

Assistant Professor

Robotics, Embedded and Integrated Systems Laboratory

KOZANI OCTOBER 2023

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Ανάπτυξη 3D σκοπευτικού παιχνιδιού τρίτου προσώπου πολλαπλών παικτών τύπου Roguelite" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος Δρ. Μηνά Δασυγένη αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Copyright (C) Μετρεβέλι Νικολόζι, Δρ. Μηνάς Δασυγένης, 2022, Κοζάνη

Υπογραφή Φοιτητή:

Μετρεβέλι Νικολόζι

Περίληψη

Τα παιχνίδια είναι ένα είδος ψυχαγωγίας και μία εναλλακτική μορφή τέχνης όπου συνδυάζεται μουσική, αφήγηση, και σχεδίαση για την δημιουργία ενός ενιαίου ψυχαγωγικού μέσου. Στις μέρες μας τα ηλεκτρονικά παιχνίδια αποτελούν μία από τις μεγαλύτερες και πιο κερδοφόρες βιομηχανίες που αναπτύσσεται όλο και περισσότερο. Ένα κομμάτι των πλεονεκτημάτων των παιχνιδιών είναι η δυνατότητα διασύνδεσης με άλλους χρήστες προάγοντας την επικοινωνία, την ομαδικότητα, και την ανταγωνιστικότητα παράλληλα. Όμως, παρόλο που σήμερα έχει αποσαφηνιστεί η δυσκολία στην δημιουργία των απλών παιχνιδιών, παραμένει μία πρόκληση η ενσωμάτωση της ομαλής λειτουργίας των παιχνιδιών με την χρήση διακομιστών.

Ο στόχος της διπλωματικής εργασίας είναι η σχεδίαση και η υλοποίηση ενός multiplayer παιχνιδιού, όπου ο παίκτης θα μπορεί να έχει τους βασικούς χειρισμούς, όπως κίνηση, άλμα, και επίθεση. Η επίθεση θα μπορεί να γίνεται είτε με το σπαθί που θα κρατάει, ή με το όπλο που θα έχει στην διάθεσή του ο παίκτης. Καθ' όλη την διάρκεια του παιχνιδιού θα μπορεί ο παίκτης να βρίσκει νέα όπλα, να τα ενδυναμώνει, να προσθέτει επιπλέον διαμορφώσεις που μεταβάλλουν την λειτουργία του όπλου, ή και εξοπλισμό που μπορούν να έχουν μια πληθώρα από επιδράσεις στο όπλο, τον παίκτη, ή ακόμα και στους εχθρούς. Σκοπός θα είναι η ολοκλήρωση του παιχνιδιού νικώντας τα δέκα κύματα από εχθρούς, χωρίς να έρθει σε κατάσταση αποτυχίας. Το παιχνίδι θα μπορεί να παιχτεί από έναν ή και περισσότερους παίκτες μέσω του δικτύου, δηλαδή θα είναι multiplayer. Κύριος σκοπός είναι η μελέτη του παιχνιδιού με την δικτύωση πολλαπλών παικτών στον ίδιο διακομιστή και η αναλυτική επεξήγηση της υλοποίησης του.

Το αποτέλεσμα που θα αποφέρει η παρούσα διπλωματική εργασία είναι τα οφέλη που θα μπορούσε να προσφέρει ένα ηλεκτρονικό παιχνίδι στους χρήστες. Τα οφέλη αυτά κυμαίνονται στον συνδυασμό της αλληλεπίδρασης με την ταυτόχρονη ψυχαγωγία που μπορεί να παρέχει το παιχνίδι. Επιπλέον, στην εργασία αυτή, θα παρουσιαστεί αναλυτικά ο τρόπος με τον οποίο πραγματοποιείται η διασύνδεση πολλαπλών χρηστών μέσω ενός διακομιστή, παρέχοντας έτσι πληροφορίες στο θέμα της δικτύωσης ενός παιχνιδιού.

Λέξεις Κλειδιά: Multiplayer, διακομιστής, παιχνίδι, roguelite, διασύνδεση, ψυχαγωγία, αλληλεπίδραση.

Abstract

Games are a form of entertainment and an alternate form of art which combines music, narration, and design for the creation of a uniform entertainment medium. Nowadays electronic games form one of the biggest and most profitable industries that are growing increasingly. One of the advantages of games is the ability to connect with other users promoting communication, teamwork, and competitiveness in parallel. Nevertheless, even though today's games have been mainstreamed, there is still a challenge in the integration of the smooth operation of multiplayer games.

The aim of this thesis is the design and implementation of a multiplayer game, where the player will be able to have the basic controls, like movement, jump, and attack. The attack can be performed either with the sword that is held, or the gun that will be available to the player. During the playthrough the player will be able to find new guns, or equipment that can have a variety of effects on the gun, player, or even to the enemies. The goal for winning the game is by defeating the ten waves of enemies, without losing beforehand. The game will be available to be played by one or more players through the network, meaning it will be multiplayer. Main target of the project is the study of the game with the networking of multiple players on the same server and the detailed explanation of its implementation.

The result that will be derived from this thesis will be the benefits that could present a game to its users. These benefits range from the mixture of interactivity, with the simultaneous entertainment that can be provided. Moreover, in this project the handling of the networking between the users on the server will be presented in detail, providing information on the topic of networking a game.

Keywords: Multiplayer, server, game, roguelite, networking, entertainment, interactivity

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω την οικογένειά μου για την ατελείωτη υποστήριξή τους καθ’ όλα τα χρόνια των φοιτητικών μου σπουδών.

Επιπλέον, θα ήθελα να ευχαριστήσω τον Δρ. Μηνά Δασυγένη και τον Δρ. Κώστα Καρπούζη για την κατανόηση και την καθοδήγηση τους που συνέβαλαν στην επιτυχή ολοκλήρωση της παρούσας διπλωματικής εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω θερμά τους φίλους μου που ποτέ δεν δίστασαν να μου προσφέρουν την βοήθειά τους όταν την χρειάστηκα.

ΚΟΖΑΝΗ ΟΚΤΩΒΡΙΟΣ 2023

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1: Εισαγωγή

Στο κεφάλαιο αυτό γίνεται μια σύντομη εισαγωγή για τα ψηφιακά παιχνίδια και την συμβολή τους στην ανθρωπότητα. Εξηγείται η δυσκολία στην παραγωγή των ψηφιακών παιχνιδιών και πιο συγκεκριμένα των παιχνιδιών που διασυνδέουν παίκτες μέσω του δικτύου. Επίσης αναγράφεται ο σκοπός της διπλωματικής εργασίας και τέλος αναγράφονται παρόμοια έργα.

1.1 Ψηφιακά παιχνίδια

Τα παιχνίδια έχουν συμβάλλει στην ζωή του ανθρώπου από την αρχή του χρόνου μέχρι και σήμερα προσφέροντας αμέτρητη ψυχαγωγία, και ατελείωτες ώρες διασκέδασης. Πέρα όμως από αυτά τα προσόντα, μπορούν να βοηθήσουν στην αντιμετώπιση του άγχους, της κατάθλιψης και της μοναξιάς [1]. Πρέπει όμως να υπάρχει περιορισμός στην πολύωρη κατανάλωση των παιχνιδιών για να μην αποφέρουν αρνητική επίδραση στο άτομο [2]. Τα παιχνίδια έχουν πάρει πολλές διαφορετικές μορφές με τους καιρούς και ειδικά τις τελευταίες δεκαετίες η ψηφιακή μορφή των παιχνιδιών έχει εξελιχθεί σε μια γιγαντιαία βιομηχανία όπου πλέον ξεπερνάει την μουσική, αλλά και την κινηματογραφική βιομηχανία [3]. Με την πάροδο του χρόνου τα παιχνίδια αναπτύσσονται και γίνονται όλο και καλύτερα σε διάφορες πτυχές, με αποτέλεσμα όλο και περισσότερος κόσμος να απορροφάται στον εικονικό κόσμο των παιχνιδιών.

Μερικοί από τους λόγους που τα παιχνίδια επιτυγχάνουν την πλήρη απορρόφηση των ατόμων είναι πως τα παιχνίδια αποφέρουν χαρά μέσω της ψυχαγωγίας τους, παρέχουν νοητική διέγερση και επίσης είναι αποτελεσματικά στην ανακούφιση από τον απαιτητικό πραγματικό κόσμο. Μεγάλο αντίκτυπο έχουν επίσης τα παιχνίδια που επιτρέπουν την πολλαπλή διασύνδεση των παικτών, επειδή προσφέρουν έναν επιπλέον, διαδραστικό τρόπο για να κοινωνικοποιηθούν και να συνδεθούν με την οικογένεια τους, φίλους ή με ξένους [4].

Οι θετικές αποκομίσεις που έχει ένα τόσο μεγάλο ποσοστό ατόμων από τα παιχνίδια, αλλά και οι θέσεις εργασίας που δημιουργούνται, δίνουν κίνητρο σε πολλούς για να ασχοληθούν με την ανάπτυξη ψηφιακών παιχνιδιών. Ανάλογα με τα κριτήρια ενός σχεδιαστή παιχνιδιών, κύριος στόχος της δημιουργίας παιχνιδιών είναι είτε για το καλύτερο οικονομικό όφελος, είτε για το πάθος και την αγάπη της δημιουργίας των παιχνιδιών ως τέχνη.

1.2 Δημιουργοί ψηφιακών παιχνιδιών

Η ανάπτυξη των ψηφιακών παιχνιδιών είναι μια πολύπλοκη διαδικασία, με πολλές παγίδες και δυσκολίες. Για την δημιουργία ενός ψηφιακού παιχνιδιού ανάλογα με την κλίμακά του, ένα άτομο χρειάζεται γνώσεις από πολλούς τομείς. Για αρχή είναι αναγκαία η εκπαίδευση πάνω στον σχεδιασμό παιχνιδιών (game design) ή τουλάχιστον να προϋπάρχει εμπειρία. Μετά πρέπει να υπάρχει το απαραίτητο θεωρητικό υπόβαθρο στο προγραμματιστικό κομμάτι για να γίνει η σωστή υλοποίηση του παιχνιδιού, όπου ανάλογα με την μηχανή παιχνιδιού, η διαδικασία αυτή μπορεί να είναι εύκολη αλλά περιοριστική στις δυνατότητές της, ή δύσκολη και απεριόριστη. Πέρα από το τεχνικό κομμάτι, υπάρχει και η καλλιτεχνική πλευρά στην ανάπτυξη

παιχνιδιών, με αποτέλεσμα ο προγραμματιστής να χρειάζεται να έχει γνώσεις και από δημιουργία μοντέλων, μουσικής, ηχητικών εφέ και οπτικών εφέ. Είναι πολύ δύσκολο και σπάνιο να έχει ένα άτομο όλες τις ικανότητες, και για αυτόν τον λόγο υπάρχουν δύο λύσεις:

1. Δωρεάν υλικό: Στο διαδίκτυο εντοπίζουμε υλικό για οτιδήποτε θα μπορούσε να ζητήσει ένας δημιουργός παιχνιδιών είτε δωρεάν, είτε επί πληρωμή. Το υλικό αυτό μπορεί να κυμαίνεται από προγραμματιστικές λύσεις προβλημάτων, σε ηχητικά ή οπτικά εφέ.
2. Διαμοιρασμός δουλειάς: Είναι συχνό να δημιουργούνται ομάδες, όπου μοιράζονται οι ρόλοι, ώστε το κάθε άτομο να ασχολείται στον τομέα που γνωρίζει καλύτερα, και να γίνεται καλύτερη διαχείριση του χρόνου αφού δεν χρειάζεται όλος ο φόρτος της δουλειάς να διεκπεραιωθεί από ένα μόνο άτομο.

1.3 Multiplayer παιχνίδια

Η δημοτικότητα των multiplayer παιχνιδιών αυξάνεται κάθε χρόνο, δημιουργώντας όλο και μεγαλύτερη ανάγκη και ζήτηση για τέτοιου είδους παιχνιδιών [5]. Οι λόγοι που οι παίκτες αναζητούν τα multiplayer παιχνίδια χωρίζονται κυρίως στους εξής [6]:

1. Συνδεσιμότητα: Οι ευκαιρίες για αλληλεπίδραση και σύνδεση μεταξύ ατόμων λιγοστεύουν όσο ο κόσμος μας παίρνει την στροφή προς την ψηφιοποίηση, οπότε τα multiplayer παιχνίδια προσφέρουν ένα παράθυρο ευκαιρίας για την σύνδεση μεταξύ φίλων, οικογένειας αλλά και αγνώστων με έναν πιο διασκεδαστικό τρόπο, που δεν θα ήταν δυνατός στον πραγματικό κόσμο.
2. Ανταγωνιστικότητα: Οι παίκτες απολαμβάνουν την αίσθηση της ανταγωνιστικότητας και της πρόκλησης που προσφέρουν τα παιχνίδια. Με την ενσωμάτωση πολλαπλών παικτών, δημιουργείται η αίσθηση και η ανάγκη να φέρουν μια καλύτερη απόδοση σε σχέση με τους υπόλοιπους παίκτες.
3. Συνεργασία: Για άλλους παίκτες η ανταγωνιστικότητα δεν είναι απαραίτητη και αντ' αυτού προτιμούν ένα πιο ήρεμο περιβάλλον όπου σε συνεργασία με άλλους παίκτες προσπαθούν να ολοκληρώσουν τις προκλήσεις του παιχνιδιού.
4. Κοινότητα: Πίσω από κάθε multiplayer παιχνίδι υπάρχει η αντίστοιχη κοινότητα με τα δικά της φόρουμ, και σελίδες για την κοινωνικοποίηση των παικτών.

Εν τέλει, οι παίκτες έχουν αρκετούς λόγους που μπορούν να τους προσελκύσουν στα multiplayer παιχνίδια, και σήμερα που η σύνδεση στο διαδίκτυο με υψηλές ταχύτητες είναι δεδομένη, τα multiplayer παιχνίδια είναι πιο προσβάσιμα από πότε.

1.4 Σκοπός

Σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη ενός 3D σκοπευτικού παιχνιδιού τρίτου προσώπου με μηχανισμούς Roguelite αλλά και με την δυνατότητα σύνδεσης στο δίκτυο, άρα είναι δυνατό για πολλαπλούς παίκτες να παίξουν μαζί. Τα παιχνίδια πολλαπλών παικτών (multiplayer) έχουν πάντα μια αυξημένη δυσκολία για την υλοποίησή τους και χρειάζονται ειδική αντιμετώπιση για την επίλυσή τους.

Στην εργασία αυτή γίνονται ξεκάθαρα τα βήματα που χρειάστηκαν σε κάθε κομμάτι του παιχνιδιού για την ομαλή και σωστή εκτέλεσή του από την διαδικτυακή του πλευρά. Ταυτόχρονα εισάγονται στοιχεία από roguelite παιχνίδια για να κρατήσουν το ενδιαφέρον του παίκτη αλλά

και για να προωθήσουν την εκκίνηση διαδοχικών διαδρομών. Όλα αυτά μπορούν να πραγματοποιηθούν με την ελαχιστοποίηση της καθυστέρησης από τους διακομιστές, εφόσον ο παίκτης συνδεθεί στο διακομιστή με την καλύτερη σύνδεση (πραγματοποιείται ο έλεγχος αυτόματα).

1.5 Παρόμοια παιχνίδια

Έχουν πραγματοποιηθεί έργα που σχετίζονται με την ανάπτυξη ενός παιχνιδιού στην Unity με πολλαπλούς παίκτες και εφαρμόζουν μηχανισμούς από roguelite παιχνίδια, όπως:

- Risk of Rain 2

Το Risk of Rain 2 [7] είναι ένα roguelite παιχνίδι τρίτου προσώπου όπου υπάρχει η δυνατότητα της διασύνδεσης πολλαπλών παικτών. Στο παιχνίδι αυτό οι παίκτες επιλέγουν τους χαρακτήρες τους και αρχίζουν το παιχνίδι όπου έρχονται αντιμέτωποι με πολλαπλούς εχθρούς και προκλήσεις. Οι παίκτες κατά την διάρκεια του παιχνιδιού προσπαθούν να βρουν τυχαία αντικείμενα που με την απόκτησή τους ενδυναμώνονται επιθετικά αλλά και αμυντικά ώστε να γίνει δυνατή η αντιμετώπιση των εχθρών που με την πάροδο του χρόνου γίνονται όλο και δυσκολότεροι. Οι εχθροί όπως οι αναβαθμίσεις δημιουργούνται τυχαία, προσφέροντας μια εντελώς διαφορετική εμπειρία με κάθε καινούργια εκκίνηση του παιχνιδιού. Ο σκοπός του παιχνιδιού είναι η επιτυχής ολοκλήρωση όλων των επιπέδων που περιέχονται.

- Gunfire Reborn

Το Gunfire Reborn [8] είναι άλλο ένα roguelite παιχνίδι με την δυνατότητα της διασύνδεσης πολλαπλών παικτών, με την διαφορά ότι σε αυτήν την περίπτωση το παιχνίδι είναι πρώτου προσώπου. Ομοίως και σε αυτό το παιχνίδι οι παίκτες επιλέγουν τους χαρακτήρες που επιθυμούν, και αρχίζει το παιχνίδι. Το περιβάλλον και οι εχθροί δημιουργούνται κάθε φορά τυχαία. Στόχος των παικτών είναι να κερδίσουν όλους τους εχθρούς που παρουσιάζονται καθώς αξιοποιούν τα πολλαπλά μοναδικά όπλα που προσφέρει το παιχνίδι. Στο τέλος κάθε παιχνιδιού οι παίκτες μπορούν να χρησιμοποιήσουν τους πόντους που συλλέχθηκαν για την μόνιμη ενδυνάμωσή τους.

Τα παιχνίδια που προαναφέρθηκαν παρείχαν πληροφορίες και ιδέες για την ανάπτυξη της παρούσας διπλωματικής εργασίας. Συγκεκριμένα, έγινε συνδυασμός από μηχανισμούς που περιέχουν και τα δύο παιχνίδια για την ανάπτυξη του παιχνιδιού της παρούσας διπλωματικής εργασίας. Συμπληρωματικά, έχουν γίνει μετρήσεις για την μέγιστη καθυστέρηση που μπορεί να υπάρχει σε ένα παιχνίδι κρατώντας όμως την επίδοση των παικτών σε σχετικά υψηλά επίπεδα [9]. Ένας επιπλέον στόχος είναι να μην υπερφορτωθεί με μηνύματα ο διακομιστής, ώστε η καθυστέρηση να μείνει μικρότερη από τα 500 ms που καθίσταται ως το μέγιστο αποδεκτό όριο για την ομαλή επίδοση του παίκτη στα πλαίσια ενός παιχνιδιού τρίτου προσώπου [9].

1.6 Επισκόπηση κεφαλαίων

Στο κεφάλαιο 2 παρουσιάζεται το θεωρητικό υπόβαθρο της εργασίας αναλύοντας όλα τα λογισμικά και εργαλεία που ήταν απαραίτητα ή βοήθησαν στην εκπόνηση της διπλωματικής εργασίας. Στο κεφάλαιο 3 αναδεικνύεται η σχεδίαση του παιχνιδιού, δηλαδή γίνεται η ανάλυση από την κάθε πτυχή που σχετίζεται με την δημιουργία του παιχνιδιού, όπως τα βασικά θέματα του παιχνιδιού, οι γενικοί μηχανισμοί, οι κανόνες, τα γραφικά, η μουσική, κ.α. Στο κεφάλαιο 4

αναγράφεται το τεχνικό κομμάτι της εργασίας, η υλοποίηση του παιχνιδιού, ο κώδικας που έπρεπε να γραφτεί για την εκπλήρωση των συστημάτων, οι τρόποι που επιλέχθηκαν για την καλύτερη επίδοση του παιχνιδιού σε παλαιότερα συστήματα υπολογιστή, και επίσης όλοι οι μέθοδοι που χρησιμοποιήθηκαν για την αποφυγή του φόρτου στο δίκτυο ώστε να ελαχιστοποιηθούν οι καθυστερήσεις από τα μηνύματα που αποστέλλονται από κάθε χρήστη. Στο κεφάλαιο 5 αναγράφονται τα αποτελέσματα και συμπεράσματα της διπλωματικής εργασίας μετά από τον διαμοιρασμό του παιχνιδιού σε δοκιμαστές. Τέλος στο κεφάλαιο 6 είναι ο επίλογος της εργασίας με τελικά συμπεράσματα, μετρικές κώδικα, και μελλοντικές επεκτάσεις.

Κεφάλαιο 2: Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζονται όλα τα λογισμικά που χρησιμοποιήθηκαν κατά την διάρκεια της διπλωματικής εργασίας, καθώς και αναλύονται οι κατηγορίες στις οποίες ανήκει το παιχνίδι.

2.1 Εργαλεία ανάπτυξης παιχνιδιού

2.1.1 Game Engine – Μηχανή Παιχνιδιού

Η ανάπτυξη ψηφιακών παιχνιδιών μπορεί να γίνει δυνατή με την χρήση μιας μηχανής παιχνιδιών [10]. Οι μηχανές παιχνιδιών είναι λογισμικά εξειδικευμένα για την ανάπτυξη ψηφιακών παιχνιδιών, παρέχοντας εργαλεία όπως μια μηχανή απόδοσης για 2D ή 3D γραφικά, μια μηχανή φυσικής ή εντοπισμού συγκρούσεων, ήχο, scripting, animation, τεχνητή νοημοσύνη, δικτύωση, streaming, διαχείριση μνήμης, νήματα (threading), υποστήριξη τοπικοποίησης, και ένα γράφο σκηνής (scene graph). Έχουν δημιουργηθεί πολλές μηχανές παιχνιδιών όπου η κάθε μια έχει τα δικά της πλεονεκτήματα, αλλά αυτές που ξεχωρίζουν με διαφορά είναι:

- Unity
- Unreal Engine
- Godot

2.1.2 Unity

Για το παιχνίδι που αναπτύχθηκε επιλέχθηκε η μηχανή παιχνιδιού Unity. Ο λόγος είναι επειδή η Unity είναι η μηχανή παιχνιδιού με τα περισσότερα εγχειρίδια και ενεργούς χρήστες με αποτέλεσμα να υπάρχει πληθώρα λύσεων σε τυχόν προβλήματα ή ακόμα και πολλά εκπαιδευτικά βίντεο και έγγραφα. Η μηχανή παιχνιδιών Unity δημιουργήθηκε από την Unity Technologies το 2005, εξαρχής για τα λογισμικά Mac OS αποκλειστικά [11]. Σήμερα χρησιμοποιείται παγκοσμίως με πάνω από 2.5 εκατομμύρια εγγεγραμμένους προγραμματιστές και υποστηρίζει μια μεγάλη γκάμα από πλατφόρμες όπως τα λογισμικά ενός υπολογιστή, κονσόλες, κινητά, και πλατφόρμες εικονικής πραγματικότητας (VR). Μερικά από τα δημοφιλέστερα παιχνίδια έχουν αναπτυχθεί στην Unity όπως, Hollow Knight, Fall guys, και το Risk of Rain 2 που είναι η κύρια έμπνευση του παιχνιδιού της διπλωματικής εργασίας.

Η Unity οφείλει την επιτυχία της στο γεγονός ότι [12]:

- Είναι διαπλατφορμική, προσφέροντας υποστήριξη για πάνω από 25 πλατφόρμες δίνοντας την δυνατότητα για περισσότερη προσβασιμότητα, άρα και περισσότερους πιθανούς χρήστες.
- Επιτρέπει την δημιουργία παιχνιδιών σε δυο διαστάσεις (2D), τρεις διαστάσεις (3D), εικονική πραγματικότητα (VR), και επαυξημένη πραγματικότητα (AR).
- Είναι δωρεάν, παρέχοντας όλα τα απαραίτητα εργαλεία για έναν ανεξάρτητο προγραμματιστή ή ακόμα και μια μικρή εταιρεία, με την επιλογή της πληρωμής του επαγγελματικού πακέτου που περιέχει επιπρόσθετες λειτουργίες.
- Παρέχει γραφικά υψηλού επιπέδου ώστε οι χρήστες να μπορούν να φτιάχνουν αισθητικά όμορφα παιχνίδια.

- Προσφέρει το Unity Asset Store, δηλαδή το κατάστημα προϊόντων της Unity, όπου χρήστες μπορούν να αγοράσουν και να πουλήσουν όλων των ειδών προϊόντα που χρησιμεύουν στην ανάπτυξη παιχνιδιών.
- Δίνει την δυνατότητα της ανάπτυξης παιχνιδιών χωρίς να κατέχει ο χρήστης κατάρτιση σε κάποια προγραμματιστική γλώσσα, οπότε ελαχιστοποιείται η χρήση του κώδικα.
- Έχει μια μεγάλη κοινότητα, και έτσι δίνονται με ευκολία απαντήσεις στις ερωτήσεις και στα προβλήματα που μπορεί να έχει ο κάθε χρήστης.

Οι γλώσσες προγραμματισμού που χρησιμοποιούνται για την ανάπτυξη του κώδικα στην Unity είναι η C#, JavaScript, και Boo. Σήμερα, λόγω της μεγάλης πλειοψηφίας της κοινότητας που χρησιμοποιεί την γλώσσα προγραμματισμού C#, διακόπηκε η υποστήριξη για τις γλώσσες JavaScript και Boo.

2.1.3 Unity Asset Store

Από το Unity Asset Store χρησιμοποιήθηκαν μερικά από τα δωρεάν 3D μοντέλα για τον χαρακτήρα και τους εχθρούς του παιχνιδιού, όπως και διάφορα οπτικά εφέ. Το Unity Asset Store είναι το εικονικό κατάστημα για τους χρήστες της Unity, με απώτερο σκοπό την αγορά μιας ευρείας ποικιλίας από περιεχόμενο όπως textures, μοντέλα, animations, εφέ, ήχους αλλά και εργαλεία όπως κώδικα για την επέκταση της λειτουργίας της Unity [13].

2.1.4 C#

Η C# είναι η κύρια γλώσσα προγραμματισμού που υποστηρίζεται από την Unity, οπότε έγινε η συγγραφή του κώδικα σε C#. Η C# είναι μια γλώσσα προγραμματισμού που δημιουργήθηκε από την Microsoft μέσα από την πλατφόρμα .NET [14] και αργότερα αναγνωρίστηκε επισήμως από την Ecma και την ISO το 2003. Είναι μια από τις γλώσσες προγραμματισμού που δημιουργήθηκαν για την Κοινή Υποδομή Γλώσσας (αγγλικά). Ο κύριος σκοπός της είναι η απλή αντικειμενοστραφής γλώσσα για γενική χρήση.

2.1.5 IDE

Ένα IDE, ή αλλιώς ολοκληρωμένο περιβάλλον ανάπτυξης, είναι ένα λογισμικό που βοηθάει στην ανάπτυξη προγραμμάτων του υπολογιστή. Συνήθως περιλαμβάνει κάποιον επεξεργαστή πηγαίου κώδικα, έναν μεταγλωττιστή, εργαλεία αυτόματης παραγωγής κώδικα, αποσφαλμάτωση, συνδέτη, σύστημα ελέγχου εκδόσεων και εργαλεία κατασκευής γραφικών διασυνδέσεων χρήστη για τις υπό ανάπτυξη εφαρμογές [15]. Μερικά από τα δημοφιλέστερα IDE είναι τα:

- Visual Studio
- IntelliJ IDEA
- Eclipse

2.1.6 Visual Studio Code

Το IDE που επιλέχθηκε είναι το Visual Studio λόγω προϋπάρχουσας εμπειρίας με το συγκεκριμένο λογισμικό, και επίσης υπάρχει καλή συμβατότητα με την μηχανή παιχνιδιού Unity. Το Visual Studio Code, ή αλλιώς VS Code, είναι ένας επεξεργαστής πηγαίου κώδικα αναπτυγμένο από την Microsoft με την χρήση του Electron framework για τα λειτουργικά συστήματα Windows, Linux και macOS. Το VS Code υποστηρίζει λειτουργίες όπως αποσφαλμάτωση, επισήμανση σύνταξης, έξυπνη συμπλήρωση κώδικα, αποσπάσματα, ανακατασκευή κώδικα, και ενσωματωμένο Git. Ένα από τα μεγάλα πλεονεκτήματα του VS Code είναι ότι οι χρήστες έχουν την δυνατότητα να εγκαταστήσουν επεκτάσεις, οι οποίες μπορούν να προσθέσουν λειτουργίες στο βασικό πρόγραμμα [16].

2.1.7 Github

Λόγω της έκτασης της εργασίας και του χρονικού διαστήματος, κρίνεται απαραίτητη η χρήση ενός λογισμικού που μπορεί να συντηρεί πολλαπλές εκδοχές του κώδικα. Το Github είναι μία πλατφόρμα για τον έλεγχο έκδοσης χρησιμοποιώντας το Git. Είναι χρήσιμο για την αποθήκευση, συντήρηση και διαχείριση του κώδικα [17].

2.1.8 Blender

Έγινε χρήση του Blender για την διόρθωση των animations που δεν ταίριαζαν με τα υπόλοιπα προεπιλεγμένα animations. Το Blender είναι ένα δωρεάν open-source λογισμικό, με πρωταρχικό στόχο την σχεδίαση 3D γραφικών. Χρησιμοποιείται για modeling, rigging, προσομοιώσεις νερού, animations, rendering, μη γραμμική επεξεργασία και για δημιουργία αλληλεπιδραστικών 3D εφαρμογών όπως τα βιντεοπαιχνίδια [18].

2.1.9 Photon

Για την δικτύωση των παιχνιδιών υπάρχουν αρκετές επιλογές, αλλά η υπηρεσία που προσφέρει διακομιστές για την διαδικτύωση των παικτών γίνεται με το Photon Unity Networking. Το Photon Unity Networking (PUN) είναι ένα πακέτο της Unity για την δημιουργία των παιχνιδιών με πολλαπλούς παίκτες μέσω του δικτύου. Αφού το παιχνίδι είναι μέχρι 4 άτομα και υπάρχει συνεργασία μεταξύ των παικτών αντί για ανταγωνισμό, δεν υπάρχει ανάγκη για αυστηρό έλεγχο από τους διακομιστές. Το PUN χρησιμοποιεί εξειδικευμένους διακομιστές για να βοηθήσει στην ευέλικτη διασύνδεση των παικτών και την εισαγωγή τους σε δωμάτια όπου διάφορα αντικείμενα παιχνιδιού και μεταβλητές μπορούν να συγχρονιστούν. Για την επιτυχή εύρεση δωματίων οι παίκτες πρέπει να συνδεθούν στον ίδιο διακομιστή. Η Photon παρέχει πολλούς διακομιστές παγκοσμίως [19]. Αφού συνδεθούν οι παίκτες στο ίδιο δωμάτιο, αναλαμβάνει ο δημιουργός του δωματίου (host) τους ελέγχους και την συμπεριφορά των διαδικτυακών αντικειμένων. Οι υπόλοιποι παίκτες θεωρούνται ως clients. Οι κύριες λειτουργίες για τον συγχρονισμό των παικτών είναι τα RPC (Remote Procedure Calls), τα Custom Properties, και τα Photon events "χαμηλού επιπέδου". Οι λειτουργίες που χρησιμοποιήθηκαν ήταν τα RPC και τα Photon events. Άλλο ένα χρήσιμο εργαλείο της Photon για τον συγχρονισμό όχι μόνο των παικτών αλλά οποιουδήποτε αντικειμένου, είναι το PhotonView.

Το PhotonView είναι ένα εξάρτημα (component) που προστίθεται σε ένα αντικείμενο παιχνιδιού για να γίνει αυτόματα ένα δικτυωμένο αντικείμενο και να αναγνωρίζεται μέσω του μοναδικού id που του ανατίθεται. Κάθε αντικείμενο που περιέχει από ένα PhotonView ανήκει στον παίκτη που το δημιούργησε, και άμα αποσυνδεθεί ο παίκτης, όλα τα αντικείμενα με PhotonView που του ανήκουν θα εξαφανιστούν. Επιπλέον το PhotonView κάνει εύκολο τον συγχρονισμό της τοποθεσίας, της περιστροφής, και του μεγέθους του αντικειμένου.

Τα RPC είναι διαδικτυακά μηνύματα για την εκτέλεση κώδικα που στέλνονται προς όλους τους χρήστες που βρίσκονται στο ίδιο δωμάτιο. Μόνο αντικείμενα που έχουν ένα Photon View μπορούν να καλούν συναρτήσεις με την μέθοδο RPC. Τα RPC βοηθάνε στον εντοπισμό του αντικειμένου που πρέπει να εκτελέσει την ίδια συνάρτηση σε όλους τους χρήστες. Μια συνάρτηση μπορεί να κληθεί με RPC όταν αναγράφεται [PunRPC] ακριβώς από πάνω της. Όταν εκτελείται ένα RPC από ένα αντικείμενο, στέλνει αυτόματα και το id από το PhotonView που περιέχεται και όποια άλλη παράμετρος οριστεί. Η αποστολή του RPC έχει 4 επιλογές:

- RpcTarget.All – Η συνάρτηση RPC καλείται σε όλους τους χρήστες, όμως ο χρήστης που έστειλε το RPC τρέχει τον κώδικα αμέσως χωρίς καθυστέρηση.
- RpcTarget.Buffered – Αποθηκεύονται τα RPC που καλούνται και στέλνονται στους νέους παίκτες που συνδέονται.
- RpcTarget.Others – Η συνάρτηση RPC καλείται για όλους τους παίκτες εκτός από τον παίκτη που κάλεσε την συνάρτηση RPC.
- RpcTarget.ViaServer – Η συνάρτηση RPC περνάει πρώτα από τον διακομιστή της Photon προτού γυρίσει και τρέξει ο κώδικας στους παίκτες.

Τα Photon events (RaiseEvents) είναι επίσης διαδικτυακά μηνύματα, όπου το event στέλνει μόνο έναν κωδικό προς όλους τους παίκτες για την αναγνώριση της συνάρτησης που πρέπει να εκτελεστεί με τις παραμέτρους που προστίθενται. Το πλεονέκτημα των events είναι το γεγονός ότι δεν απαιτείται το αντικείμενο να έχει το component PhotonView. Από την άλλη η αποστολή ενός event χρειάζεται μεγαλύτερη διαδικασία από την μεριά του προγραμματισμού σε σχέση με τα RPC. Υπάρχει η επιλογή της αξιοπιστίας ή της αναξιοπιστίας κατά τον τρόπο αποστολής του event. Τα events έχουν επίσης επιλογές παρόμοιες για την αποστολή με RPC με την ρύθμιση RaiseEventOptions:

- All – Ο κωδικός του event στέλνεται σε όλους τους παίκτες.
- Others – Ο κωδικός του event στέλνεται σε όλους εκτός από τον παίκτη που κάλεσε το RaiseEvents.
- MasterClient – Ο κωδικός του event στέλνεται μόνο στον host του δωματίου.

Υποστηρίζονται όλες οι πλατφόρμες που υποστηρίζει και η Unity. Οι θύρες που χρειάζονται για την ομαλή λειτουργία του πακέτου μαζί με το πρωτόκολλο και τον σκοπό τους είναι οι εξής (Πίνακας 1):

Πίνακας 1 Θύρες, πρωτόκολλα και η χρήση τους για την λειτουργία του πακέτου Photon

Αριθμός θύρας	Πρωτόκολλο	Χρήση
5058 ή 27000	UDP	Client σε Nameserver (UDP)

5055 ή 27001	UDP	Client σε Master Server (UDP)
5056 ή 27002	UDP	Client σε Game Server (UDP)
4533	TCP	Client σε Nameserver (TCP)
4530	TCP	Client σε Master Server (TCP)
4531	TCP	Client σε Game Server (TCP)
9090	TCP	Client σε Master Server (WebSockets)
9091	TCP	Client σε Game Server (WebSockets)
9093	TCP	Client σε Nameserver (WebSockets)
19090	TCP	Client σε Master Server (Ασφαλή WebSockets)
19091	TCP	Client σε Game Server (Ασφαλή WebSockets)

2.1.10 Draw.io

Τα διαγράμματα που εξηγούν τον σχεδιασμό του παιχνιδιού φτιάχτηκαν στο Draw.io. Το Draw.io, η αλλιώς Diagrams.net είναι ένα διαπλατφορμικό λογισμικό για την δημιουργία πολλών ειδών γραφημάτων. Είναι δωρεάν, και μπορεί να χρησιμοποιηθεί χωρίς την σύνδεση στο δίκτυο [20].

2.1.11 Sketchfab

Πολλά από τα 3D μοντέλα για τα όπλα και τις αναβαθμίσεις προμηθεύτηκαν από την ιστοσελίδα Sketchfab.com. Το Sketchfab είναι μία ιστοσελίδα γεμάτη με 3D, VR και AR περιεχόμενο. Η κύρια χρήση της σελίδας είναι ο διαμοιρασμός, η αγορά και η πώληση 3D μοντέλων [21]. Η σελίδα αυτή είναι πολύ χρήσιμη για την εύρεση 3D μοντέλων δωρεάν.

2.1.12 Paint.net

Στο παιχνίδι χρειάστηκαν μερικές εικόνες για τον εμπλουτισμό της διεπαφής του χρήστη, άρα είναι αναγκαίος ένας επεξεργαστής εικόνων. Το Paint.net είναι ένα δωρεάν λογισμικό για την επεξεργασία των εικόνων και φωτογραφιών που παρέχει όλες τις βασικές λειτουργίες με παρόμοια λογισμικά. Είναι ένα χρήσιμο εργαλείο για την δημιουργία ή την μεταποίηση εικόνων που μπορούν να χρησιμοποιηθούν στην ανάπτυξη ενός παιχνιδιού [22].

2.1.13 ZapSplat

Εκτός από τα γραφικά και τα οπτικά εφέ είναι απαραίτητος και ο ήχος για την καλύτερη εμπειρία του παίκτη με την μουσική, αλλά και για να μεταφέρεται η σωστή ανταπόκριση ανάλογα με την κατάσταση του παιχνιδιού. Το ZapSplat είναι μια βιβλιοθήκη με δωρεάν, χωρίς δικαιώματα μουσική και ηχητικά εφέ για πολλαπλές καταστάσεις [23].

2.1.14 Audacity

Στις περιπτώσεις που δεν είναι πάντα δυνατή η εύρεση του κατάλληλου ηχητικού εφέ γίνεται η χρήση του Audacity. Το Audacity είναι άλλο ένα δωρεάν open-source λογισμικό, όπου εξειδικεύεται στην καταγραφή και στην επεξεργασία ψηφιακού ήχου [24]. Βασικό πλεονέκτημα είναι η απλή διεπαφή χρήστη που διαθέτει, με αποτέλεσμα να μπορούν να το χρησιμοποιήσουν εύκολα και χρήστες με χαμηλό επίπεδο γνώσεων στην επεξεργασία ήχου. Ένα παιχνίδι χρειάζεται πολλά προσαρμοσμένα ηχητικά εφέ τα οποία μπορούν να καλυφθούν με τις λειτουργίες που παρέχει το Audacity.

2.2 Κατηγορίες παιχνιδιών

2.2.1 Third Person Shooter

Τα παιχνίδια Σκοποβολής Τρίτου Προσώπου (Third Person Shooter) είναι μία υποκατηγορία των 3D παιχνιδιών σκοποβολής όπου η κύρια οπτική εστίαση είναι στην σκοποβολή, και η κάμερα σε αντίθεση με τα First Person Shooter (όπου η κάμερα φαίνεται να είναι στο επίπεδο των ματιών του χαρακτήρα) είναι σε απόσταση και πίσω από τον χαρακτήρα.

2.2.2 Roguelike

Τα roguelike παιχνίδια είναι μία υποκατηγορία παιχνιδιών ρόλου που έχουν πολύ καθορισμένα χαρακτηριστικά αλλά στις μέρες μας ο όρος αυτός χρησιμοποιείται πολύ συχνά εσφαλμένα για την κατηγοριοποίηση μιας πληθώρας παιχνιδιών [25]. Ο όρος του roguelike προήλθε από το παιχνίδι Rogue, το οποίο κυκλοφόρησε το 1980 και είχε τους μηχανισμούς παιχνιδιού που ορίζουν τα roguelike παιχνίδια. Πιο συγκεκριμένα ένα παιχνίδι κανονικά ορίζεται roguelike όταν περιέχει [26]:

- Τυχαία παραγωγή περιβάλλοντος: Το περιβάλλον δημιουργείται με διαδικαστική δημιουργία, δηλαδή όχι τελείως τυχαία για να γίνει αποφυγή των ανίκητων περιπτώσεων.
- Μόνιμο θάνατο: Όταν χάσει ο παίκτης, χάνει ολόκληρη την πρόοδο του και ξεκινάει ξανά από την αρχή.
- Turn based κίνηση σε πλέγμα: Ο παίκτης βρίσκεται σε ένα πλέγμα στο περιβάλλον και οι κινήσεις του παίκτη και των εχθρών εναλλάσσονται διαδοχικά.
- Περιπλοκότητα: Το παιχνίδι πρέπει να επιτρέπει πολλαπλές λύσεις στα ενδεχόμενα προβλήματα, για να μην δημιουργούνται ανίκητα σενάρια λόγω της τυχαίας φύσης του παιχνιδιού.
- Εστίαση στους εχθρούς: Υπάρχει μεγάλη τάση στην συχνή μάχη με εχθρούς.
- Έμφαση στην εξερεύνηση: Συγκεκριμένες αποφάσεις ή σκέψεις είναι αναμενόμενες από τον παίκτη με βάση την περιέργεια και την αίσθηση της εξερεύνησης για την δημιουργία μιας εντελώς διαφορετικής εμπειρίας καθ’ όλη την διάρκεια του παιχνιδιού.

2.2.3 Roguelite

Τα roguelite παιχνίδια παίρνουν πολλούς ή λίγους από τους μηχανισμούς ενός roguelike παιχνιδιού και επενδύουν σε νέους. Τα roguelite παιχνίδια συνήθως συνδυάζονται με άλλες μεγάλες κατηγορίες παιχνιδιών. Μερικές από τις κυριότερες διαφορές που υπάρχουν σε σχέση με τα κλασικά roguelike [27]:

- Μόνιμη πρόοδος: Όταν χάνει ο παίκτης μπορεί να ξεκλειδώσει νέο εξοπλισμό ή αναβαθμίσεις, και αυτά παραμένουν μόνιμα διαθέσιμα στο παιχνίδι.
- Συνδυασμοί με άλλες κατηγορίες παιχνιδιών: Π.χ. το Slay the Spire, συνδυάζει roguelike μηχανισμούς με την δημιουργία τράπουλας, ή π.χ. το Crypt of the Necrodancer συνδυάζει roguelike μηχανισμούς με μηχανισμούς από παιχνίδια ρυθμού μουσικής.

Στο επόμενο κεφάλαιο γίνεται η ανάλυση του σχεδιασμού του παιχνιδιού από όλες τις κατηγορίες που περιέχει, όπως το θέμα του παιχνιδιού, τους μηχανισμούς, τον χειρισμό, τα γραφικά, τους ήχους, κ.α.

Κεφάλαιο 3: Ανάλυση και περιγραφή συστήματος

Στην ενότητα αυτήν παρουσιάζεται ο σχεδιασμός του παιχνιδιού στη μορφή ενός Game Design Document, όπως είναι κατάλληλο για κάθε παιχνίδι κατά την σχεδιαστική φάση του. Το παιχνίδι ονομάζεται Bullet Drifters, και είναι ένα σκοπευτικό παιχνίδι τρίτου προσώπου με πολλαπλούς παίκτες τύπου roguelite.

3.1 Εισαγωγή

Το Bullet Drifters, που αναπτύχθηκε σε αυτή τη διπλωματική εργασία, είναι ένα roguelite σκοπευτικό παιχνίδι τρίτου προσώπου έως και 4 άτομα, όπου οι παίκτες βρίσκονται σε μια αρένα αντιμετωπίζοντας μια σειρά από μάχες. Κατά την διάρκεια του παιχνιδιού οι παίκτες μπορούν να βρουν νέα και καλύτερα όπλα, να ενδυναμώσουν τον εξοπλισμό τους, και να παλέψουν για να αποκτήσουν τον τίτλο ενός Bullet Drifter. Όταν χάσουν όλοι οι παίκτες, δεν έχουν άλλη επιλογή παρά μόνο να ξεκινήσουν ξανά από την αρχή, όμως με την δυνατότητα να ξεκλειδώσουν μόνιμες αναβαθμίσεις.

3.2 Κατηγορία

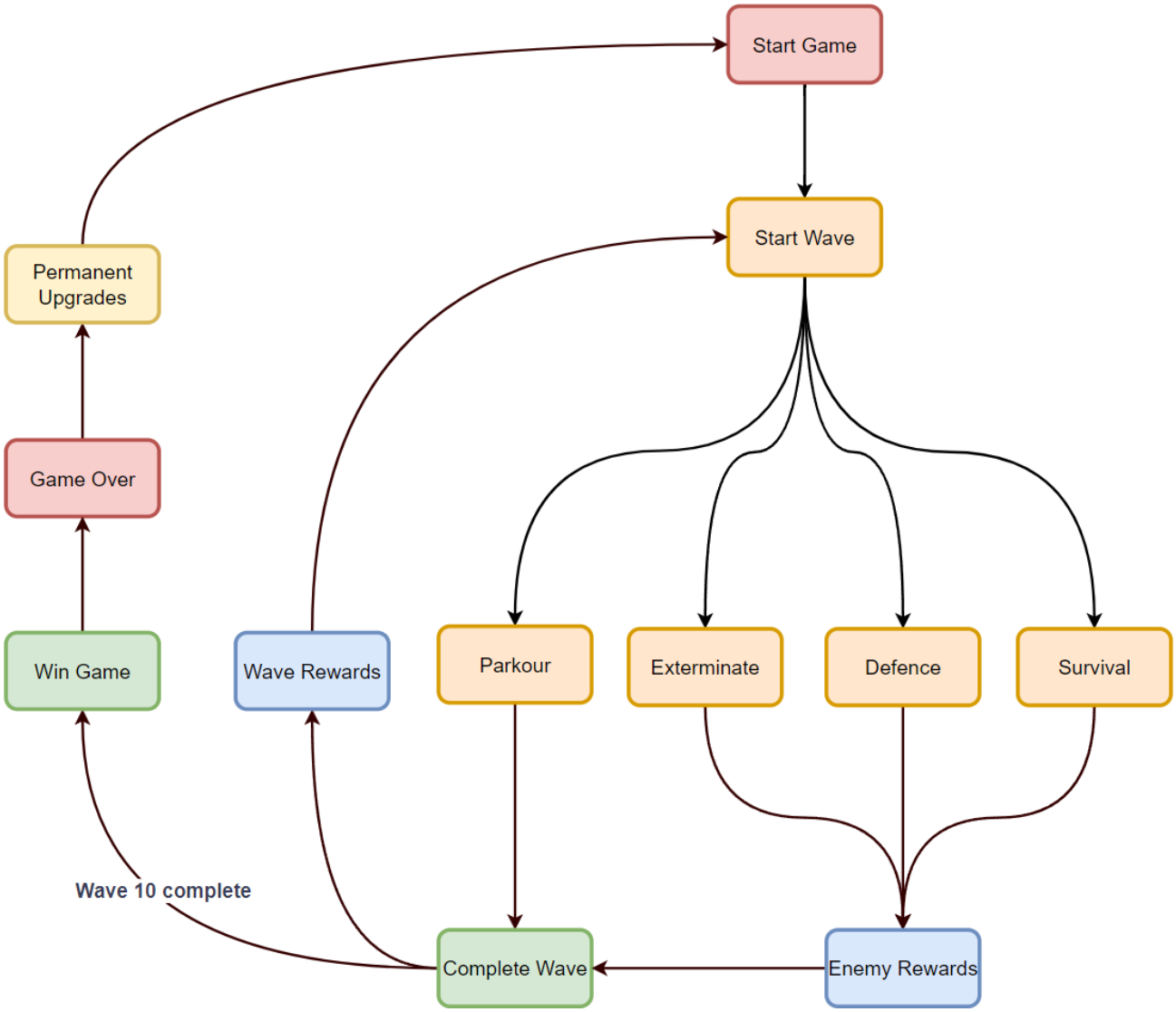
Το παιχνίδι είναι 3D σκοπευτικό τρίτου προσώπου με μηχανισμούς roguelite, περιλαμβάνοντας τις λειτουργίες ενός multiplayer παιχνιδιού. Η έμπνευση για το παιχνίδι προέρχεται από τον συνδυασμό των παιχνιδιών Risk of Rain 2 και Ratchet Gladiator (ή αλλιώς Ratchet Deadlocked). Πιο συγκεκριμένα το παιχνίδι συνδυάζει τους μηχανισμούς που περιέχει το Risk of Rain 2, όπως οι αναβαθμίσεις κατά την διάρκεια του παιχνιδιού, και η τυχαία δημιουργία των εχθρών, μαζί με το περιβάλλον και το γενικότερο θέμα του Ratchet Gladiator, όπως η αρένα στην οποία διαδραματίζεται το παιχνίδι και η ποικιλία των όπλων.

3.3 Μηχανισμοί παιχνιδιού

Για την κίνηση του χαρακτήρα, οι παίκτες μπορούν να τρέχουν στην αρένα που βρίσκονται, μπορούν να πηδήξουν, και να κάνουν μια γρήγορη επιτάχυνση προς μια κατεύθυνση (dash). Για την διαχείριση των όπλων, οι παίκτες έχουν την δυνατότητα να πυροβολήσουν άμα κρατάνε κάποιο όπλο, να ξαναγεμίσουν το όπλο τους σε περίπτωση που ξεμείνουν από σφαίρες αλλά και όποτε δώσουν την εντολή, να εναλλάσσουν μεταξύ των όπλων που έχουν στην κατοχή τους (μέγιστο τρία όπλα), ή να πετάξουν το όπλο τους κάτω. Υπάρχει η δυνατότητα αλληλεπίδρασης ανάλογα με αυτό που είναι μπροστά στους παίκτες, δηλαδή μπορούν να ανοίξουν ένα κουτί, να εξοπλιστούν με ένα όπλο, ή να αποκτήσουν power ups (προσωρινές αναβαθμίσεις) που θα βρεθούν μπροστά τους. Τέλος μπορούν επίσης να χτυπήσουν τους εχθρούς και με το σπαθί.

Το παιχνίδι αρχίζει πάντα από την αρχή με ένα κουτί για την τυχαία επιλογή όπλου για τον κάθε παίκτη. Υπάρχουν δέκα γύροι, ο καθένας με τα δικά του χαρακτηριστικά, όπου οι παίκτες έχουν ως στόχο να επιβιώσουν ενώ διάφοροι εχθροί τους κυνηγάνε για να τους προκαλέσουν ζημιά. Οι εχθροί όταν χάνουν αφήνουν πίσω νομίσματα για να συλλέγουν οι παίκτες, και επίσης έχουν την πιθανότητα να ρίξουν power ups που μπορούν οι παίκτες να μαζέψουν για να ενδυναμώσουν τα όπλα και τα στατιστικά τους. Με την εκπλήρωση κάθε γύρου, εμφανίζεται ένα καινούργιο κουτί που θα παρέχει στους παίκτες καινούργια όπλα. Όταν ο παίκτης χάσει όλους

τους πόντους ζωής του, χάνει όλα τα power ups, και όλα τα όπλα που είχαν συλλεχθεί καθ' όλη την διάρκεια του παιχνιδιού και αρχίζει ξανά από την αρχή της πίστας. Στην περίπτωση που είναι πολλαπλοί παίκτες, άμα χάσει ο ένας, μπορεί να παρακολουθήσει τους επιζήσαντες. Με τα νομίσματα που συσσωρεύονται γίνεται δυνατή η αγορά των upgrades (μόνιμων αναβαθμίσεων) για να βοηθήσει τους παίκτες στα επόμενά τους παιχνίδια, και για να έχουν την αίσθηση της προόδου. Παρουσιάζεται ο βασικός βρόχος του παιχνιδιού (Εικόνα 1):



Εικόνα 1 Βασικός βρόχος παιχνιδιού

3.3.1 Όπλα

Τα όπλα προέρχονται από τα κουτιά που δημιουργούνται στο τέλος κάθε γύρου. Είναι όλα διαφορετικά μεταξύ τους έχοντας το καθένα το δικό του ρόλο στο παιχνίδι. Όταν δημιουργούνται, έχουν την κατηγορία σπανιότητας (Εικόνα 2), όπου όσο πιο σπάνια είναι η εκδοχή του όπλου, τόσο καλύτερα στατιστικά έχει. Μερικά όπλα μπορούν να περιλαμβάνονται μόνο σε υψηλότερες κατηγορίες σπανιότητας. Υπάρχουν τέσσερις κατηγορίες σπανιότητας, Common, Uncommon, Rare, και Legendary. Τα όπλα αυτά είναι τα εξής:

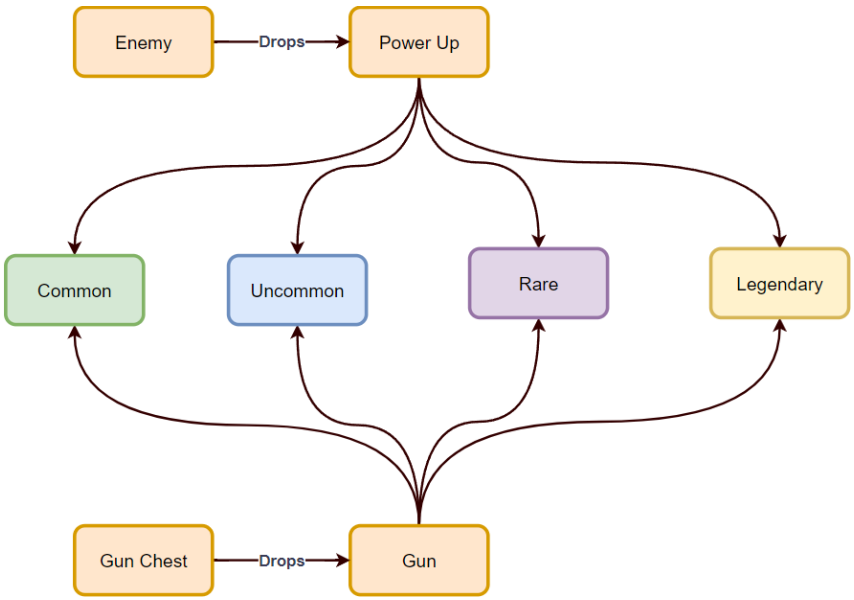
1. Pistol: Απλό πιστόλι που έχει τα βασικά στατιστικά, χρήσιμο για την αρχή του παιχνιδιού, υπάρχει σε όλες τις σπανιότητες.
2. AutoPistol: Αυτόματο πιστόλι με υψηλή ταχυβολία αλλά χαμηλή ζημιά.
3. AutoRifle: Αυτόματο τουφέκι με καλή ταχυβολία και καλή ζημιά
4. Rifle: Τουφέκι ελεύθερου σκοπευτή με υψηλή ζημιά όπου η σφαίρα διαπερνά μέσα από τους εχθρούς
5. Shotgun: Καραμπίνα που πυροβολεί έξι σφαίρες ταυτόχρονα με μέτρια ζημιά, και οι σφαίρες απλώνονται όσο απομακρύνονται.
6. Railgun: Δυνατό όπλο που παρομοιάζει με το τουφέκι ελεύθερου σκοπευτή αλλά έχει μεγαλύτερη εμβέλεια σύγκρουσης.

3.3.2 Power Ups

Τα power ups είναι μικρά αντικείμενα που ενδυναμώνουν τον παίκτη. Προέρχονται τυχαία από τους εχθρούς όταν χάνουν. Μπορούν να αποκτηθούν πολλαπλές φορές αυξάνοντας τις δυνατότητες των παικτών όλο και περισσότερο με τη πάροδο των γύρων. Όταν δημιουργούνται έχουν, σαν τα όπλα, την κατηγορία σπανιότητας (Εικόνα 2), όπου όσο πιο σπάνια είναι η εκδοχή της επαύξησης, τόσο καλύτερη η επίδρασή της. Τα power ups είναι τα εξής:

1. Health: Αυξάνει τον μέγιστο αριθμό πόντων ζωής του παίκτη κατά ένα ποσό, αλλά αυξάνει και την τρέχουσα ζωή.
2. Ammo: Αυξάνει τον μέγιστο αριθμό από τις σφαίρες των όπλων κατά ένα ποσοστό.
3. Attack Speed: Αυξάνει την ταχυβολία των όπλων κατά ένα ποσοστό.
4. Damage: Αυξάνει την δύναμη των όπλων κατά ένα ποσοστό.
5. Speed: Αυξάνει την ταχύτητα του παίκτη κατά ένα ποσοστό.
6. Super Buff: Αυξάνει την δύναμη, την ταχυβολία, και τους μέγιστους πόντους ζωής του παίκτη.

Στην περίπτωση των πολλαπλών παικτών, άμα πάρει ο ένας μια επαύξηση, τότε προστίθεται σε όλους.



Εικόνα 2 Ιδιότητα σπανιότητας όπλων και power ups

3.3.3 Upgrades

Τα upgrades μπορούν να ξεκλειδωθούν στο αρχικό μενού για την μόνιμη ενδυνάμωση του παίκτη, προσφέροντας την αίσθηση της προόδου. Για να ξεκλειδωθούν πρέπει να γίνει η συλλογή νομισμάτων μέσω του παιχνιδιού. Τα upgrades χωρίζονται σε τρεις κατηγορίες, «επίθεση», «άμυνα», και «χρησιμότητα», και είναι οι παρακάτω:

Επίθεση:

1. Αύξηση της δύναμης των όπλων
2. Αύξηση της ταχυβολίας των όπλων
3. Μείωση του χρόνου γεμίσματος των όπλων

Άμυνα:

1. Αύξηση των πόντων ζωής του παίκτη
2. Αύξηση της άμυνας του παίκτη
3. Αύξηση της δύναμης των θεραπευτικών μέσων

Χρησιμότητα:

1. Αύξηση της ταχύτητας του παίκτη
2. Αύξηση του αριθμού των αλμάτων του παίκτη
3. Μείωση της καθυστέρησης μέχρι την επομένη φορά που γίνεται δυνατό το dash

3.4 Στοχευμένο κοινό

Το Bullet Drifters προορίζεται προς όλες τις ηλικίες εφόσον έχουν εξοικειωθεί με ψηφιακά παιχνίδια, και πιο συγκεκριμένα παιχνίδια που απαιτούν λίγο υψηλότερα αντανακλαστικά και την γρήγορη στόχευση του κέρσορα.

3.5 Περίληψη Marketing

Το Bullet Drifters από την φύση του ως roguelite προσφέρει μια νέα εμπειρία παιχνιδιού σε κάθε εκκίνηση, με την τυχαιοποίηση των εχθρών, όπλων, και αναβαθμίσεων. Είναι το κατάλληλο παιχνίδι για μια ομάδα μέχρι και τεσσάρων ατόμων για να συνδεθούν μαζί και να διεκδικήσουν τον τίτλο ενός Bullet Drifter.

3.5.1 Διαθέσιμες πλατφόρμες

Το παιχνίδι διατίθεται μόνο για την πλατφόρμα ηλεκτρονικού υπολογιστή, και το λειτουργικό σύστημα Windows.

3.6 Ιστορία

Το παιχνίδι λαμβάνει μέρος σε έναν κόσμο όπου η τεχνολογία είναι πολύ πιο εξελιγμένη από τον σημερινό κόσμο, και ταυτόχρονα, ο καπιταλισμός έχει φτάσει στην κορύφωσή της. Ως αποτέλεσμα, διοργανώνονται θανάσιμοι διαγωνισμοί όπου μπορούν διάφοροι διεκδικητές να προσπαθήσουν να αγωνιστούν και να κερδίσουν τον τίτλο ενός Bullet Drifter, δηλαδή απεριόριστη δόξα και χρήματα. Με την υπερβολική προώθηση των διαφόρων προϊόντων και διαφημίσεων, έχει επιτευχθεί η απευαισθητοποίηση του πληθυσμού, οπότε το γεγονός ότι διαφημίζεται ο διαγωνισμός ακόμα και ως θανάσιμος, δεν είναι αρκετό για να αποθαρρύνει τον κόσμο από το να συμμετάσχει ή και να τον παρακολουθήσει.

3.7 Προκλήσεις

Οι προκλήσεις του παιχνιδιού χωρίζονται στις περιβαλλοντικές και στις προκλήσεις από εχθρούς. Ανάλογα με το είδος του γύρου, ο παίκτης βρίσκεται αντιμέτωπος με διαφορετικές προκλήσεις. Τα είδη γύρων είναι πέντε:

1. Exterminate: Δημιουργούνται εχθροί μέχρι έναν αριθμό που υπολογίζεται από τον αριθμό του γύρου και τον αριθμό των παικτών. Για την ολοκλήρωση του γύρου, οι παίκτες πρέπει να νικήσουν τους εχθρούς.
2. Defense: Στην μέση της αρένας εμφανίζεται ένας κρύσταλλος τον οποίο οι παίκτες έχουν ως ανώτερο σκοπό να προστατέψουν μέχρι να τελειώσει ο αριθμός των εχθρών. Στην περίπτωση που ο κρύσταλλος καταστραφεί από τους εχθρούς, το παιχνίδι καταλήγει σε ήττα.
3. Parkour: Στον γύρο αυτό εμφανίζονται λέιζερ όπου οι παίκτες καλούνται να τα αποφύγουν για να μην χάσουν πόντους ζωής και ανάλογα την δυσκολία του παιχνιδιού που κρίνεται από τον γύρο που βρίσκονται οι παίκτες, ο αριθμός των λέιζερ, η ταχύτητα τους, και η κίνησή τους μεταβάλλεται. Στην πρώτη δυσκολία εμφανίζονται τέσσερα λέιζερ στο επίπεδο της αρένας τα οποία περιστρέφονται δεξιόστροφα. Στην δεύτερη δυσκολία εμφανίζονται αλλά δυο επίπεδα από λέιζερ πάνω από τα αρχικά τέσσερα, τα οποία επίσης περιστρέφονται δεξιόστροφα αλλά πιο αργά, και ύστερα από μια μικρή καθυστέρηση, περιστρέφονται αριστερόστροφα, αντίθετα από την περιστροφή των αρχικών τεσσάρων λέιζερ. Στην τρίτη και τελευταία δυσκολία, εμφανίζονται ξανά συνολικά 3 επίπεδα από λέιζερ, αλλά κινούνται και περιστρέφονται με τον ίδιο τρόπο. Η διαφορά σε αυτήν την δυσκολία είναι η προσθήκη δύο καινούργιων εμποδίων. Ένα εμπόδιο είναι τα οριζόντια λέιζερ που έχουν μια προεπιλεγμένη κατεύθυνση, και το δεύτερο εμπόδιο είναι ένας κύκλος που κυνηγάει τους παίκτες για ένα μικρό χρονικό διάστημα, και μετά από μια μικρή παύση, εκρήγνυται χτυπώντας όποιον παίκτη είχε παραμείνει κοντά στην έκρηξη.
4. Survival: Κάθε δευτερόλεπτο μειώνεται ένας πόντος ζωής από κάθε παίκτη καθώς εχθροί τρέχουν να τους χτυπήσουν όμως ταυτόχρονα εμφανίζονται περιοχές που επουλώνουν τους παίκτες. Σκοπός των παικτών είναι να κερδίσουν τους εχθρούς όσο πιο γρήγορα μπορούν για να μην χάσουν όλους τους πόντους ζωής τους από την φύση του γύρου.
5. Rampage: Ο τελευταίος γύρος του παιχνιδιού είναι ιδιαίτερος, συνδυάζοντας μηχανισμούς από τα προηγούμενα είδη. Για αρχή, εμφανίζονται εχθροί μέχρι έναν συγκεκριμένο αριθμό που υπολογίζεται όπως και προηγουμένως με βάση τον αριθμό των παικτών. Παράλληλα οι παίκτες βρίσκονται υπό απειλή των περιβαλλοντικών προκλήσεων που εμφανίστηκαν στους γύρους Survival και Parkour. Πιο συγκεκριμένα, μαζί με τους εχθρούς που πρέπει να αντιμετωπίσουν, οι παίκτες χάνουν έναν πόντο ζωής ανά δευτερόλεπτο. Ακόμη τους ακολουθεί ο κύκλος που εκρήγνυται όπως και στον γύρο του Parkour. Το παιχνίδι έρχεται στο τέλος του όταν οι παίκτες καταφέρουν να νικήσουν και τον τελευταίο εχθρό του γύρου.

Τα είδη των εχθρών είναι δύο:

1. Βασικός εχθρός που τρέχει προς τον παίκτη ο οποίος μόλις φτάσει στην εμβέλεια της επίθεσής του, χτυπά τον παίκτη με μια ενσωματωμένη λεπίδα.
2. Ιπτάμενος εχθρός που έχει μεγαλύτερη εμβέλεια για επίθεση, και μπορεί να πυροβολήσει προς τον παίκτη.

3.8 Χειρισμός

Ο χειρισμός του παίκτη περιγράφεται με τα προεπιλεγμένα κουμπιά, τα οποία μπορεί να αλλάξει οποιαδήποτε στιγμή, είτε στο μενού, είτε αφότου συνδεθεί στο παιχνίδι.

W: Κίνηση προς τα μπροστά
S: Κίνηση προς τα πίσω
A: Κίνηση προς τα αριστερά
D: Κίνηση προς τα δεξιά
Space: Άλμα
Shift: Κατρακύλα προς την επιλεγμένη κατεύθυνση
Αριστερό Κλικ: Πυροβόληση όπλου
E: Επίθεση με σπαθί
F: Αλληλεπίδραση
G: Εγκατάλειψη εξοπλισμένου όπλου
3: Επιλογή του επόμενου εξοπλισμένου όπλου
1: Επιλογή του προηγούμενου εξοπλισμένου όπλου
Tab: Εμφάνιση στατιστικών και των αποκτημένων power ups

Υπάρχουν και μερικοί χειρισμοί που δεν μπορούν να αλλαχθούν:

Esc: Άνοιγμα του μενού κατά την διάρκεια του παιχνιδιού
Αριστερό Κλικ: Άμα έχει χάσει ο παίκτη, αλλάζει τον παίκτη που παρακολουθεί.

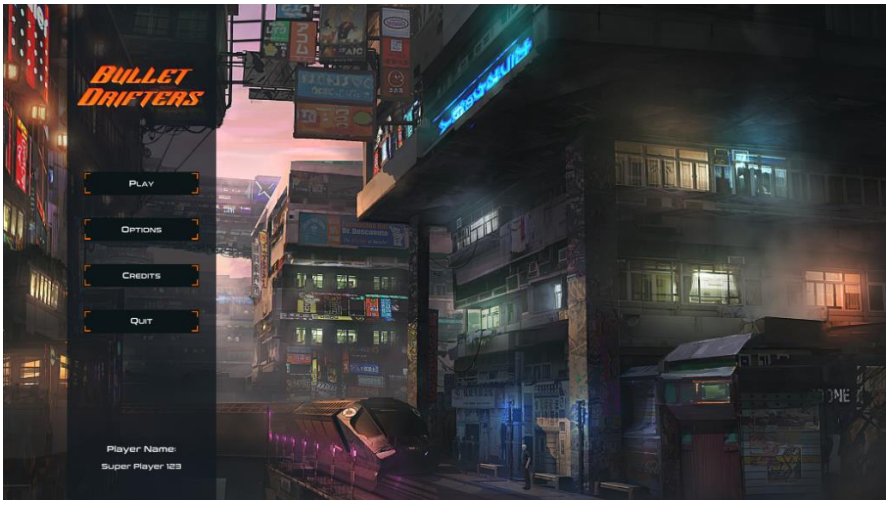
3.9 Γραφικά

Τα γραφικά του παιχνιδιού είναι επιλεγμένα έτσι ώστε να ταιριάζουν με την ιστορία και το κλίμα που θέτει, ώστε να δημιουργήσει την αίσθηση ότι οι παίκτες βρίσκονται σε ένα παιχνίδι επιστημονικής φαντασίας. Επίσης για την καλύτερη γραφική πιστότητα του παιχνιδιού, έγινε χρήση της μεταπαραγωγικής επεξεργασίας της Unity.

3.10 Διεπαφή Χρήστη

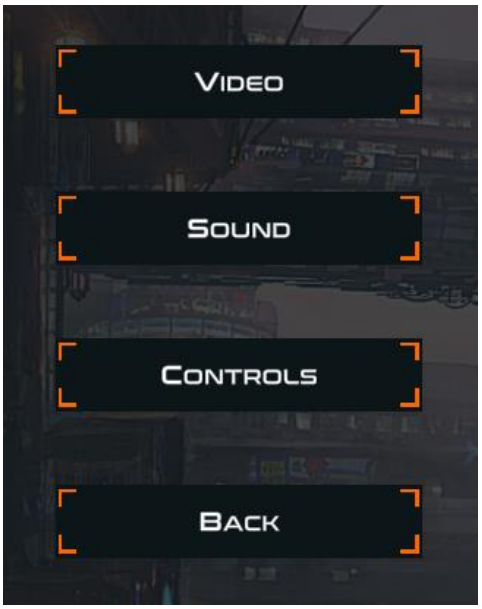
Η διεπαφή του χρήστη χωρίζεται σε δύο μέρη, το Μενού, και την Αρένα.

3.10.1 Μενού



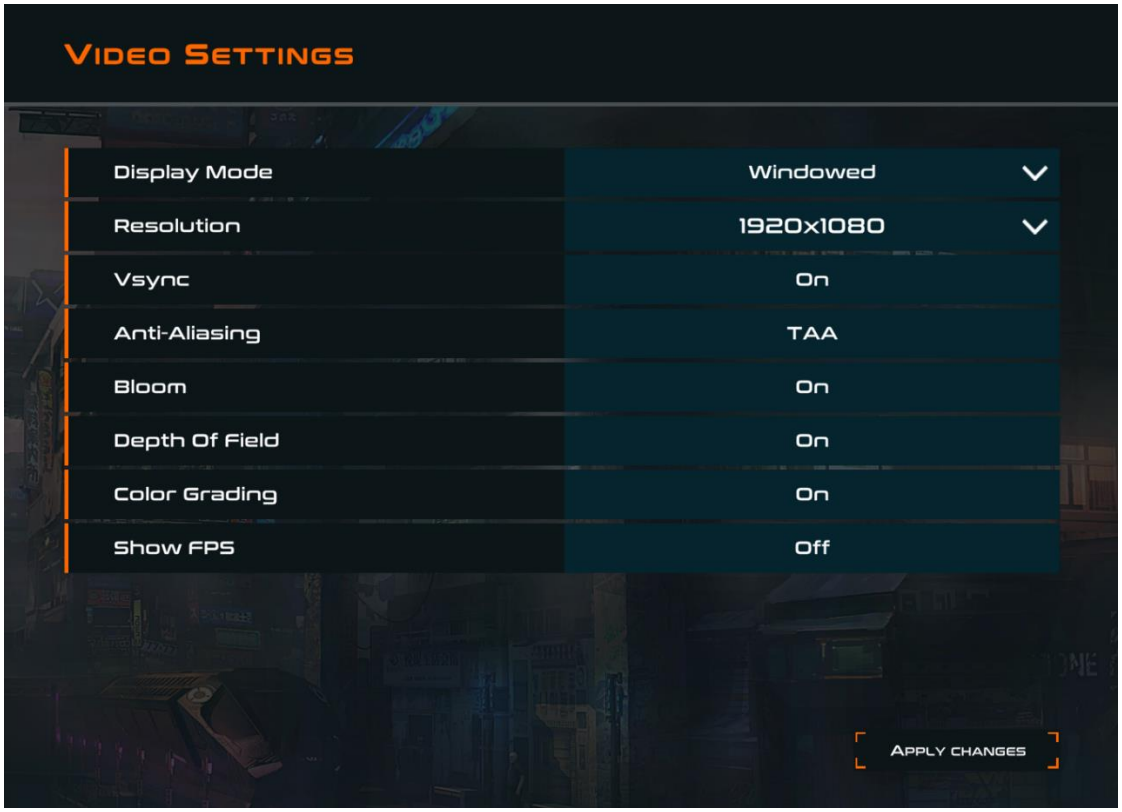
Εικόνα 3 Διεπαφή χρήστη στο αρχικό μενού

Ο χρήστης έχει την επιλογή να ανοίξει τις επιλογές για το παιχνίδι (Play), τις ρυθμίσεις (Options), να δει τους τίτλους (Credits), ή να κλείσει το παιχνίδι (Quit) (Εικόνα 3). Γίνεται αρχικά ανάλυση του μενού ρυθμίσεων.



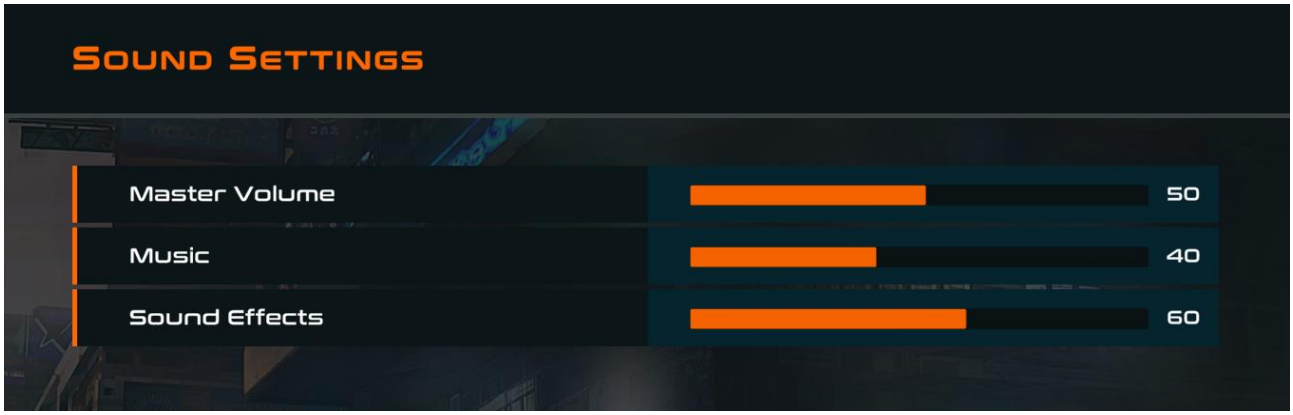
Εικόνα 4 Επιλογές ρυθμίσεων

Αν ο χρήστης διαλέξει να ανοίξει τις ρυθμίσεις, έχει τέσσερις επιλογές, ρυθμίσεις βίντεο, ρυθμίσεις ήχου, ρυθμίσεις χειρισμού, και τέλος να γυρίσει πίσω στο προηγούμενο μενού (Εικόνα 4). Στην πρώτη περίπτωση, ρυθμίσεις βίντεο, υπάρχουν οι εξής ρυθμίσεις (Εικόνα 5):



Εικόνα 5 Ρυθμίσεις βίντεο

Στις ρυθμίσεις βίντεο, ο χρήστης μπορεί να αλλάξει την λειτουργία προβολής και την ανάλυση για να προσαρμόσει το παράθυρο του παιχνιδιού όπως τον ικανοποιεί. Επιπλέον μπορεί να απενεργοποιήσει τις ρυθμίσεις γραφικών Vsync, Anti-Aliasing, Bloom, Depth of Field, και Color Grading, σε περίπτωση που επιβραδύνουν τον υπολογιστή του χρήστη ή επιθυμεί χαμηλότερο επίπεδο γραφικών. Το παιχνίδι έχει βελτιστοποιηθεί ώστε να μην χρειάζεται η απενεργοποίηση αυτών την ρυθμίσεων σε πολλά συστήματα υπολογιστών, αλλά υπάρχει η ελευθερία της επιλογής στον χρήστη. Τέλος, η ενεργοποίηση της επιλογής “Show FPS” εμφανίζει την επίδοση του παιχνιδιού με έναν μετρητή των καρέ ανά δευτερόλεπτο.



Εικόνα 6 Ρυθμίσεις ήχου

Στις ρυθμίσεις ήχου ο χρήστης έχει την επιλογή να μεταποιήσει την γενική ένταση, την ένταση της μουσικής, αλλά και την ένταση των ηχητικών εφέ ξεχωριστά (Εικόνα 6).

Εικόνα 7 Ρυθμίσεις χειρισμού

Στις ρυθμίσεις χειρισμού ο χρήστης έχει την δυνατότητα να αλλάξει τα κουμπιά για κάθε ενέργεια του χαρακτήρα μέσα στο παιχνίδι. Το κουμπί “Reset to Default”(Εικόνα 7) μεταβάλλει όλα τα κουμπιά χειρισμού στις αρχικές τους τιμές. Μια ακόμα ρύθμιση είναι η ευαισθησία του κέρσορα στον οριζόντιο και στον κάθετο άξονα μέσα στο παιχνίδι.

Εικόνα 8 Επιλογές για την εκκίνηση παιχνιδιού

Στην περίπτωση που ο χρήστης ανοίξει τις επιλογές για την εκκίνηση του παιχνιδιού, πρέπει να γίνει η απόφαση για την εκκίνηση του παιχνιδιού με ή χωρίς την διαδικτύωση του παιχνιδιού, και την αναβάθμιση του παίκτη (Upgrades) (Εικόνα 8). Με την πρώτη επιλογή “Singleplayer” ο χρήστης φορτώνει την σκηνή για την αρχή του παιχνιδιού χωρίς να γίνει καμία διαδικτύωση, οπότε παίζει μόνος του.

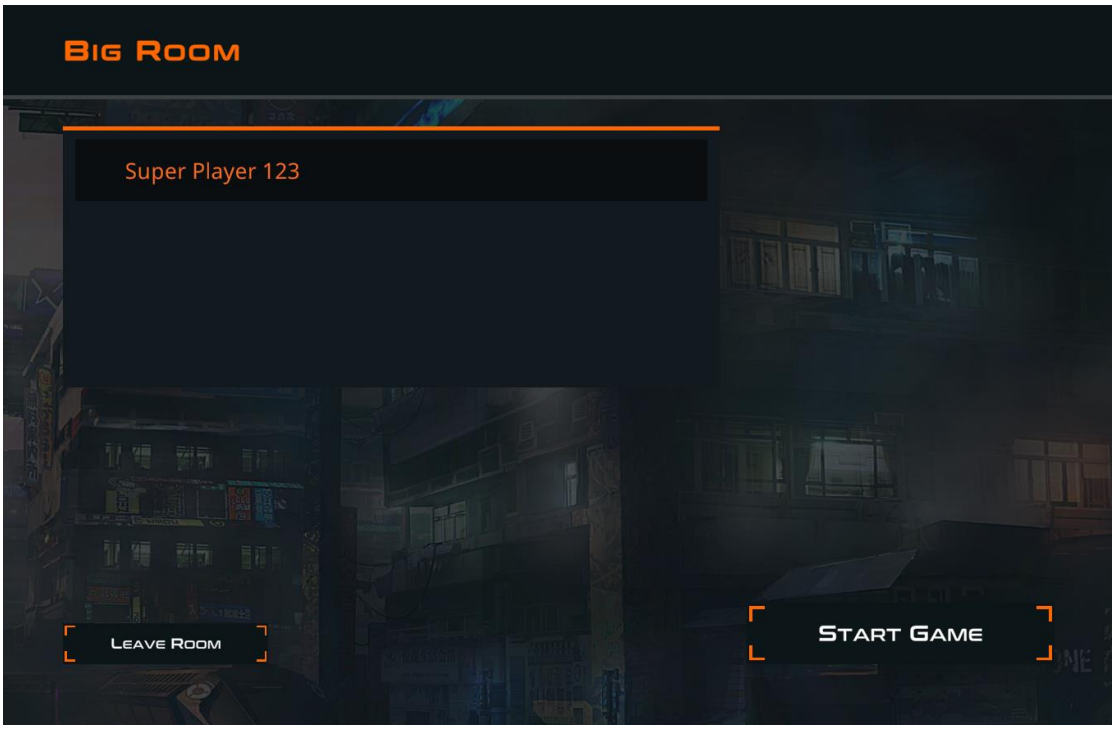
Με την δεύτερη επιλογή “Multiplayer” ο χρήστης μπορεί να αποφασίσει αν θα κάνει host ένα δωμάτιο, δηλαδή να το δημιουργήσει και να είναι ο αρχηγός του δωματίου, ή να κάνει join ένα δωμάτιο ως client, όπου στην προκειμένη περίπτωση, ψάχνει ανοιχτά δωμάτια από άλλους δημιουργούς δωματίων, για να εγγραφεί (Εικόνα 9).

Εικόνα 9 Επιλογές για την δημιουργία ή εγγραφή δωματίου

Άμα ο χρήστης αποφασίσει να δημιουργήσει ένα νέο δωμάτιο, πρέπει να το ονομάσει και να ορίσει τον μέγιστο αριθμό παικτών, μέχρι τέσσερις, που θα μπορούν να συνδεθούν στο δωμάτιο (Εικόνα 10).

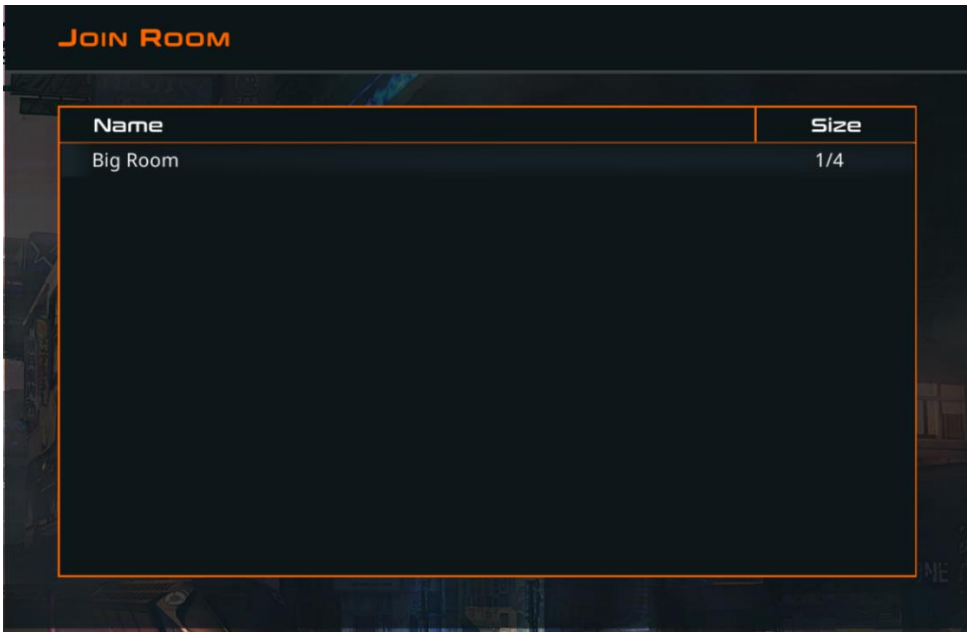
Εικόνα 10 Δημιουργία δωματίου

Με την δημιουργία του δωματίου ο παίκτης εισάγεται στο δωμάτιο και περιμένει να συνδεθούν και άλλοι παίκτες που βρίσκονται στον ίδιο διακομιστή. Το παιχνίδι μπορεί να ξεκινήσει μόνο από τον δημιουργό του δωματίου όταν πατήσει το κουμπί «Start Game» (Εικόνα 11).



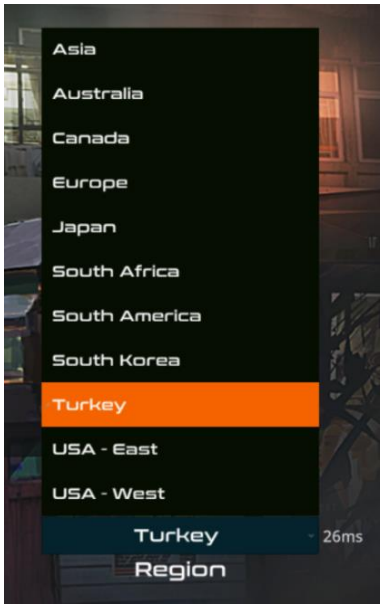
Εικόνα 11 Δωμάτιο που άνοιξε ο host

Στην περίπτωση που ο παίκτης δεν θέλει να δημιουργήσει ένα δωμάτιο αλλά απλά να βρει και να συνδεθεί σε δωμάτιο άλλου, τότε επιλέγει το “Join Room”, και θα του εμφανιστεί μια λίστα με όλα τα διαθέσιμα δωμάτια (Εικόνα 12).



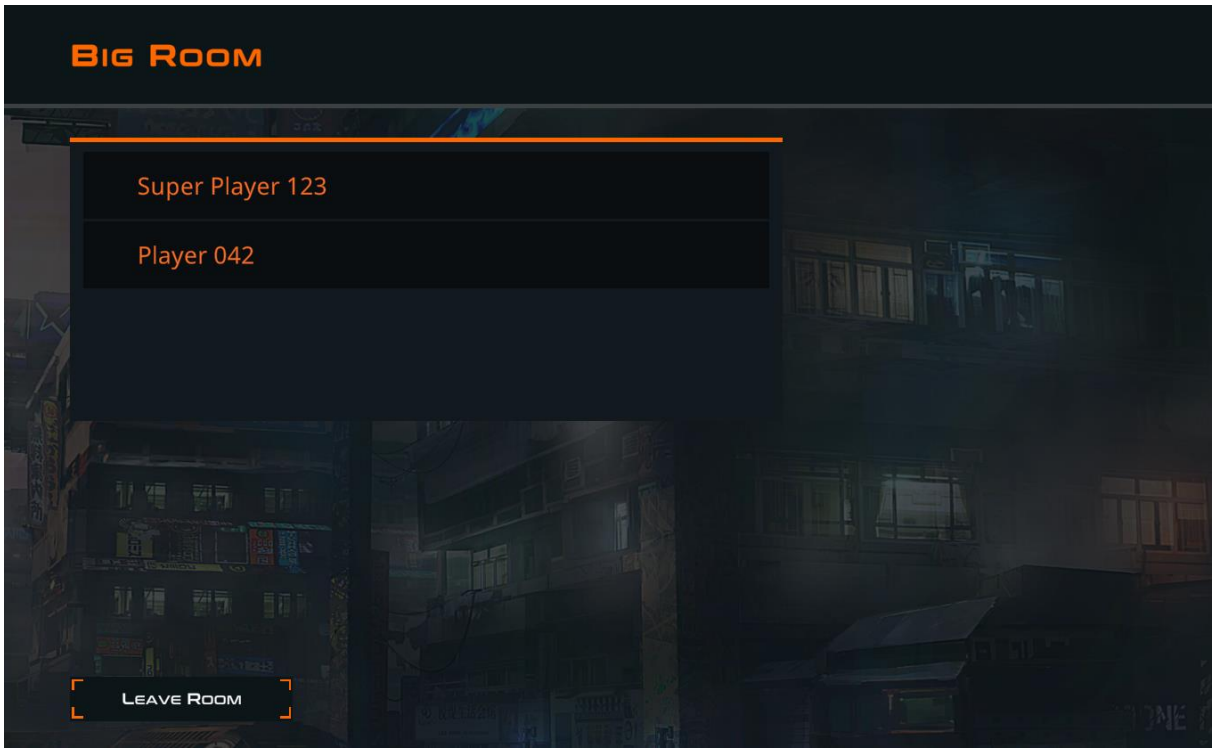
Εικόνα 12 Εύρεση ανοιχτών δωματίων

Πρέπει οι παίκτες που θέλουν να συνδεθούν μαζί, να δώσουν προσοχή στον διακομιστή που είναι συνδεδεμένοι, γιατί αν δεν έχουν κοινό διακομιστή, τότε είναι αδύνατον να βρεθούν μεταξύ τους. Η αλλαγή του διακομιστή βρίσκεται κάτω δεξιά. Η Photon παρέχει διακομιστές παγκοσμίως οπότε υπάρχει καλή πιθανότητα ανάλογα την τοποθεσία των παικτών να συνδεθούν σε διαφορετικούς διακομιστές επειδή πάντα επιλέγεται ο καλύτερος ως προς την καθυστέρηση (ping) διακομιστής. Σε περίπτωση που οι παίκτες επιθυμούν να αλλάξουν τον διακομιστή στον οποίο έχουν συνδεθεί, τους δίνεται η ευκαιρία από την αναπτυσσόμενη λίστα (Εικόνα 13).



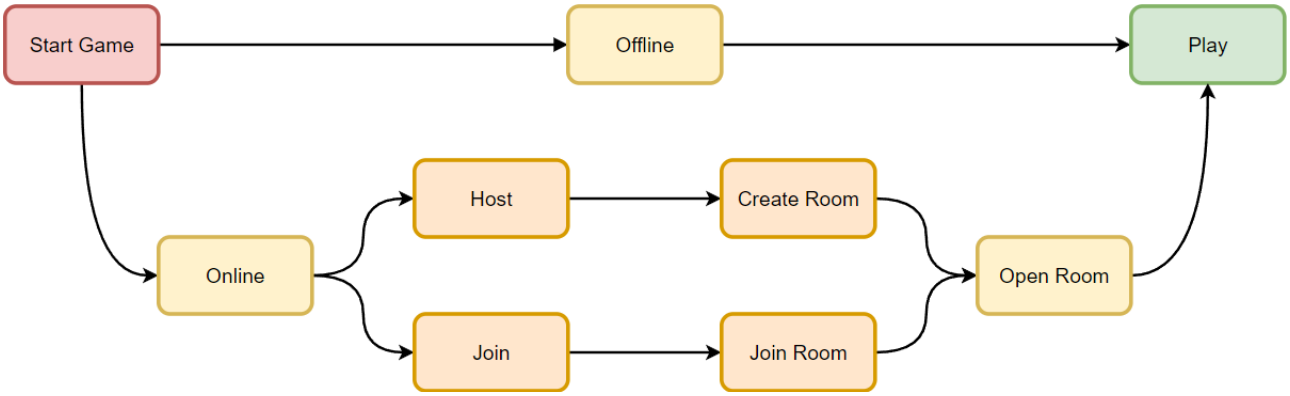
Εικόνα 13 Επιλογές διακομιστή

Στην περίπτωση που ο παίκτης είναι client, όταν συνδέεται σε ένα δωμάτιο, δεν έχει την δυνατότητα να αρχίσει το παιχνίδι, οπότε δεν εμφανίζεται το κουμπί “Start Game” (Εικόνα 14).



Εικόνα 14 Δωμάτιο με παίκτες που συνδέθηκαν στο ίδιο δωμάτιο

Στο διάγραμμα (Εικόνα 15) περιγράφεται συνοπτικά η διαδικασία για την διασύνδεση των παικτών μεταξύ τους.



Εικόνα 15 Διάγραμμα της διασύνδεσης των παικτών

Στην τρίτη επιλογή (Upgrades) ο χρήστης ανοίγει το μενού των upgrades όπου μπορεί να δει όλες τις μόνιμες αναβαθμίσεις που έχει ξεκλειδώσει, τις αναβαθμίσεις που είναι διαθέσιμες, και τα νομίσματα που έχει συλλέξει. Δίνεται επίσης η δυνατότητα να αναιρέσει όλες τις αναβαθμίσεις που ξεκλείδωσε άμα επιθυμεί να αλλάξει τις επιλογές του (Εικόνα 16).

Εικόνα 16 Μενού Upgrades

3.10.2 Αρένα

Όταν οι παίκτες μπουν μέσα στην αρένα και ουσιαστικά αρχίσει το παιχνίδι, εμφανίζονται πολλά στοιχεία της διεπαφής χρήστη που αναλύουμε παρακάτω (Εικόνα 17):

Εικόνα 17 Διεπαφή παίκτη στο Singleplayer

(1) Πόντοι ζωής του παίκτη, και ακριβώς από πάνω αναγράφεται το όνομα του παίκτη. Κάτω ακριβώς από τους πόντους ζωής εμφανίζεται το κουμπί που μπορεί να πατήσει ο παίκτης για να δει τα στατιστικά του.
(2) Νομίσματα που συλλέγει ο παίκτης για την αγορά των upgrades. Από κάτω υπολογίζεται το ping και το RTT (Round Trip Time) για να κρατάμε μετρικές τις οποίες μπορεί να συμβουλευτεί ο παίκτης σε περίπτωση που νιώσει πως υπάρχει καθυστέρηση στο παιχνίδι. Αν οι τιμές αυτές είναι υψηλές τότε ο παίκτης μπορεί να αντιληφθεί την καθυστέρηση του δικτύου.
(3) Κουμπί για τον χειρισμό του Dash.
(4) Εξοπλισμός όπλων, υπάρχει ένα πλαίσιο για το κάθε όπλο που μπορεί να κρατήσει ο παίκτης. Επίσης εμφανίζονται τα κουμπιά για το προηγούμενο και επόμενο όπλο.
(5) Κουμπιά για τον χειρισμό της επίθεσης με το σπαθί (melee) και για τον χειρισμό της εγκατάλειψης του όπλου.

Εικόνα 18 Διεπαφή παίκτη στο Multiplayer

Στην περίπτωση του Multiplayer η διεπαφή χρήστη είναι ίδια, με την διαφορά ότι πάνω αριστερά, κάτω από τους πόντους ζωής του παίκτη, εμφανίζονται οι πόντοι ζωής και τα ονόματα των υπόλοιπων παικτών που βρίσκονται στο κοινό παιχνίδι (Εικόνα 18).

Υπάρχουν δείκτες για να ενημερώνουν τον παίκτη με την καθυστέρηση μιας ενέργειας όπως το γέμισμα του όπλου και το dash (Εικόνα 19).

Εικόνα 19 Δείκτες καθυστέρησης γεγονότων

Υπάρχουν επίσης στοιχεία της διεπαφής χρήστη που εμφανίζονται μόνο υπό συγκεκριμένες συνθήκες. Πρόκειται για την καρτέλα με τα στοιχεία του όπλου όταν ο παίκτης πλησιάσει κοντά (Εικόνα 20) όπως και η καρτέλα που περιγράφει την επίδραση της επαύξησης (Εικόνα 21).

Εικόνα 20 Καρτέλα με τα στοιχεία του όπλου

Εικόνα 21 Καρτέλα με την περιγραφή της επαύξησης

3.11 Επίπεδα

Υπάρχει ένα επίπεδο στο παιχνίδι η αρένα, στην οποία εκτυλίσσονται οι δέκα γύροι που περιλαμβάνει. Υπάρχουν 5 κατηγορίες για τα είδη των γύρων, το Exterminate, το Defense, το Parkour, το Survival και το Rampage. Για το μοίρασμα των γύρων ισχύουν οι εξής κανόνες:

1. Ό πρώτος γύρος είναι πάντα Exterminate για να εισάγει τους παίκτες στο παιχνίδι.
2. Μόνο ο τελευταίος γύρος είναι του είδους Rampage, και είναι ο πιο δύσκολος καθώς αποτελεί την τελευταία πρόκληση των παικτών.
3. Οι ενδιάμεσοι γύροι αποφασίζονται τυχαία, περιλαμβάνοντας πάντα το πολύ δύο Defense, δύο Survival, δύο Parkour, και ότι απομένει γίνεται Exterminate.
4. Τέλος, γίνεται ο απαραίτητος έλεγχος ώστε να μην υπάρχουν δύο διαδοχικοί γύροι του ίδιο είδους, με εξαίρεση το Exterminate και το Rampage.

Το παιχνίδι κρατάει μια μεταβλητή ως χρήματα για να «αγοράζει» τους εχθρούς που θα δημιουργεί και το ποσό των χρημάτων αυξάνεται με κάθε γύρο. Κάθε εχθρός ανάλογα με την δυσκολία του έχει το δικό του κόστος. Έτσι για κάθε γύρο που περιέχει εχθρούς το παιχνίδι αποφασίζει τυχαία ποιοι θα είναι οι εχθροί αυτοί, μειώνοντας τα χρήματα που διαθέτει με βάση το κόστος τους. Αυτό επιτυγχάνει μια ισορροπημένη δυσκολία όσον αφορά τον αριθμό των εχθρών κάθε είδους με τους οποίους θα έρθουν αντιμέτωποι οι παίκτες.

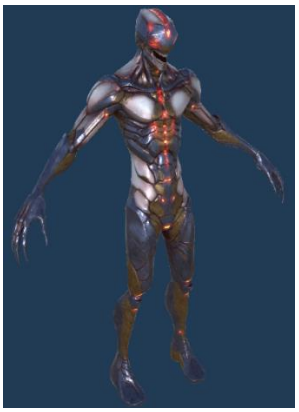
Τα στατιστικά των εχθρών, όπως οι πόντοι ζωής, οι πόντοι ζημιάς, και τα νομίσματα, κλιμακώνονται ανάλογα τον αριθμό των παικτών που βρίσκονται στο ίδιο παιχνίδι, και τον γύρο που έχουν καταφέρει να φτάσουν.

3.12 Τέχνη

Όλα τα 3D μοντέλα είναι επιλεγμένα από το κατάστημα Unity Asset Store, και την σελίδα Sketchfab. Το μοντέλο του παίκτη επιλέγεται τυχαία κατά την εκκίνηση του παιχνιδιού (Εικόνα 22). Τα μοντέλα είναι rigged, δηλαδή έχουν τον σκελετό για την προσθήκη των animations.

Εικόνα 22 Μερικά από τα μοντέλα των παικτών

Το μοντέλο του βασικού εχθρού (Εικόνα 23) περιέχει animation για το τρέξιμο, και το χτύπημα. Επίσης περιλαμβάνει ηχητικό εφέ κάθε φορά που χτυπιέται.



Εικόνα 23 Βασικός εχθρός

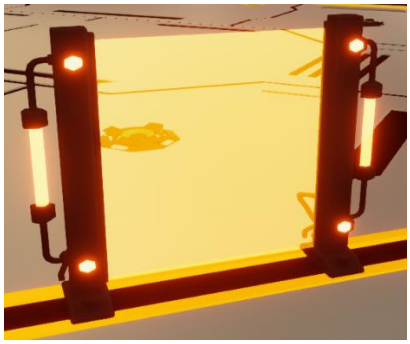
Το μοντέλο του ιπτάμενου εχθρού (Εικόνα 24) είναι το μόνο μοντέλο με χειροποίητα animations για την πτήση προς μια κατεύθυνση, και την επίθεση. Περιλαμβάνει ηχητικό εφέ κάθε φορά που χτυπιέται.

Εικόνα 24 Ιπτάμενος εχθρός

Έξω από την αρένα υπάρχει μια μεγάλη οθόνη (Εικόνα 25), για να μπορεί να ενημερωθεί ο παίκτης για τον γύρο στον οποίο βρίσκεται ανα πάσα στιγμή.

Εικόνα 25 Οθόνη περιγραφής του γύρου

Μέσα στην αρένα υπάρχουν μερικοί τοίχοι προστασίας (Εικόνα 26) τους οποίους δεν μπορούν να διαπεράσουν οι βασικοί εχθροί, ούτε οι παίκτες, ούτε και τα πυρά, είτε αυτά είναι φιλικά είτε εχθρικά.



Εικόνα 26 Τοίχος προστασίας

Στην αρχή του παιχνιδιού, καθώς και στο τέλος από κάθε γύρο, εμφανίζεται μια οθόνη (Εικόνα 27) με την οποία μπορεί να αλληλεπιδράσει ο παίκτης για να ξεκινήσει τον επόμενο γύρο.

Εικόνα 27 Οθόνη για την εκκίνηση του επόμενου γύρου

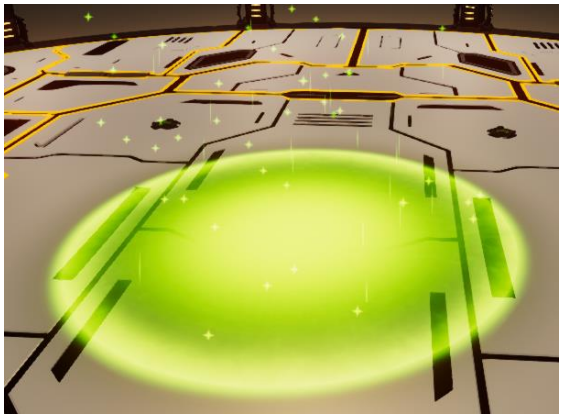
Η αρένα στην οποία λαμβάνει μέρος όλο το παιχνίδι (Εικόνα 28). Είναι αρκετά φωτισμένη, με πολλές κολώνες να περικλείουν το περιβάλλον. Είναι άδεια εξαρχής, αλλά γεμίζει με αντικείμενα, και άλλα μοντέλα περιβάλλοντος κατά την διάρκεια του παιχνιδιού.

Εικόνα 28 Ολόκληρο το περιβάλλον της αρένας

Στον γύρο Defense, εμφανίζεται στο κέντρο της αρένας το μοντέλο από τον κρύσταλλο (Εικόνα 29) που πρέπει να προστατέψουν οι παίκτες από τους εχθρούς.

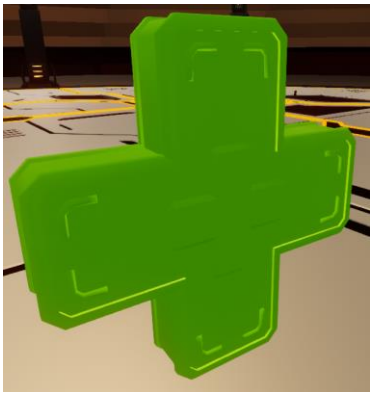
Εικόνα 29 Κρύσταλλος που πρέπει να προστατέψουν οι παίκτες

Στους γύρους Survival και Rampage, εμφανίζεται περιοδικά μια ζώνη επούλωσης (Εικόνα 30) όπου μπορούν να μπουν μέσα οι παίκτες για να αποκτήσουν πόντους ζωής για κάθε δευτερόλεπτο που βρίσκονται εσωτερικά της.



Εικόνα 30 Ζώνη επούλωσης

Αποκλειστικά στον γύρο Survival, εμφανίζονται στην αρχή του γύρου μερικά αντικείμενα (Εικόνα 31) που μπορούν να προσθέσουν πόντους ζωής στου παίκτες αν έρθουν σε επαφή με αυτά.



Εικόνα 31 Αντικείμενο με χαμηλή θεραπευτική ιδιότητα

Όταν χάνουν οι εχθροί, υπάρχει η πιθανότητα να αφήσουν πίσω τους κάποιο power up (Εικόνα 32), όπου το καθένα έχει την δική του επίδραση στον παίκτη. Το χρώμα περιγράμματος του κάθε power up αλλάζει ανάλογα με την σπανιότητά του.

Εικόνα 32 Μοντέλα των power ups

Με κάθε άνοιγμα από το κουτί όπλων, οι παίκτες λαμβάνουν τυχαία ένα από το σύνολο των όπλων που υπάρχουν στο παιχνίδι (Εικόνα 33). Το κάθε όπλο έχει τις δικές του ιδιότητες, και επίσης αλλάζει το χρώμα του περιγράμματός του ανάλογα με την σπανιότητά του.

Εικόνα 33 Μοντέλα όπλων

3.13 Ηχητικά εφέ και Μουσική

Όλοι οι ήχοι του παιχνιδιού περνάνε από επεξεργασία μέσω του μίκτη ήχου για να μπορεί να γίνει ο διαχωρισμός της έντασης μεταξύ της μουσικής και των ηχητικών εφέ. Υπάρχει και η επιλογή της ρύθμισης της γενικής έντασης που μεταβάλλει την ένταση για την μουσική και τα ηχητικά εφέ μαζί. Τα ηχητικά εφέ προστέθηκαν από την σελίδα ZapSplat, και για τα ηχητικά εφέ που χρειαζόντουσαν τροποποιήσεις, χρησιμοποιήθηκε το λογισμικό Audacity. Τα μουσικά κομμάτια του παιχνιδιού είναι τα εξής:

1. Victory Track: Neon Glow – White Bat Audio
2. Battle Theme 1: Kill Crew – White Bat Audio
3. Battle Theme 2: Atomic Punk – White Bat Audio
4. Battle Theme 3: Overkill – White Bat Audio
5. Battle Theme 4: Ultra Meta – White Bat Audio

Στο επόμενο κεφάλαιο γίνεται η αναλυτική επεξήγηση για την υλοποίηση του παιχνιδιού. Παρουσιάζονται οι κώδικες, και οι μέθοδοι που χρησιμοποιήθηκαν για την ολοκλήρωση του παιχνιδιού αλλά και την επίτευξη της υψηλής επίδοσης, χαμηλής χωρητικότητας, και την ελάχιστη καθυστέρηση σε διαδικτυακά μηνύματα μεταξύ των παικτών.

Κεφάλαιο 4: Υλοποίηση

Στο κεφάλαιο αυτό γίνεται η ανάλυση των σύνθετων προβλημάτων που αντιμετωπίστηκαν κατά την υλοποίηση του παιχνιδιού. Αυτό συμπεριλαμβάνει τον κώδικα για την διασύνδεση των παικτών, τον συγχρονισμό των μεταβλητών στο δίκτυο, την βελτιστοποίηση της απόδοσης του παιχνιδιού, την μείωση του αποθηκευτικού χώρου και την διαχείριση του φωτισμού. Το παιχνίδι περιέχει δύο σκηνές, μία για το αρχικό μενού και μία για την αρένα. Στο αρχικό μενού μπορεί να γίνει η εύρεση και διασύνδεση των παικτών (matchmaking).

4.1 Matchmaking

Τα scripts που χρησιμοποιήθηκαν για την διασύνδεση των παικτών είναι τα: MatchmakingLobbyController.cs, MatchmakingRoomController.cs, RoomButton.cs, Menu.cs και RegionScript.cs. Το script Menu.cs είναι υπεύθυνο για τις λειτουργίες του αρχικού μενού και την σύνδεση στο δίκτυο. Με το MatchmakingLobbyController.cs γίνεται η εύρεση και με το RoomButton.cs η εμφάνιση των δωματίων. Στο MatchmakingRoomController.cs γίνεται η διαχείριση του δωματίου. Το RegionScript.cs είναι υπεύθυνο μόνο για την οπτική εμφάνιση των διαθέσιμων διακομιστών.

4.1.1 Διαχείριση Μενού

Το πακέτο της Photon παρέχει την μεταβλητή *OfflineMode* για να ορίσουμε αν το παιχνίδι χρειάζεται να συνδεθεί στους διακομιστές της Photon, ή αν ο παίκτης θα παίξει μόνος του. Στην περίπτωση που ο παίκτης θελήσει να παίξει χωρίς να συνδεθεί με άλλους παίκτες, τότε μόλις πατήσει το κουμπί “Singleplayer” (Εικόνα 8) ορίζουμε την μεταβλητή *OfflineMode* σε true, ενεργοποιούμε την μπάρα φόρτωσης, και ξεκινά το παιχνίδι (Εικόνα 34). Άμα όμως αποφασίσει να διασυνδεθεί με άλλους παίκτες, τότε όταν πατήσει το κουμπί “Multiplayer” (Εικόνα 8) εκτελείται ο κατάλληλος κώδικας για να οριστεί η μεταβλητή *OfflineMode* σε false, γίνεται η σύνδεση με τον διακομιστή της Photon με την εντολή *PhotonNetwork.ConnectUsingSettings(),* και αρχίζουμε την χρονομέτρηση σε περίπτωση που είναι αδύνατη η επικοινωνία με τον διακομιστή για να εμφανίσουμε το κατάλληλο μήνυμα (Εικόνα 35).

```
88      public void StartSinglePlayer()
89      {
90          PhotonNetwork.OfflineMode = true;
91
92          mainMenu.SetActive(false);
93          loadingScreen.SetActive(true);
94          StartCoroutine(LoadLevelAsync(gameSceneIndex));
95      }
```

Εικόνα 34 Αρχή του παιχνιδιού σε Singleplayer

```
97      public void StartMultiPlayer()
98      {
99          PhotonNetwork.OfflineMode = false;
100         PhotonNetwork.ConnectUsingSettings();
101         connectToServerLoading.SetActive(true);
102
103         timeoutTimer = 0;
104         startTimer = true;
105     }
106
```

Εικόνα 35 Σύνδεση στον διακομιστή της Photon

Πρώτα, για να είναι δυνατή η χρήση των συναρτήσεων και ανακλήσεων της Photon, πρέπει η κλάση του κώδικα να κληρονομεί την κλάση MonoBehaviourPunCallbacks της Photon. Η συνάρτηση *PhotonNetwork.ConnectUsingSettings*() συνδέεται στον διακομιστή της Photon με τις προεπιλεγμένες ρυθμίσεις του παιχνιδιού, και καλεί αυτόματα την συνάρτηση *OnConnectedToMaster()* στην επιτυχή σύνδεση. Για να προσαρμόσουμε τον κώδικα και να προσθέσουμε τις δικές μας εντολές, κάνουμε override όποια συνάρτηση καλείται αυτόματα (ανάκληση) από το πακέτο της Photon. Στην προκειμένη περίπτωση, κάνουμε override την συνάρτηση OnConnectedToMaster(), ώστε όταν γίνεται η επιτυχής σύνδεση στον διακομιστή της Photon, να αποθηκεύεται το όνομα του παίκτη, ή αν δεν υπάρχει, να δημιουργείται ένα τυχαίο (Εικόνα 36). Πρέπει επίσης να εκτελεστεί η εντολή *PhotonNetwork.AutomaticallySyncScene* σε true, ώστε οι παίκτες να φορτώσουν την ίδια σκηνή όταν αρχίσει το παιχνίδι. Στο τέλος καλούμε την συνάρτηση *PhotonNetwork.JoinLobby()* για να αναθέσει τον παίκτη σε μια από τις αίθουσες (lobby).

```
131     public override void OnConnectedToMaster()
132     {
133         if (PhotonNetwork.OfflineMode == false)
134         {
135             PhotonNetwork.AutomaticallySyncScene = true;
136 
137             serverPanel.SetActive(true);
138             regionDropdown.SetActive(true);
139 
140             PhotonNetwork.NickName = playerNameInput.text;
141             PlayerPrefs.SetString("NickName", playerNameInput.text);
142 
143             if (PlayerPrefs.HasKey("NickName"))
144             {
145                 if (PlayerPrefs.GetString("NickName") == "")
146                 {
147                     PhotonNetwork.NickName = "Player " + Random.Range(0, 1000);
148                 }
149                 else
150                 {
151                     PhotonNetwork.NickName = PlayerPrefs.GetString("NickName");
152                 }
153             }
154             else
155             {
156                 PhotonNetwork.NickName = "Player " + Random.Range(0, 1000);
157             }
158 
159             PhotonNetwork.JoinLobby();
160         }
161         else
162         {
163             PhotonNetwork.NickName = playerNameInput.text;
164             PhotonNetwork.JoinRandomOrCreateRoom();
165         }
166     }
```

Εικόνα 36 Override της συνάρτησης OnConnectedToMaster()

Άμα δεν γίνει η επιτυχής σύνδεση στον διακομιστή της Photon μετά το χρονικό διάστημα που έχει οριστεί, τότε εμφανίζουμε το μήνυμα της αποτυχίας σύνδεσης (Εικόνα 37).

```
53      public void Update()
54      {
55          if (enterNamePanel.activeSelf == true)
56          {
57              if (playerNameInput.text != "")
58              {
59                  nameConfirmButton.interactable = true;
60                  nameConfirmHover.enabled = true;
61              }
62              else
63              {
64                  nameConfirmButton.interactable = false;
65                  nameConfirmHover.enabled = false;
66              }
67          }
68          if (startTimer)
69          {
70              timeoutTimer += Time.deltaTime;
71          }
72          if (timeoutTimer >= 35f)
73          {
74              startTimer = false;
75              timeoutTimer = 0;
76              connectToServerLoading.SetActive(false);
77              failedToConnect.SetActive(true);
78          }
79      }
```

Εικόνα 37 Εμφάνιση μηνύματος timeout

Είναι σημαντικό να σημειωθεί πως ο διακομιστής που επιλέγεται εξαρτάται από την σύνδεση του παίκτη, οπότε επιλέγεται αυτόματα ο διακομιστής με την καλύτερη σύνδεση. Όμως ο παίκτης έχει την δυνατότητα να αλλάξει διακομιστή, εάν το επιθυμεί (Εικόνα 38). Το Enum *CloudRegionCode* περιέχει τις τιμές που αποδέχεται η συνάρτηση *PhotonNetwork.ConnectToRegion* για να αναγνωρίσει τον διακομιστή που επέλεξε να συνδεθεί ο παίκτης. Για να γίνει η εναλλαγή του διακομιστή, πρώτα αποσυνδεόμαστε από τον διακομιστή της Photon με την συνάρτηση *PhotonNetwork.Disconnect()*, αποθηκεύουμε την επιλογή του παίκτη στην μεταβλητή *regionSwitchCode* και γίνεται επανασύνδεση στον διακομιστή δίνοντας την παράμετρο *regionSwitchCode*.

```
187     [SerializeField] public enum CloudRegionCode { asia, au, cae, eu, jp, za, sa, kr, tr, us, usw, none }
188
        0 references
189     public void ChangeRegion(int dropdownOption)
190     {
191         PhotonNetwork.Disconnect();
192         CloudRegionCode regionSwitchCode = (CloudRegionCode)System.Enum.Parse(typeof(CloudRegionCode), dropdownOption.ToString());
193         PhotonNetwork.ConnectToRegion(regionSwitchCode.ToString());
194
195         connectToServerLoading.SetActive(true);
196
197         timeoutTimer = 0;
198         startTimer = true;
199     }
```

Εικόνα 38 Εναλλαγή του διακομιστή

4.1.2 Διαχείριση αίθουσας

Όταν ο παίκτης συνδεθεί σε μια αίθουσα, γίνεται ανάκληση της συνάρτησης *OnJoinedLobby()*. Σε αυτό το σημείο ο παίκτης έχει την επιλογή να δημιουργήσει ή να συνδεθεί σε ένα δωμάτιο. Αρχικά, άμα ο παίκτης δημιουργήσει ένα δωμάτιο πατώντας το κουμπί “Host Room” εμφανίζεται η εισαγωγή των στοιχείων του δωματίου (Εικόνα 10) και με την επιλογή “Create room” εκτελείται η συνάρτηση *CreateRoom()* θέτοντας το δωμάτιο ως ανοιχτό και

διαθέσιμο στην κατάλληλη μεταβλητή της κλάσης *RoomOptions* της Photon, όπου εισάγουμε τα στοιχεία του δωματίου (Εικόνα 39). Τέλος, δημιουργούμε το δωμάτιο με την συνάρτηση *PhotonNetwork.CreateRoom()* με παραμέτρους τις ρυθμίσεις του δωματίου και το όνομα που επέλεξε ο παίκτης.

```
120     public void CreateRoom()
121     {
122         RoomOptions roomOps = new RoomOptions() { IsVisible = true, IsOpen = true, MaxPlayers = (byte)roomSize };
123         PhotonNetwork.CreateRoom(roomName, roomOps);
124     }
```

Εικόνα 39 Δημιουργία δωματίου

Στην δεύτερη περίπτωση όπου ο παίκτης θα πατήσει το κουμπί “Join Room” (Εικόνα 10) λαμβάνονται όλες οι πληροφορίες και οι ενημερώσεις για τα ανοιχτά δωμάτια με την ανάκληση της συνάρτησης *OnRoomListUpdate()* (Εικόνα 40) την οποία κάνουμε override για να αποθηκεύσουμε τις πληροφορίες των δωματίων. Για το κάθε δωμάτιο που δεν είναι άδειο, το εμφανίζουμε με την συνάρτηση *ListRoom()* (Εικόνα 41).

```
46      public override void OnRoomListUpdate(List<RoomInfo> roomList)
47      {
48          int tempIndex;
49          foreach (RoomInfo room in roomList)
50          {
51              if (roomListings != null)
52              {
53                  tempIndex = roomListings.FindIndex(ByName(room.Name));
54              }
55              else
56              {
57                  tempIndex = -1;
58              }
59              if (tempIndex != -1)
60              {
61                  roomListings.RemoveAt(tempIndex);
62                  Destroy(roomsContainer.GetChild(tempIndex).gameObject);
63              }
64              if (room.PlayerCount > 0)
65              {
66                  roomListings.Add(room);
67
68                  ListRoom(room);
69              }
70          }
71      }
72
        1 reference
73      static System.Predicate<RoomInfo> ByName(string name)
74      {
75          return delegate (RoomInfo room)
76          {
77              return room.Name == name;
78          };
79      }
```

Εικόνα 40 Override της συνάρτησης OnRoomListUpdate()

```
81      void ListRoom(RoomInfo room)
82      {
83          if (room.IsOpen && room.IsVisible)  // Checking if room is open and visible
84          {
85              GameObject tempListing = Instantiate(roomListingPrefab, roomsContainer);    // Creating room listing UI
86              RoomButton tempButton = tempListing.GetComponent<RoomButton>();
87              tempButton.SetRoom(room.Name, room.MaxPlayers, room.PlayerCount);           // Setting room information
88          }
89      }
```

Εικόνα 41 Εμφάνιση δωματίου

Μόλις ο παίκτης βρει και πατήσει το δωμάτιο που θέλει να συνδεθεί, καλείται η συνάρτηση *JoinRoomOnClick()* από το script RoomButton.cs (Εικόνα 42) όπου εκτελείται η εντολή *PhotonNetwork.Join Room(roomName)*. Μόλις επιτευχθεί η σύνδεση με το δωμάτιο γίνεται η ανάκληση της συνάρτησης *OnJoinedRoom()*.

```
16      public void JoinRoomOnClick()
17      {
18          try
19          {
20              PhotonNetwork.JoinRoom(roomName);
21          }
22          catch (Exception e)
23          {
24              Debug.Log("Unable to join room");
25          }
26      }
```

Εικόνα 42 Σύνδεση σε δωμάτιο

Για να επιτευχθεί η ανανέωση της λίστας με τα διαθέσιμα δωμάτια, τότε αποσυνδεόμαστε από την αίθουσα, και συνδεόμαστε ξανά (Εικόνα 44). Αυτό όμως μπορεί να προκαλέσει τον διπλασιασμό των δωματίων επειδή γίνεται η ανάκληση της συνάρτησης *OnRoomListUpdate()*, οπότε είναι σημαντικό να διαγράψουμε όλα τα παλαιότερα δωμάτια προτού τα ανανεώσουμε με την συνάρτηση *OnJoinedLobby()* (Εικόνα 43).

```
25      public override void OnJoinedLobby()
26      {
27          foreach (Transform trans in roomsContainer)
28          {
29              Destroy(trans.gameObject);
30          }
31          roomListings.Clear();
32      }
```

Εικόνα 43 Override της συνάρτησης OnJoinedLobby()

```
34      public void RefreshLobby()
35      {
36          PhotonNetwork.LeaveLobby();
37          StartCoroutine(rejoinLobby());
38      }
39
        1 reference
40      IEnumerator rejoinLobby()
41      {
42          yield return new WaitForSeconds(1);
43          PhotonNetwork.JoinLobby();
44      }
```

Εικόνα 44 Επανασύνδεση στην αίθουσα

4.1.3 Διαχείριση δωματίου

Την στιγμή που ο παίκτης συνδεθεί σε ένα δωμάτιο, γίνεται η αναμενόμενη ανάκληση *OnJoinedRoom()* (Εικόνα 45). Στην συνάρτηση αυτή για να εμφανίσουμε το όνομα του δωματίου καλούμε την συνάρτηση *PhotonNetwork.CurrentRoom.Name*. Επίσης ενεργοποιείται η δυνατότητα της εκκίνησης του παιχνιδιού μόνο στον host. Μετέπειτα διαγράφουμε τους παίκτες με την συνάρτηση *ClearPlayerListings*(), και τους εμφανίζουμε με την συνάρτηση *ListPlayers()* (Εικόνα 46). Ο λόγος που γίνεται πρώτα η διαγραφή, είναι για να αποφύγουμε την περίπτωση του

διπλασιασμού των παικτών, όπως γίνεται αντίστοιχα και στην περίπτωση των δωματίων όταν βρισκόμαστε σε μια αίθουσα.

```
54      public override void OnJoinedRoom()
55      {
56          if (PhotonNetwork.OfflineMode == false)
57          {
58              roomPanel.SetActive(true);
59              lobbyPanel.SetActive(false);
60              roomNameDisplay.text = PhotonNetwork.CurrentRoom.Name;
61
62              startButton.SetActive(PhotonNetwork.IsMasterClient);
63
64              ClearPlayerListings();
65              ListPlayers();
66          }
67      }
```

Εικόνα 45 Override της συνάρτησης OnJoinedRoom

```
36      void ClearPlayerListings()
37      {
38          for (int i = playersContainer.childCount - 1; i >= 0; i--)
39          {
40              Destroy(playersContainer.GetChild(i).gameObject);
41          }
42      }
43
        3 references
44      void ListPlayers()
45      {
46          foreach (Player player in PhotonNetwork.PlayerList)
47          {
48              GameObject tempListing = Instantiate(playerListingPrefab, playersContainer);
49              TMP_Text tempText = tempListing.transform.GetChild(0).GetComponent<TMP_Text>();
50              tempText.text = player.NickName;
51          }
52      }
```

Εικόνα 46 Διαγραφή και εμφάνιση παικτών

Στην εμφάνιση παικτών χρησιμοποιούμε την μεταβλητή PlayerList της PhotonNetwork, η οποία μας γυρίζει την λίστα από όλους τους παίκτες που είναι συνδεδεμένοι στο ίδιο δωμάτιο. Άμα συνδεθεί ή αποσυνδεθεί ένας καινούργιος παίκτης, γίνεται η ανάκληση των συναρτήσεων *OnPlayerEnteredRoom()* και *OnPlayerLeftRoom(),* συναρτήσεις τις οποίες κάνουμε override για να ανανεώσουμε την διεπαφή χρήστη με τους παίκτες που βρίσκονται στο δωμάτιο (Εικόνα 47).

```
69      public override void OnPlayerEnteredRoom(Player newPlayer)
70      {
71          ClearPlayerListings();
72          ListPlayers();
73      }
74
        2 references
75      public override void OnPlayerLeftRoom(Player otherPlayer)
76      {
77          ClearPlayerListings();
78          ListPlayers();
79          if (PhotonNetwork.IsMasterClient)
80          {
81              startButton.SetActive(true);
82          }
83      }
```

Εικόνα 47 Override των συναρτήσεων εισχώρησης και αποχώρησης παίκτη

Όταν οι παίκτες είναι έτοιμοι να ξεκινήσουν το παιχνίδι, ο host μπορεί να το αρχίσει πατώντας το κουμπί “Start Game”. Το δωμάτιο κλείνει θέτοντας την μεταβλητή *PhotonNetwork.CurrentRoom.IsOpen* σε false, και ο host αρχίζει να φορτώνει την νέα σκηνή με την συνάρτηση *LoadLevelAsync().* Ταυτόχρονα στέλνεται ένα event με κωδικό το *ChangeSceneCode* προς τους clients, με παράμετρο τον δείκτη της σκηνής, για να εμφανιστεί η οθόνη φόρτωσης σε όλους, αλλά με κάθε παίκτη να έχει την δικιά του μπάρα φόρτωσης. Επίσης εκεί καλείται η συνάρτηση *LoadClientLevelAsync()* (Εικόνα 48) για να φορτώσει η νέα σκηνή στους clients. Η Photon μας παρέχει την μεταβλητή *PhotonNetwork.LevelLoadingProgress* για να γνωρίζουμε σε πιο στάδιο βρίσκεται η φόρτωση της σκηνής, οπότε την χρησιμοποιούμε για να το εμφανίσουμε στους παίκτες με την μορφή της μπάρας φόρτωσης.

```
85      public void StartGame()
86      {
87          if (PhotonNetwork.IsMasterClient)
88          {
89              // Adding the variable we want to send
90              object[] content = new object[] { multiplayerSceneIndex };
91              // Setting the target of the event to Others
92              RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.Others };
93              // Sending the event with the correct code, parameter, target, and message importance option
94              PhotonNetwork.RaiseEvent(ChangeSceneCode, content, raiseEventOptions, SendOptions.SendReliable);
95
96              PhotonNetwork.CurrentRoom.IsOpen = false;
97
98              mainMenu.SetActive(false);
99              loadingScreen.SetActive(true);
100
101             StartCoroutine(LoadLevelAsync(multiplayerSceneIndex));
102         }
103     }
104     // Host load scene
        1 reference
105     IEnumerator LoadLevelAsync(int sceneIndex)
106     {
107         PhotonNetwork.LoadLevel(sceneIndex);
108
109         while (PhotonNetwork.LevelLoadingProgress < 1)
110         {
111             float progress = Mathf.Clamp01(PhotonNetwork.LevelLoadingProgress / 0.9f);
112             loadingBar.fillAmount = progress;
113             yield return null;
114         }
115     }
116     // Clients load scene
        1 reference
117     IEnumerator LoadClientLevelAsync()
118     {
119         while (PhotonNetwork.LevelLoadingProgress < 1)
120         {
121             float progress = Mathf.Clamp01(PhotonNetwork.LevelLoadingProgress / 0.9f);
122             loadingBar.fillAmount = progress;
123             yield return null;
124         }
125     }
```

Εικόνα 48 Εκκίνηση του παιχνιδιού

Η συνάρτηση OnEvent της Photon εκτελείται όταν εντοπίζεται ένα event από το δίκτυο. Γίνεται ανάλυση του κωδικού που στέλνεται μέσω του δικτύου, και αν ισούται με τον κωδικό για την αλλαγή σκηνής, τότε εκτελεί την συνάρτηση *LoadClientLevelAsync()* (Εικόνα 49). Για να είναι εφικτό ο κώδικας να λαμβάνει τα events, είναι απαραίτητο να γίνει η εγγραφή της συνάρτησης OnEvent ως ανάκληση (Εικόνα 50).

```
127     private void OnEvent(EventData photonEvent)
128     {
129         byte eventCode = photonEvent.Code;
130         if (eventCode == ChangeSceneCode)
131         {
132             object[] data = (object[])photonEvent.CustomData;
133             int id = (int)data[0];
134 
135             mainMenu.SetActive(false);
136             loadingScreen.SetActive(true);
137 
138             StartCoroutine(LoadClientLevelAsync());
139         }
140     }
```

Εικόνα 49 Λήψη του event για την αλλαγή σκηνής

```
24      public override void OnEnable()
25      {
26          PhotonNetwork.NetworkingClient.EventReceived += OnEvent;
27          PhotonNetwork.AddCallbackTarget(this);
28      }
29 
        4 references
30      public override void OnDisable()
31      {
32          PhotonNetwork.NetworkingClient.EventReceived -= OnEvent;
33          PhotonNetwork.RemoveCallbackTarget(this);
34      }
```

Εικόνα 50 Εγγραφή στην λήψη των event

Ο κωδικός για την αλλαγή σκηνής επιλέχθηκε αυθαίρετα, όπως και με κάθε άλλο κωδικό που επιλέχθηκε για την αποστολή των Event (Εικόνα 51).

```
        2 references
23      public const byte ChangeSceneCode = 20;
```

Εικόνα 51 Κωδικός αλλαγής σκηνής

Εάν ο παίκτης θέλει να αποσυνδεθεί από το δωμάτιο, πρέπει να πατήσει το κουμπί “Leave Room” (Εικόνα 11). Γίνεται η αποσύνδεση του δωματίου με την συνάρτηση *PhotonNetwork.LeaveRoom()* και η επανασύνδεση με την αίθουσα για την ανανέωση των δωματίων (Εικόνα 52).

```
142     public void BackOnClick()
143     {
144         lobbyPanel.SetActive(true);
145         roomPanel.SetActive(false);
146         PhotonNetwork.LeaveRoom();
147         PhotonNetwork.LeaveLobby();
148         StartCoroutine(rejoinLobby());
149     }
150 
        1 reference
151     IEnumerator rejoinLobby()
152     {
153         yield return new WaitForSeconds(1);
154         PhotonNetwork.JoinLobby();
155     }
```

Εικόνα 52 Αποσύνδεση από το δωμάτιο

Στο script RegionScript.cs γίνεται η οπτική εμφάνιση των διαθέσιμων διακομιστών. Έχουμε το Enum CloudRegionCode για να αντιπροσωπεύει τους διακομιστές, και την λίστα *cloudRegionList* για να περιγράφει στον παίκτη το όνομα του κάθε διακομιστή (Εικόνα 53). Με την συνάρτηση *GetRegions()* εισχωρούμε την λίστα *cloudRegionList* στην αναπτυσσόμενη λίστα (Εικόνα 13). Μέσα σε μια επανάληψη ψάχνουμε σε ποιόν διακομιστή είναι συνδεδεμένος ο παίκτης, και εμφανίζουμε την ονομασία.

```
6 references
public TMP_Dropdown regionDropdown;
3 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 0 references | 1 reference
[SerializeField] public enum CloudRegionCode { asia, au, cae, eu, jp, za, sa, kr, tr, us, usw, none }
1 reference
private List<string> cloudRegionList = new List<string> {
    "Asia", "Australia", "Canada", "Europe", "Japan", "South Africa", "South America", "South Korea", "Turkey", "USA - East", "USA - West"
};

1 reference
void GetRegions()
{
    regionDropdown.AddOptions(cloudRegionList);
    int counter = 0;
    foreach (CloudRegionCode region in System.Enum.GetValues(typeof(CloudRegionCode)))
    {
        if (region != CloudRegionCode.none)
        {
            if (region.ToString() == PhotonNetwork.CloudRegion)
            {
                regionIndex = counter;
            }
        }
        counter++;
    }
    regionDropdown.value = regionIndex;
    regionDropdown.RefreshShownValue();
}
```

Εικόνα 53 Εμφάνιση διαθέσιμων διακομιστών

Τέλος, στην συνάρτηση Update της Unity, υπολογίζουμε το ping του διακομιστή, για να είναι ενημερωμένος και ο παίκτης, με την συνάρτηση Ph*otonNetwork.GetPing() (Εικόνα 54).*

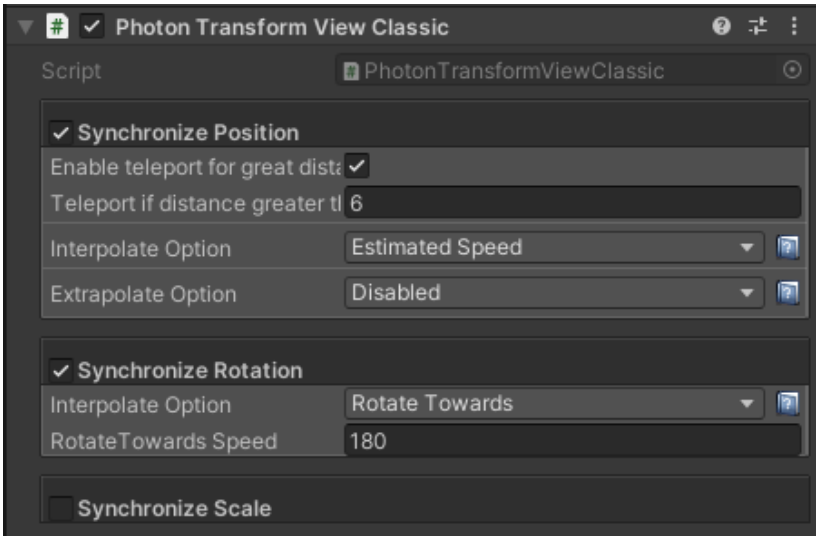
```
void Update()
{
    if (PhotonNetwork.OfflineMode == true)
    {
        pingText.gameObject.SetActive(false);
    }
    else
    {
        pingText.gameObject.SetActive(true);
        pingText.text = PhotonNetwork.GetPing().ToString() + "ms";
    }
}
```

Εικόνα 54 Εμφάνιση ping

4.2 Συγχρονισμός καταστάσεων παιχνιδιού

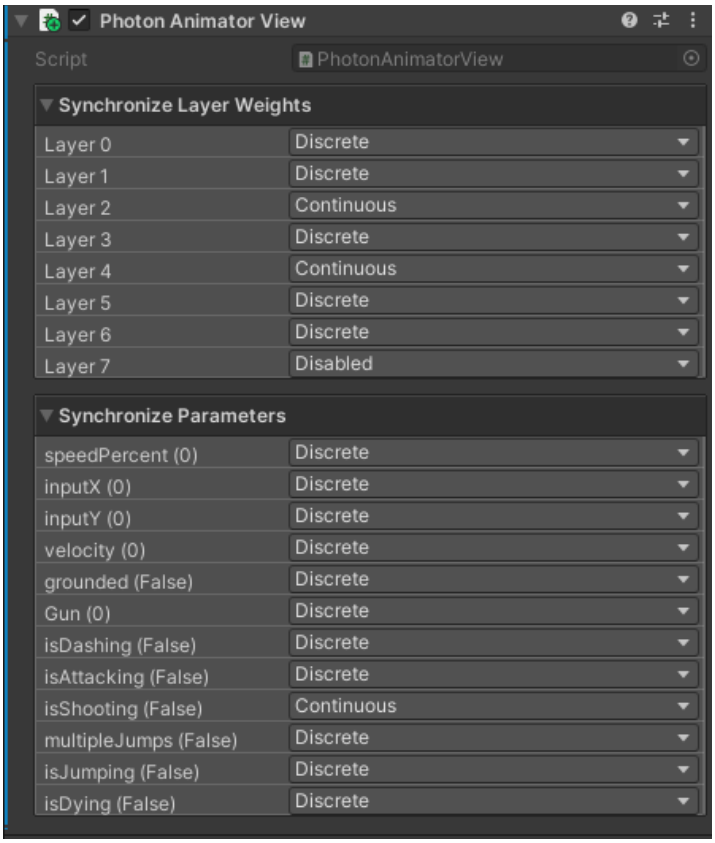
Τα αντικείμενα που έχουν συνεχή κίνηση ή έχουν κώδικες με συναρτήσεις για τον συγχρονισμό τους μεταξύ των παικτών έχουν από ένα PhotonView το καθένα. Συγκεκριμένα, το PhotonView το διαθέτουν οι παίκτες, τα όπλα, οι εχθροί, τα power ups, τα κουτιά και το αντικείμενο που ονομάστηκε Game Manager. Για τον εύκολο συγχρονισμό της τοποθεσίας, της περιστροφής, και του μεγέθους των αντικειμένων, όπως τα μοντέλο του παίκτη, των εχθρών, και

των όπλων προσθέτουμε το component Photon Transform View Classic (Εικόνα 55) στον Inspector της Unity και έτσι επιτυγχάνεται ο συγχρονισμός τους.



Εικόνα 55 Ρυθμίσεις Photon Transform View Classic

Στην περίπτωση των animations, όποιο μοντέλο έχει έναν Animator, χρειάζεται και το Photon Animator View (Εικόνα 56) για να βλέπουν όλοι οι παίκτες τα ίδια animations. Επίσης πρέπει χειροκίνητα να επιλεχθεί πως κάθε layer η κάθε animation θα συγχρονιστεί με τους υπόλοιπους παίκτες. Οι επιλογές που παρέχονται είναι απενεργοποιημένος (disabled), διακριτός (discrete), ή συνεχής (continuous) συγχρονισμός.



Εικόνα 56 Ρυθμίσεις Photon Animator View

4.2.1 Συγχρονισμός παικτών

Τα scripts υπεύθυνα για τον συγχρονισμό των παικτών είναι τα: PlayerManager.cs, PlayerStats.cs, PlayerUI.cs, ThirdPersonController.cs και CameraScript.cs.

Στο script PlayerManager.cs όταν αρχίζει το παιχνίδι καλούμε την συνάρτηση *Spawn()* και επιλέγουμε ένα τυχαίο σημείο και ένα τυχαίο μοντέλο, για να δημιουργήσουμε τον παίκτη με την συνάρτηση *PhotonNetwork.Instantiate(),* όπου στην συνάρτηση αυτή εισάγουμε ως παράμετρο το μοντέλο, την τοποθεσία, και την περιστροφή. Για την περιστροφή δίνουμε την τιμή *Quaternion.identity* για να παραμείνει η πρωτότυπη περιστροφή του μοντέλου. Χρησιμοποιούμε την *PhotonNetwork.Instantiate* της Photon, επειδή πρέπει το μοντέλο να ανήκει στον παίκτη που μπήκε στο παιχνίδι, αλλά και επειδή το αντικείμενο του παίκτη περιέχει ένα PhotonView, άρα είναι και δικτυωμένο αντικείμενο. Επίσης αποθηκεύουμε τα στατιστικά του παίκτη από ένα άλλο script, το PlayerStats.cs, για τους επόμενους ελέγχους (Εικόνα 57).

```
27      public void Start()
28      {
29          Spawn();
30      }
31
            1 reference
32      public void Spawn()
33      {
34          float x = Random.Range(-5f, 5f);
35          float z = Random.Range(-5f, 5f);
36          Vector3 randomPosition = new Vector3(x, 0, z);
37          Vector3 playerSpawner = spawner.position + randomPosition;
38
39          // Selecting random player model
40          int rand = Random.Range(0, playerPrefabs.Count);
41
42          player = PhotonNetwork.Instantiate(playerPrefabs[rand].name, playerSpawner, Quaternion.identity);
43          playerStats = player.GetComponent<PlayerStats>();
44
45          playerInput = player.GetComponent<PlayerInput>();
46          playerMap = playerInput.actions.FindActionMap("Player");
47          menuMap = playerInput.actions.FindActionMap("Menu");
48
49          if (playerSpawnedCallback != null)
50              playerSpawnedCallback.Invoke();
51      }
```

Εικόνα 57 Δημιουργία παίκτη από το PlayerManager

Σε κάθε καρέ γίνεται ο έλεγχος για τους πόντους ζωής του παίκτη για να εντοπιστεί η στιγμή που θα χάσει, για να εκτελεστεί η συνάρτηση *Die().* Ταυτόχρονα γίνεται και ο έλεγχος για την περίπτωση που ο παίκτης καταφέρει να κερδίσει, για να εκτελεστεί η συνάρτηση *Victory()* (Εικόνα 58).

```
53      public void Update()
54      {
55          if (playerStats.getHealth() <= 0 && !isDead && !runOnlyOnce)
56          {
57              runOnlyOnce = true;
58              StartCoroutine(Die());
59          }
60
61          if (hasWon && !isDead && !runOnlyOnce)
62          {
63              runOnlyOnce = true;
64              StartCoroutine(Victory());
65          }
66      }
```

Εικόνα 58 Έλεγχος συνθηκών για την ήττα ή την νίκη του παίκτη

Άμα χάσει ο παίκτης, την στιγμή που θα ξεμείνει από πόντους ζωής, απενεργοποιείται ο χειρισμός του παίκτη, λαμβάνεται το PhotonView για το μοναδικό Id, και στέλνουμε ένα RPC σε όλους τους παίκτες για να εκτελεστεί η συνάρτηση *ChangeTag* (Εικόνα 59) όπου αλλάζει το tag από “Player” σε “Spectator” (Εικόνα 60). Ως παράμετρος του RPC στέλνεται το id του παίκτη που μόλις έχασε, για να γνωρίζουν όλοι οι παίκτες, ποιανού το tag πρέπει να αλλάξει. Μετά πετιούνται τα όπλα που κρατούσε για να μην χαθούν, και αφού περάσει ένα χρονικό διάστημα, στέλνεται ένα δεύτερο RPC με την συνάρτηση *PlayerDeath* (Εικόνα 61) σε όλους τους παίκτες για να εμφανιστεί το οπτικό εφέ όταν χάνει ένας παίκτης. Οι συναρτήσεις *ChangeTag()* και *PlayerDeath()* βρίσκονται σε ένα άλλο script, το ThirdPersonController.cs. Τέλος, εμφανίζονται τα αποτελέσματα στον παίκτη που έχασε. Στην περίπτωση που ο παίκτης κερδίσει το παιχνίδι, δεν υπάρχει ανάγκη για κάποιο συγχρονισμό, οπότε αρχίζουν τα πυροτεχνήματα και η κατάλληλη μουσική τοπικά στον κάθε παίκτη.

```
68      IEnumerator Die()
69      {
70          playerMap.Disable();                                // Disable controls
71          menuMap.Enable();                                   // Enable Menu controls
72          isDead = true;
73          player.GetPhotonView().RPC("ChangeTag", RpcTarget.All, player.GetPhotonView().ViewID);
74          player.tag = "Spectator";
75
76          EquipmentManager equipManager = player.transform.GetChild(0).GetComponent<EquipmentManager>();
77          equipManager.UnequipAllAtOnce();
78          yield return new WaitForSeconds(1.5f);
79          player.transform.GetChild(2).GetChild(0).GetComponent<Animator>().enabled = false;    // Disable animator
80          player.GetPhotonView().RPC("PlayerDeath", RpcTarget.All);
81
82          yield return new WaitForSeconds(1.5f);
83          playerUI.ShowResultsScreen(0); // 0 is defeat screen
84
85          yield return new WaitForSeconds(1f);
86
87          cameraScript.isSpectating = true;
88          cameraScript.ChangeTarget();
89      }
90
        1 reference
91      IEnumerator Victory()
92      {
93          yield return new WaitForSeconds(3f);
94          StartCoroutine(Fireworks());
95          musicManager.PlayVictoryMusic();
96          yield return new WaitForSeconds(6f);
97          playerUI.ShowResultsScreen(1); // 1 is victory screen
98
99          yield return null;
100     }
```

Εικόνα 59 Συναρτήσεις νίκης και ήττας

```
381     [PunRPC]
        0 references
382     private void ChangeTag(int id)
383     {
384         GameObject playerToChangeTag = PhotonView.Find(id).gameObject;
385         playerToChangeTag.tag = "Spectator";
386     }
```

Εικόνα 60 Μεταβολή του tag του αντικειμένου σε Spectator

```
371     [PunRPC]
        0 references
372     private void PlayerDeath(PhotonMessageInfo info)
373     {
374         Vector3 VFXPositionCorrection = new Vector3(0, 2, 0);
375         Instantiate(playerDeathVFX, info.photonView.transform.position + VFXPositionCorrection, Quaternion.identity);
376         playerDeathVFX.Play();
377         shakeScript.ShakeCamera(8, 0.6f);
378         Destroy(info.photonView.transform.GetChild(2).gameObject);   // Disable GFX
379     }
```

Εικόνα 61 Συγχρονισμός οπτικού εφέ της ήττας ενός παίκτη

Στο script PlayerStats.cs υπάρχουν όλα τα αρχικά στατιστικά του παίκτη (Εικόνα 62). Για να γίνει η αποφυγή του προβλήματος όπου ο ένας παίκτης θα εκτελέσει κώδικα από script που ανήκει σε άλλον παίκτη, πρέπει να γίνει ο διαχωρισμός τους. Δηλαδή πρέπει να γίνει ο έλεγχος εάν το PhotonView που βρίσκεται στο μοντέλο του παίκτη, ανήκει στο άτομο που το χειρίζεται. Ο έλεγχος αυτός γίνεται με την ιδιότητα PhotonView.IsMine όπου το PhotonView πρέπει να αναφερθεί πρώτα (Εικόνα 63).

```
8       [SerializeField] private float health = 0.0f;
        8 references
9       [SerializeField] private float maxHealth = 100.0f;
        4 references
10      [SerializeField] private float armor = 0.0f;
        2 references
11      [SerializeField] private float speed = 12.0f;
        3 references
12      [SerializeField] private float additiveSpeed = 0.0f;
        4 references
13      [SerializeField] private float additiveDamage = 0.0f; // additive percentage of damage
        3 references
14      [SerializeField] private float multiplicativeDamage = 1.0f; // damage multiplier
        4 references
15      [SerializeField] private float additiveAttackSpeed = 0.0f; // additive percentage of attack speed
        4 references
16      [SerializeField] private float additiveReloadTime = 0.0f; // percentage
        3 references
17      [SerializeField] private float additiveAmmo = 0.0f; // percentage
        2 references
18      [SerializeField] private int maxJumpCount = 1;
        2 references
19      [SerializeField] private float dashCooldown = 1.6f;
        4 references
20      [SerializeField] private float healPotency = 1f;
        4 references
21      public List<PowerUp> powerUps = new List<PowerUp>();
```

Εικόνα 62 Στατιστικά παίκτη

Στην γραμμή 34 (Εικόνα 63) γίνεται η αποθήκευση του PhotonView του παίκτη με την συνάρτηση *transform.GetComponent<PhotonView>()* στην μεταβλητή *pv* τύπου PhotonView. Μετά γίνεται ο έλεγχος “if (!pv.IsMine) return;” ώστε ο υπόλοιπος κώδικας να μην εκτελεστεί από τα άτομα στα οποία δεν ανήκει το μοντέλο. Αφού γίνει ο έλεγχος καλούμε την συνάρτηση *GetPermanentUpgrades()* για να ανανεώσουμε τα στατιστικά του παίκτη με βάση τα upgrades που έχει ξεκλειδώσει.

```
28      private const string CurrentLootKey = "CurrentLootKey";
29
        0 references
30      private void Awake()
31      {
32          kills = 0;
33          gameLoot = 0;
34          pv = transform.GetComponent<PhotonView>();
35
36          if (!pv.IsMine)
37              return;
38
39          GetPermanentUpgrades();
40          health = maxHealth;
41          if (PlayerPrefs.HasKey(CurrentLootKey))
42          {
43              collectedLoot = PlayerPrefs.GetInt(CurrentLootKey);
44          }
45      }
```

Εικόνα 63 Αναφορά στο PhotonView του παίκτη

Το *OnPhotonSerializeView()* είναι μια συνάρτηση της Photon, για την συνεχή ενημέρωση μεταβλητών μέσω του δικτύου. Σε αυτήν την περίπτωση η συνάρτηση χρησιμοποιείται ώστε ο ένας παίκτης να ενημερώνει τους υπόλοιπους για την τιμή που έχει στους πόντους ζωής. Η συνθήκη "if (stream.IsWriting)" ισχύει πάντα για το άτομο στο οποίο ανήκει το μοντέλο με το script, και έτσι στέλνει τις τιμές που πρέπει να ενημερωθούν με την συνάρτηση *SendNext().* Για τα άλλα άτομα στα οποία δεν ανήκει αυτό το μοντέλο με το script, το "if (stream.IsWriting)" δεν ισχύει ποτέ, οπότε λαμβάνουν τις τιμές με την συνάρτηση *ReceiveNext()* (Εικόνα 64).

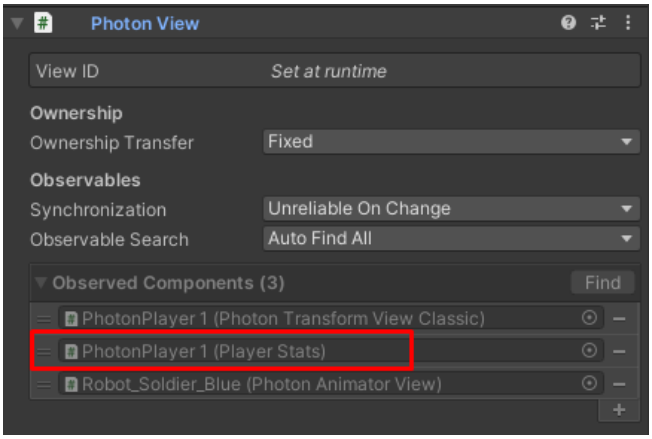
```
47      public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
48      {
49          if (stream.IsWriting)
50          {
51              stream.SendNext(health);
52              stream.SendNext(maxHealth);
53          }
54          else
55          {
56              health = (float)stream.ReceiveNext();
57              maxHealth = (float)stream.ReceiveNext();
58          }
59      }
```

Εικόνα 64 Ανανέωση πόντων ζωής σε όλους τους παίκτες

Για την σωστή λειτουργία της συνάρτησης *OnPhotonSerializeView()*, πρέπει η κλάση να κληρονομεί από την κλάση IPunObservable (Εικόνα 65), και το script να βρίσκεται στα "Observed Components" του PhotonView (Εικόνα 66).

```
        39 references
5       public class PlayerStats : MonoBehaviourPunCallbacks, IPunObservable
6       {
```

Εικόνα 65 Κληρονομία κλάσης PlayerStats



Εικόνα 66 PhotonView με Observed Components

Στην συνάρτηση *Update* (Εικόνα 67) γίνεται ξανά έλεγχος για το εάν το PhotonView ανήκει στον παίκτη ώστε να μην αλλάζει η τιμή της ζωής στους υπόλοιπους παίκτες.

```
        0 references
61      public void Update()
62      {
63          if (!pv.IsMine)
64              return;
65
66          if (health > maxHealth)
67              health = maxHealth;
68      }
```

Εικόνα 67 Έλεγχος του PhotonView μέσα σε Update

Στο script PlayerUI.cs μόλις ξεκινήσει το παιχνίδι, κάθε παίκτης στέλνει προς τους υπόλοιπους ένα event (Εικόνα 68) με κωδικό *SetOtherPlayersHealthbarCode*. Όταν ένας παίκτης λαμβάνει το event αυτό, βρίσκει από ποιόν παίκτη το έλαβε μέσω του id, δημιουργώντας στην διεπαφή χρήστη του μια καινούργια μπάρα ζωής και εισχωρώντας τα ανάλογα στοιχεία (Εικόνα 69).

```
139         object[] content = new object[] { pv.ViewID };
140         RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.Others };
141         PhotonNetwork.RaiseEvent(SetOtherPlayersHealthbarCode, content, raiseEventOptions, SendOptions.SendReliable);
```

Εικόνα 68 Αποστολή id παίκτη στο PlayerUI

```
449     public void OnEvent(EventData photonEvent)
450     {
451         byte eventCode = photonEvent.Code;
452         if (eventCode == SetOtherPlayersHealthbarCode)
453         {
454             object[] data = (object[])photonEvent.CustomData;
455             int id = (int)data[0];
456
457             GameObject playerSent = PhotonView.Find(id).gameObject;
458             GameObject healthbar = Instantiate(otherPlayerHealthbarPrefab, otherPlayersHealthbar);
459             OtherPlayersHealthbar healthbarScript = healthbar.GetComponent<OtherPlayersHealthbar>();
460             healthbarScript.player = playerSent;
461             healthbarScript.setNickname(playerSent.GetComponent<PhotonView>().Owner.NickName);
462             healthbarScript.setStats(playerSent.GetComponent<PlayerStats>());
463         }
464     }
```

Εικόνα 69 Εύρεση και εισχώρηση στοιχείων του παίκτη που έστειλε το event

Το script ThirdPersonController.cs ελέγχει όλους τους μηχανισμούς του παίκτη, όπως την κίνηση, το άλμα, το dash, τον εντοπισμό αντικειμένων, την βαρύτητα και τις συγκρούσεις. Πρέπει να γίνει ο απαραίτητος έλεγχος ώστε να είναι διαχωρισμένος ο κώδικας μεταξύ των παικτών, οπότε χρησιμοποιείται η συνθήκη “if (!photonView.IsMine) return;” (Εικόνα 70).

```
90      private void Update()
91      {
92          if (!photonView.IsMine)
93              return;
```

Εικόνα 70 Έλεγχος συγχρονισμού ThirdPersonController

Πέρα από τις συναρτήσεις *ChangeTag()* και *PlayerDeath()* που αναλύθηκαν ήδη, υπάρχει και η συνάρτηση *TouchedLava()*, η οποία συγχρονίζει το οπτικό εφέ όταν ένας παίκτης ακουμπάει έξω από την αρένα (Εικόνα 71).

```
347     public void TouchedLava(float force)
348     {
349         photonView.RPC("RPC_TouchedLava", RpcTarget.Others);
350         lavaTouchVFX.Play();
351         velocity.y = force;
352
353         controller.Move(velocity * Time.deltaTime);
354     }
355
356     [PunRPC]
        0 references
357     public void RPC_TouchedLava(PhotonMessageInfo info)
358     {
359         if (info.photonView.transform != null)
360         {
361             Transform playerHit = info.photonView.transform;
362             if (playerHit.GetComponent<ThirdPersonController>() != null)
363             {
364                 ThirdPersonController playerHitScript = playerHit.GetComponent<ThirdPersonController>();
365                 if (playerHitScript.lavaTouchVFX != null)
366                     playerHitScript.lavaTouchVFX.Play();
367             }
368         }
369     }
```

Εικόνα 71 Συγχρονισμός οπτικού εφέ μέσω RPC

Το script CameraScript.cs ασχολείται με την συμπεριφορά της κάμερας του παίκτη. Όταν όμως ένας παίκτης χάνει, μπορεί να παρακολουθεί τους υπόλοιπους παίκτες. Στην γραμμή 52 (Εικόνα 72) γίνεται ο έλεγχος για το αν ο παίκτης είναι θεατής, και αν πατήσει αριστερό κλικ, αλλάζει το άτομο που παρακολουθεί με την συνάρτηση *ChangeTarget().* Στην συνάρτηση αυτή πρώτα γίνεται η μέτρηση των παικτών που έχουν απομείνει ψάχνοντας για αντικείμενα με το tag “Player”. Αν έχει μείνει κάποιος χωρίς να χάσει, τότε η κάμερα τον θέτει ως αντικείμενο παρακολούθησης και εμφανίζει το όνομα του στην οθόνη. Το κλικ του θεατή αυξάνει έναν μετρητή για να αλλάζει και το άτομο που παρακολουθεί, με τον έλεγχο ότι ο μετρητής δεν ξεπερνάει τον αριθμό των διαθέσιμων παικτών.

```
41      private void Update()
42      {
43          spectateChangeTimer -= Time.deltaTime;
44          if (!cameraSet)
45          {
46              if (PlayerManager.instance.player != null)
47              {
48                  SetCamera();
49                  cameraSet = true;
50              }
51          }
52          if (isSpectating)
53          {
54              if (Mouse.current.leftButton.wasPressedThisFrame)
55              {
56                  if (spectateChangeTimer <= 0)
57                  {
58                      spectateChangeTimer = 0.5f;
59                      playerIndex++;
60                      ChangeTarget();
61                  }
62              }
63          }
64      }
65
            2 references
66      public void ChangeTarget()
67      {
68          GameObject[] playerTags = GameObject.FindGameObjectsWithTag("Player");
69          if (playerTags.Length > 0)
70          {
71              if (playerIndex > playerTags.Length - 1)
72              {
73                  playerIndex = 0;
74              }
75              virtualCamera.Follow = playerTags[playerIndex].transform.GetChild(1);
76              virtualCamera.LookAt = playerTags[playerIndex].transform.GetChild(1);
77              playerSpectating.text = playerTags[playerIndex].transform.GetComponent<PhotonView>().Owner.NickName;
78          }
79          else
80          {
81              playerSpectating.text = "";
82          }
83      }
```

Εικόνα 72 Παρακολούθηση παικτών

4.2.2 Συγχρονισμός γύρων

Η λογική και οι μηχανισμοί των γύρων εξαρτώνται από τα script: GameManager.cs, GameMode.cs, Defence.cs, Exterminate.cs, Parkour.cs, Survival.cs και FinalGameMode.cs.

Το GameManager.cs κατέχει τον κώδικα που διαχειρίζεται και συγχρονίζει τους γύρους μεταξύ των παικτών. Γίνεται η αρχικοποίηση των υπόλοιπων script σε αυτό και μετά γίνεται ο έλεγχος *PhotonNetwork.IsMasterClient* (Εικόνα 73). Η ιδιότητα αυτή είναι true όταν ο παίκτης που τρέχει τον κώδικα είναι ο host του παιχνιδιού. Είναι σημαντικό ο έλεγχος να γίνεται μέσω του host για να μην χαθεί ο συγχρονισμός. Οπότε γίνεται αυτός ο αρχικός έλεγχος για να δημιουργηθούν τα πρώτα κουτιά όπλων ανάλογα με τον αριθμό τον παικτών στο παιχνίδι. Τα κουτιά δημιουργούνται με την συνάρτηση *PhotonNetwork.Instantiate()* επειδή είναι δικτυωμένα αντικείμενα, δηλαδή περιέχουν από ένα PhotonView.

```
51      void Awake()
52      {
53          exterminateScript = GetComponent<Exterminate>();
54          defenceScript = GetComponent<Defence>();
55          parkourScript = GetComponent<Parkour>();
56          survivalScript = GetComponent<Survival>();
57          finalGameModeScript = GetComponent<FinalGameMode>();
58          pv = GetComponent<PhotonView>();
59          m_audio = GetComponent<AudioSource>();
60          if (PhotonNetwork.IsMasterClient)
61          {
62              int offset = 0;
63              foreach (Player player in PhotonNetwork.PlayerList)
64              {
65                  Vector3 offsetVector = new Vector3(offset, 0, 0);
66                  PhotonNetwork.Instantiate(gunChest.name, gunChestSpawn.position + offsetVector, Quaternion.Euler(new Vector3(0, 180, 0)));
67                  offset += 3;
68              }
69          }
70      }
```

Εικόνα 73 Αρχικοποίηση του Game Manager

Για την δημιουργία των γύρων, πρώτα ελέγχεται πως ο host εκτελεί των κώδικα, και μετά διορίζει τον πίνακα με τους γύρους όπως έχουν αναλυθεί στον σχεδιασμό του παιχνιδιού (Εικόνα 74). Στο τέλος στέλνεται στους υπόλοιπους παίκτες το RPC με την συνάρτηση *SyncWaveModes()* για να συγχρονιστεί ο πίνακας με τους γύρους του παιχνιδιού (Εικόνα 75).

```
72      void Start()
73      {
74          currentWave = 0;
75          waveModes = new int[maxWaves];
76          waveModes[0] = 0; // First wave is always exterminate
77
78          // Starting from second wave up to max waves -1 because the last wave
79          // is reserved as a special wave
80          for (int i = 1; i < maxWaves - 1; i++)
81          {
82              waveModes[i] = 0;
83          }
84
85          if (PhotonNetwork.IsMasterClient)
86          {
87              for (int i = 0; i < maxDefenceWaves; i++)
88              {
89                  int randSpot = Random.Range(1, maxWaves - 1);
90                  if (waveModes[randSpot] == exterminateCode && waveModes[randSpot - 1] != defenceCode && waveModes[randSpot + 1] != defenceCode)
91                  {
92                      waveModes[randSpot] = defenceCode;
93                  }
94                  else
95                  {
96                      i--;
97                  }
98              }
99              for (int i = 0; i < maxParkourWaves; i++)
100             {
101                 int randSpot = Random.Range(1, maxWaves - 1);
102                 if (waveModes[randSpot] == exterminateCode && waveModes[randSpot - 1] != parkourCode && waveModes[randSpot + 1] != parkourCode)
103                 {
104                     waveModes[randSpot] = parkourCode;
105                 }
106                 else
107                 {
108                     i--;
109                 }
110             }
111
112             for (int i = 0; i < maxSurvivalWaves; i++)
113             {
114                 int randSpot = Random.Range(1, maxWaves - 1);
115                 if (waveModes[randSpot] == exterminateCode && waveModes[randSpot - 1] != survivalCode && waveModes[randSpot + 1] != survivalCode)
116                 {
117                     waveModes[randSpot] = survivalCode;
118                 }
119                 else
120                 {
121                     i--;
122                 }
123             }
124             waveModes[maxWaves - 1] = 4;
125             pv.RPC("SyncWaveModes", RpcTarget.Others, waveModes);
126         }
127
128         //pv.RPC("StartWave", RpcTarget.AllViaServer, currentWave);
129     }
```

Εικόνα 74 Δημιουργία των γύρων από τον host

```
131     [PunRPC]
        0 references
132     private void SyncWaveModes(int[] masterWaveModes)
133     {
134         int i = 0;
135         foreach (int mode in masterWaveModes)
136         {
137             waveModes[i] = mode;
138             i++;
139         }
140     }
```

Εικόνα 75 Συγχρονισμός των γύρων μεταξύ των clients

Όταν πρέπει να αρχίσει ο επόμενος γύρος στέλνεται, μέσω του διακομιστή, σε όλους τους παίκτες ένα RPC με την συνάρτηση *StartWave()* και παράμετρο τον τρέχων γύρο για να αρχίσουν όλοι την ίδια στιγμή (Εικόνα 76).

```
298     public void IncreaseCurrentWave()
299     {
300         currentWave++;
301         pv.RPC("StartWave", RpcTarget.AllViaServer, currentWave);
302     }
```

Εικόνα 76 RPC για την εκκίνηση του επόμενου γύρου

Την στιγμή που αρχίζει ο γύρος, ο host ψάχνει όλα τα όπλα που δεν είναι εξοπλισμένα από κανέναν παίκτη και τα όπλα κουτιών ώστε να διαγραφτούν από το παιχνίδι (Εικόνα 77) με την συνάρτηση *PhotonNetwork.Destroy()* και παράμετρο το PhotonView των αντικειμένων επειδή είναι δικτυωμένα αντικείμενα. Η καταστροφή ενός PhotonView είναι δυνατή μόνο από τον παίκτη που το δημιούργησε. Επειδή τα κουτιά και τα όπλα που διαθέτουν από ένα PhotonView δημιουργούνται από τον host, πρέπει να καταστραφούν και από τον host.

```
142     [PunRPC]
        0 references
143     private void StartWave(int wave)
144     {
145         if (PhotonNetwork.IsMasterClient)
146         {
147             GameObject[] droppedGuns = GameObject.FindGameObjectsWithTag("Gun");
148             GameObject[] droppedChests = GameObject.FindGameObjectsWithTag("Chest");
149
150             foreach (GameObject gun in droppedGuns)
151             {
152                 if (gun.transform.parent == null)
153                 {
154                     PhotonNetwork.Destroy(gun.GetPhotonView());
155                 }
156             }
157             foreach (GameObject chest in droppedChests)
158             {
159                 if (chest.GetPhotonView() != null)
160                     PhotonNetwork.Destroy(chest.GetPhotonView());
161             }
162         }
```

Εικόνα 77 Διαγραφή όπλων και κουτιών

Μετέπειτα συγχρονίζονται όλες οι μεταβλητές που χρειάζονται οι clients για να ενημερώσουν την διεπαφή χρήστη τους όπως ο αριθμός του γύρου, το είδος του γύρου, αλλά και να ενεργοποιούν το script για τον γύρο που αρχίζει ώστε να μπορούν να συνεχίσουν το παιχνίδι σε περίπτωση που αποσυνδεθεί ο host (Εικόνα 78).

```
        SetCurrentWave(wave);
        waveInterface.SetActive(false);

        if (wave <= 3)
        {
            difficulty = 1;
        }
        else if (wave <= 6)
        {
            difficulty = 2;
        }
        else if (wave <= 9)
        {
            difficulty = 3;
        }
        else
        {
            difficulty = 4;
        }

        SetCurrentWaveText(wave);
        currency = 50 * PhotonNetwork.PlayerList.Length + 25 * (wave);

        SetCurrentWaveMode(waveModes[wave - 1]);
        StartCoroutine(SetWaveUI());

        switch (currentWaveMode)
        {
            case 0:
                currentGameModeScript = exterminateScript;
                modeText.text = "Exterminate";
                break;
            case 1:
                currentGameModeScript = defenceScript;
                modeText.text = "Defence";
                break;
            case 2:
                currentGameModeScript = parkourScript;
                modeText.text = "Parkour";
                break;
            case 3:
                currentGameModeScript = survivalScript;
                modeText.text = "Survival";
                break;
            case 4:
                currentGameModeScript = finalGameModeScript;
                modeText.text = "Rampage";
                break;
        }

        currentGameModeScript.enabled = true;
        currentGameModeScript.StartWave(difficulty);
    }
```

Εικόνα 78 Συγχρονισμός γύρου

Όταν τελειώσει ο γύρος, ο host ξανά εμφανίζει τα κουτιά όπλων ανάλογα με τον αριθμό των παικτών (Εικόνα 79).

```
219     void LateUpdate()
220     {
221         // Behaviour when wave is complete
222         if (currentGameModeScript != null)
223         {
224             if (currentGameModeScript.getWaveComplete() == true)
225             {
226                 if (currentWave == 10)
227                 {
228                     StartCoroutine(WaveCompleteUI());
229                     m_audio.Play();
230                     PlayerManager.instance.hasWon = true;
231                     currentGameModeScript = null;
232                 }
233                 else
234                 {
235                     waveInterface.SetActive(true);
236
237                     StartCoroutine(WaveCompleteUI());
238                     m_audio.Play();
239
240                     if (PhotonNetwork.IsMasterClient)
241                     {
242                         int offset = 0;
243                         foreach (Player player in PhotonNetwork.PlayerList)
244                         {
245                             Vector3 offsetVector = new Vector3(offset, 0, 0);
246                             PhotonNetwork.Instantiate(gunChest.name, gunChestSpawn.position + offsetVector, Quaternion.Euler(new Vector3(0, 180, 0)));
247                             offset += 3;
248                         }
249                     }
250
251                     currentGameModeScript.SetWaveComplete(false);
252                     currentGameModeScript.enabled = false;
253                 }
254             }
255         }
256     }
```

Εικόνα 79 Δημιουργία κουτιών όπλων στο τέλος του γύρου

Κάθε είδος γύρου όπως το Exterminate, Defence, Survival, Parkour και το FinalGameMode, εφαρμόζουν το interface GameMode. Όλα τα είδη γύρων έχουν την μεταβλητή *waveComplete*, και μπορούν να στείλουν το event με κωδικό *SetWaveCompleteCode* για να συγχρονιστεί το γεγονός ότι ολοκληρώθηκε ο γύρος (Εικόνα 80).

```
4   public class GameMode : MonoBehaviourPunCallbacks
5   {
        12 references
6       protected bool waveComplete;
        6 references
7       protected GameManager gm;
8
        6 references
9       public const byte SetWaveCompleteCode = 15;
10
        4 references
11      public virtual void Awake()
12      {
13          gm = GetComponent<GameManager>();
14      }
        3 references
15      public override void OnEnable()
16      {
17          PhotonNetwork.NetworkingClient.EventReceived += OnEvent;
18      }
19
        4 references
20      public override void OnDisable()
21      {
22          PhotonNetwork.NetworkingClient.EventReceived -= OnEvent;
23      }
        1 reference
24      public virtual void StartWave(int difficulty)
25      {
26      }
27
        1 reference
28      public bool getWaveComplete()
29      {
30          return waveComplete;
31      }
        1 reference
32      public void SetWaveComplete(bool setWaveComplete)
33      {
34          waveComplete = setWaveComplete;
35      }
36
        2 references
37      private void OnEvent(EventData photonEvent)
38      {
39          byte eventCode = photonEvent.Code;
40
41          if (eventCode == SetWaveCompleteCode)
42              waveComplete = true;
43      }
44  }
```

Εικόνα 80 Interface GameMode

Το είδος γύρου Exterminate ελέγχεται από το Exterminate.cs. Σκοπός του script είναι η δημιουργία των εχθρών μέχρι το τέλος των χρημάτων του γύρου. Επίσης γίνεται έλεγχος για να εντοπιστεί αν έχει μείνει κάποιος εχθρός για την ολοκλήρωση του γύρου. Αρχικά παίρνουμε τα χρήματα που έχει το παιχνίδι από το Game Manager, αποθηκεύουμε τον αριθμό των παικτών και ξεκινάει η δημιουργία των εχθρών με την συνάρτηση *SpawnEnemy()* (Εικόνα 81).

```
29      public override void StartWave(int difficulty)
30      {
31          localCurrency = gm.GetCurrency();
32          firstEnemySpawned = false;
33          waveComplete = false;
34          time = 0;
35          playerAmount = PhotonNetwork.PlayerList.Length;
36
37          StartCoroutine(SpawnEnemy(difficulty));
38      }
```

Εικόνα 81 Αρχικοποίηση του γύρου

Στην συνάρτηση *Update* (Εικόνα 82), ο host ελέγχει αν έχει μείνει κάποιος εχθρός, ώστε όταν οι παίκτες καταφέρουν να τους κερδίσουν όλους, να στείλει σε όλους ένα event με τον κωδικό *SetWaveCompleteCode*. Η λήψη του event γίνεται στο script GameMode.cs.

```
41      void Update()
42      {
43          if (PhotonNetwork.IsMasterClient)
44          {
45              time += Time.deltaTime;
46              if (enemiesToSpawn.Count == 0 && localCurrency < 0)
47              {
48                  object[] content = new object[] { };
49                  RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.All };
50                  PhotonNetwork.RaiseEvent(SetWaveCompleteCode, content, raiseEventOptions, SendOptions.SendReliable);
51              }
52
53              if (time > pollingTime && enemiesToSpawn.Count > 0)
54              {
55                  time = 0;
56                  for (int i = enemiesToSpawn.Count; i > 0; i--)
57                  {
58                      if (enemiesToSpawn[i - 1] == null)
59                      {
60                          enemiesToSpawn.RemoveAt(i - 1);
61                      }
62                  }
63              }
64          }
65      }
```

Εικόνα 82 Έλεγχος για το τέλος του γύρου

Στην συνάρτηση *SpawnEnemy()*, μόνο ο host μόνο εκτελεί την συνάρτηση για την δημιουργία των εχθρών με την χρήση του script EnemySpawner.cs που αναλύεται αργότερα, με την συνάρτηση *EnemySpawner.SpawnEnemy()* και παραμέτρους τον επιλεγμένο εχθρό, την δυσκολία, και την συμπεριφορά του εχθρού (Εικόνα 83).

```
68      IEnumerator SpawnEnemy(int difficulty)
69      {
70          if (PhotonNetwork.IsMasterClient)
71          {
72              while (localCurrency >= 0)
73              {
74                  float randomInterval = Random.Range(0f, 2f);
75                  yield return new WaitForSeconds(randomInterval);
76
77                  int enemyIndex = Random.Range(0, 2);
78                  enemySpawner.SpawnEnemy(enemyIndex, difficulty, 0);
79
80                  GameObject enemy = enemySpawner.getEnemy();
81                  localCurrency -= enemy.GetComponent<EnemyStats>().cost;
82
83                  enemiesToSpawn.Add(enemySpawner.getEnemy());
84                  firstEnemySpawned = true;
85              }
86          }
87      }
```

Εικόνα 83 Δημιουργία εχθρών

Με την ίδια λογική γίνεται ο συγχρονισμός και των γύρων Defense, Parkour, Survival και FinalGameMode όταν δημιουργούνται εχθροί και όταν ολοκληρώνεται ο γύρος.

4.2.3 Συγχρονισμός όπλων

Τα όπλα έχουν πολλές λειτουργίες που πρέπει να συγχρονιστούν, ώστε οι παίκτες να βλέπουν τα σωστά όπλα μεταξύ τους, και να εμφανίζονται οι σωστές σφαίρες. Τα scripts που σχετίζονται με τα όπλα είναι τα: GunPickUp.cs, EquipmentManager.cs, Gun.cs και Projectile.cs.

Το GunPickUp.cs script συγχρονίζει το όπλο όταν ένας παίκτης εξοπλίζεται με αυτό. Στην αλληλεπίδραση με το όπλο, στέλνεται σε όλους ένα RPC για την εκτέλεση της συνάρτησης *RPC_PickUpGun()* που περιέχει το id του παίκτη. Στην συνάρτηση *RPC_PickUpGun(),* γίνεται η εύρεση του παίκτη με το id που παραλήφθηκε, και ενημερώνεται η κατοχή του όπλου, ώστε όλοι οι παίκτες να γνωρίζουν σε ποιον ανήκει το όπλο (Εικόνα 84).

```
39      public override void Interact(int id)
40      {
41          player = PhotonView.Find(id).gameObject;
42          equipmentScript = player.transform.GetChild(0).GetComponent<EquipmentManager>();
43          gameObject.SetActive(false);
44          pv.RPC("RPC_PickUpGun", RpcTarget.All, id);
45
46          if (Inventory.instance.isFull() == false){
47              equipmentScript.EquipGun(gameObject, true);
48          } else {
49              equipmentScript.FullInventoryUnequipGun();
50              equipmentScript.EquipGun(gameObject, false);
51          }
52
53          Inventory.instance.Add(0, gun);
54      }
55
56      [PunRPC]
        0 references
57      public void RPC_PickUpGun(int id)
58      {
59          GameObject _player = PhotonView.Find(id).gameObject;
60
61          transform.SetParent(_player.transform.GetChild(0));
62          transform.localPosition = Vector3.zero;
63          transform.localRotation = Quaternion.Euler(Vector3.zero);
64          transform.localScale = Vector3.one;
65          transform.SetAsFirstSibling();
66      }
```

Εικόνα 84 Συγχρονισμός του εξοπλισμού του παίκτη

Το EquipmentManager.cs παρέχει τις λειτουργίες του εξοπλισμού του παίκτη, όπως η απόκτηση, πτώση και εναλλαγή όπλων. Όταν ο παίκτης αποκτά ένα όπλο πρέπει να το εμφανίσουμε στα χέρια του μοντέλου και στους υπόλοιπους παίκτες. Άμα ο παίκτης κρατούσε ήδη ένα όπλο προτού αποκτήσει το καινούργιο, τότε πρέπει να εξαφανίσουμε το παλιό. Οι λειτουργίες αυτές εκτελούνται στην συνάρτηση *EquipGun() (Εικόνα 85)* και συγκεκριμένα η εμφάνιση του όπλου με την συνάρτηση *ShowEquipment()* ενώ η εξαφάνιση με την συνάρτηση *HideEquipment().* Εκτός από αυτές τις λειτουργίες, υπάρχει και η συνάρτηση *SetupGunComponents()* για τον συγχρονισμό της ενεργοποίησης και απενεργοποίησης άλλων component του όπλου. Το component PhotonView αποθηκεύεται στην γραμμή 75 (Εικόνα 85) για την χρήση του στις υπόλοιπες συναρτήσεις.

```
66      public void EquipGun(GameObject gunP, bool hideGun)
67      {
68          if (pv.IsMine)
69              return;
70
71          if (minimumGunAmount(2) && hideGun)
72              HideEquipment();
73
74          currentGun = gunP;
75          gunPv = currentGun.GetPhotonView();
76          currentEquipment = currentGun.GetComponent<Gun>().gunInfo;
77
78          ShowEquipment();
79          SetupGunComponents();
80
81          if (OnEquippedCallback != null)
82              OnEquippedCallback.Invoke();
83      }
```

Εικόνα 85 Εξοπλισμός ενός όπλου

Κάθε συνάρτηση από τις *ShowEquipment()*, *HideEquipment()*, και *SetupGunComponents()* στέλνουν από ένα event (Εικόνα 86) σε όλους τους παίκτες εκτός από την *SetupGunComponents()* όπου στέλνεται μόνο στους υπόλοιπους παίκτες, επειδή τοπικά πρέπει ο κώδικας να εκτελεστεί αμέσως. Σε κάθε περίπτωση πρώτα αποθηκεύεται το PhotonView του όπλου, και με το event στέλνεται το id του.

```
196     public void HideEquipment()
197     {
198         gunPv = currentGun.GetPhotonView();
199
200         object[] content = new object[] { gunPv.ViewID };
201         RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.All };
202         PhotonNetwork.RaiseEvent(HideEquipmentEventCode, content, raiseEventOptions, SendOptions.SendReliable);
203     }
204
        1 reference
205     public void ShowEquipment()
206     {
207         gunPv = currentGun.GetPhotonView();
208
209         object[] content = new object[] { gunPv.ViewID };
210         RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.All };
211         PhotonNetwork.RaiseEvent(ShowEquipmentEventCode, content, raiseEventOptions, SendOptions.SendReliable);
212     }
213
        1 reference
214     private void SetupGunComponents()
215     {
216         gunPv = currentGun.GetPhotonView();
217
218         object[] content = new object[] { gunPv.ViewID };
219         RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.Others };
220         PhotonNetwork.RaiseEvent(SetupGunComponentsEventCode, content, raiseEventOptions, SendOptions.SendReliable);
221
222         currentGun.transform.GetChild(0).gameObject.SetActive(false);
223
224         rb = currentGun.GetComponent<Rigidbody>();
225         gunScript = currentGun.GetComponent<Gun>();
226         pickUpScript = currentGun.GetComponent<GunPickUp>();
227         gunCollider = currentGun.GetComponent<Collider>();
228
229         gunScript.enabled = true;
230         gunScript.setGunParent(true);
231         rb.isKinematic = true;
232         pickUpScript.enabled = false;
233         gunCollider.enabled = false;
234     }
```

Εικόνα 86 HideEquipment, ShowEquipment και SetupGunComponents

Στην συνάρτηση της *Update* (Εικόνα 87) πρώτα ελέγχεται αν το PhotonView ανήκει στον παίκτη, και μετά αν ο παίκτης ρίξει ή εναλλάξει το όπλο του για να κληθούν οι ανάλογες συναρτήσεις *UnequipGun()* και *EquipGun()*.

```
162     void Update()
163     {
164         if (!pv.IsMine)
165             return;
166 
167         if (dropAction.triggered && minimumGunAmount(1) && !PlayerManager.instance.player.GetComponent<MeleeController>().isAttacking)
168         {
169             UnequipGun();
170         }
171         // Min number of guns = 1, Max number of guns = gunContainer.childCount - 1
172         if (nextWeaponAction.triggered)
173         {
174             if (minimumGunAmount(2))
175             {
176                 inventory.Remove(0);
177                 inventory.Add(gunContainer.childCount - 1, currentEquipment);
178 
179                 currentGun.transform.SetAsLastSibling();
180                 EquipGun(gunContainer.GetChild(0).gameObject, true);
181             }
182         }
183         if (previousWeaponAction.triggered)
184         {
185             if (minimumGunAmount(2))
186             {
187                 gunContainer.GetChild(gunContainer.childCount - 1).SetAsFirstSibling();
188                 EquipGun(gunContainer.GetChild(0).gameObject, true);
189 
190                 inventory.Remove(gunContainer.childCount - 1);
191                 inventory.Add(0, currentEquipment);
192             }
193         }
194     }
```

Εικόνα 87 Πτώση ή εναλλαγή όπλου

Στην συνάρτηση *UnequipGun()* ελέγχουμε αν υπάρχουν πάνω από δύο όπλα, ώστε αν ρίξει το όπλο του ο παίκτης, να στείλει ένα event σε όλους τους παίκτες, για να εξοπλιστεί με το αμέσως επόμενο με την συνάρτηση *EquipGun()* (Εικόνα 88).

```
85      public void UnequipGun()
86      {
87          gunPv = currentGun.GetPhotonView();
88          inventory.Remove(0);
89 
90          GameObject nextGun;
91          int nextGunCode;
92          bool hasNextGun = minimumGunAmount(2);
93 
94          if (hasNextGun)
95          {
96              nextGun = gunContainer.GetChild(1).gameObject;
97              nextGunCode = nextGun.GetComponent<Gun>().gunInfo.GunCode;
98          }
99          else
100         {
101             nextGunCode = -1;
102             nextGun = null;
103         }
104 
105         object[] content = new object[] { gunPv.ViewID, nextGunCode };
106         RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.All };
107         PhotonNetwork.RaiseEvent(UnequipWeaponEventCode, content, raiseEventOptions, SendOptions.SendReliable);
108 
109         if (hasNextGun)
110         {
111             EquipGun(nextGun, false);
112         }
113     }
```

Εικόνα 88 Unequip του όπλου

Στην συνάρτηση *OnEvent* που είναι εγγεγραμμένη η EquipmentManager.cs λαμβάνονται όλα τα events που στέλνονται από τις συναρτήσεις. Όταν ο κωδικός είναι ο *HideEquipmentEventCode,* γίνεται η εύρεση του όπλου μέσω του id, και απενεργοποιείται το μοντέλο του όπλου (Εικόνα 89). Όταν ο κωδικός είναι ο *ShowEquipmentEventCode*, γίνεται η εύρεση του όπλου μέσω του id, και ενεργοποιείται το μοντέλο του όπλου (Εικόνα 89).

```
236     private void OnEvent(EventData photonEvent)
237     {
238         byte eventCode = photonEvent.Code;
239         if (eventCode == HideEquipmentEventCode)
240         {
241             object[] data = (object[])photonEvent.CustomData;
242             int id = (int)data[0];
243 
244             GameObject currentGunTemp = PhotonView.Find(id).gameObject;
245             Transform weapons = currentGunTemp.transform.parent.GetComponent<EquipmentManager>().weaponsContainer;
246 
247             int code = currentGunTemp.GetComponent<Gun>().gunInfo.GunCode;
248             currentGunPrefab = weapons.GetChild(code).gameObject;
249 
250             currentGunPrefab.SetActive(false);
251             currentGunTemp.SetActive(false);
252         }
253         else if (eventCode == ShowEquipmentEventCode)
254         {
255             object[] data = (object[])photonEvent.CustomData;
256             int id = (int)data[0];
257 
258             GameObject currentGunTemp = PhotonView.Find(id).gameObject;
259             Transform weapons = currentGunTemp.transform.parent.GetComponent<EquipmentManager>().weaponsContainer;
260 
261             int code = currentGunTemp.GetComponent<Gun>().gunInfo.GunCode;
262             currentGunPrefab = weapons.GetChild(code).gameObject;
263 
264             currentGunPrefab.SetActive(true);
265             currentGunTemp.SetActive(true);
266         }
```

Εικόνα 89 Συγχρονισμός εμφάνισης και εξαφάνισης όπλου

Στην περίπτωση που ο κωδικός είναι ο *SetupGunComponentsEventCode*, γίνεται η εύρεση του όπλου, και η ενεργοποίηση και απενεργοποίηση των κατάλληλων components (Εικόνα 90).

```
304         else if (eventCode == SetupGunComponentsEventCode)
305         {
306             object[] data = (object[])photonEvent.CustomData;
307             int id = (int)data[0];
308 
309             Transform currentGunTemp = PhotonView.Find(id).transform;
310             currentGunTemp.transform.GetChild(0).gameObject.SetActive(false);
311 
312             rb = currentGunTemp.GetComponent<Rigidbody>();
313             gunScript = currentGunTemp.GetComponent<Gun>();
314             pickUpScript = currentGunTemp.GetComponent<GunPickUp>();
315             gunCollider = currentGunTemp.GetComponent<Collider>();
316 
317             gunScript.enabled = true;
318             gunScript.setGunParent(true);
319             rb.isKinematic = true;
320             pickUpScript.enabled = false;
321             gunCollider.enabled = false;
322         }
```

Εικόνα 90 Συγχρονισμός των components του όπλου

Όταν ο κωδικός είναι ο *UnequipWeaponEventCode*, γίνεται η εύρεση του όπλου μέσω του id, ενεργοποιεί και απενεργοποιεί τα αντίθετα components από την περίπτωση του *SetupGunComponents*, και πετάει το όπλο με μικρή δύναμη μπροστά από τον παίκτη (Εικόνα 91).

```
267         else if (eventCode == UnequipWeaponEventCode)
268         {
269             object[] data = (object[])photonEvent.CustomData;
270             int id = (int)data[0];
271             int nextGunCode = (int)data[1];
272 
273             Transform currentGunTemp = PhotonView.Find(id).transform;
274             currentGunTemp.GetChild(0).gameObject.SetActive(true);
275 
276             int code = currentGunTemp.GetComponent<Gun>().gunInfo.GunCode;
277 
278             if (weaponsContainer != null)
279                 weaponsContainer.GetChild(code).gameObject.SetActive(false);
280 
281             if (code != nextGunCode)
282                 currentGunPrefab = null;
283 
284             currentGunTemp.SetParent(null);
285 
286             rb = currentGunTemp.GetComponent<Rigidbody>();
287             gunScript = currentGunTemp.GetComponent<Gun>();
288             pickUpScript = currentGunTemp.GetComponent<GunPickUp>();
289             gunCollider = currentGunTemp.GetComponent<Collider>();
290 
291             gunScript.setGunParent(false);
292             gunScript.enabled = false;
293             rb.isKinematic = false;
294             pickUpScript.enabled = true;
295             gunCollider.enabled = true;
296 
297             cam = Camera.main.transform;
298             rb.AddForce(cam.forward * dropForwardForce, ForceMode.Impulse);
299             rb.AddForce(cam.up * dropUpwardForce, ForceMode.Impulse);
300 
301             if (OnEquippedCallback != null)
302                 OnEquippedCallback.Invoke();
303         }
```

Εικόνα 91 Συγχρονισμός όταν ένα όπλο είναι unequipped

Το script Gun.cs ελέγχει τον τρόπο λειτουργίας του όπλου όπως τον πυροβολισμό, το γέμισμα του όπλου, τον ρυθμό ταχυβολίας και την στόχευση. Είναι σημαντικό να διαχωριστεί ο κώδικας μεταξύ τον παικτών ώστε να πυροβολεί μόνο ο παίκτης στον οποίο ανήκει το όπλο με τον έλεγχο του PhotonView (Εικόνα 92).

```
70      void Update()
71      {
72          if (!PlayerPv.IsMine)
73              return;
```

Εικόνα 92 Έλεγχος PhotonView στο Gun.cs

Στην συνέχεια της *Update* (Εικόνα 93), γίνονται οι απαραίτητοι έλεγχοι, για την κατάσταση του παίκτη είτε αυτός βρίσκεται σε κάποιο animation όπως του dash ή των πολλαπλών αλμάτων είτε του χτυπήματος με το σπαθί. Ελέγχεται επίσης εάν το όπλο έχει ξεμείνει από σφαίρες, ή αν είναι έτοιμο να πυροβολήσει. Μετά τον υπολογισμό της κατεύθυνσης που πρέπει να πάρει η σφαίρα, στέλνεται ένα RPC με την εκτέλεση της συνάρτησης *ShootGun()* και παράμετρο την κατεύθυνση της σφαίρας, την ζημιά και το id του παίκτη προς όλους τους παίκτες.

```
78      timer -= Time.deltaTime;
79
80      // Can't shoot when the player is:
81      // Reloading, dashing, multi jumping, or meleeing
82      if (isReloading || playerController.isDashing || playerController.multipleJumps || playerMeleeController.isAttacking)
83          return;
84
85      // Reload when out of ammo, or manually reloading
86      if (currentAmmo <= 0 || (reloadAction.triggered && currentAmmo < maxAmmo))
87      {
88          StartCoroutine(Reload());
89          return;
90      }
91
92      if (shooting && timer <= 0)
93      {
94          shakeScript.ShakeCamera(gunInfo.ShakePower);    // Shake the screen on client side
95          timer = fireRate;
96          isShooting = true;
97
98          Vector3 mouseWorldPosition = Vector3.zero;  // Reseting mouse world position
99          Vector2 screenCenterPoint = new Vector2(Screen.width / 2f, Screen.height / 2f); // Crosshair is on the center of the screen always
100         Ray ray = Camera.main.ScreenPointToRay(screenCenterPoint);  // Shooting a ray at the center of the screen
101         if (Physics.Raycast(ray, out RaycastHit raycastHit, 999f, aimColliderLayerMask))
102         {
103             mouseWorldPosition = raycastHit.point;  // Saving the point the ray collided with something
104         }
105
106         // Inserting gun spread
107         Vector3 randomSpread = new Vector3(Random.Range(-gunInfo.Spread, gunInfo.Spread), Random.Range(-gunInfo.Spread, gunInfo.Spread), Random.Range(-gunInfo.Spread, gunInfo.Spread));
108         // Shot direction
109         Vector3 aimDir = (mouseWorldPosition - fireposition.position + randomSpread).normalized;
110         // Shoot through the network and send direction and gun damage
111         gunPv.RPC("ShootGun", RpcTarget.All, aimDir, damage, PlayerPv.ViewID);
112
113         if (!gunInfo.IsAutomatic)   // Checking if gun is automatic
114             shooting = false;
115     }
116     else
117     {
118         isShooting = false; // needs to be set false to not spam shooting animation
119     }
```

Εικόνα 93 Περιορισμοί πυροβολισμού

Στην συνάρτηση *ShootGun()* (Εικόνα 94) γίνεται ο έλεγχος για το είδος του όπλου καθώς στην περίπτωση που το όπλο είναι καραμπίνα θα πρέπει να δημιουργηθούν πολλαπλές σφαίρες. Ασχέτως της περίπτωσης, αποθηκεύουμε το script Projectile.cs της σφαίρας και θέτουμε την ζημιά, καθώς και τον μεταβλητή που δηλώνει τον κάτοχο της σφαίρας.

```
122     [PunRPC]
        0 references
123     private void ShootGun(Vector3 direction, float damage, int owner)
124     {
125         if (gunInfo.IsShotgun)
126         {
127             for (int i = 0; i < gunInfo.Pellets; i++)
128             {
129                 Vector3 spreadDirection = direction + new Vector3(Random.Range(-maxSpread, maxSpread), Random.Range(-maxSpread, maxSpread), Random.Range(-maxSpread, maxSpread));
130                 GameObject bullet = ObjectPooling.instance.GetBullet(gunInfo.GunCode);
131
132                 if (bullet != null)
133                 {
134                     bullet.transform.position = fireposition.position;
135                     bullet.transform.rotation = Quaternion.LookRotation(spreadDirection);
136                     Projectile pr = bullet.GetComponent<Projectile>();
137                     pr.SetDamage(damage);
138                     pr.owner = owner;
139                     bullet.SetActive(true);
140                 }
141             }
142         }
143         else
144         {
145             GameObject bullet = ObjectPooling.instance.GetBullet(gunInfo.GunCode);
146             if (bullet != null)
147             {
148                 bullet.transform.position = fireposition.position;
149                 bullet.transform.rotation = Quaternion.LookRotation(direction);
150                 Projectile pr = bullet.GetComponent<Projectile>();
151                 pr.SetDamage(damage);
152                 pr.owner = owner;
153                 bullet.SetActive(true);
154             }
155         }
156
157         if (playAudio)
158         {
159             m_audio.Play(); // Play gun sound effect
160         }
161         currentAmmo--;
162     }
```

Εικόνα 94 Συγχρονισμός πυροβολισμού

Κατά την σύγκρουση της σφαίρας, γίνεται ο έλεγχος του αντικειμένου με τον οποίο συγκρούεται. Στην περίπτωση που η σφαίρα πετύχει κάποιον εχθρό, και εφόσον αναπαραχθούν τα κατάλληλα οπτικά εφέ, αποθηκεύεται το script EnemyStats.cs του εχθρού (Εικόνα 95). Αν το script υπάρχει, τότε γίνεται ένας επιπλέον έλεγχος, όπου συγκρίνεται ο κάτοχος της σφαίρας με τον παίκτη, ώστε μόνο όταν ο κάτοχος της σφαίρας πετυχαίνει τον εχθρό να σταλθεί ένα RPC στο PhotonView του εχθρού για να κληθεί η συνάρτηση *RPC_TakeDamage()* με παραμέτρους την ζημιά και τον κάτοχο της σφαίρας. Ο έλεγχος αυτός συμβαίνει, για να μην μειώνονται οι πόντοι ζωής του εχθρού για κάθε παίκτη που υπάρχει στο παιχνίδι, αλλά μόνο μια φορά, όταν ο κάτοχος της σφαίρας πετυχαίνει τον εχθρό.

```
83      private void OnCollisionEnter(Collision col)
84      {
85          if (col.gameObject.tag == "Ground" || col.gameObject.tag == "Defence")
86          {
87              ParticleSystem hi = Instantiate(hitImpactVFX, transform);
88              hi.transform.SetParent(null);
89              hi.Play();
90
91              gameObject.SetActive(false);
92          }
93          if (col.gameObject.tag == "Enemy")
94          {
95              GameObject damagePopUp = Instantiate(damageTextPrefab, transform.position, Quaternion.identity);
96              damagePopUp.transform.SetParent(col.transform);
97              damagePopUp.GetComponent<DamagePopUp>().Initialise(Mathf.Floor(damage));
98
99              ParticleSystem hi = Instantiate(hitImpactVFX, transform);
100             hi.transform.SetParent(null);
101             hi.Play();
102
103             gameObject.SetActive(false);
104
105             EnemyStats enemyHealth = col.gameObject.GetComponent<EnemyStats>();
106             if (enemyHealth != null)
107             {
108                 PhotonView bulletOWner = PhotonView.Find(owner);
109                 if (bulletOWner.IsMine)
110                 {
111                     col.gameObject.GetPhotonView().RPC("RPC_TakeDamage", RpcTarget.All, damage, owner);
112                 }
113             }
114         }
115     }
```

Εικόνα 95 Σύγκρουση σφαίρας

4.2.4 Συγχρονισμός εχθρών

Οι εχθροί δημιουργούνται και κυνηγάνε τους παίκτες για να τους χτυπήσουν. Ο συγχρονισμός τους γίνεται κυρίως από τον host όσον αφορά τον προορισμό τους, πότε εξαφανίζονται από το περιβάλλον και πότε επιτίθενται. Τα script που συγχρονίζουν τους εχθρούς είναι τα: EnemyAI.cs, EnemySpawner.cs και EnemyStats.cs.

Το script EnemyStats.cs περιέχει όλα τα στατιστικά του εχθρού. Για την κλιμάκωση των στατιστικών αυτών, στέλνεται ένα RPC από τις συναρτήσεις των γύρων, για να εκτελεστεί η *ScaleStats()* με παράμετρο την δυσκολία του γύρου (Εικόνα 96).

```
121     [PunRPC]
        0 references
122     private void ScaleStats(int difficulty)
123     {
124         maxHealth += maxHealth * difficulty / 2 + maxHealth * (PhotonNetwork.PlayerList.Length - 1) * 0.75f;
125         damage += (int)((damage * difficulty / 3) + damage * (PhotonNetwork.PlayerList.Length - 1) * 0.5f);
126         loot += (int)((loot * difficulty / 3) + loot * (PhotonNetwork.PlayerList.Length - 1) * 0.5f);
127 
128     }
```

Εικόνα 96 Κλιμάκωση στατιστικών εχθρού

Όταν ο εχθρός έρχεται σε σύγκρουση με τις σφαίρες των παικτών εκτελείται η συνάρτηση *TakeDamage() (Εικόνα 97)* η οποία με τη σειρά της στέλνει ένα RPC για να εκτελεστεί η συνάρτηση *RPC_TakeDamage()* με παραμέτρους την ζημιά και το id του παίκτη που προκάλεσε την ζημιά. Στην συνάρτηση αυτή, μειώνονται οι πόντοι ζωής από τον εχθρό για όλους τους παίκτες ανάλογα με την ζημιά της σφαίρας. Επίσης άμα χάσει ο εχθρός, τότε αυξάνεται ο μετρητής *kills* μόνο για τον παίκτη που τελευταία χτύπησε τον εχθρό, και εκτελείται η συνάρτηση *EnemyDeath() (Εικόνα 98).*

```
66      public void TakeDamage(float damage, int owner)
67      {
68          pv.RPC("RPC_TakeDamage", RpcTarget.AllViaServer, damage, owner);
69      }
70 
71      [PunRPC]
        0 references
72      private void RPC_TakeDamage(float damage, int owner)
73      {
74          lerpTimer = 0;
75          health -= damage;
76 
77          if (m_audio)
78              m_audio.Play();
79 
80          if (health <= 0 && !isDead)
81          {
82              isDead = true;
83              PhotonView playerPv = PhotonView.Find(owner);
84              if (playerPv.IsMine)
85              {
86                  PlayerStats pStats = playerPv.gameObject.GetComponent<PlayerStats>();
87                  pStats.kills += 1;
88              }
89 
90              if (!PlayerManager.instance.isDead)
91              {
92                  ParticleSystem particles = Instantiate(lootParticles, transform);
93 
94                  Vector3 particlesHeight = new Vector3(transform.position.x, 5f, transform.position.z);
95                  particles.transform.position = particlesHeight;
96                  particles.transform.parent = null;
97                  particles.Play();
98              }
99 
100             health = 0;
101 
102             StartCoroutine(EnemyDeath());
103         }
104     }
```

Εικόνα 97 Μείωση ζωής εχθρού

Η συνάρτηση *EnemyDeath()* (Εικόνα 98) εκτελείται μόνο από τον host, όπου καταστρέφεται το αντικείμενο του εχθρού με την συνάρτηση *PhotonNetwork.Destroy()* και με παράμετρο το PhotonView του εχθρού.

```
111     IEnumerator EnemyDeath()
112     {
113         yield return new WaitForSeconds(1);
114
115         if (PhotonNetwork.IsMasterClient)
116         {
117             PhotonNetwork.Destroy(gameObject.GetPhotonView());
118         }
119     }
```

Εικόνα 98 Θάνατος εχθρού

Το script EnemySpawner.cs έχει ως στόχο την δημιουργία των εχθρών. Η συνάρτηση *SpawnEnemy()* (Εικόνα 99) καλείται από τα script των γύρων, για να δίνουν τις παραμέτρους του είδους του εχθρού, της δυσκολίας, και της συμπεριφοράς τους εχθρού. Αρχικά υπολογίζεται η τυχαία τοποθεσία του εχθρού. Μετά δημιουργείται με την συνάρτηση *PhotonNetwork.Instantiate()*. Έπειτα θέτονται τα στατιστικά του εχθρού στέλνοντας ένα RPC προς όλους τους παίκτες για να εκτελέσουν την συνάρτηση *ScaleStats()* με παράμετρο την δυσκολία. Επίσης, με την χρήση του script EnemyAI.cs ορίζεται η συμπεριφορά του εχθρού. Τέλος στέλνεται ένα event σε όλους τους παίκτες για την εμφάνιση των οπτικών εφέ όταν δημιουργείται ο εχθρός.

```
26      public void SpawnEnemy(int enemyIndex, int difficulty, int behaviour)
27      {
28          float x = Random.Range(-30f, 30f);
29          float z = Random.Range(-10f, 10f);
30          Vector3 randomPosition = new Vector3(x, 3, z);
31          Vector3 enemySpawner = spawnArea.position + randomPosition;
32
33          GameObject enemy = PhotonNetwork.Instantiate("Enemies/" + enemyPrefabs[enemyIndex].name, enemySpawner, Quaternion.identity);
34          enemy.GetPhotonView().RPC("ScaleStats", RpcTarget.All, difficulty);
35          enemy.GetComponent<EnemyAI>().SetTarget(behaviour);
36
37          enemySpawned = enemy;
38          // Sending the position for the portal spawn VFX
39          object[] content = new object[] { enemySpawner };
40          RaiseEventOptions raiseEventOptions = new RaiseEventOptions { Receivers = ReceiverGroup.All };
41          PhotonNetwork.RaiseEvent(EnemySpawnCode, content, raiseEventOptions, SendOptions.SendReliable);
42      }
43
        2 references
44      public void OnEvent(EventData photonEvent)
45      {
46          byte eventCode = photonEvent.Code;
47          if (eventCode == EnemySpawnCode)
48          {
49              object[] data = (object[])photonEvent.CustomData;
50              Vector3 enemySpawner = (Vector3)data[0];
51
52              // VFX portal spawn
53              Vector3 vfxPosition = new Vector3(enemySpawner.x, enemySpawner.y - 1.5f, enemySpawner.z);
54              GameObject go = new GameObject();
55              GameObject ph = Instantiate(go, vfxPosition, Quaternion.identity);
56              ParticleSystem spawnVfx = Instantiate(spawnVFX, ph.transform);
57              spawnVfx.Play();
58              Destroy(go, 1);
59              Destroy(ph, 1);
60          }
61      }
```

Εικόνα 99 Δημιουργία εχθρού

Το script EnemyAI.cs ρυθμίζει την συμπεριφορά των εχθρών, όπως τον προορισμό τους, την κίνησή τους και την επίθεση. Μέσα στην συνάρτηση της *Update* (Εικόνα 100) , μόνο ο host ελέγχει αν η συμπεριφορά του εχθρού ισούται με 0 και αν ισχύει καλείται η συνάρτηση *SeekTarget().* Μετά υπολογίζεται η περιστροφή του εχθρού, το animation που πρέπει να εκτελείται και η περίπτωση που ο εχθρός πρέπει να επιτεθεί. Στην περίπτωση της επίθεσης γίνονται έλεγχοι πολλών συνθηκών και τελικά μπορούν να αποσταλούν δύο είδη RPC ανάλογα με το είδος της επίθεσης του εχθρού, αν επιτίθεται από κοντά ή μακριά. Οι συναρτήσεις αυτές είναι οι *StartMeleeAttack()* και *StartFlyingRangedAttack()* αντίστοιχα. Στην περίπτωση της *StartFLyingRangedAttack()* στέλνεται και η παράμετρος της κατεύθυνσης από την σφαίρα του εχθρού.

```
137         if (PhotonNetwork.IsMasterClient)
138         {
139             // Default behaviour, chase the closest player
140             if (behaviour == 0)
141             {
142                 SeekTarget();
143             }
144 
145             // Rotating enemy
146             if (canRotate && destination != null)
147             {
148                 Vector3 dir = destination.transform.position - transform.position;  // Finding direction
149                 Quaternion lookRotation = Quaternion.LookRotation(dir); // Look direction in quaternion
150                 Vector3 rotation = lookRotation.eulerAngles;    // Converting it to eulerAngles and storing it in a vector3
151                 agent.transform.rotation = Quaternion.Euler(0f, rotation.y, 0f);    // Setting the necessary rotation to the enemy
152             }
153 
154             if (agent.velocity.magnitude > 0.1f)
155             {
156                 animator.SetBool("isRunning", true);
157             }
158             else
159             {
160                 animator.SetBool("isRunning", false);
161             }
162 
163             // If enemy is in attacking range
164             if (agent.remainingDistance < attackingDistance && agent.isStopped == false && !agent.pathPending && canAttack)
165             {
166                 if (attackCooldownTimer < 0)
167                 {  // Check attack cooldown of the enemy
168                     if (isMelee)
169                     {    // If the enemy is a melee humanoid
170                         attackCooldownTimer = attackCooldown;
171                         pv.RPC("StartMeleeAttack", RpcTarget.AllViaServer);
172                     }
173                     else if (isFlying && isRanged)
174                     {   // If the enemy is flying and ranged
175                         attackCooldownTimer = attackCooldown;
176 
177                         Vector3 higherAim = new Vector3(0, 1, 0);
178                         Vector3 aimDir = (destination.transform.position - firepoint.position + higherAim).normalized;
179                         pv.RPC("StartFlyingRangedAttack", RpcTarget.AllViaServer, aimDir);
180                     }
181                 }
182             }
```

Εικόνα 100 Γενική συμπεριφορά εχθρών

Στην συνάρτηση *SeekTarget()* (Εικόνα 101) γίνεται η εύρεση των παικτών και υπολογίζεται η ελάχιστη απόσταση μεταξύ τους, για να οριστεί ο προορισμός των εχθρών. Ελέγχεται ο στόχος των εχθρών ώστε μόνο όταν ένας διαφορετικός παίκτης πλησιάσει πιο κοντά, να στέλνεται ένα RPC προς όλους τους παίκτες μέσω του διακομιστή για την εκτέλεση της συνάρτησης *RPC_SetDestination()* με την παράμετρο το id του παίκτη που πρέπει να κυνηγήσει ο εχθρός.

```
266     private void SeekTarget()
267     {
268         if (behaviour == 0)
269         {
270             // Finding all the players
271             players = GameObject.FindGameObjectsWithTag("Player");
272             // Finding the closest player
273             float minDist = Mathf.Infinity;
274             float tempMinDist = minDist;
275             if (players.Length > 0)
276             {
277                 foreach (GameObject p in players)
278                 {
279                     minDist = Mathf.Min(minDist, Vector3.Distance(transform.position, p.transform.position));
280                     if (minDist < tempMinDist)
281                     {
282                         tempMinDist = minDist;
283                         playerToHunt = p;
284                     }
285                 }
286                 // Check if the closest player found is different than the already hunted one
287                 if (destination != playerToHunt)
288                 {
289                     destination = playerToHunt;
290                     PhotonView playerPv = playerToHunt.GetComponent<PhotonView>();
291                     pv.RPC("RPC_SetDestination", RpcTarget.AllViaServer, playerPv.ViewID);
292                 }
293             }
294             else
295             {
296                 //agent.SetDestination(transform.position);
297                 agent.isStopped = true;
298                 canAttack = false;
299             }
300         }
301         else if (behaviour == 1)
302         {
303             attackingDistance = defenceAttackingDistance;
304             destination = GameObject.FindGameObjectWithTag("Defence");
305             agent.stoppingDistance = attackingDistance - 2f;
306         }
307     }
```

Εικόνα 101 Επιλογή προορισμού εχθρού

Στην συνάρτηση *RPC_SetDestination()* μεταβάλλεται ο προορισμός του εχθρού με βάση το id του παίκτη (Εικόνα 102).

```
257     [PunRPC]
        0 references
258     private void RPC_SetDestination(int playerId, PhotonMessageInfo info)
259     {
260         EnemyAI enemyAI = info.photonView.GetComponent<EnemyAI>();
261         enemyAI.destination = PhotonView.Find(playerId).gameObject;
262         NavMeshAgent enemyAgent = info.photonView.GetComponent<NavMeshAgent>();
263         enemyAgent.SetDestination(enemyAI.destination.transform.position);
264     }
```

Εικόνα 102 Ορισμός προορισμού εχθρού

Τέλος, όταν εντοπιστεί στην συνάρτηση της *Update* ότι έχει χάσει ο εχθρός, εκτελείται ο κώδικας μόνο από την μεριά του host (Εικόνα 103) για την τυχαία επιλογή του power up που θα αφήσει πίσω ο εχθρός. Η δημιουργία των power ups γίνεται με την συνάρτηση *PhotonNetwork.Instantiate().*

```
94          if (PhotonNetwork.IsMasterClient)
95          {
96              float dropPowerUpChance = Random.Range(0f, 1f);
97              if (dropPowerUpChance < chanceToDrop)
98              {
99                  float powerUpRarity = Random.Range(0f, 1f);
100
101                 GameObject powerUp = null;
102
103                 if (powerUpRarity < AllPowerUps.commonChance)
104                 {
105                     int randomPowerUp = Random.Range(0, allPowerUps.commonPowerUps.Length);
106                     powerUp = PhotonNetwork.Instantiate("PowerUps/" + allPowerUps.commonPowerUps[randomPowerUp].name, transform.position, Quaternion.identity);
107                 }
108                 else if (powerUpRarity < AllPowerUps.uncommonChance)
109                 {
110                     int randomPowerUp = Random.Range(0, allPowerUps.uncommonPowerUps.Length);
111                     powerUp = PhotonNetwork.Instantiate("PowerUps/" + allPowerUps.uncommonPowerUps[randomPowerUp].name, transform.position, Quaternion.identity);
112                 }
113                 else if (powerUpRarity < AllPowerUps.rareChance)
114                 {
115                     int randomPowerUp = Random.Range(0, allPowerUps.rarePowerUps.Length);
116                     powerUp = PhotonNetwork.Instantiate("PowerUps/" + allPowerUps.rarePowerUps[randomPowerUp].name, transform.position, Quaternion.identity);
117                 }
118                 else if (powerUpRarity <= AllPowerUps.legendaryChance)
119                 {
120                     int randomPowerUp = Random.Range(0, allPowerUps.legendaryPowerups.Length);
121                     powerUp = PhotonNetwork.Instantiate("PowerUps/" + allPowerUps.legendaryPowerups[randomPowerUp].name, transform.position, Quaternion.identity);
122                 }
123
124                 if (powerUp != null)
125                 {
126                     powerUp.transform.position = new Vector3(transform.position.x, transform.position.y + 2, transform.position.z);
127                     powerUp.transform.SetParent(null);
128
129                     Rigidbody powerRb = powerUp.GetComponent<Rigidbody>();
130                     int dropUpwardForce = 5;
131                     powerRb.AddForce(Vector3.up * dropUpwardForce, ForceMode.Impulse);
132                 }
133             }
134         }
```

Εικόνα 103 Πτώση power up από τον εχθρό

4.2.5 Συγχρονισμός κουτιών και power ups

Στο script PowerUpPickUp.cs γίνεται η διαχείριση της αλληλεπίδρασης με τα power ups. Για την απόκτηση ενός power up, όταν ο παίκτης αλληλεπιδρά με αυτό, στέλνεται το RPC (Εικόνα 104) για την κλήση της συνάρτησης *RPC_AddPowerUp()* προς όλους τους παίκτες. Δεν χρειάζεται η εύρεση του id του παίκτη γιατί τα power ups προστίθενται σε όλους τους παίκτες.

```
17      public override void Interact(int id)
18      {
19          player = PhotonView.Find(id).gameObject;
20          base.Interact(id);
21
22          pv.RPC("RPC_AddPowerUp", RpcTarget.All);
23      }
24
25      [PunRPC]
        0 references
26      private void RPC_AddPowerUp()
27      {
28          powerUp.Effect(PlayerManager.instance.player);
29          StartCoroutine(DestroyPowerUp());
30      }
```

Εικόνα 104 Απόκτηση power up

Στο script ChestController.cs γίνεται η διαχείριση της αλληλεπίδρασης με τα κουτιά όπλων. Όταν ο παίκτης αλληλεπιδρά με το κουτί όπλων (Εικόνα 105), αρχίζει το animation του

κουτιού και στέλνεται ένα RPC προς όλους τους παίκτες μέσω του διακομιστή για την εκτέλεση της συνάρτησης *RPC_OpenChest().* Μέσα στην συνάρτηση αυτή γίνεται η κλήση της συνάρτησης *SpawnGunDelay()* όπου τυχαία επιλέγεται το όπλο που θα δημιουργηθεί στο παιχνίδι με την ίδια λογική που επιλέγονται και δημιουργούνται τα power ups.

```
34      public override void Interact(int id)
35      {
36          if (interactable == false)
37              return;
38
39          interactable = false;
40          animator.SetBool("openingChest", true);
41          pv.RPC("RPC_OpenChest", RpcTarget.AllViaServer);
42      }
43
44      [PunRPC]
        0 references
45      public void RPC_OpenChest()
46      {
47          interactable = false;
48          animator.SetBool("openingChest", true);
49          chestSound.Play();
50
51          foreach (Collider col in cols)
52          {
53              col.enabled = false;
54          }
55
56          if (isGunChest)
57          {
58              StartCoroutine(SpawnGunDelay());
59          }
60
61          if (isPowerUpChest)
62          {
63              StartCoroutine(SpawnPowerUpDelay());
64          }
65
66          Destroy(gameObject, 2);
67          spawnVfx.Stop();
68          Destroy(spawnVfx.gameObject, 1);
69      }
```

Εικόνα 105 Άνοιγμα κουτιού όπλων

4.2.4 Διαχωρισμός κώδικα παικτών

Τέλος, υπάρχουν μερικά scripts όπου για την αποφυγή προβλημάτων κατά τον συγχρονισμό τους χρειάζονται μια απλή συνθήκη, που ελέγχει ότι το PhotonView ανήκει στον παίκτη με την ιδιότητα *IsMine*, όπως και σε πολλά προηγούμενα script.

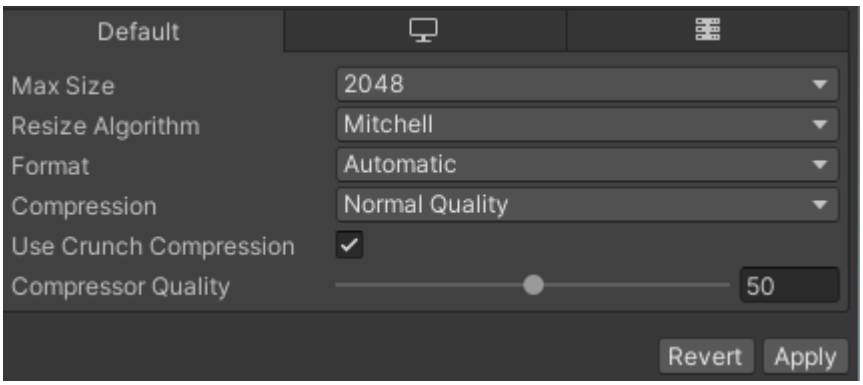
4.3 Βελτιστοποίηση

4.3.1 Φωτισμός

Τα είδη του φωτισμού είναι το Realtime, το Baked, και το Mixed. Ο Realtime φωτισμός ενεργοποιεί τον φωτισμό πραγματικού χρόνου, δηλαδή δημιουργούνται αντανακλάσεις φωτός όταν αντικείμενα πλησιάζουν πηγές φωτός. Ο Baked φωτισμός αποθηκεύει τον φωτισμό στα textures του αντικειμένου, έτσι ώστε να μην πραγματοποιούνται οι υπολογισμοί του φωτός κατά την διάρκεια του παιχνιδιού. Ο Mixed φωτισμός είναι μια ενδιάμεση λύση όπου μερικά αντικείμενα έχουν αποθηκευμένα τον φωτισμό στα textures τους όπως στον Baked φωτισμό, καθώς άλλα αντικείμενα πραγματοποιούν υπολογισμούς για την αντανάκλαση του φωτισμού σε πραγματικό χρόνο, όπως στον Realtime φωτισμό. Για να ελαχιστοποιείται η επιβάρυνση του συστήματος, χρησιμοποιήθηκε Baked φωτισμός για όλες τις πηγές φωτισμού.

4.3.2 Αποθηκευτικός χώρος

Το μέγεθος του παιχνιδιού χωρίς καμία χρήση συμπίεσης ανερχόταν στα 882MB και το 99% από το μέγεθος προερχόταν από τα textures. Χρησιμοποιήθηκε η συμπίεση που παρέχεται από την Unity (Εικόνα 106) σε όλα τα textures του παιχνιδιού. Μετά την συμπίεση, το παιχνίδι μειώθηκε στα μόλις 334MB (Εικόνα 107).



Εικόνα 106 Συμπίεση των textures

```
Build Report
Uncompressed usage by category (Percentages based on user generated assets only):
Textures                196.6 mb  75.5%
Meshes                  24.2 mb   9.3%
Animations              8.1 mb    3.1%
Sounds                  24.0 mb   9.2%
Shaders                 1.4 mb    0.5%
Other Assets            5.7 mb    2.2%
Levels                  0.0 kb    0.0%
Scripts                 169.6 kb  0.1%
Included DLLs           0.0 kb    0.0%
File headers            226.8 kb  0.1%
Total User Assets       260.4 mb  100.0%
Complete build size     333.9 mb
```

Εικόνα 107 Τελικό μέγεθος παιχνιδιού

4.3.3 Object Pooling

Κατά την διάρκεια του παιχνιδιού, όσο αυξάνεται η ταχυβολία των όπλων, δημιουργούνται και καταστρέφονται όλο και περισσότερες σφαίρες το δευτερόλεπτο. Ειδικά στην περίπτωση που παίζουν ταυτόχρονα πολλαπλοί παίκτες, ο αριθμός από τα αντικείμενα μπορούν να μειώσουν τις επιδόσεις του παιχνιδιού. Μία λύση στο πρόβλημα της επίδοσης είναι η χρήση ενός Object pool. Με ένα Object pool μπορούμε να αντικαταστήσουμε την δημιουργία και την καταστροφή αντικειμένων, με την ενεργοποίηση και απενεργοποίηση αντικειμένων.

Στην αρχή του παιχνιδιού, δημιουργείται το αντικείμενο σε έναν υψηλό αριθμό και απενεργοποιείται, για να είναι διαθέσιμος οποιαδήποτε στιγμή. Όταν υπάρχει η ανάγκη εμφάνισης του αντικειμένου, το Object pool είναι υπεύθυνο για την ενεργοποίησή του. Ανάλογα με το αντικείμενο που δημιουργείται και την χρήση του, πρέπει σε ξεχωριστό σημείο να απενεργοποιηθεί το αντικείμενο και να «γυρίσει» πίσω στο Object pool. Στην περίπτωση που τελειώσει το απόθεμα από το αντικείμενο λόγω της υψηλής ανάγκης από το παιχνίδι, τότε δημιουργούνται καινούργια.

Στο παιχνίδι η χρήση του Object pooling έγινε με τις σφαίρες των όπλων, όπου στην αρχή του παιχνιδιού δημιουργούνται πολλές μαζί και απενεργοποιούνται. Όταν ο παίκτης πυροβολεί, οι σφαίρες ενεργοποιούνται και ακολουθούν την κανονική τους συμπεριφορά. Όταν η σφαίρα συγκρουστεί με ένα αντικείμενο, ή τελειώσει το χρονικό περιθώριό της, απενεργοποιείται ξανά.

4.4 Ασφάλεια

Δεν περιέχεται κάποια μορφή ασφάλειας καθώς το παιχνίδι δεν είναι ανταγωνιστικό, αλλά συνεργατικό. Δηλαδή οι παίκτες δεν εναντιώνονται μεταξύ τους, αλλά παίζουν μαζί ενάντια στους εχθρούς που δημιουργούνται από το παιχνίδι. Πέρα από αυτό, δεν υπάρχουν πίνακες κατάταξης για τα αποτελέσματα των παιχνιδιών, οπότε οι παίκτες που παίζουν δίκαια δεν θα αισθανθούν αδικημένοι από άλλους παίκτες που θα προσπαθούσαν να «χακάρουν» το παιχνίδι.

Στο επόμενο κεφάλαιο παρουσιάζονται τα αποτελέσματα από τους δοκιμαστές του παιχνιδιού και αναλύονται τα συμπεράσματα για το δικτυωμένο κομμάτι και για το σχεδιαστικό κομμάτι του παιχνιδιού χωριστά.

Κεφάλαιο 5: Αποτελέσματα

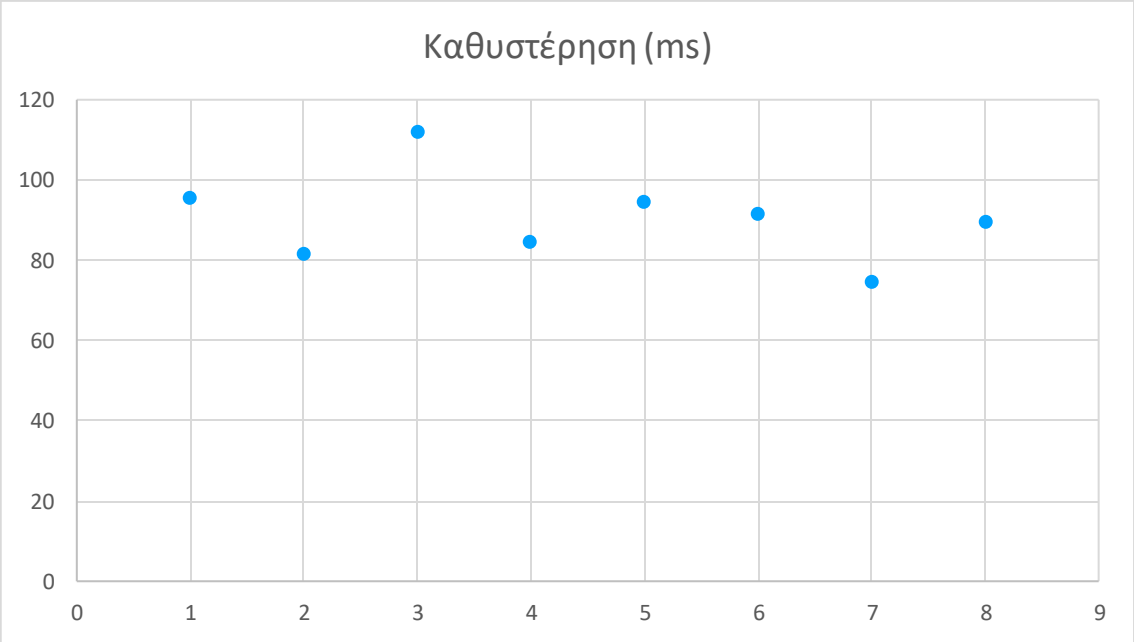
Σε αυτό το κεφάλαιο γίνεται αναφορά στα αποτελέσματα που προσκομίσθηκαν από το ερωτηματολόγιο που συμπλήρωσαν οι 8 δοκιμαστές του παιχνιδιού. Οι δοκιμαστές δώσανε συναίνεση για τη συμμετοχή τους στην αξιολόγηση και συμφώνησαν με τους κανονισμούς που δίνονται από την Επιτροπή Ηθικής Δεοντολογίας της Έρευνας (Ε.Η.Δ.Ε) του Πανεπιστήμιου Δυτικής Μακεδονίας [28]. Τα άτομα για τον δοκιμαστικό έλεγχο επιλέχθηκαν με κριτήρια την ηλικία, και την εμπειρία τους στα παιχνίδια. Με βάση προηγούμενες έρευνες η ηλικιακή ομάδα με τα περισσότερα άτομα που παίζουν παιχνίδια είναι μεταξύ 18 – 34 χρονών [29] οπότε οι δοκιμαστές ανήκουν σε αυτήν την ομάδα. Λόγω του στοχευμένου κοινού του παιχνιδιού, κρίθηκε αναγκαίο οι δοκιμαστές να έχουν εμπειρία με τα ψηφιακά παιχνίδια.

Αναλύονται τα σχεδιαστικά προβλήματα που εντοπίστηκαν από την επίδοση των παικτών, τα σφάλματα, και την καθυστέρηση που υπάρχει στη διακίνηση των μηνυμάτων μέσω του δικτύου.

Με την ολοκλήρωση του παιχνιδιού, διαμοιράστηκε το ερωτηματολόγιο σε δοκιμαστές για την λήψη της εποικοδομητικής κριτικής. Εντοπίστηκαν τεχνικά κομμάτια που δεν λειτουργούσαν, όποτε έγιναν οι απαραίτητες αλλαγές για την διόρθωσή τους σε επόμενο χρονικό διάστημα. Επιπρόσθετα έγιναν σχεδιαστικές αλλαγές με βάση τα σχόλια που προέκυψαν από το παρεχόμενο ερωτηματολόγιο.

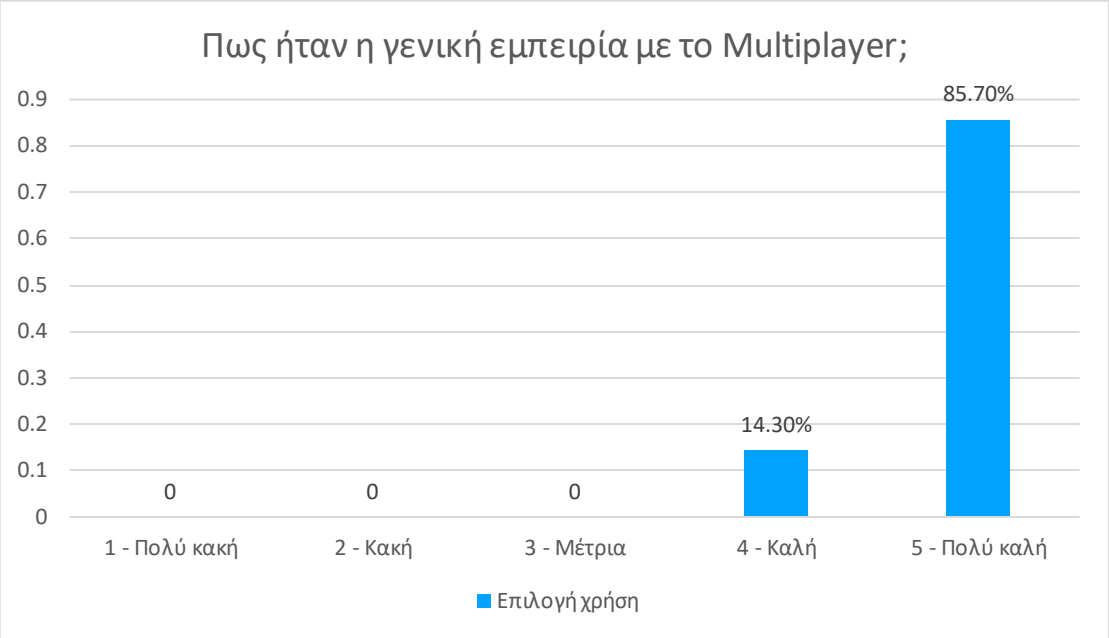
5.1 Αποτελέσματα δικτύωσης

Μεγάλος παράγοντας της καθυστέρησης είναι ο επιλεγμένος διακομιστής για την διασύνδεση, και η απόσταση του από τους παίκτες. Στην περίπτωση που οι παίκτες συνδέονταν στον διακομιστή που το παιχνίδι προεπέλεγε, δηλαδή τον πιο ιδανικό για τον παίκτη ως προς την καθυστέρηση, παρατηρήθηκαν καθυστερήσεις μεταξύ των τιμών 80 - 120ms κατά μέσο όρο (Σχήμα 1).



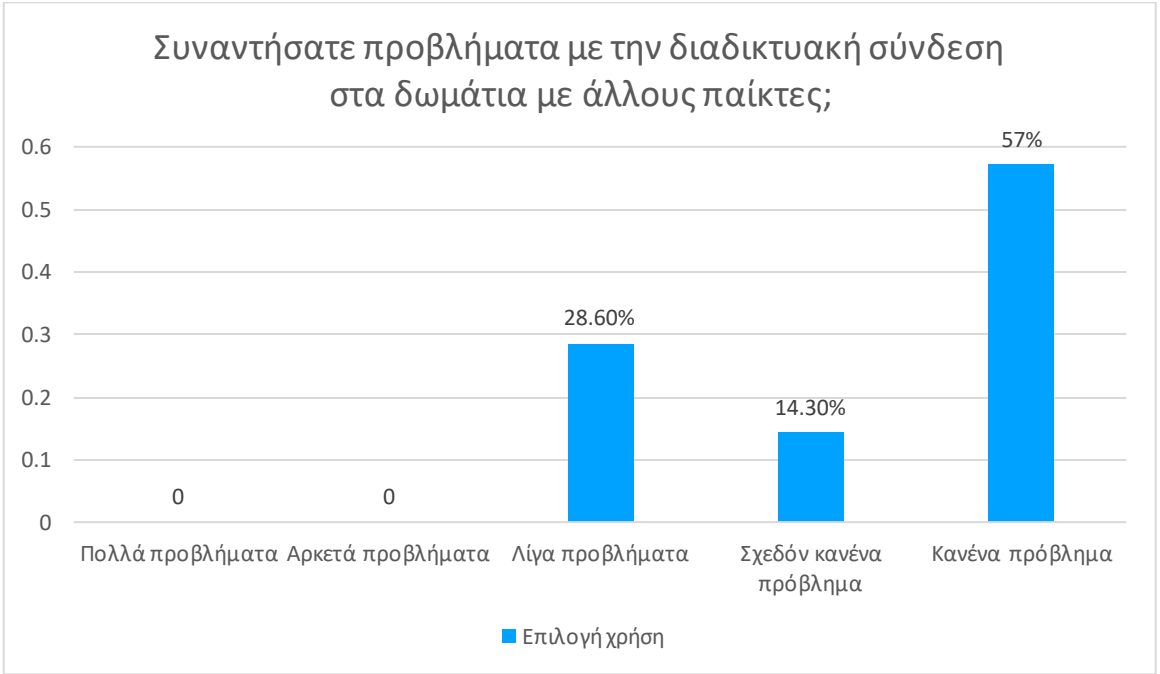
Σχήμα 1 Καθυστέρηση παικτών κατά μέσο όρο

Οι καθυστερήσεις αυτού του επιπέδου, όπως προαναφέραμε στην αρχή, είναι οι αποδεκτές καθυστερήσεις για το είδος του παιχνιδιού. Εφόσον το αποδεκτό όριο για ένα σκοπευτικό παιχνίδι τρίτου προσώπου είναι στα 500 ms, το παρών παιχνίδι παραμένει μακριά από τα όρια, παρέχοντας την ιδανική τιμή καθυστέρησης. Υποστηρίζεται το γεγονός ότι οι καθυστερήσεις της κλίμακας των 80 – 120ms δεν επηρέασαν αρνητικά τους παίκτες από το ερωτηματολόγιο που κλήθηκαν να συμπληρώσουν οι δοκιμαστές (Σχήμα 2).



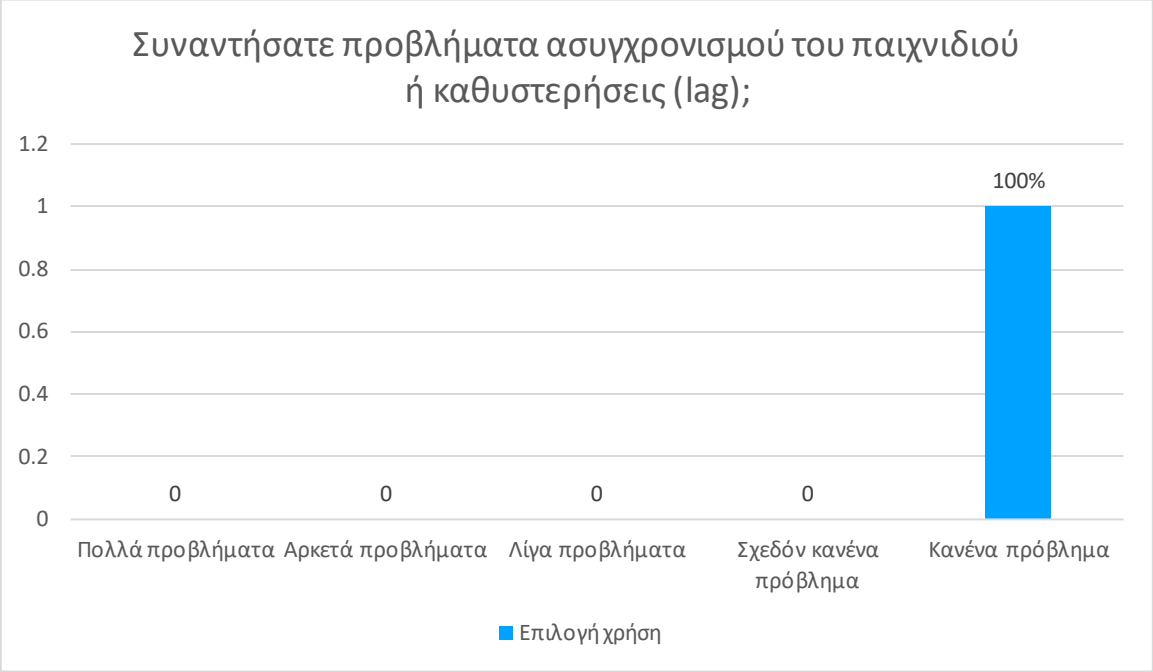
Σχήμα 2 Γενική εμπειρία παικτών με το Multiplayer

Η γενική εμπειρία του παιχνιδιού που δήλωσαν οι χρήστες ήταν «καλή» με την εμπειρία «πολύ καλή» να υπερισχύει (Σχήμα 2).



Σχήμα 3 Αναφορά προβλημάτων με την σύνδεση των δωματίων

Η διασύνδεση των χρηστών σε δωμάτια μπορεί να φέρει μια πρόκληση στην αρχή, γεγονός που συνέβη και στην προκειμένη περίπτωση (Σχήμα 3), μιας και οι χρήστες που είχαν συνδεθεί σε διαφορετικό διακομιστή δεν μπορούσαν να βρεθούν μεταξύ τους, και δεν είχαν την δυνατότητα αλλαγής του διακομιστή που ήταν συνδεδεμένοι. Όμως η λειτουργία αυτή προστέθηκε, και έγινε πλέον δυνατή η σύνδεση στον διακομιστή που επιλέγει ο παίκτης.



Σχήμα 4 Αναφορά προβλημάτων συγχρονισμού και καθυστέρησης κατά την διάρκεια του παιχνιδιού

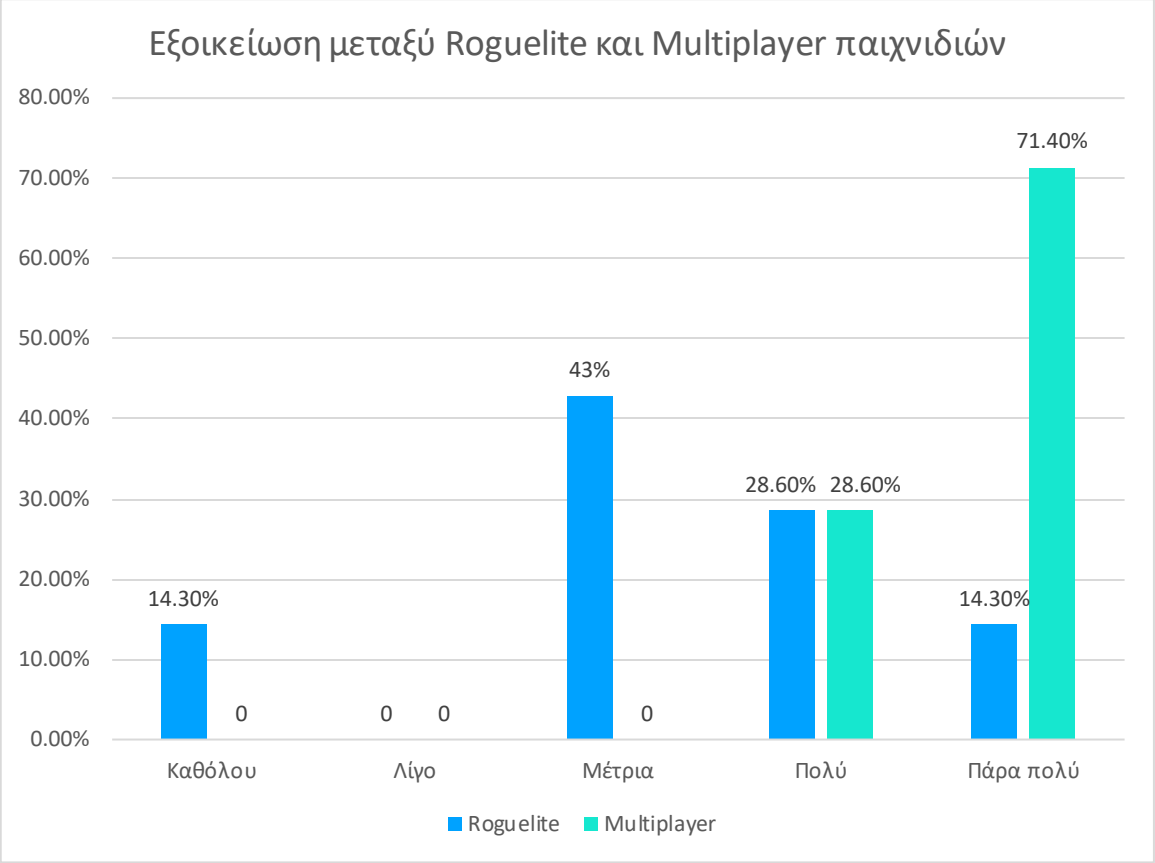
Κατά την διάρκεια του παιχνιδιού δεν έγινε αντιληπτή καμία καθυστέρηση και κανένας ασυγχρονισμός του παιχνιδιού από τους παίκτες εφόσον η συντριπτική πλειοψηφία ανέφερε πως δεν βίωσε κανένα πρόβλημα (Σχήμα 4). Παρόλο που η καθυστέρηση ή αλλιώς ping, ήταν στην κλίμακα των 80 – 120 ms, τιμή που σε άλλα πιο απαιτητικά, χρονικά ευαίσθητα παιχνίδια θα προκαλούσε αξιοπρόσεχτη διαφορά στις επιδόσεις του παίκτη, είναι ασήμαντος βαθμός καθυστέρησης στην κατηγορία που ανήκει το παιχνίδι της διπλωματικής εργασίας.

Εκτός από της στατιστικές ερωτήσεις που είχαν ως κύριο στόχο την εξαγωγή πληροφοριών από τους παίκτες πάνω στην εμπειρία τους με το multiplayer κομμάτι, δεν υπήρχε κάποιο αρνητικό σχόλιο περιγράφοντας γραπτώς πιθανά ελαττώματα στην διασύνδεση ή στην καθυστέρηση του παιχνιδιού.

5.2 Αποτελέσματα σχεδιασμού

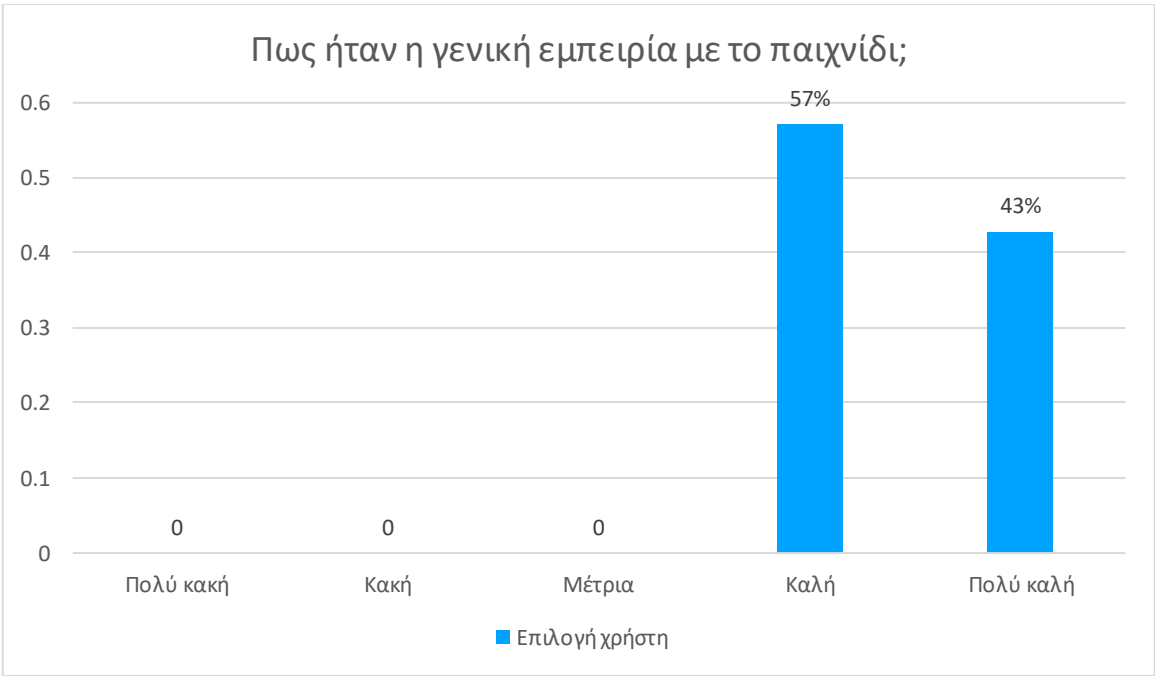
Ένα παιχνίδι πρέπει πάντα να λαμβάνει εποικοδομητική κριτική για να εξελίσσεται, και οι δοκιμές φανέρωσαν ποια σημεία εκτελέστηκαν σωστά, και ποια σημεία χρειάστηκαν βελτιώσεις ή μελλοντικές επεκτάσεις. Η μετρική για πολλούς από τους μηχανισμούς του παιχνιδιού είναι η αίσθηση της ικανοποίησης που ένιωθε ο παίκτης μέχρι το τέλος του παιχνιδιού.

Οι χρήστες έχουν μέτρια εξοικείωση με τα roguelite παιχνίδια κατά μέσο όρο, όμως είναι πλήρως εξοικειωμένοι με τα multiplayer παιχνίδια (Σχήμα 5), αποδεικνύοντας για άλλη μια φορά ότι όλο και περισσότερα άτομα παίζουν παιχνίδια με πολλαπλούς παίκτες.



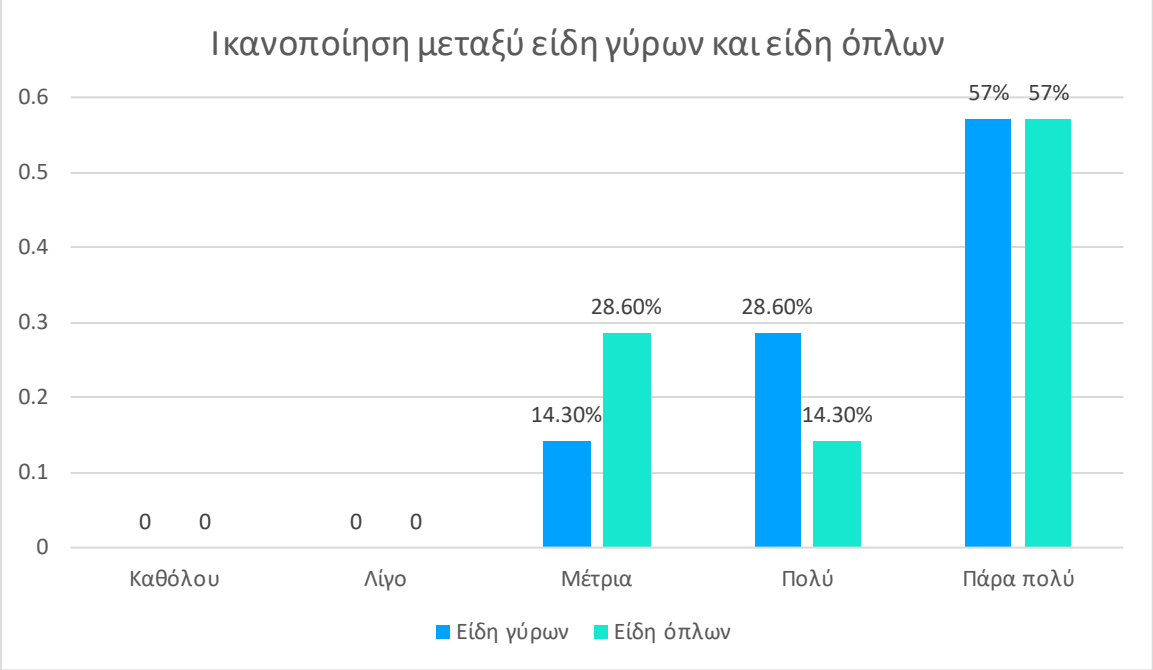
Σχήμα 5 Εξοικείωση μεταξύ Roguelite και Multiplayer παιχνιδιών

Οι παίκτες εξέφρασαν πως το παιχνίδι σαν σύνολο ήταν καλό έως και πολύ καλό (Σχήμα 6). Η πληροφορία αυτή δεν κατατοπίζει που βρίσκονται πιθανά προβλήματα, αλλά δείχνει την γενικότερη εικόνα των παικτών για το παιχνίδι.



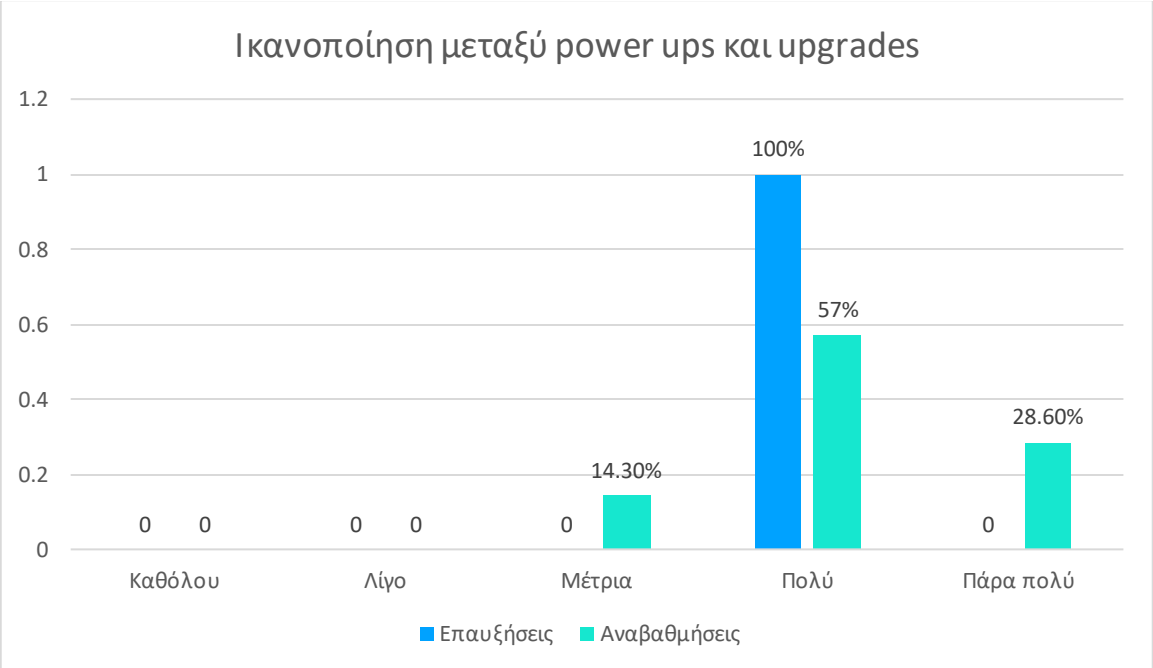
Σχήμα 6 Γενική εμπειρία χρήστη

Η ικανοποίηση της ποικιλίας στα είδη γύρων και στα είδη όπλων, έχει εύρος απαντήσεων, αρχίζοντας από μέτρια ικανοποίηση, με την πλειοψηφία να μένει πάρα πολύ ικανοποιημένη (Σχήμα 7).



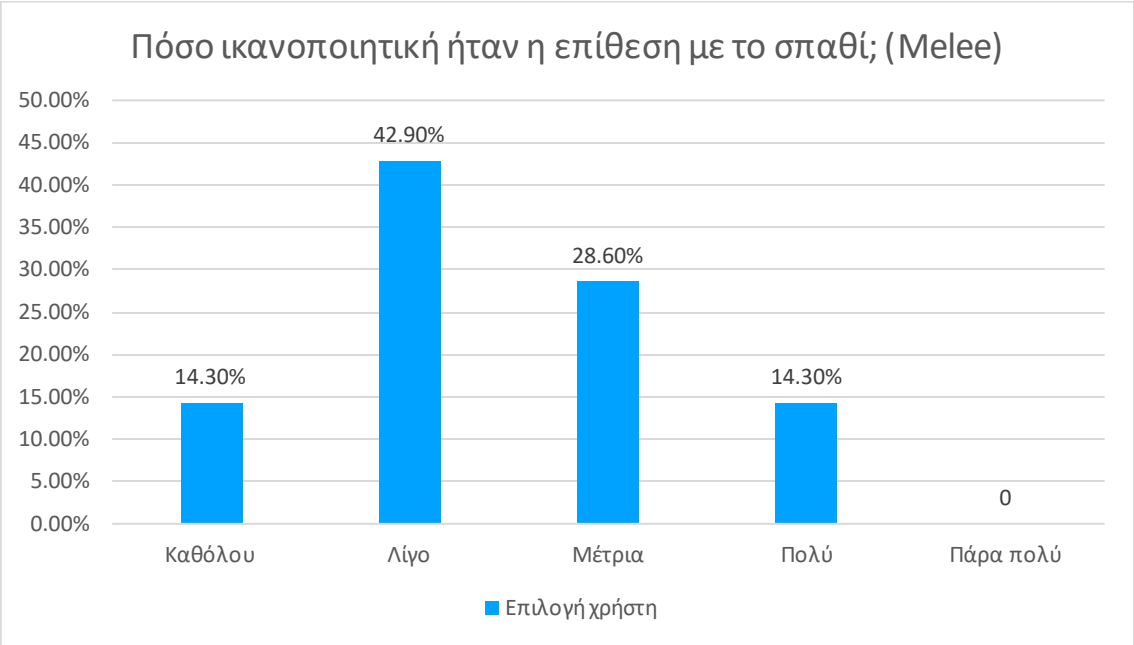
Σχήμα 7 Ικανοποίηση μεταξύ είδη γύρων και είδη όπλων

Στο θέμα της ικανοποίησης των power ups υπάρχει ομοφωνία στο επίπεδο ικανοποίησης «πολύ», ενώ στα upgrades κυμαίνονται ξανά οι απαντήσεις από μέτρια έως πάρα πολύ (Σχήμα 8). Οι κύριες κριτικές των συστημάτων αυτών είναι η μικρή ποικιλία που υπάρχει στα power ups, και πως τα upgrades θα μπορούσαν να έχουν μεγαλύτερη επίδραση στο παιχνίδι.



Σχήμα 8 Ικανοποίηση μεταξύ power ups και upgrades

Στην ικανοποίηση χρήσης σπαθιού (Σχήμα 9) εμφανίζονται σοβαρά προβλήματα καθώς οι απαντήσεις τείνουν στις χαμηλότερες βαθμολογίες, προσφέροντας κατά μέσο όρο λίγη ικανοποίηση. Το πρόβλημα που αναφέρθηκε ήταν πως το σύστημα για την χρήση του σπαθιού ήταν απλοϊκό και δεν ήταν χρήσιμο σε σχέση με τα όπλα.

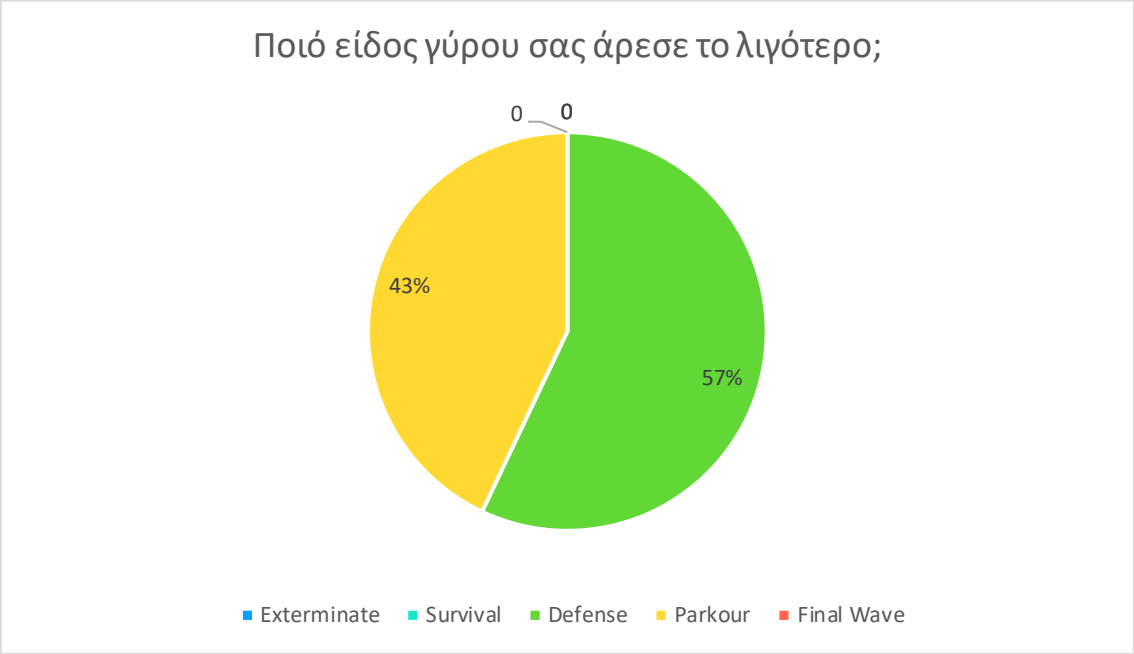


Σχήμα 9 Ικανοποίηση χρήσης σπαθιού

Ο κάθε παίκτης μπορεί να βρίσκει διαφορετικούς στόχους πιο διασκεδαστικούς, το οποίο αντικατοπτρίζεται και στα αποτελέσματα του ερωτηματολογίου (Σχήμα 10). Με την εξαίρεση του γύρου Parkour, κάθε άλλως γύρος ήταν αρεστός από τους παίκτες με τον γύρο Survival να ξεχωρίζει.

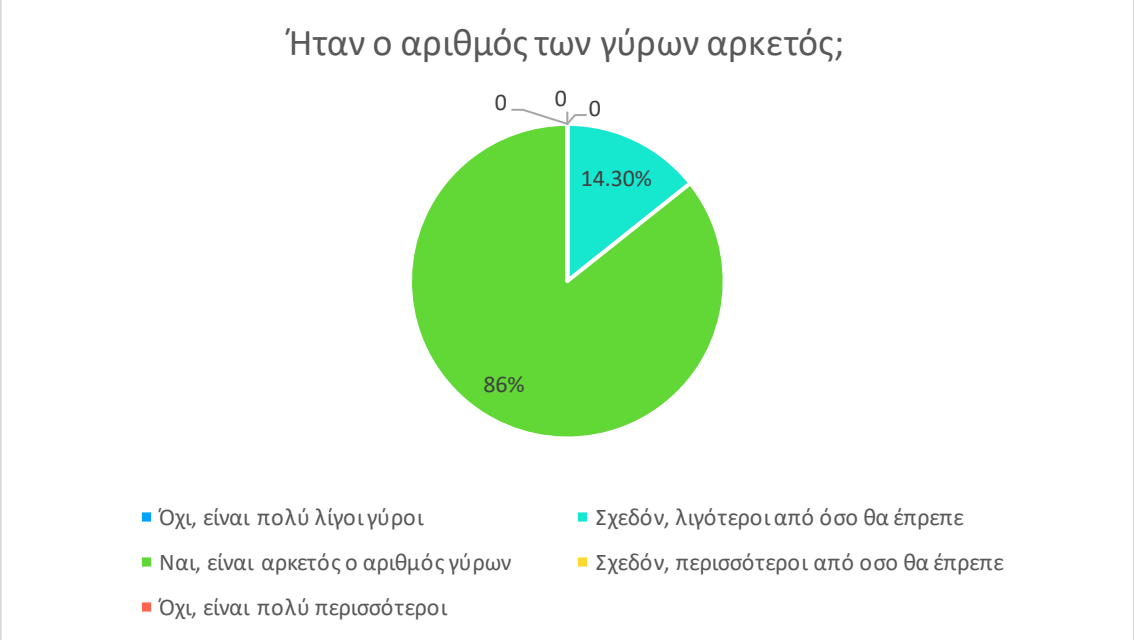
Σχήμα 10 Μεγαλύτερη αρέσκεια μεταξύ των γύρων

Από την άλλη, ήταν ξεκάθαρο ποια είδη γύρων δεν πέτυχαν στην ικανοποίηση των παικτών, όπως το Parkour και το Defense (Σχήμα 11). Στην περίπτωση του γύρου Parkour παρατηρήθηκε μεγάλη δυσκολία από τους παίκτες στην αντίδραση και αποφυγή των εμποδίων, με αποτέλεσμα να χάνουν συχνά. Το Defense είχε το αντίθετο πρόβλημα όπου η δυσκολία ήταν χαμηλότερη σε σχέση με τα υπόλοιπα είδη γύρων και ήταν πιο άνετο επειδή δεν συγκεντρώνονται όλοι οι εχθροί γύρω από τους παίκτες.



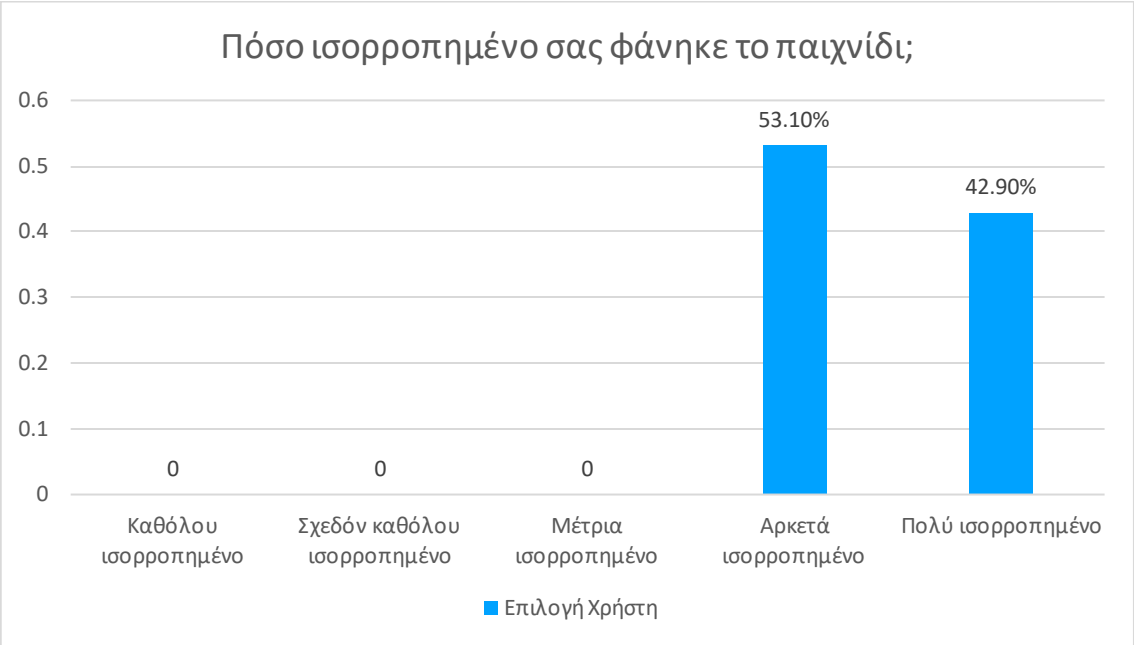
Σχήμα 11 Μικρότερη αρέσκεια μεταξύ των γύρων

Στην ερώτηση για τον μέγιστο αριθμό των γύρων, η πλειοψηφία βρήκε τον αριθμό των γύρων αρκετό (Σχήμα 12).



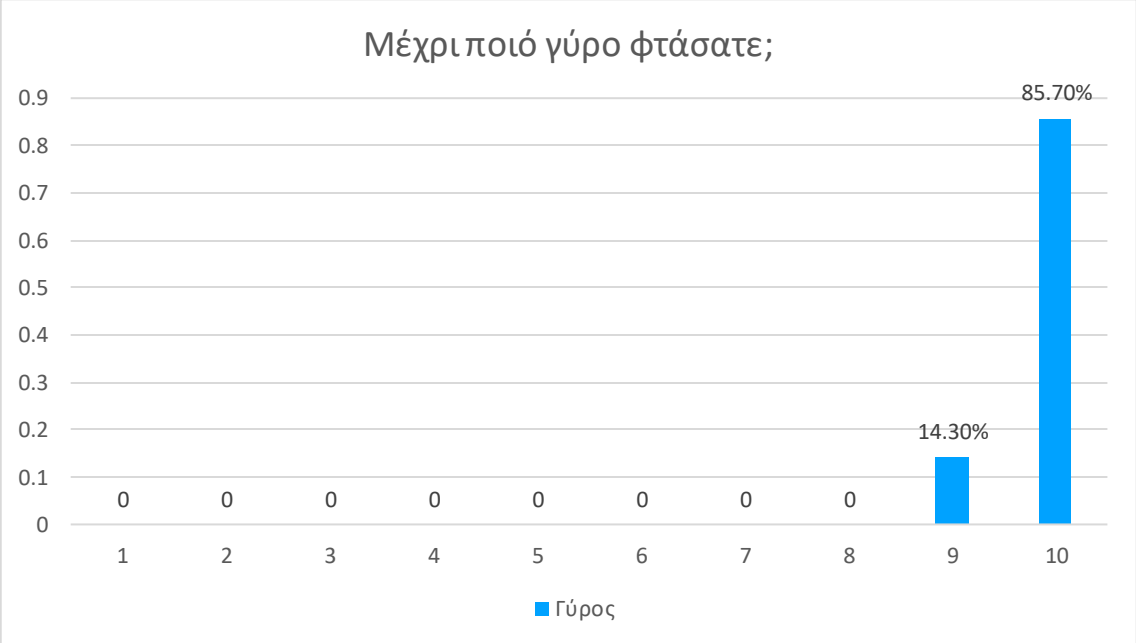
Σχήμα 12 Αριθμός γύρων

Όσον αφορά την γενική ισορροπία, οι παίκτες έμειναν επαρκώς ικανοποιημένοι (Σχήμα 13) σημειώνοντας πως το παιχνίδι είναι αρκετά ισορροπημένο. Είναι σημαντική η μετρική αυτή, επειδή είναι πιο πιθανό ένας παίκτης να αρχίσει το παιχνίδι ξανά όταν νιώθει πως έρχεται αντιμέτωπος με δυσκολία που αρμόζει για τις ικανότητές του [30].



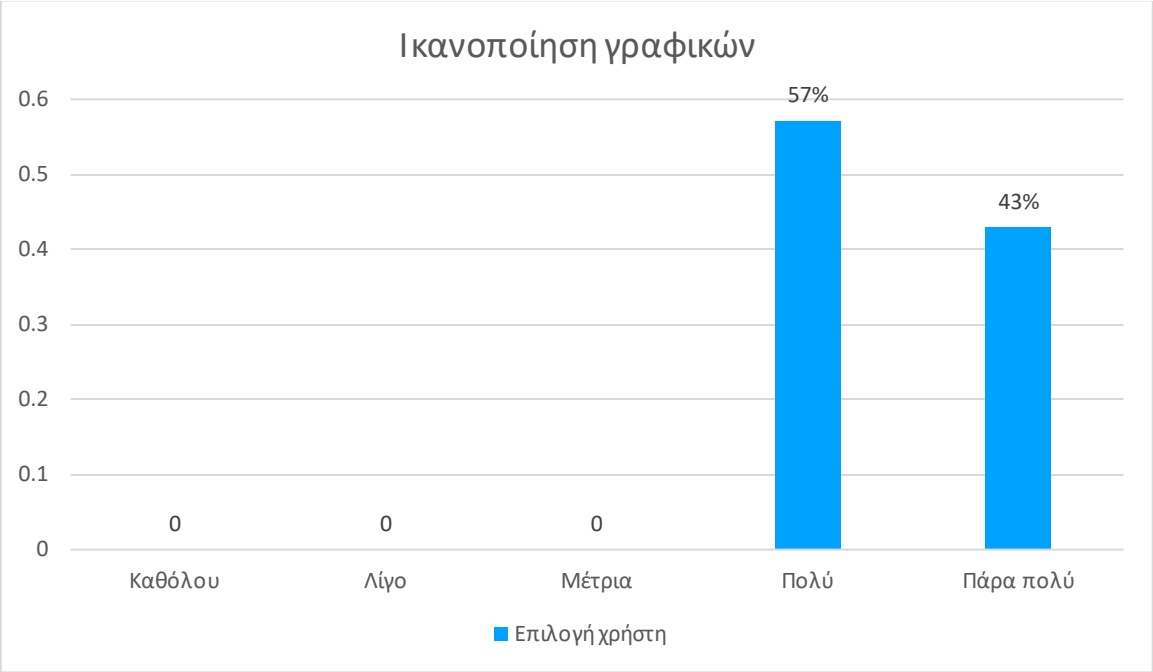
Σχήμα 13 Ισορροπία παιχνιδιού

Ο μέγιστος γύρος που κατάφεραν οι παίκτες να κατακτήσουν ήταν κατά κύριο λόγο ο γύρος 10, δηλαδή τερμάτιζαν το παιχνίδι. Ένα μικρό ποσοστό επίσης έφτασε έως τον γύρο 9, δηλαδή πολύ κοντά στην νίκη (Σχήμα 14). Αυτή η πληροφορία μας ενημερώνει πως οι παίκτες θα έπαιζαν το παιχνίδι μέχρι να το κερδίσουν ή τουλάχιστον θα έφταναν πολύ κοντά.



Σχήμα 14 Μέγιστος αριθμός γύρων που επιτεύχθηκε από τους παίκτες

Τέλος, στο κομμάτι της γραφικής ποιότητας του παιχνιδιού, οι παίκτες δεν είχαν παράπονα (Σχήμα 15), και μάλιστα τα γραφικά πέτυχαν την αισθητική που σκόπευαν να προκαλέσουν.



Σχήμα 15 Ικανοποίηση από τα γραφικά

Με βάση όλες τις απαντήσεις που προαναφέρθηκαν στο ερωτηματολόγιο που δόθηκε στους δοκιμαστές του παιχνιδιού μπορούμε ένα συμπεράνουμε πως η εμπειρία ήταν ευχάριστη με ελάχιστα προβλήματα στο σχεδιαστικό κομμάτι. Από την μεριά της διασύνδεσης οι παίκτες είχαν λίγα έως καθόλου προβλήματα στην εύρεση και εισαγωγή των δωματίων, η καθυστέρηση ήταν εντός αποδεκτών ορίων και ταυτόχρονα δεν γινόταν αισθητή, με αποτέλεσμα να μην μειώνονται οι επιδόσεις λόγω των διαδικτυακών επιβραδύνσεων.

Στο επόμενο και τελευταίο κεφάλαιο γίνεται η ανακεφαλαίωση της διπλωματικής εργασίας με τα συμπεράσματα, δίνεται αναφορά των μετρικών του κώδικα, και τέλος αναγράφονται οι μελλοντικές επεκτάσεις.

Κεφάλαιο 6: Επίλογος

Τα παιχνίδια προσφέρουν πολλά πλεονεκτήματα στον άνθρωπο, αλλά είναι επίσης μέσο ψυχαγωγίας και αλληλεπίδρασης με άλλους ανθρώπους. Σε αυτήν τη διπλωματική εργασία πραγματοποιήθηκε η ανάπτυξη ενός 3D σκοπευτικού παιχνιδιού τρίτου προσώπου πολλαπλών παικτών τύπου Roguelite με ονομασία Bullet Drifters με κύριο σκοπό την εκπλήρωση των βασικών στόχων που πρέπει να ικανοποιεί ένα παιχνίδι και παράλληλα την επεξήγηση της διαδικτύωσης από την μεριά του κώδικα. Η διαδικασία για την ανάπτυξη του παιχνιδιού είχε φέρει πολλαπλά προβλήματα και προκλήσεις σε πολλούς τομείς τόσο στο τεχνικό κομμάτι όσο και στο δημιουργικό. Έπρεπε να γίνει ανάμειξη προγραμματιστικών γνώσεων για την ομαλή λειτουργία του παιχνιδιού χωρίς σφάλματα και καθυστερήσεις από την διαδικτύωση, με τις σχεδιαστικές γνώσεις παιχνιδιού για να υπάρχουν τα σωστά θεμέλια ώστε ο παίκτης να διασκεδάσει με τον χρόνο που επενδύει στο παιχνίδι.

6.1 Συμπεράσματα

Στο τεχνικό κομμάτι του παιχνιδιού το μέγεθος βελτιστοποιήθηκε ώστε να μην δεσμεύει πολύ αποθηκευτικό χώρο, και πολλά αντικείμενα δημιουργούνται κατά την εισαγωγή στην αρένα για την βελτίωση των επιδόσεων. Επιπρόσθετα έγινε η κατάλληλη χρήση του φωτισμού ώστε να μην επιβαρύνεται περαιτέρω το σύστημα. Στην διαδικτύωση του κώδικα έγιναν οι απαραίτητοι έλεγχοι ώστε να μην στέλνονται μηνύματα που δεν είναι απαραίτητα για την λειτουργία του παιχνιδιού ή τον συγχρονισμό των μεταβλητών, με αποτέλεσμα την ιδανική καθυστέρηση που δεν μειώνει τις επιδόσεις των παικτών.

Το σχεδιαστικό κομμάτι, όπως και με κάθε έργο όπου συνδυάζονται στοιχεία τέχνης, είναι πάντα ανοιχτό για εποικοδομητική κριτική. Η ισορροπία του παιχνιδιού προσαρμόστηκε μετά από πολύωρο δοκιμαστικό έλεγχο με αποτέλεσμα οι παίκτες να έρχονται αντιμέτωποι με προκλήσεις ανάλογες των ικανοτήτων τους. Η γενική εμπειρία των παικτών ήταν ευχάριστη, όμως παραμένουν στοιχεία που ήταν προβληματικά όπως η δυσκολία του γύρου Parkour και η χρήση του σπαθιού, η οποία δεν είχε το σωστό αντίκτυπο σε σχέση με κάθε άλλο όπλο.

6.2 Μετρικά κώδικα

Το συνολικό μέγεθος του παιχνιδιού ανέρχεται στα 336MB. Στο μέγεθος αυτό συμπεριλαμβάνονται όλα τα αρχεία που χρειάστηκαν για την δημιουργία, όπως τα μοντέλα, τα textures, τους ήχους, τους κώδικες, την ίδια την μηχανή της Unity, κ.α.

Τα αρχεία του κώδικα φτάνουν στα 298KB και στον παρακάτω πίνακα (Πίνακας 2) αναλύεται η κατανομή των γραμμών του κώδικα.

Πίνακας 2 Μετρικά κώδικα παιχνιδιού

Τύπος αρχείου	Αρχεία	Γραμμές κώδικα	Κενές γραμμές	Σχόλια
C#	73	6,550	1,058	120

Κατά την ανάπτυξη του παιχνιδιού, κατασκευάστηκαν 68 υλικά (materials) και 21 textures (υφή υλικών). Συνολικά τα αρχεία ήχου που συμπεριλαμβάνονται στο παιχνίδι είναι 40.

6.3 Μελλοντικές επεκτάσεις

Το παιχνίδι που αναπτύχθηκε κατά την διάρκεια της διπλωματικής εργασίας είχε περισσότερους στόχους και πιο μεγάλη κλίμακα σε μηχανισμούς, που λόγω χρονικών περιορισμών δεν ήταν δυνατό να υλοποιηθούν. Ένα χαρακτηριστικό των Roguelite παιχνιδιών που λείπει είναι η δημιουργία διαδικαστικού χάρτη, δηλαδή η τυχαία παραγωγή του περιβάλλοντος. Αυτός ο μηχανισμός ενθαρρύνει τον παίκτη στην επαναληπτική εκκίνηση του παιχνιδιού και αυξάνει την αίσθηση της εξερεύνησης. Ύστερα από αρκετές δοκιμές του παιχνιδιού από τους παίκτες γίνεται φανερή η έλλειψη επιπλέον περιεχομένου, το οποίο θα μπορούσε να επιτευχθεί με περισσότερο χρόνο. Συγκεκριμένα:

- Το εύρος των power ups και των upgrades θα μπορούσε να είναι μεγαλύτερο, αλλά αντί να αυξάνουν τα στατιστικά του παίκτη, θα μπορούσαν να έχουν πιο ενεργές δυνατότητες, ώστε να είναι πιο αισθητή η ενδυνάμωση του παίκτη, αλλά και να δημιουργούν μια καλύτερη εμπειρία.

- Τα όπλα επίσης θα μπορούσαν να διευρυνθούν σε περισσότερες κατηγορίες με πιο καινοτόμους μηχανισμούς για να προκαλεί την θέληση του παίκτη να εξοπλιστεί με καινούργια όπλα.

- Οι εχθροί χρειάζονται μεγαλύτερη ποικιλία, τόσο στα μοντέλα όσο και στις μορφές επίθεσης ώστε ο παίκτης να σκέφτεται πως να αντιμετωπίσει τον γύρο ανάλογα με τους εχθρούς που έρχεται αντιμέτωπος.

Από την διαδικτυακή πλευρά του παιχνιδιού, θα μπορούσε να βελτιστοποιηθεί σε μεγαλύτερο βαθμό, για την επιτυχία μικρότερης καθυστέρησης των παικτών. Η περαιτέρω βελτίωση δίνει βάση για την δημιουργία ενός παιχνιδιού πρώτου προσώπου ή και ενός ανταγωνιστικού παιχνιδιού αποφεύγοντας την μείωση της επίδοσης του παίκτη λόγω της καθυστέρησης στο δίκτυο. Τέλος, θα ήταν υποχρεωτική η ενσωμάτωση ασφάλειας για την καταπολέμηση κακόβουλων χρηστών, ειδικά σε περίπτωση που το παιχνίδι θα ήταν ανταγωνιστικό.

Παραρτήματα

Οδηγίες εγκατάστασης παιχνιδιού

Το παιχνίδι δεν προαπαιτεί κανένα λογισμικό, οπότε μπορεί να εκτελεστεί με το αρχείο Bullet Drifters.exe.

Βιβλιογραφία

[1] A. P. a. F. M. Federica Pallavicini, «The Effects of Playing Video Games on Stress, Anxiety, Depression, Loneliness, and Gaming Disorder During the Early Stages of the COVID-19 Pandemic: PRISMA Systematic Review,» *Cyberpsychology, Behavior, and Social Networking,* τόμ. 25, αρ. 6, pp. 329-405, 2022.

[2] A. T. O. R. M. M. Yemaya J. Halbrook, «When and How Video Games Can Be Good: A Review of the Positive Effects of Video Games on Well-Being,» *Sage Journals,* τόμ. 14, αρ. 6, 1 November 2019.

[3] G. Divers, «Gaming Industry Dominates as the Highest-Grossing Entertainment Industry,» Gamerhub, 24 01 2023. [Ηλεκτρονικό]. Available: https://gamerhub.co.uk/gaming-industry-dominates-as-the-highest-grossing-entertainment-industry/.

[4] M. Saltzman, «More adults play video games than kids – and more surprising stats,» USA TODAY TECH, 2022. [Ηλεκτρονικό]. Available: https://eu.usatoday.com/story/tech/gaming/2022/06/11/gaming-study-finds-adults-play-more-than-kids/7581101001/.

[5] K. Balasubramanian, «The Rise of Online Multiplayer,» GAMEOPEDIA, 29 August 2022. [Ηλεκτρονικό]. Available: https://www.gameopedia.com/online-multiplayer-games/.

[6] No Surrender Heroes, «Why Do People Enjoy Multiplayer Games So Much?,» Medium, 19 December 2022. [Ηλεκτρονικό]. Available: https://medium.com/blockchain-biz/why-do-people-enjoy-multiplayer-games-so-much-60930d954235.

[7] A. King, «Risk of Rain 2 Review,» GameSpot, 28 March 2019. [Ηλεκτρονικό]. Available: https://www.gamespot.com/reviews/risk-of-rain-2-review/1900-6417527/. [Πρόσβαση 9 October 2023].

[8] «Gunfire Reborn, the popular roguelite FPS, has officially launched on Android and iOS,» PocketGamer, 18 May 2022. [Ηλεκτρονικό]. Available: https://www.pocketgamer.com/gunfire-reborn-mobile/roguelite-fps-official-launch-android-ios/. [Πρόσβαση 9 October 2023].

[9] M. C. a. K. Claypool, «LATENCY AND PLAYER ACTIONS IN ONLINE GAMES,» *Communications of the ACM,* 2006.

[10] K. L.-O. G. A.-M. J. d. C. N. V.-L. Rafael Valencia-García, «HTML5-Based Frameworks for Developing Education and Serious Games,» σε *Technologies and Innovation: Second International Conference*, Guayaquil, Springer, 2016, pp. 146-147.

[11] M. K. Tim Bradshaw, «Epic and Unity rev their engines for the next era of entertainment,» FINANCIAL TIMES, 12 August 2020. [Ηλεκτρονικό]. Available: https://www.ft.com/content/f77b7979-c943-4b9d-b7b7-7953b63bea7e. [Πρόσβαση 9 October 2023].

[12] Arnia Software, «What Makes Unity So Popular in Game Development?,» Arnia Software, 14 September 2022. [Ηλεκτρονικό]. Available: https://www.arnia.com/what-makes-unity-so-popular-in-game-development/. [Πρόσβαση 9 October 2023].

[13] Ocean, «What is the Unity Asset Store and how do I purchase Assets?,» Unity, May 2023. [Ηλεκτρονικό]. Available: https://support.unity.com/hc/en-us/articles/210142503-What-is-

the-Unity-Asset-Store-and-how-do-I-purchase-Assets-. [Πρόσβαση 9 October 2023].

[14] InfoQ, C# Language Specification, InfoQ, 2016.

[15] S. J. Zeil, «Integrated Development Environments,» Old Dominion Univ., 14 September 2017. [Ηλεκτρονικό]. Available: https://www.cs.odu.edu/~zeil/cs350/f17/Public/IDEs/index.html. [Πρόσβαση 9 October 2023].

[16] F. Lardinois, «Microsoft Launches Visual Studio Code, A Free Cross-Platform Code Editor For OS X, Linux And Windows,» TechCrunch+, 29 April 2015. [Ηλεκτρονικό]. Available: https://techcrunch.com/2015/04/29/microsoft-shocks-the-world-with-visual-studio-code-a-free-code-editor-for-os-x-linux-and-windows/. [Πρόσβαση 9 October 2023].

[17] TechCrunch+, «GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz,» TechCrunch+, 10 July 2012. [Ηλεκτρονικό]. Available: https://techcrunch.com/2012/07/09/github-pours-energies-into-enterprise-raises-100-million-from-power-vc-andreesen-horowitz/. [Πρόσβαση 9 October 2023].

[18] J. Mussi, «What is Blender, and Is It Right for You?,» PCMag, 17 October 2022. [Ηλεκτρονικό]. Available: https://www.pcmag.com/reviews/blender. [Πρόσβαση 9 October 2023].

[19] Photon, «Features Overview,» Photon Engine, 2023. [Ηλεκτρονικό]. Available: https://doc.photonengine.com/pun/current/getting-started/feature-overview. [Πρόσβαση 9 October 2023].

[20] J. Payne, «Diagrams.net - An essential tool for techies,» The Serpent, 23 January 2021. [Ηλεκτρονικό]. Available: https://www.theserpent.co.uk/posts/diagrams.net-an-essential-tool-for-techies/. [Πρόσβαση 9 October 2023].

[21] Sketchfab, «ABOUT SKETCHFAB,» Sketchfab, 2023. [Ηλεκτρονικό]. Available: https://sketchfab.com/about. [Πρόσβαση 9 October 2023].

[22] B. Stockton, «A Beginner's Guide To Paint.NET & How Does It Differ From Photoshop?,» Online Tech Tips, 10 June 2020. [Ηλεκτρονικό]. Available: https://www.online-tech-tips.com/software-reviews/a-beginners-guide-to-paint-net-how-does-it-differ-from-photoshop/. [Πρόσβαση 9 October 2023].

[23] ZapSplat, «About ZapSplat,» ZapSplat, 2023. [Ηλεκτρονικό]. Available: https://www.zapsplat.com/about/. [Πρόσβαση 9 October 2023].

[24] Audacity, «About,» Audacity, 2023. [Ηλεκτρονικό]. Available: https://www.audacityteam.org/about/. [Πρόσβαση 9 October 2023].

[25] T. X. Short, «Never Say Roguelike,» Game Developer, 19 November 2013. [Ηλεκτρονικό]. Available: https://www.gamedeveloper.com/business/never-say-roguelike. [Πρόσβαση 9 October 2023].

[26] B. Stegner, «What Are Roguelike and Roguelite Video Games?,» Make Use Of, 24 October 2021. [Ηλεκτρονικό]. Available: https://www.makeuseof.com/what-are-roguelike-and-roguelite-video-games/. [Πρόσβαση 9 October 2023].

[27] E. V. Allen, «Roguelike vs. Roguelite: What's the difference between the two?,» Destructoid, 4 November 2022. [Ηλεκτρονικό]. Available: https://www.destructoid.com/the-difference-between-roguelike-and-roguelite-games/. [Πρόσβαση 9 October 2023].

[28] Επιτροπή Ηθικής και Δεοντολογίας της Έρευνας (Ε.Η.Δ.Ε.), Πανεπιστήμιο Δυτικής Μακεδονίας, [Ηλεκτρονικό]. Available: https://ehde.uowm.gr/.

[29] J. Howarth, «How Many Gamers Are There? (New 2023 Statistics),» Exploding Topics, 10

August 2023. [Ηλεκτρονικό]. Available: https://explodingtopics.com/blog/number-of-gamers. [Πρόσβαση 9 October 2023].

[30] G. L. S. N. Maria-Virginia Aponte, «Measuring the level of difficulty in single player video games,» *Entertainment Computing,* 9 April 2011.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

κ.ά.	και άλλα

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
προσωρινή αναβάθμιση	power up
μόνιμη αναβάθμιση	upgrade
αρχείο κώδικα	script
εξάρτημα	component
δημιουργός παιχνιδιού	host