ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΠΛΗΡΟΦΟΡΙΚΉ

Σχεδιασμός και Υλοποίηση MMORPG διαδικτυακού παιχνιδιού πολλαπλών παικτών με κεντρικό διακομιστή και client

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Παύλος Μολλάς

Επιβλέπων καθηγητής : Δρ Μηνάς Δασυγένης

Μέλη επιτροπής : Δρ Νικόλαος Πλόσκας

Δρ Χαϊρή Κιούρτ

ΠΑΤΡΑ ΟΚΤΩΒΡΙΟΣ 2025

HELLENIC OPEN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND INFOMATICS

Design and Implementation of an MMORPG Online Multiplayer Game Using a Central Server and Client Architecture.

THESIS

Pavlos Mollas

SUPERVISOR: Dr Minas Dasygenis

Committee members: Dr Nikolas Ploskas

Dr Chairee Kiourt

PATRAS OCTOBER 2025

Περίληψη

Ο στόχος της πτυχιακής εργασίας είναι η σχεδίαση και η υλοποίηση ενός multiplayer παιχνιδιού, όπου ο παίκτης έχει τους χειρισμούς για την κίνηση και την επίθεση του χαρακτήρα. Κάθε παίκτης επιλέγει μία από τις διαθέσιμες κλάσεις χαρακτήρων, Πολεμιστή, Μάγο ή Σκοπευτή, οι οποίες διαθέτουν μοναδικές ικανότητες. Οι παίκτες μπορούν να συλλέγουν χρυσό μέσω μαχών, να αγοράζουν αντικείμενα, καθώς και να βελτιώνουν τα στατιστικά του χαρακτήρα τους μέσω ενός συστήματος επιπέδων. Οι εχθροί αποδίδουν ανταμοιβές, όπως χρυσό, με τυχαίο τρόπο, ενισχύοντας την επαναληψιμότητα και τη συνεργατική διάσταση του παιχνιδιού.

Σκοπός του παιχνιδιού είναι η κατάκτηση όλων των περιοχών του χάρτη μέσω της συνεργασίας των παικτών, το οποίο προϋποθέτει τη νίκη επί των τελικών εχθρών στην εκάστοτε περιοχή και την αποφυγη αποτυχίας. Οι παίκτες αντιμετωπίζουν διαδοχικά κύματα εχθρών αυξανόμενης δυσκολίας σε κάθε περιοχή, συλλέγοντας χρυσό, αντικείμενα και εμπειρία για να ενισχύσουν τους χαρακτήρες τους. Το παιχνίδι μπορεί να παιχτεί από έναν ή και περισσότερους παίκτες μέσω του δικτύου, δηλαδή θα είναι multiplayer. Ο κύριος τεχνικός στόχος είναι η μελέτη και ανάπτυξη της δικτύωσης πολλαπλών παικτών στον ίδιο διακομιστή, με σταθερή επικοινωνία μεταξύ πελατών και server. Παράλληλα, η πτυχιακή εργασία στοχεύει στην αναλυτική τεκμηρίωση της αρχιτεκτονικής και της υλοποίησης του συστήματος, από το δίκτυο και τη διαχείριση των δεδομένων έως τους μηχανισμούς συνεργασίας και προόδου μέσα στο παιχνίδι.

Το αποτέλεσμα που επιδιώκεται με την παρούσα πτυχιακή εργασία είναι η ανάδειξη των οφελών και δυνατοτήτων που μπορεί να προσφέρει ένα ηλεκτρονικό παιχνίδι συνεργατικού τύπου στους χρήστες. Το παιχνίδι συνδυάζει την αλληλεπίδραση μεταξύ παικτών με την ψυχαγωγία και την ομαδικότητα, ενισχύοντας ταυτόχρονα δεξιότητες όπως ο συντονισμός, η επικοινωνία και η στρατηγική συνεργασία.Επιπλέον, μέσα από την ανάπτυξη του παιχνιδιού, παρουσιάζεται αναλυτικά η διαδικασία διασύνδεσης πολλαπλών χρηστών μέσω ενός κοινού διακομιστή. Η πτυχιακή εργασία συμβάλλει στην κατανόηση της δικτύωσης σε περιβάλλοντα παιχνιδιών, παρέχοντας τεχνικές πληροφορίες και παραδείγματα για τον τρόπο με τον οποίο μπορεί να επιτευχθεί σταθερή, αποδοτική και συνεχής επικοινωνία μεταξύ πελατών και server σε ένα σύγχρονο multiplayer περιβάλλον.

Λέξεις Κλειδιά: Multiplayer, διακομιστής, παιχνίδι, MMORPG, διασύνδεση, ψυχαγωγία, αλληλεπίδραση, στρατηγική.

Abstract

The aim of this thesis is the design and implementation of a multiplayer game, in which the player has controls for character movement and attacks. Each player chooses one of the available character classes, Warrior, Mage, or Marksman, each of which possesses unique abilities. Players can collect gold through battles and enhance their character's attributes through a leveling system. Enemies provide rewards, such as gold, in a random manner, reinforcing the game's replayability and cooperative dimension.

The main objective of the game is the conquest of all areas on the map through player cooperation, which requires defeating the area's final enemies while avoiding failure. Players face successive waves of enemies with increasing difficulty in each area, collecting gold, items, and experience to strengthen their characters. The game can be played by a single player or multiple players over a network, making it a multiplayer experience. The primary technical goal is the study and development of multi-player networking on a single server, ensuring stable communication between clients and server. Moreover, the thesis aims at a detailed documentation of the system's architecture and implementation, covering the network layer, data management, and the mechanisms for cooperation and player progression within the game.

The expected outcome of this thesis is to highlight the benefits and potential offered by a cooperative electronic game to its users. The game combines player interaction, entertainment, and teamwork, while also enhancing skills such as coordination, communication, and strategic collaboration. Additionally, through the development process, the thesis presents in detail the connection of multiple users via a common server. This work contributes to understanding networking in gaming environments, providing technical information and examples of how stable, efficient, and continuous communication between clients and server can be achieved in a modern multiplayer setting.

Keywords: Multiplayer, server, game, MMORPG, networking, entertainment, interaction, strategy.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1: Εισαγωγή

Στο κεφάλαιο αυτό γίνεται μια επισκόπηση του κόσμου των ψηφιακών παιχνιδιών, αναδεικνύοντας τον ρόλο τους στην ψυχαγωγία, την εκπαίδευση και την κοινωνική αλληλεπίδραση. Παρουσιάζονται οι προκλήσεις και οι απαιτήσεις που συνεπάγεται η ανάπτυξη ψηφιακών παιχνιδιών, με έμφαση σε εκείνα που υποστηρίζουν δικτυακή σύνδεση πολλών παικτών ταυτόχρονα (multiplayer) και συγκεκριμένα τα παιχνίδια πολλαπλών παικτών όπου ο παίκτης αναλαμβάνει τον ρόλο ενός ή περισσότερων χαρακτήρων (RPG). Επιπλέον, περιγράφεται ο σκοπός της πτυχιακής εργασίας, ενώ γίνεται αναφορά σε παρόμοια έργα που αποτέλεσαν έμπνευση και δείχνουν τις δυνατότητες συνεργασίας σε περιβάλλον παιχνιδιού πολλαπλών παικτών.

1.1 Ψηφιακά παιχνίδια

Τα ψηφιακά παιχνίδια αποτελούν έναν από τους πλέον δημοφιλείς τρόπους ψυχαγωγίας και αλληλεπίδρασης στον σύγχρονο ψηφιακό κόσμο. Η συνεχής ανάπτυξή τους ανταποκρίνεται στη διαρκώς αυξανόμενη ζήτηση του κοινού, το οποίο ενδιαφέρεται για νέα και πιο σύνθετα είδη παιχνιδιών [1]. Τα παιχνίδια είναι ιδιαίτερα χρήσιμα καθώς συνδυάζουν ψυχαγωγία, στρατηγική, δημιουργικότητα και κοινωνική αλληλεπίδραση, προσφέροντας στους χρήστες δυνατότητες που ξεπερνούν την απλή διασκέδαση [2]. Ωστόσο η σωστή χρήση των παιχνιδιών είναι απαραίτητη, καθώς με υπερβολική και άσκοπη χρήση αποφέρουν αρνητικές επιπτώσεις [3][4].

Τα δημοφιλέστερα παιχνίδια συχνά είναι αυτά που επιτρέπουν πολλαπλή διασύνδεση των παικτών (multiplayer) [5], καθώς οι παίκτες αλληλεπιδρούν σε πραγματικό χρόνο και ενισχύουν την εμπειρία του παιχνιδιού. Η αυξημένη ζήτηση για τέτοια παιχνίδια ανοίγει νέους ορίζοντες για επαγγελματική ενασχόληση στον χώρο της ανάπτυξης ψηφιακών παιχνιδιών, προσφέροντας ευκαιρίες τόσο για δημιουργική έκφραση όσο και για την ανάπτυξη τεχνικών και επιχειρηματικών δεξιοτήτων [6].

1.2 Παιχνίδια Πολλαπλών Παικτών (Multiplayer)

Τα παιχνίδια πολλαπλών παικτών αποτελούν ιδιαίτερη κατηγορία ψηφιακών παιχνιδιών, καθώς επιτρέπουν σε πολλούς παίκτες να συμμετέχουν ταυτόχρονα σε έναν κοινό ψηφιακό κόσμο, είτε σε συνεργατικό είτε σε ανταγωνιστικό πλαίσιο [7][8]. Η δυνατότητα αυτή ενισχύει την κοινωνική αλληλεπίδραση, τη συνεργασία και την ανταλλαγή στρατηγικών, ενώ προσφέρει μια πιο δυναμική και αλληλεπιδραστική εμπειρία σε σχέση με τα παιχνίδια για έναν μόνο παίκτη [9].

Μια επίσης ιδιαίτερα δημοφιλής υποκατηγορία παιχνιδιών πολλαπλών παικτών είναι τα παιχνίδια ρόλων (RPG). Σε αυτά, ο παίκτης αναλαμβάνει τον ρόλο ενός ή περισσότερων χαρακτήρων μέσα σε έναν φανταστικό κόσμο, ενώ βασικός σκοπός του παιχνιδιού αποτελεί η εξέλιξη του χαρακτήρα, η ενίσχυση των ικανοτήτων του και η ενεργή συμμετοχή στην πλοκή και τις αποστολές. Ο συνδυασμός αυτών των στοιχείων καθιστά τα παιχνίδια ρόλων ιδιαίτερα ενδιαφέροντα, καθώς η ανάληψη ρόλων επιτρέπει στους παίκτες να διαμορφώνουν την ιστορία μέσα από τις επιλογές και τις ενέργειές τους, οδηγώντας συχνά σε διαφορετικά αποτελέσματα ανάλογα με τον

χαρακτήρα και το ύφος παιχνιδιού που ακολουθεί κάθε παίκτης [10].

Παράλληλα, τα παιχνίδια πολλαπλών παικτών φέρουν τεχνικές προκλήσεις απόδοσης όπως η σχεδίαση σταθερής επικοινωνίας μεταξύ παικτών και διακομιστή, πράγμα που καθιστά την ανάπτυξή τους πιο σύνθετη αλλά ταυτόχρονα πιο ενδιαφέρουσα. Η σωστή ισορροπία μεταξύ εμπειρίας χρήστη και απόδοσης συστήματος είναι καθοριστική για την επιτυχία τους, καθώς επηρεάζει άμεσα τη διάσταση του παιχνιδιού [11].

1.3 Σκοπός

Σκοπός της πτυχιακής εργασίας είναι η σχεδίαση και υλοποίηση ενός Massively Multiplayer Online Role Playing Game (MMORPG) παιχνιδού, όπου πολλοί παίκτες μπορούν να αλληλεπιδρούν ταυτόχρονα, σε πραγματικό χρόνο.

Το παιχνίδι διαδραματίζεται σε έναν μαγικό, φανταστικό κόσμο όπου τα στοιχεία της φύσης ελέγχουν το περιβάλλον, ενώ μαγεία αρχαίας και σύγχρονης τεχνολογίας συνυπάρχουν. Σε αυτόν τον διαρκώς εξελισσόμενο χώρο, οι παίκτες αναλαμβάνουν διαφορετικούς ρόλους, καλούμενοι να συνεργαστούν, να αναπτυχθούν και να ανταποκριθούν στις προκλήσεις του κόσμου.

Η εργασία επικεντρώνεται κυρίως στην υλοποίηση της δικτυακής αρχιτεκτονικής και στη σχεδίαση του διακομιστή που θα εξασφαλίζει αδιάλειπτη λειτουργία, συγχρονισμό δεδομένων και ομαλή επικοινωνία μεταξύ των παικτών. Ιδιαίτερη έμφαση δίνεται στη σταθερότητα του συστήματος, στη διαχείριση πολλαπλών συνδέσεων και στη βελτιστοποίηση της απόδοσης ώστε να διατηρείται ικανοποιητική εμπειρία χρήσης ανεξαρτήτως του αριθμού των ενεργών παικτών.

Παράλληλα, η ανάπτυξη του παιχνιδιού αποσκοπεί στο να αναδείξει τις προκλήσεις και τις τεχνικές λύσεις που σχετίζονται με την επικοινωνία πελάτη διακομιστή, προσφέροντας ένα παράδειγμα ολοκληρωμένης διαδικτυακής εφαρμογής στον χώρο των ηλεκτρονικών παιχνιδιών.

1.4 Παρόμοια Έργα

Ένα σύγχρονο παιχνίδι πολλαπλών παικτών με ρόλους είναι το Legends of Aria [12], το οποίο προσφέρει ελευθερία στον παίκτη όσον αφορά τον τρόπο παιχνιδιού. Ο παίκτης τοποθετείται στον κόσμο χωρίς καμία καθοδήγηση και καλείται να δημιουργήσει μόνος του το δικό του μονοπάτι. Το σύστημα εξέλιξης βασίζεται σε δεξιότητες, οι οποίες μπορούν να αναπτυχθούν ανεξάρτητα μεταξύ τους. Ο παίκτης διαθέτει συνολικά 600 πόντους δεξιοτήτων, γεγονός που του επιτρέπει να δημιουργεί μοναδικές, ευέλικτες και συνεργιστικές εξειδικεύσεις, αντί για κλειδωμένες κλάσεις. Η μάχη χαρακτηρίζεται από πλήρη ελευθερία παίκτη εναντίον παίκτη και παίκτη εναντίον περιβάλλοντος. Σε περίπτωση που ο παίκτης σκοτωθεί από άλλον πραγματικό παίκτη, χάνει ολόκληρο το απόθεμά του, το οποίο μπορεί να συλλεχθεί από οποιονδήποτε χαρακτηριστικό που ενισχύει την ένταση και το ρίσκο στις αλληλεπιδράσεις. Πέρα από τις αλληλεπιδράσεις με άλλους παίκτες, ο χρήστης μπορεί να έρθει σε επαφή και με

τέρατα ή γενικότερα πλάσματα του κόσμου, τα οποία ελέγχονται από τεχνητή νοημοσύνη και παίζουν σημαντικό ρόλο στο εμπόριο, τις αποστολές και τη λειτουργία του κόσμου.

Figure 1: Ο κόσμος του Legends of Aria

Η παραπάνω εικόνα απεικονίζει μια πανοραμική θέα ενός οικισμού στο παιχνίδι Legends of Aria. Ο χαρακτήρας που απεικονίζεται προσφέρει οπτική τρίτου προσώπου, ενώ τα κτίρια και τα δέντρα δημιουργούν ένα ζωντανό και ατμοσφαιρικό περιβάλλον.

Ένα άλλο σύγχρονο παιχνίδι πολλαπλών παικτών με ρόλους είναι το New World [13], το οποίο τοποθετεί τον παίκτη σε έναν ανοιχτό και δυναμικό κόσμο εμπνευσμένο από την εποχή της αποικιοκρατίας. Ο παίκτης ξεκινά στο νησί Aeternum, ένα μυστηριώδες και μαγικό περιβάλλον γεμάτο κινδύνους, ευκαιρίες και αρχαία μυστικά, όπου καλείται να χαράξει τη δική του πορεία χωρίς αυστηρή καθοδήγηση. Το σύστημα εξέλιξης του παιχνιδιού βασίζεται σε ένα ευέλικτο μοντέλο δεξιοτήτων, όπου ο τρόπος μάχης καθορίζουν τον ρόλο του παίκτη. Δεν υπάρχουν προκαθορισμένες κλάσεις, ο παίκτης αναπτύσσει τις ικανότητές του μέσω της χρήσης διαφορετικών όπλων, επιτρέποντας τη δημιουργία μοναδικών συνδυασμών και εξειδικεύσεων. Η μάχη στο New World δίνει έμφαση στη δράση και στο χρονικό σημείο, τόσο σε παίκτη εναντίον περιβάλλοντος όσο και σε έντονες μάχες παίκτη εναντίον παίκτη. Σε καταστάσεις παίκτη εναντίον παίκτη, ο εξοπλισμός και οι πόροι του παίκτη αποκτούν κρίσιμη σημασία, καθώς η νίκη ή η ήττα μπορεί να επηρεάσει την πρόοδο και τον έλεγχο περιοχών στον χάρτη. Πέρα από τις μάχες, ο κόσμος του Aeternum φιλοξενεί πλήθος πλασμάτων και αντιπάλων που ελέγχονται από τεχνητή νοημοσύνη, ενώ το σύστημα κατασκευής αντικειμένων, συλλογής πόρων και το εμπόριο παίζουν καθοριστικό ρόλο στην οικονομία και στη λειτουργία των οικισμών. Οι παίκτες μπορούν να ενταχθούν σε φατρίες, να συμμετέχουν σε πολιορκίες και να διαμορφώνουν ενεργά το κοινωνικό τοπίο του παιχνιδιού.

Figure 2: Ο κόσμος του New World

Η παραπάνω εικόνα απεικονίζει έναν χαρακτήρα του *New World* σε οπτική τρίτου προσώπου, μαζί με τον εξοπλισμό που χρησιμοποιεί για τη μάχη, ενώ το περιβάλλον γύρω του δείχνει την ανοιχτή και εξερευνητική φύση του κόσμου του παιχνιδιού.

Η ιδέα για την ανάπτυξη της παρούσας πτυχιακής εργασίας προήλθε από το Warcraft III [14], ένα από τα πιο εμβληματικά παιχνίδια στρατηγικής που συνδύαζε στοιχεία δράσης, τακτικής και ανάπτυξης χαρακτήρων. Το παιχνίδι βασίζεται σε ένα πλούσιο σύστημα ηρώων, όπου κάθε ήρωας διαθέτει διαφορετικές ικανότητες (skills), τα οποία επηρεάζουν άμεσα τον ρυθμό και το ύφος της μάχης. Η ποικιλία αυτή στις ικανότητες, από επιθετικά ξόρκια μέχρι θεραπευτικά ή ενισχυτικά, αποτέλεσε βασική πηγή έμπνευσης για τη σχεδίαση του συστήματος με τις κλάσεις ηρώων και ικανοτήτων στην παρούσα πτυχιακή. Παράλληλα, το Warcraft III διακρίνεται για τους μηχανισμούς διαχείρισης πόρων, την εξέλιξη των ηρώων και την προοδευτική αύξηση δυσκολίας κατά την πορεία του παιχνιδιού. Αυτά τα στοιχεία βοήθησαν στο σχεδιασμό της οικονομίας, των αποστολών και της σταδιακής προόδου του παίκτη. Επίσης η δυνατότητα προσαρμογής των ηρώων και η ποικιλία στρατηγικών που προσέφερε το Warcraft III επηρέασε τον σχεδιασμό των δεξιοτήτων στο παιχνίδι της παρούσας πτυχιακής, δίνοντας στους παίκτες την ελευθερία να διαμορφώνουν τον τρόπο παιχνιδιού τους.

Figure 3: Ο κόσμος και οι ήρωες του παιχνιδιού εμπνευσμένοι από το Warcraft III

Η παραπάνω εικόνα δείχνει τον χάρτη του Warcraft III μαζί με τους χαρακτήρες, όπου στον έναν από αυτούς βλέπουμε το μέρος που κρατάει τα αντικείμενά του (Inventory), τα στατιστικά του και τις κινήσεις που μπορεί να κάνει.

Επιπλέον, σημαντική επιρροή αποτέλεσε το Rogue Heroes: Ruins of Tasos [15], ένα συνεργατικό παιχνίδι δράσης με top down κάμερα και με έμφαση στην ομαδικότητα, την εξερεύνηση και την ανακάλυψη εξοπλισμού. Από το Rogue Heroes προήλθε η ιδέα για τη μορφή της κάμερας ένα top down perspective που επιτρέπει την πλήρη ορατότητα του περιβάλλοντος και του χαρακτήρα, όμως σε αντίθεση με το Rogue Heroes, κάθε παίκτης θα διαθέτει τη δική του κάμερα και ανεξάρτητη οπτική, χωρίς κοινό πεδίο προβολής. Επίσης αντλήθηκαν ιδέες που σχετίζονται με τη συνεργασία μεταξύ παικτών, τη δυναμική μάχη και τη συνεχή βελτίωση των χαρακτήρων μέσω εξοπλισμού και αναβαθμίσεων.

Figure 4: Το σύστημα top-down κάμερας και το σύστημα με τα πλακίδια εμπνευσμένα από το Rogue Heroes: Ruins of Tasos

Η παραπάνω εικόνα δείχνει τον κόσμο παιχνιδιού του Rogue Heroes φτιαγμένο σε πλακίδια (tiles) αλλά και τον χειρισμό της κάμερας που δείχνει ολόκληρο τον χαρακτήρα.

Τα παραπάνω παιχνίδια αποτέλεσαν βασική πηγή έμπνευσης για την ανάπτυξη της παρούσας πτυχιακής εργασίας. Ο συνδυασμός μηχανισμών αυτών των δύο παιχνιδιών δημιούργησε την ιδέα για τη ανάπτυξη ενός παιχνιδιού που ενσωματώνει τακτική σκέψη και εξέλιξη χαρακτήρων, ενώ ταυτόχρονα προσφέρει δυναμική και συνεργατική εμπειρία μάχης σε πραγματικό χρόνο, όπως απαιτεί ένα σύγχρονο MMORPG. Επιπλέον, στόχος στην ανάπτυξης ενός MMORPG είναι να υπάρχει αποδοτική δικτυακή επικοινωνία μεταξύ των παικτών και του διακομιστή, ώστε να διασφαλίζεται σταθερή και ομαλή εμπειρία παιχνιδιού καθώς και μικρή καθυστέρηση (latency).

Στην παρούσα πτυχιακή θα αναπτυχθεί ένα παιχνίδι που συνδυάζει τους ρόλους και τις δυνατότητες χαρακτήρων του Warcraft III όπως οι κλάσεις, τα ξόρκια, οι ικανότητες και οι επιθέσεις με τους μηχανισμούς του Rogue Heroes, όπως το σύστημα κάμερας και η δομή του κόσμου σε πλακίδια.

1.5 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 πτυχιακής εργασίας. Στο κεφάλαιο 3 αναδεικνύεται η σχεδίαση του παιχνιδιού, περιγράφοντας αναλυτικά τη δομή, τους μηχανισμούς και τις βασικές λειτουργίες του. Στο Κεφάλαιο 4 παρουσιάζεται η υλοποίηση του παιχνιδιού, οι επιλογές που έγιναν για τη βελτίωση της απόδοσης, καθώς και οι μέθοδοι που χρησιμοποιήθηκαν για τη μείωση του φόρτου στο δίκτυο, ώστε να ελαχιστοποιηθούν οι καθυστερήσεις (latency) κατά τη διαδικτυακή επικοινωνία των παικτών. Στο Κεφάλαιο 5 γίνεται αξιολόγηση του παιχνιδιού από χρήστες, βασισμένη σε προκαθορισμένα κριτήρια, όπως η εμπειρία χρήστη και η λειτουργικότητα των μηχανισμών. Τέλος, στο Κεφάλαιο 6 παρατίθενται τα τελικά συμπεράσματα της εργασίας, καθώς και προτάσεις για μελλοντικές επεκτάσεις και βελτιώσεις του παιχνιδιού.

Κεφάλαιο 2: Θεωρητικό Υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζονται τα λογισμικά και τα εργαλεία που χρησιμοποιήθηκαν κατά τη διάρκεια της πτυχιακής εργασίας, καθώς και οι βασικές θεωρητικές έννοιες που σχετίζονται με την ανάπτυξη ψηφιακών παιχνιδιών. Επιπλέον, αναλύονται οι κατηγορίες στις οποίες εντάσσεται το παιχνίδι που αναπτύχθηκε.

2.1 Εργαλεία ανάπτυξης παιχνιδιού

Στην παρούσα πτυχιακή εργασία χρησιμοποιήθηκαν διάφορα εργαλεία λογισμικού για την ανάπτυξη του ψηφιακού παιχνιδιού. Τα εργαλεία αυτά περιλαμβάνουν τη γλώσσα προγραμματισμού, καθώς και τις βιβλιοθήκες που υποστηρίζουν την ανάπτυξη δισδιάστατων παιχνιδιών. Η επιλογή τους έγινε με βάση την ευχρηστία, την ευελιξία και την καταλληλότητά τους για ακαδημαϊκή χρήση.

2.1.1 Γλώσσα Προγραμματισμού Python

Η Python είναι μία υψηλού επιπέδου, διερμηνευόμενη γλώσσα προγραμματισμού, η οποία χρησιμοποιείται ευρέως σε ποικίλους τομείς της πληροφορικής [16]. Η απλότητα της σύνταξής της, η αναγνωσιμότητα του κώδικα και η εκτεταμένη υποστήριξη βιβλιοθηκών την καθιστούν κατάλληλη επιλογή για την ανάπτυξη δισδιάστατων παιχνιδιών και διαδραστικών εφαρμογών [17].

Στην παρούσα πτυχιακή εργασία, το παιχνίδι που υλοποιήθηκε κατατάσσεται στην κατηγορία των δισδιάστατων (2D) παιχνιδιών, στα οποία η απεικόνιση και η κίνηση των αντικειμένων πραγματοποιούνται σε επίπεδο δύο διαστάσεων, συνήθως στους άξονες x και y, χωρίς πραγματικό βάθος στον χώρο [18].

2.1.2 Βιβλιοθήκη Python Arcade

Για την υλοποίηση του παιχνιδιού χρησιμοποιήθηκε η βιβλιοθήκη Python Arcade [19], η οποία αποτελεί βιβλιοθήκη ανάπτυξης 2D παιχνιδιών και όχι ολοκληρωμένη μηχανή παιχνιδιών (game engine). Μία μηχανή παιχνιδιών (game engine) αποτελεί ένα ολοκληρωμένο λογισμικό πλαίσιο που παρέχει εργαλεία και συστήματα για τη διαχείριση γραφικών, φυσικής, ήχου, σεναρίων και πόρων, διευκολύνοντας την ανάπτυξη ψηφιακών παιχνιδιών μέσω ενοποιημένου περιβάλλοντος εργασίας [20].

Η βιβλιοθήκη παρέχει λειτουργίες για τη δημιουργία γραφικών, τη διαχείριση sprites, την ανίχνευση συγκρούσεων και τον χειρισμό εισόδου από τον χρήστη. Η βιβλιοθήκη βασίζεται σε τεχνολογίες OpenGL για την απόδοση των γραφικών και επιτρέπει την ανάπτυξη δισδιάστατων παιχνιδιών μέσω καθαρού κώδικα Python, χωρίς τη χρήση γραφικού περιβάλλοντος επεξεργασίας. Η επιλογή της Python Arcade έγινε λόγω της απλότητας χρήσης της, καθώς και της διαθεσιμότητας εκτενούς υλικού καθοδήγησης και υποστηρικτικών πηγών στο διαδίκτυο.

2.1.3 OpenGL

Το OpenGL αποτελεί ένα πρότυπο προγραμματιστικό περιβάλλον γραφικών που χρησιμοποιείται για την απόδοση δισδιάστατων και τρισδιάστατων γραφικών [21]. Παρέχει ένα σύνολο συναρτήσεων για τη σχεδίαση γεωμετρικών σχημάτων, τη διαχείριση χρωμάτων και την αξιοποίηση της κάρτας γραφικών για αποδοτική απεικόνιση γραφικών. Η χρήση του OpenGL από τη βιβλιοθήκη Python Arcade επιτρέπει την αποδοτική απόδοση γραφικών, αξιοποιώντας τις δυνατότητες της κάρτας γραφικών, χωρίς να απαιτείται άμεση αλληλεπίδραση με το OpenGL.

2.1.4 IDE

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) αποτελεί ένα λογισμικό εργαλείο που συνδυάζει λειτουργίες επεξεργασίας κώδικα, μεταγλώττισης ή εκτέλεσης, εντοπισμού σφαλμάτων και διαχείρισης έργων σε ένα ενιαίο περιβάλλον [22]. Η χρήση ενός IDE συμβάλλει στην οργάνωση της διαδικασίας ανάπτυξης λογισμικού, παρέχοντας εργαλεία που υποστηρίζουν την αποτελεσματική συγγραφή, εκτέλεση και έλεγχο του κώδικα.

Για την ανάπτυξη του παιχνιδιού χρησιμοποιήθηκε το ολοκληρωμένο περιβάλλον ανάπτυξης Visual Studio Code (VS Code). Το VS Code υποστηρίζει τη γλώσσα Python μέσω επεκτάσεων, προσφέροντας λειτουργίες όπως επισήμανση σύνταξης, αυτόματη συμπλήρωση κώδικα, εντοπισμό σφαλμάτων και διαχείριση έργων [23].

Το IDE επιλέχθηκε λόγω της προϋπάρχουσας εμπειρίας με το συγκεκριμένο λογισμικό, γεγονός που διευκόλυνε τη διαδικασία ανάπτυξης. Επιπλέον, πρόκειται για ένα διαλειτουργικό εργαλείο, το οποίο αναπτύσσεται από τη Microsoft και υποστηρίζει τα λειτουργικά συστήματα Windows, Linux και macOS.

Παρέχει επίσης ενσωματωμένη υποστήριξη για έλεγχο εκδόσεων μέσω Git, καθώς και δυνατότητα επέκτασης των λειτουργιών του μέσω πρόσθετων (extensions), επιτρέποντας την προσαρμογή του περιβάλλοντος ανάπτυξης στις ανάγκες του εκάστοτε έργου.

2.1.5 GitHub

Για τη διαχείριση του πηγαίου κώδικα του παιχνιδιού χρησιμοποιήθηκε η πλατφόρμα GitHub, η οποία βασίζεται στο σύστημα ελέγχου εκδόσεων Git. Το GitHub επιτρέπει την αποθήκευση, οργάνωση και παρακολούθηση αλλαγών στον πηγαίο κώδικα, διευκολύνοντας τη διαχείριση της εξέλιξης ενός έργου λογισμικού [24].

Μέσω του GitHub παρέχεται η δυνατότητα ελέγχου εκδόσεων, καταγραφής αλλαγών και επαναφοράς προηγούμενων καταστάσεων του κώδικα, συμβάλλοντας στη συστηματική ανάπτυξη και συντήρηση του λογισμικού. Επιπλέον, η πλατφόρμα υποστηρίζει τη συνεργατική ανάπτυξη και την ενσωμάτωση εργαλείων ανάπτυξης, όπως ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDE).

2.1.6 Δικτύωση (TCP, ZeroMQ και asyncio)

Η δικτυακή επικοινωνία στο πλαίσιο της παρούσας πτυχιακής εργασίας βασίστηκε στο πρωτόκολλο TCP (Transmission Control Protocol), το οποίο παρέχει αξιόπιστη, προσανατολισμένη στη σύνδεση μεταφορά δεδομένων μεταξύ συστημάτων [25]. Το TCP εξασφαλίζει την ορθή και διαδοχική παράδοση των δεδομένων, γεγονός που το καθιστά κατάλληλο για εφαρμογές που απαιτούν αξιόπιστη επικοινωνία.

Για την υλοποίηση της επικοινωνίας χρησιμοποιήθηκε η βιβλιοθήκη ZeroMQ, η οποία παρέχει μηχανισμούς ανταλλαγής μηνυμάτων βασισμένους σε καθορισμένα πρότυπα επικοινωνίας μεταξύ διεργασιών και συστημάτων. Η βιβλιοθήκη επιτρέπει την ανταλλαγή δεδομένων με οργανωμένο τρόπο, υποστηρίζοντας διαφορετικούς τρόπους επικοινωνίας μεταξύ εφαρμογών, χωρίς να απαιτείται η άμεση διαχείριση των τεχνικών λεπτομερειών της σύνδεσης [26].

Μέσω αυτής της προσέγγισης, η ZeroMQ απλοποιεί την ανάπτυξη δικτυακών εφαρμογών, παρέχοντας έναν αφαιρετικό μηχανισμό επικοινωνίας πάνω από πρωτόκολλα μεταφοράς όπως το TCP, ενώ ταυτόχρονα διατηρεί υψηλή απόδοση και ευελιξία στη σχεδίαση της επικοινωνίας.

Η επιλογή της ZeroMQ πραγματοποιήθηκε λόγω της απλότητας στη χρήση της, καθώς επιτρέπει την ανάπτυξη δικτυακών εφαρμογών χωρίς την ανάγκη διαχείρισης χαμηλού επιπέδου λεπτομερειών.

Παράλληλα, αξιοποιήθηκε η βιβλιοθήκη asyncio της Python για την υποστήριξη ασύγχρονης και μη αποκλειστικής (non-blocking) εκτέλεσης, επιτρέποντας τη διαχείριση δικτυακών λειτουργιών χωρίς παρεμπόδιση της κύριας ροής εκτέλεσης του προγράμματος [27]. Ο συνδυασμός TCP, ZeroMQ και asyncio συνέβαλε στην αποδοτική και αξιόπιστη υλοποίηση της δικτυακής επικοινωνίας.

2.1.7 Βάση Δεδομένων

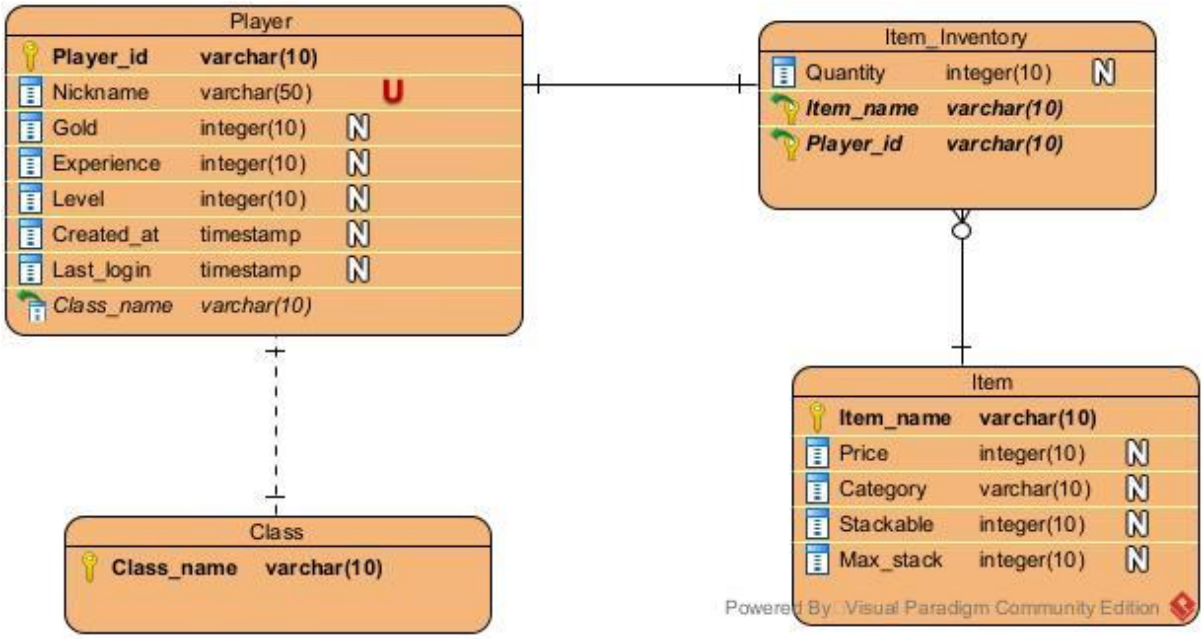
Για την αποθήκευση και διαχείριση των δεδομένων του παιχνιδιού χρησιμοποιήθηκε η βάση δεδομένων SQLite, μέσω της βιβλιοθήκης sqlite3 της Python [28]. Η βάση δεδομένων αξιοποιήθηκε για την αποθήκευση πληροφοριών που σχετίζονται με τον παίκτη, όπως η κλάση του χαρακτήρα, τα βασικά στοιχεία προόδου και τα αντικείμενα που διαθέτει. Μέσω της αποθήκευσης αυτών των δεδομένων καθίσταται δυνατή η διαχείριση πολλαπλών παικτών, καθώς και η διατήρηση της προόδου τους. Επιπλέον, η χρήση βάσης δεδομένων παρέχει τη δυνατότητα επέκτασης του παιχνιδιού στο μέλλον, υποστηρίζοντας πιο σύνθετους μηχανισμούς gameplay και διαχείρισης δεδομένων.

2.1.7.1 Διάγραμμα Οντοτήτων-Συσχετίσεων (ER)

Στο πλαίσιο του σχεδιασμού της βάσης δεδομένων του παιχνιδιού, δημιουργήθηκε διάγραμμα οντοτήτων-συσχετίσεων (Entity–Relationship diagram – ER), το οποίο αποτυπώνει τη δομή των δεδομένων και τις σχέσεις μεταξύ των βασικών οντοτήτων του συστήματος.

Κεντρική οντότητα αποτελεί ο παίκτης (Player), ο οποίος συνδέεται με την κλάση του χαρακτήρα και το απόθεμά του. Η οντότητα Class αποθηκεύει τις διαθέσιμες κλάσεις χαρακτήρων, ενώ η οντότητα Item περιλαμβάνει τα αντικείμενα που μπορούν να χρησιμοποιηθούν ή να διαχειριστούν μέσα στο παιχνίδι. Η σχέση ανάμεσα στον παίκτη και τα αντικείμενα αποτυπώνεται μέσω του πίνακα Item_Inventory, ο οποίος επιτρέπει τη σύνδεση κάθε παίκτη με τα αντικείμενα που διαθέτει, καθώς και την αποθήκευση της ποσότητας κάθε αντικειμένου.

Figure 5: Διάγραμμα Οντοτήτων-Συσχετίσεων



2.1.7.2 Λειτουργία πινάκων στο παιχνίδι

Με βάση το παραπάνω διάγραμμα, η λειτουργία της βάσης δεδομένων ενσωματώνεται άμεσα στη ροή του παιχνιδιού. Κατά τη διαδικασία δημιουργίας νέου παίκτη, ο παίκτης καλείται να επιλέξει ψευδώνυμο (nickname) και κλάση χαρακτήρα. Τα στοιχεία αυτά ελέγχονται ως προς την εγκυρότητά τους και, εφόσον είναι αποδεκτά, πραγματοποιούνται οι αντίστοιχες εγγραφές (insert) στους πίνακες της βάσης δεδομένων.

Συγκεκριμένα, κατά τη δημιουργία νέου παίκτη πραγματοποιείται εγγραφή στον πίνακα Player, όπου αποθηκεύονται τα βασικά στοιχεία του, όπως το ψευδώνυμο, η επιλεγμένη κλάση χαρακτήρα, το επίπεδο, η εμπειρία, ο χρυσός και οι χρονικές πληροφορίες δημιουργίας ή τελευταίας σύνδεσης. Οι διαθέσιμες κλάσεις αντιστοιχούν σε συγκεκριμένες προκαθορισμένες τιμές, οι οποίες καθορίζουν τα χαρακτηριστικά και τις δυνατότητες του χαρακτήρα μέσα στο παιχνίδι.

Ο πίνακας Class χρησιμοποιείται για την αποθήκευση των διαθέσιμων κλάσεων χαρακτήρων, ενώ ο πίνακας Item περιλαμβάνει τα αντικείμενα του παιχνιδιού, όπως την ονομασία, την κατηγορία, την τιμή, τη δυνατότητα στοίβαξης και το μέγιστο πλήθος ανά στοίβα. Με αυτόν τον τρόπο, τα αντικείμενα οργανώνονται σε ξεχωριστή οντότητα και μπορούν να συσχετιστούν με διαφορετικούς παίκτες.

Η σύνδεση των παικτών με τα αντικείμενα πραγματοποιείται μέσω του πίνακα Item_Inventory. Ο πίνακας αυτός λειτουργεί ως πίνακας συσχέτισης ανάμεσα στον Player και το Item, αποθηκεύοντας το αναγνωριστικό του παίκτη, το όνομα του αντικειμένου και την ποσότητα που διαθέτει ο παίκτης. Έτσι, κάθε παίκτης μπορεί να έχει πολλά αντικείμενα στο απόθεμά του, ενώ το ίδιο αντικείμενο μπορεί να υπάρχει στο απόθεμα πολλών διαφορετικών παικτών.

Κατά τη διάρκεια του παιχνιδιού, όταν ο παίκτης αποκτά ή αγοράζει αντικείμενα, δημιουργούνται νέες εγγραφές ή ενημερώνονται υπάρχουσες εγγραφές στον πίνακα Item_Inventory. Η προβολή του inventory πραγματοποιείται μέσω των δεδομένων που αντλούνται από τον πίνακα Item_Inventory σε συνδυασμό με τον πίνακα Item, ώστε να εμφανίζονται τα αντικείμενα που διαθέτει ο παίκτης και οι αντίστοιχες ποσότητές τους.

2.1.8 Tiled

Για τη δημιουργία και επεξεργασία του χάρτη (map) του παιχνιδιού χρησιμοποιήθηκε η δωρεάν εφαρμογή Tiled, η οποία αποτελεί εργαλείο σχεδίασης χαρτών βασισμένων σε πλακίδια (tiles). Μέσω του Tiled κατέστη δυνατή η οργάνωση και τοποθέτηση των διάφορων σετ πλακιδίων (tilesets), καθώς και η διαμόρφωση του περιβάλλοντος του παιχνιδιού με δομημένο και αποδοτικό τρόπο [29].

Η χρήση του Tiled διευκόλυνε τη διαδικασία σχεδίασης του χάρτη, προσφέροντας ευελιξία στην τροποποίηση και επέκταση του περιβάλλοντος, γεγονός που είναι ιδιαίτερα χρήσιμο για δισδιάστατα παιχνίδια τύπου top-down.

2.1.9 Διαγράμματα

Για τη δημιουργία των διαγραμμάτων που χρησιμοποιήθηκαν στην παρούσα πτυχιακή εργασία αξιοποιήθηκε το διαδικτυακό εργαλείο draw.io [30]. Το εργαλείο χρησιμοποιήθηκε για την αποτύπωση της δομής και της λειτουργίας του συστήματος μέσω διαγραμμάτων, συμβάλλοντας στην καλύτερη κατανόηση της αρχιτεκτονικής και των βασικών ροών του παιχνιδιού.

2.1.10 Piskel

Για την επεξεργασία των γραφικών του παιχνιδιού χρησιμοποιήθηκε το εργαλείο Piskel [36], το οποίο αποτελεί εφαρμογή δημιουργίας και επεξεργασίας pixel art και sprites για δισδιάστατα παιχνίδια. Μέσω του Piskel πραγματοποιήθηκε η επεξεργασία των spritesheets που χρησιμοποιήθηκαν για την απεικόνιση του χαρακτήρα και των κινήσεών του μέσα στο παιχνίδι.

Συγκεκριμένα, τα αρχικά spritesheets περιείχαν καρέ τοποθετημένα με τρόπο που δεν εξυπηρετούσε άμεσα τη χρήση τους στην υλοποίηση του παιχνιδιού, καθώς οι διαφορετικές κατευθύνσεις προς τις οποίες ήταν στραμμένος ο χαρακτήρας, αλλά και οι καταστάσεις animation, ήταν ανακατεμένες.

Figure 6: Death animation χαρακτήρα για δεξιά κατεύθυνση)

Για τον λόγο αυτό, πραγματοποιήθηκε αναδιάταξη των sprites, ώστε τα καρέ που αντιστοιχούν στην ίδια κατάσταση, όπως για παράδειγμα η κατάσταση θανάτου (death), να ομαδοποιηθούν ανά κατεύθυνση και να οργανωθούν με πιο δομημένο τρόπο. Με τον τρόπο αυτό διευκολύνθηκε η διαχείριση των animations στον κώδικα και η ορθή αντιστοίχισή τους με τις κινήσεις του χαρακτήρα.

Figure 7: Death animation χαρακτήρα για όλες τις κατευθύνσεις

Επιπλέον, μέσω του Piskel πραγματοποιήθηκε αλλαγή του μεγέθους των sprites, ώστε να προσαρμοστούν στις διαστάσεις των υπόλοιπων χαρακτήρων του παιχνιδιού και να επιτευχθεί πιο ομαλή αλληλεπίδραση με τα εμπόδια του περιβάλλοντος.

2.2 Πόροι Περιεχομένου Παιχνιδιού (Game Assets)

Για την πτυχιακή εργασία χρησιμοποιήθηκαν πόροι περιεχομένου (game assets) για την οπτική και ηχητική υλοποίηση του παιχνιδιού. Οι πόροι αυτοί περιλαμβάνουν γραφικά στοιχεία χαρακτήρων, αντικειμένων και περιβάλλοντος, μουσική και ηχητικά εφέ, καθώς και οπτικό υλικό που αξιοποιήθηκε στο μενού επιλογών. Οι συγκεκριμένοι πόροι προέρχονται από δωρεάν πλατφόρμες ή δημιουργήθηκαν με τη βοήθεια εργαλείων τεχνητής νοημοσύνης και χρησιμοποιήθηκαν με στόχο τη βελτίωση της αισθητικής, της χρηστικότητας και της συνολικής εμπειρίας του παίκτη.

2.2.1 Gemini

Για τη δημιουργία και τον σχεδιασμό των καρτών επιλογής κλάσης και χαρακτήρων αξιοποιήθηκε το εργαλείο Gemini και συγκεκριμένα το μοντέλο δημιουργίας εικόνων Nano Banana, το οποίο αποτελεί σύστημα τεχνητής νοημοσύνης για την παραγωγή οπτικού περιεχομένου. Το Gemini χρησιμοποιήθηκε υποστηρικτικά κατά τη διαδικασία σχεδίασης, παρέχοντας ιδέες και προτάσεις για το περιεχόμενο και τη μορφή των καρτών [31].

2.2.2 Χαρακτήρες

Για την οπτική αναπαράσταση των χαρακτήρων του παιχνιδιού χρησιμοποιήθηκαν sprites χαρακτήρων με ενσωματωμένες κινήσεις (animations) από την πλατφόρμα CraftPix, τα οποία διατίθενται δωρεάν [32]. Τα γραφικά αυτά στοιχεία είναι σχεδιασμένα για top-down gameplay και περιλαμβάνουν διαφορετικές καταστάσεις κίνησης, επιτρέποντας την ομαλή απεικόνιση των χαρακτήρων κατά την κίνηση και την εκτέλεση ενεργειών μέσα στο παιχνίδι.

2.2.3 Assets

Για την υλοποίηση των γραφικών στοιχείων του παιχνιδιού χρησιμοποιήθηκαν sprites αντικειμένων (items) και εικονιδίων διεπαφής χρήστη (UI icons) από διαδικτυακές πλατφόρμες παροχής γραφικών πόρων, όπως οι OpenGameArt [33], itch.io [34] και CraftPix, τα οποία διατίθενται για δωρεάν χρήση. Τα συγκεκριμένα assets αξιοποιήθηκαν για την οπτική αναπαράσταση αντικειμένων που αλληλεπιδρούν με τον παίκτη, καθώς και για τη διαμόρφωση της διεπαφής του παιχνιδιού.

2.2.4 Tilesets

Για τη δημιουργία του χάρτη (map) του παιχνιδιού χρησιμοποιήθηκαν tilesets, τα οποία αποτελούνται από επαναχρησιμοποιήσιμα γραφικά πλακίδια που συνδυάζονται για τη διαμόρφωση του περιβάλλοντος. Τα tilesets που αξιοποιήθηκαν προέρχονται από την πλατφόρμα CraftPix και διατίθενται για δωρεάν χρήση.

Η χρήση tilesets επέτρεψε την αποδοτική κατασκευή του χάρτη του παιχνιδιού, εξασφαλίζοντας οπτική συνοχή και ευκολία στην τροποποίηση ή επέκταση του περιβάλλοντος, γεγονός που είναι ιδιαίτερα χρήσιμο σε δισδιάστατα παιχνίδια τύπου top-down.

2.2.5 Επεξεργασία γραφικών στοιχείων

Κατά τη διαδικασία προετοιμασίας των γραφικών πόρων του παιχνιδιού χρησιμοποιήθηκε το διαδικτυακό εργαλείο remove.bg, για την αφαίρεση του φόντου (background) από εικόνες και sprites. Η αφαίρεση του φόντου επέτρεψε τη δημιουργία εικόνων με διαφανές υπόβαθρο (transparent background), ώστε να μπορούν να

ενσωματωθούν ομαλά στο περιβάλλον του παιχνιδιού χωρίς ανεπιθύμητα οπτικά στοιχεία [35].

2.2.6 Μουσική

Για τη διαμόρφωση του ηχητικού περιβάλλοντος του παιχνιδιού χρησιμοποιήθηκαν μουσικά κομμάτια και ηχητικά εφέ, τα οποία διατίθενται δωρεάν από την πλατφόρμα OpenGameArt. Μουσική χρησιμοποιήθηκε τόσο στο μενού του παιχνιδιού όσο και κατά τη διάρκεια του παιχνιδιού, συμβάλλοντας στη δημιουργία κατάλληλης ατμόσφαιρας και στη βελτίωση της εμπειρίας του χρήστη. Παράλληλα, αξιοποιήθηκαν ηχητικά εφέ για την παροχή ακουστικής ανατροφοδότησης στις ενέργειες του παίκτη και στα γεγονότα του παιχνιδιού.

2.3 Σύνοψη

Στο παρόν κεφάλαιο παρουσιάστηκε το θεωρητικό υπόβαθρο της πτυχιακής εργασίας, καθώς και τα βασικά εργαλεία και οι πόροι που αξιοποιήθηκαν για την ανάπτυξη του ψηφιακού παιχνιδιού. Αρχικά, αναλύθηκαν τα εργαλεία ανάπτυξης και οι τεχνολογίες που χρησιμοποιήθηκαν, όπως η γλώσσα προγραμματισμού Python, η βιβλιοθήκη Python Arcade, το OpenGL, τα εργαλεία ανάπτυξης λογισμικού και οι τεχνολογίες δικτύωσης, τα οποία συνέβαλαν στην υλοποίηση της λογικής και της αρχιτεκτονικής του παιχνιδιού.

Επιπλέον, παρουσιάστηκε ο σχεδιασμός της βάσης δεδομένων του παιχνιδιού, μέσω του διαγράμματος Οντοτήτων-Συσχετίσεων (ER), καθώς και η λειτουργία των βασικών πινάκων στο πλαίσιο του gameplay. Αναλύθηκε ο τρόπος με τον οποίο αποθηκεύονται και συσχετίζονται τα δεδομένα των παικτών, του εξοπλισμού και των αντικειμένων, υποστηρίζοντας τη διαχείριση της προόδου και του αποθέματος κατά τη διάρκεια του παιχνιδιού.

Στη συνέχεια, παρουσιάστηκαν οι πόροι περιεχομένου του παιχνιδιού, συμπεριλαμβανομένων των γραφικών στοιχείων, των χαρακτήρων, των tilesets, της μουσικής και των ηχητικών εφέ, οι οποίοι χρησιμοποιήθηκαν για τη διαμόρφωση της οπτικής και ακουστικής ταυτότητας του παιχνιδιού. Ο συνδυασμός των παραπάνω εργαλείων και πόρων αποτέλεσε τη βάση για την υλοποίηση του παιχνιδιού, όπως αυτή αναλύεται στο κεφάλαιο 4.

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 Celestial Lands, και είναι ένα παιχνίδι τύπου MMORPG (Massively Multiplayer Online Role Playing Game).

3.1 Εισαγωγή

Το Celestial Lands, που αναπτύχθηκε σε αυτη τη πτυχιακή εργασία, είναι ένα MMORPG συνεργατικού τύπου, όπου οι παίκτες βρίσκονται σε έναν χάρτη χωρισμένο σε τέσσερις περιοχές και καλούνται να συνεργαστούν για να ολοκληρώσουν αποστολές και μάχες ενάντια σε εχθρούς. Κάθε παίκτης επιλέγει μία από τις τρεις διαθέσιμες κλάσεις χαρακτήρων: Πολεμιστής, Μάγος ή Σκοπευτής. Κάθε κλάση διαθέτει ξεχωριστές ικανότητες και διαφορετικό τρόπο μάχης, προσφέροντας ποικιλία στον τρόπο παιχνιδιού.

Κατά τη διάρκεια του παιχνιδιού, οι παίκτες αποκτούν εμπειρία και χρυσό μέσα από την εξόντωση εχθρών. Η εμπειρία συμβάλλει στην εξέλιξη του χαρακτήρα και στη βελτίωση των στατιστικών του, ενώ ο χρυσός μπορεί να χρησιμοποιηθεί για την αγορά αντικειμένων. Σε περίπτωση ήττας, ο χαρακτήρας επανέρχεται μετά από σύντομο χρονικό διάστημα, διατηρώντας την πρόοδό του. Ο τελικός στόχος του παιχνιδιού είναι η εξόντωση του τελικού εχθρού και η πλήρης κατάκτηση του χάρτη, μέσα από συνεργασία, στρατηγική και συνεχή εξέλιξη των χαρακτήρων.

3.2 Κατηγορία

Το παιχνίδι ανήκει στην κατηγορία των 2D MMORPG και χρησιμοποιεί top-down perspective, δηλαδή η κάμερα προβάλλει τον κόσμο από ψηλά προς τα κάτω, επιτρέποντας στον παίκτη να βλέπει ολόκληρο τον χαρακτήρα και το περιβάλλον γύρω του. Το Celestial Lands αντλεί έμπνευση από παιχνίδια όπως το Warcraft III, κυρίως ως προς την οπτική γωνία και τη μάχη σε πραγματικό χρόνο, αλλά διαφοροποιείται καθώς επικεντρώνεται περισσότερο στη συνεργατική δράση πολλών παικτών και στη χρήση ικανοτήτων ανά κλάση.

Η κίνηση του χαρακτήρα, οι επιθέσεις και οι ικανότητες πραγματοποιούνται με τα πλήκτρα του πληκτρολογίου. Ο τρόπος χειρισμού και η αίσθηση της δράσης του παιχνιδιού παρουσιάζουν ομοιότητες με τίτλους όπως το Soul Knight και το Rogue Heroes, όπου ο παίκτης κινείται ελεύθερα στον χάρτη, αποφεύγει εχθρούς και χρησιμοποιεί επιθέσεις και ικανότητες σε πραγματικό χρόνο.

3.3 Μηχανισμοί Παιχνιδιού

Κάθε παίκτης ελέγχει έναν χαρακτήρα, ο οποίος ανήκει σε μία από τις τρεις διαθέσιμες κλάσεις: Πολεμιστής, Μάγος ή Σκοπευτής. Η επιλογή της κλάσης γίνεται στην αρχή του παιχνιδιού και δεν μπορεί να αλλάξει κατά τη διάρκεια της ίδιας

προσπάθειας. Κάθε κλάση διαθέτει τρεις βασικές ικανότητες: μία βασική επίθεση και δύο ειδικές ικανότητες με χρόνο επαναφόρτισης.

Η βασική επίθεση μπορεί να χρησιμοποιείται συχνά από τον παίκτη και αποτελεί τον κύριο τρόπο πρόκλησης ζημιάς στους εχθρούς. Δεν απαιτεί κατανάλωση ενέργειας και δεν έχει σημαντικό περιορισμό χρήσης. Αντίθετα, οι δύο ειδικές ικανότητες προκαλούν μεγαλύτερη ζημιά ή έχουν πιο ισχυρό αποτέλεσμα στη μάχη, όμως απαιτούν ενέργεια και διαθέτουν χρόνο επαναφόρτισης. Με αυτόν τον τρόπο, ο παίκτης πρέπει να χρησιμοποιεί τις ικανότητες στρατηγικά και όχι συνεχόμενα, επιλέγοντας την κατάλληλη στιγμή για κάθε επίθεση.

Οι εξοντώσεις εχθρών προσφέρουν εμπειρία και χρυσό στον χαρακτήρα. Η εμπειρία χρησιμοποιείται στο σύστημα επιπέδων, μέσω του οποίου βελτιώνονται τα στατιστικά του χαρακτήρα, όπως η ζωή και η ζημιά. Ο χρυσός μπορεί να χρησιμοποιηθεί για την αγορά αντικειμένων από το inventory του παίκτη.

Όταν κάποιος παίκτης πεθάνει, επανέρχεται σε κάποιο σημείο του χάρτη μετά από λίγο χρόνο. Το παιχνίδι τελειώνει όταν πεθάνει ο τελικός εχθρός, αλλά μπορεί να επαναληφθεί διατηρώντας την πρόοδο που έχει αποκτήσει κάθε παίκτης. Σε περίπτωση που δεν απομείνει κανένας ζωντανός παίκτης, το παιχνίδι ξεκινάει από την αρχή.

3.3.1 Κίνηση Χαρακτήρα

Η κίνηση του χαρακτήρα μέσα σε κάθε περιοχή πραγματοποιείται σε χάρτη που αποτελείται από πλακίδια (tiles), τα οποία συνθέτουν τη δομή και τη μορφολογία της εκάστοτε περιοχής. Ο συνολικός χάρτης του παιχνιδιού αποτελείται από τέσσερις διακριτές περιοχές, καθεμία με διαφορετικό περιβάλλον και επίπεδο δυσκολίας. Η μετακίνηση του χαρακτήρα γίνεται ελεύθερα και υποστηρίζει τέσσερις βασικές κατευθύνσεις: πάνω, κάτω, αριστερά και δεξιά. Οι ενέργειες του παίκτη, όπως η κίνηση, οι επιθέσεις και οι ικανότητες, εκτελούνται προς την κατεύθυνση στην οποία είναι στραμμένος ο χαρακτήρας του (facing direction).

3.3.2 Μηχανισμοί Μάχης

Οι μάχες έχουν συγκεκριμένη εμβέλεια και γίνονται με δύο τρόπους:

- Σώμα με σώμα (melee): γίνεται μόνο όταν ο εχθρός είναι στο αμέσως επόμενο πλακίδιο (σε οποιαδήποτε κατεύθυνση)

- Από απόσταση (ranged): γίνεται σε απομακρισμένα μεταξύ τους πλακίδια, ανάλογα με την εμβέλεια που έχει ο χαρακτήρας

Οι μάχες που επιτρέπονται είναι: melee vs melee, melee vs ranged, ranged vs ranged.

Επίσης υπάρχει σύστημα αντίστασης ζημιάς που απορροφάει ένα μέρος της ζημιάς. Κάθε εχθρός και χαρακτήρας έχει ζωή, που είναι ένα νούμερο το οποίο όταν πάει στο 0 ο χαρακτήρας πεθαίνει. Αυτό καθορίζεται από τη ζημιά που έχει ο αντίπαλός του.

3.3.3 Περιοχές

Ο χάρτης αποτελείται από τέσσερις περιοχές οι οποίες είναι αυξανόμενης δυσκολίας και η κάθε μία έχει διαφορετικό περιβάλλον και εχθρούς. Οι περιοχές είναι:

- Μπλε : Πρώτο επίπεδο με εχθρούς και εξοικείωση με τις ικανότητες του χαρακτήρα

- Πράσινη : Περιοχή που απαιτεί πιο συντονισμένη συνεργασία λόγω των μαζικών επιθέσεων των εχθρών

- Καφέ : Περιοχή όπου οι εχθροί έχουν υψηλότερα στατιστικά και πιο δυνατές επιθέσεις.

- Κοκκίνη : Τελική περιοχή με το τελικό boss που απαιτεί πλήρη συνεργασία και χρήση όλων των ικανοτήτων των χαρακτήρων

Οι περιοχές είναι αυξανόμενης δυσκολίας και πρέπει οι παίκτες να κατακτήσουν την περιοχή που βρίσκονται ώστε να προχωρίσουν στην επόμενη.

3.3.4 Εχθροί

Οι εχθροί χωρίζονται σε απλούς και πιο δυνατούς (bosses). Κάθε περιοχή έχει απλούς εχθρούς ή/και bosses. Οι περιοχές μπορεί να έχουν από κανένα εως και πολλά bosses. Οι εχθροί είναι διαφορετικοί ανάλογα με την περιοχή.

- Μπλε: Orcs, Magic Goblins, Wolves

- Πράσινη: Orcs, Magic Goblins, Wolves, High Orcs, Big Wolves, High Magic Goblins, εχθροί με καλύτερα χαρακτηριστικά

- Καφέ: High Orcs, High Magic Goblins, Big Wolves, Dragon, Elite Orcs, King Wolves, Elite Magic Goblins, εχθροί που κάνουν μεγαλύτερη ζημιά

- Κόκκινη: Dragons, Elite Orcs, King Wolves, Elite Goblins, μαζικές επιθέσεις εχθρών που κάνουν πολύ μεγάλη ζημιά

  Κάθε ένας από αυτούς τους εχθρούς έχει τη δυνατότητα να είναι παράλληλα και boss με κάποια στατιστικά τους να είναι αυξημένα.

3.3.4.1 Χαρακτηριστικά Εχθρών

Κάθε εχθροί έχουν διαφορετικά χαρακτηριστικά:

- Orcs: είναι αργοί στην κίνηση αλλά έχουν μεγάλη ζημιά (melee)

- High Orcs: επιπλέον από τα χαρακτηριστικά των απλών Orc, έχουν μεσσαία αντίσταση ζημιάς

- Elite Orcs: επιπλέον από τα χαρακτηριστικά των απλών Orc, έχουν μεγάλη αντίσταση ζημιάς

- Magic Goblins: έχουν μεγάλη εμβέλεια επίθεσης (ranged)

- High Magic Goblins: επιπλέον από τα χαρακτηριστικά των απλών Goblin, έχουν μεσσαία ταχύτητα επίθεσης

- Elite Magic Goblins: επιπλέον από τα χαρακτηριστικά των απλών Goblins, έχουν μεγάλη ταχύτητα επίθεσης

- Wolves: έχουν μεγάλη ταχύτητα κίνησης

- Big Wolves

- King Wolves

- Dragons: επιτίθονται από πολύ μακριά και η επίθεση τους έχει μεγάλη ακτίνα και μεγάλη ζημιά

3.3.4.2 Ενδεικτικά Στατιστικά Εχθρών

Τα στατιστικά των εχθρών είναι:

Table 1: Πίνακας στατιστικών εχθρών

Εχθρός	Ζωή	Ζημιά	Αντίσταση Ζημιάς	Ταχύτητα Επίθεσης	Ταχύτητα Κίνησης	Εμβέλεια Επίθεσης
Orc	120	15	10	1.0 /sec	3.2	1 tile
Goblin	90	8	4	1.2 /sec	3.8	3 tiles
Wolf	80	10	7	1.2 /sec	4.0	1 tile
High Orc	160	25	15	1.0 /sec	3.2	1 tile
High Goblin	110	12	7	1.4 /sec	3.8	3 tiles
Big Wolf	130	15	12	1.2 /sec	4.5	1 tile

Εχθρός	Ζωή	Ζημιά	Αντίσταση Ζημιάς	Ταχύτητα Επίθεσης	Ταχύτητα Κίνησης	Εμβέλεια Επίθεσης
Elite Orc	230	40	50	1.0 /sec	2.5	1 tile
Elite Goblin	150	15	10	1.6 /sec	3.8	3 tiles
King Wolf	200	18	15	1.2 /sec	4.5	1 tile
Dragon	350	50	25	0.8 /sec	3.4	8 tiles
Boss	+20%	+20%	+15%	Ιδια	Ιδια	Ιδια

Ο παραπάνω πίνακας παρουσιάζει τα βασικά στατιστικά των εχθρών στο παιχνίδι. Κάθε εχθρός διαθέτει συγκεκριμένες τιμές για Ζωή, Ζημιά, Αντίσταση Ζημιάς, Ταχύτητα Επίθεσης, Ταχύτητα Κίνησης και Εμβέλεια Επίθεσης. Τα στατιστικά αυτά καθορίζουν τη δυσκολία του εχθρού, τον τρόπο με τον οποίο συμπεριφέρεται στη μάχη και τον ρόλο του μέσα στο παιχνίδι.

- **Ζωή:** Δείχνει πόση ζημιά μπορεί να αντέξει ο εχθρός πριν εξουδετερωθεί. Όσο μεγαλύτερη η τιμή, τόσο πιο ανθεκτικός είναι.

- **Ζημιά:** Η ποσότητα ζημιάς που προκαλεί ο εχθρός σε κάθε επίθεση.

- **Αντίσταση Ζημιάς:** Μειώνει ένα ποσοστό της εισερχόμενης ζημιάς. Υψηλότερη θωράκιση σημαίνει ότι ο εχθρός δέχεται λιγότερη πραγματική ζημιά.

- **Ταχύτητα Επίθεσης:** Πόσο συχνά μπορεί να επιτεθεί ο εχθρός. Εκφράζεται συνήθως ως επιθέσεις ανά δευτερόλεπτο.

- **Ταχύτητα Κίνησης:** Η ταχύτητα με την οποία μετακινείται ο εχθρός στον χάρτη. Υψηλότερες τιμές σημαίνουν ταχύτερη προσέγγιση του στόχου.

- **Εμβέλεια Επίθεσης:** Η απόσταση από την οποία ο εχθρός μπορεί να επιτεθεί. Εκφράζεται σε tiles (κελιά του χάρτη).

Γενική Ανάλυση Εχθρών:

Orc, Goblin, Wolf

Βασικοί εχθροί.

High Orc, High Goblin, Big Wolf

Ενισχυμένες εκδόσεις των βασικών εχθρών, με αυξημένες τιμές σε ζωή, ζημιά ή ταχύτητες.

Elite Orc, Elite Goblin, King Wolf

Ακόμη πιο ισχυροί εχθροί με σημαντική αύξηση στη αντίσταση ζημιάς και τη ζημιά.

Dragon

Ισχυρός εχθρός με μεγάλη ζωή, υψηλή ζημιά και πολύ μεγάλη εμβέλεια (8 tiles).

Boss

Το Boss δεν αποτελεί νέο εχθρό, αλλά εχθρό με καλύτερα στατιστικά.
Οποιοσδήποτε εχθρός οριστεί ως Boss λαμβάνει:

- **+20% Ζωή**
- **+20% Ζημιά**
- **+15% Αντίσταση Ζημιάς**
- Ίδια ταχύτητα επίθεσης, κίνησης και εμβέλεια

Μετατρέπει δηλαδή οποιονδήποτε αντίπαλο σε πολύ πιο απαιτητικό.

Πλεονεκτήματα και Μειονεκτήματα Εχθρών

Κάθε εχθρός στο παιχνίδι είναι σχεδιασμένος με συγκεκριμένα πλεονεκτήματα αλλά και αντίστοιχα μειονεκτήματα, ώστε να διατηρείται η ισορροπία της μάχης και να ενθαρρύνονται διαφορετικές τακτικές. Για παράδειγμα, εχθροί με υψηλή ζωή μπορεί να έχουν χαμηλότερη ταχύτητα, ενώ όσοι έχουν μεγάλη εμβέλεια επίθεσης συνήθως έχουν μικρότερη αντίσταση ζημιάς. Με αυτόν τον τρόπο, κανένας εχθρός δεν είναι απόλυτα ισχυρός σε όλους τους τομείς και κάθε ένας αποτελεί ξεχωριστική πρόκληση που απαιτεί διαφορετική στρατηγική προσέγγιση από τον παίκτη.

3.3.5 Χαρακτήρες

Αποτελούνται από 3 κλάσεις: Πολεμιστής, Μάγος, Σκοπευτής. Η κάθε κλάση έχει διαφορετικά χαρακτηριστικά.

Ο Πολεμιστής έχει μικρή εμβέλεια επίθεσης αλλά με μεγάλη ζημιά στη βασική επίθεση. Επίσης έχει περισσότερη ταχύτητα κίνησης από τις άλλες δύο κλάσεις.

Ο Μάγος εχει μεγαλύτερη εμβέλεια από τον Πολεμιστή και κάνει μεγάλη ζημιά με τις ικανότητες.

Ο Σκοπευτής έχει αυξημένη ταχύτητα επίθεσης σε σχέση με τις άλλες κλάσεις, όπως και εμβέλεια επίθεσης.

3.3.6 Ικανότητες

Κάθε χαρακτήρας διαθέτει τρεις βασικές ικανότητες, οι οποίες διαφοροποιούνται ανάλογα με την κλάση του. Οι ικανότητες αυτές περιλαμβάνουν μία βασική επίθεση και δύο ειδικές ικανότητες με χρόνο επαναφόρτισης. Η βασική επίθεση αποτελεί την κύρια μορφή επίθεσης του χαρακτήρα και μπορεί να χρησιμοποιείται συχνά, χωρίς να απαιτεί κατανάλωση ενέργειας ή χρόνο επαναφόρτισης.

Η βασική επίθεση διαφέρει ανάλογα με τον τύπο της κλάσης. Αν ο χαρακτήρας είναι ranged, δηλαδή επιτίθεται από απόσταση, μπορεί να προκαλέσει ζημιά σε εχθρούς που βρίσκονται μέσα στην εμβέλειά του. Αντίθετα, αν ο χαρακτήρας είναι melee, δηλαδή επιτίθεται από κοντινή απόσταση, πρέπει να βρίσκεται κοντά στον εχθρό για να του προκαλέσει ζημιά.

Εκτός από τη βασική επίθεση, κάθε χαρακτήρας μπορεί να χρησιμοποιήσει δύο ειδικές ικανότητες. Για τη χρήση αυτών των ικανοτήτων απαιτείται κατανάλωση ενέργειας. Μετά την ενεργοποίησή τους, οι ικανότητες δεν μπορούν να χρησιμοποιηθούν αμέσως ξανά, καθώς χρειάζονται κάποιο χρονικό διάστημα μέχρι να επαναφορτιστούν. Το χρονικό αυτό διάστημα ονομάζεται χρόνος επαναφόρτισης ή cooldown (CD).

Σε περίπτωση που ο παίκτης ενεργοποιήσει μία ειδική ικανότητα ενώ χρησιμοποιεί τη βασική επίθεση, η βασική επίθεση διακόπτεται ή καθυστερεί μέχρι να ολοκληρωθεί η ενέργεια που βρίσκεται ήδη σε εξέλιξη. Στη συνέχεια εκτελείται η αντίστοιχη ικανότητα που επέλεξε ο παίκτης. Με αυτόν τον τρόπο, η χρήση των ειδικών ικανοτήτων απαιτεί σωστό χρονισμό και στρατηγική.

Η πρώτη ειδική ικανότητα ξεκλειδώνεται στο επίπεδο 3. Έχει μικρότερο χρόνο επαναφόρτισης, επομένως μπορεί να χρησιμοποιείται πιο συχνά, αλλά προκαλεί μικρότερη ζημιά στους εχθρούς. Η δεύτερη ειδική ικανότητα, η οποία μπορεί να θεωρηθεί ως υπέρτατη ικανότητα, ξεκλειδώνεται στο επίπεδο 5. Διαθέτει μεγαλύτερο χρόνο επαναφόρτισης, όμως προκαλεί μεγαλύτερη ζημιά και έχει ισχυρότερο αποτέλεσμα στη μάχη.

3.3.6.1 Ενδεικτικές Ικανότητες Πολεμιστή

Ο Πολεμιστής αποτελεί κλάση κοντινής μάχης και βασίζεται σε επιθέσεις μικρής εμβέλειας, υψηλής αντοχής και άμεσης επαφής με τους εχθρούς. Οι ικανότητές του είναι σχεδιασμένες ώστε να προκαλούν ζημιά σε κοντινούς στόχους, είτε με απλές επιθέσεις είτε με ισχυρότερες ειδικές ικανότητες που διαθέτουν χρόνο επαναφόρτισης. Η βασική επίθεση μπορεί να χρησιμοποιείται συνεχόμενα, ενώ οι δύο ειδικές ικανότητες ξεκλειδώνονται σταδιακά μέσω του συστήματος επιπέδων.

Table 2: Πίνακας ικανοτήτων πολεμιστή

Ικανότητα	Τύπος	Ζημιά	CD	Ενέργεια	Εμβέλεια
Βασική επίθεση	Απλή επίθεση	40	0	0	1 tile
Ειδική ικανότητα 1	Απλή ικανότητα	45	20sec	20	2 tiles
Ειδική ικανότητα 2	Υπέρτατη ικανότητα	35	15sec	50	3 tiles

Η βασική επίθεση του Πολεμιστή αποτελεί την κύρια μορφή πρόκλησης ζημιάς και χρησιμοποιείται χωρίς κατανάλωση ενέργειας ή χρόνο επαναφόρτισης. Επειδή ο Πολεμιστής είναι χαρακτήρας κοντινής μάχης, η βασική επίθεση έχει μικρή εμβέλεια και απαιτεί ο εχθρός να βρίσκεται κοντά στον χαρακτήρα.

Η πρώτη ειδική ικανότητα ξεκλειδώνεται στο επίπεδο 3 και προκαλεί μεγαλύτερη ζημιά από τη βασική επίθεση. Διαθέτει μικρότερο χρόνο επαναφόρτισης σε σχέση με την υπέρτατη ικανότητα, επομένως μπορεί να χρησιμοποιείται πιο συχνά κατά τη διάρκεια της μάχης. Η δεύτερη ειδική ικανότητα ξεκλειδώνεται στο επίπεδο 5 και λειτουργεί ως υπέρτατη ικανότητα. Έχει μεγαλύτερο χρόνο επαναφόρτισης, αλλά προκαλεί υψηλότερη ζημιά ή επηρεάζει μεγαλύτερη περιοχή γύρω από τον χαρακτήρα.

3.3.6.2 Ενδεικτικές Ικανότητες Μάγου

Ο Μάγος αποτελεί κλάση κοντινής μάχης, όπως και ο Πολεμιστής, όμως διαφοροποιείται ως προς τον τρόπο χρήσης των ικανοτήτων του. Η βασική του επίθεση εκτελείται σε μικρή απόσταση και χρησιμοποιείται για σταθερή πρόκληση ζημιάς στους εχθρούς. Ωστόσο, οι ειδικές ικανότητες του Μάγου έχουν μεγαλύτερη εμβέλεια από αυτές του Πολεμιστή, επιτρέποντάς του να επηρεάζει εχθρούς που βρίσκονται σε μεγαλύτερη απόσταση ή σε ευρύτερη περιοχή.

Table 3: Πίνακας ικανοτήτων μάγου

Ικανότητα	Τύπος	Ζημιά	CD	Ενέργεια	Εμβέλεια
Βασική επίθεση	Απλή επίθεση	15	0	0	1 tile
Ειδική ικανότητα 1	Απλή ικανότητα	40	22s	30	3 tiles
Ειδική ικανότητα 2	Υπέρτατη ικανότητα	65	25s	60	5 tiles

Η πρώτη ειδική ικανότητα του Μάγου ξεκλειδώνεται στο επίπεδο 3 και προκαλεί μεγαλύτερη ζημιά από τη βασική επίθεση. Διαθέτει χρόνο επαναφόρτισης και απαιτεί κατανάλωση ενέργειας, επομένως δεν μπορεί να χρησιμοποιείται συνεχόμενα. Η δεύτερη ειδική ικανότητα λειτουργεί ως υπέρτατη ικανότητα και έχει ακόμη μεγαλύτερη εμβέλεια και ισχυρότερο αποτέλεσμα στη μάχη, αλλά απαιτεί περισσότερη ενέργεια και μεγαλύτερο χρόνο επαναφόρτισης.

Σε αντίθεση με τον Πολεμιστή, ο Μάγος βασίζεται περισσότερο στη σωστή χρήση των ειδικών ικανοτήτων του, καθώς αυτές του επιτρέπουν να προκαλεί ζημιά σε μεγαλύτερη απόσταση και να ελέγχει καλύτερα τη θέση των εχθρών μέσα στη μάχη. Για τον λόγο αυτό, ο παίκτης πρέπει να διαχειρίζεται προσεκτικά την ενέργεια και τους χρόνους επαναφόρτισης, ώστε να χρησιμοποιεί τις ικανότητες την κατάλληλη στιγμή.

3.3.6.3 Ενδεικτικές Ικανότητες Σκοπευτή

Ο Σκοπευτής αποτελεί κλάση που βασίζεται κυρίως σε επιθέσεις από απόσταση, σε αντίθεση με τον Πολεμιστή και τον Μάγο που χρησιμοποιούν επιθέσεις κοντινής μάχης. Η βασική του επίθεση έχει μεγαλύτερη εμβέλεια, επιτρέποντάς του να προκαλεί ζημιά στους εχθρούς πριν αυτοί πλησιάσουν αρκετά. Με αυτόν τον τρόπο, ο Σκοπευτής μπορεί να παραμένει σε πιο ασφαλή θέση κατά τη διάρκεια της μάχης, αξιοποιώντας την απόσταση προς όφελός του.

Table 4: Πίνακας ικανοτήτων σκοπευτή

Ικανότητα	Τύπος	Ζημιά	CD	Ενέργεια	Εμβέλεια
Βασική επίθεση	Απλή επίθεση	15	0	0	3 tiles
Ειδική ικανότητα 1	Απλή ικανότητα (buff)	0	20sec	30	-

Ικανότητα	Τύπος	Ζημιά	CD	Ενέργεια	Εμβέλεια
Ειδική ικανότητα 2	Υπέρτατη ικανότητα	20	15sec	60	3 tiles

Η πρώτη ειδική ικανότητα του Σκοπευτή λειτουργεί ως ενίσχυση (buff) και δεν προκαλεί άμεση ζημιά στους εχθρούς. Αντίθετα, προσφέρει προσωρινή βελτίωση στον χαρακτήρα, όπως αύξηση της ταχύτητας ή της αποτελεσματικότητάς του στη μάχη. Επειδή δεν είναι επιθετική ικανότητα, δεν διαθέτει εμβέλεια, αλλά απαιτεί ενέργεια και έχει χρόνο επαναφόρτισης.

Η δεύτερη ειδική ικανότητα λειτουργεί ως υπέρτατη ικανότητα και έχει τη μορφή γρήγορης μετακίνησης (dash). Με τη χρήση της, ο Σκοπευτής μετακινείται άμεσα προς την κατεύθυνση στην οποία είναι στραμμένος, επιτρέποντάς του να αποφεύγει επιθέσεις, να απομακρύνεται από εχθρούς ή να αλλάζει γρήγορα θέση μέσα στη μάχη. Η ικανότητα αυτή δεν βασίζεται κυρίως στην πρόκληση ζημιάς, αλλά στην ευελιξία και στην επιβίωση του χαρακτήρα.

3.3.7 Επιβραβεύσεις

Ο παίκτης που σκοτώνει έναν απλό εχθρό παίρνει χρυσό και μόνο αυτός, ανάλογα με τη δυσκολία του εχθρού. Ο εχθρός που πέθανε όμως δίνει εμπειρία, ανάλογα με τη δυσκολία του, σε όλους γύρω του για να ανέβουν στο επόμενο επίπεδο. Η εμπειρία διαιρείται ανάλογα με τους παίκτες που βρίσκονται κοντά (5 tiles). Σε κάθε επίπεδο αυξάνονται τα στατιστικά του χαρακτήρα ανάλογα με την κλάση του.

Όταν πεθαίνει ένα boss δίνει χρυσό σε όλους γύρω του, ανεξάρτητα από το ποιος το σκότωσε και ρίχνει επίσης κάποια σπάνια κουτιά, που περιέχουν περισσότερο χρυσό, σε όποιον προλάβει να τα πάρει (είναι πιο κοντά σε αυτό, 1 tile).

3.3.7.1 Ενδεικτικές Επιβραβεύσεις

Table 5: Πίνακας επιβραβεύσεων παίκτη

Εχθρός	Χρυσός	Εμπειρία
Orc	5	10
Goblin	4	8
Wolves	6	12

Εχθρός	Χρυσός	Εμπειρία
High Orc	10	20
High Goblin	8	15
King Wolves	12	25
Elite Orcs	15	30
Elite Goblins	14	28
Dragon	20	40
Boss / Τελικός Εχθρός	+50	+30

Ο πίνακας παρουσιάζει τις ανταμοιβές που λαμβάνει ο παίκτης για την εξουδετέρωση των διαφόρων εχθρών. Οι ανταμοιβές χωρίζονται κυρίως σε χρυσό και εμπειρία, οι οποίες αυξάνονται ανάλογα με τη δύναμη και τη δυσκολία του αντίπαλου.

- Χρυσός (Gold): Χρησιμοποιείται για αγορά αντικειμένων μέσα στο παιχνίδι. Οι ισχυρότεροι εχθροί προσφέρουν περισσότερο χρυσό.

- Εμπειρία (XP): Αυξάνει το επίπεδο του παίκτη, ενισχύοντας στατιστικά και ξεκλειδώνοντας νέες ικανότητες. Οι πιο επικίνδυνοι ή σπάνιοι εχθροί δίνουν μεγαλύτερη εμπειρία.

Τα Boss προσφέρουν επιπλέον ανταμοιβές πέρα από χρυσό και εμπειρία, όπως:

- +50 Χρυσός επιβραβεύοντας τον παίκτη για τη δυσκολία της μάχης.

- +30 Εμπειρία, για να αντανακλά τη μεγαλύτερη πρόκληση.

Αυτό το σύστημα ανταμοιβών ενθαρρύνει τον παίκτη να αντιμετωπίζει πιο δύσκολους εχθρούς και προσφέρει κίνητρο για προοδευτική ενίσχυση του χαρακτήρα.

3.3.8 Ενδεικτικό Σύστημα Επιπέδων

Ο χαρακτήρας, με την εμπειρία που κερδίζει όταν σκοτώνει εχθρούς, ανεβαίνει επίπεδο ξεκινώντας από το επίπεδο 1 και φτάνοντας έως το επίπεδο 10. Σε κάθε επίπεδο ανεβαίνουν τα στατιστικά του, τα οποία είναι διαφορετικά ανά κλάση.

Table 6: Πίνακας επιπέδων χαρακτήρα

Επίπεδο	Απαιτούμενη Εμπειρία
1	0
2	50
3	120
4	200
5	300
6	450
7	650
8	900
9	1200
10	1600

Με την αύξηση επιπέδου, οι χαρακτήρες γίνονται ισχυρότεροι, αποκτώντας μεγαλύτερη ζωή, ενέργεια, αντίσταση ζημιάς και ταχύτητα. Αυτό το σύστημα δημιουργεί κίνητρο για συνεχή μάχη και προοδευτική ενίσχυση του παίκτη.

3.3.8.1 Ενδεικτικά Στατιστικά κλάσης κάθε επιπέδου

Πολεμιστής:

Table 7: Πίνακας στατιστικών πολεμιστή

Επίπεδο	Ζωή	Ενέργεια	Αντίσταση Ζημιάς	Ταχύτητα Κίνησης
1	120	40	15	4
2	140	45	19	4
3	160	50	23	4

Επίπεδο	Ζωή	Ενέργεια	Αντίσταση Ζημιάς	Ταχύτητα Κίνησης
4	180	55	27	4
5	200	60	31	4
6	220	65	35	4
7	240	70	39	4
8	260	75	43	4
9	280	80	47	4
10	300	90	51	4

Ο Πολεμιστής εστιάζει κυρίως στη ζωή και την αντίσταση στη ζημιά, καθώς βρίσκεται στην πρώτη γραμμή της μάχης και απορροφά τις επιθέσεις των εχθρών, προστατεύοντας τους υπόλοιπους συμμάχους.

Μάγος:

Table 8: Πίνακας στατιστικών μάγου

Επίπεδο	Ζωή	Ενέργεια	Αντίσταση Ζημιάς	Ταχύτητα Κίνησης
1	80	65	8	3.4
2	90	80	11	3.4
3	100	95	14	3.4
4	110	110	17	3.4
5	120	125	20	3.4
6	135	140	23	3.4
7	150	155	26	3.4

Επίπεδο	Ζωή	Ενέργεια	Αντίσταση Ζημιάς	Ταχύτητα Κίνησης
8	165	170	29	3.4
9	180	185	32	3.4
10	200	200	35	3.4

Ο Μάγος εστιάζει κυρίως στην ενέργεια, η οποία χρησιμοποιείται για τις ικανότητές του, ενώ η ζωή και η αντίσταση στη ζημιά είναι χαμηλότερες, καθώς παραμένει πίσω από τη γραμμή μάχης και βασίζεται στις μαγικές του επιθέσεις για να προκαλέσει ζημιά.

Σκοπευτής:

Table 9: Πίνακας στατιστικών σκοπευτή

Επίπεδο	Ζωή	Ενέργεια	Αντίσταση Ζημιάς	Ταχύτητα Κίνησης
1	90	50	7	3.2
2	100	60	9	3.2
3	115	70	11	3.2
4	130	80	13	3.2
5	145	90	15	3.2
6	160	100	17	3.2
7	175	110	19	3.2
8	190	120	21	3.2
9	205	130	23	3.2
10	220	140	25	3.2

Ο Σκοπευτής είναι ευάλωτος σε ζημιά και έχει μικρή ταχύτητα κίνησης, επειδή αποτελεί την κύρια πηγή ζημιάς από απόσταση και δεν βασίζεται στις ικανότητές του. Η μεγάλη δύναμη που έχει στην επίθεση εξισορροπείται από αυτές τις αδυναμίες, ώστε να διατηρείται η ισορροπία στο παιχνίδι.

3.3.8.2 Ενδεικτικά Στατιστικά ικανοτήτων κάθε επιπέδου

Πολεμιστή:

Table 10: Πίνακας στατιστικών ικανοτήτων πολεμιστή

Επίπεδο	Ζημιά Βασικής Επίθεσης (+6.2%)	Ζημιά Ειδικής Ικανότητας 1 (+5.1%)	Ζημιά Ειδικής Ικανότητας 2 (+8%)
1	25	-	-
2	27	-	-
3	28	33	-
4	30	35	-
5	32	37	27
6	34	39	29
7	36	40	32
8	38	43	34
9	40	45	37
10	43	47	40

Οι ικανότητες του Πολεμιστή ενισχύονται σταδιακά καθώς ο χαρακτήρας ανεβαίνει επίπεδο. Κάθε ικανότητα έχει διαφορετικό ποσοστό αύξησης, ανάλογα με τη σημαντικότητα και τη χρησιμότητά της στη μάχη.

Μάγου:

Table 11: Πίνακας στατιστικών ικανοτήτων μάγου

Επίπεδο	Ζημιά Βασικής Επίθεσης (+7.7%)	Ζημιά Ειδικής Ικανότητας 1 (+9.2%)	Ζημιά Ειδικής Ικανότητας 2 (+7.1%)
1	18	-	-
2	19	-	-
3	21	18	-
4	22	20	-
5	24	21	26
6	26	23	28
7	28	25	30
8	30	28	32
9	33	30	35
10	35	33	37

Οι ικανότητες του Μάγου ενισχύονται σταδιακά καθώς ο χαρακτήρας ανεβαίνει επίπεδο. Κάθε ικανότητα έχει διαφορετικό ποσοστό αύξησης, ανάλογα με τη σημαντικότητα και τη χρησιμότητά της στη μάχη.

Σκοπευτή:

Table 12: Πίνακας στατιστικών ικανοτήτων σκοπευτή

Επίπεδο	Ζημιά Βασικής Επίθεσης (+5.3%)	Αύξηση ταχύτητας επίθεσης (Ικανότητα 1)
1	24	-
2	25	-
3	27	1.1%
4	28	1.15%

Επίπεδο	Ζημιά Βασικής Επίθεσης (+5.3%)	Αύξηση ταχύτητας επίθεσης (Ικανότητα 1)
5	29	1.2%
6	31	1.25%
7	33	1.3%
8	34	1.35%
9	36	1.4%
10	38	1.5%

Οι ικανότητες του Σκοπευτή ενισχύονται σταδιακά καθώς ο χαρακτήρας ανεβαίνει επίπεδο. Κάθε ικανότητα έχει διαφορετικό ποσοστό αύξησης, ανάλογα με τη σημαντικότητα και τη χρησιμότητά της στη μάχη.

3.3.9 Inventory/Shop

Με το χρυσό που έχει ο κάθε παίκτης μπορεί να μπει στο inventory του και να αγοράσει φίλτρα (potions), ελιξήρια (elixirs). Υπάρχουν δύο potions και 3 elixirs.

3.3.9.1 Φίλτρα

- Φίλτρο ζωής: αναπλήρωση χαμένης ζωής (+20)
- Φίλτρο ενέργειας: αναπλήρωση χαμένης ενέργειας (+25)

Η χρήση του φίλτρου δεν έχει διάρκεια εκτέλεσης, δηλαδή γίνεται άμεσα, και κάθε παίκτης μπορεί να κουβαλάει έως δύο ταυτόχρονα πριν τα χρησιμοποιήσει.

3.3.9.2 Ελιξήρια

Τα ελιξήρια δίνουν επιπλέον στατιστικά για ένα μικρό χρονικό διάστημα.

- Ελιξήριο Πολεμιστή: Δίνει επιπλέον ζημιά και ζωή για ένα λεπτό
- Ελιξήριο Μάγου: Δίνει επιπλέον ζημιά και ενέργεια για ένα λεπτό
- Ελιξήριο Σκοπευτή: Δίνει επιπλέον ζημιά και ταχύτητα επίθεσης για ένα λεπτό

3.3.9.3 Ενδεικτικά Κόστη Αντικειμένων

Table 13: Πίνακας με τα κόστη των αντικειμένων

Αντικείμενο	Τύπος	Εφέ	Διάρκεια	Όριο	Κόστος (χρυσός)
Φίλτρο Ζωής	Φίλτρο	+20 Ζωή	Άμεσο	2/παίκτη	50
Φίλτρο Ενέργειας	Φίλτρο	+25 Ενέργεια	Άμεσο	2/παίκτη	50
Ελιξήριο Πολεμιστή	Ελιξήριο	+20 Ζημιά, +30 Ζωή	1 λεπτό	1/παίκτη	100
Ελιξήριο Μάγου	Ελιξήριο	+15 Ζημιά, +50 Ενέργεια	1 λεπτό	1/παίκτη	100
Ελιξήριο Σκοπευτή	Ελιξήριο	+15 Ζημιά, +30% Ταχύτητα Επίθεσης	1 λεπτό	1/παίκτη	100

Ο πίνακας παρουσιάζει τα κόστη και τις επιδράσεις των βασικών αντικειμένων στο παιχνίδι. Τα αντικείμενα χωρίζονται σε φίλτρα και ελιξήρια:

- Φίλτρα Ζωής / Ενέργειας: Παρέχουν άμεση αύξηση ζωής ή ενέργειας, με περιορισμό χρήσης ανά παίκτη.

- Ελιξήρια: Αυξάνουν προσωρινά στατιστικά όπως ζημιά, ζωή ή ταχύτητα επίθεσης για 1 λεπτό.

Οι τιμές σε χρυσό καθορίζουν το κόστος απόκτησης ή αναβάθμισης, δίνοντας στον παίκτη τη δυνατότητα στρατηγικής διαχείρισης πόρων.

3.3.10 Επαναφορά/Επιστροφή

Κάθε παίκτης που πεθαίνει επιστρέφει στην αρχή της προηγούμενης περιοχής από εκεί που βρίσκονται όλοι, όμως κάποιοι εχθροί έχουν επανέλθει σε αυτήν και θα πρέπει να τους περάσει αλλιώς θα τον ακολουθήσουν μέχρι εκεί που θα πάει.

3.4 Αποστολές (Quests)

Υπάρχουν οι βασικές αποστολές ώστε να μπορεί να προχωρήσει το παιχνίδι τύπου:

- Σκότωσε έναν (τυχαίο) αριθμό από συγκεκριμένο είδος εχθρού

- Σκότωσε το boss της περιοχής

Υπάρχουν και κάποιες προαιρετικές αποστολές που μπορεί ένας παίκτης να πάρει επιπλέον χρυσό ή κάποιος που δεν έχει πολύ, λόγω των λίγων εξοντώσεων. Αυτά δεν είναι υποχρεωτικά ώστε να συνεχίσει το παιχνίδι, είναι απλά βοήθημα στους παίκτες. Αυτά είναι του τύπου:

- Σκότωσε έναν τυχαίο αριθμό εχθρών ως μια συγκεκριμένη κλάση

3.5 Επικοινωνία

Οι παίκτες δεν έχουν τη δυνατότητα να επικοινωνούν μεταξύ τους, αλλά μπορούν να το κάνουν με διάφορους τρόπους χρησιμοποιώντας άλλες πλατφόρμες όπως το Discord.

3.6 Server

Η σύνδεση των παικτών θα γίνεται σε ένα δωμάτιο ώστε να εισέλθουν μαζί στο παιχνίδι. Σε περίπτωση αποσύνδεσης κάποιου παίκτη το παιχνίδι συνεχίζεται κανονικά και ο χαρακτήρας του παραμένει στο παιχνίδι για δύο λεπτά σε περίπτωση που επιστρέψει αλλιώς πεθαίνει.

3.7 Στοχευμένο κοινό

Το Celestial Lands προορίζεται για όλες τις ηλικίες εφόσον είναι εξοικειωμένοι τα με ψηφιακά παιχνίδια και πιο συγκεκριμένα παιχνίδια που απαιτούν συνεργασία αλλά και προσωπική πρόοδο.

3.8 Διαθέσιμες πλατφόρμες

Το παιχνίδι διατίθεται μόνο για την πλατφόρμα ηλεκτρονικού υπολογιστή και το λειτουργικό σύστημα Microsoft Windows 11.

3.9 LAN Δίκτυο

Ο server θα λειτουργεί σε LAN (Local Area Network) και θα είναι προσβάσιμος μέσω direct IP. Αυτό σημαίνει ότι οι υπολογιστές που βρίσκονται στο ίδιο τοπικό δίκτυο θα μπορούν να συνδεθούν απευθείας χρησιμοποιώντας τη διεύθυνση IP του server.

3.10 Ήχοι και Μουσική

Το παιχνίδι θα περιλαμβάνει πλήρη ηχητικό περιβάλλον με μουσική υπόκρουση και ηχητικά εφέ. Η μουσική θα δημιουργεί την ατμόσφαιρα κάθε περιοχής ή μάχης, ενισχύοντας την εμπειρία του παίκτη. Παράλληλα, κάθε χαρακτήρας και εχθρός θα έχει ξεχωριστά ηχητικά εφέ για τις βασικές επιθέσεις και τις ικανότητες, ενώ ήχοι περιβάλλοντος θα προσθέτουν ρεαλισμό και αίσθηση παρουσίας στον κόσμο του παιχνιδιού.

3.11 Γραφικά

Τα γραφικά του παιχνιδιού θα είναι σχεδιασμένα, ώστε κάθε χαρακτήρας, εχθρός και αντικείμενο να είναι εύκολα αναγνωρίσιμο. Το περιβάλλον θα έχει σαφή οπτική ταυτότητα με χρωματικούς κώδικες που βοηθούν στην κατανόηση των λειτουργιών και της σημασίας των αντικειμένων. Οι χαρακτήρες θα διαθέτουν διαφορετικά μοντέλα, ανάλογα με την κλάση τους, και η αισθητική θα συνδυάζει λειτουργικότητα και μάχη.

3.12 Animation

Κάθε χαρακτήρας και εχθρός θα διαθέτει φυσικά animations για τις κινήσεις, τις επιθέσεις και τις ικανότητές του. Οι ικανότητες θα τονίζουν τη δύναμη και την επίδρασή τους στο παιχνίδι. Τα animation θα είναι συγχρονισμένα με τους ήχους, δημιουργώντας μια πιο ρεαλιστική και δυναμική εμπειρία για τον παίκτη, ενώ παράλληλα θα προσδίδουν ρυθμό και οπτική ικανοποίηση στη μάχη και την εξερεύνηση του κόσμου.

Κεφάλαιο 4: Υλοποίηση

Στο παρόν κεφάλαιο παρουσιάζεται η υλοποίηση του ψηφιακού παιχνιδιού που αναπτύχθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας. Περιγράφονται τα βασικά λειτουργικά μέρη του παιχνιδιού και ο τρόπος με τον οποίο υλοποιήθηκαν, συγκεκριμένα για τα γραφικά, τον ήχο και τους μηχανισμούς gameplay, οι οποίοι περιλαμβάνουν εξυπηρετητή (server) και πελάτη (client).

4.1 Γραφικά

Για την υλοποίηση του γραφικού περιβάλλοντος του παιχνιδιού χρησιμοποιήθηκε το εργαλείο Tiled, για τη δημιουργία του χάρτη (map). Μέσω του Tiled σχεδιάστηκε ο δισδιάστατος χάρτης του παιχνιδιού, βασισμένος σε tilesets, τα οποία οργανώθηκαν σε επίπεδα (layers) για τη διαμόρφωση του περιβάλλοντος. Ο χάρτης περιλαμβάνει τόσο οπτικά στοιχεία του περιβάλλοντος όσο και περιοχές που χρησιμοποιούνται για τον ορισμό της κίνησης και των συγκρούσεων του παίκτη.

Για την οπτική αναπαράσταση του χαρακτήρα του παίκτη χρησιμοποιήθηκαν δισδιάστατα (2D) γραφικά, σχεδιασμένα για top-down απεικόνιση. Για την απόδοση της κίνησης του χαρακτήρα αξιοποιήθηκαν διαφορετικές απεικονίσεις του, ώστε η μετακίνηση να αποδίδεται οπτικά με φυσικό τρόπο.

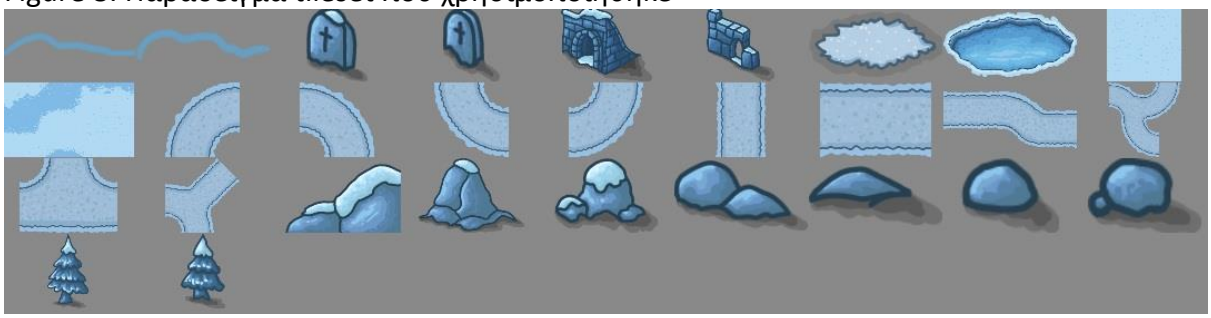
4.1.1 Υλοποίηση Χάρτη

Ο χάρτης του παιχνιδιού σχεδιάστηκε εξ ολοκλήρου από την αρχή, χωρίς τη χρήση έτοιμων ή προκαθορισμένων χαρτών. Η διάταξη των περιοχών, η τοποθέτηση των στοιχείων του περιβάλλοντος και ο διαχωρισμός των layers πραγματοποιήθηκαν με στόχο τη δημιουργία ενός λειτουργικού και συνεκτικού παιχνιδόκοσμου, προσαρμοσμένου στις ανάγκες του gameplay.

4.1.1.1 Δημιουργία περιοχών και layers χάρτη

Για την υλοποίηση των περιοχών του χάρτη συνδυάστηκαν διάφορα tilesets, τα οποία προέρχονται από την πλατφόρμα CraftPix και διατίθενται δωρεάν. Τα tilesets περιλαμβάνουν γραφικά στοιχεία όπως δέντρα, βράχους και άλλα στοιχεία περιβάλλοντος, τα οποία συνδυάστηκαν για τη δημιουργία των layers του χάρτη.

Figure 8: Παράδειγμα tileset που χρησιμοποιήθηκε



Συγκεκριμένα, ο χάρτης αποτελείται από τρία βασικά layers. Το layer του εδάφους (terrain) χρησιμοποιείται για την απεικόνιση της επιφάνειας πάνω στην οποία κινείται ο παίκτης και δεν περιλαμβάνει περιορισμούς στην κίνηση.

Το layer των τοίχων (walls) περιλαμβάνει στοιχεία με τα οποία ο παίκτης έρχεται σε σύγκρουση (collision), περιορίζοντας την κίνησή του στον χώρο. Στο layer αυτό προστέθηκε δυναμικά collision στα αντικείμενα, χρησιμοποιώντας το collision editor που παρέχει το Tiled. Μέσω του collision editor είναι δυνατός ο ορισμός συγκεκριμένων σχημάτων σύγκρουσης πάνω στα αντικείμενα, επιτρέποντας πιο ρεαλιστική αλληλεπίδραση. Για παράδειγμα, σε αντικείμενα όπως οι βράχοι, το collision εφαρμόζεται μόνο στο κάτω μέρος τους και όχι σε ολόκληρο το γραφικό στοιχείο, ώστε να αποφεύγονται μη ρεαλιστικοί περιορισμοί στην κίνηση του παίκτη.

Figure 9: Παράδειγμα collision editor σε βράχο

Το layer Road περιλαμβάνει δρόμους και μικρούς θάμνους. Τα στοιχεία αυτά χρησιμοποιούνται κυρίως για την οπτική διαμόρφωση του χάρτη και δεν επηρεάζουν την κίνηση του παίκτη, καθώς δεν διαθέτουν collision. Επιπλέον, χρησιμοποιείται το layer Bridge, το οποίο αφορά τις γέφυρες του χάρτη. Οι γέφυρες επιτρέπουν στον παίκτη να διασχίζει περιοχές όπως ποτάμια ή λάβα, τα οποία διαφορετικά αντιμετωπίζονται ως εμπόδια και δεν μπορούν να περαστούν.

Figure 10: Υλοποίηση της πρώτης περιοχής του χάρτη

4.1.1.2 Σημεία εμφάνισης και μεταβάσεις μεταξύ περιοχών

Επιπλέον, χρησιμοποιήθηκε layer τύπου Object, στο οποίο ορίστηκαν σημεία εμφάνισης (spawn points) για τους παίκτες και τους εχθρούς. Τα σημεία αυτά υλοποιήθηκαν με τη χρήση δεικτών (pointers), οι οποίοι επιτρέπουν τον δυναμικό καθορισμό της αρχικής θέσης των οντοτήτων κατά την εκτέλεση του παιχνιδιού. Στο ίδιο layer ορίστηκαν και ειδικά ορθογώνια αντικείμενα που λειτουργούν ως περιοχές μετάβασης (transition areas) μεταξύ των διαφορετικών περιοχών. Κάθε τέτοιο αντικείμενο διαθέτει τις ιδιότητες target_map και target_spawn. Η ιδιότητα target_map δηλώνει το επόμενο region στο οποίο θα μεταφερθεί ο παίκτης, ενώ η ιδιότητα target_spawn δηλώνει το σημείο εμφάνισης στο νέο region. Όταν ο παίκτης εισέλθει στο ορθογώνιο transition, ο server τον μεταφέρει στην αντίστοιχη περιοχή και τον τοποθετεί στο spawn point που έχει οριστεί.

Για τη λειτουργία των μεταβάσεων μεταξύ περιοχών χρησιμοποιούνται βοηθητικές μέθοδοι στον server. Αρχικά, η μέθοδος get_region επιστρέφει το αντικείμενο της περιοχής με βάση το όνομά της. Με αυτόν τον τρόπο ο server μπορεί να ανακτά δυναμικά τα δεδομένα κάθε region, όπως τα transition objects και τα διαθέσιμα spawn points.

Figure 11: Μέθοδος ανάκτησης περιοχής με βάση το όνομα του region

```
# Επιστρέφει το αντικείμενο Region με βάση το όνομα της περιοχής
def get_region(region_name: str) -> Region:
    return regions[region_name]
```

Στη συνέχεια, η μέθοδος rect_contains_point ελέγχει αν ένα σημείο με συντεταγμένες (x, y) βρίσκεται μέσα σε ένα ορθογώνιο object. Η μέθοδος αυτή χρησιμοποιείται κυρίως για τον έλεγχο των transition rectangles που έχουν οριστεί στο Tiled. Αν η θέση του παίκτη βρίσκεται μέσα στα όρια ενός τέτοιου ορθογωνίου, τότε ενεργοποιείται η διαδικασία μετάβασης.

Figure 12: Έλεγχος θέσης παίκτη μέσα σε ορθογώνιο transition object

```
# Ελέγχει αν ένα σημείο x, y βρίσκεται μέσα σε ένα ορθογώνιο object, χρησιμοποιείται για τα transition
def rect_contains_point(rect, x, y):
    return (
        rect["x"] <= x <= rect["x"] + rect["width"] and
        rect["y"] <= y <= rect["y"] + rect["height"]
    )
```

Η βασική μέθοδος που πραγματοποιεί τη μετάβαση είναι η player_transition. Η μέθοδος αυτή ελέγχει όλα τα transition objects της τρέχουσας περιοχής του παίκτη. Όταν ο παίκτης εισέλθει σε ένα transition rectangle, ο server διαβάζει τις ιδιότητες target_map και target_spawn. Η ιδιότητα target_map δηλώνει το region στο οποίο θα μεταφερθεί ο παίκτης, ενώ η ιδιότητα target_spawn δηλώνει το σημείο εμφάνισης μέσα στο νέο region.

Figure 13: Μέθοδος μετάβασης παίκτη μεταξύ περιοχών μέσω transition objects

```python
# Ελέγχει αν ο παίκτης βρίσκεται μέσα σε κάποιο transition object και τον μεταφέρει στο αντίστοιχο region
def player_transition(player):
    current_region = get_region(player["region"])  # Παίρνουμε το region στο οποίο βρίσκεται αυτή τη στιγμή ο παίκτης

    # Ελέγχουμε όλα τα transition rectangles που έχουν οριστεί στο τρέχον region
    for tr in current_region.transitions:
        if rect_contains_point(tr, player["x"], player["y"]):  # Αν η θέση του παίκτη βρίσκεται μέσα στο ορθογώνιο transition
            # Παίρνουμε από τα properties του transition το region προορισμού και το spawn point στο οποίο πρέπει να εμφανιστεί ο παίκτης
            target_region_name = tr["target_map"]
            target_spawn_name = tr["target_spawn"]

            if target_region_name not in regions:  # Αν το target region δεν υπάρχει στο dictionary των regions, εμφανίζουμε μήνυμα σφάλματος και ακυρώνουμε τη μετάβαση
                print(f"Unknown target region: {target_region_name}")
                return

            target_region = get_region(target_region_name)                # Παίρνουμε το αντικείμενο του region προορισμού
            spawn_pos = target_region.get_named_spawn(target_spawn_name)  # Αναζητούμε το spawn point μέσα στο target region με βάση το όνομά του

            # Αν δεν βρεθεί spawn point με αυτό το όνομα, εμφανίζουμε μήνυμα σφάλματος και ακυρώνουμε τη μετάβαση
            if spawn_pos is None:
                print(f"Spawn '{target_spawn_name}' not found in region '{target_region_name}'")
                return

            player["region"] = target_region_name  # Ενημερώνουμε το region του παίκτη
            player["x"], player["y"] = spawn_pos   # Μεταφέρουμε τον παίκτη στη θέση του target spawn point

            print(f"Player {player['nickname']} transitioned to {target_region_name} at spawn {target_spawn_name}") # Μήνυμα επιβεβαίωσης για debugging
            return  # Σταματάμε μετά την πρώτη έγκυρη μετάβαση, ώστε να μη γίνει δεύτερο transition στο ίδιο tick
```

Με τον τρόπο αυτό, η μετάβαση μεταξύ περιοχών γίνεται δυναμικά, χωρίς να απαιτείται να είναι σταθερά γραμμένες οι συνδέσεις μέσα στον κώδικα για κάθε σημείο μετάβασης. Οι πληροφορίες ορίζονται απευθείας στο Tiled μέσω των properties των transition objects, ενώ ο server αναλαμβάνει να ελέγξει τη θέση του παίκτη και να τον μεταφέρει στο κατάλληλο region και spawn point.

4.1.1.3 Ταξινόμηση sprites και απεικόνιση βάθους

Για τη σωστή απεικόνιση των αντικειμένων στον χάρτη χρησιμοποιείται ταξινόμηση των sprites με βάση τη θέση τους στον άξονα Y. Η διαδικασία αυτή είναι απαραίτητη σε top-down παιχνίδια, ώστε ο παίκτης και οι εχθροί να εμφανίζονται μπροστά ή πίσω από αντικείμενα του περιβάλλοντος, ανάλογα με τη θέση τους. Για παράδειγμα, όταν ο παίκτης βρίσκεται πίσω από ένα δέντρο, το δέντρο πρέπει να εμφανίζεται μπροστά του, ενώ όταν βρίσκεται μπροστά από αυτό, ο παίκτης πρέπει να σχεδιάζεται πάνω από το δέντρο.

Για να λειτουργήσει σωστά η ταξινόμηση βάθους, τα αντικείμενα του χάρτη που επηρεάζουν την οπτική απεικόνιση, όπως τα walls, καθώς και τα sprites των παικτών και των εχθρών, τοποθετούνται στην ίδια λίστα σχεδίασης (actor_list). Με αυτόν τον τρόπο, όλα τα σχετικά sprites ταξινομούνται μαζί πριν σχεδιαστούν στην οθόνη. Αν τα walls και οι χαρακτήρες σχεδιάζονταν σε ξεχωριστές λίστες, δεν θα ήταν δυνατό να εναλλάσσεται σωστά η σειρά εμφάνισής τους, με αποτέλεσμα ο παίκτης να φαίνεται πάντα είτε μπροστά είτε πίσω από τα αντικείμενα, ανεξάρτητα από τη θέση του.

Για τον σκοπό αυτό χρησιμοποιείται η μέθοδος sort_key, η οποία επιστρέφει την τιμή με βάση την οποία γίνεται η ταξινόμηση των sprites. Για τα sprites παικτών και εχθρών χρησιμοποιείται η τιμή bottom, δηλαδή η κάτω πλευρά του sprite, ώστε η ταξινόμηση να βασίζεται στη θέση των ποδιών τους. Για τα υπόλοιπα αντικείμενα του χάρτη, όπως δέντρα ή διακοσμητικά στοιχεία, χρησιμοποιείται το center_y σε συνδυασμό με την ιδιότητα sort_offset, η οποία μπορεί να οριστεί μέσα από το Tiled.

Το sort_offset επιτρέπει τη μικρή προσαρμογή της σειράς σχεδίασης κάθε αντικειμένου, ώστε η απεικόνιση να φαίνεται πιο φυσική και να αποφεύγονται οπτικά λάθη σε αντικείμενα με μεγάλο ύψος ή ιδιαίτερο σχήμα.

Figure 14: Ταξινόμηση sprites με βάση το βάθος

```
# Μέθοδος για την κίνηση του παίκτη πίσω από τα walls (έξω από το collision point)
def sort_key(self, sprite):
    offset = 0
    if hasattr(sprite, "properties"):
        offset = sprite.properties.get("sort_offset", 0)

    # Αν είναι player sprite, κάνουμε sort με βάση τα "πόδια" (bottom)
    if isinstance(sprite, (PlayerSprite, EnemySprite)):
        return sprite.bottom

    return sprite.center_y + offset     # Για όλα τα άλλα sprites, sort με βάση το center_y + offset
```

Με αυτόν τον τρόπο, κάθε αντικείμενο του χάρτη μπορεί να σχεδιαστεί στη σωστή σειρά σε σχέση με τον παίκτη και τους εχθρούς. Η μέθοδος καλείται πριν από τη σχεδίαση της λίστας των sprites, ώστε το actor_list να ταξινομείται δυναμικά και η απεικόνιση να παραμένει ομαλή κατά την κίνηση μέσα στον χάρτη.

4.1.2 Υλοποίηση χαρακτήρα παίκτη και animations

Για την υλοποίηση του χαρακτήρα χρησιμοποιήθηκαν δισδιάστατα γραφικά σχεδιασμένα για top-down απεικόνιση, τα οποία προέρχονται από την πλατφόρμα CraftPix και διατίθενται δωρεάν. Τα sprites του χαρακτήρα παρέχονται σε μορφή sprite sheet, δηλαδή ως μία ενιαία εικόνα που περιλαμβάνει διαδοχικά καρέ (frames) του χαρακτήρα. Κατά την εκτέλεση του παιχνιδιού, τα επιμέρους frames απομονώνονται από το sprite sheet και προβάλλονται με σειρά, ώστε να δημιουργούνται τα αντίστοιχα animations.

Ο χαρακτήρας υλοποιήθηκε με πέντε βασικές καταστάσεις: ακίνητος (idle), περπάτημα (walk), επίθεση (attack), τραυματισμού (hurt) και πεθαίνει (death). Επιπλέον, για την υποστήριξη του top-down gameplay, κάθε κατάσταση παρέχεται για τέσσερις κατευθύνσεις κίνησης: προς τα πάνω, προς τα κάτω, προς τα αριστερά και προς τα δεξιά. Με τον τρόπο αυτό διασφαλίζεται η ορθή οπτική αναπαράσταση της κίνησης και της κατεύθυνσης του χαρακτήρα κατά τη διάρκεια του παιχνιδιού.

Η εναλλαγή των καταστάσεων και των κατευθύνσεων συγχρονίζεται με την είσοδο που δίνει ο παίκτης και με την κατάσταση που λαμβάνεται από τον server. Οι καταστάσεις idle και walk είναι επαναλαμβανόμενες, δηλαδή τα frames τους προβάλλονται κυκλικά όσο ο χαρακτήρας παραμένει ακίνητος ή κινείται. Αντίθετα, οι καταστάσεις attack και death δεν εκτελούνται κυκλικά, αλλά παίζονται ως animations μίας φοράς. Με αυτόν τον τρόπο η επίθεση, ο τραυματισμός και ο θάνατος εμφανίζονται ως διακριτές ενέργειες και δεν επαναλαμβάνονται.

Figure 15: Κατάσταση idle του χαρακτήρα σε sprite sheet (όταν είναι ακίνητος)

Η κατάσταση τραυματισμού ενεργοποιείται όταν ο παίκτης δέχεται ζημιά από εχθρό, παρέχοντας οπτική ανατροφοδότηση ότι έχει υποστεί απώλεια ζωής. Στην περίπτωση αυτή δεν χρησιμοποιείται ξεχωριστό hurt animation, αλλά εφαρμόζεται οπτικό εφέ blinking, κατά το οποίο το sprite του χαρακτήρα αναβοσβήνει για σύντομο χρονικό διάστημα. Το εφέ αυτό κάνει πιο εμφανές στον παίκτη ότι δέχτηκε χτύπημα και βοηθά στην καλύτερη κατανόηση της κατάστασης μάχης.

Figure 16: Ζημιά που δέχεται ο παίκτης και hurt animation

Αντίστοιχα, η κατάσταση θανάτου ενεργοποιείται όταν η ζωή του παίκτη μηδενιστεί. Σε αυτή την περίπτωση προβάλλεται το death animation και, μετά από μικρό χρονικό διάστημα, το sprite του χαρακτήρα απομακρύνεται από τον χάρτη. Όταν ο παίκτης πεθάνει, οι εχθροί που τον είχαν ως στόχο σταματούν να τον καταδιώκουν και επιστρέφουν στο αρχικό τους σημείο εμφάνισης.

Figure 17: Θάνατος παίκτη και death animation

4.1.2.1 Σύγκρουση παίκτη με τοίχους και χαρακτήρες

Οι παίκτες διαθέτουν μηχανισμό σύγκρουσης (collision) με τα εμπόδια του χάρτη, ώστε να μην είναι δυνατή η διέλευσή τους μέσα από τοίχους ή άλλες μη προσπελάσιμες περιοχές του περιβάλλοντος. Ο έλεγχος αυτός είναι απαραίτητος για τη σωστή κίνηση του χαρακτήρα στον χώρο του παιχνιδιού, καθώς διασφαλίζει ότι η μετακίνησή του περιορίζεται μόνο στις επιτρεπτές περιοχές του χάρτη. Για την υλοποίηση του μηχανισμού αυτού χρησιμοποιείται έλεγχος σύγκρουσης με βάση ορθογώνιο πλαίσιο, το οποίο ορίζεται από τη θέση και τις διαστάσεις του hitbox του παίκτη. Το ορθογώνιο πλαίσιο αποτελεί μια απλοποιημένη περιοχή ελέγχου γύρω από τον χαρακτήρα, η οποία έχει σχήμα ορθογωνίου και ευθυγραμμίζεται με τους άξονες του συστήματος συντεταγμένων. Μέσω αυτού του πλαισίου ελέγχεται αν ο παίκτης επικαλύπτεται με κάποιο εμπόδιο του χάρτη. Η προσέγγιση αυτή επιτρέπει αποδοτικό και σχετικά απλό έλεγχο συγκρούσεων κατά την κίνηση των οντοτήτων στον χώρο.

Figure 18: Μέθοδος για τη σύγκρουση χαρακτήρων σε τοίχους

```
# Μέθοδος για το collision
# Ελέγχει collision με τοίχους χρησιμοποιώντας ορθογώνιο hitbox (AABB)
def collides_with_walls_aabb(x, y, w, h):
    # Αν ο παίκτης ή ο εχθρός είναι πάνω στη γέφυρα, αγνοούμε το collision από walls
    if on_bridge(x, y, w, h):
        return False

    # Υπολογισμός ορίων AABB (Axis-Aligned Bounding Box)
    left   = x - w / 2
    right  = x + w / 2
    bottom = y - h / 2
    top    = y + h / 2

    # Έλεγχος overlap με κάθε wall sprite
    for wall in wall_list:
        if right > wall.left and left < wall.right and top > wall.bottom and bottom < wall.top:
            return True
    return False
```

Η μέθοδος collides_with_walls_aabb(x, y, w, h) χρησιμοποιείται για τον έλεγχο σύγκρουσης μιας οντότητας με τα εμπόδια του χάρτη. Ο έλεγχος βασίζεται σε ορθογώνιο πλαίσιο σύγκρουσης, το οποίο ορίζεται από τη θέση του χαρακτήρα και τις διαστάσεις του hitbox του. Η μέθοδος ελέγχει αν μια οντότητα βρίσκεται πάνω σε γέφυρα μέσω της on_bridge(x, y, w, h). Στην περίπτωση αυτή, η σύγκρουση με τους τοίχους αγνοείται και η μέθοδος επιστρέφει την τιμή False, επιτρέποντας την κίνηση του χαρακτήρα στην περιοχή της γέφυρας.

Στη συνέχεια, υπολογίζονται τα όρια του ορθογωνίου hitbox, δηλαδή οι τιμές left, right, bottom και top, με βάση το κέντρο του χαρακτήρα και τις διαστάσεις πλάτους (w) και ύψους (h). Οι τιμές αυτές χρησιμοποιούνται για να προσδιοριστεί η ακριβής περιοχή που καταλαμβάνει ο παίκτης στον χώρο του παιχνιδιού.

Έπειτα, η μέθοδος διατρέχει όλα τα εμπόδια που περιλαμβάνονται στη λίστα wall_list και ελέγχει αν υπάρχει επικάλυψη (overlap) μεταξύ του hitbox του παίκτη και του αντίστοιχου τοίχου. Αν διαπιστωθεί ότι τα δύο ορθογώνια επικαλύπτονται, τότε θεωρείται ότι υπάρχει σύγκρουση και η μέθοδος επιστρέφει True.

Αν δεν εντοπιστεί σύγκρουση με κανένα από τα εμπόδια του χάρτη, η μέθοδος επιστρέφει False. Με τον τρόπο αυτό, η μέθοδος αποτελεί βασικό μηχανισμό ελέγχου της κίνησης του παίκτη, διασφαλίζοντας ότι δεν μπορεί να διέρχεται μέσα από τοίχους ή άλλα μη προσπελάσιμα σημεία του περιβάλλοντος.

Η μέθοδος player_hits_walls(x, y) αποτελεί βοηθητική μέθοδο για την εξειδίκευση της γενικής μεθόδου ελέγχου σύγκρουσης για τον παίκτη. Πιο συγκεκριμένα, καλεί την collides_with_walls_aabb(x, y, w, h) χρησιμοποιώντας τις σταθερές διαστάσεις του hitbox του παίκτη, δηλαδή PLAYER_WIDTH και PLAYER_HEIGHT.

Figure 19: Βοηθητική μέθοδος ελέγχου σύγκρουσης του παίκτη με τα εμπόδια του χάρτη

```
# Μέθοδος για τον έλεγχο collision του παίκτη με τις σταθερές διαστάσεις του
def player_hits_walls(x, y):
    return collides_with_walls_aabb(x, y, PLAYER_WIDTH, PLAYER_HEIGHT)
```

Οι παίκτες διαθέτουν επίσης μηχανισμό σύγκρουσης με τους εχθρούς, ώστε να αποτρέπεται η αλληλοεπικάλυψη των οντοτήτων κατά την κίνησή τους στον χώρο του παιχνιδιού. Ο μηχανισμός αυτός είναι απαραίτητος για τη διατήρηση ρεαλιστικής συμπεριφοράς και για την αποφυγή φαινομένων διείσδυσης, κατά τα οποία ένας χαρακτήρας θα μπορούσε να περάσει μέσα από έναν εχθρό.

Κατά την αρχική υλοποίηση της κίνησης, παρατηρήθηκε ότι όταν ένας παίκτης ή ένας εχθρός παρέμενε ακίνητος και η άλλη οντότητα κινούνταν προς το μέρος του, η μεταξύ τους επαφή μπορούσε να προκαλέσει ανεπιθύμητη μετατόπιση της ακίνητης οντότητας. Η συμπεριφορά αυτή δεν ήταν επιθυμητή, καθώς οδηγούσε σε μη ρεαλιστική αλληλεπίδραση μεταξύ παικτών και εχθρών. Για την αντιμετώπιση του προβλήματος αυτού υλοποιήθηκε η μέθοδος player_enemy_blocking(prev_players, prev_enemies), η οποία αποτρέπει την αλληλοεπικάλυψη και το «σπρώξιμο» μεταξύ των οντοτήτων.

Για τον έλεγχο της σύγκρουσης μεταξύ παίκτη και εχθρού χρησιμοποιείται αρχικά η βοηθητική μέθοδος aabb_overlap(ax, ay, aw, ah, bx, by, bw, bh), η οποία ελέγχει αν δύο ορθογώνια πλαίσια σύγκρουσης επικαλύπτονται. Η μέθοδος αυτή αξιοποιείται για την ανίχνευση σύγκρουσης μεταξύ δύο οντοτήτων με βάση τη θέση και τις διαστάσεις των hitbox τους.

Figure 20: Βοηθητική μέθοδος για τον έλεγχο επικάλυψης δύο οντοτήτων

```
# Μέθοδος που ελέγχει αν δύο ορθογώνια επικαλύπτονται, ώστε να ανιχνευθεί σύγκρουση
def aabb_overlap(ax, ay, aw, ah, bx, by, bw, bh):
    # Ελέγχει αν τα κέντρα των δύο ορθογωνίων είναι αρκετά κοντά ώστε τα hitbox τους να επικαλύπτονται σε X και Y
    return (abs(ax - bx) * 2 < (aw + bw)) and (abs(ay - by) * 2 < (ah + bh))
```

Η μέθοδος δέχεται ως παραμέτρους το κέντρο του πρώτου και του δεύτερου ορθογωνίου, δηλαδή τις θέσεις ax, ay και bx, by, καθώς και τις αντίστοιχες διαστάσεις πλάτους και ύψους (aw, ah, bw, bh).

Στη συνέχεια, πραγματοποιείται έλεγχος της απόστασης μεταξύ των κέντρων των δύο ορθογωνίων στον άξονα x και στον άξονα y. Αν η απόσταση αυτή είναι μικρότερη από το άθροισμα των μισών διαστάσεων των δύο ορθογωνίων σε κάθε άξονα, τότε προκύπτει επικάλυψη και η μέθοδος επιστρέφει True. Διαφορετικά, επιστρέφει False. Με τον τρόπο αυτό, η μέθοδος αυτή αποτελεί βασικό βοηθητικό μηχανισμό για τον έλεγχο σύγκρουσης μεταξύ δυναμικών οντοτήτων του παιχνιδιού, όπως οι παίκτες και οι εχθροί.

Figure 21: Μέθοδος για την αποφυγή του σπρωξίματος οντότητας, χωρίς να κινείται

```
# Μέθοδος για την αποφυγή του "σπρωξίματος" μεταξύ παίκτη και εχθρού (δεν κινεί ο ένας τον άλλο)
# Αν παίκτης και εχθρός επικαλυφθούν τότε και οι δύο επιστρέφουν στις προηγούμενες θέσεις τους
def player_enemy_blocking(prev_players, prev_enemies):
    for pid, p in players.items():
        # Προηγούμενη θέση παίκτη
        prev_player_x, prev_player_y = prev_players.get(pid, (p["x"], p["y"]))

        for eid, e in enemies.items():
            # Αγνοούμε νεκρούς εχθρούς
            if e.get("dead"):
                continue

            # Προηγούμενη θέση enemy
            prev_enemy_x, prev_enemy_y = prev_enemies.get(eid, (e["x"], e["y"]))

            # Αν υπάρχει overlap, επαναφέρουμε και τους δύο
            if aabb_overlap(
                p["x"], p["y"], PLAYER_WIDTH, PLAYER_HEIGHT,
                e["x"], e["y"], e["hitbox_w"], e["hitbox_h"]
            ):
                p["x"], p["y"] = prev_player_x, prev_player_y
                e["x"], e["y"] = prev_enemy_x, prev_enemy_y
```

Η μέθοδος player_enemy_blocking(prev_players, prev_enemies) χρησιμοποιείται για την αποφυγή του «σπρωξίματος» μεταξύ παικτών και εχθρών. Πιο συγκεκριμένα, ελέγχει αν, μετά την κίνηση, ένας παίκτης και ένας εχθρός αλληλοεπικαλύπτονται. Αν διαπιστωθεί επικάλυψη, και οι δύο οντότητες επαναφέρονται στις προηγούμενες θέσεις τους, ώστε να αποτραπεί η διέλευσή τους η μία μέσα από την άλλη. Αρχικά, η μέθοδος διατρέχει όλους τους παίκτες του παιχνιδιού και ανακτά για κάθε έναν την προηγούμενη θέση του από το όρισμα prev_players, στο οποίο αποθηκεύονται οι προηγούμενες θέσεις των παικτών. Αν δεν υπάρχει καταγεγραμμένη προηγούμενη θέση, χρησιμοποιείται ως προεπιλογή η τρέχουσα θέση του παίκτη.

Στη συνέχεια, για κάθε παίκτη, η μέθοδος διατρέχει όλους τους εχθρούς και αγνοεί όσους έχουν ήδη πεθάνει, δηλαδή όσους βρίσκονται σε κατάσταση “dead”. Για κάθε ενεργό εχθρό ανακτάται επίσης η προηγούμενη θέση του από το όρισμα prev_enemies, με τρόπο αντίστοιχο με αυτόν του παίκτη.

Έπειτα, πραγματοποιείται έλεγχος επικάλυψης μέσω της μεθόδου aabb_overlap(ax, ay, aw, ah, bx, by, bw, bh), η οποία ελέγχει αν το ορθογώνιο hitbox του παίκτη επικαλύπτεται με το αντίστοιχο hitbox του εχθρού. Για τον παίκτη χρησιμοποιούνται οι σταθερές διαστάσεις PLAYER_WIDTH και PLAYER_HEIGHT, ενώ για τον εχθρό χρησιμοποιούνται οι διαστάσεις hitbox_w και hitbox_h.

Αν εντοπιστεί επικάλυψη, η μέθοδος επαναφέρει τόσο τον παίκτη όσο και τον εχθρό στις προηγούμενες θέσεις τους. Με τον τρόπο αυτό αποφεύγεται το φαινόμενο αλληλοδιείσδυσης μεταξύ των δύο οντοτήτων και διασφαλίζεται πιο ομαλή και ρεαλιστική αλληλεπίδραση κατά τη διάρκεια του παιχνιδιού.

4.1.2.2 Κίνηση παικτών

Η κίνηση των παικτών βασίζεται στην αποστολή input από τον client προς τον server. Όταν ο παίκτης πατά κάποιο πλήκτρο κίνησης, ο client δεν αποφασίζει οριστικά τη νέα θέση του χαρακτήρα, αλλά στέλνει την αντίστοιχη κατεύθυνση κίνησης στον server. Ο server επεξεργάζεται το input, ενημερώνει τη θέση του παίκτη με βάση τους κανόνες του παιχνιδιού και στη συνέχεια αποστέλλει τη νέα κατάσταση πίσω στους clients. Με αυτόν τον τρόπο η θέση των παικτών παραμένει ελεγχόμενη από τον server και μειώνονται πιθανά προβλήματα ασυμφωνίας μεταξύ των clients.

Για την ομαλότερη εμφάνιση της κίνησης των παικτών στον client εφαρμόστηκε τεχνική παρεμβολής (interpolation) και πρόβλεψης θέσης (extrapolation). Ο client δεν μετακινεί τα sprites των παικτών απότομα στην τελευταία θέση που λαμβάνει από τον server, αλλά αποθηκεύει τις δύο πιο πρόσφατες θέσεις κάθε παίκτη μαζί με το αντίστοιχο server tick. Με βάση τη διαφορά των θέσεων και του χρόνου υπολογίζεται η ταχύτητα του παίκτη και γίνεται μια μικρή πρόβλεψη της επόμενης θέσης. Στη συνέχεια, το sprite μετακινείται ομαλά προς αυτή τη θέση με γραμμική παρεμβολή.

Figure 22: Αρχικό τμήμα της μεθόδου εξομάλυνσης κίνησης παικτών

```python
# Μέθοδος που κάνει interpolation και extrapolation ώστε η κίνηση των παικτών να φαίνεται ομαλή
def apply_player_smoothing(self, delta_time):
    # Αν δεν υπάρχει player ή client id, δεν κάνουμε τίποτα
    if CLIENT_PLAYER_ID is None or self.player_sprite is None:
        return

    # Δημιουργούμε ενιαίο dict με όλα τα sprites
    all_sprites = {CLIENT_PLAYER_ID: self.player_sprite}
    all_sprites.update(self.other_sprites)

    for pid, sprite in all_sprites.items():
        buf = self.position_buffers.get(pid)
        if not buf:
            continue

        # Αν έχουμε μόνο μία θέση, πάμε κατευθείαν εκεί
        if len(buf) == 1:
            x, y, _ = buf[0]
            sprite.center_x = x
            sprite.center_y = y
            continue

        # Παίρνουμε τις δύο πιο πρόσφατες θέσεις από τον server
        (x0, y0, tick0), (x1, y1, tick1) = buf[0], buf[1]
        dt_ticks = tick1 - tick0
        dt_server = dt_ticks * getattr(self, "tick_dt", 0.02)

        # Αν κάτι πάει στραβά με τα ticks
        if dt_ticks <= 0 or dt_server <= 0:
            # Πάμε απευθείας στην τελευταία θέση
            target_x, target_y = x1, y1
        else:
            # Υπολογισμός ταχύτητας
            vx = (x1 - x0) / dt_server
            vy = (y1 - y0) / dt_server

            # Extrapolation: πρόβλεψη θέσης λίγο μπροστά
            prediction_dt = getattr(self, "tick_dt", 0.02)
            target_x = x1 + vx * prediction_dt
            target_y = y1 + vy * prediction_dt
```

Στο πρώτο τμήμα της μεθόδου ελέγχεται αν υπάρχουν διαθέσιμες θέσεις για κάθε παίκτη στο αντίστοιχο position buffer. Το buffer αυτό περιέχει τις πιο πρόσφατες θέσεις που έχει στείλει ο server για τον συγκεκριμένο παίκτη. Αν δεν υπάρχει καμία διαθέσιμη θέση, η μέθοδος δεν εκτελεί κάποια ενέργεια για το συγκεκριμένο sprite. Αν υπάρχει μόνο μία θέση, το sprite μεταφέρεται απευθείας σε αυτή, επειδή δεν υπάρχουν ακόμη αρκετά δεδομένα για να υπολογιστεί ομαλή ενδιάμεση κίνηση. Όταν υπάρχουν δύο θέσεις, η μέθοδος παίρνει την προηγούμενη και τη νεότερη θέση μαζί με τα αντίστοιχα server ticks. Στη συνέχεια υπολογίζεται η χρονική διαφορά ανάμεσα στα δύο ticks και, με βάση αυτή, υπολογίζεται η ταχύτητα κίνησης του παίκτη στον οριζόντιο και στον κατακόρυφο άξονα. Με τη χρήση αυτής της ταχύτητας γίνεται μια μικρή πρόβλεψη της επόμενης θέσης του παίκτη, ώστε η κίνηση να φαίνεται πιο φυσική και να μειώνεται η αίσθηση καθυστέρησης.

Figure 23: Υπολογισμός interpolation και extrapolation για την κίνηση παικτών

```
# Τοπικός χρόνος interpolation
t_local = self.interp_t.get(pid, 0.0) + delta_time
self.interp_t[pid] = t_local

# Παράμετρος interpolation 0 ή 1
if dt_server > 0:
    x_param = t_local / dt_server
else:
    x_param = 1.0

if x_param > 1.0:
    x_param = 1.0
elif x_param < 0.0:
    x_param = 0.0

# Θέση snapshot (αφετηρία interpolation)
snap_x, snap_y = self.snapshots.get(pid, (sprite.center_x, sprite.center_y))

# Linear interpolation (LERP)
sprite.center_x = snap_x + (target_x - snap_x) * x_param
sprite.center_y = snap_y + (target_y - snap_y) * x_param
```

Στο δεύτερο τμήμα της μεθόδου υπολογίζεται ο τοπικός χρόνος παρεμβολής για κάθε παίκτη. Ο χρόνος αυτός αυξάνεται σε κάθε frame του client με βάση το delta_time, δηλαδή τον χρόνο που πέρασε από το προηγούμενο frame. Στη συνέχεια υπολογίζεται η παράμετρος x_param, η οποία παίρνει τιμές από 0 έως 1 και δείχνει την πρόοδο της μετάβασης προς τη νέα θέση. Όταν η τιμή είναι κοντά στο 0, το sprite βρίσκεται πιο κοντά στην αρχική snapshot θέση, ενώ όταν πλησιάζει το 1, το sprite πλησιάζει τη νέα προβλεπόμενη θέση. Η τιμή περιορίζεται στο διάστημα 0–1, ώστε το sprite να μην ξεπερνά τον στόχο. Τέλος, με χρήση γραμμικής παρεμβολής, το sprite μετακινείται σταδιακά από τη snapshot θέση προς τη νέα προβλεπόμενη θέση. Με αυτόν τον τρόπο αποφεύγονται οι απότομες αλλαγές θέσης και η κίνηση των παικτών εμφανίζεται πιο ομαλή στον client.

4.1.2.3 Επίθεση παικτών

Για την υλοποίηση της επίθεσης των παικτών προς τους εχθρούς χρησιμοποιείται η μέθοδος apply_player_attacks(). Η μέθοδος αυτή είναι υπεύθυνη για τον έλεγχο του αν ένας παίκτης πραγματοποιεί επίθεση, καθώς και για την επιλογή του κατάλληλου στόχου και την εφαρμογή της αντίστοιχης ζημιάς.

Στη μέθοδο αποθηκεύεται η τρέχουσα χρονική στιγμή στη μεταβλητή now μέσω της time.time(), ώστε να ελέγχεται ο χρόνος επαναφόρτισης (cooldown) μεταξύ διαδοχικών επιθέσεων. Η μέθοδος διατρέχει όλους τους παίκτες του παιχνιδιού και αγνοεί όσους βρίσκονται σε κατάσταση dead, καθώς και όσους δεν έχουν ενεργοποιήσει αίτημα επίθεσης μέσω της μεταβλητής attack_requested, η οποία ενεργοποιείται όταν ο παίκτης πατήσει το κουμπί επίθεσης. Όταν το αίτημα επίθεσης είναι ενεργό, η τιμή του επαναφέρεται σε False, ώστε η ίδια επίθεση να μην εκτελεστεί ξανά χωρίς νέα εντολή από τον χρήστη.

Figure 24: Μέθοδος επιθέσεων παίκτη

Παράλληλα, ελέγχεται αν έχει παρέλθει ο απαιτούμενος χρόνος επαναφόρτισης μέσω της μεταβλητής next_attack_time. Αν ο χρόνος αυτός δεν έχει περάσει, ο παίκτης δεν μπορεί να επιτεθεί. Διαφορετικά, ενημερώνεται η επόμενη διαθέσιμη χρονική στιγμή επίθεσης με βάση τη μεταβλητή attack_cooldown.

Figure 25: Μέθοδος επιθέσεων παίκτη

```
# Μέθοδος που εφαρμόζει τις επιθέσεις των παικτών πάνω στους εχθρούς
def apply_player_attacks():
    now = time.time()

    for p in players.items():
        # Αγνοούμε νεκρούς παίκτες
        if p.get("dead", False):
            continue

        # Αν ο παίκτης δεν πάτησε πλήκτρο για επίθεση, πάμε στον επόμενο
        if not p.get("attack_requested", False):
            continue

        p["attack_requested"] = False   # Θέτουμε το αίτημα επίθεσης σε false μετά την επίθεση

        # Έλεγχος επαναφόρτισης (cooldown)
        if now < p.get("next_attack_time", 0.0):
            continue

        # Ορισμός επόμενου διαθέσιμου attack time
        p["next_attack_time"] = now + p.get("attack_cooldown", 0.45)

        attack_dir = p.get("attack_dir", "down")
        px = p["x"]
        py = p["y"]

        # Μέγιστη απόσταση επίθεσης παίκτη
        attack_range = 70

        # Θα επιλεγεί ο κοντινότερος enemy που είναι μπροστά από τον παίκτη
        target_eid = None
        best_dist = 999999
```

Στη συνέχεια, ανακτώνται η κατεύθυνση επίθεσης από τη μεταβλητή attack_dir, οι συντεταγμένες x και y του παίκτη, καθώς και η μέγιστη εμβέλεια επίθεσης μέσω της μεταβλητής attack_range. Παράλληλα, αρχικοποιούνται οι μεταβλητές target_eid και best_dist, οι οποίες χρησιμοποιούνται για την επιλογή του πλησιέστερου έγκυρου στόχου.

Η μέθοδος διατρέχει όλους τους εχθρούς του παιχνιδιού και αγνοεί όσους βρίσκονται σε κατάσταση dead. Για κάθε ενεργό εχθρό υπολογίζονται οι μεταβολές dx και dy, δηλαδή η διαφορά της θέσης του εχθρού από τη θέση του παίκτη στους άξονες x και y, ώστε να προσδιοριστεί η σχετική θέση του ως προς τον παίκτη.

Η επίθεση επιτρέπεται μόνο προς την κατεύθυνση στην οποία κοιτάζει ο παίκτης, όπως αυτή αποθηκεύεται στη μεταβλητή attack_dir, γεγονός που αποκλείει εχθρούς που βρίσκονται πίσω ή εκτός της κατεύθυνσης επίθεσης.

Figure 26: Επιλογή του πλησιέστερου έγκυρου στόχου για την επίθεση του παίκτη

```
for eid, e in enemies.items():
    # Αγνοούμε νεκρούς εχθρούς
    if e.get("dead", False):
        continue

    dx = e["x"] - px
    dy = e["y"] - py

    # Ο παίκτης χτυπά μόνο προς τη μεριά που κοιτάζει
    if attack_dir == "up" and dy <= 0:
        continue
    if attack_dir == "down" and dy >= 0:
        continue
    if attack_dir == "left" and dx >= 0:
        continue
    if attack_dir == "right" and dx <= 0:
        continue

    # Έλεγχος απόστασης
    d = dist(px, py, e["x"], e["y"])
    if d > attack_range:
        continue

    # Επιλέγουμε τον κοντινότερο έγκυρο στόχο
    if d < best_dist:
        best_dist = d
        target_eid = eid
```

Παράλληλα, υπολογίζεται η απόσταση μεταξύ παίκτη και εχθρού μέσω της μεθόδου dist(px, py, e["x"], e["y"]). Αν η απόσταση αυτή υπερβαίνει τη μέγιστη εμβέλεια επίθεσης, ο εχθρός απορρίπτεται ως μη έγκυρος στόχος. Από τους εχθρούς που ικανοποιούν τόσο τον έλεγχο κατεύθυνσης όσο και τον έλεγχο απόστασης, επιλέγεται ο κοντινότερος, με βάση τις μεταβλητές best_dist και target_eid.

Αν δεν βρεθεί έγκυρος στόχος, δηλαδή αν η μεταβλητή target_eid έχει τιμή None, δεν πραγματοποιείται επίθεση. Σε διαφορετική περίπτωση, ανακτάται ο αντίστοιχος εχθρός από τη συλλογή enemies και υπολογίζεται η ζημιά που θα δεχθεί, αφαιρώντας από την τιμή ζημιάς του παίκτη (damage) την τιμή αντίστασης (resist) του εχθρού.

Η τελική ζημιά αποθηκεύεται στη μεταβλητή dmg και αφαιρείται από τη ζωή του εχθρού (hp). Αν η ζωή του εχθρού μηδενιστεί ή γίνει αρνητική, ορίζεται σε 0, ο εχθρός χαρακτηρίζεται ως νεκρός μέσω της μεταβλητής dead και η κατάστασή του αλλάζει σε death, ώστε να ενεργοποιηθεί το αντίστοιχο animation θανάτου.

Αν ο εχθρός εξακολουθεί να έχει διαθέσιμη ζωή, αυξάνεται η μεταβλητή

hurt_seq, η οποία χρησιμοποιείται για την ενεργοποίηση της κατάστασης τραυματισμού (hurt) και παρέχει οπτική ανατροφοδότηση ότι ο εχθρός δέχθηκε επιτυχημένο χτύπημα χωρίς να πεθάνει.

Figure 27: Υπολογισμός ζημιάς και ενημέρωση της κατάστασης του εχθρού μετά από επιτυχημένη επίθεση του παίκτη

```
# Αν δεν βρέθηκε στόχος, δεν γίνεται hit
if target_eid is None:
    continue

e = enemies[target_eid]

# Υπολογισμός damage μετά το resist του εχθρού
dmg = max(0, p.get("damage", 35) - e.get("resist", 0))
e["hp"] -= dmg

# Αν ο enemy πεθάνει
if e["hp"] <= 0:
    e["hp"] = 0
    e["dead"] = True
    e["state"] = "death"
else:
    # Διαφορετικά αυξάνουμε hurt sequence για animation
    e["hurt_seq"] = e.get("hurt_seq", 0) + 1
```

4.1.2.3.1 Επίθεση παικτών από απόσταση

Για την υλοποίηση των επιθέσεων του παίκτη χρησιμοποιείται έλεγχος κατεύθυνσης και εμβέλειας. Η μέθοδος enemy_in_player_directional_range ελέγχει αν ένας εχθρός βρίσκεται μέσα στην περιοχή που καλύπτει η επίθεση του παίκτη. Αρχικά υπολογίζεται η σχετική θέση του εχθρού ως προς τον παίκτη στους άξονες X και Y. Στη συνέχεια λαμβάνεται η κατεύθυνση της επίθεσης, δηλαδή αν ο παίκτης επιτίθεται προς τα πάνω, κάτω, αριστερά ή δεξιά.

Figure 28: Μέθοδος ελέγχου εμβέλειας και κατεύθυνσης επίθεσης παίκτη

```python
# Μέθοδος που ελέγχει αν ένας enemy βρίσκεται μέσα στην κατεύθυνση επίθεσης του παίκτη
def enemy_in_player_directional_range(p, e, attack_defs):
    dx = e["x"] - p["x"]
    dy = e["y"] - p["y"]

    # Παίρνουμε την κατεύθυνση της επίθεσης του παίκτη
    # Αν δεν υπάρχει attack_dir, χρησιμοποιούμε την τρέχουσα κατεύθυνση του παίκτη
    direction = p.get("attack_dir", p.get("dir", "down"))

    attack_range = attack_defs.get("range", 70)                # Απόσταση που φτάνει η επίθεση

    # Πλάτος της επίθεσης που χρησιμοποιείται ώστε η επίθεση να χτυπά μόνο enemies που είναι περίπου στην ίδια ευθεία
    lane_half_width = attack_defs.get("lane_half_width", 36)

    # Αν ο παίκτης χτυπάει δεξιά, ο enemy πρέπει να είναι δεξιά του και μέσα στο επιτρεπτό range και πλάτος
    if direction == "right":
        return dx > 0 and dx <= attack_range and abs(dy) <= lane_half_width

    # Αν ο παίκτης χτυπάει αριστερά, ο enemy πρέπει να είναι αριστερά του
    if direction == "left":
        return dx < 0 and abs(dx) <= attack_range and abs(dy) <= lane_half_width

    # Αν ο παίκτης χτυπάει πάνω, ο enemy πρέπει να είναι πάνω του
    if direction == "up":
        return dy > 0 and dy <= attack_range and abs(dx) <= lane_half_width

    # Αν ο παίκτης χτυπάει κάτω, ο enemy πρέπει να είναι κάτω του
    if direction == "down":
        return dy < 0 and abs(dy) <= attack_range and abs(dx) <= lane_half_width

    return False    # Αν η κατεύθυνση δεν είναι έγκυρη, δεν υπάρχει hit
```

Κάθε επίθεση διαθέτει συγκεκριμένη εμβέλεια (range) και πλάτος λωρίδας (lane_half_width). Η εμβέλεια καθορίζει πόσο μακριά μπορεί να φτάσει η επίθεση, ενώ το πλάτος λωρίδας καθορίζει πόσο κοντά πρέπει να βρίσκεται ο εχθρός στην ίδια ευθεία με τον παίκτη. Με αυτόν τον τρόπο, η επίθεση δεν χτυπά όλους τους εχθρούς γύρω από τον παίκτη, αλλά μόνο όσους βρίσκονται μπροστά του και μέσα στα επιτρεπτά όρια.

Για παράδειγμα, όταν ο παίκτης επιτίθεται προς τα δεξιά, ο εχθρός πρέπει να βρίσκεται δεξιά από τον παίκτη, μέσα στην απόσταση της επίθεσης και με μικρή απόκλιση στον κάθετο άξονα. Αντίστοιχοι έλεγχοι γίνονται για τις κατευθύνσεις αριστερά, πάνω και κάτω. Αν ο εχθρός δεν βρίσκεται στην κατάλληλη κατεύθυνση ή βρίσκεται εκτός εμβέλειας, η μέθοδος επιστρέφει False και δεν εφαρμόζεται ζημιά.

4.1.3 Υλοποίηση εχθρών και animations

Για την υλοποίηση των εχθρών χρησιμοποιήθηκαν spritesheets από την πλατφόρμα CraftPix, τα οποία διατίθενται δωρεάν. Τα γραφικά αυτά είναι σχεδιασμένα για δισδιάστατη top-down απεικόνιση και παρέχουν τα απαραίτητα καρέ (frames) για την οπτική αναπαράσταση των εχθρών μέσα στο παιχνίδι. Κατά την εκτέλεση του παιχνιδιού, τα επιμέρους καρέ απομονώνονται από τα spritesheets και προβάλλονται διαδοχικά, ώστε να δημιουργούνται τα αντίστοιχα animations.

Οι εχθροί υλοποιήθηκαν στις ίδιες καταστάσεις με τους χαρακτήρες των παικτών, δηλαδή για τον καθένα: ακίνητος (idle), περπάτημα (walk), επίθεση (attack), τραυματισμού (hurt) και πεθαίνει (death). Οι εχθροί υποστηρίζουν κίνηση προς τις τέσσερις βασικές κατευθύνσεις, δηλαδή προς τα πάνω, προς τα κάτω, προς τα αριστερά και προς τα δεξιά, αλλά και διαγώνια κίνηση προς όλες τις ενδιάμεσες κατευθύνσεις. Η δυνατότητα αυτή καθιστά την κίνησή τους πιο αποδοτική κατά την καταδίωξη, δυσκολεύοντας τον παίκτη να απομακρυνθεί ή να αποφύγει τις επιθέσεις τους.

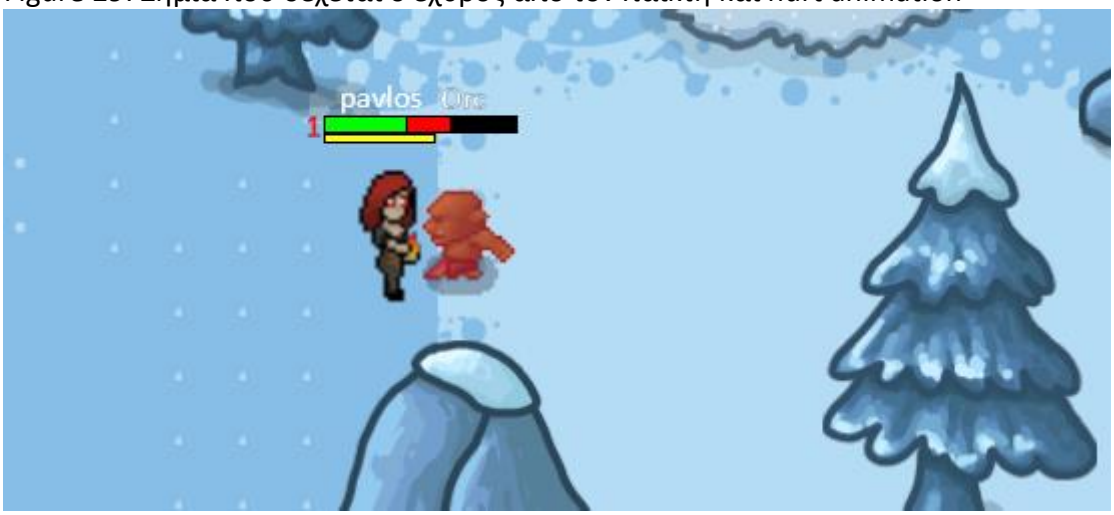
Οι εχθροί τοποθετούνται στα προκαθορισμένα σημεία εμφάνισης που έχουν οριστεί στον χάρτη. Όταν κάποιος παίκτης εισέλθει στην εμβέλεια ανίχνευσης ενός εχθρού, ενεργοποιείται η συμπεριφορά καταδίωξης και ο εχθρός αρχίζει να κινείται προς τη θέση του. Με τον τρόπο αυτό, η συμπεριφορά των εχθρών προσαρμόζεται δυναμικά στην παρουσία του παίκτη, ενισχύοντας τη διαδραστικότητα και το επίπεδο δυσκολίας του παιχνιδιού.

Οι εχθροί διαθέτουν μηχανισμό σύγκρουσης (collision) με τους παίκτες, με αποτέλεσμα να μην είναι δυνατή η διέλευσή τους μέσα από αυτούς. Όταν ένας εχθρός φτάσει σε σημείο επαφής με έναν παίκτη, παρεμποδίζεται η περαιτέρω κίνησή του προς την ίδια κατεύθυνση. Εφόσον ο παίκτης παραμένει ακίνητος και βρίσκεται σε επαφή με τον εχθρό, ο τελευταίος αρχίζει να του προκαλεί ζημιά, με αποτέλεσμα τη σταδιακή μείωση της ζωής του.

Κάθε εχθρός ανήκει σε συγκεκριμένο είδος, όπως για παράδειγμα Orc, και η ονομασία του εμφανίζεται πάνω από τον χαρακτήρα, διευκολύνοντας την άμεση αναγνώρισή του από τον παίκτη. Παράλληλα, πάνω από κάθε εχθρό προβάλλεται κόκκινη μπάρα ζωής, η οποία ενισχύει την οπτική διάκρισή του ως εχθρικού χαρακτήρα και παρέχει πληροφορίες σχετικά με την τρέχουσα κατάστασή του.

Κάθε φορά που ένας εχθρός δέχεται ζημιά, ενεργοποιείται η κατάσταση τραυματισμού (hurt), η οποία χρησιμοποιείται για την οπτική απόδοση της επίδρασης του χτυπήματος στον χαρακτήρα. Με τον τρόπο αυτό παρέχεται άμεση ανατροφοδότηση στον παίκτη ότι η επίθεσή του ήταν επιτυχής και ότι ο εχθρός έχει υποστεί απώλεια ζωής.

Figure 29: Ζημιά που δέχεται ο εχθρός απο τον παίκτη και hurt animation



Όταν η μπάρα ζωής ενός εχθρού μηδενιστεί, δηλαδή όταν η ζωή του φτάσει στο 0, ενεργοποιείται η κατάσταση θανάτου (death). Μετά την ολοκλήρωση του αντίστοιχου animation, ο εχθρός θεωρείται νεκρός και το σώμα του εξαφανίζεται από το παιχνίδι έπειτα από λίγα δευτερόλεπτα.

Figure 30: Θάνατος εχθρού και death animation

4.1.3.1 Σύγκρουση εχθρών

Ο έλεγχος σύγκρουσης των εχθρών με τα εμπόδια του χάρτη πραγματοποιείται με τρόπο αντίστοιχο με εκείνον του παίκτη, αξιοποιώντας την ίδια γενική μέθοδο ελέγχου σύγκρουσης με ορθογώνιο πλαίσιο. Ωστόσο, στην περίπτωση του εχθρού χρησιμοποιείται η μέθοδος enemy_hits_walls(e, x, y), η οποία λαμβάνει υπόψη τις διαστάσεις του δικού του hitbox.

Figure 31: Μέθοδος για τον έλεγχο collision των εχθρών

```
# Μέθοδος για τον έλεγχο collision του εχθρού με βάση το δικό του hitbox
def enemy_hits_walls(e, x, y):
    return collides_with_walls_aabb(x, y, e["hitbox_w"], e["hitbox_h"])
```

Πιο συγκεκριμένα, η μέθοδος καλεί τη γενική μέθοδο collides_with_walls_aabb(x, y, w, h), χρησιμοποιώντας ως πλάτος και ύψος τις τιμές hitbox_w και hitbox_h του εκάστοτε εχθρού. Με τον τρόπο αυτό, ο έλεγχος σύγκρουσης προσαρμόζεται στα ιδιαίτερα χαρακτηριστικά κάθε εχθρικής οντότητας και διασφαλίζεται ότι η κίνησή της περιορίζεται σωστά από τα εμπόδια του χάρτη.

4.1.3.2 Κίνηση εχθρών

Για την κίνηση των εχθρών χρησιμοποιούνται δύο διακριτές μέθοδοι. Η μέθοδος move_enemy(e, delta_x, delta_y) χρησιμοποιείται για την κίνηση ενός εχθρού στον χάρτη, με ξεχωριστή επεξεργασία της μετατόπισης στους άξονες x και y. Η προσέγγιση αυτή επιλέχθηκε ώστε να επιτυγχάνεται καλύτερος έλεγχος των συγκρούσεων (collision detection) και πιο ομαλή κίνηση του εχθρού.

Figure 32: Μέθοδος κίνησης εχθρών

```python
# Μέθοδος κίνησης εχθρού ξεχωριστά στους άξονες X, Y για καλύτερο έλεγχο collision και πιο ομαλή κίνηση
def move_enemy(e, delta_x, delta_y):
    moved = False
    current_x, current_y = e["x"], e["y"]

    # Κίνηση στον άξονα X
    new_x = current_x + delta_x

    # Περιορισμός στον άξονα X ώστε να μείνει μέσα στα όρια του χάρτη
    w = e["hitbox_w"]
    h = e["hitbox_h"]
    new_x = max(w/2, min(new_x, MAP_WIDTH - w/2))

    # Αν δεν υπάρχει collision, εφαρμόζουμε τη νέα θέση στον άξονα X
    if not enemy_hits_walls(e, new_x, current_y):
        e["x"] = new_x
        moved = moved or (abs(delta_x) > 1e-6)

    # Κίνηση στον άξονα Y
    new_y = current_y + delta_y

    # Περιορισμός στον άξονα Y ώστε να μείνει μέσα στα όρια του χάρτη
    new_y = max(h/2, min(new_y, MAP_HEIGHT - h/2))

    # Αν δεν υπάρχει collision, εφαρμόζουμε τη νέα θέση στον άξονα Y
    if not enemy_hits_walls(e, e["x"], new_y):
        e["y"] = new_y
        moved = moved or (abs(delta_y) > 1e-6)

    return moved
```

Αρχικά, η μέθοδος αποθηκεύει την τρέχουσα θέση του εχθρού στις μεταβλητές current_x και current_y, ενώ η μεταβλητή moved χρησιμοποιείται για να καταγράφει αν τελικά πραγματοποιήθηκε μετακίνηση. Στη συνέχεια, υπολογίζεται η νέα πιθανή θέση στον άξονα x με βάση την τιμή delta_x.

Πριν εφαρμοστεί η νέα θέση, πραγματοποιείται έλεγχος ώστε ο εχθρός να παραμένει εντός των ορίων του χάρτη. Για τον σκοπό αυτό λαμβάνονται υπόψη το πλάτος και το ύψος του hitbox του εχθρού, ώστε η θέση του να περιορίζεται σωστά μέσα στις διαστάσεις του χάρτη.

Έπειτα, καλείται η μέθοδος enemy_hits_walls(e, x, y), η οποία ελέγχει αν η νέα θέση προκαλεί σύγκρουση με τοίχους ή άλλα εμπόδια. Αν δεν εντοπιστεί σύγκρουση, τότε ενημερώνεται η συντεταγμένη x του εχθρού.

Ακολουθεί αντίστοιχη διαδικασία για τον άξονα y. Υπολογίζεται η νέα πιθανή θέση με βάση την τιμή delta_y, εφαρμόζεται περιορισμός στα όρια του χάρτη και στη συνέχεια ελέγχεται αν η μετακίνηση προκαλεί σύγκρουση. Εφόσον δεν υπάρχει εμπόδιο, ενημερώνεται η συντεταγμένη y του εχθρού. Τέλος, η μέθοδος επιστρέφει τη λογική τιμή moved, η οποία δείχνει αν ο εχθρός μετακινήθηκε τελικά σε κάποια από τις δύο κατευθύνσεις.

Figure 33: Μέθοδος κίνησης εχθρών προς ένα στόχο

```python
# Μέθοδος που υπολογίζει την κατεύθυνση του εχθρού προς τον στόχο και καλεί τη μέθοδο κίνησης
def move_enemy_towards_target(e, target_x, target_y):
    current_x, current_y = e["x"], e["y"]
    dx = target_x - current_x
    dy = target_y - current_y

    # Ευκλείδεια απόσταση
    d = dist(current_x, current_y, target_x, target_y)

    # Αν η απόσταση είναι σχεδόν μηδενική, σταματάει η κίνηση
    if d < 0.001:
        return

    # Κανονικοποιημένο διάνυσμα κατεύθυνσης προς τον στόχο
    dir_x = dx / d
    dir_y = dy / d

    # Ταχύτητα enemy σε pixels ανά tick
    speed = e["move_speed"]

    now = time.time()

    # Αν ο εχθρός είναι σε unstuck mode, κινείται πλάγια αντί για ευθεία προς τον στόχο
    if now < e.get("unstuck_until", 0.0):
        side = e.get("unstuck_side", 1)

        # Κάθετο διάνυσμα στην κατεύθυνση προς τον στόχο
        px = -dir_y * side
        py =  dir_x * side

        move_enemy(e, px * speed, py * speed)
        e["dir"] = dir_from_delta(px, py)
        return
```

Αντίθετα, η μέθοδος move_enemy_towards_target(e, target_x, target_y) χρησιμοποιείται για τον υπολογισμό της κατεύθυνσης προς έναν στόχο, όπως είναι η θέση του παίκτη, και στη συνέχεια καλεί τη μέθοδο κίνησης ώστε ο εχθρός να μετακινηθεί προς αυτόν.

Πιο συγκεκριμένα, η move_enemy_towards_target(e, target_x, target_y) υπολογίζει τη διαφορά θέσης μεταξύ εχθρού και στόχου στους άξονες x και y, καθώς και την ευκλείδεια απόστασή τους. Με βάση αυτά τα δεδομένα προκύπτει το κανονικοποιημένο διάνυσμα κατεύθυνσης προς τον στόχο, το οποίο συνδυάζεται με την ταχύτητα του εχθρού για την παραγωγή της μετατόπισής του.

Αν η ευθεία κίνηση προς τον στόχο δεν είναι εφικτή λόγω σύγκρουσης, η μέθοδος δοκιμάζει εναλλακτικά κίνηση μόνο στον άξονα x ή μόνο στον άξονα y. Επιπλέον, σε περιπτώσεις όπου ο εχθρός μπορεί να παγιδευτεί, ενεργοποιείται προσωρινά μηχανισμός αποκόλλησης (unstuck), κατά τον οποίο κινείται πλάγια αντί ευθεία προς τον στόχο.

Figure 34: Εφαρμογή τριών επιλογών κίνησης

```
# Κίνηση: δοκιμάζει 3 επιλογές (ευθεία, μόνο άξονα X, μόνο άξονα Y)
moved = False

# Προσπάθεια πλήρους διαγώνιας / ευθείας κίνησης
moved = move_enemy(e, dir_x * speed, dir_y * speed)

# Αν απέτυχε, δοκιμή μόνο στον άξονα X
if not moved:
    moved = move_enemy(e, dir_x * speed, 0)

# Αν απέτυχε και πάλι, δοκιμή μόνο στον άξονα Y
if not moved:
    moved = move_enemy(e, 0, dir_y * speed)

# Ενημέρωση direction για animation
e["dir"] = dir_from_delta(dx, dy)
```

Για την οπτική απόδοση της κίνησης χρησιμοποιείται η μέθοδος dir_from_delta(dx, dy), η οποία μετατρέπει το διάνυσμα μετατόπισης σε μία από τις βασικές κατευθύνσεις του εχθρού. Η μέθοδος συγκρίνει τις μεταβολές στους άξονες x και y και επιστρέφει την αντίστοιχη κατεύθυνση (right, left, up, down) ανάλογα με τον άξονα που υπερισχύει. Με τον τρόπο αυτό ενημερώνεται σωστά η μεταβλητή κατεύθυνσης του εχθρού και επιλέγεται το κατάλληλο animation.

Figure 35: Μετατροπή διανύσματος κίνησης σε κατεύθυνση animation

```
# Μέθοδος που μετατρέπει ένα διάνυσμα (dx, dy) σε κατεύθυνση για animation / state
def dir_from_delta(dx, dy):
    if abs(dx) > abs(dy):
        return "right" if dx > 0 else "left"
    else:
        return "up" if dy > 0 else "down"
```

Για την υλοποίηση της συμπεριφοράς των εχθρών δεν αρκεί μόνο η δυνατότητα κίνησης και σύγκρουσης με το περιβάλλον, αλλά απαιτείται και ένας μηχανισμός που να καθορίζει δυναμικά τις ενέργειές τους κατά τη διάρκεια του παιχνιδιού.

Οι εχθροί πρέπει να μπορούν να εντοπίζουν τον κοντινότερο παίκτη, να αποφασίζουν πότε θα τον ακολουθήσουν, να σταματούν την καταδίωξη όταν αυτός απομακρυνθεί αρκετά, να επιστρέφουν στο αρχικό σημείο εμφάνισής τους όταν δεν υπάρχει ενεργός στόχος και να προσαρμόζουν ανάλογα την κατάστασή τους, όπως idle, walk ή attack. Παράλληλα, είναι απαραίτητο να ελέγχεται αν κάποιος εχθρός έχει παγιδευτεί κατά την κίνησή του, ώστε να ενεργοποιείται ο κατάλληλος μηχανισμός αποκόλλησης και να αποφεύγεται η ακινητοποίησή του σε εμπόδια ή σε σημεία του χάρτη.

Η λειτουργικότητα αυτή υλοποιείται μέσω της μεθόδου update_enemy_chase_and_movement(), η οποία αποτελεί τον βασικό μηχανισμό ενημέρωσης της συμπεριφοράς και της κίνησης όλων των εχθρών του παιχνιδιού. Η μέθοδος εκτελεί διαδοχικά ελέγχους που σχετίζονται με την εύρεση στόχου, τη διατήρηση ή απώλεια του στόχου, την επιλογή κατάλληλης κατάστασης συμπεριφοράς, καθώς και την αντιμετώπιση περιπτώσεων στις οποίες ο εχθρός αδυνατεί να προχωρήσει κανονικά προς τον προορισμό του.

Figure 36: Εντοπισμός κοντινότερου παίκτη και έλεγχος στόχου εχθρού

```
def update_enemy_chase_and_movement():
    for e in enemies.values():
        # Παραλείπουμε νεκρούς εχθρούς
        if e.get("dead"):
            continue

        # Εύρεση κοντινότερου ζωντανού παίκτη
        nearest_pid = None
        nearest_d = 1e9  # Αρχικοποίηση με πολύ μεγάλη τιμή απόστασης, ώστε να βρεθεί ο κοντινότερος παίκτης

        for pid, p in players.items():
            if p.get("dead", False):
                continue

            d = dist(e["x"], e["y"], p["x"], p["y"])
            if d < nearest_d:
                nearest_d = d
                nearest_pid = pid

        # Αν δεν έχει ήδη στόχο, στοχεύει τον κοντινότερο παίκτη αν είναι μέσα στο aggro radius
        if e["target"] is None:
            if nearest_pid is not None and nearest_d <= e["aggro_radius"]:
                e["target"] = nearest_pid

        # Απώλεια στόχου
        if e["target"] is not None:
            tp = players.get(e["target"])

            # Αν ο στόχος δεν υπάρχει πια ή είναι νεκρός, τον αφήνει
            if tp is None or tp.get("dead", False):
                e["target"] = None
            else:
                d = dist(e["x"], e["y"], tp["x"], tp["y"])

                # Αν ο στόχος απομακρύνθηκε πολύ, σταματά το κυνήγι
                if d > e["lose_radius"]:
                    e["target"] = None
```

Η μέθοδος αυτή χρησιμοποιείται για την ενημέρωση της συμπεριφοράς και της κίνησης όλων των εχθρών του παιχνιδιού. Αρχικά, η μέθοδος διατρέχει όλους τους εχθρούς και παραλείπει όσους βρίσκονται σε κατάσταση dead. Για κάθε ενεργό εχθρό αναζητείται ο κοντινότερος ζωντανός παίκτης, υπολογίζοντας την απόσταση μεταξύ τους. Για τον σκοπό αυτό χρησιμοποιούνται οι μεταβλητές nearest_pid και nearest_d, στις οποίες αποθηκεύονται αντίστοιχα το αναγνωριστικό του πλησιέστερου παίκτη και η μικρότερη απόσταση που έχει εντοπιστεί.

Figure 37: Καθορισμός συμπεριφοράς εχθρού ως προς επίθεση, κίνηση ή επιστροφή στο σημείο εμφάνισης

```
# Συμπεριφορά εχθρού
if e["target"] is not None:
    tp = players[e["target"]]
    d = dist(e["x"], e["y"], tp["x"], tp["y"])

    # Αν ο στόχος είναι εντός attack range, ο εχθρός επιτίθεται
    if d <= e["attack_range"]:
        e["state"] = "attack"
        e["dir"] = dir_from_delta(tp["x"] - e["x"], tp["y"] - e["y"])
        e["unstuck_until"] = 0.0    # Επαναφορά unstuck mode

    # Αλλιώς κινείται προς τον στόχο
    else:
        e["state"] = "walk"
        move_enemy_towards_target(e, tp["x"], tp["y"])
else:
    # Αν δεν υπάρχει στόχος, ο εχθρός επιστρέφει στο αρχικό spawn point
    sx, sy = e["spawn_x"], e["spawn_y"]
    d = dist(e["x"], e["y"], sx, sy)

    # Αν έφτασε κοντά στο spawn, μένει σε idle state
    if d <= 2.0:
        e["state"] = "idle"
        e["unstuck_until"] = 0.0    # Επαναφορά unstuck mode
    else:
        e["state"] = "walk"
        move_enemy_towards_target(e, sx, sy)
```

Αν ο εχθρός δεν έχει ήδη στόχο, τότε ελέγχεται αν ο κοντινότερος παίκτης βρίσκεται εντός της ακτίνας ανίχνευσης (aggro_radius). Σε αυτή την περίπτωση, ο παίκτης αυτός ορίζεται ως νέος στόχος του εχθρού. Αντίθετα, όταν ο εχθρός έχει ήδη στόχο, η μέθοδος ελέγχει αν αυτός εξακολουθεί να είναι έγκυρος. Αν ο στόχος δεν υπάρχει πλέον, αν έχει πεθάνει ή αν έχει απομακρυνθεί πέρα από την ακτίνα απώλειας στόχου (lose_radius), τότε ο εχθρός παύει να τον καταδιώκει και η μεταβλητή target επανέρχεται σε κενή τιμή.

Figure 38: Έλεγχος παγίδευσης εχθρού και ενεργοποίηση του μηχανισμού αποκόλλησης

```
# Έλεγχος αν ο εχθρός έχει κολλήσει
lx = e.get("last_x", e["x"])
ly = e.get("last_y", e["y"])

# Υπολογισμός πραγματικής μετακίνησης από το προηγούμενο tick
moved_dist = dist(e["x"], e["y"], lx, ly)

# Αν κινείται ("walk state") αλλά δεν προχωρά αρκετά, μετράμε stuck time
if moved_dist < MIN_PROGRESS_PX and e["state"] == "walk":
    e["stuck_time"] = e.get("stuck_time", 0.0) + TICK_DT
else:
    e["stuck_time"] = 0.0

# Αποθήκευση τρέχουσας θέσης για το επόμενο tick
e["last_x"] = e["x"]
e["last_y"] = e["y"]

# Αν μείνει stuck για αρκετό χρόνο, ενεργοποιούμε unstuck mode
if e["stuck_time"] >= STUCK_THRESHOLD:
    e["stuck_time"] = 0.0
    e["unstuck_until"] = time.time() + UNSTUCK_DURATION

    # Αν έχει target, ξεκολλάει προς εκείνον
    if e.get("target") is not None and e["target"] in players:
        tx = players[e["target"]]["x"]
        ty = players[e["target"]]["y"]

    # Αλλιώς ξεκολλάει προς το spawn point του
    else:
        tx = e["spawn_x"]
        ty = e["spawn_y"]

    # Επιλέγουμε την πλάγια κατεύθυνση που τον φέρνει πιο κοντά στον στόχο
    e["unstuck_side"] = choose_unstuck_side_towards_target(e, tx, ty)
```

Παράλληλα, στο πλαίσιο της ενημέρωσης της συμπεριφοράς των εχθρών, έχει ενσωματωθεί και μηχανισμός αντιμετώπισης περιπτώσεων παγίδευσης κατά την κίνησή τους. Ο μηχανισμός αυτός ενεργοποιείται όταν ένας εχθρός δεν καταφέρνει να προχωρήσει κανονικά προς τον στόχο ή προς το σημείο επιστροφής του, εξαιτίας εμποδίων ή της γεωμετρίας του χάρτη. Στην περίπτωση αυτή, εφαρμόζεται προσωρινά ειδική λογική αποκόλλησης (unstuck), ώστε ο εχθρός να μπορέσει να ανακτήσει ομαλή πορεία. Η λειτουργία αυτή αναλύεται αναλυτικότερα στην επόμενη ενότητα.

4.1.3.2.1 Εξομάλυνση κίνησης εχθρών μετά τη μείωση της συχνότητας ενημέρωσης

Μετά τη βελτιστοποίηση του server, όπου η λογική των εχθρών δεν εκτελείται πλέον σε κάθε tick αλλά ανά μεγαλύτερο χρονικό διάστημα, παρατηρήθηκε ότι η κίνηση των εχθρών στον client μπορούσε να φαίνεται πιο διακεκομμένη. Αυτό συνέβαινε επειδή οι νέες θέσεις των εχθρών αποστέλλονταν λιγότερο συχνά από τον

server, με αποτέλεσμα τα sprites να μετακινούνται απότομα από τη μία θέση στην άλλη. Για την αντιμετώπιση αυτού του προβλήματος εφαρμόστηκε εξομάλυνση κίνησης μέσω interpolation. Ο client αποθηκεύει τις δύο πιο πρόσφατες θέσεις κάθε εχθρού και υπολογίζει ενδιάμεσες θέσεις ανάμεσα στην προηγούμενη και τη νέα θέση, ώστε η κίνηση να εμφανίζεται πιο ομαλή.

Figure 39: Αρχικό τμήμα της μεθόδου εξομάλυνσης κίνησης εχθρών

```
# Μέθοδος που κάνει interpolation και extrapolation ώστε η κίνηση των εχθρών να φαίνεται ομαλή
def apply_enemy_smoothing(self, delta_time):
    # Διατρέχουμε όλα τα enemy sprites που υπάρχουν στον client
    for eid, sprite in self.enemy_sprites.items():
        # Παίρνουμε το position buffer του συγκεκριμένου εχθρού
        # Το buffer κρατά τις πιο πρόσφατες θέσεις που έστειλε ο server
        buf = self.enemy_position_buffers.get(eid)

        if not buf:     # Αν δεν υπάρχουν διαθέσιμες θέσεις για τον εχθρό, δεν κάνουμε τίποτα
            continue

        # Αν έχουμε μόνο μία θέση, πάμε απευθείας εκεί
        if len(buf) == 1:
            x, y, _ = buf[0]
            sprite.center_x = x
            sprite.center_y = y
            continue

        # Παίρνουμε τις δύο πιο πρόσφατες θέσεις, κάθε θέση περιλαμβάνει x, y και server tick
        (x0, y0, tick0), (x1, y1, tick1) = buf[0], buf[1]

        dt_ticks = tick1 - tick0    # Υπολογίζουμε πόσα server ticks πέρασαν ανάμεσα στις δύο θέσεις

        # Μετατρέπουμε τη διαφορά των ticks σε πραγματικό χρόνο με βάση τη διάρκεια κάθε server tick
        dt_server = dt_ticks * getattr(self, "tick_dt", 0.02)

        # Αν για οποιονδήποτε λόγο τα ticks ή ο χρόνος δεν είναι έγκυρα, το sprite μεταφέρεται απευθείας στην τελευταία γνωστή θέση
        if dt_ticks <= 0 or dt_server <= 0:
            sprite.center_x = x1
            sprite.center_y = y1
            continue
```

Στο πρώτο τμήμα της μεθόδου apply_enemy_smoothing, ο client διατρέχει όλα τα sprites των εχθρών που υπάρχουν στην τρέχουσα περιοχή. Για κάθε εχθρό ανακτάται το αντίστοιχο position buffer, το οποίο περιέχει τις πιο πρόσφατες θέσεις που έχουν ληφθεί από τον server. Αν δεν υπάρχει διαθέσιμη θέση για κάποιον εχθρό, τότε η διαδικασία συνεχίζει στον επόμενο. Αν υπάρχει μόνο μία θέση, το sprite μεταφέρεται απευθείας σε αυτή, καθώς δεν υπάρχουν ακόμη δύο σημεία ώστε να υπολογιστεί ενδιάμεση κίνηση.

Όταν υπάρχουν δύο διαθέσιμες θέσεις, η μέθοδος λαμβάνει την προηγούμενη και τη νεότερη θέση του εχθρού, μαζί με τα αντίστοιχα server ticks. Στη συνέχεια υπολογίζεται η διαφορά των ticks και μετατρέπεται σε πραγματικό χρόνο με βάση τη διάρκεια κάθε server tick. Αν η διαφορά αυτή δεν είναι έγκυρη, το sprite τοποθετείται απευθείας στη νεότερη θέση. Με αυτόν τον τρόπο αποφεύγονται λάθη στην κίνηση σε περιπτώσεις όπου τα δεδομένα του server δεν μπορούν να χρησιμοποιηθούν σωστά για interpolation.

Figure 40: Υπολογισμός interpolation και ενημέρωση θέσης εχθρού

```
# Αυξάνουμε τον τοπικό χρόνο interpolation για τον συγκεκριμένο εχθρό
# Το delta_time είναι ο χρόνος που πέρασε από το προηγούμενο frame του client
t_local = self.enemy_interp_t.get(eid, 0.0) + delta_time
self.enemy_interp_t[eid] = t_local

# Υπολογίζουμε την παράμετρο interpolation στο διάστημα 0..1
# Όσο το t πλησιάζει το 1, το sprite πλησιάζει περισσότερο τη νέα θέση
t = t_local / dt_server

# Περιορίζουμε την τιμή του t ώστε να μην ξεπερνά τα όρια 0..1
if t > 1.0:
    t = 1.0
elif t < 0.0:
    t = 0.0

# Παίρνουμε τη θέση του sprite τη στιγμή που έφτασε η νέα θέση από τον server, αυτή χρησιμοποιείται ως αφετηρία για την ομαλή μετάβαση
snap_x, snap_y = self.enemy_snapshots.get(eid, (sprite.center_x, sprite.center_y))

# Ομαλή μετάβαση προς τη νέα θέση
sprite.center_x = snap_x + (x1 - snap_x) * t
sprite.center_y = snap_y + (y1 - snap_y) * t
```

Στο δεύτερο τμήμα της μεθόδου υπολογίζεται ο τοπικός χρόνος interpolation για κάθε εχθρό. Ο χρόνος αυτός αυξάνεται σε κάθε frame του client με βάση το delta_time, δηλαδή τον χρόνο που πέρασε από το προηγούμενο frame. Στη συνέχεια υπολογίζεται η παράμετρος t, η οποία παίρνει τιμές από 0 έως 1 και δείχνει την πρόοδο της μετάβασης από την προηγούμενη θέση προς τη νέα θέση που έστειλε ο server.

Η τιμή της παραμέτρου t περιορίζεται στο διάστημα 0–1, ώστε το sprite να μην ξεπεράσει τη θέση-στόχο. Έπειτα χρησιμοποιείται η snapshot θέση του εχθρού, δηλαδή η θέση που είχε το sprite τη στιγμή που έφτασε η νέα ενημέρωση από τον server. Με βάση αυτή τη θέση και τη νεότερη server position, υπολογίζεται γραμμικά η νέα θέση του sprite. Έτσι ο εχθρός δεν μετακινείται απότομα, αλλά κινείται ομαλά προς τη νέα θέση.

Με την τεχνική αυτή η οπτική κίνηση των εχθρών παραμένει ομαλή, ακόμη και όταν η λογική τους ενημερώνεται λιγότερο συχνά στον server. Επομένως, διατηρείται το όφελος της βελτιστοποίησης στην πλευρά του server, χωρίς να επηρεάζεται σημαντικά η οπτική εμπειρία του παίκτη στον client.

4.1.3.3 Αποφυγή παγίδευσης εχθρών

Για την αντιμετώπιση περιπτώσεων στις οποίες ένας εχθρός παγιδεύεται σε εμπόδια κατά την προσπάθεια καταδίωξης του στόχου, χρησιμοποιείται η μέθοδος choose_unstuck_side_towards_target(e, target_x, target_y). Σκοπός της μεθόδου είναι να επιλέξει ποια από τις δύο δυνατές πλάγιες κατευθύνσεις αποκόλλησης είναι καταλληλότερη, ώστε ο εχθρός να συνεχίσει να κινείται προς τον στόχο του με τον πιο αποδοτικό τρόπο.

Figure 41: Υπολογισμός διανύσματος κατεύθυνσης

```
# Επιλέγει ποια πλάγια κατεύθυνση unstuck (1 ή -1) φέρνει τον εχθρό πιο κοντά στον στόχο
def choose_unstuck_side_towards_target(e, target_x, target_y):
    current_x, current_y = e["x"], e["y"]
    dx = target_x - current_x
    dy = target_y - current_y

    # Ευκλείδεια απόσταση
    d = dist(current_x, current_y, target_x, target_y)

    # Αν εχθρός και στόχος είναι σχεδόν στο ίδιο σημείο, κρατάμε την προεπιλεγμένη τιμή
    if d < 1e-6:
        return 1

    # Κανονικοποιημένο διάνυσμα προς τον στόχο
    dir_x = dx / d
    dir_y = dy / d
    speed = e["move_speed"]
```

Η μέθοδος ανακτά αρχικά την τρέχουσα θέση του εχθρού και υπολογίζει τη διαφορά θέσης ως προς τον στόχο στους άξονες x και y, καθώς και την ευκλείδεια απόσταση μεταξύ τους. Αν η απόσταση αυτή είναι πρακτικά μηδενική, επιστρέφεται η προεπιλεγμένη τιμή 1, καθώς δεν απαιτείται ουσιαστική επιλογή κατεύθυνσης. Διαφορετικά, υπολογίζεται το κανονικοποιημένο διάνυσμα προς τον στόχο, δηλαδή η μοναδιαία κατεύθυνση κίνησης, και ανακτάται η ταχύτητα του εχθρού μέσω της μεταβλητής move_speed, ώστε να χρησιμοποιηθεί στα επόμενα βήματα της δοκιμαστικής κίνησης.

Στο επόμενο στάδιο, η μέθοδος εξετάζει και τις δύο πιθανές πλάγιες κατευθύνσεις αποκόλλησης. Για τον σκοπό αυτό δημιουργούνται δύο προσωρινά αντίγραφα του εχθρού, τα οποία χρησιμοποιούνται μόνο για δοκιμαστικό υπολογισμό, χωρίς να επηρεάζεται η πραγματική θέση του στο παιχνίδι. Για κάθε αντίγραφο εφαρμόζεται δοκιμαστική μετακίνηση προς μία διαφορετική κάθετη κατεύθυνση σε σχέση με το διάνυσμα προς τον στόχο, μέσω της μεθόδου move_enemy(e, delta_x, delta_y).

Μετά από κάθε δοκιμαστική μετακίνηση, υπολογίζεται η νέα απόσταση του προσωρινού εχθρού από τον στόχο. Έτσι προκύπτουν δύο αποστάσεις, μία για κάθε πιθανή πλάγια πορεία. Η μέθοδος συγκρίνει τις δύο αυτές τιμές και επιστρέφει την πλευρά που μειώνει περισσότερο την απόσταση από τον στόχο. Με τον τρόπο αυτό επιλέγεται η καταλληλότερη κατεύθυνση αποκόλλησης, ώστε ο εχθρός να ξεπερνά τα εμπόδια και να συνεχίζει την καταδίωξη όσο το δυνατόν αποτελεσματικότερα.

Figure 42: Δοκιμαστικός έλεγχος των δύο πιθανών πλάγιων κατευθύνσεων και επιλογή της καταλληλότερης πορείας αποκόλλησης

```
tmp1 = {"x": current_x, "y": current_y, "hitbox_w": e["hitbox_w"], "hitbox_h": e["hitbox_h"]}
tmp1.update(e)  # Αντιγράφουμε και τα υπόλοιπα στοιχεία που χρειάζονται για collision / movement

# Δοκιμαστική μετακίνηση στην πρώτη πλάγια κατεύθυνση
move_enemy(tmp1, (-dir_y) * speed, (dir_x) * speed)

# Παίρνουμε τη νέα δοκιμαστική θέση
x1, y1 = tmp1["x"], tmp1["y"]

# Υπολογίζουμε την απόσταση από τον στόχο μετά από αυτή τη δοκιμή
d1 = dist(target_x, target_y, x1, y1)

# Δοκιμή πλάγιας κίνησης με πλευρά -1
# Δημιουργούμε δεύτερο προσωρινό αντίγραφο του εχθρού, ώστε να ελέγξουμε την αντίθετη κάθετη κατεύθυνση
tmp2 = {"x": current_x, "y": current_y, "hitbox_w": e["hitbox_w"], "hitbox_h": e["hitbox_h"]}
tmp2.update(e)

# Δοκιμαστική μετακίνηση στη δεύτερη πλάγια κατεύθυνση
move_enemy(tmp2, (dir_y) * speed, (-dir_x) * speed)

# Παίρνουμε τη νέα δοκιμαστική θέση
x2, y2 = tmp2["x"], tmp2["y"]

# Υπολογίζουμε την απόσταση από τον στόχο μετά από αυτή τη δοκιμή
d2 = dist(target_x, target_y, x2, y2)

# Επιλέγουμε την πλευρά που μικραίνει περισσότερο την απόσταση από τον στόχο
return 1 if d1 <= d2 else -1
```

4.1.3.4 Επίθεση εχθρών

Για την υλοποίηση των επιθέσεων των εχθρών προς τους παίκτες χρησιμοποιείται η μέθοδος apply_enemy_attacks(). Η μέθοδος αυτή είναι υπεύθυνη για τον έλεγχο του αν ένας εχθρός μπορεί να επιτεθεί, καθώς και για τη διαχείριση του χρόνου ενεργοποίησης της επίθεσης πριν από την εφαρμογή της ζημιάς.

Στη μέθοδο αποθηκεύεται αρχικά η τρέχουσα χρονική στιγμή στη μεταβλητή now μέσω της time.time(), ώστε να είναι δυνατός ο έλεγχος του χρόνου επαναφόρτισης μεταξύ διαδοχικών επιθέσεων. Έπειτα, η μέθοδος διατρέχει όλους τους εχθρούς του παιχνιδιού και αγνοεί όσους βρίσκονται σε κατάσταση dead, καθώς και όσους δεν έχουν ενεργό στόχο. Επιπλέον, αν ένας εχθρός δεν βρίσκεται πλέον σε κατάσταση attack, ακυρώνεται οποιαδήποτε εκκρεμής επίθεση μέσω της μεταβλητής pending_hit_time.

Στη συνέχεια, ανακτάται ο παίκτης που αποτελεί στόχο του εχθρού και ελέγχεται αν εξακολουθεί να είναι έγκυρος. Αν ο στόχος δεν υπάρχει πλέον ή έχει πεθάνει, η μεταβλητή target μηδενίζεται και ακυρώνεται η εκκρεμής επίθεση. Παράλληλα, πραγματοποιείται έλεγχος της απόστασης μεταξύ εχθρού και παίκτη. Αν ο παίκτης έχει απομακρυνθεί πέρα από την τιμή attack_range, η επίθεση ακυρώνεται.

Figure 43: Μέθοδος επίθεσης εχθρών

```
def apply_enemy_attacks():
    now = time.time()

    for e in enemies.values():
        # Αγνοούμε νεκρούς εχθρούς ή εχθρούς χωρίς στόχο
        if e.get("dead") or e["target"] is None:
            continue

        # Αν ο εχθρός δεν είναι πλέον σε attack state, ακυρώνουμε τυχόν pending hit
        if e["state"] != "attack":
            e["pending_hit_time"] = 0.0
            continue

        tp = players.get(e["target"])

        # Αν ο στόχος δεν υπάρχει ή πέθανε, καθαρίζουμε στόχο και pending attack
        if tp is None or tp.get("dead", False):
            e["target"] = None
            e["pending_hit_time"] = 0.0
            continue

        # Αν ο παίκτης βγήκε εκτός range, ακυρώνεται το attack
        d = dist(e["x"], e["y"], tp["x"], tp["y"])
        if d > e["attack_range"]:
            e["pending_hit_time"] = 0.0
            continue

        # Έλεγχος επαναφόρτισης (cooldown)
        if now < e["next_attack_time"]:
            continue

        # Αν ξεκινά τώρα το attack, ορίζουμε πότε θα γίνει το πραγματικό hit
        if e["pending_hit_time"] <= 0.0:
            e["pending_hit_time"] = now + e["windup"]

            # Το cooldown ξεκινάει από τώρα για να μην ξεκινά πολλές επιθέσεις μαζί
            e["next_attack_time"] = now + e["attack_cooldown"]
            continue
```

Όταν ολοκληρωθεί ο απαιτούμενος χρόνος προετοιμασίας της επίθεσης και η τρέχουσα χρονική στιγμή γίνει μεγαλύτερη ή ίση από την τιμή pending_hit_time, η μέθοδος θεωρεί ότι έφτασε η στιγμή εκτέλεσης του πραγματικού χτυπήματος. Στο σημείο αυτό η τιμή της μεταβλητής pending_hit_time επαναφέρεται σε 0.0, ώστε να μην παραμένει ενεργή η εκκρεμής επίθεση.

Πριν εφαρμοστεί η ζημιά, πραγματοποιείται ένας τελικός έλεγχος της απόστασης μεταξύ εχθρού και παίκτη, ώστε να επιβεβαιωθεί ότι ο στόχος εξακολουθεί να βρίσκεται εντός της εμβέλειας επίθεσης. Με τον τρόπο αυτό αποφεύγεται η περίπτωση να δεχθεί ζημιά ένας παίκτης που έχει ήδη απομακρυνθεί από τον εχθρό.

Εφόσον ο παίκτης παραμένει εντός εμβέλειας, υπολογίζεται η τελική ζημιά που θα δεχθεί, αφαιρώντας από την τιμή ζημιάς του εχθρού (damage) την τιμή αντίστασης

(resist) του παίκτη. Η τιμή αυτή αποθηκεύεται στη μεταβλητή dmg και εφαρμόζεται στη ζωή του παίκτη. Για τη διαχείριση της ζωής χρησιμοποιούνται τόσο η τρέχουσα απόλυτη τιμή ζωής (hp_cur) όσο και η μέγιστη τιμή ζωής (hp_max), ενώ παράλληλα ενημερώνεται και η κανονικοποιημένη τιμή ζωής hp, ώστε να παραμένει συνεπής η εσωτερική αναπαράσταση της κατάστασης του παίκτη. Αν η ζωή του παίκτη μειωθεί σε μηδενική ή αρνητική τιμή, ορίζεται σε 0, ο παίκτης χαρακτηρίζεται ως νεκρός μέσω της μεταβλητής dead και η κατάστασή του αλλάζει σε death, ώστε να ενεργοποιηθεί το αντίστοιχο animation θανάτου.

Αντίθετα, όταν ο παίκτης εξακολουθεί να έχει διαθέσιμη ζωή, αυξάνεται η μεταβλητή hurt_seq, η οποία χρησιμοποιείται για την ενεργοποίηση της κατάστασης τραυματισμού (hurt). Με τον τρόπο αυτό παρέχεται άμεση οπτική ανατροφοδότηση ότι ο παίκτης δέχθηκε επιτυχημένο χτύπημα, χωρίς όμως να διακόπτεται απότομα η συνέχεια του animation.

Figure 44: Εφαρμογή ζημιάς στον παίκτη και ενημέρωση της κατάστασης ζωής του μετά από επίθεση εχθρού

```
if now >= e["pending_hit_time"]:
    e["pending_hit_time"] = 0.0

    # Τελικός έλεγχος ότι ο παίκτης είναι ακόμα εντός εμβέλειας
    d2 = dist(e["x"], e["y"], tp["x"], tp["y"])
    if d2 <= e["attack_range"]:
        # Υπολογισμός τελικής ζημιάς μετά το resist του παίκτη
        player_resist = tp.get("resist", 0)
        dmg = max(0, e["damage"] - player_resist)

        # Το hp αποθηκεύεται και ως normalized (0..1) και ως τρέχον hp
        player_hp_max = tp.get("hp_max", 100)
        hp_cur = tp.get("hp_cur", tp["hp"] * player_hp_max)

        # Εφαρμογή damage
        hp_cur -= dmg
        if hp_cur < 0:
            hp_cur = 0

        # Ενημέρωση τιμών ζωής
        tp["hp_cur"] = hp_cur
        tp["hp_max"] = player_hp_max
        tp["hp"] = hp_cur / player_hp_max

        # Αν ο παίκτης πέθανε, αλλάζουμε state
        if hp_cur <= 0:
            tp["dead"] = True
            tp["state"] = "death"
        else:
            # Διαφορετικά αυξάνουμε hurt sequence για animation
            # Θέλουμε νέο animation όταν τελειώσει το προηγούμενο και όχι ενδιάμεσα να γίνεται reset
            tp["hurt_seq"] = tp.get("hurt_seq", 0) + 1
```

4.1.4 Υλοποίηση δράκων και animations

Σε αντίθεση με τους απλούς εχθρούς, οι δράκοι δεν χρησιμοποιούν τη βασική λογική συνεχούς καταδίωξης του παίκτη. Οι απλοί εχθροί εντοπίζουν τον κοντινότερο παίκτη και κινούνται προς αυτόν μέχρι να βρεθούν σε εμβέλεια επίθεσης. Αντίθετα, οι δράκοι λειτουργούν ως ειδικοί εχθροί τύπου boss, οι οποίοι ελέγχουν έναν συγκεκριμένο χώρο του χάρτη. Όταν ενεργοποιηθούν, δεν ακολουθούν ελεύθερα τον παίκτη σε όλη την περιοχή, αλλά κινούνται οριζόντια, δηλαδή προς τα δεξιά και προς τα αριστερά, μέσα στα όρια που έχουν οριστεί για την περιοχή τους.

Η ενεργοποίηση του δράκου γίνεται όταν ο παίκτης πλησιάσει σε συγκεκριμένη απόσταση. Μετά την ενεργοποίηση, ο δράκος εισέρχεται σε ground mode, όπου κινείται στον χώρο που ελέγχει και παραμένει σε αυτή τη φάση μέχρι να δεχθεί συγκεκριμένο αριθμό χτυπημάτων. Στο ground mode η επίθεση δεν εκτελείται τυχαία. Ο δράκος επιτίθεται όταν εντοπίσει παίκτη μπροστά του μέσα στην περιοχή επίθεσης ή όταν δεχθεί χτύπημα από πίσω. Με αυτόν τον τρόπο, ο παίκτης πρέπει να προσέχει τόσο τη θέση του όσο και τη χρονική στιγμή που επιτίθεται.

Όταν ο δράκος δεχθεί τον απαιτούμενο αριθμό χτυπημάτων στο ground mode, αλλάζει φάση και περνά σε εναέρια κατάσταση. Η μετάβαση γίνεται με τη σειρά rise σε flight mode. Στην κατάσταση rise προβάλλεται το animation απογείωσης, ενώ στη συνέχεια ο δράκος μπαίνει σε flight mode, όπου βρίσκεται στον αέρα. Κατά τη διάρκεια αυτής της φάσης, οι επιθέσεις του εκτελούνται τυχαία προς τις δύο κατευθύνσεις, αριστερά και δεξιά, κάνοντας τη συμπεριφορά του λιγότερο προβλέψιμη για τον παίκτη.

Μετά από συγκεκριμένο αριθμό εναέριων επιθέσεων, ο δράκος ολοκληρώνει την air phase και επιστρέφει στο έδαφος. Η επιστροφή γίνεται με τη σειρά landing σε walk. Στην κατάσταση landing προβάλλεται το animation προσγείωσης και στη συνέχεια ο δράκος επανέρχεται σε κατάσταση walk, συνεχίζοντας την κίνησή του στο ground mode. Έτσι, ο κύκλος συμπεριφοράς του δράκου οργανώνεται σε φάσεις όπου αρχικά κινείται στο έδαφος και ελέγχει τον χώρο του, στη συνέχεια απογειώνεται μετά από συγκεκριμένα hits, επιτίθεται από τον αέρα και τελικά προσγειώνεται για να επιστρέψει ξανά στη βασική του κίνηση.

Η ύπαρξη διαφορετικών καταστάσεων και animations κάνει τον δράκο πιο σύνθετο από τους υπόλοιπους εχθρούς. Εκτός από τις βασικές καταστάσεις, όπως idle, walk, attack, hurt και death, χρησιμοποιούνται επιπλέον καταστάσεις όπως rise, flight, landing και attack_on_air. Οι καταστάσεις αυτές επιτρέπουν την οπτική αναπαράσταση της μετάβασης από το έδαφος στον αέρα και αντίστροφα, ενώ παράλληλα συνδέονται με διαφορετική λογική συμπεριφοράς. Έτσι, ο δράκος λειτουργεί ως πιο απαιτητικός εχθρός, ο οποίος διαφοροποιείται από τους απλούς εχθρούς του παιχνιδιού.

4.1.4.2 Εύρεση κοντινότερου παίκτη

Η μέθοδος find_nearest_player_in_region χρησιμοποιείται για τον εντοπισμό του κοντινότερου παίκτη που βρίσκεται στην ίδια περιοχή με τον εχθρό. Αρχικά, αρχικοποιούνται μεταβλητές για την αποθήκευση του κοντινότερου παίκτη και της μικρότερης απόστασης. Στη συνέχεια, γίνεται επανάληψη σε όλους τους παίκτες του παιχνιδιού και αγνοούνται όσοι είναι νεκροί ή βρίσκονται σε διαφορετική περιοχή από τον εχθρό. Για κάθε έγκυρο παίκτη υπολογίζεται η απόσταση από τον εχθρό και, αν αυτή είναι μικρότερη από την τρέχουσα αποθηκευμένη τιμή, ενημερώνονται οι αντίστοιχες μεταβλητές. Τέλος, η μέθοδος επιστρέφει το αναγνωριστικό του κοντινότερου παίκτη, τα στοιχεία του και την απόστασή του από τον εχθρό. Με τον τρόπο αυτό, η μέθοδος αποτελεί βασικό μέρος της λογικής εντοπισμού στόχου από τους εχθρούς.

Figure 45: Μέθοδος εύρεσης του κοντινότερου παίκτη στην ίδια περιοχή

```python
# Μέθοδος που βρίσκει τον κοντινότερο παίκτη που βρίσκεται στην ίδια περιοχή με τον εχθρό
def find_nearest_player_in_region(e, players, dist):
    nearest_pid = None      # Id του κοντινότερου παίκτη που βρίσκεται μέσα στην ίδια περιοχή
    nearest_player = None   # Δεδομένα κοντινότερου παίκτη
    nearest_d = 1e9         # Αρχική μεγάλη τιμή ώστε να βρεθεί σίγουρα μικρότερη από αυτήν

    for pid, p in players.items():
        if p.get("dead", False):    # Αγνοούμε παίκτες που είναι νεκροί
            continue

        if p.get("region") != e.get("region"):  # Αγνοούμε παίκτες που βρίσκονται σε άλλη περιοχή
            continue

        d = dist(e["x"], e["y"], p["x"], p["y"])    # Υπολογίζουμε την απόσταση εχθρού-παίκτη

        # Κρατάμε τον παίκτη με τη μικρότερη απόσταση
        if d < nearest_d:
            nearest_d = d
            nearest_pid = pid
            nearest_player = p

    return nearest_pid, nearest_player, nearest_d   # Επιστροφή κοντινότερου παίκτη με βάση: id παίκτη, στοιχεία παίκτη, απόσταση
```

4.1.4.2 Καταστάσεις

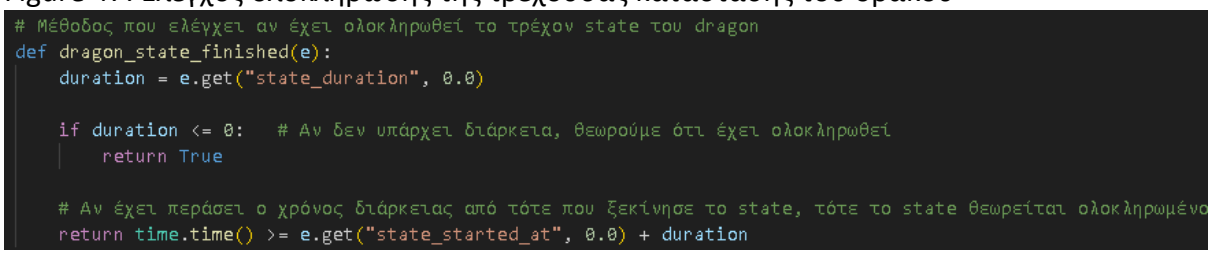
Η μέθοδος set_dragon_state χρησιμοποιείται για την αλλαγή της κατάστασης του δράκου. Αρχικά καταγράφεται η τρέχουσα χρονική στιγμή, ώστε να αποθηκευτεί πότε ξεκίνησε το νέο state. Αν δοθεί νέα κατεύθυνση, ενημερώνεται και η κατεύθυνση του δράκου. Στη συνέχεια ελέγχεται αν ο δράκος βρίσκεται ήδη στην ίδια κατάσταση. Σε αυτή την περίπτωση δεν γίνεται επανεκκίνηση του animation, ώστε να αποφεύγεται άσκοπο reset. Αν η κατάσταση είναι διαφορετική, ενημερώνεται το νέο state, αποθηκεύεται ο χρόνος έναρξης, υπολογίζεται η διάρκεια του αντίστοιχου animation με βάση τα frames του state και μηδενίζεται το flag εφαρμογής ζημιάς.

Figure 46: Μέθοδος αλλαγής κατάστασης του δράκου

```python
# Μέθοδος που αλλάζει το state του dragon
def set_dragon_state(e, new_state, direction=None):
    now = time.time()

    if direction is not None:   # Αν δόθηκε νέα κατεύθυνση, ενημερώνουμε το direction του dragon
        e["dir"] = direction

    if e.get("state") == new_state:    # Αν ο dragon είναι ήδη στο ίδιο state, δεν κάνουμε reset το animation
        return

    e["state"] = new_state          # Ορίζουμε το νέο state
    e["state_started_at"] = now     # Αποθηκεύουμε τη χρονική στιγμή που ξεκίνησε το state
    e["state_duration"] = get_enemy_state_duration(e["type"], new_state)    # Υπολογίζουμε τη διάρκεια του animation με βάση τα frames του state
    e["damage_applied"] = False     # Reset του flag ζημιάς όταν αλλάζει state
```

Η μέθοδος dragon_state_finished ελέγχει αν έχει ολοκληρωθεί η τρέχουσα κατάσταση του δράκου. Αρχικά παίρνει τη διάρκεια του state από τα δεδομένα του δράκου. Αν δεν υπάρχει διάρκεια ή η τιμή είναι μηδενική, τότε θεωρείται ότι η κατάσταση έχει ολοκληρωθεί. Διαφορετικά, συγκρίνεται η τρέχουσα χρονική στιγμή με τον χρόνο έναρξης της κατάστασης και τη διάρκειά της. Όταν περάσει ο απαιτούμενος χρόνος, η μέθοδος επιστρέφει ότι το state έχει ολοκληρωθεί, ώστε ο δράκος να μπορεί να περάσει στην επόμενη κατάσταση, όπως από rise σε flight ή από landing σε walk.

Figure 47: Έλεγχος ολοκλήρωσης της τρέχουσας κατάστασης του δράκου

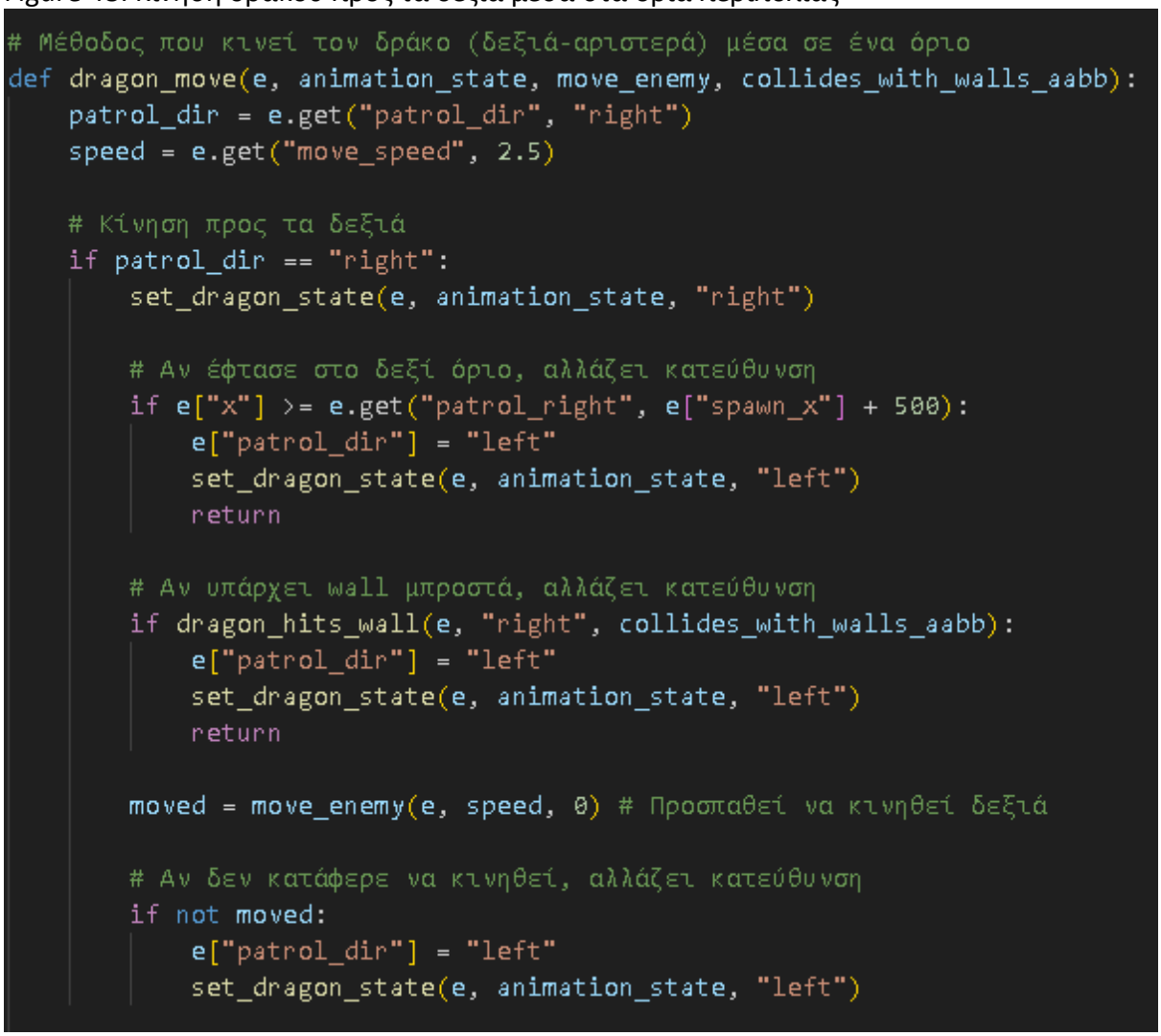


```python
# Μέθοδος που ελέγχει αν έχει ολοκληρωθεί το τρέχον state του dragon
def dragon_state_finished(e):
    duration = e.get("state_duration", 0.0)

    if duration <= 0:   # Αν δεν υπάρχει διάρκεια, θεωρούμε ότι έχει ολοκληρωθεί
        return True

    # Αν έχει περάσει ο χρόνος διάρκειας από τότε που ξεκίνησε το state, τότε το state θεωρείται ολοκληρωμένο
    return time.time() >= e.get("state_started_at", 0.0) + duration
```

4.1.4.3 Κίνηση

Η μέθοδος dragon_move υλοποιεί την οριζόντια κίνηση του δράκου μέσα στον χώρο που ελέγχει. Σε αντίθεση με τους απλούς εχθρούς, ο δράκος δεν κινείται ελεύθερα προς τον παίκτη, αλλά περιπολεί δεξιά και αριστερά ανάμεσα σε δύο προκαθορισμένα όρια, τα patrol_left και patrol_right. Η τρέχουσα κατεύθυνση περιπολίας αποθηκεύεται στη μεταβλητή patrol_dir, ενώ η ταχύτητα κίνησης προκύπτει από το move_speed.

Figure 48: Κίνηση δράκου προς τα δεξιά μέσα στα όρια περιπολίας



```python
# Μέθοδος που κινεί τον δράκο (δεξιά-αριστερά) μέσα σε ένα όριο
def dragon_move(e, animation_state, move_enemy, collides_with_walls_aabb):
    patrol_dir = e.get("patrol_dir", "right")
    speed = e.get("move_speed", 2.5)

    # Κίνηση προς τα δεξιά
    if patrol_dir == "right":
        set_dragon_state(e, animation_state, "right")

        # Αν έφτασε στο δεξί όριο, αλλάζει κατεύθυνση
        if e["x"] >= e.get("patrol_right", e["spawn_x"] + 500):
            e["patrol_dir"] = "left"
            set_dragon_state(e, animation_state, "left")
            return

        # Αν υπάρχει wall μπροστά, αλλάζει κατεύθυνση
        if dragon_hits_wall(e, "right", collides_with_walls_aabb):
            e["patrol_dir"] = "left"
            set_dragon_state(e, animation_state, "left")
            return

        moved = move_enemy(e, speed, 0) # Προσπαθεί να κινηθεί δεξιά

        # Αν δεν κατάφερε να κινηθεί, αλλάζει κατεύθυνση
        if not moved:
            e["patrol_dir"] = "left"
            set_dragon_state(e, animation_state, "left")
```

Στο πρώτο τμήμα της μεθόδου ελέγχεται η κίνηση προς τα δεξιά. Αν ο δράκος κινείται δεξιά, ενημερώνεται το animation state και η κατεύθυνσή του. Στη συνέχεια ελέγχεται αν έχει φτάσει στο δεξί όριο της περιοχής περιπολίας ή αν υπάρχει εμπόδιο μπροστά του. Σε οποιαδήποτε από αυτές τις περιπτώσεις, η κατεύθυνση αλλάζει σε αριστερά, ώστε ο δράκος να συνεχίσει την κίνησή του προς την αντίθετη πλευρά. Αν δεν υπάρχει εμπόδιο και δεν έχει φτάσει στο όριο, γίνεται προσπάθεια μετακίνησης προς τα δεξιά.

Figure 49: Κίνηση δράκου προς τα αριστερά μέσα στα όρια περιπολίας

```
# Κίνηση προς τα αριστερά
else:
    set_dragon_state(e, animation_state, "left")

    # Αν έφτασε στο αριστερό όριο, αλλάζει κατεύθυνση
    if e["x"] <= e.get("patrol_left", e["spawn_x"] - 500):
        e["patrol_dir"] = "right"
        set_dragon_state(e, animation_state, "right")
        return

    # Αν υπάρχει wall μπροστά, αλλάζει κατεύθυνση
    if dragon_hits_wall(e, "left", collides_with_walls_aabb):
        e["patrol_dir"] = "right"
        set_dragon_state(e, animation_state, "right")
        return

    moved = move_enemy(e, -speed, 0)    # Προσπαθεί να κινηθεί αριστερά

    # Αν δεν κατάφερε να κινηθεί, αλλάζει κατεύθυνση
    if not moved:
        e["patrol_dir"] = "right"
        set_dragon_state(e, animation_state, "right")
```

Στο δεύτερο τμήμα εφαρμόζεται η αντίστοιχη λογική για την κίνηση προς τα αριστερά. Ο δράκος ελέγχει αν έχει φτάσει στο αριστερό όριο ή αν υπάρχει εμπόδιο μπροστά του. Αν συμβεί κάτι από αυτά, η κατεύθυνση αλλάζει σε δεξιά. Αν η μετακίνηση αποτύχει για οποιονδήποτε λόγο, ο δράκος επίσης αλλάζει κατεύθυνση, ώστε να μη μένει κολλημένος στο ίδιο σημείο.

Με αυτόν τον τρόπο, ο δράκος παραμένει μέσα στα όρια της περιοχής που ελέγχει και κινείται συνεχώς δεξιά-αριστερά, λειτουργώντας ως εχθρός που περιπολεί συγκεκριμένο χώρο αντί να καταδιώκει συνεχώς τον παίκτη.

4.1.4.4 Επίθεση

Η μέθοδος dragon_damage_player είναι υπεύθυνη για την εφαρμογή της ζημιάς που προκαλεί ο δράκος στον παίκτη. Αρχικά λαμβάνεται υπόψη η άμυνα του παίκτη μέσω της τιμής resist. Αν δεν έχει δοθεί συγκεκριμένη τιμή ζημιάς, χρησιμοποιείται η βασική ζημιά του δράκου από τα δεδομένα του enemy. Στη συνέχεια υπολογίζεται η τελική ζημιά αφαιρώντας το resist του παίκτη, ενώ με τη χρήση της max διασφαλίζεται

ότι η ζημιά δεν μπορεί να γίνει αρνητική.

Figure 50: Εφαρμογή ζημιάς του δράκου στον παίκτη

```python
# Μέθοδος που εφαρμόζει τη ζημιά του δράκου προς τον παίκτη
def dragon_damage_player(p, e, damage_amount=None):
    player_resist = p.get("resist", 0)

    # Αν δεν δοθεί συγκεκριμένο damage, χρησιμοποιείται το damage του enemy
    if damage_amount is None:
        damage_amount = e.get("damage", 1)

    dmg = max(0, damage_amount - player_resist) # Τελικό damage μετά το resist του παίκτη

    # Παίρνουμε το μέγιστο και το τρέχον HP του παίκτη
    player_hp_max = p.get("hp_max", 100)
    hp_cur = p.get("hp_cur", p.get("hp", 1.0) * player_hp_max)

    hp_cur -= dmg   # Αφαιρούμε το damage

    # Δεν αφήνουμε το HP να πέσει κάτω από 0
    if hp_cur < 0:
        hp_cur = 0

    # Ενημερώνουμε και το πραγματικό HP και το normalized HP για το UI
    p["hp_cur"] = hp_cur
    p["hp_max"] = player_hp_max
    p["hp"] = hp_cur / player_hp_max

    # Αν ο παίκτης πέθανε, αλλάζουμε state σε death
    if hp_cur <= 0:
        p["dead"] = True
        p["state"] = "death"
    else:
        p["hurt_seq"] = p.get("hurt_seq", 0) + 1    # Αλλιώς αυξάνουμε το hurt_seq ώστε ο client να δείξει hurt animation
```

Έπειτα η μέθοδος ανακτά το μέγιστο και το τρέχον HP του παίκτη και αφαιρεί την τελική ζημιά από τη ζωή του. Αν το HP πέσει κάτω από το μηδέν, περιορίζεται στο 0, ώστε να μην υπάρχουν αρνητικές τιμές ζωής. Στη συνέχεια ενημερώνονται τόσο το πραγματικό HP του παίκτη (hp_cur) όσο και η κανονικοποιημένη τιμή HP (hp) που χρησιμοποιείται για την εμφάνιση της μπάρας ζωής στο UI.

Τέλος, ελέγχεται αν ο παίκτης έχει πεθάνει. Αν η ζωή του μηδενιστεί, η μεταβλητή dead γίνεται True και το state του παίκτη αλλάζει σε death. Αν ο παίκτης παραμένει ζωντανός, αυξάνεται η τιμή hurt_seq, ώστε ο client να εμφανίσει την οπτική ανατροφοδότηση τραυματισμού, όπως το blinking effect.

Η μέθοδος ground_attack_target χρησιμοποιείται για να ελέγξει αν υπάρχει παίκτης μπροστά από τον δράκο, ώστε να εκτελεστεί επίθεση στο έδαφος. Αρχικά, η μέθοδος παίρνει τις παραμέτρους της επίθεσης από τα δεδομένα του δράκου, όπως την εμβέλεια της επίθεσης (ground_attack_range), το πλάτος της περιοχής επίθεσης (ground_attack_width) και την κατεύθυνση στην οποία κοιτάει ο δράκος.

Στη συνέχεια, η μέθοδος διατρέχει όλους τους παίκτες και αγνοεί όσους είναι νεκροί ή βρίσκονται σε διαφορετική περιοχή από τον δράκο. Για κάθε έγκυρο παίκτη υπολογίζεται η σχετική θέση του ως προς τον δράκο στους άξονες X και Y. Ο παίκτης πρέπει να βρίσκεται περίπου στο ίδιο ύψος με τον δράκο, δηλαδή μέσα στο επιτρεπτό πλάτος της επίθεσης, ώστε να μπορεί να θεωρηθεί στόχος.

Figure 51: Εντοπισμός στόχου για την επίθεση εδάφους του δράκου

```python
# Μέθοδος που βρίσκει αν υπάρχει παίκτης μπροστά από τον dragon, ώστε να εκτελέσει επίθεση στο έδαφος
def ground_attack_target(e, players):
    defs = get_enemy_type_defs(e["type"])

    attack_range = defs.get("ground_attack_range", e.get("attack_range", 110))  # Απόσταση που φτάνει η επίθεση του dragon στο έδαφος
    attack_half_width = defs.get("ground_attack_width", 90) / 2     # Πλάτος που μπορεί να βρίσκεται ο παίκτης για να χτυπηθεί
    direction = e.get("dir", "right")   # Κατεύθυνση που κοιτάει ο dragon

    best_pid = None     # Id του κοντινότερου παίκτη που βρίσκεται μέσα στην περιοχή επίθεσης
    best_player = None  # Δεδομένα του κοντινότερου παίκτη
    best_d = 1e9        # Αρχική μεγάλη απόσταση για σύγκριση με τους παίκτες που θα ελεγχθούν

    for pid, p in players.items():
        if p.get("dead", False):    # Αγνοούμε νεκρούς παίκτες
            continue

        if p.get("region") != e.get("region"):  # Αγνοούμε παίκτες που βρίσκονται σε άλλη περιοχή
            continue

        # Υπολογίζουμε τη θέση του παίκτη σε σχέση με τον dragon
        dx = p["x"] - e["x"]
        dy = p["y"] - e["y"]

        if abs(dy) > attack_half_width:     # Ο παίκτης πρέπει να είναι περίπου στο ίδιο ύψος με τον dragon
            continue

        if direction == "right":            # Αν ο dragon κοιτάει δεξιά, μπορεί να χτυπήσει μόνο παίκτες δεξιά του
            if dx <= 0 or dx > attack_range:
                continue
            d = dx

        elif direction == "left":           # Αν ο dragon κοιτάει αριστερά, μπορεί να χτυπήσει μόνο παίκτες αριστερά του
            if dx >= 0 or abs(dx) > attack_range:
                continue
            d = abs(dx)

        else:
            continue

        # Κρατάμε τον πιο κοντινό παίκτη μέσα στην περιοχή επίθεσης
        if d < best_d:
            best_d = d
            best_pid = pid
            best_player = p

    return best_pid, best_player, direction # Επιστρέφουμε τον παίκτη που μπορεί να χτυπηθεί και την κατεύθυνση του attack
```

Έπειτα γίνεται έλεγχος με βάση την κατεύθυνση του δράκου. Αν ο δράκος κοιτάει δεξιά, ο παίκτης πρέπει να βρίσκεται δεξιά του και μέσα στην επιτρεπτή εμβέλεια. Αν κοιτάει αριστερά, ο παίκτης πρέπει να βρίσκεται αριστερά του. Αν ο παίκτης δεν βρίσκεται στη σωστή κατεύθυνση ή είναι εκτός εμβέλειας, αγνοείται.

Αν υπάρχουν περισσότεροι από ένας παίκτες μέσα στην περιοχή επίθεσης, επιλέγεται ο κοντινότερος. Η μέθοδος επιστρέφει το αναγνωριστικό του παίκτη, τα δεδομένα του και την κατεύθυνση της επίθεσης. Με αυτόν τον τρόπο ο δράκος εκτελεί επίθεση εδάφους μόνο όταν υπάρχει έγκυρος στόχος μπροστά του, αντί να επιτίθεται τυχαία.

Η μέθοδος player_is_behind_dragon ελέγχει αν ο παίκτης βρίσκεται πίσω από τον δράκο τη στιγμή που τον χτυπά. Ο έλεγχος γίνεται με βάση τη θέση του παίκτη στον άξονα X και την κατεύθυνση στην οποία κοιτάει ο δράκος. Αν ο δράκος κοιτάει προς τα δεξιά, τότε πίσω του θεωρείται η αριστερή πλευρά, άρα ο παίκτης βρίσκεται πίσω του όταν η τιμή dx είναι μικρότερη από 0. Αντίστοιχα, όταν ο δράκος κοιτάει προς τα αριστερά, πίσω του θεωρείται η δεξιά πλευρά, οπότε ο παίκτης βρίσκεται πίσω του όταν η τιμή dx είναι μεγαλύτερη από 0.

Figure 52: Έλεγχος χτυπήματος του δράκου από πίσω

```
# Μέθοδος που ελέγχει αν ο παίκτης βρίσκεται πίσω από τον dragon
def player_is_behind_dragon(p, e):
    dx = p["x"] - e["x"]
    dragon_dir = e.get("dir", "right")

    if dragon_dir == "right":   # Αν ο dragon κοιτάει δεξιά, τότε πίσω του είναι η αριστερή πλευρά
        return dx < 0

    if dragon_dir == "left":    # Αν ο dragon κοιτάει αριστερά, τότε πίσω του είναι η δεξιά πλευρά
        return dx > 0

    return False
```

Η μέθοδος direction_from_dragon_to_player επιστρέφει την κατεύθυνση προς την οποία πρέπει να στραφεί ο δράκος για να κοιτάξει τον παίκτη. Αν ο παίκτης βρίσκεται δεξιά από τον δράκο, επιστρέφεται η τιμή right, ενώ αν βρίσκεται αριστερά επιστρέφεται η τιμή left. Η μέθοδος αυτή χρησιμοποιείται όταν ο δράκος δεχθεί χτύπημα από πίσω, ώστε να μπορεί να γυρίσει προς την πλευρά του παίκτη και να εκτελέσει επίθεση.

Figure 53: Υπολογισμός κατεύθυνσης του δράκου προς τον παίκτη

```
# Μέθοδος που επιστρέφει την κατεύθυνση του dragon προς τον παίκτη
# Χρησιμοποιείται όταν ο dragon χτυπηθεί από πίσω, ώστε να γυρίσει προς τον παίκτη και να κάνει attack
def direction_from_dragon_to_player(e, p):
    return "right" if p["x"] >= e["x"] else "left"
```

Η μέθοδος apply_dragon_ground_attack εφαρμόζει τη ζημιά της επίθεσης εδάφους του δράκου στους παίκτες που βρίσκονται μέσα στην περιοχή χτυπήματος. Αρχικά ελέγχει αν η ζημιά έχει ήδη εφαρμοστεί για το συγκεκριμένο attack animation, ώστε να μην χτυπήσει ο ίδιος παίκτης περισσότερες από μία φορές στην ίδια επίθεση.

Στη συνέχεια, η μέθοδος υπολογίζει τη χρονική στιγμή στην οποία πρέπει να εφαρμοστεί το πραγματικό χτύπημα. Αυτό γίνεται με βάση το ground_attack_impact_frame, δηλαδή το frame του animation στο οποίο θεωρείται ότι η επίθεση πετυχαίνει τον στόχο. Αν δεν έχει φτάσει ακόμη αυτή η χρονική στιγμή, η μέθοδος δεν εφαρμόζει ζημιά και περιμένει.

Όταν φτάσει η στιγμή του χτυπήματος, ελέγχονται όλοι οι παίκτες. Αγνοούνται οι νεκροί παίκτες και όσοι βρίσκονται σε διαφορετική περιοχή από τον δράκο. Για κάθε έγκυρο παίκτη υπολογίζεται η σχετική του θέση ως προς τον δράκο. Ο παίκτης πρέπει να βρίσκεται περίπου στο ίδιο ύψος με τον δράκο, μέσα στο πλάτος της επίθεσης, και ταυτόχρονα στη σωστή κατεύθυνση, δηλαδή μπροστά από τον δράκο.

Αν ο δράκος κοιτάει δεξιά, μπορεί να χτυπήσει μόνο παίκτες που βρίσκονται δεξιά του και μέσα στην εμβέλεια της επίθεσης. Αν κοιτάει αριστερά, μπορεί να χτυπήσει μόνο παίκτες που βρίσκονται αριστερά του. Όταν ένας παίκτης πληροί αυτές τις προϋποθέσεις, καλείται η μέθοδος dragon_damage_player, ώστε να αφαιρεθεί η αντίστοιχη ζωή από τον παίκτη.

Figure 54: Εφαρμογή επίθεσης εδάφους του δράκου στον παίκτη

```python
def apply_dragon_ground_attack(e, players):
    if e.get("damage_applied", False):  # Αν έχει ήδη εφαρμοστεί ζημιά για αυτό το attack animation, δεν κάνουμε τίποτα
        return

    defs = get_enemy_type_defs(e["type"])
    now = time.time()

    impact_frame = defs.get("ground_attack_impact_frame", 6)            # Frame του animation στο οποίο γίνεται το πραγματικό hit
    impact_time = e.get("state_started_at", now) + impact_frame * 0.12  # Υπολογίζουμε τη χρονική στιγμή που πρέπει να εφαρμοστεί η ζημιά

    if now < impact_time:   # Αν δεν έχει φτάσει ακόμα η στιγμή του hit, περιμένουμε
        return

    attack_range = defs.get("ground_attack_range", e.get("attack_range", 110))  # Απόσταση που φτάνει η επίθεση μπροστά από τον dragon
    attack_half_width = defs.get("ground_attack_width", 90) / 2                 # Πλάτος που μπορεί να βρίσκεται ο παίκτης για να χτυπηθεί
    direction = e.get("dir", "right")   # Κατεύθυνση στην οποία κοιτάει ο dragon

    for p in players.values():
        if p.get("dead", False):    # Αγνοούμε νεκρούς παίκτες
            continue

        if p.get("region") != e.get("region"):  # Αγνοούμε παίκτες που βρίσκονται σε άλλη περιοχή
            continue

        # Υπολογίζουμε τη θέση του παίκτη σε σχέση με τον dragon
        dx = p["x"] - e["x"]
        dy = p["y"] - e["y"]

        # Αν ο παίκτης δεν είναι στο ίδιο ύψος, δεν μπορεί να χτυπηθεί από το ground attack
        if abs(dy) > attack_half_width:
            continue

        can_hit = False

        if direction == "right":    # Αν ο dragon κοιτάει δεξιά, χτυπάει μόνο παίκτες δεξιά του
            can_hit = dx > 0 and dx <= attack_range
        elif direction == "left":   # Αν ο dragon κοιτάει αριστερά, χτυπάει μόνο παίκτες αριστερά του
            can_hit = dx < 0 and abs(dx) <= attack_range

        # Αν ο παίκτης βρίσκεται μέσα στην περιοχή επίθεσης, εφαρμόζουμε τη ζημιά
        if can_hit:
            dragon_damage_player(p, e)

    e["damage_applied"] = True  # Το flag ενημερώνεται ότι η ζημιά εφαρμόστηκε, ώστε να μη χτυπήσει ξανά στο ίδιο attack animation
```

Τέλος, ενημερώνεται το flag damage_applied, ώστε η ζημιά να μην εφαρμοστεί ξανά κατά τη διάρκεια του ίδιου animation επίθεσης. Με αυτόν τον τρόπο η επίθεση του δράκου συγχρονίζεται με το κατάλληλο frame του animation και αποφεύγεται η πολλαπλή εφαρμογή ζημιάς στο ίδιο χτύπημα.

Η μέθοδος apply_dragon_air_attack εφαρμόζει τη ζημιά της εναέριας επίθεσης του δράκου. Όπως και στην επίθεση εδάφους, αρχικά ελέγχει αν η ζημιά έχει ήδη εφαρμοστεί στο συγκεκριμένο attack animation, ώστε να αποφευχθεί η πολλαπλή εφαρμογή ζημιάς στην ίδια επίθεση.

Στη συνέχεια υπολογίζεται η χρονική στιγμή στην οποία πρέπει να εφαρμοστεί το χτύπημα, με βάση το air_attack_impact_frame. Αν το animation δεν έχει φτάσει ακόμη στο συγκεκριμένο frame, η μέθοδος δεν εφαρμόζει ζημιά και περιμένει μέχρι να έρθει η σωστή χρονική στιγμή.

Η περιοχή της εναέριας επίθεσης καθορίζεται από την εμβέλεια air_attack_range και το πλάτος air_attack_width. Επιπλέον, επειδή το οπτικό εφέ της εναέριας επίθεσης εμφανίζεται χαμηλότερα από το κέντρο του δράκου, χρησιμοποιείται το air_attack_y_offset, ώστε η περιοχή χτυπήματος να ευθυγραμμίζεται καλύτερα με το animation.

Figure 55: Εφαρμογή εναέριας επίθεσης του δράκου στον παίκτη

```
def apply_dragon_air_attack(e, players):
    if e.get("damage_applied", False):  # Αν έχει ήδη εφαρμοστεί ζημιά για αυτό το air attack animation, δεν κάνουμε τίποτα
        return

    defs = get_enemy_type_defs(e["type"])
    now = time.time()

    impact_frame = defs.get("air_attack_impact_frame", 8)   # Frame του animation στο οποίο γίνεται το πραγματικό hit
    impact_time = e.get("state_started_at", now) + impact_frame * 0.12  # Υπολογίζουμε τη χρονική στιγμή που πρέπει να εφαρμοστεί η ζημιά

    if now < impact_time:   # Αν δεν έχει φτάσει ακόμα η στιγμή του hit, περιμένουμε
        return

    attack_range = defs.get("air_attack_range", 300)            # Απόσταση που φτάνει η επίθεση μπροστά από τον dragon
    attack_half_width = defs.get("air_attack_width", 120) / 2   # Πλάτος που μπορεί να βρίσκεται ο παίκτης για να χτυπηθεί
    direction = e.get("dir", "right")       # Κατεύθυνση στην οποία κοιτάει ο dragon

    # Το οπτικό air attack φαίνεται πιο κάτω από το κέντρο του dragon και για αυτό βάζουμε offset για τον Y άξονα
    attack_y = e["y"] + defs.get("air_attack_y_offset", 0)

    for p in players.values():
        if p.get("dead", False):    # Αγνοούμε νεκρούς παίκτες
            continue

        if p.get("region") != e.get("region"):  # Αγνοούμε παίκτες που βρίσκονται σε άλλη περιοχή
            continue

        dx = p["x"] - e["x"]    # Υπολογίζουμε τη θέση του παίκτη σε σχέση με τον dragon στον X άξονα
        dy = p["y"] - attack_y  # Υπολογίζουμε τη θέση του παίκτη σε σχέση με το κέντρο της air attack περιοχής

        # Αν ο παίκτης δεν είναι στο ύψος της επίθεσης, δεν μπορεί να χτυπηθεί από το air attack
        if abs(dy) > attack_half_width:
            continue

        can_hit = False

        if direction == "right":    # Αν ο dragon κοιτάει δεξιά, χτυπάει μόνο παίκτες δεξιά του
            can_hit = dx > 0 and dx <= attack_range
        elif direction == "left":   # Αν ο dragon κοιτάει αριστερά, χτυπάει μόνο παίκτες αριστερά του
            can_hit = dx < 0 and abs(dx) <= attack_range

        # Αν ο παίκτης βρίσκεται μέσα στην περιοχή επίθεσης, εφαρμόζουμε τη ζημιά
        if can_hit:
            dragon_damage_player(p, e)

    e["damage_applied"] = True  # Το flag ενημερώνεται ότι η ζημιά εφαρμόστηκε, ώστε να μη χτυπήσει ξανά στο ίδιο attack animation
```

Έπειτα, η μέθοδος ελέγχει όλους τους παίκτες που βρίσκονται στην ίδια περιοχή με τον δράκο και αγνοεί όσους είναι νεκροί. Για κάθε παίκτη υπολογίζεται η σχετική του θέση ως προς την περιοχή της επίθεσης. Ο παίκτης πρέπει να βρίσκεται μέσα στο κατάλληλο πλάτος στον άξονα Y και στη σωστή κατεύθυνση στον άξονα X, ανάλογα με το αν ο δράκος κοιτάει δεξιά ή αριστερά.

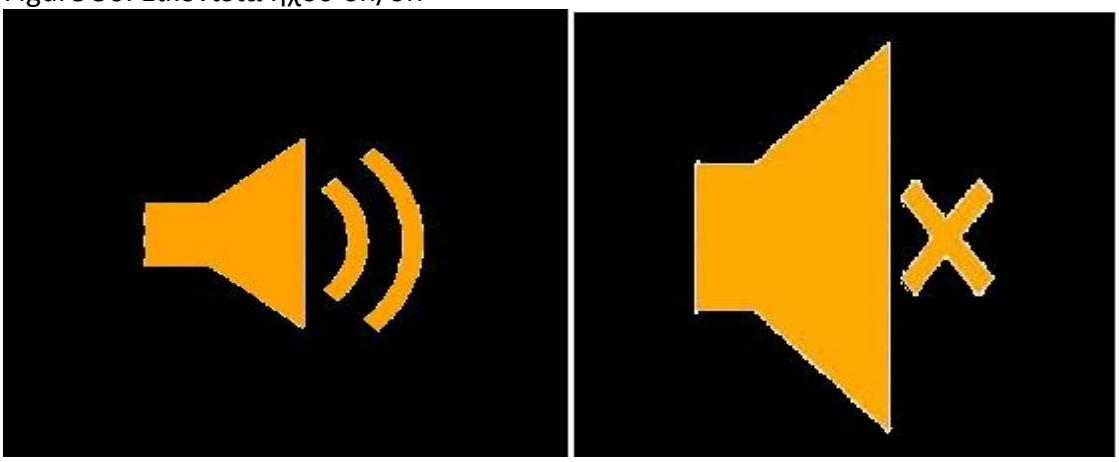
Αν ο παίκτης βρίσκεται μέσα στα όρια της εναέριας επίθεσης, καλείται η μέθοδος dragon_damage_player, ώστε να εφαρμοστεί η ζημιά. Τέλος, ενημερώνεται το flag damage_applied, ώστε η ζημιά να μην εφαρμοστεί ξανά στο ίδιο animation επίθεσης.

4.2 Ήχος

Για την υλοποίηση του ηχητικού μέρους του παιχνιδιού ενσωματώθηκαν μουσικά κομμάτια και ηχητικά εφέ, τα οποία διατίθενται δωρεάν από την πλατφόρμα OpenGameArt. Ο ήχος χρησιμοποιείται τόσο στο μενού όσο και κατά τη διάρκεια του παιχνιδιού, συμβάλλοντας στη βελτίωση της εμπειρίας του παίκτη και στην παροχή ακουστικής ανατροφοδότησης.

Στο μενού του παιχνιδιού χρησιμοποιείται μουσική υπόκρουση, καθώς και ηχητικά εφέ για τις επιλογές που πραγματοποιεί ο παίκτης, όπως για παράδειγμα κατά την επιλογή δημιουργίας νέου παίκτη (new player). Επιπλέον, παρέχεται δυνατότητα ενεργοποίησης και απενεργοποίησης του ήχου μέσω ειδικού εικονιδίου (on/off icon), το οποίο εναλλάσσεται ανάλογα με την επιλογή του παίκτη. Η προεπιλεγμένη κατάσταση του ήχου είναι ενεργοποιημένη, ενώ ο παίκτης μπορεί να απενεργοποιήσει τόσο τη μουσική όσο και τα ηχητικά εφέ του μενού.

Figure 56: Εικονίδια ήχου on/off



Κατά τη διάρκεια του gameplay, η μουσική προσαρμόζεται δυναμικά ανάλογα με την κατάσταση του κόσμου. Όταν ο παίκτης βρίσκεται σε κατάσταση μάχης, η μουσική γίνεται πιο έντονη, ενισχύοντας την αίσθηση έντασης, ενώ σε περιόδους εκτός μάχης χρησιμοποιείται πιο χαλαρή μουσική υπόκρουση. Με τον τρόπο αυτό επιτυγχάνεται καλύτερη σύνδεση του ήχου με τη ροή και την ατμόσφαιρα του παιχνιδιού.

Παράλληλα, υλοποιήθηκαν ηχητικά εφέ που συγχρονίζονται με τα animations του παίκτη και των εχθρών, καθώς και με τις επιθέσεις τους. Τα ηχητικά αυτά εφέ ενεργοποιούνται κατά την εκτέλεση αντίστοιχων ενεργειών, προσφέροντας άμεση ακουστική ανατροφοδότηση και ενισχύοντας την αίσθηση αλληλεπίδρασης κατά τη διάρκεια του παιχνιδιού.

4.3 Gameplay

Στην αρχή του παιχνιδιού οι παίκτες εμφανίζονται σε ένα κοινό σημείο

εκκίνησης, το οποίο έχει οριστεί στο object layer του χάρτη μέσω του εργαλείου Tiled. Η εκκίνηση πραγματοποιείται στην πρώτη περιοχή του παιχνιδιού, η οποία αντιστοιχεί σε ένα παγωμένο περιβάλλον και αναφέρεται ως μπλε περιοχή.

Οι παίκτες καλούνται να συνεργαστούν και να επιβιώσουν στις περιοχές του χάρτη, με στόχο την αναβάθμιση του εξοπλισμού τους και την εξόντωση του τελικού εχθρού, η οποία αποτελεί και τη συνθήκη ολοκλήρωσης του παιχνιδιού.

Η συνεργασία μεταξύ των παικτών επιτυγχάνεται μέσω της ομαδικής εξερεύνησης των περιοχών και της αριθμητικής υπεροχής απέναντι στους εχθρούς. Ωστόσο, η προσέγγιση αυτή οδηγεί σε πιο αργή εξέλιξη, καθώς η εμπειρία που αποκτάται μοιράζεται μεταξύ περισσότερων παικτών. Αντίθετα, η κίνηση σε μικρότερες ομάδες επιτρέπει ταχύτερη απόκτηση εμπειρίας και γρηγορότερη εξέλιξη, αλλά ενέχει αυξημένο κίνδυνο, καθώς οι εχθροί ενδέχεται να υπερτερούν αριθμητικά.

Σε περίπτωση θανάτου όλων των παικτών μιας ομάδας, η πρόοδός τους χάνεται και το παιχνίδι ξεκινά εκ νέου από την αρχή.

4.3.1 Εξυπηρετητής (Server)

Ο εξυπηρετητής είναι υπεύθυνος για τη διαχείριση της σύνδεσης των παικτών, καθώς και για τη συνολική κατάσταση του παιχνιδιού και των οντοτήτων του, όπως παίκτες και εχθροί. Συγκεκριμένα, ο εξυπηρετητής ελέγχει τη λογική του παιχνιδιού, όπως τις συγκρούσεις των χαρακτήρων με τους τοίχους του χάρτη και τον υπολογισμό της ζημιάς που δέχονται οι παίκτες και οι εχθροί.

Τα δεδομένα εισόδου (input) που αποστέλλονται από κάθε παίκτη αφορούν την κίνηση του χαρακτήρα και την ενεργοποίηση ικανοτήτων ή επιθέσεων. Τα δεδομένα αυτά επεξεργάζονται από τον εξυπηρετητή σε κάθε tick, ο οποίος τα συνδυάζει με την προηγούμενη κατάσταση του παιχνιδιού, προκειμένου να υπολογίσει τη νέα κατάσταση των οντοτήτων. Με τον τρόπο αυτό διασφαλίζεται ότι όλες οι ενέργειες των παικτών εφαρμόζονται με συνεπή και συγχρονισμένο τρόπο σε όλους τους συνδεδεμένους πελάτες (clients).

Ο όρος tick αναφέρεται σε έναν σταθερό κύκλο εκτέλεσης του εξυπηρετητή, κατά τον οποίο ενημερώνεται η κατάσταση του παιχνιδιού. Σε κάθε tick, ο εξυπηρετητής επεξεργάζεται τις ενέργειες των παικτών, ελέγχει συγκρούσεις, υπολογίζει ζημιές και ενημερώνει την κατάσταση όλων των οντοτήτων του παιχνιδιού.

Μετά την ολοκλήρωση κάθε tick, η ενημερωμένη κατάσταση αποστέλλεται στους συνδεδεμένους πελάτες. Η διαδικασία αυτή επαναλαμβάνεται συνεχώς, επιτρέποντας τη συγχρονισμένη και ομαλή εξέλιξη του παιχνιδιού σε πραγματικό χρόνο.

4.3.1.1 Σύνδεση

Οι παίκτες συνδέονται στο παιχνίδι μέσω του μενού, αφού προηγουμένως δημιουργήσουν ένα ψευδώνυμο και επιλέξουν την κλάση του χαρακτήρα τους. Τα στοιχεία αυτά, σε συνδυασμό με έναν μοναδικό αναγνωριστικό αριθμό που δημιουργείται αυτόματα με τη σύνδεση, αποθηκεύονται σε βάση δεδομένων, με σκοπό την καταγραφή και διατήρηση της προόδου του παίκτη στο παιχνίδι.

Κατά τη διαδικασία σύνδεσης, ο εξυπηρετητής επαληθεύει τα δεδομένα του παίκτη μέσω της βάσης δεδομένων και δημιουργεί την αντίστοιχη οντότητα εντός του παιχνιδιού, ενώ σε περίπτωση μη έγκυρων στοιχείων η σύνδεση απορρίπτεται.

Στο παιχνίδι δεν υπάρχει περιορισμός στον αριθμό των παικτών που μπορούν να συμμετέχουν. Κάθε παίκτης που πραγματοποιεί σύνδεση γίνεται αποδεκτός από τον εξυπηρετητή (server) και τοποθετείται στην πρώτη περιοχή του χάρτη. Σε περίπτωση αποσύνδεσης ενός παίκτη, ο χαρακτήρας του παραμένει ενεργός για μικρό χρονικό διάστημα, ώστε να είναι δυνατή η επανασύνδεσή του. Αν ο παίκτης επιστρέψει μετά από μεγαλύτερο χρονικό διάστημα, ο χαρακτήρας του εμφανίζεται εκ νέου στην αφετηρία, εφόσον το παιχνίδι παραμένει ενεργό και δεν έχει ολοκληρωθεί, χωρίς να χάσει τα αντικείμενα και τον χρυσό που είχε πριν αποσυνδεθεί.

Ο παίκτης μπορεί να συνδεθεί ξανά, ανοίγοντας πάλι το παιχνίδι και επιλέγοντας επανασύνδεση, καθώς ο εξυπηρετητής θα του δίνει αυτή την επιλογή.

4.3.1.2 Αρχικοποίηση καταστάσεων

Οι παίκτες ξεκινούν με στόχο τη μετάβαση στην επόμενη περιοχή του χάρτη και έχουν στη διάθεσή τους μόνο τη βασική ικανότητα του χαρακτήρα τους. Κατά την έναρξη του παιχνιδιού, οι παίκτες δεν διαθέτουν επιπλέον αντικείμενα ή πόρους, καθώς ξεκινούν χωρίς χρυσό και χωρίς δωρεάν εξοπλισμό. Η κατάσταση ζωής του παίκτη απεικονίζεται μέσω μπάρας ζωής, η οποία κατά την έναρξη βρίσκεται στο 100%. Σε περίπτωση που ο παίκτης δεχθεί ζημιά από εχθρό, η τιμή της ζωής μειώνεται ανάλογα με τη ζημιά που δέχεται. Ένας παίκτης θεωρείται νεκρός όταν η ζωή του φτάσει στο 0%.

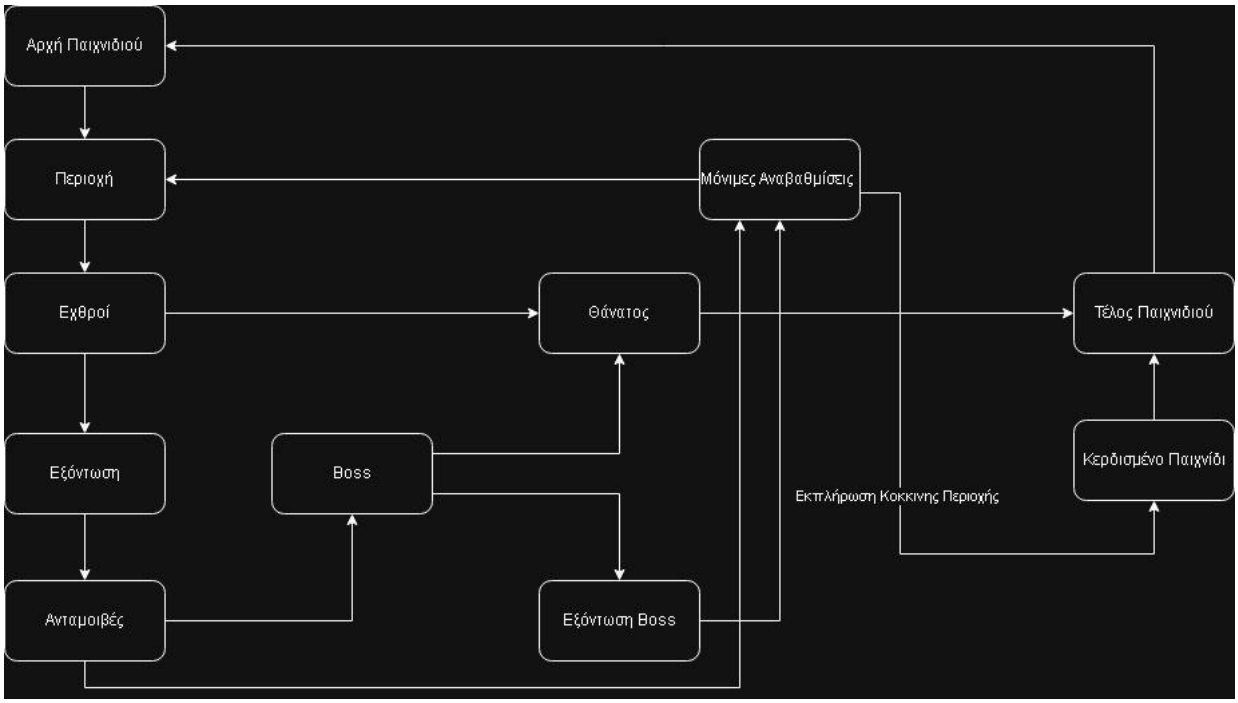
Οι εχθροί τοποθετούνται σε προκαθορισμένα σημεία της περιοχής, τα οποία έχουν επίσης οριστεί στο object layer του χάρτη. Όταν οι παίκτες προσεγγίσουν τους εχθρούς σε συγκεκριμένη απόσταση, ώστε να εισέλθουν στο οπτικό τους πεδίο, ενεργοποιείται η επιθετική συμπεριφορά των εχθρών και ξεκινά η μάχη. Οι εχθροί διαθέτουν επίσης κατάσταση ζωής, η οποία αρχικά βρίσκεται στο 100% και μειώνεται όταν δεχθούν ζημιά από τους παίκτες. Ένας εχθρός πεθαίνει όταν η ζωή του φτάσει στο 0%.

Το παιχνίδι περιλαμβάνει χρονόμετρο, το οποίο επιτρέπει στους παίκτες να παρακολουθούν τη συνολική διάρκεια ενασχόλησής τους, καθώς και τον χρόνο που απαιτείται για τη μετάβαση από τη μία περιοχή στην άλλη. Το χρονόμετρο ενεργοποιείται με τη σύνδεση του πρώτου παίκτη στο παιχνίδι.

Η κατάσταση του παιχνιδιού μεταβάλλεται όταν όλοι οι ενεργοί παίκτες πεθάνουν ή αποσυνδεθούν, ή όταν φτάσουν στην τελευταία περιοχή και εξουδετερώσουν τον τελικό εχθρό. Σε όλες αυτές τις περιπτώσεις το παιχνίδι ολοκληρώνεται και ο εξυπηρετητής τερματίζει τη λειτουργία του.

4.3.1.3 Διάγραμμα καταστάσεων εξυπηρετητή

Το διάγραμμα καταστάσεων του εξυπηρετητή απεικονίζει τη ροή και τις βασικές καταστάσεις του παιχνιδιού όπως αυτές διαχειρίζονται στον εξυπηρετητή, από την έναρξη του παιχνιδιού έως τον τερματισμό του. Παρουσιάζει τις μεταβάσεις μεταξύ περιοχών, μαχών, ανταμοιβών και συνθηκών νίκης ή ήττας, αποτυπώνοντας τη συνολική λογική λειτουργίας του παιχνιδιού σε επίπεδο εξυπηρετητή.



4.3.2 Πελάτης (Client)

Ο πελάτης είναι υπεύθυνος για τη γραφική απεικόνιση του παιχνιδιού και την αλληλεπίδραση του παίκτη με το περιβάλλον μέσω του γραφικού περιβάλλοντος χρήστη (User Interface - UI). Συγκεκριμένα, ο πελάτης λαμβάνει τις ενημερώσεις κατάστασης από τον εξυπηρετητή και προβάλλει τα αντίστοιχα γραφικά, κινούμενα γραφικά (animations) και στοιχεία διεπαφής στον παίκτη.

4.3.2.1 Γραφικό περιβάλλον χρήστη και μηχανισμοί gameplay

Για κάθε παίκτη εμφανίζεται στο UI μπάρα ζωής και μπάρα ενέργειας, οι οποίες κατά την αρχικοποίηση βρίσκονται στο 100%. Η μπάρα ζωής μειώνεται όταν ο παίκτης δέχεται ζημιά από εχθρούς, ενώ η μπάρα ενέργειας μειώνεται με τη χρήση ικανοτήτων. Η ενέργεια απαιτείται για την ενεργοποίηση τόσο των απλών όσο και της υπέρτατης ικανότητας του χαρακτήρα.

Η διαχείριση της ενέργειας αποτελεί βασικό στοιχείο του gameplay, καθώς ο παίκτης οφείλει να περιορίζει τη χρήση των απλών ικανοτήτων, ώστε να διαθέτει επαρκή ενέργεια για την ενεργοποίηση της υπέρτατης ικανότητας όταν αυτή είναι διαθέσιμη. Η ενέργεια αναπληρώνεται σταδιακά με μηχανισμό αναπλήρωσης (regeneration), έως το μέγιστο όριο του 100%, αντίστοιχα με τον τρόπο που αναπληρώνεται και η ζωή του χαρακτήρα.

Επιπλέον, κάθε ικανότητα διαθέτει χρόνο επαναφόρτισης (cooldown), ο οποίος απεικονίζεται στο UI. Οι παίκτες καλούνται να διαχειρίζονται και τους χρόνους επαναφόρτισης, ώστε οι ικανότητες να είναι διαθέσιμες σε περιπτώσεις αυξημένης έντασης, όπως κατά την αντιμετώπιση πολλαπλών εχθρών.

Το γραφικό περιβάλλον περιλαμβάνει επίσης σύστημα αποθέματος (inventory), το οποίο ο παίκτης μπορεί να ανοίξει κατά τη διάρκεια του παιχνιδιού. Μέσω του inventory, ο παίκτης έχει τη δυνατότητα να προσθέτει αντικείμενα δυναμικά, αγοράζοντάς τα με το χρυσό που έχει αποκτήσει κατά την πρόοδό του στο παιχνίδι.

Παράλληλα, υλοποιήθηκε σύστημα κάμερας, το οποίο ακολουθεί τον παίκτη και τον διατηρεί στο κέντρο της οθόνης. Η κάμερα ενημερώνει τη θέση της με βάση τη θέση του χαρακτήρα και τις διαστάσεις του παραθύρου του παιχνιδιού, εξασφαλίζοντας ομαλή παρακολούθηση της κίνησης του παίκτη προς όλες τις κατευθύνσεις.

4.3.2.2 Ομαλοποίηση κίνησης και συγχρονισμός οντοτήτων

Πέρα από την απεικόνιση των γραφικών και του γραφικού περιβάλλοντος χρήστη, ο πελάτης είναι υπεύθυνος για την ομαλή και ρεαλιστική απόδοση της κίνησης των οντοτήτων στο παιχνίδι. Για τον σκοπό αυτό, υλοποιούνται μηχανισμοί ταξινόμησης, συγχρονισμού και εξομάλυνσης της κίνησης των χαρακτήρων.

4.3.2.2.1 Απόδοση βάθους και σωστή επικάλυψη οντοτήτων

Συγκεκριμένα, ο πελάτης εφαρμόζει δυναμική ταξινόμηση (sorting) των χαρακτήρων με βάση τη θέση τους στον κατακόρυφο άξονα, ώστε οι χαρακτήρες να εμφανίζονται σωστά μπροστά ή πίσω από στοιχεία του περιβάλλοντος, όπως τοίχους ή εμπόδια. Η τεχνική αυτή επιτρέπει την οπτικά ορθή απεικόνιση της κίνησης του παίκτη πίσω από αντικείμενα με collision, ενισχύοντας την αίσθηση βάθους του περιβάλλοντος.

4.3.2.2.2 Επεξεργασία κατάστασης από τον εξυπηρετητή

Ο πελάτης λαμβάνει σε τακτά χρονικά διαστήματα την ενημερωμένη κατάσταση του παιχνιδιού από τον εξυπηρετητή και διατηρεί τοπικά τις πιο πρόσφατες πληροφορίες θέσης και χρόνου για κάθε οντότητα. Με τον τρόπο αυτό, ενημερώνονται οι χαρακτήρες παικτών και εχθρών, καθώς και στοιχεία του UI, όπως το χρονόμετρο του παιχνιδιού.

4.3.2.2.3 Ομαλοποίηση κίνησης (interpolation & extrapolation)

Για την αποφυγή απότομων μεταβολών στη θέση των χαρακτήρων λόγω καθυστερήσεων δικτύου, ο πελάτης εφαρμόζει τεχνικές παρεμβολής (interpolation) και πρόβλεψης (extrapolation). Η κίνηση των χαρακτήρων υπολογίζεται βάσει των δύο πιο πρόσφατων καταστάσεων που αποστέλλονται από τον εξυπηρετητή, εξασφαλίζοντας ομαλή μετάβαση μεταξύ διαδοχικών θέσεων και συνεπή απεικόνιση σε πραγματικό χρόνο.

4.3.2.2.4 Συγχρονισμός κίνησης με animations

Παράλληλα, η κατεύθυνση και η κατάσταση κίνησης κάθε χαρακτήρα υπολογίζονται τοπικά από τον πελάτη, με βάση τη μεταβολή της θέσης του. Ανάλογα με το αν ο χαρακτήρας κινείται ή παραμένει ακίνητος, ενεργοποιούνται τα αντίστοιχα animations, διασφαλίζοντας ότι η οπτική αναπαράσταση της κίνησης συμβαδίζει με τη λογική κατάσταση του παιχνιδιού.

4.4 Σύνοψη

Στο παρόν κεφάλαιο παρουσιάστηκε αναλυτικά η υλοποίηση του ψηφιακού παιχνιδιού που αναπτύχθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας. Αρχικά, περιγράφηκε η υλοποίηση των γραφικών του παιχνιδιού, με έμφαση στη δημιουργία του χάρτη μέσω του εργαλείου Tiled, τη δομή των επιπέδων (layers), καθώς και την ενσωμάτωση των γραφικών στοιχείων του περιβάλλοντος και των χαρακτήρων. Παρουσιάστηκε επίσης ο τρόπος υλοποίησης του χαρακτήρα του παίκτη και των αντίστοιχων animations, τα οποία συμβάλλουν στην ομαλή και ρεαλιστική οπτική απεικόνιση της κίνησης και των ενεργειών του.

Επιπλέον, παρουσιάστηκε αναλυτικά η υλοποίηση των χαρακτήρων των παικτών και των εχθρών, με έμφαση στις καταστάσεις κίνησης και στα αντίστοιχα animations, όπως οι καταστάσεις αδράνειας, περπατήματος, τραυματισμού και θανάτου. Παράλληλα, αναλύθηκαν οι μηχανισμοί σύγκρουσης των οντοτήτων με τα εμπόδια του χάρτη, καθώς και οι συγκρούσεις μεταξύ παικτών και εχθρών, ώστε να διασφαλίζεται η ορθή αλληλεπίδρασή τους μέσα στο περιβάλλον του παιχνιδιού. Τέλος, παρουσιάστηκαν οι βασικοί μηχανισμοί μάχης, που περιλαμβάνουν τις επιθέσεις των παικτών και των εχθρών, τον υπολογισμό της ζημιάς, την ενεργοποίηση των καταστάσεων τραυματισμού και θανάτου, καθώς και τη συνολική συμπεριφορά των εχθρικών οντοτήτων κατά την καταδίωξη και την επίθεση.

Στη συνέχεια, αναλύθηκαν οι βασικοί μηχανισμοί gameplay, με αναφορά στη συνολική ροή του παιχνιδιού, τη συνεργασία μεταξύ των παικτών, τη διαχείριση της εξέλιξης και τις συνθήκες ολοκλήρωσής του. Παράλληλα, παρουσιάστηκε η δικτυακή αρχιτεκτονική του παιχνιδιού, αναδεικνύοντας τον ρόλο του εξυπηρετητή στη διαχείριση της λογικής και στον συγχρονισμό των παικτών, καθώς και τον ρόλο του πελάτη στη γραφική απεικόνιση, στη διαχείριση του γραφικού περιβάλλοντος χρήστη και στην ομαλοποίηση της κίνησης μέσω τεχνικών συγχρονισμού.

Με τον τρόπο αυτό, το κεφάλαιο ανέδειξε πώς ο συνδυασμός γραφικών, μηχανισμών gameplay και αρχιτεκτονικής client-server οδήγησε στην υλοποίηση ενός λειτουργικού και συνεργατικού ψηφιακού παιχνιδιού σε πραγματικό χρόνο.

Κεφάλαιο 5: Αξιολόγηση

Στο κεφάλαιο αυτό παρουσιάζεται η αξιολόγηση της απόδοσης του παιχνιδιού που αναπτύχθηκε. Σκοπός της αξιολόγησης είναι η διερεύνηση της συμπεριφοράς της εφαρμογής σε διαφορετικές συνθήκες εκτέλεσης, με έμφαση στη σταθερότητα του ρυθμού ανανέωσης στον client και στον χρόνο επεξεργασίας της λογικής του παιχνιδιού στον server. Δεδομένου ότι η εφαρμογή ακολουθεί αρχιτεκτονική client-server, η απόδοση δεν εξαρτάται μόνο από την απεικόνιση των γραφικών, αλλά και από την ικανότητα του server να επεξεργάζεται έγκαιρα την κίνηση των παικτών, τη συμπεριφορά των εχθρών, τις συγκρούσεις και τις επιθέσεις. Για τον λόγο αυτό χρησιμοποιήθηκαν μετρικές όπως τα καρέ ανά δευτερόλεπτο (Frames Per Second – FPS), ο μέσος και μέγιστος χρόνος frame στον client, καθώς και ο μέσος και μέγιστος χρόνος tick στον server. Τα FPS δείχνουν πόσα καρέ προβάλλονται από τον client μέσα σε ένα δευτερόλεπτο και αποτελούν βασική ένδειξη της ομαλότητας της απεικόνισης. Ο χρόνος frame, μετρημένος σε milliseconds, εκφράζει τον χρόνο που χρειάζεται ο client για να επεξεργαστεί και να εμφανίσει ένα καρέ του παιχνιδιού. Αντίστοιχα, ο χρόνος tick του server δείχνει πόσο χρόνο χρειάζεται ο server για να εκτελέσει έναν κύκλο ενημέρωσης της λογικής του παιχνιδιού.

Για την ερμηνεία των μετρήσεων του server χρησιμοποιήθηκε ως βασικό όριο ο χρόνος των 20 ms ανά tick. Το όριο αυτό προκύπτει από τη ρύθμιση του συστήματος, καθώς ο server λειτουργεί με TICK_DT = 0.02 δευτερόλεπτα, δηλαδή με στόχο τις 50 ενημερώσεις ανά δευτερόλεπτο. Σε παιχνίδια πραγματικού χρόνου, όπως το MMORPG που αναπτύχθηκε, είναι σημαντικό ο server να μπορεί να επεξεργάζεται την κίνηση των παικτών, τη συμπεριφορά των εχθρών, τις συγκρούσεις και τις επιθέσεις μέσα στο διαθέσιμο χρονικό διάστημα κάθε tick. Όταν ο μέσος χρόνος επεξεργασίας παραμένει κάτω από τα 20 ms, ο server μπορεί να διατηρήσει τον επιθυμητό ρυθμό ενημέρωσης. Αντίθετα, όταν ο μέσος χρόνος tick ξεπερνά συστηματικά το όριο αυτό, υπάρχει κίνδυνος καθυστέρησης στην ενημέρωση της κατάστασης του παιχνιδιού και υποβάθμισης της ομαλότητας της εμπειρίας του χρήστη.

5.1 Αρχική αξιολόγηση απόδοσης

Στην αρχική αξιολόγηση πραγματοποιήθηκε μέτρηση με έναν συνδεδεμένο παίκτη, ώστε να εξεταστεί πρώτα η βασική συμπεριφορά της εφαρμογής πριν την αύξηση του αριθμού των ταυτόχρονων χρηστών. Οι μετρήσεις καταγράφονταν ανά χρονικό διάστημα 20 δευτερολέπτων, ώστε κάθε γραμμή των πινάκων να παρουσιάζει συγκεντρωτικά αποτελέσματα για το αντίστοιχο χρονικό διάστημα. Από την πλευρά του client παρατηρήθηκε ότι ο μέσος ρυθμός ανανέωσης παρέμεινε κοντά στα 60 FPS στις περισσότερες περιοχές, γεγονός που δείχνει ότι η απεικόνιση των γραφικών ήταν γενικά ομαλή. Ωστόσο, καταγράφηκαν ορισμένες υψηλές τιμές στον μέγιστο χρόνο frame, κυρίως κατά τη μετάβαση από μία περιοχή σε άλλη. Οι τιμές αυτές δείχνουν στιγμιαία κολλήματα στην απεικόνιση, τα οποία πιθανότατα σχετίζονται με τη φόρτωση νέου χάρτη, αντικειμένων και sprites. Σε ορισμένους πίνακες, όταν στην τελευταία γραμμή εμφανίζεται αριθμός παικτών ίσος με 0, αυτό αντιστοιχεί στη φάση αποσύνδεσης του client από τον server κατά την ολοκλήρωση της μέτρησης.

Table 14: Αρχικές μετρήσεις απόδοσης client

Χρόνος (s)	Μέσο FPS	Ελάχιστο FPS	Μέσος χρόνος frame (ms)	Μέγιστος χρόνος frame (ms)	Παίκτες	Εχθροί	Περιοχή
20.008	59.57	1.09	17.76	919.27	1	21	firstRegion
40.011	59.70	1.81	17.20	551.35	1	34	secondRegion
60.012	59.71	53.63	16.75	18.65	1	34	secondRegion
80.017	59.74	55.13	16.74	18.14	1	34	secondRegion
100.034	59.66	0.82	17.82	1223.74	1	47	thirdRegion
120.035	59.71	52.62	16.75	19.01	1	47	thirdRegion
141.021	59.72	0.79	17.80	1272.02	1	79	fourthRegion
161.031	59.36	48.37	16.86	20.67	1	79	fourthRegion
181.041	59.27	48.71	16.89	20.53	1	79	fourthRegion
201.041	59.25	48.28	16.89	20.71	1	79	fourthRegion
221.042	59.34	50.70	16.86	19.72	1	79	fourthRegion
241.056	59.31	45.82	16.88	21.82	1	79	fourthRegion

Αντίθετα, μεγαλύτερη επιβάρυνση παρατηρήθηκε στην πλευρά του server. Στις αρχικές μετρήσεις, όταν αυξανόταν ο αριθμός των εχθρών και ιδιαίτερα όταν πολλοί από αυτούς ενεργοποιούσαν ταυτόχρονα τη συμπεριφορά καταδίωξης, ο μέσος χρόνος επεξεργασίας tick αυξανόταν σημαντικά. Παρόλο που στις πρώτες μετρήσεις ο μέσος χρόνος tick παρέμενε κάτω από το όριο των 20 ms, στη συνέχεια έφτασε τα 25.672 ms, ενώ ο μέγιστος χρόνος tick έφτασε τα 45.251 ms. Δεδομένου ότι το σύστημα είχε οριστεί να λειτουργεί με TICK_DT = 0.02 δευτερόλεπτα, δηλαδή με επιθυμητό διάστημα ενημέρωσης 20 ms και στόχο τα 50 ticks ανά δευτερόλεπτο, οι τιμές αυτές δείχνουν ότι ο server δεν προλάβαινε πάντα να ολοκληρώσει την επεξεργασία της λογικής του παιχνιδιού μέσα στον προβλεπόμενο χρόνο. Αυτό μπορούσε να επηρεάσει την ομαλότητα της κίνησης και τη συνολική αίσθηση του παιχνιδιού, γεγονός που κατέστησε απαραίτητη την εφαρμογή βελτιστοποιήσεων στην επεξεργασία που πραγματοποιείται στην πλευρά του server.

Table 15: Αρχικές μετρήσεις απόδοσης server

Χρόνος (s)	Μέσος χρόνος tick (ms)	Μέγιστος χρόνος tick (ms)	Παίκτες	Εχθροί
31.054	5.413	11.740	1	181
62.523	8.024	12.577	1	181
95.938	9.751	15.312	1	181
127.206	8.336	14.991	1	181
165.510	12.048	22.302	1	181
218.403	25.672	45.251	1	181

Η βασική αιτία της επιβάρυνσης εντοπίστηκε στον τρόπο ενημέρωσης των εχθρών. Αρχικά, ο server επεξεργαζόταν όλους τους εχθρούς σε κάθε tick, ανεξάρτητα από την περιοχή στην οποία βρίσκονταν ή από το αν υπήρχε παίκτης κοντά τους. Αυτό είχε ως αποτέλεσμα να εκτελούνται συνεχώς έλεγχοι απόστασης, συγκρούσεις, επιθέσεις και ενημερώσεις κατάστασης για μεγάλο αριθμό εχθρών. Για τον λόγο αυτό

κρίθηκε απαραίτητη η εφαρμογή βελτιστοποιήσεων, ώστε ο server να επεξεργάζεται μόνο τους εχθρούς που βρίσκονται σε ενεργές περιοχές, δηλαδή σε περιοχές όπου υπάρχει τουλάχιστον ένας παίκτης, και να μειωθεί η συχνότητα ενημέρωσης της λογικής των εχθρών χωρίς να επηρεαστεί σημαντικά η εμπειρία του χρήστη.

5.2 Βελτιστοποιήσεις απόδοσης

Με βάση τα αποτελέσματα της αρχικής αξιολόγησης, κρίθηκε απαραίτητη η εφαρμογή βελτιστοποιήσεων κυρίως στην πλευρά του server, καθώς εκεί εντοπίστηκε η μεγαλύτερη επιβάρυνση. Η πρώτη βελτιστοποίηση που εφαρμόστηκε ήταν ο περιορισμός της επεξεργασίας των εχθρών μόνο στις ενεργές περιοχές. Ως ενεργή περιοχή θεωρείται μια περιοχή στην οποία βρίσκεται τουλάχιστον ένας ζωντανός παίκτης. Με αυτόν τον τρόπο, οι εχθροί που βρίσκονται σε περιοχές χωρίς παίκτες δεν ενημερώνονται προσωρινά από τον server, μειώνοντας τον αριθμό των υπολογισμών που εκτελούνται σε κάθε tick. Η αλλαγή αυτή ήταν σημαντική, καθώς το παιχνίδι περιλαμβάνει πολλούς εχθρούς συνολικά, αλλά σε κάθε χρονική στιγμή ο παίκτης αλληλεπιδρά μόνο με τους εχθρούς της τρέχουσας περιοχής του.

5.2.1 Πρώτη βελτιστοποίηση: Επεξεργασία εχθρών μόνο σε ενεργές περιοχές

Table 16: Μετρήσεις client μετά από την πρώτη βελτιστοποίηση

Χρόνος (s)	Μέσο FPS	Ελάχιστο FPS	Μέσος χρόνος frame (ms)	Μέγιστος χρόνος frame (ms)	Παίκτες	Εχθροί	Περιοχή
20.008	59.50	1.14	17.73	881.02	1	21	firstRegion
40.014	59.65	1.81	17.22	553.76	1	34	secondRegion
60.016	59.70	54.36	16.75	18.40	1	34	secondRegion
80.017	59.71	0.82	17.81	1224.05	1	47	thirdRegion
100.019	59.70	53.76	16.75	18.60	1	47	thirdRegion
120.020	59.71	52.85	16.75	18.92	1	47	thirdRegion
140.026	59.48	0.78	17.94	1287.01	1	79	fourthRegion
160.034	59.23	47.62	16.90	21.00	1	79	fourthRegion
180.035	59.16	47.81	16.92	20.92	1	79	fourthRegion
200.041	59.37	50.99	16.85	19.61	1	79	fourthRegion
220.053	59.18	48.94	16.92	20.43	1	79	fourthRegion
240.056	59.29	50.80	16.88	19.69	1	79	fourthRegion
260.067	59.04	49.21	16.96	20.32	1	79	fourthRegion
280.082	59.08	45.44	16.95	22.01	1	79	fourthRegion
300.096	59.30	15.98	16.90	62.57	1	79	fourthRegion
320.109	59.14	42.87	16.93	23.33	1	79	fourthRegion

Από τις μετρήσεις του client μετά την πρώτη βελτιστοποίηση παρατηρείται ότι η απόδοση παρέμεινε στα ίδια περίπου επίπεδα με την αρχική αξιολόγηση. Ο μέσος ρυθμός ανανέωσης εξακολούθησε να βρίσκεται κοντά στα 59 FPS, ενώ ο μέσος χρόνος frame παρέμεινε περίπου στα 16.75–17.94 ms. Οι υψηλές τιμές στον μέγιστο χρόνο

frame εμφανίζονται και πάλι κυρίως κατά τις μεταβάσεις μεταξύ περιοχών, όπως είχε παρατηρηθεί και στην αρχική αξιολόγηση. Επομένως, η πρώτη βελτιστοποίηση δεν επηρέασε αρνητικά την ομαλότητα της απεικόνισης στον client, ενώ η κύρια επίδρασή της αναμενόταν στην πλευρά του server.

Table 17: Μετρήσεις server μετά από την πρώτη βελτιστοποίηση

Χρόνος (s)	Μέσος χρόνος tick (ms)	Μέγιστος χρόνος tick (ms)	Παίκτες	Εχθροί
31.057	1.340	2.104	0	181
62.056	2.606	7.328	1	181
93.384	5.731	14.422	1	181
127.251	8.340	14.839	1	181
158.559	7.805	13.457	1	181
192.403	9.013	19.568	1	181
237.486	16.033	28.740	1	181
279.577	13.342	26.470	1	181
324.702	15.339	27.197	1	181
367.608	14.342	36.008	0	181

Από τις μετρήσεις του server μετά την πρώτη βελτιστοποίηση φαίνεται ότι υπήρξε βελτίωση σε σχέση με την αρχική κατάσταση. Ο μέσος χρόνος επεξεργασίας tick στις περισσότερες μετρήσεις παρέμεινε κάτω από το όριο των 20 ms, ενώ στην αρχική αξιολόγηση είχε φτάσει τα 25.672 ms. Ωστόσο, σε ορισμένα σημεία ο μέσος χρόνος tick αυξήθηκε, φτάνοντας τα 16.033 ms, ενώ ο μέγιστος χρόνος tick έφτασε τα 36.008 ms. Αυτό δείχνει ότι ο περιορισμός της επεξεργασίας των εχθρών στις ενεργές περιοχές μείωσε τον φόρτο του server, αλλά δεν εξάλειψε πλήρως τα στιγμιαία φορτία που εμφανίζονται όταν υπάρχουν πολλοί ενεργοί εχθροί ή όταν εκτελούνται ταυτόχρονα αρκετοί έλεγχοι τεχνητής νοημοσύνης, συγκρούσεων και επιθέσεων.

5.2.2 Δεύτερη βελτιστοποίηση: Μείωση συχνότητας ενημέρωσης εχθρών

Επιπλέον, μειώθηκε η συχνότητα ενημέρωσης των εχθρών. Αρχικά, η λογική των εχθρών εκτελούνταν σε κάθε server tick, δηλαδή κάθε 0.02 δευτερόλεπτα. Μετά τη βελτιστοποίηση, η ενημέρωση πραγματοποιείται ανά μεγαλύτερο χρονικό διάστημα, συγκεκριμένα κάθε 0.04 δευτερόλεπτα, ώστε να μειωθεί ο φόρτος του server. Για να μην μειωθεί η πραγματική ταχύτητα κίνησης των εχθρών, η μετακίνηση προσαρμόστηκε με βάση το χρονικό διάστημα που έχει περάσει από την προηγούμενη ενημέρωση. Έτσι, οι εχθροί διατηρούν την ίδια συνολική ταχύτητα, ενώ ο server εκτελεί λιγότερους υπολογισμούς ανά δευτερόλεπτο.

Table 18: Μετρήσεις client μετά από τη δεύτερη βελτιστοποίηση

Χρόνος (s)	Μέσο FPS	Ελάχιστο FPS	Μέσος χρόνος frame (ms)	Μέγιστος χρόνος frame (ms)	Παίκτες	Εχθροί	Περιοχή
20.002	59.55	1.03	17.85	968.22	1	21	firstRegion
40.014	59.67	1.82	17.21	550.63	1	34	secondRegion

Χρόνος (s)	Μέσο FPS	Ελάχιστο FPS	Μέσος χρόνος frame (ms)	Μέγιστος χρόνος frame (ms)	Παίκτες	Εχθροί	Περιοχή
60.025	59.68	54.35	16.76	18.40	1	34	secondRegion
80.026	59.65	0.82	17.83	1219.91	1	47	thirdRegion
100.026	59.66	52.81	16.76	18.94	1	47	thirdRegion
120.038	59.72	54.69	16.75	18.29	1	47	thirdRegion
140.039	59.56	0.78	17.92	1288.96	1	79	fourthRegion
160.054	59.43	17.18	16.86	58.21	1	79	fourthRegion
180.070	58.99	37.09	16.99	26.96	1	79	fourthRegion

Από τις μετρήσεις του client μετά τη μείωση της συχνότητας ενημέρωσης των εχθρών παρατηρείται ότι η απόδοση παρέμεινε σταθερή και στα ίδια περίπου επίπεδα με τις προηγούμενες μετρήσεις. Ο μέσος ρυθμός ανανέωσης διατηρήθηκε κοντά στα 59 FPS, ενώ ο μέσος χρόνος frame παρέμεινε περίπου στα 16.75–17.92 ms. Οι υψηλές τιμές στον μέγιστο χρόνο frame εμφανίζονται και πάλι κυρίως κατά τις μεταβάσεις μεταξύ περιοχών, γεγονός που δείχνει ότι σχετίζονται περισσότερο με τη φόρτωση νέου περιεχομένου και όχι με τη συχνότητα ενημέρωσης των εχθρών. Επομένως, η δεύτερη βελτιστοποίηση δεν επηρέασε αρνητικά την απόδοση του client.

Table 19: Μετρήσεις server μετά από τη δεύτερη βελτιστοποίηση

Χρόνος (s)	Μέσος χρόνος tick (ms)	Μέγιστος χρόνος tick (ms)	Παίκτες	Εχθροί
30.743	3.037	8.678	1	181
61.763	3.626	11.304	1	181
92.947	4.749	12.385	1	181
124.129	5.515	14.943	1	181
157.185	6.895	18.336	1	181
195.233	10.384	37.937	0	181

Από τις μετρήσεις του server μετά τη δεύτερη βελτιστοποίηση παρατηρείται περαιτέρω μείωση του μέσου χρόνου επεξεργασίας tick. Σε όλες τις καταγραφές ο μέσος χρόνος tick παρέμεινε κάτω από το όριο των 20 ms, φτάνοντας έως τα 10.384 ms, ενώ πριν από τις βελτιστοποιήσεις είχε φτάσει τα 25.672 ms. Αυτό δείχνει ότι η μείωση της συχνότητας ενημέρωσης των εχθρών περιόρισε σημαντικά τον φόρτο του server, καθώς η τεχνητή νοημοσύνη και η κίνηση των εχθρών δεν εκτελούνται πλέον σε κάθε tick. Παρόλα αυτά, εμφανίζεται στιγμιαία αύξηση στον μέγιστο χρόνο tick, με τιμή 37.937 ms, η οποία δείχνει ότι εξακολουθούν να υπάρχουν στιγμιαία φορτία, χωρίς όμως να επηρεάζεται σημαντικά ο μέσος χρόνος επεξεργασίας.

5.2.3 Εξομάλυνση κίνησης εχθρών στον client

Μετά τη μείωση της συχνότητας ενημέρωσης των εχθρών στον server, παρατηρήθηκε ότι η κίνησή τους μπορούσε να φαίνεται πιο διακεκομμένη στην πλευρά του client. Αυτό συνέβαινε επειδή οι θέσεις των εχθρών δεν ενημερώνονταν πλέον σε

κάθε server tick, αλλά ανά μεγαλύτερο χρονικό διάστημα. Για τον λόγο αυτό εφαρμόστηκε εξομάλυνση της κίνησης των εχθρών στον client. Συγκεκριμένα, ο client αποθηκεύει τις πιο πρόσφατες θέσεις που λαμβάνει από τον server και παρεμβάλλει ομαλά την κίνηση μεταξύ τους. Με αυτόν τον τρόπο μειώνεται η αίσθηση διακεκομμένης κίνησης, ενώ διατηρείται η μείωση του φόρτου στην πλευρά του server.

Στη συνέχεια πραγματοποιήθηκαν νέες μετρήσεις απόδοσης με διαφορετικό αριθμό ταυτόχρονων παικτών, συγκεκριμένα με 1, 3 και 5 παίκτες. Σκοπός των μετρήσεων αυτών ήταν να εξεταστεί αν οι βελτιστοποιήσεις που εφαρμόστηκαν διατηρούν την απόδοση του συστήματος σε αποδεκτά επίπεδα, όχι μόνο σε βασικό σενάριο με έναν παίκτη, αλλά και σε πιο απαιτητικά σενάρια με περισσότερους συνδεδεμένους clients και αυξημένο αριθμό ενεργών εχθρών.

Στις τελικές μετρήσεις ο client δεν παρουσίασε σημαντικές διαφοροποιήσεις σε σχέση με τις προηγούμενες μετρήσεις, καθώς ο μέσος ρυθμός ανανέωσης παρέμεινε κοντά στα 59 FPS και ο μέσος χρόνος frame διατηρήθηκε περίπου στα ίδια επίπεδα. Για τον λόγο αυτό, η τελική σύγκριση εστιάζει κυρίως στην πλευρά του server, όπου επηρεάζεται περισσότερο η απόδοση από τον αριθμό των ταυτόχρονων παικτών και των ενεργών εχθρών. Στους τελικούς πίνακες του server χρησιμοποιήθηκε η στήλη «Ενεργοί εχθροί» αντί για τον συνολικό αριθμό εχθρών, ώστε να αποτυπώνεται πιο σωστά ο πραγματικός φόρτος του server μετά τις βελτιστοποιήσεις.

5.2.3.1 Μέτρηση με έναν παίκτη

Table 20: Μετρήσεις server με έναν παίκτη

Χρόνος (s)	Μέσος χρόνος tick (ms)	Μέγιστος χρόνος tick (ms)	Παίκτες	Ενεργοί εχθροί
30.760	3.034	9.599	1	20
61.602	3.515	9.530	1	33
92.328	3.741	13.012	1	32
123.138	3.408	9.828	1	29
153.611	3.466	7.319	1	27
184.335	3.308	6.956	1	25
215.188	3.304	5.122	1	45
245.904	3.594	8.005	1	44
276.819	4.328	9.404	1	43
307.803	4.629	8.284	1	43
338.858	4.155	9.784	1	42
369.710	3.786	6.756	1	42
400.797	3.967	8.020	1	40
431.525	4.586	12.855	1	79
462.589	5.607	15.137	1	77
494.343	6.883	14.638	1	76
525.446	5.608	14.912	1	75
558.836	7.013	19.475	1	74
596.800	9.892	21.843	1	73
635.071	9.767	26.291	1	72

Χρόνος (s)	Μέσος χρόνος tick (ms)	Μέγιστος χρόνος tick (ms)	Παίκτες	Ενεργοί εχθροί
673.625	10.721	32.788	1	70
711.962	10.269	30.638	1	70
749.483	9.465	29.627	1	69
788.107	10.207	26.746	1	69
826.204	9.662	25.150	1	68
863.423	9.197	25.963	1	0

Στη μέτρηση με έναν παίκτη, ο μέσος χρόνος επεξεργασίας tick παρέμεινε σε όλες τις περιπτώσεις κάτω από το όριο των 20 ms. Ακόμη και όταν ο αριθμός των ενεργών εχθρών αυξήθηκε σε περίπου 70–79, ο μέσος χρόνος tick κυμάνθηκε περίπου από 9 έως 10.7 ms. Παρατηρούνται ορισμένες στιγμιαίες αυξήσεις στον μέγιστο χρόνο tick, με τιμές έως 32.788 ms, οι οποίες όμως δεν επηρεάζουν σημαντικά τον μέσο χρόνο επεξεργασίας. Η τελευταία γραμμή, όπου οι ενεργοί εχθροί είναι 0, αντιστοιχεί στη φάση αποσύνδεσης/ολοκλήρωσης της μέτρησης.

5.2.3.2 Μέτρηση με τρεις παίκτες

Table 21: Μετρήσεις server με τρεις παίκτες

Χρόνος (s)	Μέσος χρόνος tick (ms)	Μέγιστος χρόνος tick (ms)	Παίκτες	Ενεργοί εχθροί
31.107	2.565	5.036	3	21
62.214	5.442	12.537	3	55
95.133	7.055	18.382	3	68
129.979	8.048	20.771	3	68
166.442	10.155	25.096	3	100
205.461	12.757	31.170	3	100

Στη μέτρηση με τρεις ταυτόχρονους παίκτες, ο server διαχειρίστηκε έως 100 ενεργούς εχθρούς, με τον μέσο χρόνο tick να φτάνει τα 12.757 ms. Η τιμή αυτή παραμένει κάτω από το όριο των 20 ms, γεγονός που δείχνει ότι ο server διατηρεί ικανοποιητική απόδοση ακόμη και με περισσότερους συνδεδεμένους clients. Ωστόσο, ο μέγιστος χρόνος tick έφτασε τα 31.170 ms, δείχνοντας ότι εξακολουθούν να εμφανίζονται στιγμιαία φορτία όταν αυξάνεται σημαντικά ο αριθμός των ενεργών εχθρών.

5.2.3.3 Μέτρηση με πέντε παίκτες

Table 22: Μετρήσεις server με πέντε παίκτες

Χρόνος (s)	Μέσος χρόνος tick (ms)	Μέγιστος χρόνος tick (ms)	Παίκτες	Ενεργοί εχθροί
31.157	1.493	2.655	0	0
62.403	2.142	4.311	2	21
93.654	4.028	7.531	5	21
125.354	6.624	14.114	5	55
159.039	8.463	21.737	5	55
192.963	8.647	20.622	5	68
230.184	10.546	25.075	5	100
269.357	13.663	33.427	5	100

Στη μέτρηση με πέντε ταυτόχρονους παίκτες παρατηρείται περαιτέρω αύξηση του φόρτου στον server, κυρίως όταν οι ενεργοί εχθροί φτάνουν τους 100. Παρόλα αυτά, ο μέσος χρόνος επεξεργασίας tick παρέμεινε κάτω από το όριο των 20 ms, με μέγιστη μέση τιμή 13.663 ms. Ο μέγιστος χρόνος tick έφτασε τα 33.427 ms, γεγονός που δείχνει στιγμιαίες αυξήσεις του φόρτου, χωρίς όμως ο μέσος χρόνος επεξεργασίας να ξεπερνά το επιθυμητό όριο. Οι πρώτες γραμμές, όπου εμφανίζονται λιγότεροι από πέντε παίκτες, αντιστοιχούν στη φάση σύνδεσης των clients πριν σταθεροποιηθεί το σενάριο με 5 παίκτες.

5.3 Συμπεράσματα μετρήσεων

Με βάση το όριο των 20 ms ανά tick που τέθηκε για την αξιολόγηση του server, τα τελικά αποτελέσματα κρίνονται ικανοποιητικά. Σε όλα τα σενάρια δοκιμών, δηλαδή με 1, 3 και 5 ταυτόχρονους παίκτες, ο μέσος χρόνος επεξεργασίας tick παρέμεινε κάτω από το επιθυμητό όριο. Συγκεκριμένα, στο σενάριο με 1 παίκτη ο μέσος χρόνος έφτασε περίπου τα 10.721 ms, στο σενάριο με 3 παίκτες τα 12.757 ms, ενώ στο σενάριο με 5 παίκτες τα 13.663 ms. Οι τιμές αυτές δείχνουν ότι ο server μπορούσε να ολοκληρώνει την επεξεργασία της λογικής του παιχνιδιού μέσα στο διαθέσιμο χρονικό διάστημα των 20 ms, ακόμη και όταν ο αριθμός των ενεργών εχθρών αυξανόταν έως περίπου τους 100.

Παρόλο που σε ορισμένες περιπτώσεις ο μέγιστος χρόνος tick ξεπέρασε τα 20 ms, οι αυξήσεις αυτές ήταν στιγμιαίες και δεν επηρέασαν τον μέσο χρόνο επεξεργασίας. Αυτό σημαίνει ότι ο server ενδέχεται σε ορισμένα ticks να δέχεται μεγαλύτερο φόρτο, κυρίως όταν υπάρχουν πολλοί ενεργοί εχθροί και εκτελούνται ταυτόχρονα έλεγχοι τεχνητής νοημοσύνης, συγκρούσεων και επιθέσεων. Ωστόσο, επειδή ο μέσος χρόνος παρέμεινε σταθερά κάτω από το όριο, η συνολική λειτουργία του server θεωρείται ομαλή και εντός των απαιτήσεων του παιχνιδιού.

Επομένως, οι βελτιστοποιήσεις που εφαρμόστηκαν κρίνονται αποτελεσματικές. Ο περιορισμός της επεξεργασίας μόνο στις ενεργές περιοχές, η μείωση της συχνότητας ενημέρωσης των εχθρών και η εξομάλυνση της κίνησης στον client βοήθησαν ώστε το σύστημα να διατηρήσει ικανοποιητική απόδοση. Τα αποτελέσματα δείχνουν ότι η

εφαρμογή μπορεί να υποστηρίξει πολλαπλούς ταυτόχρονους παίκτες και μεγάλο αριθμό ενεργών εχθρών, χωρίς ο μέσος χρόνος επεξεργασίας του server να ξεπερνά το επιθυμητό όριο των 20 ms.

5.4 Μετρικά κώδικα

Εκτός από την αξιολόγηση της απόδοσης του συστήματος, πραγματοποιήθηκε ανάλυση της πολυπλοκότητας του κώδικα, με στόχο να εκτιμηθεί η δομή και η συντηρησιμότητα της εφαρμογής. Για τον σκοπό αυτό χρησιμοποιήθηκε η μετρική της κυκλωματικής πολυπλοκότητας, η οποία δείχνει τον αριθμό των διαφορετικών διαδρομών εκτέλεσης που μπορεί να ακολουθήσει μία μέθοδος. Όσο μεγαλύτερη είναι η τιμή της, τόσο πιο σύνθετη θεωρείται η ροή ελέγχου του κώδικα.

Συνολικά αναλύθηκαν 174 blocks κώδικα, δηλαδή κλάσεις και μέθοδοι. Η μέση κυκλωματική πολυπλοκότητα του έργου ήταν βαθμίδα A (4.84), τιμή που αντιστοιχεί σε χαμηλή πολυπλοκότητα. Αυτό δείχνει ότι, συνολικά, ο κώδικας παραμένει σε ικανοποιητικό επίπεδο ως προς τη δομική του πολυπλοκότητα.

Table 23: Μέθοδοι με την υψηλότερη κυκλωματική πολυπλοκότητα

Αρχείο	Μέθοδος	Πολυπλοκότητα	Βαθμίδα
client.py	MyGame.process_server_state	40	E
client.py	MyGame.on_update	37	E
server.py	update_enemy_chase_and_movement	28	D
server.py	apply_player_attacks	28	D
client.py	MyGame.on_key_press	25	D
dragon_enemy.py	update_dragon	25	D
server.py	collides_with_walls_aabb	23	D
server.py	broadcast_state	23	D
server.py	apply_enemy_attacks	18	C
region.py	Region.load_objects	14	C
sprites.py	PlayerSprite.update_animation	13	C
sprites.py	EnemySprite.update_animation	13	C

Οι υψηλότερες τιμές πολυπλοκότητας εμφανίζονται κυρίως στα αρχεία client.py και server.py, όπου υλοποιούνται οι βασικοί μηχανισμοί του παιχνιδιού. Για παράδειγμα, η μέθοδος process_server_state είναι υπεύθυνη για την επεξεργασία της κατάστασης που λαμβάνει ο client από τον server, ενώ η on_update συντονίζει την ενημέρωση της κίνησης, των animations, των projectiles και της κάμερας. Αντίστοιχα, στον server υψηλότερη πολυπλοκότητα εμφανίζεται στις μεθόδους που αφορούν την τεχνητή νοημοσύνη των εχθρών, τις επιθέσεις, τις συγκρούσεις και τη μετάδοση της κατάστασης του παιχνιδιού. Οι τιμές αυτές είναι αναμενόμενες, καθώς οι συγκεκριμένες μέθοδοι περιλαμβάνουν πολλές διαφορετικές περιπτώσεις εκτέλεσης και ελέγχους ροής.

Επίσης, εξετάστηκαν και βασικές μετρικές μεγέθους του κώδικα, όπως οι συνολικές γραμμές, οι γραμμές πηγαίου κώδικα, οι γραμμές σχολίων και οι κενές γραμμές. Οι μετρικές αυτές βοηθούν στην αποτύπωση της έκτασης του έργου και της τεκμηρίωσης που υπάρχει μέσα στον κώδικα. Για την ανάλυση χρησιμοποιήθηκε το εργαλείο radon raw, το οποίο υπολογίζει τις αντίστοιχες τιμές για τα αρχεία Python του project.

Table 24: Μετρικές γραμμών κώδικα

Μετρική	Τιμή
Συνολικές γραμμές (LOC)	5888
Λογικές γραμμές κώδικα (LLOC)	2832
Γραμμές πηγαίου κώδικα (SLOC)	3847
Γραμμές σχολίων	1308
Κενές γραμμές	1231
Ποσοστό σχολίων επί των συνολικών γραμμών	22%
Ποσοστό σχολίων επί των γραμμών κώδικα	34%

Από την ανάλυση προέκυψε ότι το έργο αποτελείται συνολικά από 5888 γραμμές, εκ των οποίων οι 3847 είναι γραμμές πηγαίου κώδικα, οι 1308 είναι γραμμές σχολίων και οι 1231 είναι κενές γραμμές. Το ποσοστό σχολίων επί των συνολικών γραμμών είναι 22%, ενώ σε σχέση με τις γραμμές κώδικα φτάνει το 34%. Οι τιμές αυτές δείχνουν ότι ο κώδικας περιλαμβάνει σημαντικό αριθμό σχολίων, γεγονός που βοηθά στην κατανόηση και στη συντήρησή του.

Κεφάλαιο 6: Συμπεράσματα και μελλοντικές επεκτάσεις

Η παρούσα πτυχιακή εργασία είχε ως αντικείμενο τον σχεδιασμό και την ανάπτυξη ενός διαδικτυακού παιχνιδιού ρόλων πολλαπλών παικτών σε περιβάλλον 2D. Στόχος της εργασίας ήταν η δημιουργία ενός λειτουργικού παιχνιδιού τύπου MMORPG, στο οποίο οι παίκτες μπορούν να συνδέονται σε έναν κοινό κόσμο, να επιλέγουν χαρακτήρα, να κινούνται σε διαφορετικές περιοχές, να αλληλεπιδρούν με εχθρούς και να συμμετέχουν σε μάχες σε πραγματικό χρόνο. Για να γίνει αυτό εφικτό, χρειάστηκε να συνδυαστούν αρκετές διαφορετικές λειτουργίες και τεχνολογίες, όπως η απεικόνιση γραφικών, η διαχείριση animations, η δημιουργία χαρτών, η ανίχνευση συγκρούσεων, ο έλεγχος των εχθρών, η επικοινωνία client-server και η συγχρονισμένη ενημέρωση της κατάστασης του παιχνιδιού. Η ανάπτυξη ενός τέτοιου συστήματος απαιτούσε όχι μόνο την υλοποίηση των επιμέρους μηχανισμών, αλλά και τη σωστή συνεργασία τους, ώστε το παιχνίδι να λειτουργεί ομαλά, χωρίς σημαντικές καθυστερήσεις και με αποδεκτή εμπειρία για τον χρήστη.

6.1 Συμπεράσματα

Η ανάπτυξη του παιχνιδιού αποτέλεσε μια σύνθετη διαδικασία, καθώς απαιτήθηκε ο συνδυασμός πολλών διαφορετικών λειτουργιών σε ένα ενιαίο σύστημα. Κατά τη διάρκεια της εργασίας υλοποιήθηκαν μηχανισμοί για την κίνηση των παικτών, την επιλογή κλάσης χαρακτήρα, τη διαχείριση διαφορετικών περιοχών, τη φόρτωση χαρτών, την εμφάνιση animations, τη συμπεριφορά των εχθρών, τις συγκρούσεις και τις μάχες σε πραγματικό χρόνο. Παράλληλα, ιδιαίτερη σημασία είχε η επικοινωνία μεταξύ client και server, καθώς ο server έπρεπε να διαχειρίζεται την κατάσταση του παιχνιδιού και να ενημερώνει τους clients με τρόπο συγχρονισμένο και αποδοτικό.

Ένα βασικό συμπέρασμα που προέκυψε είναι ότι σε ένα multiplayer παιχνίδι πραγματικού χρόνου η απόδοση δεν εξαρτάται μόνο από την απεικόνιση των γραφικών στον client, αλλά και από την ταχύτητα με την οποία ο server μπορεί να επεξεργαστεί τη λογική του παιχνιδιού. Η κίνηση των παικτών, η συμπεριφορά των εχθρών, οι επιθέσεις και οι συγκρούσεις πρέπει να υπολογίζονται μέσα σε συγκεκριμένο χρονικό διάστημα, ώστε το παιχνίδι να παραμένει ομαλό και άμεσο για τον χρήστη. Οι αρχικές μετρήσεις έδειξαν ότι ο client διατηρούσε γενικά σταθερό ρυθμό ανανέωσης, ενώ η μεγαλύτερη επιβάρυνση εμφανιζόταν στην πλευρά του server, κυρίως όταν αυξανόταν ο αριθμός των ενεργών εχθρών.

Μέσα από την αξιολόγηση της απόδοσης εντοπίστηκαν τα σημεία που προκαλούσαν τη μεγαλύτερη επιβάρυνση και εφαρμόστηκαν κατάλληλες βελτιστοποιήσεις. Ο περιορισμός της επεξεργασίας των εχθρών μόνο στις ενεργές περιοχές, η μείωση της συχνότητας ενημέρωσης της λογικής των εχθρών και η εξομάλυνση της κίνησής τους στον client συνέβαλαν στη βελτίωση της συνολικής απόδοσης. Μετά τις αλλαγές αυτές, ο μέσος χρόνος επεξεργασίας tick του server παρέμεινε κάτω από το επιθυμητό όριο των 20 ms, ακόμη και σε σενάρια με περισσότερους παίκτες και μεγάλο αριθμό ενεργών εχθρών.

Με βάση την τελική μορφή της εφαρμογής και τα αποτελέσματα των μετρήσεων, το αποτέλεσμα κρίνεται ικανοποιητικό. Κατά τη διάρκεια της ανάπτυξης εντοπίστηκαν και διορθώθηκαν αρκετά ζητήματα που αφορούσαν τη σύνδεση client-server, τη μετάβαση μεταξύ περιοχών, την κίνηση των χαρακτήρων, τη συμπεριφορά των εχθρών και τη συνολική απόδοση του συστήματος. Οι αλλαγές που εφαρμόστηκαν συνέβαλαν στη σταθερότερη λειτουργία του παιχνιδιού, ενώ οι τελικές μετρήσεις έδειξαν ότι ο server παρέμεινε εντός του επιθυμητού ορίου των 20 ms ανά tick στα σενάρια δοκιμών. Επομένως, το παιχνίδι μπορεί να θεωρηθεί ένα λειτουργικό και αποδοτικό πρωτότυπο, το οποίο καλύπτει τις βασικές απαιτήσεις ενός MMORPG περιβάλλοντος και μπορεί να αποτελέσει βάση για περαιτέρω ανάπτυξη.

6.2 Μελλοντικές επεκτάσεις

Το παιχνίδι που αναπτύχθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας αποτελεί ένα λειτουργικό πρωτότυπο, το οποίο μπορεί να επεκταθεί περαιτέρω με την προσθήκη νέων μηχανισμών και περιεχομένου. Παρότι υλοποιήθηκαν οι βασικές λειτουργίες ενός 2D MMORPG, λόγω χρονικών περιορισμών δεν ήταν δυνατό να ενσωματωθούν όλες οι δυνατότητες που θα μπορούσαν να υπάρχουν σε ένα πιο ολοκληρωμένο MMORPG. Για τον λόγο αυτό, υπάρχουν αρκετές μελλοντικές επεκτάσεις που θα μπορούσαν να βελτιώσουν τόσο τον τρόπο παιχνιδιού όσο και τη συνολική εμπειρία του χρήστη.

Μία βασική μελλοντική επέκταση του παιχνιδιού θα μπορούσε να είναι ο εμπλουτισμός του περιεχομένου του με περισσότερες διαφορετικές κλάσεις χαρακτήρων, εχθρούς και περιοχές. Η προσθήκη νέων κλάσεων χαρακτήρων θα έδινε στους παίκτες περισσότερες επιλογές στον τρόπο παιχνιδιού, καθώς κάθε χαρακτήρας θα μπορούσε να διαθέτει διαφορετικά στατιστικά, ικανότητες και ρόλο στη μάχη. Με αυτόν τον τρόπο, η εμπειρία του παίκτη θα γινόταν πιο ποικιλόμορφη και θα υπήρχε μεγαλύτερο κίνητρο για επαναληπτικό παιχνίδι.

Επιπλέον, η προσθήκη νέων τύπων εχθρών θα μπορούσε να κάνει τις μάχες πιο ενδιαφέρουσες και απαιτητικές. Οι νέοι εχθροί θα μπορούσαν να διαθέτουν διαφορετικά μοτίβα κίνησης, επιθέσεις, επίπεδα δυσκολίας και ειδικές συμπεριφορές, ώστε ο παίκτης να χρειάζεται να προσαρμόζει τη στρατηγική του ανάλογα με την περίπτωση. Παράλληλα, η δημιουργία περισσότερων περιοχών προς εξερεύνηση θα επέκτεινε τον κόσμο του παιχνιδιού και θα έδινε τη δυνατότητα για μεγαλύτερη πρόοδο και ποικιλία στο περιβάλλον.

Μια ακόμη σημαντική επέκταση θα ήταν η εισαγωγή νέων αποστολών. Οι αποστολές θα μπορούσαν να περιλαμβάνουν διαφορετικούς στόχους, όπως την εξόντωση συγκεκριμένων εχθρών, την εξερεύνηση νέων περιοχών, τη συλλογή αντικειμένων ή την ολοκλήρωση γεγονότων μέσα στον κόσμο του παιχνιδιού. Με την προσθήκη αποστολών, ο παίκτης θα είχε πιο ξεκάθαρη κατεύθυνση και περισσότερα κίνητρα για να συνεχίσει την πρόοδό του μέσα στο παιχνίδι.

Συνολικά, οι παραπάνω επεκτάσεις θα μπορούσαν να ενισχύσουν σημαντικά τη διάρκεια ζωής και το ενδιαφέρον του παιχνιδιού. Η προσθήκη περισσότερου και πιο διαφοροποιημένου περιεχομένου θα έκανε τον κόσμο πιο πλούσιο, θα βελτίωνε την εμπειρία εξερεύνησης και θα έφερνε το παιχνίδι πιο κοντά στη μορφή ενός ολοκληρωμένου MMORPG.

Βιβλιογραφία

[1] A. De Luisa, J. Hartman, D. Nabergoj, S. Pahor, M. Rus, B. Stevanoski, J. Demšar, and E. Štrumbelj, "Predicting the Popularity of Games on Steam," arXiv, arXiv:2110.02896, 2021. [Online]. Available: https://arxiv.org/abs/2110.02896. [Accessed: 20-Oct-2025].

[2] A. H. Aldossari and K. A. Almutairi, "The association between playing digital games and social and psychological factors among college students," Int. J. Environ. Res. Public Health, vol. 19, no. 3, p. 1595, 2022. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/35171097/. [Accessed: 20-Oct-2025].

[3] F. Pallavicini and A. Pepe, "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, vol. 25, no. 6, pp. 329–405, 2022.

[4] H. K. Pontes and M. D. Griffiths, "The role of structural characteristics in problematic video game play: A systematic review," Cyberpsychology, Behavior, and Social Networking, vol. 17, no. 6, pp. 359–366, 2014.

[5] "Online Multiplayer Video Game Market Size, Share, Growth, and Industry Analysis, By Type (Action, RPG, Simulation, Strategy), By Application (Online Gaming Platforms, Mobile Apps, PC Gaming), Regional Insights and Forecast to 2033," MarketGrowthReports, Nov. 10, 2025. [Online]. Available: https://www.marketgrowthreports.com/market-reports/online-multiplayer-video-game-market-114209. [Accessed: 12-Nov-2025].

[6] "Why are game developers in demand? Key factors explained," Techneeds, Jun. 25, 2025. [Online]. Available: https://www.techneeds.com/2025/06/25/why-are-game-developers-in-demand-key-factors-explained/. [Accessed: 12-Nov-2025].

[7] C. A. Steinkuehler and D. Williams, "Where everybody knows your (screen) name: Online games as 'third places'," Journal of Computer-Mediated Communication, vol. 11, no. 4, pp. 885–909, 2006.

[8] P. J. C. Adachi and T. Willoughby, "More than just violence: The importance of competition in the enjoyment of violent video games," Journal of Youth and Adolescence, vol. 42, no. 7, pp. 1041–1052, 2013.

[9] N. Vegt, V. Visch, A. Vermeeren, and H. de Ridder, "Player experiences and behaviors in a multiplayer game: Designing game rules to change interdependent behavior," International Journal of Serious Games, vol. 3, no. 4, p. 150, 2016.

[10] M. Consalvo and C. Paul, "Real players: The evolution of role-playing games across digital spaces," Games and Culture, vol. 13, no. 7, pp. 715–732, 2018.

[11] J. Smed, H. Hakonen, and A. M. Knuutila, "Aspects of networking in multiplayer computer games," The Electronic Library, vol. 20, no. 2, pp. 87–97, 2002.

[12] "Legends of Aria – Game Review," MMOs.com. [Online]. Available:

https://mmos.com/review/legends-of-aria. [Accessed: 12-Nov-2025].

[13] “New World (video game),” Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/New_World_(video_game). [Accessed: 12-Nov-2025].

[14] S. Machkovech, “How Warcraft III birthed a genre, changed a franchise, and earned a Reforge-ing,” Ars Technica, Jan. 21, 2020. [Online]. Available: https://arstechnica.com/features/2020/01/how-warcraft-iii-birthed-a-genre-changed-a-franchise-and-earned-a-reforge-ing/. [Accessed: 12-Nov-2025].

[15] T. Quirk, “Rogue Heroes: Ruins of Tasos Review – Dungeon delving with your friends,” CheckpointGaming.net, Feb. 2021. [Online]. Available: https://checkpointgaming.net/reviews/2021/02/rogue-heroes-ruins-of-tasos-review-dungeon-delving-with-your-friends/. [Accessed: 12-Nov-2025].

[16] M. Lutz, *Learning Python*, 5th ed. Sebastopol, CA, USA: O’Reilly Media, 2013.

[17] A. M. Jones, “Teaching game development using Python,” *Journal of Computing Sciences in Colleges*, vol. 26, no. 6, pp. 73–79, Jun. 2011.

[18] J. Novak, *Game Development Essentials: An Introduction*, 3rd ed. Clifton Park, NY, USA: Delmar Cengage Learning, 2012.

[19] P. Vincent, *Arcade Documentation*, Arcade Academy, 2023. [Online]. Available: https://api.arcade.academy. [Accessed: 27-Dec-2025].

[20] J. Gregory, *Game Engine Architecture*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2014.

[21] D. Shreiner, G. Sellers, J. Kessenich, and B. M. Licea-Kane, *OpenGL Programming Guide*, 8th ed. Boston, MA, USA: Addison-Wesley, 2013.

[22] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA, USA: Pearson, 2016.

[23] Microsoft, *Visual Studio Code – Getting Started*, 2023. [Online]. Available: https://code.visualstudio.com/docs/getstarted/introvideos . [Accessed: 27-Dec-2025].

[24] GitHub, Inc., *About GitHub*, 2023. [Online]. Available: https://docs.github.com/en/get-started/quickstart/github-flow. [Accessed: 27-Dec-2025].

[25] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. Boston, MA, USA: Pearson, 2011.

[26] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. Sebastopol, CA, USA: O’Reilly Media, 2013.

[27] Python Software Foundation, *asyncio — Asynchronous I/O*, 2023. [Online]. Available: https://docs.python.org/3/library/asyncio.html. [Accessed: 27-Dec-2025].

[28] Python Software Foundation, *sqlite3 — DB-API 2.0 interface for SQLite databases*, 2023. [Online]. Available: https://docs.python.org/3/library/sqlite3.html. [Accessed: 27-Dec-2025].

[29] Thorbjørn Lindeijer, *Tiled Map Editor*, 2023. [Online]. Available: https://www.mapeditor.org. [Accessed: 27-Dec-2025].

[30] JGraph Ltd., *draw.io*, 2023. [Online]. Available: https://www.drawio.com/. [Accessed: 27-Dec-2025].

[31] Google, *Gemini: A Family of Multimodal Large Language Models*, 2023. [Online]. Available: https://ai.google.dev/gemini-api/docs. [Accessed: 27-Dec-2025].

[32] CraftPix, *Free 2D Game Assets*, 2023. [Online]. Available: https://craftpix.net/freebies/. [Accessed: 27-Dec-2025].

[33] OpenGameArt, *OpenGameArt.org – Free Game Assets*, 2023. [Online]. Available: https://opengameart.org. [Accessed: 27-Dec-2025].

[34] itch.io, *Game Assets*, 2023. [Online]. Available: https://itch.io/game-assets. [Accessed: 27-Dec-2025].

[35] Kaleido AI, *remove.bg – Remove Image Background*, 2023. [Online]. Available: https://www.remove.bg/. [Accessed: 27-Dec-2025].

[36] Piskel, "Piskel - Free online sprite editor," *Piskel*. [Online]. Available: https://www.piskelapp.com/. [Accessed: 19-Feb-2026].

Παράρτημα Α:

Παράρτημα Β:

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.