

Міністерство освіти і науки України
Харківський радіотехнічний фаховий коледж
Циклова комісія “Програмування та інформаційних технологій”

КУРСОВА РОБОТА

з дисципліни

«Конструювання програмного забезпечення»

на тему:

«Розробка 3D гри «Last Fort» у жанрі Tower Defense за допомогою Unity»

Виконав:

студент групи ПІ-431

Лучанінов Петро Олександрович

Керівник роботи:

к.т.н. Федосєєва А.О.

Харків 2025

ЗАЛІКОВИЙ ЛИСТ

ХАРКІВСЬКИЙ РАДІОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

Циклова комісія ПРОГРАМУВАННЯ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Спеціальність 121 «Інженерія програмного забезпечення»

Курс – 4 Група - ПІ-431 Семестр – 7

КУРСОВА РОБОТА

студента Лучанінова Петра Олександровича

на тему: «Розробка 3D гри «Last Fort» у жанрі Tower Defense за допомогою Unity»

ДОПУЩЕНИЙ ДО ЗАХИСТУ

Керівник роботи _____/к.т.н. Федосєєва Аліна Олександрівна/

ОЦІНКА _____

Члени комісії _____/Федосєєва А.О./

_____ /Лісаєв Д.В./

_____ /Багацький Д.В. /

«28» січня 2025

ХАРКІВСЬКИЙ РАДІОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

Циклова комісія	ПРОГРАМУВАННЯ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	
Спеціальність	121 «Інженерія програмного забезпечення»	
Курс – 4	Група – ПІ-431	Семестр – 7

ЗАВДАННЯ НА КУРСОВУ РОБОТУ

Студента Лучанінова Петра Олександровича

Тема курсової роботи: «Розробка 3D гри «Last Fort» у жанрі Tower Defense за допомогою Unity»

Розробити програмний продукт:

2-D гра у жанрі Tower Defense, яка дозволяє отримати стратегічний ігровий досвід через стратегічне розміщення захисних споруд для протистояння хвилям ворогів.

Зміст пояснювальної записки:

Вступ

1. Аналіз предметної області

1.1 Моделювання предметної області

1.2 Побудова IDEF0-діаграм

1.3 Проєктування бази даних

2. Побудова профіля вимог до програмного забезпечення

3. Технічне завдання програмного забезпечення

4. Планування тестування програмного забезпечення

5. Виконання тестування програмного забезпечення

6. Опис програмного забезпечення

6.1 Засоби розробки програмного забезпечення

6.2 Інструкція користувача

Висновки

Перелік використаної літератури

Додатки

Дата видачі завдання: 01 жовтня 2024 р.

Дата здачі роботи: 28 січня 2025 р.

Керівник роботи _____/А.О. Федосєєва/

Розглянуто на засіданні ЦК

Голова ЦК _____/А.О. Федосєєва/

ЗМІСТ

ВСТУП.....	5
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	6
1.1 Моделювання предметної області	6
1.2 Побудова IDEF0-діаграм	15
2 ПОБУДОВА ПРОФІЛЮ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	19
3 ТЕХНІЧНЕ ЗАВДАННЯ.....	27
4 ПЛАНУВАННЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	34
5 ВИКОНАННЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	41
6 ВИКОНАННЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	44
6.1 Засоби розробки програмного забезпечення	44
6.2 Інструкція користувача	45
ВИСНОВКИ.....	51
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	52
ДОДАТОК А	53
ДОДАТОК Б.....	63
ДОДАТОК В	72
ДОДАТОК Г.....	81
ДОДАТОК Ґ	90
ДОДАТОК Д	98
ДОДАТОК Е	107
ДОДАТОК Ж	117

ВСТУП

Розробка ігор є однією з найпопулярніших галузей сучасного програмування, оскільки вона поєднує в собі не лише технологічні інновації, а й креативні рішення. Ігри в жанрі Tower Defense користуються великою популярністю завдяки їхній стратегічній складовій, яка дозволяє гравцям проявляти логічне мислення та планувати свої дії на кілька кроків вперед. Особливо важливим є той факт, що жанр Tower Defense дозволяє варіювати складність гри, надаючи можливість розробникам створювати продукти як для казуальних, так і для досвідчених гравців.

Актуальність теми полягає в тому, що попри численність ігор у жанрі Tower Defense, багато з них не пропонують достатньо нових механік, що призводить до одноманітності на пізніх етапах гри. Тому розробка гри, яка б поєднувала класичні елементи жанру з інноваційними рішеннями, є важливою задачею. Крім того, використання сучасних інструментів, таких як Unity, дозволяє створювати більш якісні та оптимізовані продукти для різних платформ.

Метою роботи є розробка 2D-гри у жанрі Tower Defense на базі Unity, яка надасть гравцям можливість стратегічного мислення через грамотне розташування захисних споруд для протистояння хвилям ворогів.

У ході виконання роботи планується здійснити аналіз предметної області, дослідити існуючі аналоги ігор у жанрі Tower Defense, розробити унікальні ігрові механіки та процес, які відрізнятимуться від вже існуючих рішень, спроектувати базу даних для зберігання ігрових результатів та профілів гравців, виконати тестування програмного продукту на відповідність вимогам, а також забезпечити його стабільну та ефективну роботу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Моделювання предметної області

Предметною областю даної курсової роботи є концепції та механіки жанру Tower Defense, які використовуються у відеоіграх для створення стратегічного ігрового досвіду.

Жанр Tower Defense є піджанром стратегічних ігор, який виділяється своєю унікальною механікою. Основна мета таких ігор полягає у захисті бази чи об'єкта від хвиль атакуючих ворогів за допомогою стратегічного розташування оборонних споруд. Особливістю цього жанру є поступове ускладнення хвиль ворогів, що змушує гравця адаптувати свою стратегію, використовуючи ресурси для побудови нових або покращення існуючих веж.

Кожна вежа у грі має свої унікальні характеристики, такі як радіус дії, сила атаки чи спеціальні ефекти, наприклад, уповільнення ворогів. Вороги, у свою чергу, можуть мати різні властивості, як-от підвищена швидкість, стійкість до певних атак або особливі здібності. Така різноманітність сприяє динамічності ігрового процесу, створюючи нові виклики для гравця.

Ще однією важливою складовою жанру є розвиток і прогресія. Гравці отримують ресурси за знищення ворогів, що дозволяє їм покращувати оборонні споруди, підвищуючи їх ефективність або додаючи нові функції. Баланс між складністю гри, різноманітністю ворогів і веж, а також зручністю інтерфейсу є ключовим для успішної реалізації проєктів у цьому жанрі.

Найвідоміші програмні продукти цього жанру мають свої унікальні особливості, які приваблюють гравців:

– Dota [1] (рисунок 1.1) — гра, що поєднує елементи Tower Defense з жанром МОБА. Переваги: складна тактична гра, велика кількість персонажів з унікальними здібностями. Недоліки: висока складність для новачків, тривалий час вивчення гри.

– Kingdom Rush [2] (рисунок 1.2) — класична Tower Defense гра з 2D-графікою та фентезійним сетингом. Переваги: різноманіття оборонних споруд та ворогів, добре продумана прогресія рівнів. Недоліки: одноманітність геймплею на пізніх етапах, обмежені можливості для кастомізації ігрового процесу.

– Dungeon Defenders (рисунок 1.3) [3] — це суміш Tower Defense і Action RPG, де гравець одночасно керує персонажем і будує оборонні споруди. Переваги: поєднання динамічного екшену з класичними елементами захисту бази, кооперативний режим. Недоліки: деякі елементи гри можуть здатися складними для нових гравців.

На рисунках 1.1–1.3 представлені зображення цих ігор:



Рисунок 1.1 – Скріншот з гри Dota 2



Рисунок 1.2 – Скріншот з гри Kingdom Rush



Рисунок 1.3 – Скріншот з гри Dungeon Defenders 2

Усі ці ігри мають схожу основну концепцію, проте їхні механіки реалізовані по-різному. Спільними рисами є стратегічне планування, управління

ресурсами та використання веж з унікальними здібностями для відбиття атак ворогів.

Водночас недоліки багатьох існуючих ігор полягають у тому, що після кількох рівнів геймплею може стати одноманітним, а нові механіки або відсутні, або вводяться занадто рідко.

Last Fort, новий продукт, має на меті виправити деякі з цих недоліків, запропонувавши інноваційні механіки, більше можливостей для варіативного розвитку оборонних споруд і врахування потреб гравців у новизні.

Програмний продукт буде розроблено на движку Unity, що забезпечить високу якість візуальної складової, зручність управління та можливість подальшого розширення функціональності. Використання Unity також дозволяє легко інтегрувати нові елементи геймплею, такі як різні типи веж, хвилі ворогів та вдосконалення ігрового процесу.

Для розробки програмного продукту було проведено всебічний аналіз особливостей жанру Tower Defense, що дозволило детально зрозуміти його специфіку.

Оборонні вежі займають центральне місце у грі, оскільки кожна з них має свої унікальні характеристики: дальність дії, швидкість атаки, сила та додаткові ефекти, такі як уповільнення ворогів. Це додає глибини геймплею, адже гравець повинен не тільки вибирати місце розміщення вежі, але й тип вежі, який найбільш ефективний у конкретній ситуації.

Крім того, важливою частиною ігрового процесу є управління ресурсами. Гравець отримує ресурси за знищення ворогів, які можуть бути витрачені на будівництво нових веж або покращення вже наявних. Таким чином, гравець має приймати обдумані рішення щодо розподілу ресурсів для ефективного протистояння ворогам.

У деяких іграх, таких як Dungeon Defenders, додатково вводяться керовані герої, що додає новий рівень стратегії. Гравець повинен одночасно керувати своїм персонажем та оборонними спорудами, що робить ігровий процес ще більш захоплюючим та динамічним.

Для більш глибокого розуміння жанру Tower Defense було розглянуто його класифікаційні аспекти, які дозволяють виділити різні типи програмного забезпечення та їх основні характеристики.

Програмне забезпечення для ігор у жанрі Tower Defense можна класифікувати за кількома важливими параметрами. Одним із них є тип графіки. Більшість ігор цього жанру використовують 2D або 3D графіку. Наприклад, Kingdom Rush — це класична 2D гра, тоді як Dungeon Defenders використовує 3D графіку, що забезпечує більше можливостей для візуалізації та взаємодії з середовищем.

Іншим важливим критерієм є тип гри. Ігри можуть бути класичними Tower Defense, як у випадку Kingdom Rush, де головний акцент робиться на розміщенні веж і захисті бази, або ж поєднувати елементи інших жанрів. Наприклад, Dota включає в себе елементи МОБА, що значно урізноманітнює ігровий процес, додаючи рольові елементи і командну гру.

Режим гри також є важливим класифікаційним параметром. Деякі ігри пропонують лише одиночний режим, тоді як інші підтримують кооперативну або мультиплеєрну гру. Dungeon Defenders дозволяє гравцям об'єднуватися в команди для спільної оборони, що робить ігровий процес більш динамічним і соціальним.

Ще одним аспектом є платформи, на яких доступні ігри. Tower Defense ігри можуть бути створені для мобільних платформ, ПК або консолей. Наприклад, Kingdom Rush широко представлена на мобільних пристроях, тоді як Dungeon Defenders підтримує гру на консолях і ПК, що розширює аудиторію.

Модель монетизації також є важливим аспектом класифікації. Ігри можуть бути платними, з одноразовою покупкою, або використовувати модель free-to-play з внутрішньоігровими покупками. Наприклад, Dota використовує безкоштовну модель з покупками всередині гри, тоді як Dungeon Defenders є платною грою з додатковими DLC.

Для оцінки актуальності жанру Tower Defense важливо розглянути статистичні дані, які відображають його популярність серед гравців та динаміку розвитку на ринку ігрового програмного забезпечення.

Жанр Tower Defense залишається популярним серед стратегічних ігор як на мобільних платформах, так і на ПК та консолях. За останні кілька років цей жанр стабільно займає значну частку ринку, особливо серед мобільних ігор. За даними аналітичних компаній, у 2023 році Tower Defense ігри становили приблизно 5% загального ринку мобільних стратегічних ігор. Ця популярність пояснюється простотою управління, відносно низькими вимогами до ресурсів пристроїв та глибокою стратегічною складовою, що залучає широку аудиторію [4].

Більшість ігор цього жанру поширюються через мобільні платформи, такі як Google Play та App Store. Наприклад, Kingdom Rush завантажили понад 10 мільйонів разів, що підкреслює високу популярність жанру серед мобільних гравців.

На ПК і консолях також існує значна аудиторія Tower Defense ігор, особливо тих, що поєднують у собі елементи інших жанрів. Dungeon Defenders стала популярною завдяки поєднанню Tower Defense з Action RPG, що зробило її однією з найпродаваніших ігор свого жанру на платформах Steam та Xbox. У Steam гра отримала понад 2 мільйони активних гравців на момент свого піку, і досі залишається популярною серед шанувальників жанру.

Для проектування програмного продукту було розроблено діаграму класів (рисунки 1.4), яка відображає ключові компоненти системи та їх взаємозв'язки. Вона дозволяє визначити структуру класів, їхні атрибути, методи та взаємодію між об'єктами, що є основою для подальшої реалізації гри. Діаграма класів є важливою складовою проектування, оскільки вона наочно демонструє структуру системи та ключові характеристики її елементів. Для гри у жанрі Tower Defense діаграма відображає основні об'єкти, такі як гравець, вороги, оборонні вежі, хвилі ворогів, карта та гра в цілому. Це допомагає зрозуміти взаємодію між компонентами гри, їхню поведінку та спадкування властивостей, що сприяє

ефективному впровадженню ігрових механік та забезпеченню стабільної роботи системи.

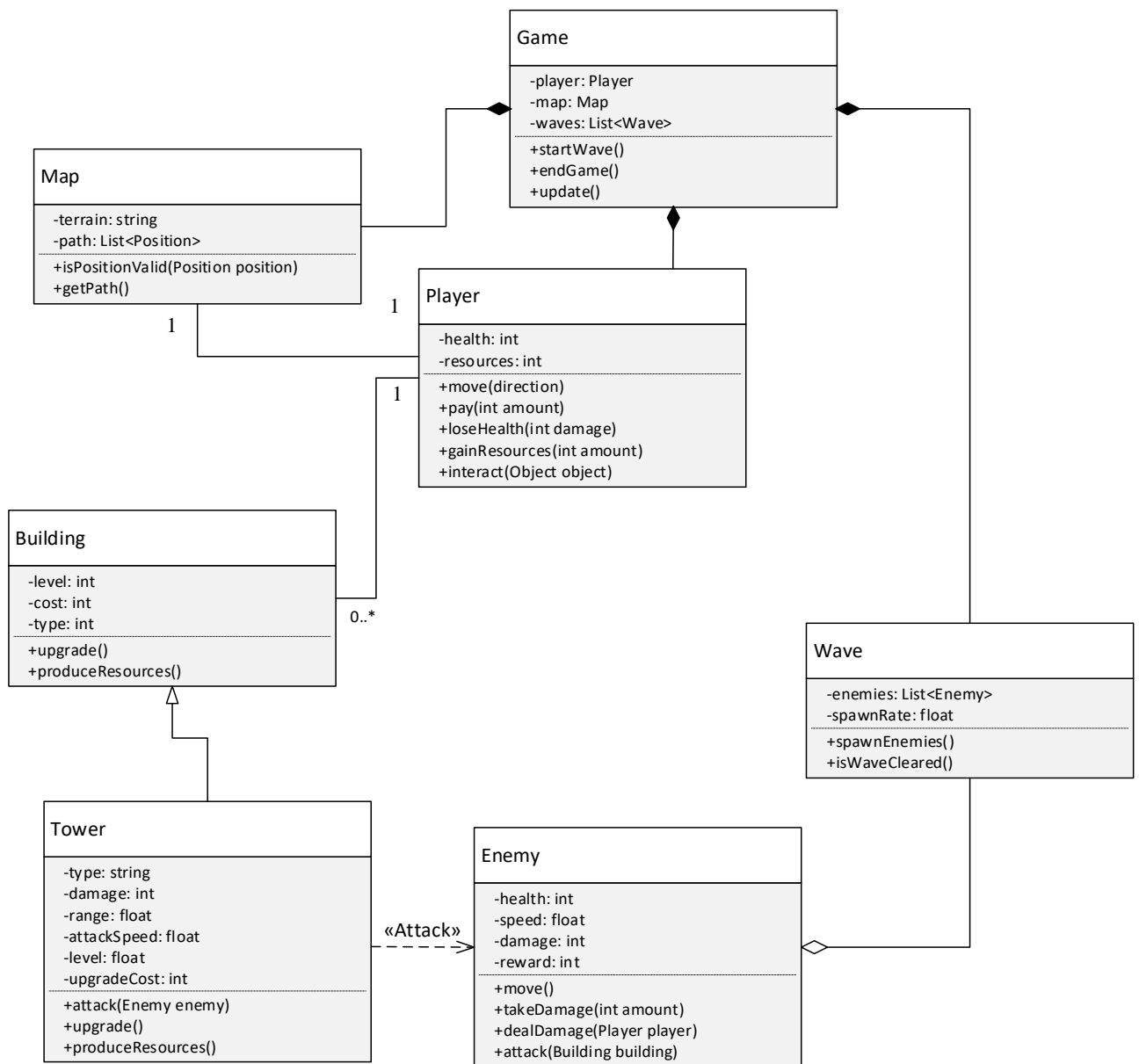


Рисунок 1.4 – UML-діаграма класів для гри «Last Fort»

Діаграма класів відображає основні компоненти гри “Last Fort” та їх взаємозв'язки. Вона містить ключові класи, кожен із яких представляє окремий аспект гри:

- **Game (Гра)**: Основний клас, який координує взаємодію між гравцем, картою та хвилями ворогів. Містить такі атрибути: `player`, `map`, `waves`. Методи: `startWave()`, `endGame()`, `update()`.

- Player (Гравець): Клас, що представляє гравця, який контролює ігровий процес. Має атрибути `health` і `resources`, а також методи для руху, витрати ресурсів, отримання шкоди, збирання ресурсів та взаємодії з об'єктами.
- Map (Карта): Клас, який визначає ігрове середовище, його атрибути (`terrain`, `path`) і методи (`isPositionValid()`, `getPath()`), що забезпечують доступність позицій та рух ворогів.
- Building (Будівля): Абстрактний клас, що представляє будівлі на карті. Включає атрибути `level`, `cost`, `type` і методи для покращення (`upgrade()`) та виробництва ресурсів (`produceResources()`).
- Tower (Вежа): Дочірній клас від `Building`, який відображає оборонну вежу. Додає атрибути, такі як `damage`, `range`, `attackSpeed`, та методи `attack()`, `upgrade()` і `produceResources()`.
- Enemy (Ворог): Клас, що відображає ворогів, які атакують базу гравця. Має атрибути `health`, `speed`, `damage`, а також методи для руху (`move()`), отримання шкоди (`takeDamage()`), та нанесення шкоди гравцю або будівлям (`dealDamage()`).
- Wave (Хвиля): Клас, що представляє хвилю ворогів, з атрибутами `enemies` і `spawnRate`, а також методами для спавну ворогів (`spawnEnemies()`) та перевірки завершення хвилі (`isWaveCleared()`).

Для проектування програмного продукту було також розроблено діаграму прецедентів (рисунок 1.5), яка демонструє основні сценарії використання системи гравцем та їхню взаємодію. Діаграма відображає, які дії може виконувати гравець у грі та як система реагує на ці дії, що є важливим етапом при розробці функціональних вимог гри.

Діаграма прецедентів демонструє основні функції гри, з якими взаємодіє гравець. Вона охоплює сценарії, пов'язані з управлінням персонажем, побудовою споруд, управлінням підданими, збиранням ресурсів, а також налаштуванням та збереженням прогресу гри. Діаграма прецедентів для гри “Last Fort” зображена на рисунку 1.5.

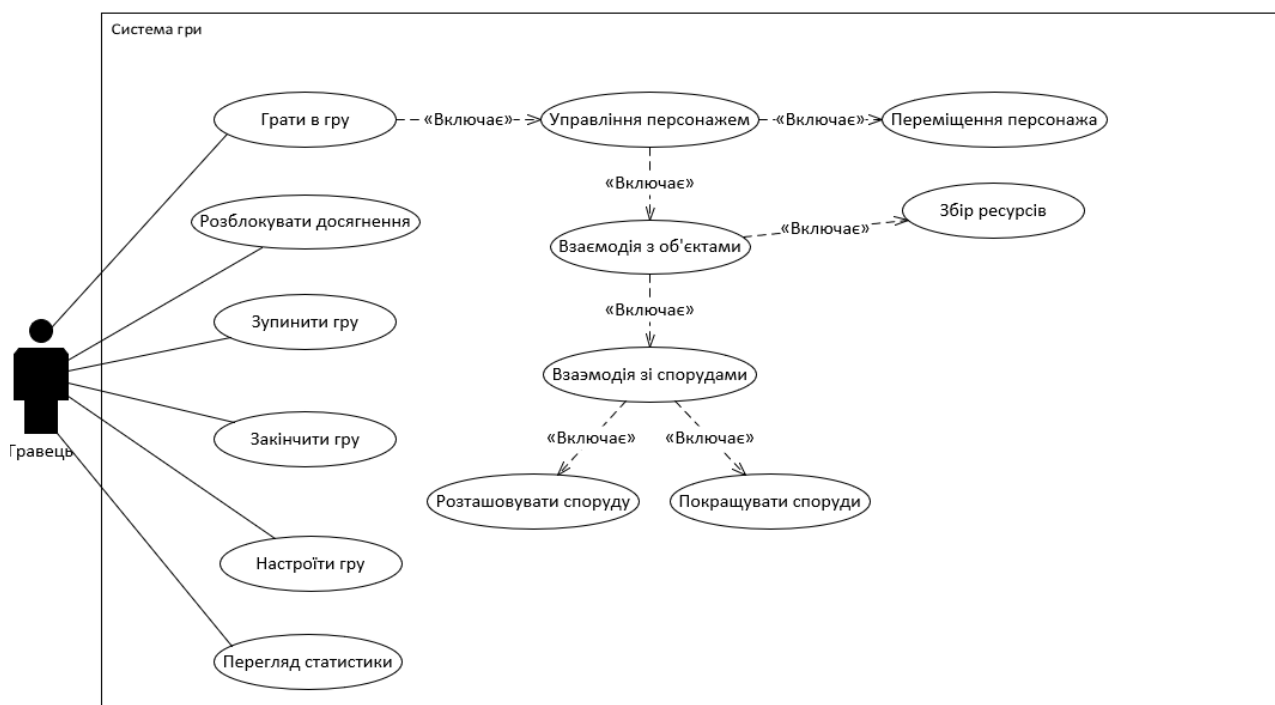


Рисунок 1.5 – UML-діаграма прецедентів для гри «Last Fort»

Основним актором у діаграмі прецедентів є гравець, який безпосередньо взаємодіє з системою, керує персонажем, виконує основні дії у грі.

Основними прецедентами на діаграмі є:

- Грати в гру: включає управління персонажем, його переміщення, збір ресурсів та взаємодію з об'єктами гри.
- Взаємодія зі спорудами: охоплює побудову нових споруд, їх покращення, а також наймання підданих та надання їм наказів.
- Розблокувати досягнення: гравець може досягти спеціальних цілей у процесі гри, що підвищує мотивацію.
- Зупинити гру та Закінчити гру: дозволяють гравцю контролювати ігровий процес.
- Налаштування гри та Перегляд статистики: забезпечують зручність для гравця у зміні параметрів гри та аналізі своїх результатів.

1.2 Побудова IDEF0-діаграм

IDEF0-діаграма є важливим інструментом для моделювання функціональних процесів програмного продукту. Ця діаграма дозволяє наочно відобразити основні функції системи, їхні входні та вихідні дані, механізми виконання та обмеження. У контексті розробки гри "Last Fort" IDEF0-діаграма допомагає зрозуміти ключові процеси взаємодії користувача з грою, а також внутрішні логічні зв'язки між компонентами гри.

IDEF0-діаграма для гри "Last Fort" зображена на рисунках 1.6-1.8.

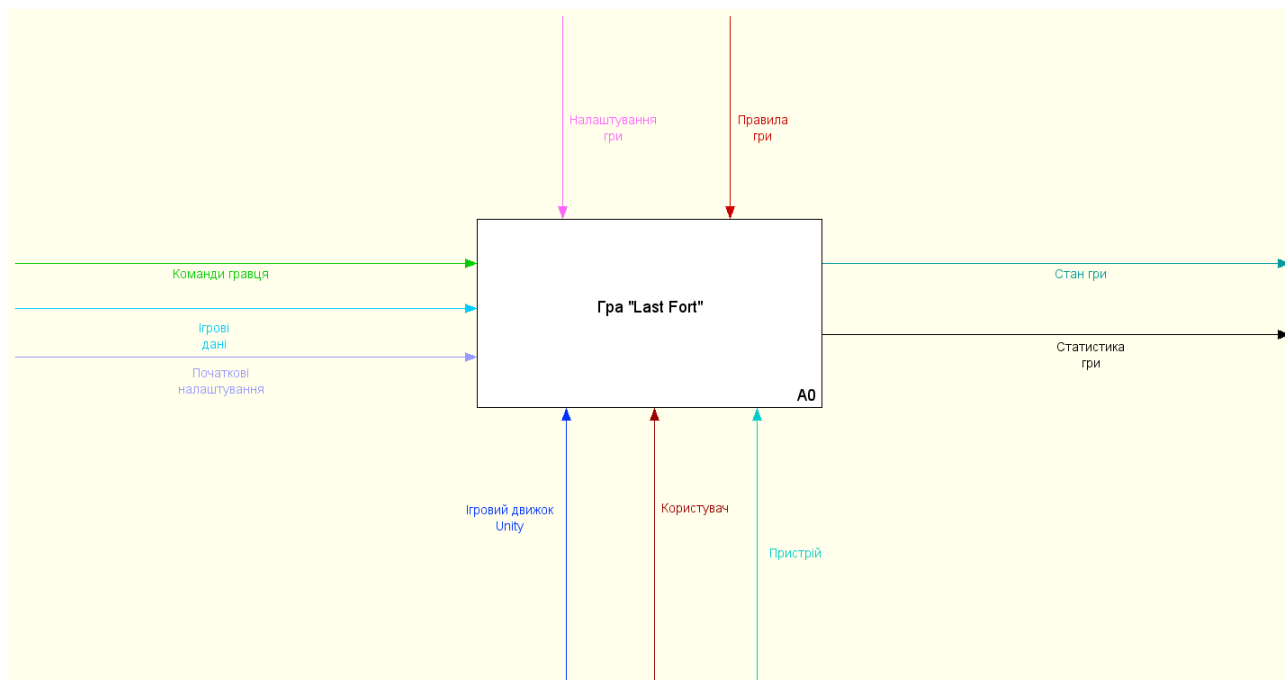


Рисунок 1.6 – Контекстна діаграма IDEF0 (A0) для гри "Last Fort"

Контекстна діаграма IDEF0 відображає загальний процес гри "Last Fort". Вона включає основні елементи, необхідні для запуску та управління грою. Входи складаються з команд гравця, ігрових даних та початкових налаштувань. Механізмами є ігровий движок Unity, інтерфейс користувача та пристрій, які забезпечують основну функціональність гри. Управління включає налаштування гри та правила, що визначають логіку та поведінку гри. На виході система генерує стан гри, статистику та збережений прогрес.

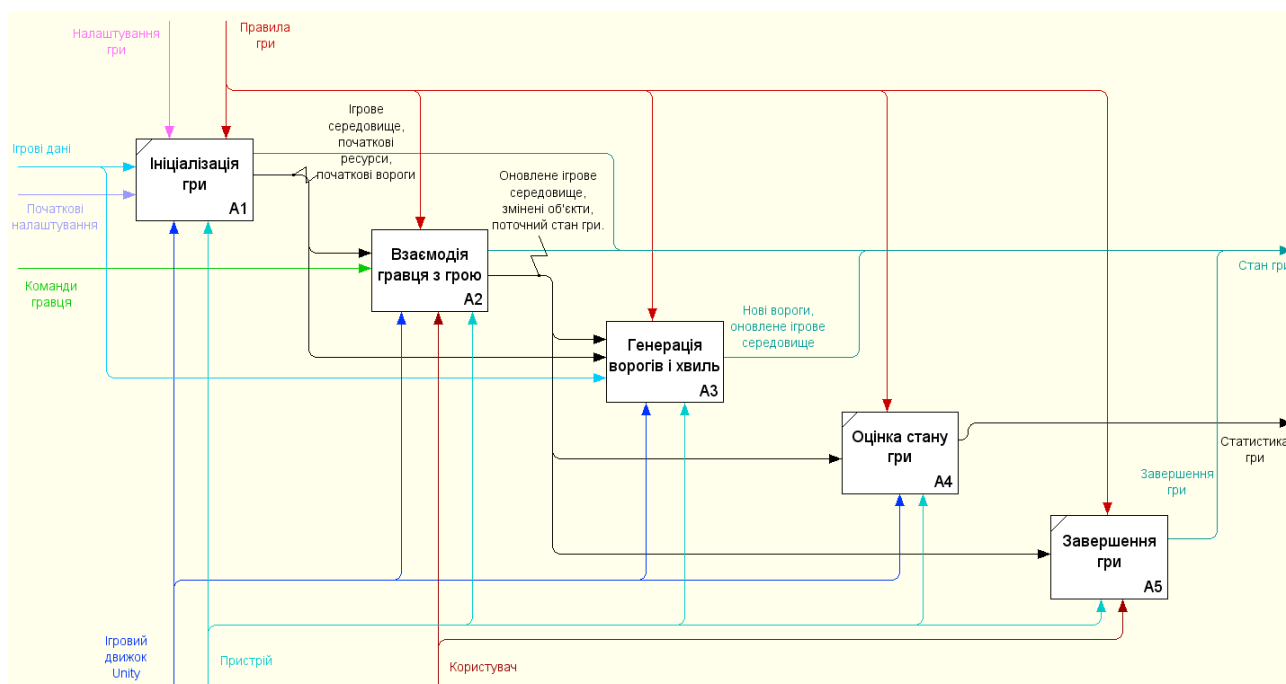


Рисунок 1.7 – Декомпозиція A0 на підрівні A1

На рисунку 1.7 показано деталізацію процесу гри до основних блоків, які включають:

- A1: Ініціалізація гри – встановлення початкових умов, таких як ресурси, середовище та початкові вороги.
- A2: Взаємодія гравця з грою – управління персонажем, спорудами, підданими та ворогами.
- A3: Генерація ворогів і хвиль – створення нових ворогів та хвиль у відповідності до поточного стану гри.
- A4: Оцінка стану гри – аналіз поточного стану для визначення перемоги, поразки або продовження гри.
- A5: Завершення гри – завершення ігрової сесії.

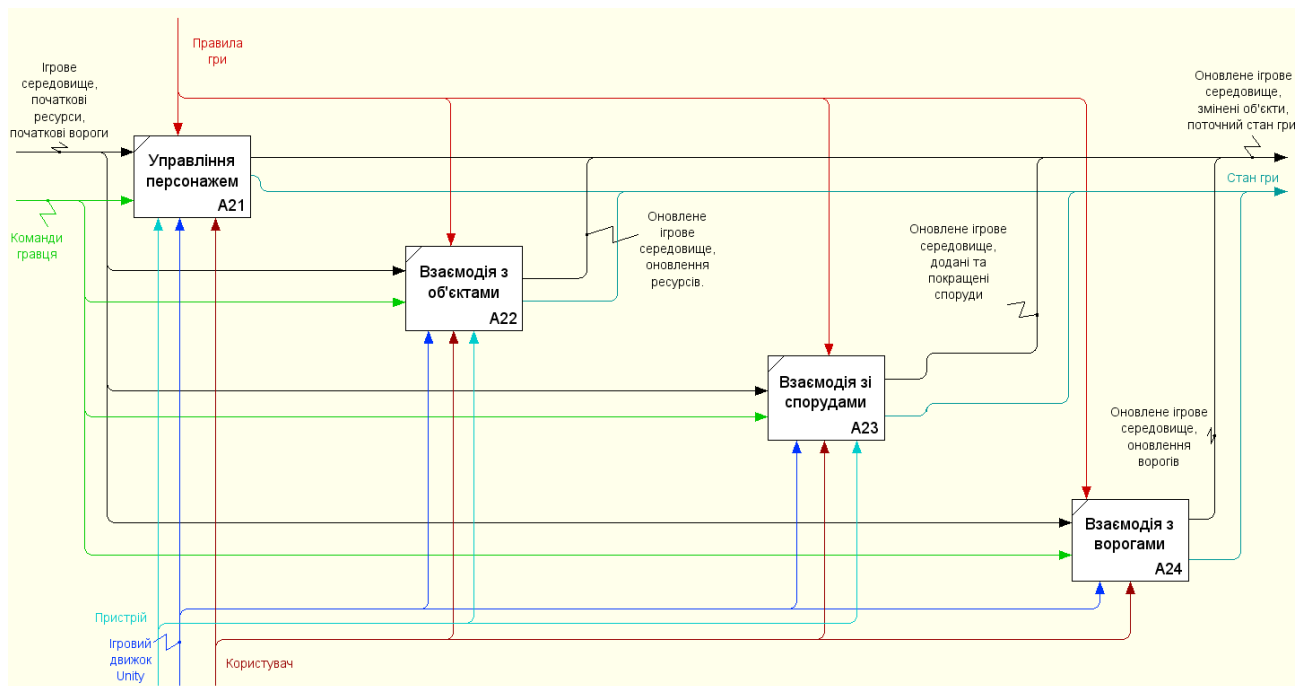


Рисунок 1.8 – Декомпозиція блоку A2: Взаємодія гравця з грою

Рисунок 1.8 деталізує взаємодію гравця з грою на підрівні A2, яка включає наступні підблоки:

- A21: Управління персонажем – відповідає за переміщення персонажа гравцем.
- A22: Взаємодія з об'єктами – охоплює збір ресурсів, взаємодію з предметами та іншими об'єктами на карті.
- A23: Взаємодія зі спорудами – включає будівництво, покращення та ремонт споруд.
- A24: Взаємодія з ворогами – відповідає за атаки та захист від ворогів.

Висновки за розділом 1:

1. Проведено всебічний аналіз предметної області розробки 2D-ігри у жанрі Tower Defense, а також вивчено існуючі аналоги у цій сфері, такі як Dota, Kingdom Rush та Dungeon Defenders.

2. Визначено переваги та недоліки аналогічних ігор, що дозволило виокремити специфічні особливості жанру.

3. Надано класифікацію програмного забезпечення у жанрі Tower Defense за різними критеріями.

4. Розглянуто статистичні дані, що підтверджують популярність жанру серед гравців на різних платформах.

5. Створено UML-діаграми, які описують структуру системи та основні сценарії взаємодії гравця з грою.

6. Розроблено IDEF0-діаграми, що моделюють функціональні процеси гри – від ініціалізації до завершення.

2 ПОБУДОВА ПРОФІЛЮ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для ефективної розробки програмного продукту гри «Last Fort» було визначено та сформовано набір функціональних та нефункціональних вимог. Ці вимоги спрямовані на забезпечення правильного функціонування системи, зручності для користувача та відповідності бізнес-цілям проекту.

Функціональні вимоги включають три основні категорії: бізнес-вимоги, вимоги користувачів та функціональні вимоги системи. Бізнес-вимоги визначають основні цілі проекту з точки зору бізнесу, що забезпечують досягнення стратегічних задач гри. Вимоги користувачів формують основні потреби користувачів у взаємодії з грою. Функціональні вимоги описують конкретні функції, які повинна реалізовувати система для досягнення цих цілей.

Функціональні вимоги до програмно забезпечення наведені в таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до програмного забезпечення

	Повнота	Однозначність	Коректність	Необхідність	Здійсненність	Перевірюваність
Бізнес вимоги						
Гра повинна стимулювати гравців проводити більше часу у грі за рахунок захопливого і стратегічного ігрового процесу, що сприятиме збільшенню кількості повторних сесій та зростанню доходів від монетизації.	+	+	+	+	+	+
Система повинна підтримувати регулярне оновлення контенту, щоб утримувати існуючих користувачів і залучати нову аудиторію, забезпечуючи зростання кількості активних користувачів.	+	+	+	+	+	+

Продовження таблиці 2.1

	Повнота	Однозначність	Коректність	Необхідність	Здійсненість	Перевірюваність
Інтерфейс гри повинен бути інтуїтивним і простим для розуміння, щоб знизити бар'єр входу для новачків і розширити потенційну користувацьку базу.	+	+	+	+	+	+
Наявність різних рівнів складності дозволить адаптувати гру до потреб як новачків, так і досвідчених гравців, підвищуючи їхній інтерес і залученість.	+	+	+	+	+	+
Вбудована система статистики повинна надавати гравцям інформацію про їхній прогрес і досягнення, водночас генеруючи аналітичні дані для розробників, що сприятиме оптимізації гри і покращенню монетизації.	+	+	+	+	+	+
Вимоги користувачів						
Як користувач, я хочу мати можливість розмішувати вежі для захисту бази, щоб ефективно протистояти хвилям ворогів	+	+	+	+	+	+
Як користувач, я хочу мати можливість покращувати вежі, щоб підвищувати їхню ефективність у боротьбі з ворогами	+	+	+	+	+	+
Як користувач, я хочу пересуватися картою, щоб досліджувати нові території	+	+	+	+	+	+
Як користувач, я хочу отримувати інформаційні повідомлення про наступну хвилю ворогів, щоб краще підготуватися до оборони	+	+	+	+	+	+

Продовження таблиці 2.1

	Повнота	Однозначність	Коректність	Необхідність	Здійсненість	Перевірюваність
Як користувач, я хочу бачити таймер до початку наступної хвили, щоб встигнути розмістити вежі або виконати інші дії	+	+	+	+	+	+
Як користувач, я хочу мати можливість переглядати статистику своїх досягнень, щоб аналізувати свій прогрес та планувати стратегію	+	+	+	+	+	+
Як користувач, я хочу мати можливість налаштовувати управління та інтерфейс гри, щоб зробити їх зручнішими для мого стилю гри	+	+	+	+	+	+
Як користувач, я хочу мати можливість ставити гру на паузу у будь-який момент, щоб перегрупуватися або розробити нову стратегію	+	+	+	+	+	+
Як користувач, я хочу завершити гру у випадку перемоги або поразки, щоб знати результат своїх дій та планувати нову сесію	+	+	+	+	+	+
Як користувач, я хочу мати можливість змінювати складність гри, щоб налаштувати її відповідно до свого рівня навичок	+	+	+	+	+	+
Функціональні вимоги						
Створення функції розміщення оборонних веж на карті за допомогою інтерфейсу користувача	+	+	+	+	+	+

Продовження таблиці 2.1

	Повнота	Однозначність	Коректність	Необхідність	Здійсненість	Перевірюваність
Створення системи покращення веж через меню взаємодії, де можна витратити ресурси на підвищення характеристик вежі	+	+	+	+	+	+
Створення системи пересування персонажа по карті за допомогою елементів управління клавіатурою або мишею	+	+	+	+	+	+
Створення інформаційної панелі, яка відображає кількість наявних ресурсів, рівень здоров'я персонажа та кількість ворогів у поточній хвилі	+	+	+	+	+	+
Створення повідомлень про наступну хвилю ворогів, щоб гравець міг готуватися до оборони	+	+	+	+	+	+
Створення таймера, що показує час до початку наступної хвилі ворогів	+	+	+	+	+	+
Створення повідомлень про помилки, які виникають під час гри, з поясненням можливих рішень	+	+	+	+	+	+
Створення системи статистики, що дозволяє гравцю переглядати інформацію про свої досягнення, знищених ворогів та зібрані ресурси	+	+	+	+	+	+
Створення можливості налаштування управління та інтерфейсу гри, щоб гравець міг змінити клавіші або інші параметри	+	+	+	+	+	+

Продовження таблиці 2.1

	Повнота	Однозначність	Коректність	Необхідність	Здійсненість	Перевірюваність
Створення кнопки "Пауза", яка дозволяє зупиняти гру у будь-який момент	+	+	+	+	+	+
Створення функції завершення гри, що активується у випадку перемоги або поразки, з відображенням результату гри	+	+	+	+	+	+
Створення можливості змінювати складність гри через меню налаштувань, щоб гравець міг налаштувати гру відповідно до своїх уподобань	+	+	+	+	+	+

Нефункціональні вимоги охоплюють такі категорії, як бізнес-правила, зовнішні інтерфейси, атрибути якості та обмеження. Бізнес-правила визначають стандарти та нормативні документи, яких повинна дотримуватися система. Зовнішні інтерфейси описують взаємодію гри з іншими системами та сервісами. Атрибути якості стосуються продуктивності, доступності та інших нефункціональних аспектів, які підвищують ефективність гри. Обмеження включають технічні вимоги до гри для забезпечення стабільної роботи та сумісності з різними платформами.

Нефункціональні вимоги до програмно забезпечення наведені в таблиці 2.2.

Таблиця 2.2 – Нефункціональні вимоги до програмного забезпечення

	Повнота	Однозначність	Коректність	Необхідність	Здійсненість	Перевірюваність
Бізнес-правила						
Адаптивність інтерфейсу гри повинна відповідати стандарту WCAG, щоб забезпечити доступність для людей з обмеженими можливостями	+	+	+	+	+	+
Всі процеси в грі повинні відповідати вимогам стандарту ISO/IEC 25010, щоб забезпечити надійність, продуктивність та цілісність програмного забезпечення	+	+	+	+	+	+
Всі помилки в грі повинні документуватися відповідно до стандарту ISO/IEC 20000, щоб забезпечити належне управління та підтримку	+	+	+	+	+	+
Зовнішні інтерфейси						
Підтримка API контролерів, таких як Xbox Controller API, для забезпечення зручного керування на консолях та ПК	+	+	+	+	+	+
Гра повинна інтегруватися з платформами цифрової дистрибуції (Steam, Epic Games Store) для розповсюдження гри та автоматичного оновлення	+	+	+	+	+	+
Атрибути якості						
Зручність використання	+	+	+	+	+	+
Адаптивність	+	+	+	+	+	+
Захищеність	+	+	+	+	+	+

Продовження таблиці 2.2

	Повнота	Однозначність	Коректність	Необхідність	Здійсненість	Перевірюваність
Конфіденційність	+	+	+	+	+	+
Надійність	+	+	+	+	+	+
Ремонтопридатність	+	+	+	+	+	+
Обмеження						
Гра повинна забезпечувати завантаження не довше 2 секунд для покращення досвіду користувача та зниження показника відмов	+	+	+	+	+	+
Максимальна кількість ворогів на карті не повинна перевищувати 100 одночасно, щоб забезпечити стабільну продуктивність гри	+	+	+	+	+	+
Максимальна кількість активних веж на рівні не повинна перевищувати 50, щоб уникнути перевантаження системи та забезпечити плавний ігровий процес	+	+	+	+	+	+
Ігрова частота кадрів повинна підтримуватися на рівні не менше 60 FPS, щоб забезпечити плавність анімації та геймплею	+	+	+	+	+	+
Розмір гри не повинен перевищувати 5 ГБ, щоб забезпечити швидке завантаження та установку на різних платформах	+	+	+	+	+	+
Підтримка роздільної здатності екрана повинна бути обмежена мінімумом 1280x720, щоб забезпечити належну якість графіки	+	+	+	+	+	+

Продовження таблиці 2.2

	Повнота	Однозначність	Коректність	Необхідність	Здійсненість	Перевірюваність
Прогрес гравця повинен автоматично зберігатися локально з регулярним інтервалом в 1 годину, щоб уникнути втрати даних під час збоїв.	+	+	+	+	+	+

Висновки за розділом 2:

1. Сформовано повний набір функціональних вимог до гри "Last Fort", які включають бізнес-вимоги, вимоги користувачів та функціональні вимоги, що повинна реалізувати система для забезпечення геймплею.

2. Визначено нефункціональні вимоги, які охоплюють бізнес-правила, зовнішні інтерфейси, атрибути якості та технічні обмеження гри.

3. Вимоги були сформовані з урахуванням повноти, однозначності, коректності, необхідності, здійсненності та перевірюваності, що забезпечує цілісність та ефективність проекту.

4. Вимоги сформовані таким чином, щоб забезпечити безпеку, доступність, стабільність та продуктивність гри на різних платформах.

3 ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання є важливим документом, що визначає основні вимоги, характеристики та умови розробки програмного забезпечення – 2D-гри в жанрі Tower Defense під назвою "Last Fort". Цей документ слугує основою для всього процесу розробки, оскільки він визначає ключові аспекти проєкту, що допомагають організувати роботу команди та забезпечити досягнення поставлених цілей.

Метою створення технічного завдання є формалізація та узгодження всіх вимог до майбутнього програмного продукту, що дозволяє розробникам чітко розуміти, які функції повинні бути реалізовані, як повинні виглядати і працювати ключові елементи гри, а також які технічні характеристики мають бути забезпечені. Цей документ також регламентує етапи розробки, вимоги до програмного середовища, системи тестування, вимоги до інтерфейсу, продуктивності та сумісності з різними платформами.

В таблиці 3.1 представлено технічне завдання до програмного забезпечення на «Last Fort»

Таблиця 3.1 – Технічне завдання на «Last Fort»

Розділ	Підрозділ	Опис
Вступ	Назва програми	Last Fort
	Предметна область	3D гра у жанрі Tower Defense
Вимоги до ПЗ	Вимоги до функціональних характеристик	<ul style="list-style-type: none"> – Як користувач, я хочу мати можливість налаштовувати управління та інтерфейс гри, щоб зробити їх зручнішими для мого стилю гри – Як користувач, я хочу мати можливість ставити гру на паузу у будь-який момент, щоб перегрупуватися або розробити нову стратегію

Продовження таблиці 3.1

Розділ	Підрозділ	Опис
		<ul style="list-style-type: none"> – Як користувач, я хочу мати можливість розміщувати вежі для захисту бази, щоб ефективно протистояти хвилям ворогів – Як користувач, я хочу мати можливість покращувати вежі, щоб підвищувати їхню ефективність у боротьбі з ворогами – Як користувач, я хочу пересуватися картою, щоб досліджувати нові території – Як користувач, я хочу отримувати інформаційні повідомлення про наступну хвилю ворогів, щоб краще підготуватися до оборони – Як користувач, я хочу бачити таймер до початку наступної хвилі, щоб встигнути розмістити вежі або виконати інші дії – Як користувач, я хочу мати можливість переглядати статистику своїх досягнень, щоб аналізувати свій прогрес та планувати стратегію – Як користувач, я хочу завершити гру у випадку перемоги або поразки, щоб знати результат своїх дій та планувати нову сесію – Як користувач, я хочу мати можливість змінювати складність гри, щоб налаштувати її відповідно до свого рівня навичок – Створення функції розміщення оборонних веж на карті за допомогою інтерфейсу користувача. – Створення системи покращення веж через меню взаємодії, з можливістю витратити ресурси на підвищення характеристик вежі.

Продовження таблиці 3.1

Розділ	Підрозділ	Опис
		<ul style="list-style-type: none"> – Створення кнопки для найму підданих з меню взаємодії, що дозволяє додати робітників для збору ресурсів і ремонту споруд. – Створення системи пересування персонажа по карті за допомогою елементів управління (клавіатура або миша). – Створення інформаційної панелі, яка відображає наявні ресурси, рівень здоров'я персонажа та кількість ворогів у поточній хвилі. – Створення повідомлень про наступну хвилю ворогів, щоб гравець міг підготуватися до оборони. – Створення таймера, що показує час до початку наступної хвилі ворогів. – Створення повідомлень про помилки, з поясненням можливих рішень для гравця. – Створення системи статистики, яка дозволяє переглядати інформацію про досягнення, знищених ворогів і зібрані ресурси. – Створення можливості налаштування управління та інтерфейсу гри для зміни клавіш або інших параметрів. – Створення кнопки "Пауза", яка дозволяє зупиняти гру у будь-який момент. – Створення функції завершення гри, що активується у випадку перемоги або поразки, з відображенням результату. – Створення можливості змінювати складність гри через меню налаштувань.

Продовження таблиці 3.1

Розділ	Підрозділ	Опис
	Вимоги до нефункціональних характеристик	<ul style="list-style-type: none"> – Адаптивність інтерфейсу гри повинна відповідати стандарту WCAG для забезпечення доступності для людей з обмеженими можливостями. – Всі процеси в грі повинні відповідати стандарту ISO/IEC 25010, щоб гарантувати надійність, продуктивність та цілісність програмного забезпечення. – Всі помилки в грі повинні документуватися відповідно до стандарту ISO/IEC 20000 для забезпечення належного управління та підтримки. – Гра повинна завантажуватися не довше 2 секунд для покращення досвіду користувача та зниження показника відмов. – Максимальна кількість ворогів на карті не повинна перевищувати 100 одночасно, щоб забезпечити стабільну продуктивність гри. – Максимальна кількість активних веж на рівні не повинна перевищувати 50, щоб уникнути перевантаження системи та забезпечити плавний ігровий процес. – Ігрова частота кадрів повинна підтримуватися на рівні не менше 60 FPS для плавної анімації та геймплею. – Розмір гри не повинен перевищувати 5 ГБ, щоб забезпечити швидке завантаження та установку на різних платформах. – Мінімальна підтримувана роздільна здатність екрана повинна бути 1280x720 для забезпечення належної якості графіки.

Продовження таблиці 3.1

Розділ	Підрозділ	Опис
		– Прогрес гравця повинен автоматично зберігатися локально з регулярним інтервалом в 1 годину, щоб уникнути втрати даних під час збоїв.
	Вимоги до складу ПЗ та параметрам технічних засобів	– Unity як основний інструмент для розробки, підтримка для платформ ПК. Мінімальні системні вимоги: Windows 7/8/10, процесор Intel Core i5, 8 ГБ оперативної пам'яті, відеокарта з підтримкою DirectX 11.
Стадії та етапи розробки		<p>01.09.2023 – 20.12.2024</p> <p>1) Моделювання предметної області 01.09.2024 – 10.09.2024</p> <p>2) Побудова IDEF0-діаграм 11.09.2024 – 20.09.2024</p> <p>3) Побудова профілю вимог до ПЗ 21.09.2024 – 01.10.2024</p> <p>4) Технічне завдання 02.10.2024 – 15.10.2024</p> <p>5) Створення додатку 16.10.2024 – 20.11.2024</p> <p>6) Виконання тестування 21.11.2024 – 10.12.2024</p> <p>7) Підготовка документації 11.12.2024 – 15.12.2024</p>
Умови експлуатації	Види обслуговування	Регулярні оновлення для виправлення помилок, додавання нового контенту та покращення продуктивності
	Необхідна кількість персоналу	Для розробки гри потрібна команда з 2 – 3 осіб, включно з програмістом і тестувальником.

Продовження таблиці 3.1

Розділ	Підрозділ	Опис
	Вимоги до вихідного коду та мов програмування	Основна мова розробки – C# з використанням Unity. Код повинен бути добре структурованим, документованим та відповідати стандартам програмування.
	Вимоги до захисту інформації та додатків	Прогрес гри зберігається локально на пристрої користувача.
Техніко-економічні показники	Економічна ефективність	середня
	Стратегічна цінність	Середня
Порядок контролю та приймання	Види тестування програмного забезпечення	Smoke testing Регресійне тестування Альфа-тестування Тестування збірки Тестування взаємодії Бета тестування
	Загальні вимоги до прийняття ПЗ	Програмний продукт повинен забезпечувати функціональну повноту не нижче 90%, відсутність багів серйозності S1-S3 та пріоритетності P1-P2, стабільну продуктивність (60 FPS, завантаження до 2 секунд), успішне проходження всіх тестових сценаріїв, сумісність із підтримуваними платформами, відповідність стандартам безпеки (GDPR, ISO 27001) та наявність усієї супровідної документації.

Висновки за розділом 3:

1. Сформовано чітке технічне завдання для розробки гри «Last Fort», що визначає ключові функціональні та нефункціональні вимоги до програмного забезпечення.

2. Деталізовано вимоги до складу програмного забезпечення, технічних засобів, необхідної кількості персоналу, вимоги до вихідного коду та мов програмування.

3. Встановлено етапи розробки гри та їхні часові рамки, що сприяє ефективному плануванню і реалізації проєкту.

4. Окреслено порядок контролю та приймання програмного продукту, зокрема види тестування та загальні вимоги до прийняття ПЗ.

4 ПЛАНУВАННЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Планування тестування є одним із найважливіших етапів у процесі розробки програмного забезпечення, оскільки забезпечує виявлення можливих дефектів та перевірку відповідності розробленого продукту заданим вимогам. Для гри «Last Fort» було розроблено детальний набір тестових випадків, які покривають основні аспекти функціональності та нефункціональних характеристик гри.

Тестування передбачає перевірку таких компонентів, як управління персонажем, взаємодія з оборонними вежами, генерація ворогів, обробка ресурсів та оновлення інтерфейсу користувача. Це дозволяє переконатися у коректності реалізації ключових функцій гри, стабільності її роботи.

Набір тестових кейсів для додатку «Last Fort» представлений таблицями 4.1 – 4.6.

Таблиця 4.1 – Тестовий кейс для перевірки переміщення персонажа та обертів камери на карті

Назва	Перевірка переміщення персонажа та обертів камери на карті	
Функція:	Переміщення персонажа по карті та оберт камери	
Дія	Очікуємий результат	Результат тесту: пройдено провалено заблоковано
Передумова:		
Запустити гру "Last Fort"	Персонаж та камера активні	пройдено

Продовження таблиці 4.1

Дія	Очікуємий результат	Результат тесту: пройдено провалено заблоковано
Кроки тесту:		
Натиснути кнопку "W"	Персонаж рухається вперед	пройдено
Натиснути кнопку "S"	Персонаж рухається назад	пройдено
Натиснути кнопку "A"	Персонаж рухається вліво	пройдено
Натиснути кнопку "D"	Персонаж рухається вправо	пройдено
Перемістити мишу вліво, утримуючи праву кнопку миші	Камера обертається вліво	пройдено
Перемістити мишу вправо, утримуючи праву кнопку миші	Камера обертається вправо	пройдено
Перемістити мишу вгору, утримуючи праву кнопку миші	Камера обертається вгору	пройдено
Перемістити мишу вниз, утримуючи праву кнопку миші	Камера обертається вниз	пройдено

Таблиця 4.2 – Тестовий кейс для перевірки можливості побудови веж на карті.

Назва	Перевірка можливості побудови веж на карті
Функція:	Побудова веж у дозволених зонах карти

Продовження таблиці 4.2

Дія	Очікуємий результат	Результат тесту: пройдено провалено заблоковано
Передумова:		
Запустити гру "Last Fort"	Гра запущена та активна	пройдено
Кроки тесту:		
Вибрати тип вежі в нижній частині екрану	Тип вежі обрано	пройдено
Клацнути на місці розташування у дозволений зоні	Вежа будується на обраній позиції	пройдено
Постумова:		
Перевірити наявність вежі та зменшення ресурсів	Вежа розташована на карті, ресурси зменшились	пройдено

Таблиця 4.3 – Тестовий кейс для перевірки коректної генерації хвиль ворогів на карті та їх переміщення.

Назва	Перевірка коректної генерації хвиль ворогів на карті та їх переміщення
Функція:	Генерація хвиль ворогів на карті та їх переміщення

Продовження таблиці 4.3

Дія	Очікуємий результат	Результат тесту: пройдено провалено заблоковано
Передумова:		
Запустити гру "Last Fort"	Гра запущена та активна	пройдено
Кроки тесту:		
Дочекатися запуску хвилі ворогів	Вороги з'являються на стартовій точці	пройдено
Спостерігати за рухом ворогів	Вороги рухаються по заданій траєкторії до бази	пройдено
Постумова:		
Порахувати кількість ворогів	Кількість ворогів відповідає параметрам хвилі	пройдено

Таблиця 4.4 – Тестовий кейс для перевірки механіки атаки вежами ворогів.

Назва	Перевірка механіки атаки вежами ворогів
Функція:	Автоматична атака ворогів при вході у радіус вежі

Продовження таблиці 4.4

Дія	Очікуємий результат	Результат тесту: пройдено провалено заблоковано
Передумова:		
Запустити гру "Last Fort"	Гра запущена та активна	пройдено
Розмістити вежу	Вежа активна та готова до атаки	пройдено
Кроки тесту:		
Дочекатися запуску хвилі ворогів	Вороги з'являються на стартовій точці	пройдено
Дочекатись входу ворога у радіус вежі	Вежа автоматично атакує ворога	пройдено
Постумова:		
Перевірити здоров'я ворога	Здоров'я ворога зменшується	пройдено

Таблиця 4.5 – Тестовий кейс для перевірки коректного оновлення інформаційної панелі.

Назва	Перевірка коректного оновлення інформаційної панелі на інтерфейсі користувача
Функція:	Відображення ресурсів, хвилі ворогів і стану бази

Продовження таблиці 4.5

Дія	Очікуємий результат	Результат тесту: пройдено провалено заблоковано
Передумова:		
Запустити гру "Last Fort"	Гра запущена та активна	пройдено
Кроки тесту:		
Дочекатися запуску хвилі ворогів	Номер хвилі оновлюється	пройдено
Дочекатися втрати здоров'я бази	Стан здоров'я бази змінюється на панелі	провалено
Побудувати вежу	Кількість ресурсів на панелі зменшується	пройдено
Постумова:		
Перевірити стан гри	Гра продовжує працювати стабільно, помилки не виникають	пройдено

Таблиця 4.6 – Тестовий кейс для перевірки коректного оновлення інформаційної панелі.

Назва	Перевірка спроби покращення вежі після досягнення максимального рівня
Функція:	Відображення ресурсів, хвилі ворогів і стану бази

Продовження таблиці 4.6

Дія	Очікуємий результат	Результат тесту: пройдено провалено заблоковано
Передумова:		
Запустити гру "Last Fort"	Гра запущена та активна	пройдено
Кроки тесту:		
Побудувати вежу	Вежа побудована	
Провести всі можливі покращення вежі	Вежа досягла максимального рівня	
Натиснути лівою кнопкою миші на вежу максимального рівня	З'являється меню взаємодії з вежею.	пройдено
Спробувати натиснути кнопку покращення вежі	Кнопка неактивна та не працює.	провалено

Висновки за розділом 4:

1. Визначено основні функції гри для тестування: переміщення персонажа, розміщення та покращення веж, генерація хвиль ворогів.


2. Розроблено тестові випадки для перевірки ключових сценаріїв використання та можливих крайніх ситуацій.

3. Проведене тестування дозволило оцінити відповідність функціональності гри "Last Fort" заданим вимогам.

5 ВИКОНАННЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є ключовим етапом забезпечення його якості, адже дозволяє виявити недоліки в роботі системи, які можуть впливати на користувацький досвід. Під час тестування гри «Last Fort» було створено баг-репорти для тих тестових кейсів, які виявилися провальними. Баг-репорти містять інформацію про проблеми, їхню серйозність, пріоритетність, а також кроки відтворення і очікувані результати. В таблицях 5.1 та 5.2 представлені баг-репорти до гри «Last Fort».

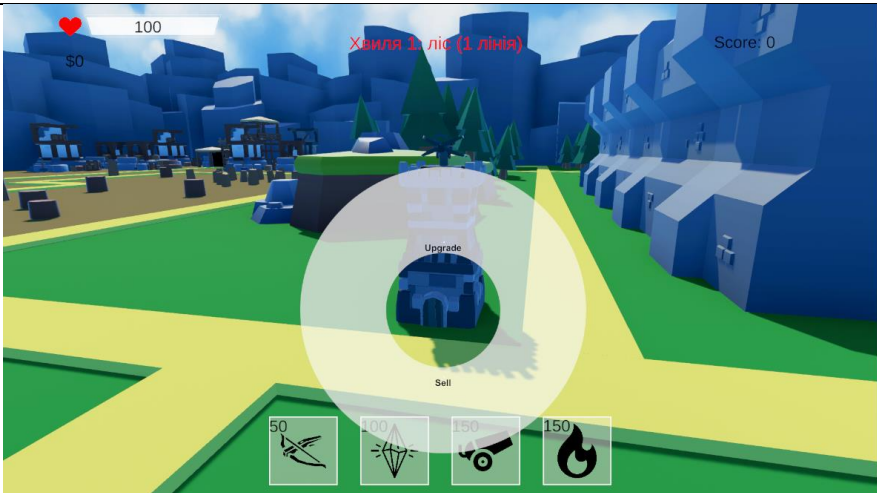
Таблиця 5.1 – Баг-репорт до п'ятого тестового кейсу

Summary	Невірне оновлення стану здоров'я бази на інформаційній панелі
Проект	Гра "Last Fort"
Компонент додатку	Інтерфейс інформаційної панелі
Номер версії	1.0
Оточення	Windows 10, Unity 2022.1.5f1
Серйозність:	S3
Пріоритет:	P3
Статус	Закритий
Автор	Лучанінов Петро
Призначений на	Розробник
Кроки відтворення	<ol style="list-style-type: none"> 1. Запустити гру "Last Fort". 2. Створити умови, за яких база втрачає здоров'я (наприклад, дозволити ворогам дійти до бази). 3. Спостерігати за значенням здоров'я бази на інформаційній панелі.
Фактичний результат	 <p>Значення здоров'я бази на інформаційній панелі не оновлюється, незважаючи на втрату здоров'я бази.</p>

Продовження таблиці 5.1

Очікуємий результат	Значення здоров'я бази повинно коректно відображатися на інформаційній панелі та зменшуватися відповідно до шкоди, завданої базі.
---------------------	---

Таблиця 5.2 – Баг-репорт до шостого тестового кейсу

Summary	Кнопка апгрейду вежі активна в меню взаємодії з вежею після досягнення максимального рівня
Проект	Гра "Last Fort"
Компонент додатку	Система апгрейду
Номер версії	1.0
Оточення	Windows 10, Unity 2022.1.5f1
Серйозність:	S3
Пріоритет:	P3
Статус	Закритий
Автор	Лучанінов Петро
Призначений на	Розробник
Кроки відтворення	<ol style="list-style-type: none"> 1. Запустити гру "Last Fort". 2. Побудувати вежу на карті. 3. Провести всі можливі апгрейди вежі до максимального рівня. 4. Натиснути кнопку апгрейду в меню взаємодії з вежею.
Фактичний результат	 <p>Кнопка апгрейду активна, проте натискання на неї не викликає жодних дій або змін.</p>
Очікуємий результат	Кнопка апгрейду повинна бути неактивною (затемненою) після досягнення вежі максимального рівня.

Висновки за розділом 5:

1. Проведено аналіз результатів тестування програмного забезпечення гри «Last Fort».
2. Створено баг-репорти з детальним описом знайдених помилок, включаючи їх типи, пріоритетність виправлення та запропоновані рішення.
3. Результати тестування підтвердили відповідність функціональних та нефункціональних характеристик гри заявленим вимогам, а також забезпечення очікуваного рівня якості.

6 ВИКОНАННЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1 Засоби розробки програмного забезпечення

Для розробки програмного забезпечення гри «Last Fort» було використано сучасні інструменти та підходи, які забезпечують високу якість продукту та ефективність роботи. У процесі розробки було враховано всі аспекти створення ігрових механік, інтерфейсу користувача, а також оптимізації продуктивності.

Програма розроблена мовою програмування C# [5], яка є основним стандартом для роботи в середовищі Unity [6]. Ця мова дозволяє реалізувати складну ігрову логіку, включаючи об'єктно-орієнтований підхід, інтерфейси, та модульність структури коду.

Основним середовищем розробки став Unity Engine (версія 2022.3.1f1), що дозволяє створювати 2D і 3D ігри з використанням потужного набору інструментів. Unity надає готові рішення для обробки фізики, роботи з графікою, анімацією та звуком.

У процесі розробки використовувалися наступні інструменти та бібліотеки:

- TextMeshPro – для роботи з текстом, створення динамічних підписів та інформаційних панелей.
- AI Navigation – для управління рухом ворогів по заздалегідь визначених маршрутах.
- PiUI [7] – для побудови зручного та стильного користувацького інтерфейсу.
- Beautify [8] – для покращення візуальних ефектів, таких як освітлення, тіні та постпроцесинг.

Програма вимагає операційну систему Windows 10 або Windows 11. Мінімальні системні вимоги включають 4 ГБ оперативної пам'яті та процесор із частотою 2 ГГц.

Основою архітектури програми є об'єктно-орієнтоване програмування, що дозволило структурувати компоненти системи у вигляді взаємодіючих класів.

Усі ключові механіки гри, такі як управління вежами, рух ворогів та обробка подій, реалізовані через використання класів. Для забезпечення універсальності та спрощення коду використовуються інтерфейси, які задають стандартизований підхід до реалізації функцій. Наприклад, інтерфейс `IDamageMethod` визначає загальні методи для нанесення шкоди, що дозволяє створювати різні типи атак (лазери, ракети, стандартні постріли) із єдиним підходом до їх реалізації.

Важливим аспектом реалізації стала побудова моделей даних, які відповідають за збереження та обробку інформації. Зокрема, клас `EnemySummonData` використовується для зберігання параметрів ворогів, таких як їхня швидкість, кількість очок здоров'я та типи атак. Це рішення дозволяє легко налаштовувати різноманітні хвилі ворогів та створювати нові типи супротивників без внесення змін у базову логіку гри.

Особлива увага була приділена оптимізації роботи програми. Для цього було використано патерн проектування `Object Pooling` [9], який дозволяє повторно використовувати об'єкти ворогів у пам'яті, замість створення та знищення їх під час кожної нової хвилі. Це значно знижує навантаження на систему, забезпечуючи стабільну продуктивність навіть за великої кількості активних об'єктів на екрані.

6.2 Інструкція користувача

Програмне забезпечення "Last Fort" пропонує користувачам зручний інтерфейс та інтуїтивно зрозумілий процес гри. У цьому розділі описано порядок роботи з програмою, включаючи елементи керування та основні функціональні можливості.

Після запуску гри користувач потрапляє до початкового меню, яке містить три кнопки:

- Start Game – почати гру.
- Settings – перейти до налаштувань гри.
- Exit – завершити роботу програми.

На рисунку 6.1 представлено вигляд початкового меню.



Рисунок 6.1 – Зображення початкового меню

Натиснувши на кнопку Settings, відкривається меню налаштувань, де можна змінити:

- Розмір екрану.
- Увімкнення/вимкнення повноекранного режиму.
- Рівень графіки.
- Гучність звуку.

На рисунку 6.2 представлено меню налаштувань гри.

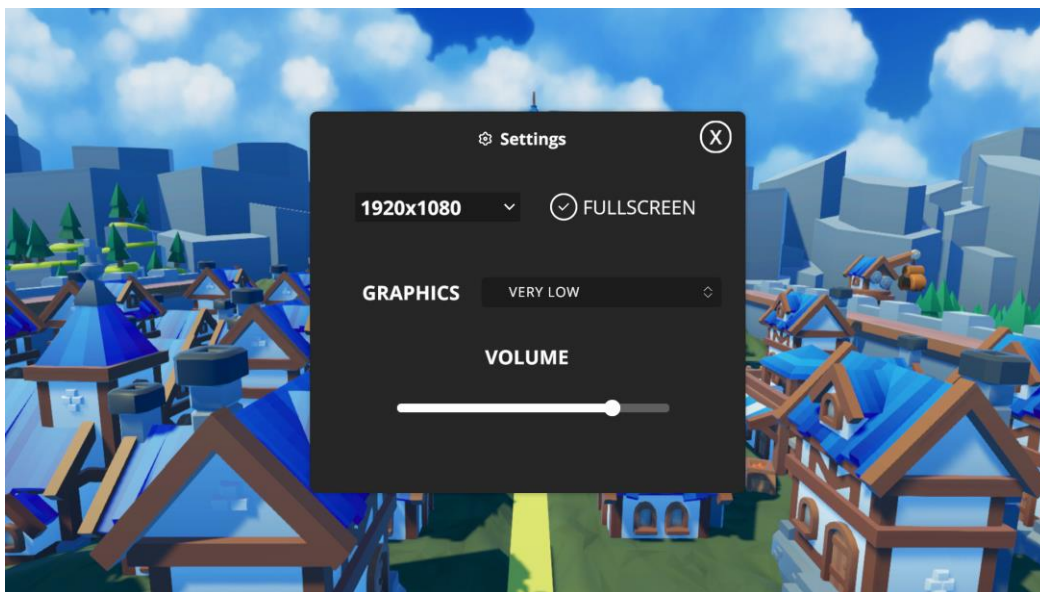


Рисунок 6.2 – Зображення меню налаштувань гри

Після натискання кнопки Start Game завантажується ігровий процес. На екрані відображається інтерфейс користувача, який містить:

- Ресурси, хвили та здоров'я бази (розташовані у верхній частині екрана).
- Панель вибору веж (розташована внизу екрана). При натисканні на вежу користувач може обрати її для розміщення на полі.

На рисунку 6.3 представлено інтерфейс користувача під час гри.



Рисунок 6.3 – Зображення інтерфейсу користувача під час гри

Для розміщення вежі користувач повинен:

1. Вибрати вежу з нижньої панелі.
2. Натиснути на ділянку землі, щоб розташувати вежу.

Вежі можна:

- Покращувати.
- Продавати.
- Змінювати цілі для атаки ворогів (перші, останні, найближчі до вежі, найсильніші чи найслабші).

Для цього необхідно натиснути на вежу, після чого відкриється меню взаємодії з вежами. На рисунку 6.4 представлено приклад взаємодії з вежами.

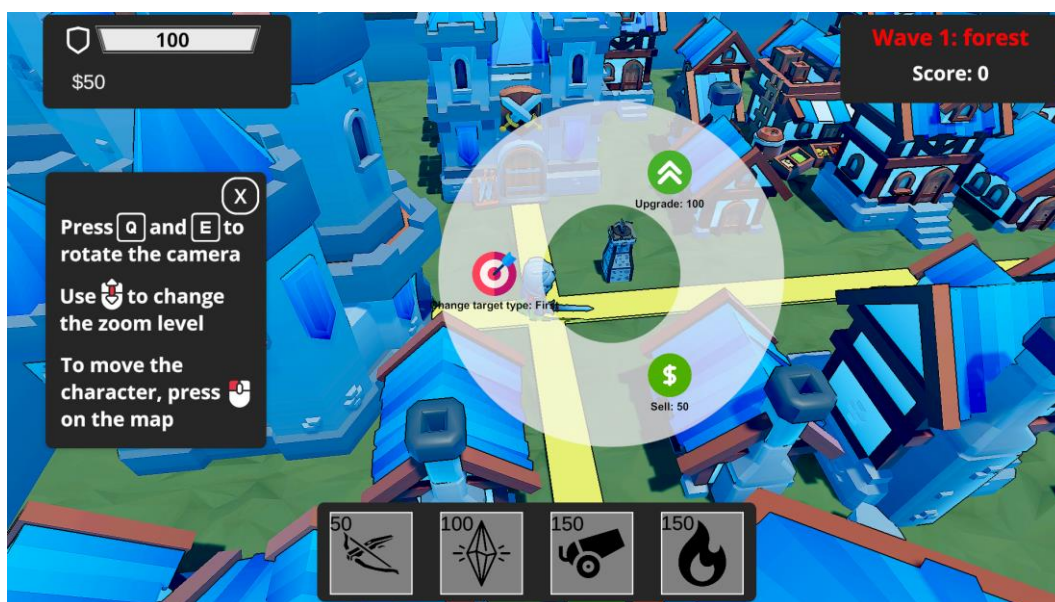


Рисунок 6.4 – Зображення інтерфейсу користувача під час гри

Для переміщення персонажа користувач повинен клацнути на ту частину карти, куди персонаж має перейти. Це забезпечує легкість у навігації та дозволяє гравцю оперативно переміщуватися між різними зонами для розміщення веж або збору ресурсів.

Натискання клавіші Escаре відкриває пауза-меню, яке містить:

- Continue – продовжити гру.
- Settings – відкрити меню налаштувань.
- Finish the game – завершити гру.

На рисунку 6.5 представлено вигляд пауза-меню.

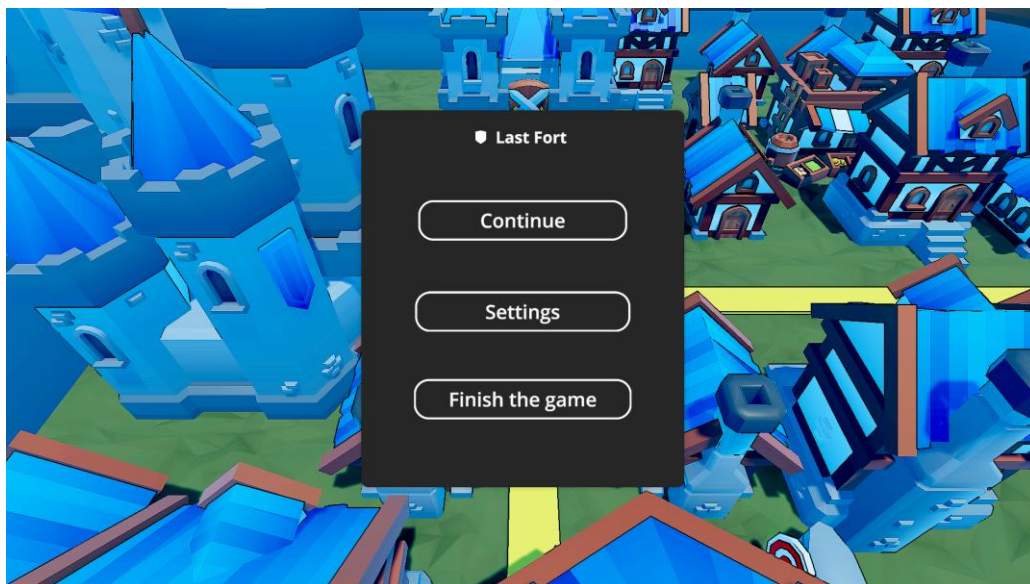


Рисунок 6.5 – Зображення інтерфейсу користувача під час гри

Після завершення гри відображається екран результатів, де користувач може:

- Вибрати Retry, щоб почати гру спочатку.
- Натиснути Main Menu, щоб повернутися до початкового меню.

На рисунку 6.6 представлено екран результатів гри.

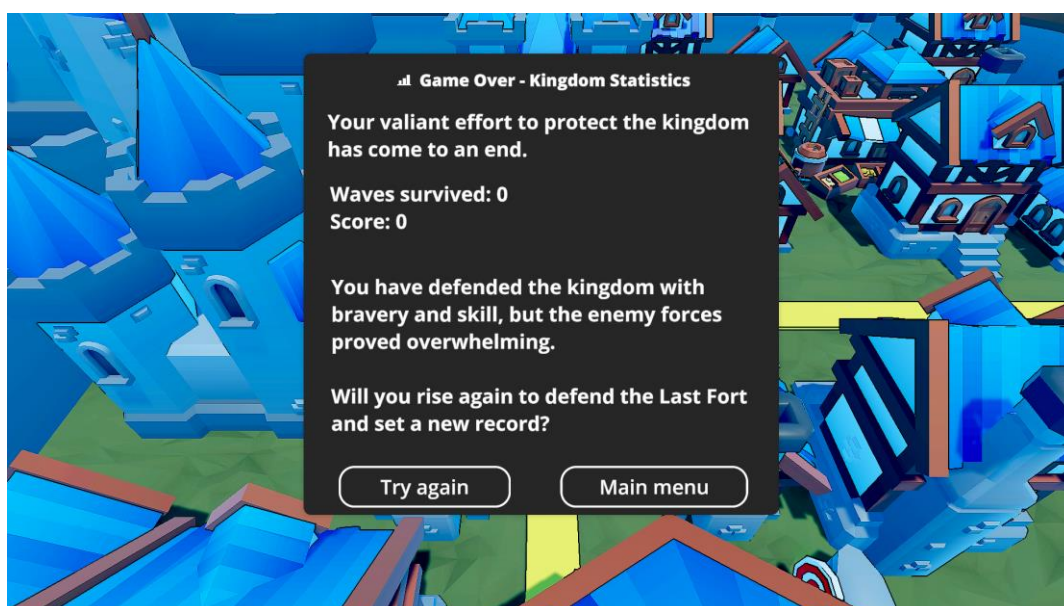


Рисунок 6.6 – Зображення інтерфейсу користувача під час гри

Висновки за розділом 6:

1. Надано комплексний опис технічної реалізації гри, що сприяє кращому розумінню її структури та можливостей.
2. Визначено мінімальні вимоги до обладнання та операційної системи для стабільної роботи програми.
3. Розроблено покрокову інструкцію використання гри з демонстрацією взаємодії користувача через скріншоти.

ВИСНОВКИ

У процесі виконання курсового проекту була проведена всебічна робота з розробки гри "Last Fort" у жанрі Tower Defense. Ця робота охоплювала повний цикл створення програмного забезпечення, включаючи аналіз предметної області, визначення вимог, проектування, реалізацію, тестування та документування програми.

Проведений аналіз предметної області дозволив визначити ключові характеристики жанру Tower Defense, такі як стратегічний ігровий процес, використання оборонних споруд із різними властивостями, хвилі ворогів і система ресурсів. Це стало основою для формування вимог до гри, які чітко визначали функціональні й нефункціональні характеристики продукту. На основі цих вимог було створено технічне завдання, що описувало всі аспекти реалізації.

Реалізований ігровий процес включає побудову оборонних веж, їх покращення, управління ресурсами та захист бази від ворогів. Інтерфейс гри інтуїтивно зрозумілий, що полегшує взаємодію користувача з програмою. Для кращого розуміння роботи гри було створено докладну інструкцію користувача з покроковими поясненнями та ілюстраціями.

Під час тестування програма була ретельно перевірена на стабільність і відповідність заданим вимогам. Виявлені помилки були оперативно виправлені, що забезпечило високу якість продукту та задовільний користувацький досвід. Підготовлено баг-репорти, які документують процес виправлення помилок і результати тестування.

Розроблений продукт має значний потенціал для подальшого вдосконалення, включаючи розширення ігрових механік, додавання нових функцій та адаптацію до потреб ширшої аудиторії. Проект демонструє високий рівень реалізації, що дозволяє розглядати його як комерційно успішний продукт.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

- [1] Dota, [online]. Режим доступу: <https://www.dota2.com/>
- [2] Kingdom Rush, [online]. Режим доступу: <https://www.kingdomrush.com/kingdom-rush>
- [3] Dungeon Defenders, [online]. Режим доступу: https://store.steampowered.com/app/65800/Dungeon_Defenders/
- [4] Глобальний звіт про ринок мобільних ігор жанру Tower Defense, [online]. Режим доступу: <https://www.astuteanalytica.com/industry-report/mobile-tower-defense-games-market>
- [5] Документація до мови програмування C#, [online]. Режим доступу: <https://learn.microsoft.com/en-us/dotnet/csharp/>
- [6] Документація до ігрового движку Unity, [online]. Режим доступу: <https://docs.unity.com/>
- [7] Документація ассету PiUI, [online]. Режим доступу: <https://assetstore.unity.com/packages/tools/gui/pi-ui-94296>
- [8] Документація ассету Beautify, [online]. Режим доступу: <https://www.dropbox.com/scl/fi/av963557ozww66eoe6ral/Documentation.pdf?rlkey=z43lb5ongx0f6z kf330b4rtm1&e=6&dl=0>
- [9] Патерн проектування Object pool, [online]. Режим доступу: https://en.wikipedia.org/wiki/Object_pool_pattern

ДОДАТОК А

Перша частина лістингу коду файлу GameLoopManager.cs

```

using System.Collections;
using System.Collections.Generic;
using Unity.Collections;
using UnityEngine.Jobs;
using UnityEngine;
using Unity.Jobs;
using static UnityEngine.GraphicsBuffer;
using TMPro;
using UnityStandardAssets.Vehicles.Car;
using System.Linq;

[System.Serializable]
public class Wave
{
    public Dictionary<int, List<int>> enemiesPerLane;
    public float spawnInterval;
}

public class GameLoopManager : MonoBehaviour
{
    public static List<TowerBehavior> TowersInGame;
    public static Vector3[] NodePositions1;
    public static Vector3[] NodePositions2;
    public static Vector3[] NodePositions3;
    public static float[] NodeDistances1;
    public static float[] NodeDistances2;
    public static float[] NodeDistances3;

    private static Queue<ApplyEffectData> EffectsQueue;
    private static Queue<EnemyDamageData> DamageData;
    private static Queue<Enemy> EnemiesToRemove;

```

```

public static Queue<int> EnemyIDsToSummon;
public static Queue<int> EnemyNodeIndexesToSummon;

private PlayerStats PlayerStatistics;
public int currentWave = 0;
private bool isWaveActive = false;
private float waveTimer = 30f;
private float waveInterval = 0f;

public Transform NodeParent1;
public Transform NodeParent2;
public Transform NodeParent3;
public bool LoopShouldEnd = true;
[SerializeField] private TextMeshProUGUI waveInfoText;
[SerializeField] private TextMeshProUGUI waveTimerText;
[SerializeField] private GameObject lane1Indicator;
[SerializeField] private GameObject lane2Indicator;
[SerializeField] private GameObject lane3Indicator;

void Start()
{
    PlayerStatistics = FindAnyObjectByType<PlayerStats>();
    EffectsQueue = new Queue<ApplyEffectData>();
    DamageData = new Queue<EnemyDamageData>();
    TowersInGame = new List<TowerBehavior>();
    EnemyIDsToSummon = new Queue<int>();
    EnemyNodeIndexesToSummon = new Queue<int>();
    EnemiesToRemove = new Queue<Enemy>();
    EntitySummoner.Init();

    NodePositions1 = new Vector3[NodeParent1.childCount];
    NodePositions2 = new Vector3[NodeParent2.childCount];
    NodePositions3 = new Vector3[NodeParent3.childCount];

```

```

for (int i = 0; i < NodePositions1.Length; i++)
{
    NodePositions1[i] = NodeParent1.GetChild(i).position;
}

for (int i = 0; i < NodePositions2.Length; i++)
{
    NodePositions2[i] = NodeParent2.GetChild(i).position;
}

for (int i = 0; i < NodePositions3.Length; i++)
{
    NodePositions3[i] = NodeParent3.GetChild(i).position;
}

NodeDistances1 = new float[NodePositions1.Length - 1];
NodeDistances2 = new float[NodePositions2.Length - 1];
NodeDistances3 = new float[NodePositions3.Length - 1];

for (int i = 0; i < NodeDistances1.Length; i++)
{
    NodeDistances1[i] = Vector3.Distance(NodePositions1[i], NodePositions1[i+1]);
}

for (int i = 0; i < NodeDistances2.Length; i++)
{
    NodeDistances2[i] = Vector3.Distance(NodePositions2[i], NodePositions2[i+1]);
}

for (int i = 0; i < NodeDistances2.Length; i++)
{
    NodeDistances2[i] = Vector3.Distance(NodePositions2[i], NodePositions2[i+1]);
}

}

```

```
public void ContinueGameLoop()
```

```
{
```

```
    LoopShouldEnd = false;
```

```
    StartCoroutine(GameLoop());
```

```
}
```

```
public void StopGameLoop()
```

```
{
```

```
    LoopShouldEnd = true;
```

```
    StopCoroutine(GameLoop());
```

```
}
```

```
Wave GenerateDynamicWave()
```

```
{
```

```
    float spawnInterval = currentWave > 50 ? 0.5f : Mathf.Max(0.5f, 4f - currentWave * 0.5f);
```

```
    List<int> laneNumbers = SelectLaneNumbers();
```

```
    Dictionary<int, List<int>> enemiesPerLane = new Dictionary<int, List<int>>();
```

```
    foreach (int lane in laneNumbers)
```

```
    {
```

```
        enemiesPerLane[lane] = GenerateEnemiesForLane();
```

```
    }
```

```
    return new Wave
```

```
    {
```

```
        enemiesPerLane = enemiesPerLane,
```

```
        spawnInterval = spawnInterval
```

```
    };
```

```
}
```

```
List<int> SelectLaneNumbers()
```

```
{
```

```
    if (currentWave <= 15) return new List<int> { 1 };
```

```
    else if (currentWave <= 30) return new List<int> { 1, 2 };
```



```

else if (currentWave <= 50) return new List<int> { 1, 2, 3 };
else return new List<int> { 1, 2, 3 };
}

List<int> GenerateEnemiesForLane()
{
    List<int> enemies = new List<int>();

    if (currentWave <= 15)
    {
        int numType2 = Mathf.FloorToInt((currentWave / 15f) * 10);
        int numType1 = 10 - numType2;

        enemies.AddRange(Enumerable.Repeat(1, numType1));
        enemies.AddRange(Enumerable.Repeat(2, numType2));
    }
    else if (currentWave <= 30)
    {
        int numType3 = Mathf.FloorToInt((currentWave - 15) / 15f * 10);
        int numType2 = 10 - numType3;

        enemies.AddRange(Enumerable.Repeat(2, numType2));
        enemies.AddRange(Enumerable.Repeat(3, numType3));
    }
    else if (currentWave <= 50)
    {
        int numType4 = Mathf.FloorToInt((currentWave - 30) / 20f * 10);
        int numType3 = 10 - numType4;

        enemies.AddRange(Enumerable.Repeat(3, numType3));
        enemies.AddRange(Enumerable.Repeat(4, numType4));
    }
    else
    {
        enemies.AddRange(Enumerable.Repeat(Random.Range(2, 5), 10));
    }
}

```

```

    }

    return enemies;
}

IEnumerator SpawnWave(Wave wave)
{
    isWaveActive = true;

    foreach (var lane in wave.enemiesPerLane)
    {
        int laneNumber = lane.Key;
        List<int> enemies = lane.Value;

        foreach (int enemyID in enemies)
        {
            EnqueueEnemyIDToSummon(enemyID, laneNumber);
            yield return new WaitForSeconds(wave.spawnInterval);
        }
    }

    isWaveActive = false;
    DeactivateLaneIndicators();
}

void UpdateWaveUI(Wave wave)
{
    string location = wave.enemiesPerLane.ContainsKey(1) ? "forest" :
        wave.enemiesPerLane.ContainsKey(2) ? "mountains" :
        wave.enemiesPerLane.ContainsKey(3) ? "scorched valley" : "all ways";

    waveInfoText.text = $"Wave {currentWave + 1}: {location}";
    ActivateLaneIndicators(wave.enemiesPerLane.Keys.ToList());
}

void ActivateLaneIndicators(List<int> lanes)

```

```

{
    lane1Indicator.SetActive(lanes.Contains(1));
    lane2Indicator.SetActive(lanes.Contains(2));
    lane3Indicator.SetActive(lanes.Contains(3));
}

void DeactivateLaneIndicators()
{
    lane1Indicator.SetActive(false);
    lane2Indicator.SetActive(false);
    lane3Indicator.SetActive(false);
}

IEnumerator GameLoop()
{
    while (LoopShouldEnd == false)
    {
        //Wave Creation
        if (!isWaveActive)
        {
            waveInfoText.gameObject.SetActive(false);
            waveTimerText.gameObject.SetActive(true);
            waveTimer += Time.deltaTime;
            waveTimerText.text = $"Next wave in: {Mathf.CeilToInt(waveInterval - waveTimer)}
sec.";

            if (waveTimer >= waveInterval)
            {
                waveTimerText.gameObject.SetActive(false);
                waveInfoText.gameObject.SetActive(true);
                Wave wave = GenerateDynamicWave();
                UpdateWaveUI(wave);
                StartCoroutine(SpawnWave(wave));
                currentWave++;
                waveTimer = 0f;
            }
        }
    }
}

```

```

    }
}

//Spawn Enemies

if (EnemyIDsToSummon.Count > 0)
{
    for (int i = 0; i < EnemyIDsToSummon.Count; ++i)
    {
        EntitySummoner.SummonEnemy(EnemyIDsToSummon.Dequeue(),
EnemyNodeIndexesToSummon.Dequeue());
    }
}

//Move Enemies

NativeArray<Vector3> NodesToUse1 = new NativeArray<Vector3>(NodePositions1,
Allocator.TempJob);
NativeArray<Vector3> NodesToUse2 = new NativeArray<Vector3>(NodePositions2,
Allocator.TempJob);
NativeArray<Vector3> NodesToUse3 = new NativeArray<Vector3>(NodePositions3,
Allocator.TempJob);
NativeArray<float> EnemySpeeds = new
NativeArray<float>(EntitySummoner.EnemiesInGame.Count, Allocator.TempJob);
NativeArray<int> NodeIndices = new
NativeArray<int>(EntitySummoner.EnemiesInGame.Count, Allocator.TempJob);
NativeArray<int> NodesIndexes = new
NativeArray<int>(EntitySummoner.EnemiesInGame.Count, Allocator.TempJob);
TransformAccessArray EnemyAcces = new
TransformAccessArray(EntitySummoner.EnemiesInGameTransform.ToArray(), 2);

for(int i = 0; i < EntitySummoner.EnemiesInGame.Count; i++)
{
    EnemySpeeds[i] = EntitySummoner.EnemiesInGame[i].Speed;
    NodeIndices[i] = EntitySummoner.EnemiesInGame[i].NodeIndex;
}

```

```

        NodesIndexes[i] = EntitySummoner.EnemiesInGame[i].NodesIndex;
    }

    MoveEnemiesJob MoveJob = new MoveEnemiesJob
    {
        NodePosition1 = NodesToUse1,
        NodePosition2 = NodesToUse2,
        NodePosition3 = NodesToUse3,
        EnemySpeed = EnemySpeeds,
        NodeIndex = NodeIndices,
        NodesIndex = NodesIndexes,
        deltaTime = Time.deltaTime
    };

    JobHandle MoveJobHandle = MoveJob.Schedule(EnemyAcces);
    MoveJobHandle.Complete();

    for(int i = 0; i < EntitySummoner.EnemiesInGame.Count; i++)
    {
        EntitySummoner.EnemiesInGame[i].NodeIndex = NodeIndices[i];

        switch (EntitySummoner.EnemiesInGame[i].NodesIndex)
        {
            case 1:
                if(EntitySummoner.EnemiesInGame[i].NodeIndex == NodePositions1.Length)
                {
                    PlayerStatistics.AddHealth(-
EntitySummoner.EnemiesInGame[i].GetComponent<Enemy>().Damage);
                    EnqueueEnemyToRemove(EntitySummoner.EnemiesInGame[i]);
                }
                break;

            case 2:
                if(EntitySummoner.EnemiesInGame[i].NodeIndex == NodePositions2.Length)
                {

```

```

        PlayerStatistics.AddHealth(-
EntitySummoner.EnemiesInGame[i].GetComponent<Enemy>().Damage);
        EnqueueEnemyToRemove(EntitySummoner.EnemiesInGame[i]);
    }
    break;

case 3:
    if(EntitySummoner.EnemiesInGame[i].NodeIndex == NodePositions3.Length)
    {
        PlayerStatistics.AddHealth(-
EntitySummoner.EnemiesInGame[i].GetComponent<Enemy>().Damage);
        EnqueueEnemyToRemove(EntitySummoner.EnemiesInGame[i]);
    }
    break;
}
if (PlayerStatistics.GetHealth() <= 0)
{

}

}

NodesToUse1.Dispose();
NodesToUse2.Dispose();
NodesToUse3.Dispose();
EnemySpeeds.Dispose();
EnemyAcces.Dispose();
NodeIndices.Dispose();
NodesIndexes.Dispose();

```

ДОДАТОК Б

Друга частина лістингу коду файлу GameLoopManager.cs та файлу Enemy.cs

Друга частина лістингу коду файлу GameLoopManager.cs:

```
//Tick Tower

foreach(TowerBehavior tower in TowersInGame)
{
    tower.Target = TowerTargeting.GetTarget(tower, tower.targetType);
    tower.Tick();
}

//Apply Effects
if (EffectsQueue.Count > 0)
{
    for (int i = 0; i < EffectsQueue.Count; ++i)
    {
        ApplyEffectData CurrentApplyEffectData = EffectsQueue.Dequeue();
        Effect EffectDublicate =
CurrentApplyEffectData.EnemyToAffect.ActiveEffects.Find(x => x.EffectName ==
CurrentApplyEffectData.EffectToApply.EffectName);
        if(EffectDublicate == null)
        {
            CurrentApplyEffectData.EnemyToAffect.ActiveEffects.Add(CurrentApplyEffectDat
a.EffectToApply);
        }
        else
        {
            EffectDublicate.ExpireTime = CurrentApplyEffectData.EffectToApply.ExpireTime;
        }
    }
}

//Tick Effects
```

```

foreach(Enemy CurrentEnemy in EntitySummoner.EnemiesInGame)
{
    CurrentEnemy.Tick();
}

//Damage Enemies

if (DamageData.Count > 0)
{
    for (int i = 0; i < DamageData.Count; ++i)
    {
        EnemyDamageData CurrentDamageData = DamageData.Dequeue();
        CurrentDamageData.TargetedEnemy.Health -= CurrentDamageData.TotalDamage /
CurrentDamageData.Resistance;

        if (CurrentDamageData.TargetedEnemy.Health <= 0f)
        {
            PlayerStatistics.AddMoney(CurrentDamageData.TargetedEnemy.GetComponent<E
nemy>().GainingMoney);
            PlayerStatistics.AddScore(CurrentDamageData.TargetedEnemy.GetComponent<Ene
my>().GainingMoney);
            EnqueueEnemyToRemove(CurrentDamageData.TargetedEnemy);
        }
    }
}

//Remove Enemies

if (EnemiesToRemove.Count > 0)
{
    for (int i = 0; i < EnemiesToRemove.Count; ++i)
    {
        EntitySummoner.RemoveEnemy(EnemiesToRemove.Dequeue());
    }
}

```



```

        }
    }

    yield return null;
}

}

public static void EnqueueEffectToApply(ApplyEffectData effectData)
{
    EffectsQueue.Enqueue(effectData);
}

public static void EnqueueDamageData(EnemyDamageData damagedata)
{
    DamageData.Enqueue(damagedata);
}

public static void EnqueueEnemyIDToSummon(int ID, int NodesIndex)
{
    EnemyIDsToSummon.Enqueue(ID);
    EnemyNodeIndexesToSummon.Enqueue(NodesIndex);
}

public static void EnqueueEnemyToRemove(Enemy EnemyToRemove)
{
    EnemiesToRemove.Enqueue(EnemyToRemove);
}

}

public class Effect
{

    public Effect(string effectName, float damageRate, float damage, float expireTime)
    {

```

```

    EffectName = effectName;
    DamageRate = damageRate;
    Damage = damage;
    ExpireTime = expireTime;
}

```

```

public string EffectName;

```

```

public float DamageDelay;

```

```

public float DamageRate;

```

```

public float Damage;

```

```

public float ExpireTime;

```

```

}

```

```

public struct ApplyEffectData

```

```

{

```

```

    public ApplyEffectData(Enemy enemytoaffect, Effect effecttoapply)

```

```

    {

```

```

        EnemyToAffect = enemytoaffect;

```

```

        EffectToApply = effecttoapply;

```

```

    }

```

```

    public Enemy EnemyToAffect;

```

```

    public Effect EffectToApply;

```

```

}

```

```

public struct EnemyDamageData

```

```

{

```

```

    public EnemyDamageData(Enemy enemy, float damage, float resistance)

```

```

    {

```

```

        TargetedEnemy = enemy;

```

```

        TotalDamage = damage;

```

```

        Resistance = resistance;

```

```

    }

    public Enemy TargetedEnemy;

    public float TotalDamage;

    public float Resistance;

}

public struct MoveEnemiesJob : IJobParallelForTransform
{
    [NativeDisableParallelForRestriction] public NativeArray<Vector3> NodePosition1;
    [NativeDisableParallelForRestriction] public NativeArray<Vector3> NodePosition2;
    [NativeDisableParallelForRestriction] public NativeArray<Vector3> NodePosition3;
    [NativeDisableParallelForRestriction] public NativeArray<float> EnemySpeed;
    [NativeDisableParallelForRestriction] public NativeArray<int> NodeIndex;
    [NativeDisableParallelForRestriction] public NativeArray<int> NodesIndex;

    public float deltaTime;

    public void Execute(int index, TransformAccess transform)
    {
        switch (NodesIndex[index])
        {
            case 1:
                if(NodeIndex[index] < NodePosition1.Length)
                {
                    Vector3 PositionToMoveTo = NodePosition1[NodeIndex[index]];
                    Vector3 direction = (PositionToMoveTo - transform.position).normalized;

                    transform.position = Vector3.MoveTowards(transform.position, PositionToMoveTo,
EnemySpeed[index] * deltaTime);

                    if (direction != Vector3.zero)
                    {

                        Quaternion lookRotation = Quaternion.LookRotation(direction);
                        transform.rotation = lookRotation;
                    }
                }
            }
        }
    }
}

```

```

    }

    if (transform.position == PositionToMoveTo)
    {
        NodeIndex[index]++;
    }
}
break;

```

case 2:

```

if(NodeIndex[index] < NodePosition2.Length)
{
    Vector3 PositionToMoveTo = NodePosition2[NodeIndex[index]];
    Vector3 direction = (PositionToMoveTo - transform.position).normalized;

    transform.position = Vector3.MoveTowards(transform.position, PositionToMoveTo,
EnemySpeed[index] * deltaTime);

    if (direction != Vector3.zero)
    {
        Quaternion lookRotation = Quaternion.LookRotation(direction);
        transform.rotation = lookRotation;
    }

    if (transform.position == PositionToMoveTo)
    {
        NodeIndex[index]++;
    }
}
break;

```

case 3:

```

if(NodeIndex[index] < NodePosition3.Length)
{

```

```

Vector3 PositionToMoveTo = NodePosition3[NodeIndex[index]];
Vector3 direction = (PositionToMoveTo - transform.position).normalized;

transform.position = Vector3.MoveTowards(transform.position, PositionToMoveTo,
EnemySpeed[index] * deltaTime);

if (direction != Vector3.zero)
{

    Quaternion lookRotation = Quaternion.LookRotation(direction);
    transform.rotation = lookRotation;
}

if (transform.position == PositionToMoveTo)
{
    NodeIndex[index]++;
}
}
break;
}
}
}

```

Лістингу коду файлу Enemy.cs:

```

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

public class Enemy : MonoBehaviour
{
    [SerializeField] private ParticleSystem FireEffect;
    public List<Effect> ActiveEffects;
    public Transform RootPart;
    public float DamageResistance = 1f;

```

```

public int NodeIndex;
public float MaxHealth;
public float Health;
public float Speed;
public int ID;
public int GainingMoney;
public int Damage;
public int NodesIndex;

public void Init(int NodesIndex)
{
    ActiveEffects = new List<Effect>();
    Health = MaxHealth;
    NodeIndex = 0;
    this.NodesIndex = NodesIndex;
    switch (NodesIndex)
    {
        case 1:
            transform.position = GameLoopManager.NodePositions1[0];
            break;
        case 2:
            transform.position = GameLoopManager.NodePositions2[0];
            break;
        case 3:
            transform.position = GameLoopManager.NodePositions3[0];
            break;
    }
}

public void Tick()
{
    for (int i = 0; i < ActiveEffects.Count; i++)
    {

```

```

    if (ActiveEffects[i].ExpireTime > 0f)
    {
        if (ActiveEffects[i].DamageDelay > 0f)
        {
            ActiveEffects[i].DamageDelay -= Time.deltaTime;
        }
        else
        {
            GameLoopManager.EnqueueDamageData(new EnemyDamageData(this,
ActiveEffects[i].Damage, 1f));
            if (ActiveEffects[i].EffectName == "Fire"){
                if (!FireEffect.isPlaying) FireEffect.Play();
            }
            ActiveEffects[i].DamageDelay = 1f / ActiveEffects[i].DamageRate;
        }

        ActiveEffects[i].ExpireTime -= Time.deltaTime;
    }
    if (ActiveEffects[i].ExpireTime <= 0f && ActiveEffects[i].EffectName == "Fire"){
        FireEffect.Stop();
    }
}
ActiveEffects.RemoveAll(x => x.ExpireTime <= 0f);

}

}

```

ДОДАТОК В

Лістинг коду файлів EntitySummoner.cs та PlayerController.cs

Лістинг коду файлу EntitySummoner.cs:

```

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

public class EntitySummoner : MonoBehaviour
{
    public static List<Enemy> EnemiesInGame;
    public static List<Transform> EnemiesInGameTransform;

    public static Dictionary<Transform, Enemy> EnemyTransformPairs;
    public static Dictionary<int, GameObject> EnemyPrefabs;
    public static Dictionary<int, Queue<Enemy>> EnemyObjectPools;

    public static Enemy CurrentEnemyScript;
    private static bool IsInitialized;

    public static void Init()
    {
        if (!IsInitialized)
        {
            EnemyTransformPairs = new Dictionary<Transform, Enemy>();
            EnemyPrefabs = new Dictionary<int, GameObject>();
            EnemyObjectPools = new Dictionary<int, Queue<Enemy>>();
            EnemiesInGameTransform = new List<Transform>();
            EnemiesInGame = new List<Enemy>();

            EnemySummonData[] Enemies = Resources.LoadAll<EnemySummonData>("Enemies");

            foreach (EnemySummonData enemy in Enemies)
            {

```



```

        EnemyPrefabs.Add(enemy.EnemyID, enemy.EnemyPrefab);
        EnemyObjectPools.Add(enemy.EnemyID, new Queue<Enemy>());
    }

    IsInitialized = true;
}
else
{
    Debug.Log("ENTITYSUMMONER: THIS CLASS IS ALREADY INITIALIZED");
}

}

public static Enemy SummonEnemy(int EnemyID, int NodesIndex)
{
    Enemy SummonedEnemy = null;

    if (EnemyPrefabs.ContainsKey(EnemyID))
    {
        Queue<Enemy> ReferencedQueue = EnemyObjectPools[EnemyID];

        if (ReferencedQueue.Count > 0)
        {
            SummonedEnemy = ReferencedQueue.Dequeue();
            SummonedEnemy.Init(NodesIndex);
            SummonedEnemy.gameObject.SetActive(true);
        }
        else
        {
            GameObject NewEnemy = Instantiate(EnemyPrefabs[EnemyID],
            GameLoopManager.NodePositions1[0], Quaternion.identity);

```

```

        SummonedEnemy = NewEnemy.GetComponent<Enemy>();
        SummonedEnemy.Init(NodesIndex);

    }

}

else
{
    Debug.Log($"ENTITYSUMMONER: ENEMY WITH ID OF {EnemyID} DOES NOT
EXIST!");
    return null;
}

if (!EnemiesInGame.Contains(SummonedEnemy)) EnemiesInGame.Add(SummonedEnemy);
if (!EnemiesInGameTransform.Contains(SummonedEnemy.transform))
EnemiesInGameTransform.Add(SummonedEnemy.transform);
if (!EnemyTransformPairs.ContainsKey(SummonedEnemy.transform))
EnemyTransformPairs.Add(SummonedEnemy.transform, SummonedEnemy);
    SummonedEnemy.ID = EnemyID;
    return SummonedEnemy;
}

public static void RemoveEnemy(Enemy EnemyToRemove)
{
    EnemyObjectPools[EnemyToRemove.ID].Enqueue(EnemyToRemove);
    EnemyToRemove.gameObject.SetActive(false);

    EnemyTransformPairs.Remove(EnemyToRemove.transform);
    EnemiesInGame.Remove(EnemyToRemove);
    EnemiesInGameTransform.Remove(EnemyToRemove.transform);
}
}

```

Лістинг коду файлу PlayerController.cs:

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.EventSystems;

public class PlayerController : MonoBehaviour
{
    public Camera cam;

    [SerializeField] private NavMeshAgent agent;

    [SerializeField] private Animator animator;

    [SerializeField] private LayerMask CollideMask;

    [SerializeField] private AudioClip[] FootstepSounds;
    [SerializeField] private AudioSource audioSource;
    [SerializeField] private string towerLayerName = "Towers";
    [SerializeField] private GameObject moveToPointer;
    private bool isPlaying = false;

    private bool IsMoving = false;

    void Update()
    {
        if (Input.GetMouseButtonDown(0) && !IsPointerOverUI())
        {
            Ray ray = cam.ScreenPointToRay(Input.mousePosition);
```

```

RaycastHit hit;

if (Physics.Raycast(ray, out hit, CollideMask))
{
    if (hit.collider.gameObject.layer != LayerMask.NameToLayer("Towers"))
    {
        moveToPointer.transform.position = hit.point + new Vector3(0,0.1f,0);
        agent.SetDestination(hit.point);
        animator.SetBool("IsMoving", true);
        IsMoving = true;

    }

}

}

if (agent.remainingDistance <= agent.stoppingDistance && !agent.pathPending)
{
    animator.SetBool("IsMoving", false);
    IsMoving = false;
    moveToPointer.transform.position = new Vector3(0,0,0);
}
else
{
    animator.SetBool("IsMoving", true);
}

isPlaying = audioSource.isPlaying;

if (IsMoving && !isPlaying)
{

    PlayFootStepAudio();
}

```

```

    }
}

private bool IsPointerOverUI()
{
    return EventSystem.current != null && EventSystem.current.IsPointerOverGameObject();
}

private void PlayFootStepAudio()
{
    int n = Random.Range(1, FootstepSounds.Length);
    audioSource.clip = FootstepSounds[n];
    audioSource.PlayOneShot(audioSource.clip);
    FootstepSounds[n] = FootstepSounds[0];
    FootstepSounds[0] = audioSource.clip;
}
}

```

Лістинг коду файлу CameraController.cs:

```

using UnityEngine;

public class CameraController : MonoBehaviour
{
    [Header("Target Settings")]
    public Transform target;
    public float followSpeed = 5f;
    public float cameraHeight = 10f;

    [Header("Movement Settings")]
    public float moveSpeed = 10f;
    public float maxDistanceFromTarget = 100f;

    [Header("Zoom Settings")]
    public float zoomSpeed = 10f;

```

```

public float minZoom = 5f;
public float maxZoom = 20f;

[Header("Rotation Settings")]
public float rotationSpeed = 100f;

private float currentZoom = 10f;
private float currentRotation = 0f;

void Start()
{
    currentZoom = Mathf.Clamp(currentZoom, minZoom, maxZoom);
}

void Update()
{
    HandleZoom();
    HandleRotation();
    HandleMovement();
}

void LateUpdate()
{
    FollowTarget();
}

private void FollowTarget()
{
    Vector3 targetPosition = target.position;
    targetPosition.y += cameraHeight;
    Vector3 offset = Quaternion.Euler(0, currentRotation, 0) * new Vector3(0, 0, -currentZoom);
    Vector3 desiredPosition = targetPosition + offset;

```

```
transform.position = Vector3.Lerp(transform.position, desiredPosition, followSpeed *
Time.deltaTime);
```

```
transform.LookAt(new Vector3(target.position.x, target.position.y, target.position.z));
}
```

```
private void HandleZoom()
{
    float scroll = Input.GetAxis("Mouse ScrollWheel");
    currentZoom -= scroll * zoomSpeed;
    currentZoom = Mathf.Clamp(currentZoom, minZoom, maxZoom);
}
```

```
private void HandleRotation()
{
    if (Input.GetKey(KeyCode.Q))
    {
        currentRotation -= rotationSpeed * Time.deltaTime;
    }
    if (Input.GetKey(KeyCode.E))
    {
        currentRotation += rotationSpeed * Time.deltaTime;
    }
}
```

```
private void HandleMovement()
{
    Vector3 direction = Vector3.zero;
    if (Input.GetKey(KeyCode.W)) direction += transform.forward;
    if (Input.GetKey(KeyCode.S)) direction -= transform.forward;
    if (Input.GetKey(KeyCode.A)) direction -= transform.right;
    if (Input.GetKey(KeyCode.D)) direction += transform.right;

    direction.y = 0;
```

```
if (direction != Vector3.zero)
{
    Vector3 newPosition = transform.position + direction.normalized * moveSpeed *
Time.deltaTime;
    Vector3 flatTargetPosition = new Vector3(target.position.x, 0, target.position.z);
    Vector3 flatCameraPosition = new Vector3(newPosition.x, 0, newPosition.z);

    if (Vector3.Distance(flatCameraPosition, flatTargetPosition) <= maxDistanceFromTarget)
    {
        transform.position = newPosition;
    }
}
}
```


ДОДАТОК Г

Лістинг коду файлів PlayerStats.cs, TowerBehavior.cs та TowerPlacamnet.cs

Лістинг коду файлів PlayerStats.cs:

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

public class PlayerStats : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI ScoreText;
    [SerializeField] private Image HealthDisplay;
    [SerializeField] private TextMeshProUGUI MoneyDisplayText;
    [SerializeField] private TextMeshProUGUI HealthDisplayText;
    [SerializeField] private int StartingMoney;
    [SerializeField] private int StartingHealth;

    private int CurrentMoney;
    private int CurrentHealth;
    private int CurrentScore;

    // Start is called before the first frame update
    void Start()
    {
        CurrentMoney = StartingMoney;
        CurrentHealth = StartingHealth;
        CurrentScore = 0;
        MoneyDisplayText.SetText($"{CurrentMoney}");
        HealthDisplayText.SetText($"{CurrentHealth}");
        HealthDisplay.fillAmount = (float)CurrentHealth/(float)StartingHealth;
        ScoreText.SetText($"Score: {CurrentScore}");
    }
}
```

```
public void AddMoney(int MoneyToAdd)
{
    CurrentMoney += MoneyToAdd;
    MoneyDisplayText.SetText($"${CurrentMoney}");
}

public int GetMoney()
{
    return CurrentMoney;
}

public void AddHealth(int HealthToAdd)
{
    CurrentHealth += HealthToAdd;
    HealthDisplayText.SetText($"{CurrentHealth}");
    HealthDisplay.fillAmount = (float)CurrentHealth/(float)StartingHealth;
}

public int GetHealth()
{
    return CurrentHealth;
}

public void AddScore(int ScoreToAdd)
{
    CurrentScore += ScoreToAdd;
    ScoreText.SetText($"Score: {CurrentScore}");
}

public int GetScore()
{
    return CurrentScore;
}
}
```

Лістинг коду файлу TowerBehavior.cs:

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

public class TowerBehavior : MonoBehaviour
{
    public LayerMask EnemiesLayer;
    public Enemy Target;
    public Transform TowerPivot;

    public float Damage;
    public float Firerate;
    public float Range;
    public int SummonCost = 100;
    public int Level = 1;
    public int MaxLevel = 4;
    public int UpgradeCost = 100;
    public int SellValue;
    public bool JustPlaced = true;

    private float Delay;

    private IDamageMethod CurrentDamageMethodClass;

    [SerializeField] private PlayerStats playerStats;

    [SerializeField] private GameObject nextUpgradePrefab;
    [SerializeField] private AudioSource StartingSound;

    public TowerTargeting.TargetType targetType = TowerTargeting.TargetType.First;
```

```

// Start is called before the first frame update
void Start()
{
    CurrentDamageMethodClass = GetComponent<IDamageMethod>();

    if (CurrentDamageMethodClass == null)
    {
        Debug.Log("TOWERS: No damage class attached to given tower!");
    }

    CurrentDamageMethodClass.Init(Damage, Firerate);
    playerStats = FindObjectOfType<PlayerStats>();
    StartingSound.Play();
    Delay = 1 / Firerate;
    SellValue = UpgradeCost / 2;
}

// Update is called once per frame
public void Tick()
{
    CurrentDamageMethodClass.DamageTick(Target);

    if (Target != null)
    {
        TowerPivot.transform.LookAt(Target.transform.position);
    }
}

private void OnDrawGizmos(){
    Gizmos.color = Color.grey;
    Gizmos.DrawWireSphere(transform.position, Range);
}

```

```

private void ResetJustPlaced()
{
    JustPlaced = false;
}

public void ResetJustPlacedInvoke()
{
    Invoke(nameof(ResetJustPlaced), 1f);
}

public void UpgradeTower()
{
    if (playerStats != null && playerStats.GetMoney() >= UpgradeCost)
    {
        if (nextUpgradePrefab != null)
        {
            playerStats.AddMoney(-UpgradeCost);

            GameObject upgradedTower = Instantiate(nextUpgradePrefab, transform.position,
transform.rotation);
            GameLoopManager.TowersInGame.Add(upgradedTower.GetComponent<TowerBehavior>());
            upgradedTower.GetComponent<TowerBehavior>().ResetJustPlacedInvoke();

            GameLoopManager.TowersInGame.Remove(this);
            Destroy(gameObject);
        }
        else
        {
            Debug.Log("No next upgrade prefab available.");
        }
    }
}

```

```

        }
    }
    else
    {
        Debug.Log("Not enough resources to upgrade the tower.");
    }
}

public void SellTower()
{
    if (playerStats != null)
    {
        playerStats.AddMoney(SellValue);
        GameLoopManager.TowersInGame.Remove(this);
        Debug.Log($"Tower sold for {SellValue} resources.");
    }

    Destroy(gameObject);
}

public void ChangeTargetType(TowerTargeting.TargetType targetType)
{
    this.targetType = targetType;
}
}

```

Лістинг коду файлу TowerPlacamnet.cs:

```

using System.Collections;
using System.Collections.Generic;
using System.Runtime.ConstrainedExecution;
using TMPro;
using UnityEngine;

```

```

public class TowerPlacament : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI TextQ;
    [SerializeField] private TextMeshProUGUI TextR;
    [SerializeField] private LayerMask PlaycementCheckMask;
    [SerializeField] private LayerMask PlaycementCollideMask;
    [SerializeField] private Camera PlayerCamera;
    [SerializeField] private PlayerStats PlayerStatistics;
    public GameObject CurrentPlacingTower;

    void Start()
    {
        PlayerStatistics = FindAnyObjectByType<PlayerStats>();
    }

    void Update()
    {
        if (CurrentPlacingTower != null)
        {
            TextQ.GetComponent<TextMeshProUGUI>().text = "Press right mouse button to cancel";
            Ray camray = PlayerCamera.ScreenPointToRay(Input.mousePosition);
            RaycastHit HitInfo;

            if (Physics.Raycast(camray, out HitInfo, 100f, PlaycementCollideMask))
            {
                CurrentPlacingTower.transform.position = HitInfo.point;
            }

            if (Input.GetMouseButtonDown(1))
            {
                Destroy(CurrentPlacingTower);
                CurrentPlacingTower = null;
            }
        }
    }
}

```

```

        return;
    }

    if (Input.GetMouseButtonDown(0) && HitInfo.collider.gameObject != null)
    {

        if (!HitInfo.collider.gameObject.CompareTag("Cant Place"))
        {
            BoxCollider TowerColider =
CurrentPlacingTower.gameObject.GetComponent<BoxCollider>();
            TowerColider.isTrigger = true;

            Vector3 BoxCenter = CurrentPlacingTower.gameObject.transform.position +
(TowerColider.center * 0.15f);
            Vector3 HalfExtents = TowerColider.size / 2;
            HalfExtents = HalfExtents * 0.15f;

            if (!Physics.CheckBox(BoxCenter, HalfExtents, Quaternion.identity,
PlacementCheckMask, QueryTriggerInteraction.Ignore))
            {
                TowerBehavior CurrentTowerBehavior =
CurrentPlacingTower.GetComponent<TowerBehavior>();
                GameLoopManager.TowersInGame.Add(CurrentTowerBehavior);
                PlayerStatistics.AddMoney(-CurrentTowerBehavior.SummonCost);
                TowerColider.isTrigger = false;
                CurrentTowerBehavior.ResetJustPlacedInvoke();
                CurrentPlacingTower = null;
            }
        }

    }

}

else

```



```

    {
        TextQ.GetComponent<TextMeshProUGUI>().text = "";
    }

}

public void SetTowerToPlace(GameObject tower)
{
    int TowerSummonCost = tower.GetComponent<TowerBehavior>().SummonCost;

    if (PlayerStatistics.GetMoney() >= TowerSummonCost)
    {
        CurrentPlacingTower = Instantiate(tower, Vector3.zero, Quaternion.identity);
    }
    else
    {
        Debug.Log("You need more money to purchase a "+ tower.name);
    }
}
}

```

ДОДАТОК Г

Лістинг коду файлу TowerTargeting.cs

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

public class TowerTargeting
{
    public enum TargetType
    {
        First,
        Last,
        Close,
        Strong,
        Weak,
    }

    public static Enemy GetTarget(TowerBehavior CurrentTower, TargetType TargetMethod)
    {
        Collider[] EnemiesInRange = Physics.OverlapSphere(CurrentTower.transform.position,
CurrentTower.Range, CurrentTower.EnemiesLayer);

        NativeArray<EnemyData> EnemiesToCalculate = new
NativeArray<EnemyData>(EnemiesInRange.Length, Allocator.TempJob);
        NativeArray<Vector3> NodePositions1 = new
NativeArray<Vector3>(GameLoopManager.NodePositions1, Allocator.TempJob);
        NativeArray<Vector3> NodePositions2 = new
NativeArray<Vector3>(GameLoopManager.NodePositions2, Allocator.TempJob);
        NativeArray<Vector3> NodePositions3 = new
NativeArray<Vector3>(GameLoopManager.NodePositions3, Allocator.TempJob);
```

```

        NativeArray<float> NodeDistances1 = new
NativeArray<float>(GameLoopManager.NodeDistances1, Allocator.TempJob);
        NativeArray<float> NodeDistances2 = new
NativeArray<float>(GameLoopManager.NodeDistances2, Allocator.TempJob);
        NativeArray<float> NodeDistances3 = new
NativeArray<float>(GameLoopManager.NodeDistances3, Allocator.TempJob);
        NativeArray<int> EnemyToIndex = new NativeArray<int>(new int[] { -1 },
Allocator.TempJob);
        int EnemyIndexToReturn = -1;

        for (int i = 0; i < EnemiesToCalculate.Length; i++)
        {
            Enemy CurrentEnemy = EnemiesInRange[i].transform.GetComponent<Enemy>();
            int EnemyIndexInList = EntitySummoner.EnemiesInGame.FindIndex(x => x ==
CurrentEnemy);
            EnemiesToCalculate[i] = new EnemyData(CurrentEnemy.transform.position,
CurrentEnemy.NodeIndex, CurrentEnemy.Health, EnemyIndexInList, CurrentEnemy.NodesIndex);
        }

        SearchForEnemy EnemySearchJob = new SearchForEnemy
        {
            _EnemiesToCalculate = EnemiesToCalculate,
            _NodeDistances1 = NodeDistances1,
            _NodeDistances2 = NodeDistances2,
            _NodeDistances3 = NodeDistances3,
            _NodePositions1 = NodePositions1,
            _NodePositions2 = NodePositions2,
            _NodePositions3 = NodePositions3,
            _EnemyToIndex = EnemyToIndex,
            TargetingType = (int)TargetMethod,
            TowerPosition = CurrentTower.transform.position
        };

        switch ((int)TargetMethod)
        {

```

```

case 0: // First
    EnemySearchJob.CompareValue = Mathf.Infinity;
    break;

case 1: // Last
    EnemySearchJob.CompareValue = Mathf.NegativeInfinity;
    break;

case 2: // Close
    goto case 0;

case 3: //Strong
    goto case 1;

case 4: //Weak
    goto case 0;
}

JobHandle dependency = new JobHandle();
JobHandle SearchJobHandle = EnemySearchJob.Schedule(EnemiesToCalculate.Length,
dependency);

SearchJobHandle.Complete();

if (EnemyToIndex[0] != -1)
{
    EnemyIndexToReturn = EnemiesToCalculate[EnemyToIndex[0]].EnemyIndex;

    EnemiesToCalculate.Dispose();
    NodeDistances1.Dispose();
    NodeDistances2.Dispose();
    NodeDistances3.Dispose();
    NodePositions1.Dispose();
    NodePositions2.Dispose();
    NodePositions3.Dispose();
}

```

```

        EnemyToIndex.Dispose();
        if (EnemyIndexToReturn < EntitySummoner.EnemiesInGame.Count) return
EntitySummoner.EnemiesInGame[EnemyIndexToReturn];
        else return null;
    }

```

```

        EnemiesToCalculate.Dispose();
        NodeDistances1.Dispose();
        NodeDistances2.Dispose();
        NodeDistances3.Dispose();
        NodePositions1.Dispose();
        NodePositions2.Dispose();
        NodePositions3.Dispose();
        EnemyToIndex.Dispose();

        return null;

    }

```

```

struct EnemyData
{
    public EnemyData(Vector3 position, int nodeindex, float hp, int enemyIndex, int nodesindex)
    {
        EnemyPosition = position;
        EnemyIndex = enemyIndex;
        NodeIndex = nodeindex;
        Health = hp;
        NodesIndex = nodesindex;
    }

    public Vector3 EnemyPosition;
    public int EnemyIndex;
    public int NodeIndex;

```

```

    public int NodesIndex;
    public float Health;

}

struct SearchForEnemy : IJobFor
{

    [ReadOnly] public NativeArray<EnemyData> _EnemiesToCalculate;
    [ReadOnly] public NativeArray<Vector3> _NodePositions1;
    [ReadOnly] public NativeArray<Vector3> _NodePositions2;
    [ReadOnly] public NativeArray<Vector3> _NodePositions3;
    [ReadOnly] public NativeArray<float> _NodeDistances1;
    [ReadOnly] public NativeArray<float> _NodeDistances2;
    [ReadOnly] public NativeArray<float> _NodeDistances3;

    [NativeDisableParallelForRestriction] public NativeArray<int> _EnemyToIndex;
    public Vector3 TowerPosition;
    public float CompareValue;
    public int TargetingType;

    public void Execute(int index)
    {
        float CurrentEnemyDistanceToEnd;
        float DstanceToEnemy = 0;
        switch (TargetingType)
        {
            case 0: //First
                CurrentEnemyDistanceToEnd = GetDistanceToEnd(_EnemiesToCalculate[index]);
                if (CurrentEnemyDistanceToEnd < CompareValue)
                {
                    _EnemyToIndex[0] = index;
                    CompareValue = CurrentEnemyDistanceToEnd;
                }
            }
        }
    }
}

```

```
break;
```

```
case 1: //Last
```

```
    CurrentEnemyDistanceToEnd = GetDistanceToEnd(_EnemiesToCalculate[index]);
    if (CurrentEnemyDistanceToEnd > CompareValue)
    {
        _EnemyToIndex[0] = index;
        CompareValue = CurrentEnemyDistanceToEnd;
    }
```

```
break;
```

```
case 2: //Close
```

```
    DstanceToEnemy = Vector3.Distance (TowerPosition,
    _EnemiesToCalculate[index].EnemyPosition);
    if (DstanceToEnemy < CompareValue)
    {
        _EnemyToIndex[0] = index;
        CompareValue = DstanceToEnemy;
    }
    break;
```

```
case 3: //Strong
```

```
    if (_EnemiesToCalculate[index].Health > CompareValue)
    {

        _EnemyToIndex[0] = index;
        CompareValue = _EnemiesToCalculate[index].Health;
    }
```

```
break;
```

```
case 4: //Weak
```

```
    if (_EnemiesToCalculate[index].Health < CompareValue)
    {
```

```

        _EnemyToIndex[0] = index;
        CompareValue = _EnemiesToCalculate[index].Health;
    }

    break;
}
}

private float GetDistanceToEnd(EnemyData EnemyToEvaluate)
{
    switch(EnemyToEvaluate.NodesIndex)
    {
        case 1:
            float FinalDistance = Vector3.Distance(EnemyToEvaluate.EnemyPosition,
            _NodePositions1[EnemyToEvaluate.NodeIndex]);

            for (int i = EnemyToEvaluate.NodeIndex; i < _NodeDistances1.Length; i++)
            {
                FinalDistance += _NodeDistances1[i];
            }
            return FinalDistance;

        case 2:
            FinalDistance = Vector3.Distance(EnemyToEvaluate.EnemyPosition,
            _NodePositions2[EnemyToEvaluate.NodeIndex]);

            for (int i = EnemyToEvaluate.NodeIndex; i < _NodeDistances2.Length; i++)
            {
                FinalDistance += _NodeDistances2[i];
            }
            return FinalDistance;

        case 3:

```



```
        FinalDistance = Vector3.Distance(EnemyToEvaluate.EnemyPosition,
        _NodePositions3[EnemyToEvaluate.NodeIndex]);

        for (int i = EnemyToEvaluate.NodeIndex; i < _NodeDistances3.Length; i++)
        {
            FinalDistance += _NodeDistances3[i];
        }
        return FinalDistance;

    default:
        goto case 1;
    }

}

}

}
```

ДОДАТОК Д

Лістинг коду файлів CameraController.cs, StandardDamage.cs та LaserDamage.cs

Лістинг коду файлу TowerUpgradeManager.cs:

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

public class TowerUpgradeManager : MonoBehaviour
{
    [SerializeField] private Camera mainCamera;
    [SerializeField] private PiUiManager piUiManager;
    [SerializeField] private string menuName = "TowerMenu";

    private TowerBehavior selectedTower;
    private string lastUpgradeName = "Upgrade";
    private string lastSellName = "Sell";
    private string lastTargetTypeName = "Change target type";

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            OpenMenuAtMouse();
        }
    }

    private void OpenMenuAtMouse()
    {
        Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
        if (Physics.Raycast(ray, out RaycastHit hit))
```

```

{
    TowerBehavior tower = hit.collider.GetComponent<TowerBehavior>();
    if (tower != null && !tower.JustPlaced)
    {
        selectedTower = tower;

        Vector2 screenPos = Input.mousePosition;
        piUiManager.ChangeMenuState(menuName, screenPos);

        UpdateMenuOptions();
    }
}

private void UpdateMenuOptions()
{
    PiUI towerMenu = piUiManager.GetPiUIOf(menuName);

    foreach (PiUI.PiData option in towerMenu.piData)
    {
        option.onSlicePressed.RemoveAllListeners();
    }

    if (selectedTower != null)
    {
        foreach (PiUI.PiData option in towerMenu.piData)
        {
            if (option.sliceLabel == lastUpgradeName)
            {

```

```

bool canUpgrade = selectedTower.Level < selectedTower.MaxLevel;
option.isInteractable = canUpgrade;

if (canUpgrade)
{
    lastUpgradeName = canUpgrade ? $"Upgrade: {selectedTower.UpgradeCost}" :
"Max Level";
    option.sliceLabel = lastUpgradeName;
    option.onSlicePressed = new UnityEngine.Events.UnityEvent();
    option.onSlicePressed.AddListener(UpgradeTower);
}
}
else if (option.sliceLabel == lastSellName)
{
    lastSellName = $"Sell: {selectedTower.SellValue}";
    option.sliceLabel = lastSellName;
    option.onSlicePressed = new UnityEngine.Events.UnityEvent();
    option.onSlicePressed.AddListener(SellTower);
}
else if (option.sliceLabel == lastTargetTypeName)
{
    lastTargetTypeName = $"Change target type: {selectedTower.targetType}";
    option.sliceLabel = lastTargetTypeName;
    option.onSlicePressed = new UnityEngine.Events.UnityEvent();
    option.onSlicePressed.AddListener(() =>
    {
        if (Input.GetMouseButtonDown(0))
        {
            ChangeTargetType(selectedTower.targetType);
        }
    });
}
}
}

```

```

        piUiManager.UpdatePiMenu(menuName);
    }

    private void UpgradeTower()
    {
        if (selectedTower != null)
        {
            selectedTower.UpgradeTower();
            Debug.Log("Tower upgraded!");
        }

        CloseMenu();
    }

    private void SellTower()
    {
        if (selectedTower != null)
        {
            selectedTower.SellTower();
            Debug.Log("Tower sold!");
        }

        CloseMenu();
    }

    private void ChangeTargetType(TowerTargeting.TargetType targetType)
    {
        if (selectedTower != null)
        {
            if (targetType == TowerTargeting.TargetType.First)
            {
                selectedTower.ChangeTargetType(TowerTargeting.TargetType.Last);
                UpdateMenuOptions();
            }
        }
    }

```

```

        return;
    }
    else if (targetType == TowerTargeting.TargetType.Last)
    {
        selectedTower.ChangeTargetType(TowerTargeting.TargetType.Close);
        UpdateMenuOptions();
        return;
    }
    else if (targetType == TowerTargeting.TargetType.Close)
    {
        selectedTower.ChangeTargetType(TowerTargeting.TargetType.Strong);
        UpdateMenuOptions();
        return;
    }
    else if (targetType == TowerTargeting.TargetType.Strong)
    {
        selectedTower.ChangeTargetType(TowerTargeting.TargetType.Weak);
        UpdateMenuOptions();
        return;
    }
    else if (targetType == TowerTargeting.TargetType.Weak)
    {
        selectedTower.ChangeTargetType(TowerTargeting.TargetType.First);
        UpdateMenuOptions();
        return;
    }
}

private void ChangeTargetTypeToClose()
{
    if (selectedTower != null)
    {

```

```

    }

    CloseMenu();
}

private void ChangeTargetTypeToStrong()
{
    if (selectedTower != null)
    {
        selectedTower.ChangeTargetType(TowerTargeting.TargetType.Strong);
    }

    CloseMenu();
}

private void ChangeTargetTypeToWeak()
{
    if (selectedTower != null)
    {
        selectedTower.ChangeTargetType(TowerTargeting.TargetType.Weak);
    }

    CloseMenu();
}

private void CloseMenu()
{
    piUiManager.ChangeMenuState(menuName);
    selectedTower = null;
}
}

```

Лістинг коду файлу StandardDamage.cs:

```

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

```

```

public interface IDamageMethod
{
    public void DamageTick(Enemy Target);
    public void Init(float Damage, float Firerate);
}

public class StandardDamage : MonoBehaviour, IDamageMethod
{
    [SerializeField] private AudioSource Sound;
    [SerializeField] private ParticleSystem ArrowShot;

    private float Damage;
    private float Firerate;
    private float Delay;

    public void Init(float Damage, float Firerate)
    {
        this.Damage = Damage;
        this.Firerate = Firerate;
        Delay = 1 / Firerate;
    }

    public void DamageTick(Enemy Target)
    {
        if (Target)
        {
            if (Delay > 0f)
            {
                Delay -= Time.deltaTime;
                return;
            }

            Sound.Play();
            ArrowShot.Play();
        }
    }
}

```



```

        GameLoopManager.EnqueueDamageData(new EnemyDamageData(Target, Damage,
Target.DamageResistance));

```

```

        Delay = 1f / Firerate;
    }

}

}

```

Лістинг коду файлу LaserDamage.cs:

```

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

public class LaserDamage : MonoBehaviour, IDamageMethod
{
    [SerializeField] private AudioSource Sound;
    [SerializeField] private Transform LaserPivot;
    [SerializeField] private LineRenderer LaserRenderer;

    private float Damage;
    private float Firerate;
    private float Delay;

    public void Init(float Damage, float Firerate)
    {
        this.Damage = Damage;
        this.Firerate = Firerate;
        Delay = 1 / Firerate;
    }

    public void DamageTick(Enemy Target)
    {
        if (Target)

```

```

{

    LaserRenderrer.enabled = true;
    LaserRenderrer.SetPosition(0, LaserPivot.position);
    LaserRenderrer.SetPosition(1, Target.RootPart.position);

    if (Delay > 0f)
    {
        Delay -= Time.deltaTime;
        return;
    }

    if (!Sound.isPlaying) Sound.Play();

    GameLoopManager.EnqueueDamageData(new EnemyDamageData(Target, Damage,
Target.DamageResistance));

    Delay = 1f / Firerate;

    LaserRenderrer.enabled = false;
}
else
{
    Sound.Stop();
    LaserRenderrer.SetPosition(0, new Vector3(5000f, 5000f, 5000f));
    LaserRenderrer.SetPosition(1, new Vector3(5000f, 5000f, 5000f));
}

}
}

```

ДОДАТОК Е

Лістинг коду файлів MissileDamage.cs, MissileCollisionManager.cs, FlamethrowerDamage.cs, FireTriggerManager.cs, EnemySummonData.cs, ButtonManager.cs, ScoreWindowScript.cs та SettingsMenu.cs

Лістинг коду файлу MissileDamage.cs:

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

public class MissileDamage : MonoBehaviour, IDamageMethod
{
    public LayerMask EnemiesLayer;

    [SerializeField] private AudioSource Sound;
    [SerializeField] private ParticleSystem MissileSystem;
    [SerializeField] private Transform TowerHead;

    private ParticleSystem.MainModule MissileSystemMain;
    public float Damage;
    private float Firerate;
    private float Delay;

    public void Init(float Damage, float Firerate)
    {
        MissileSystemMain = MissileSystem.main;

        this.Damage = Damage;
        this.Firerate = Firerate;
        Delay = 1 / Firerate;
    }

    public void DamageTick(Enemy Target)
```

```

{
    if (Target)
    {
        if (Delay > 0f)
        {
            Delay -= Time.deltaTime;
            return;
        }

        Sound.Play();

        MissileSystemMain.startRotationX = TowerHead.forward.x;
        MissileSystemMain.startRotationY = TowerHead.forward.y;
        MissileSystemMain.startRotationZ = TowerHead.forward.z;

        MissileSystem.Play();
        Delay = 1f / Firerate;
    }
}
}

```

Лістинг коду файлу MissileCollisionManager.cs:

```

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

public class MissileCollisionManager : MonoBehaviour
{
    [SerializeField] private MissileDamage BaseClass;
    [SerializeField] private ParticleSystem ExplosionSystem;
    [SerializeField] private ParticleSystem MissileSystem;
    [SerializeField] private float ExplosionRadius;

```

```

private List<ParticleCollisionEvent> MissileCollisions;

private void Start()
{
    MissileCollisions = new List<ParticleCollisionEvent>();
}

private void OnParticleCollision(GameObject other)
{
    MissileSystem.GetCollisionEvents(other, MissileCollisions);

    for (int collisionevent = 0; collisionevent < MissileCollisions.Count; collisionevent++)
    {
        ExplosionSystem.transform.position = MissileCollisions[collisionevent].intersection;
        ExplosionSystem.Play();

        Collider[] EnemiesInRadius =
Physics.OverlapSphere(MissileCollisions[collisionevent].intersection, ExplosionRadius,
BaseClass.EnemiesLayer);

        for (int i = 0; i < EnemiesInRadius.Length; i++)
        {
            Enemy EnemyToDamage =
EntitySummoner.EnemyTransformPairs[EnemiesInRadius[i].transform];
            EnemyDamageData DamageToApply = new EnemyDamageData(EnemyToDamage,
BaseClass.Damage, EnemyToDamage.DamageResistance);
            GameLoopManager.EnqueueDamageData(DamageToApply);
        }
    }
}

```

Лістинг коду файлу FlamethrowerDamage.cs:

```

using System.Collections;

```

```

using System.Collections.Generic;
using UnityEngine;

public class FlamethrowerDamage : MonoBehaviour, IDamageMethod
{
    [SerializeField] private AudioSource Sound;
    [SerializeField] private ParticleSystem FireEffect;
    [SerializeField] private Collider FireTrigger;

    [HideInInspector] public float Damage;
    [HideInInspector] public float Firerate;

    public void Init(float Damage, float Firerate)
    {
        this.Damage = Damage;
        this.Firerate = Firerate;
    }

    public void DamageTick(Enemy Target)
    {
        FireTrigger.enabled = Target != null;

        if (Target)
        {
            if (!FireEffect.isPlaying) FireEffect.Play();
            if (!Sound.isPlaying) Sound.Play();
            return;
        }

        Sound.Stop();
        FireEffect.Stop();
    }
}

```

Лістинг коду файлу FireTriggerManager.cs:

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

public class FireTriggerManager : MonoBehaviour
{
    [SerializeField] private FlamethrowerDamage BaseClass;

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Enemy"))
        {
            Effect FlameEffect = new Effect("Fire", BaseClass.Firerate, BaseClass.Damage, 5f);
            ApplyEffectData effectData = new
            ApplyEffectData(EntitySummoner.EnemyTransformPairs[other.transform], FlameEffect);
            GameLoopManager.EnqueueEffectToApply(effectData);
        }
    }
}
```

Лістинг коду файлу EnemySummonData.cs:

```
[CreateAssetMenu(fileName = "New EnemySummonData", menuName = "Create
EnemySummonData")]
public class EnemySummonData : ScriptableObject
{
    public GameObject EnemyPrefab;
    public int EnemyID;
}
```

Лістинг коду файлу ButtonManager.cs:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ButtonManager : MonoBehaviour
{
    [SerializeField] private GameObject Menu;
    [SerializeField] private GameObject PlayerInterface;
    [SerializeField] private GameObject Settings;
    [SerializeField] private GameObject Tutorial;
    [SerializeField] private GameObject ScoreWindow;
    [SerializeField] private GameObject IntroductoryWindow;

    private GameLoopManager gameLoopManager;

    void Start()
    {
        gameLoopManager = GameObject.FindAnyObjectByType<GameLoopManager>();
    }

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

    public void StartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void Continue()
    {
        Menu.SetActive(false);
        PlayerInterface.SetActive(true);
    }
}

```



```
        gameLoopManager.ContinueGameLoop();
    }

    public void ContinueGame()
    {
        IntroductoryWindow.SetActive(false);
        PlayerInterface.SetActive(true);
        Tutorial.SetActive(true);
        gameLoopManager.ContinueGameLoop();
    }

    public void OpenSettings()
    {
        Settings.SetActive(true);
    }

    public void CloseSettings()
    {
        Settings.SetActive(false);
    }

    public void TutorialClose()
    {
        Tutorial.SetActive(false);
    }

    public void FinishTheGame()
    {
        ScoreWindow.SetActive(true);
        Menu.SetActive(false);
        gameLoopManager.StopGameLoop();
    }

    public void TryAgain()
```

```

{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}

public void MainMenu()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
}
}

```

Лістинг коду файлу ScoreWindowScript.cs:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.SocialPlatforms.Impl;

public class ScoreWindowScript : MonoBehaviour
{
    [SerializeField] private GameLoopManager gameLoopManager;
    [SerializeField] private PlayerStats playerStats;
    [SerializeField] private TextMeshProUGUI statisticsText;

    void Start()
    {
        gameLoopManager = GameObject.FindAnyObjectByType<GameLoopManager>();
        playerStats = GameObject.FindAnyObjectByType<PlayerStats>();
    }

    void Update()
    {
        int score = playerStats.GetScore();
        statisticsText.text = $"Waves survived: {gameLoopManager.currentWave - 1}\n" +
            $"Score: {score}\n";
    }
}

```

```
}
```

Лістинг коду файлу SettingsMenu.cs:

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

public class SettingsMenu : MonoBehaviour
{
    public AudioManager audioMixer;
    public TMP_Dropdown resolutionDropdown;
    Resolution[] resolutions;
    void Start()
    {
        resolutions = Screen.resolutions;
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        List<Resolution> uniqueResolutions = new List<Resolution>();
        int currentResolutionIndex = 0;
        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + "x" + resolutions[i].height;
            if (!options.Contains(option))
            {
                options.Add(option);
                uniqueResolutions.Add(resolutions[i]);
                if (resolutions[i].width == Screen.currentResolution.width && resolutions[i].height ==
Screen.currentResolution.height)
                {
                    currentResolutionIndex = uniqueResolutions.Count - 1;
                }
            }
        }
    }
}
```

```

    }
}
resolutions = uniqueResolutions.ToArray();
resolutionDropdown.AddOptions(options);
resolutionDropdown.value = currentResolutionIndex;
resolutionDropdown.RefreshShownValue();
}
public void SetResolution(int resolutionIndex)
{
    Resolution resolution = resolutions[resolutionIndex];
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
}
public void SetVolume(float volume)
{
    audioMixer.SetFloat("volume", volume);
}
public void SetQuality(int qualityIndex)
{
    QualitySettings.SetQualityLevel(qualityIndex);
}
public void SetFullscreen(bool isFullscreen)
{
    Screen.fullScreen = isFullscreen;
}
}

```

ДОДАТОК Ж

Лістинг коду файлу WindowOpener.cs

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

public class WindowOpener : MonoBehaviour
{
    [SerializeField] private GameObject menu;
    [SerializeField] private GameObject playerInterface;
    private GameLoopManager gameLoopManager;

    void Start()
    {
        gameLoopManager = GameObject.FindAnyObjectByType<GameLoopManager>();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            menu.SetActive(true);
            playerInterface.SetActive(false);
            gameLoopManager.StopGameLoop();
        }
    }
}
```