НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Гра у жанрі Action з елементами просторової навігації

Виконав студент IV курсу, групи ІП-13
(шифр групи)

Жмайло Дмитро Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст.викл., д-р філософії, Головченко М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст.викл., д-р філософії, Головченко М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., доц., Жураковська О. С. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Жмайлу Дмитру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Гра у жанрі 3D Action з елементами просторової навігації

керівник проєкту Головченко Максим Миколайович, ст.викл., д-р філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: актуальність теми ігор у жанрі Action, аналіз ринку відеоігор, аналіз відомих технічних рішень та їх недоліків, постановка задачі дипломного проєкту.
2) Інформаційне забезпечення: опис роботи редактора рівнів, збереження файлів рівнів гри та їх завантаження.
3) Алгоритмічне забезпечення: обгрунтування та опис алгоритму дерева прийняття рішень для створення штучного інтелекту ворогів
4) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення.
5) Технологічний розділ: керівництво користувача, методика тестування програмного продукту, процеси розгортання та супроводу.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема структурна класів програмного забезпечення _____

3) Креслення вигляду екранних форм _____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» березня 2025 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення рекомендованої літератури	16.03.2025	
2.	Аналіз існуючих методів розв’язання задачі	02.04.2025	
3.	Постановка та формалізація задачі	10.04.2025	
4.	Розробка інформаційного забезпечення	16.04.2025	
5.	Алгоритмізація задачі	21.04.2025	
6.	Обґрунтування вибору використаних технічних засобів	22.04.2025	
7.	Розробка програмного забезпечення	09.05.2025	
8.	Налагодження програми	11.05.2025	
9.	Виконання графічних документів	13.05.2025	
10.	Оформлення пояснювальної записки	17.05.2025	
11.	Подання ДП на попередній захист	02.06.2025	
12.	Подання ДП рецензенту	10.06.2025	
13.	Подання ДП на основний захист	14.06.2025	

Студент _____ Дмитро ЖМАЙЛО

(підпис) (ініціали, прізвище)

Керівник _____ Максим ГОЛОВЧЕНКО

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 41 таблицю, 27 рисунків та 28 джерел – загалом 83 сторінки.

Дипломний проєкт присвячений розробці комп’ютерної гри у жанрі 3D Action з елементами просторової навігації.

Мета - підвищення якості гри у жанрі 3D Action з елементами просторової навігації за рахунок створення штучного інтелекту, що формалізує поведінку противників на основі оригінального дерева прийняття рішень.

У першому розділі розглянуто аналіз ринку відеоігор жанру Action, описано його недоліки та способи покращення. Проаналізовано існуючі технічні рішення та змодельовано бізнес-процеси ігрового процесу та процесу створення рівнів.

У другому розділі розглянуто варіанти використання, розроблено функціональні та нефункціональні вимоги, розглянуто системні вимоги, проаналізовано економічні показники та поставлено завдання на розробку програмного забезпечення.

У третьому розділі описано архітектуру гри, обгрунтовано вибір засобів розробки, описано алгоритм ШІ ворогів, структуру збереження даних рівнів.

У четвертому розділі проаналізовано якість гри за допомогою аналізу якості коду та аналізу продуктивності гри. Описано мануальне тестування всього функціоналу гри та наведено опис контрольного прикладу для редактора рівнів.

П’ятий розділ присвячено розгортанню програмного забезпечення за допомогою утиліт Unity. Описано супровід програмного забезпечення з використанням платформ Git та itch.io.

КЛЮЧОВІ СЛОВА: КОМП’ЮТЕРНА ГРА, 3D ACTION, UNITY, РЕДАКТОР РІВНІВ, WINDOWS, ШТУЧНИЙ ІНТЕЛЕКТ, ДЕРЕВО ПРИЙНЯТТЯ РІШЕНЬ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 41 tables, 27 figures and 28 sources – in total 83 pages.

The diploma project is dedicated to the development of a computer game in the 3D Action genre with elements of spatial navigation.

The purpose of the diploma project is to improve the quality of a 3D Action game with spatial navigation elements by creating artificial intelligence that formalizes enemy behavior based on an original decision tree.

The first section presents an analysis of the Action video game market, describes its drawbacks and ways to improve it. Existing technical solutions are analyzed, and business processes of gameplay and level editor are modeled.

The second section considers use case scenarios, develops functional and non-functional requirements, discusses system requirements, analyzes economic indicators, and sets the task for software development.

The third section describes the game architecture, justifies the choice of development tools, presents the enemy AI algorithm, and outlines the structure for level data storage.

The fourth section analyzes game quality through code quality assessment and performance analysis. Manual testing of all game functionality is described, along with a test case example for the level editor.

The fifth section focuses on deploying the software using Unity tools. It also describes software maintenance through Git and itch.io platforms.

KEYWORDS: COMPUTER GAME, 3D ACTION, UNITY, LEVEL EDITOR, WINDOWS, ARTIFICIAL INTELLIGENCE, DECISION TREE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1311.045490.01.91	Технічне завдання	21	
3	А4	КПІ.ІП-1311.045490.02.81	Пояснювальна записка	83	
4	А4	КПІ.ІП-1311.045490.03.12	Текст програми	29	
5	А4	КПІ.ІП-1311.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1311.045490.05.34	Керівництво користувача	15	
7	А3	КПІ.ІП-1311.045490.06.99	Графічний матеріал	5	

					КПІ.ІП-1311.045490.00.90		
Змін.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Жмайло Д.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Головченко М.М.					
Н.контр.		Головченко М.М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-13	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

Гра у жанрі 3D Action з елементами просторової навігації

Технічне завдання

КПІ.ІП-1311.045490.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро ЖМАЙЛО

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Гра у жанрі 3D Action з елементами просторової навігації.

Галузь застосування: гра для персональних комп’ютерів

Наведене технічне завдання поширюється на розробку гри для персонального комп'ютера в жанрі 3D Action з елементами просторової навігації, яка використовується у розважальних цілях та призначена для людей, які зацікавлені у грі на персональних комп’ютерах.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки гри в жанрі 3D Action з елементами просторової навігації є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для забезпечення ігрового процесу гри в жанрі 3D Action з елементами просторової навігації для широкого кола користувачів.

Метою розробки є підвищення якості гри у жанрі 3D Action з елементами просторової навігації за рахунок створення штучного інтелекту, що формалізує поведінку противників на основі оригінального дерева прийняття рішень.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

4.1.1.1 Головне меню гри (рисунок 4.1) з наступними елементами:

– Кнопка переходу до меню вибору рівнів (рисунок 4.2).
– Кнопка переходу до редактора рівнів (рисунок 4.7).
– Кнопка відкриття меню довідкової інформації (рисунок 4.9).
– Вихід з гри.

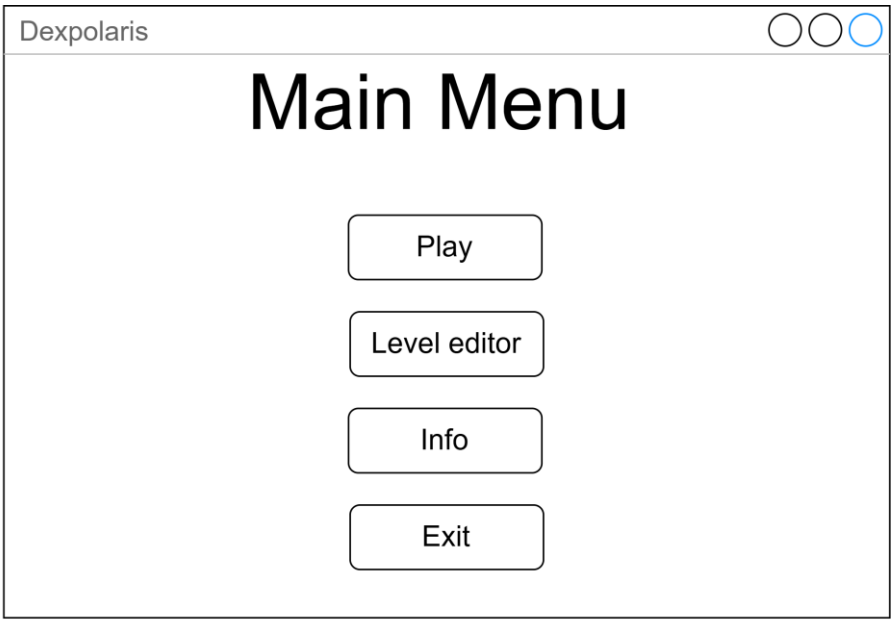


Рисунок 4.1 – Головне меню гри

4.1.1.2 Меню вибору рівнів (рисунок 4.2) з наступними елементами:

– Кнопки вибору одного з рівнів стандартної кампанії
– Кнопки вибору одного з трьох користувацьких рівнів
– Кнопка повернення до головного меню гри (Рисунок 4.1)

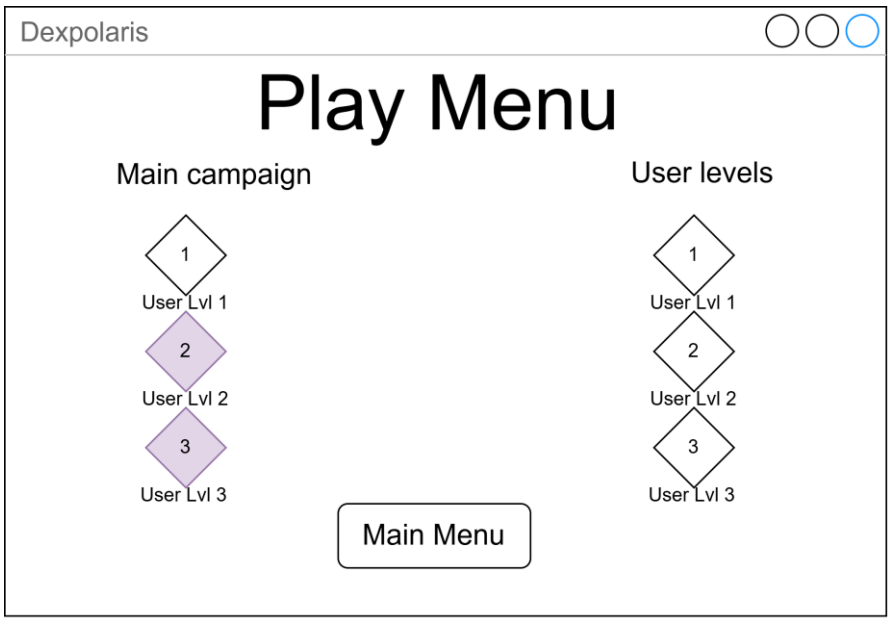


Рисунок 4.2 – Меню вибору рівнів

4.1.1.3 Вікно ігрового процесу (рисунок 4.3) з наступними елементами:

– Підрахунок ігрового рахунку

– Відображення часу, який залишився до кінця рівня

– Візуальне відображення типу озброєння

– Спільний індикатор для ємності магазину, загальної кількості патронів для обраного типу озброєння

– Відсотковий індикатор кількості здоров’я головного персонажу

– Перехід до меню паузи (Рисунок 4.4) при натисканні клавіші “ESC”

– Перехід до меню успішного завершення рівня (Рисунок 4.5) або до меню програшу (Рисунок 4.6) у відповідності до дій гравця.

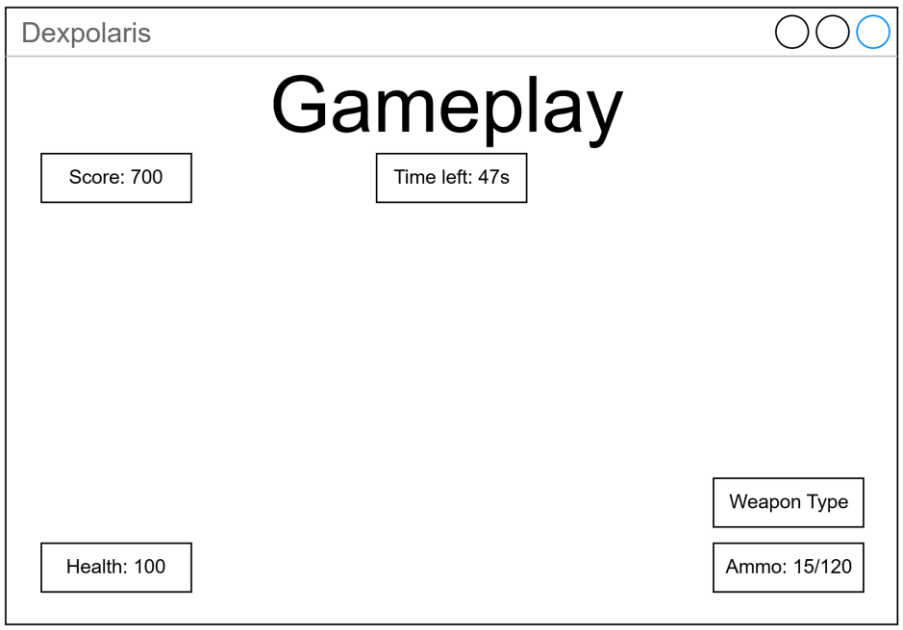


Рисунок 4.3 – Вікно ігрового процесу

4.1.1.4 Меню паузи (рисунок 4.4) з наступними елементами:

– Кнопка повернення до ігрового процесу (Рисунок 4.3)
– Кнопка перезапуску рівня (Рисунок 4.3)
– Кнопка повернення до головного меню (Рисунок 4.1)

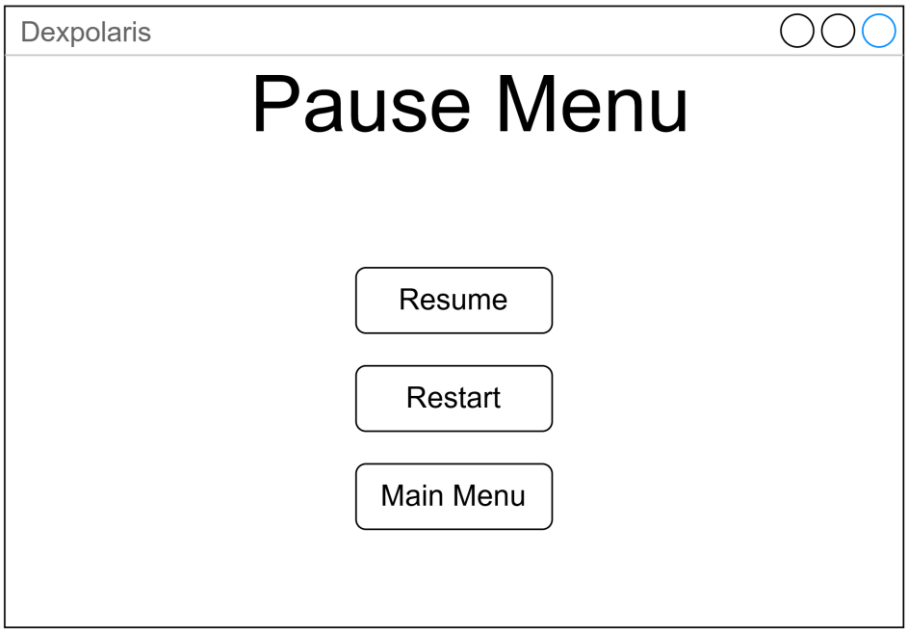


Рисунок 4.4 – Вікно паузи

4.1.1.5 Меню успішного завершення рівня (рисунок 4.5) з наступними елементами:

– Фінальний ігровий рахунок за рівень
– Час, за який було завершено рівень
– Кнопка повернення до головного меню (Рисунок 4.1)

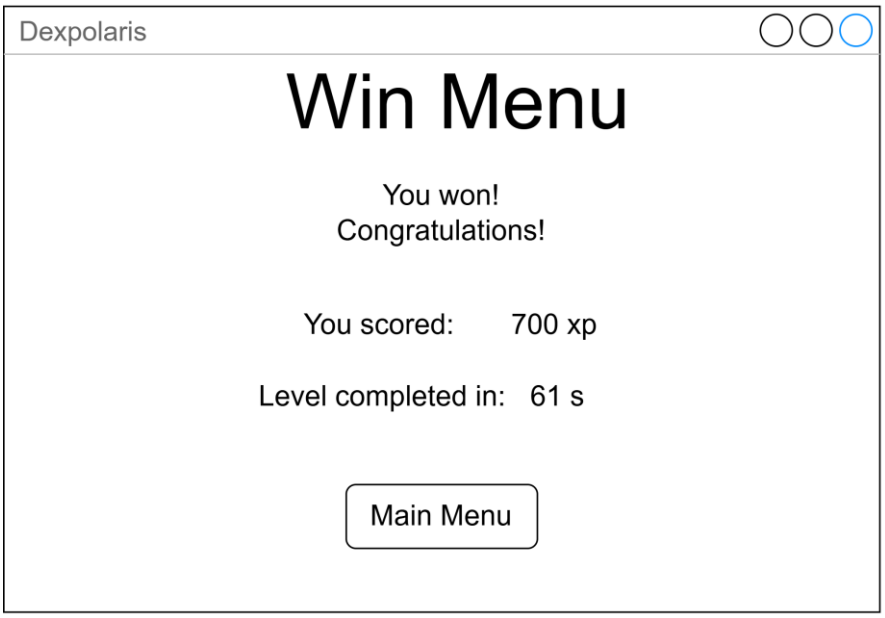


Рисунок 4.5 – Меню успішного завершення рівня

4.1.1.6 Меню програшу (рисунок 4.6) з наступними елементами:

– Фінальний ігровий рахунок за прожите життя
– Час, який було проведено на рівні
– Кнопка перезапуску рівня (Рисунок 4.3)
– Кнопка повернення до головного меню (Рисунок 4.1)

Рисунок 4.6 – Меню програшу

4.1.1.7 Меню редактору рівнів (рисунок 4.7) з наступними елементами:

– Десять кнопок з об’єктами для розташування, з підсвіткою активного обраного об’єкту

– Текстове поле з виведенням назви обраного об’єкту

– Текстове поле з виведенням помилок

– Перехід до меню збереження рівня (Рисунок 4.8) при натисканні клавіші “ESC”

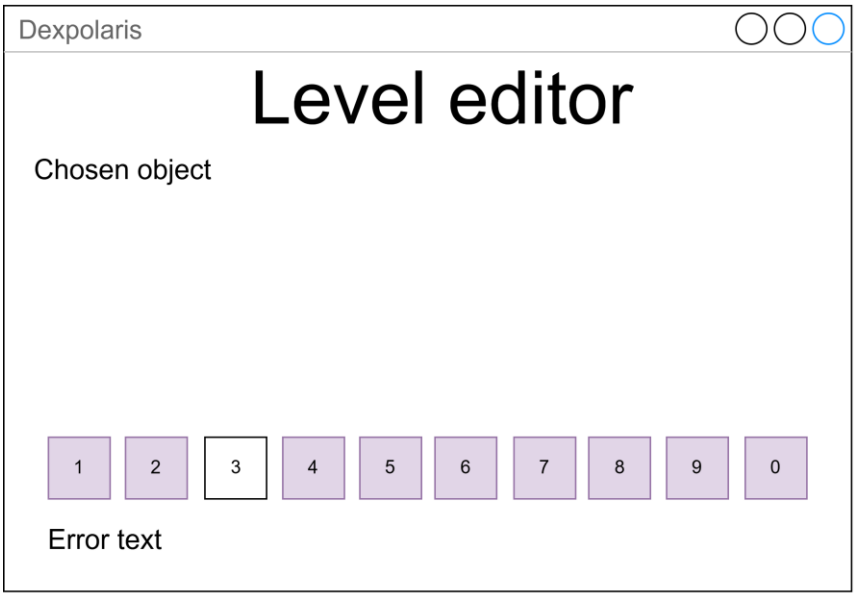


Рисунок 4.7 – Меню редактору рівнів

4.1.1.8 Меню збереження рівнів (рисунок 4.8) з наступними елементами:

– Кнопка видалення всіх об’єктів на рівні
– Кнопка повернення до головного меню (Рисунок 4.1)
– Повзунок налаштування часу проходження рівня
– Три кнопки завантаження рівня
– Три кнопки збереження рівня

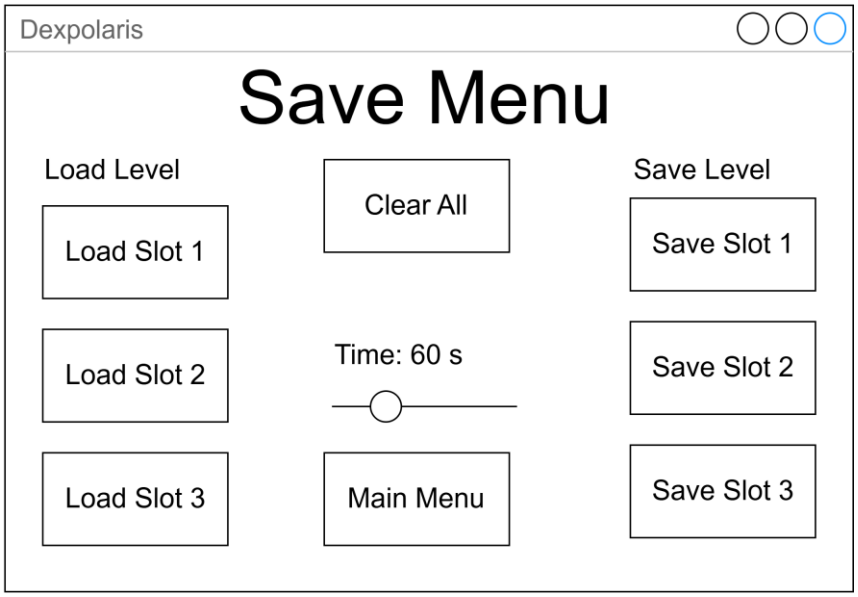


Рисунок 4.8 – збереження створення рівнів

4.1.1.9 Меню довідкової інформації (рисунок 4.9) з наступними елементами:

– Текстове поле з інформацією про гру
– Кнопка повернення до головного меню (Рисунок 4.1)

Рисунок 4.9 – Меню довідкової інформації

4.1.2 Ігровий процес

– Поява персонажа на ігровому рівні, на стартовому метеориті зі стартовою зброєю – лазерний пістолет-кулемет та запасом набоїв у кількості одного магазину до цього виду озброєння – 32 лазерних патронів.

– Послідовне створення літаючих ворогів зі спавнерів (місця, з якого з’являються вороги) доти, доки ці спавнери не будуть знищені. Обмежити максимальну кількість ворогів, яку може створювати за раз один спавнер до п’яти. Якщо гравець знищує ворогів, то дозволити спавнеру далі створювати ворогів у відповідності до ліміту.

– Ведення підрахунку зібраних гравцем очок, які збільшуються при знищенні ворогів. Різні типи ворогів повинні приносити різну кількість очок. Літаючі вороги (НЛО) повинні давати 100 очок, стаціонарні (лазерні турелі) 50 очок при знищенні. Спавнери НЛО повинні давати 150 очок при знищенні.

– Можливість призупинити ігровий процес, натиснувши клавішу “ESC”. Можливість перезапустити рівень з початку з меню паузи. Можливість вийти у головне меню з меню паузи.

– Успішне завершення рівня при вчасному потраплянні на кінцевий метеорит. Програш при наставанні стану 0 одиниць здоров’я, яке настає при отриманні шкоди від ворогів, або при падінні з метеоритів до чорної діри. Також програш настає при вичерпанні часу, відведеного на проходження рівня. Час на проходження рівня варіюється від 60 до 180 секунд, в залежності від рівня.

– Можливість підбирання та зміни типів ігрового озброєння. Типів озброєння два: лазерний пістолет-кулемет та плазменна рушниця.

4.1.3 Притаманні ігрові механіки

– Переміщення персонажу по різних метеоритах, використовуючи стрибки, рух по метеоритах з можливістю використання горизонтального прискорення гравця, яке працює як на поверхні, так і у повітрі.

– Реакція на потрапляння ворожого лазера в гравця. При цьому кількість здоров’я повинна зменшитися на 5 одиниць при потраплянні лазеру від НЛО та на 10 одиниць при потраплянні у гравця лазеру від стаціонарної турелі. Лазер від НЛО повинен бути зеленого кольору, а лазер від стаціонарної турелі – рожевого.

– Наявність варіативності у можливих шляхах потрапляння до кінцевого метеориту, забезпечуючи просторову навігацію на рівні.

4.1.4 Проходження кампанії

– Гравцю доступні на вибір 3 рівні кампанії, які створені зарані

– При першому запуску гри, гравцю доступний лиш перший рівень кампанії. Інші рівні відкриваються послідовно з успішним завершенням попередніх

4.1.5 Створення користувацьких рівнів

– Перегляд розміщених об’єктів на рівні

– Розміщення об’єктів на ігровому рівні за допомогою натискання лівої кнопки миші по вільній ділянці.

– Видалення вже розміщених об’єктів на рівні при натисканні по ним правою кнопкою миші

– Встановлення часу, відведеного на проходження одного рівня, який складає від 60 до 180 секунд

– Розміщення озброєння, амуніції та аптечок на ігровому рівні

– Розміщення стаціонарних ворогів, спавнерів літаючих ворогів на ігровому рівні. Відображення радіусу дії ворогів при їх розміщенні.

– Обмеження у створенні початкових та кінцевих метеоритів. Ці об’єкти повинні мати лише по одному екземпляру на ігровий рівень. Не дозволяти зберігати рівень, якщо на ньому немає початкового та/або кінцевого метеорита.

– Збереження збереження створеного ігрового рівня

– Можливість очистити всі об’єкти з ігрового рівня

– Можливість проходження створених ігрових рівнів

– Перевіряти ігровий рівень на відсутність колізій при розміщенні нових елементів.

– Якщо об’єкти розміщуються одне на одного, тобто створюють колізії, не дозволяти розміщувати останній розміщений об’єкт, виділивши цей об’єкт червоним кольором.

4.1.6 Система збору бонусів

– Гравець на рівні може підібрати аптечки, які збільшують його здоров’я на 25 одиниць. Максимальне здоров’я гравця складає 100 одиниць. Початкове здоров’я гравця складає 100 одиниць.

– Гравець на рівні може підібрати новий тип озброєння. При підбиранні озброєння, минуле озброєння залишається в інвентарі. Нове озброєння містить у собі запас амуніції, еквівалентний запасу патронів в одному магазині. Для лазерного пістолету-кулемету це 32 одиниці, для плазменної рушниці – 2 одиниці набоїв.

– Гравець на рівні може підібрати амуніцію до різного типу озброєння. При підбиранні набоїв до лазерного пістолета-кулемета, гравцю надається 120 лазерних патронів, при підбиранні набоїв до плазменної рушниці, гравцю надається 16 плазменних патронів. Максимальний запас набоїв складає 320 одиниць для лазерного пістолета-кулемета та 64 для плазменної рушниці.

4.2 Вимоги до надійності

Забезпечення стабільної роботи ігрового застосунку та ігрового процесу при заданих системних вимогах.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на персональних комп’ютерах, сумісних з операційною системою сімейства Windows.

Мінімальна конфігурація технічних засобів:

- ОС: Windows 10 (64 Bit);
- об‘єм ОЗП: 4 Гб;
- процесор: Intel Core 2 Duo;

Рекомендована конфігурація технічних засобів:

- ОС: Windows 11 (64 Bit);
- об‘єм ОЗП: 8 Гб;
- процесор: Intel Core i3;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C# з використанням інструментів для розробки на ігровому рушії Unity.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Unity Editor версії 2021.3.30f1, Visual Studio версії 16.11.46.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на платформі GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– технічне завдання;
– пояснювальна записка;
– текст програми;
– програма та методика тестування;
– керівництво користувача;

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна класів програмного забезпечення;
– креслення вигляду екранних форм;

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення рекомендованої літератури	16.03.2025	
2.	Аналіз існуючих методів розв’язання задачі	02.04.2025	
3.	Постановка та формалізація задачі	10.04.2025	
4.	Розробка інформаційного забезпечення	16.04.2025	
5.	Алгоритмізація задачі	21.04.2025	
6.	Обґрунтування вибору використаних технічних засобів	22.04.2025	
7.	Розробка програмного забезпечення	09.05.2025	
8.	Налагодження програми	11.05.2025	
9.	Виконання графічних документів	13.05.2025	
10.	Оформлення пояснювальної записки	17.05.2025	
11.	Подання ДП на попередній захист	02.06.2025	
12.	Подання ДП рецензенту	10.06.2025	
13.	Подання ДП на основний захист	14.06.2025	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Гра у жанрі 3D Action з елементами просторової навігації**

КПІ.ІП-1311.045490.02.81

Київ - 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ	–	Штучний інтелект
Spawner	–	Спавнер - елемент гри, який викликає появу інших дочірних об’єктів: ворогів, предметів тощо.
IDE	–	Integrated Development Environment - інтегроване середовище розробки.
Action	–	Жанр відеоігор, який вимагає від гравця певних фізичних зусиль, таких як швидкості реакції та здібності швидко приймати тактичні рішення.
FPS	–	Frames Per Seconds - кількість кадрів на секунду.
RAM	–	Random Access Memory - пам’ять з довільним доступом.
CPU	–	Central processing unit, центральний процесор комп’ютера, функціональна частина комп'ютера, що призначена для інтерпретації команд
Unity	–	Багатоплатформовий інструмент для розроблення відеоігор і застосунків, і рушій, на якому вони працюють.

ВСТУП

Дипломний проєкт присвячений розробці гри в жанрі 3D Action з елементами просторової навігації. Актуальність даної розробки зумовлена стрімким розвитком індустрії відеоігор у XXI столітті, яка перетворилася на глобальне культурне та високоприбуткове явище [1]. Жанр 3D Action посідає значне місце на ринку, пропонуючи гравцям занурення у віртуальні світи. Однак, сучасні представники цього жанру часто стикаються з такими проблемами, як обмежена свобода переміщення гравця, що знижує варіативність проходження, та використання надто простих алгоритмів штучного інтелекту для ворогів, що робить їхню поведінку передбачуваною і швидко набридає гравцям, зменшуючи мотивацію до повторного проходження. Вирішення цих проблем є актуальним завданням для підвищення якості та привабливості ігрових продуктів.

Провідні тенденції розв’язання поставлених проблем на сучасному ринку відеоігор включають надання гравцям більшої свободи дій та навігації, впровадження редакторів рівнів [2] для користувачів та вдосконалення штучного інтелекту противників із застосуванням складніших алгоритмів.

Призначенням розробки є забезпечення ігрового процесу гри в жанрі 3D Action з елементами просторової навігації для широкого кола користувачів. Метою дипломного проєкту є підвищення якості гри за рахунок створення штучного інтелекту, що формалізує поведінку противників на основі оригінального дерева прийняття рішень.

Можливими сферами застосування даної розробки є розважальна індустрія, зокрема ринок комп'ютерних ігор. Гра призначена для людей, які зацікавлені у грі на персональних комп’ютерах.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Дипломне проєктування передбачає виконання наступних завдань:

– аналіз предметної області та опис її ключових бізнес-процесів, визначення загального завдання розробки у рамках ДП;

– аналіз існуючих рішень (як програмних продуктів, так і алгоритмічних чи технічних рішень) обраного завдання розробки у рамках ДП;

– аналіз та моделювання бізнес-процесів;

– розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;

– аналіз економічних показників програмного забезпечення ДП;

– постановка завдання на розробку програмного забезпечення ДП;

– розроблення архітектури програмного забезпечення;

– розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;

– конструювання та розроблення програмного забезпечення;

– аналіз якості та тестування програмного забезпечення;

– розгортання та супровід програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

В двадцять першому столітті, індустрія відеоігор стала однією з найбільш динамічних та найбільш прибуткових сфер інформаційних технологія. За менш ніж сто років своєї історії, відеоігри перетворилися з нішового захоплення на глобальне культурне явище [3].

Станом на 2024 рік, у світі налічується понад 3.4 мільярди гравців у відеоігри, серед яких приблизно 900 мільйонів припадає саме на гравців у

відеоігри на персональних комп’ютерах. Лише за минулий рік ринок відеоігор приніс дохід у 187.7 мільярдів доларів [4].

Особливу нішу у галузі відеоігор займають ігри жанру Action, а саме їх тривимірні представники. Цей жанр дозволяє відображувати гравцю як реалістичні, так і фантастичні середовища, з якими можна взаємодіяти напряму за допомогою комп’ютера. Від того, наскільки гравець може керувати своїм персонажем, орієнтуватися у просторі, побудованому розробниками, залежить його ступінь залученості до ігрового процесу та кількість отриманих вражень.

Відеоігри стали чудовим інструментом для розробників задля передавання певної історії, яка буде резонувати з гравцем, який буквально проживає всі ті емоції, які відчуває персонаж. Але інколи відеоігри, особливо жанру Action, надто сильно обмежують гравця, вибудовуючи йому настільки прямий шлях проходження рівнів, що не залишається простору для тактики або уяви. Ці відеоігри скоріше передають емоції від красивого кіно, ніж від відеоігор, і від цього може втрачатися інтерес з боку гравців [5]. Рідко ігри жанру Action надають гравцям потрібну свободу не лише у прийнятті власних рішень, а й у створенні власних ігрових ситуацій.

Часто ігри складаються з сюжетної кампанії, після якої гравець може більше ніколи й не повернутися до гри. Всі рівні вже пройдені, всі ігрові ситуації пережиті, а варіативності в цих іграх майже немає, тому й відпадає бажання проводити більше часу в них. Вороги поводять себе майже однаково, оскільки їх поведінка описана простим кодом з примітивною логікою: побачив гравця - йди до нього, підійшов ще ближче - атакуй. Через таку просту поведінку, дії ворогів стають надто передбачуваними для гравця, й швидко набридають. Ігровий світ не відчувається живим, а за ворогами не видно нічого крім просто алгоритму [6].

Проблеми з сучасним ринком відеоігор у жанрі Action можна вирішити наступним чином: надавати гравцю більшої варіативності проходження рівнів, створювати ігрові рівні за допомогою процедурної генерації, надавати гравцю

можливість створювати та проходити власні рівні, покращити штучний інтелект ворогів у грі, дозволити гравцю боротися з іншими реальними гравцями.

У даному дипломному проєкті передбачено поєднати три ключові підходи для покращення ситуації у області ігор жанру Action. По-перше, система просторової навігації по рівню буде організована так, щоб дати гравцю максимальну свободу у проходженні рівня, забезпечуючи варіативність проходження. Фінальний астероїд буде виднітися одразу на початку рівня, підсвічуватися для простоти находження, але не буде виокремлено одного визначеного шляху для його досягання. Натомість, гравець буде сам обирати для себе маршрут проходження рівня, оцінювати ризики переміщення по різним платформам, наявність на шляху певного типу ворогів, перешкод, тощо. Такий ігровий процес забезпечить максимальне залучення гравця до проходження рівня, а також підсилить варіативність проходження, що може потенційно позитивно вплинути на вірогідність повернення гравця до гри.

По-друге, у дипломному проєкті передбачено створення оригінального штучного інтелекту ворогів на основі дерева прийняття рішень. Це дозволить ворогам реагувати не лише на присутність гравця на ігровому полі, а і на інші умови ведення бою, що додасть непередбачуваності до ігрового процесу й зробить взаємодію з ворогами більш динамічнішою. В результаті цього, поведінка ворогів з класичного «побачив - атакуй» перетвориться на щось більш «живе».

По-третє, у дипломному проєкті передбачено можливість створення власних ігрових рівнів за допомогою редактора рівнів. Використовуючи редактор рівнів, гравець може створити власний рівень зі своєю системою розгалужень, ворогами, типами озброєння. Таким чином, навіть після проходження усієї кампанії гри, у гравця буде стимул повертатися до гри знову й знову, паралельно підвищуючи свою креативність.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації гри у жанрі 3D Action з елементами просторової навігації. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

Для порівняння з дипломним проєктом, розглянемо два аналоги:

ClusterTruck - 3D-платформер від першої особи з акцентом на швидкісну просторову навігацію та уникнення перешкод («підлога - це лава») [7]. Ігровий процес гри показано на рисунку 1.1.

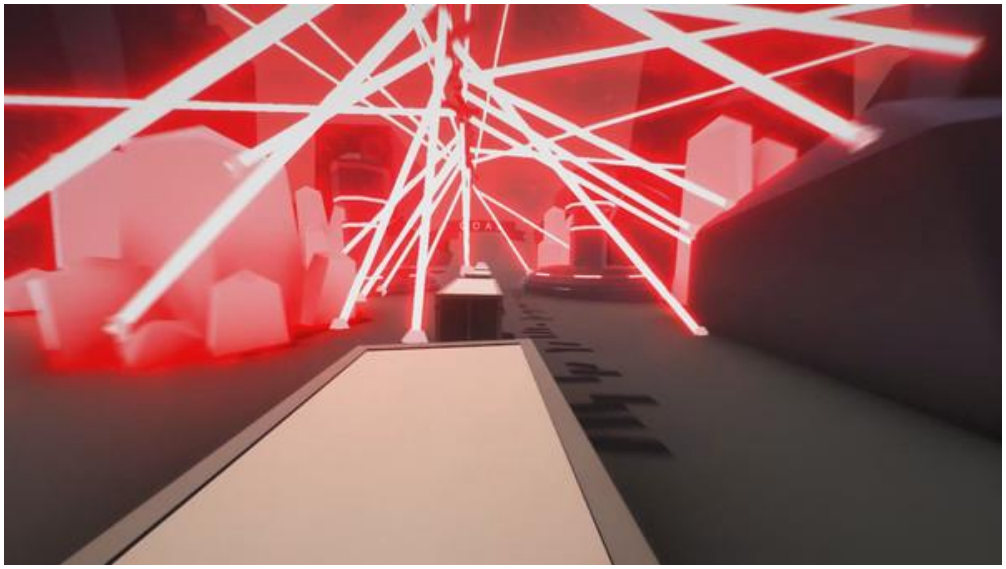


Рисунок 1.1 – Ігровий процес гри «Clustertruck» [8]

Roboquest - Швидкий шутер від першої особи з акцентом на рухливість, стрільбу та збір покращень [9]. Ігровий процес гри показано на рисунку 1.2.

Рисунок 1.2 – Ігровий процес гри «Roboquest» [10]

Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Критерії	Дипломний проєкт (Dexpolaris)	ClusterTruck	Roboquest	Пояснення
Наявність кампанії	+	+	-	Dexpolaris та ClusterTruck мають рівні кампанії, в той час як Roboquest не має попередньо створених рівнів кампанії
Наявність просторової навігації по рівню	+	+	+	Всі три гри забезпечують варіативність у проходженні рівнів, забезпечуючи високу мобільність гравця на рівні

Продовження таблиці 1.1

Критерії	Дипломний проєкт (Dexpolaris)	ClusterTruck	Roboquest	Пояснення
Наявність редактора рівнів	+	+	-	Як Dexpolaris, так і ClusterTruck надають можливість створення власних рівнів, а Roboquest - ні
Наявність ворогів з продвинутим штучним інтелектом	+	-	-	Dexpolaris реалізує поведінку ворогів з продвинутим штучним інтелектом, формалізучи поведінку противників на основі оригінального дерева прийняття рішень. ClusterTruck та Roboquest реалізують ШІ на осонові машини скінченних станів
Наявність процедурної генерації рівнів	-	-	+	Roboquest має процедурну генерацію рівнів, а Dexpolaris та ClusterTruck - ні
Наявність системи збору бонусів	+	+	+	Всі три гри мають наявні системи збору бонусів на рівнях (зброю, здоров’я, інше)

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Проведемо порівняльний аналіз відомих алгоритмів штучного інтелекту у відеоіграх з метою обгрунтування вибору найбільш доцільного алгоритму для розробки програмного відеоігри для дипломного проєкту.

Серед найпоширеніших алгоритмів ШІ у відеоіграх виділяють наступні алгоритми:

Машина скінченних станів (англійською Finite State Machine) - це один з найстаріших і найпростіших у реалізації підходів [11]. Поведінка ворогів визначається скінченною кількістю визначених станів, які перемикаються в залежності від певних умов. Цей алгоритм легкий у реалізації, але він стає малоефективним при збільшенні складності логіки, оскільки кількість станів ворогів і переходів між ними зростає експоненційно, що значно ускладнює масштабування цього ШІ.

Дерево прийняття рішень (англійською Decision Trees) - це ієрархічна структура, де кожна вершина представляє умову, а гілки дерева - можливі варіанти дій в залежності від результату перевірки [12]. На відмінно від машини скінченних станів, дерева прийняття рішень дозволяють описувати складні сценарії поведінки ворогів у більш контрольований та логічно структурований спосіб. Цей алгоритм проходить дерево зверху донизу, обираючи потрібну гілку, поки не дійде до певної потрібної дії. Це забезпечує кращу масштабованість завдяки гнучкості й зручності модифікації, а також робить поведінку ворогів більш комплексною, а отже цікавою для гравця.

Поведінкові дерева (англійською - Behavior Trees) - це розширення ідеї машини скінченних станів та дерева прийняття рішень [13]. Поведінкові дерева використовують вузли контролю для побудови гнучких шаблонів поведінки, які можна потім повторно використовувати. Цей алгоритм ШІ поширений у великих ігрових проєктах, але потребує налаштування великої кількості вузлів, що може услкаднити його реалізацію та впровадження у гру.

Планування дій (англійською GOAP - Goal-Oriented Action Planning) - це алгоритм ШІ, який грунтується на побудові плану дій задля досягнення певної заданої мети [14]. ШІ сам вирішує послідовність потрібних для виконання дій заради наближення до мети. Цей алгоритм хоч і демонструє доволі «розумну» поведінку, але має високу обчислювальну складність і

демонструє мало передбачувану поведінку, що може надто ускладнити ігровий процес для гравця.

Навчання з підкріпленням (англійською - Reinforcement Learning) - це тип машинного навчання, в якому агент вчиться, отримуючи винагороду за виконані успішні дії [15]. Хоч цей алгоритм і є перспективним серед алгоритмів ШІ у відеоіграх, але він рідко використовується через складність навчання, значну кількість ресурсів, потрібних для навчання, та відсутність прямого контролю з боку розробника над процесом навчання.

У межах дипломного проєкту було розглянуто низку технічних рішень для найкращої реалізації відеогри у жанрі 3D Action з елементами просторової навігації. Першочергово було проаналізовано наявні підходи до побудови прогрманого забезпечення, зокрема модельну архітектуру, модель Entity-Component-System, а також класичну об’єктно-орієнтовану архітектуру, яку активно використовують при розробці відеоігор [16].

Модельна архітектура передбачає поділ програмного забезпечення на ізольовані модулі, кожен з яких відповідає за певну функцію, а саме фізику, інтерфейс, ШІ, обробка подій тощо. Кожен модуль взаємодіє з іншими за допомогою визначених інтерфейсів, що забезпечує кращу підтримку проєкту. Серед переваг цієї архітектури є масштабованість, можливість незалежного тестування модулів та полегшення командної роботи. Але є і суттєві недоліки: складне керування залежностями між модулями, потреба у точному проєктуванні інтерфейсів ще на ранніх етапах, надмірна фрагментація коду. Для дипломного проєкту модульна архітектура виявилася надлишковою за рівнем складності.

Модель Entity-Component-System (скорочено ECS) є сучасним архітектурним підходом, що активно використовується у багатьох іграх. Ця модель базується на принципі розділення логіки на три основні сутності: самі сутності (Entities), які є просто ідентифікаторами, компоненти (Components), які зберігають дані, та системи (Systems), що реалізують логіку. ECS

забезпечує високу продуктивність завдяки кеш-ефективному зберіганню даних і паралельному використанню систем. З переваг цього рішення є висока продуктивність, просте масштабування та чітке розділення даних і логіки. З недоліків - високий поріг входження, складність налагодженні, відсутність прямої підтримки цього методу у багатьох інструментах розробки. Враховуючи, що реалізація ECS потребує значних зусиль, які не є доцільними у проєктах середнього рівня складності, а також те, що цей метод підтримується не усіма рушіями розробки відеоігор, було вирішено не використовувати цей метод у дипломному проєкті.

Об’єктно-орієнтована архітектура (скорочено ООА) - це класичний і добре підтримуваний підхід у розробці відеоігор. Він дозволяє представити ігрові об’єкти як об’єкти певних класів з притаманними їм методами та полями.

У процесі підготовки до розробки відеогри було проаналізовано декілька популярних ігрових рушіїв, серед яких Unity, Unreal Engine 5 та Godot Engine [17]. Метою аналізу було визначення найбільш оптимального середовища для розробки гри середнього рівня складності з високою гнучкістю, широким функціоналом і підтримкою 3D графіки у середовищі розробки.

1.4 Аналіз та моделювання бізнес-процесів

Проаналізувавши відеогру, можна сказати, що двома основними бізнес-процесами гри є її безпосередній ігровий процес та процес створення рівнів.

Для моделювання бізнес-процесу ігрового процесу використовується BPMN модель (рисунок 1.3).

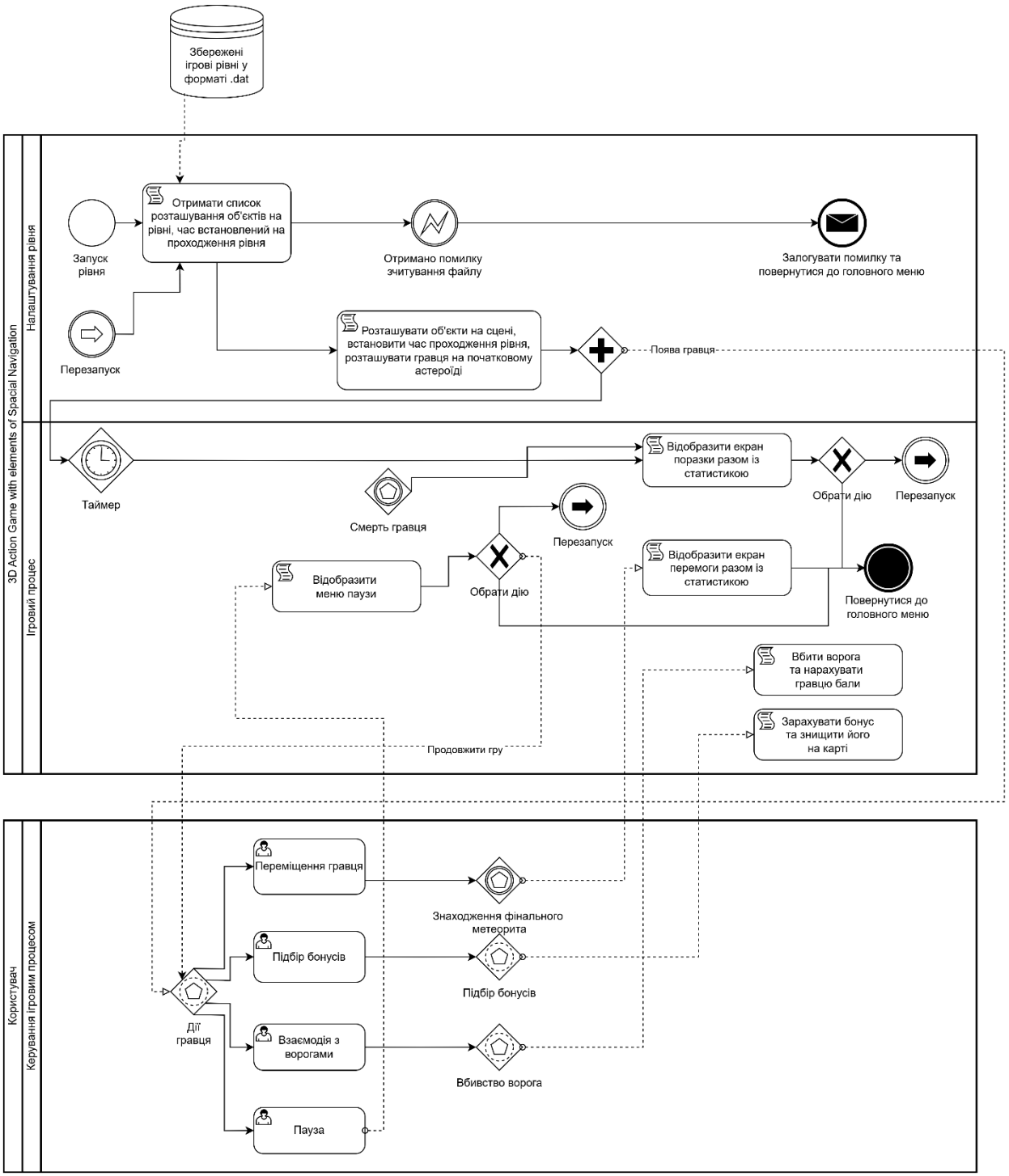


Рисунок 1.3 – Модель бізнес-процесу ігрового процесу

Бізнес-процес ігрового процесу починається із завантаження рівня. Цей етап включає в себе спробу завантаження даних рівня з файлу формату .dat. У випадку успіху процес переходить до налаштування рівня. Якщо завантаження не вдається, система логує помилку.

Після успішного завантаження рівня відбувається його налаштування. Це включає встановлення таймера рівня, розміщення всіх необхідних ігрових

об'єктів на карті та визначення початкової позиції гравця на стартовому астероїді.

Наступним ключовим етапом є безпосередньо ігровий процес. Основні дії гравця включають:

– переміщення: гравець може вільно пересуватися по рівню, переструбуючи по астероїдам (тим самим забезпечуючи просторову навігацію).

– взаємодія з ворогами: гравець може боротися з ворогами. При успішному знищенні ворога, гравцю нараховуються бали. Гравець також може отримати шкоду від ворогів.

– підбір бонусів: на рівні можуть бути розташовані бонуси, такі як аптечки, нова зброя або додаткова амуніція до зброї.

– смерть гравця: гравець може померти від отриманої шкоди від ворогів або від падіння з астероїда у чорну діру.

– пауза: у будь-який момент ігрового процесу гравець може поставити гру на паузу.

Під час ігрового процесу постійно працює таймер рівня. Якщо час на проходження рівня вичерпано, гравець програє. Рівень вважається успішно завершеним, якщо гравець вчасно дістається до кінцевого астероїда.

Ігровий процес може бути перерваний станом паузи, поразки або перемоги:

– меню паузи: доступне під час гри. З меню паузи гравець може повернутися до поточної гри, перезапустити рівень або вийти до головного меню.

– меню поразки: показується гравцю у випадку смерті або закінчення часу. З цього меню гравець може перезапустити рівень або повернутися до головного меню

– Меню перемоги: Показується гравцю після успішного завершення рівня. З цього меню гравець може лише повернутися до головного меню.

Після виходу до головного меню з меню паузи, поразки чи перемоги, поточний ігровий процес для даного рівня завершується.

Для моделювання бізнес-процесу редактора рівня використовується BPMN модель (рисунок 1.4).

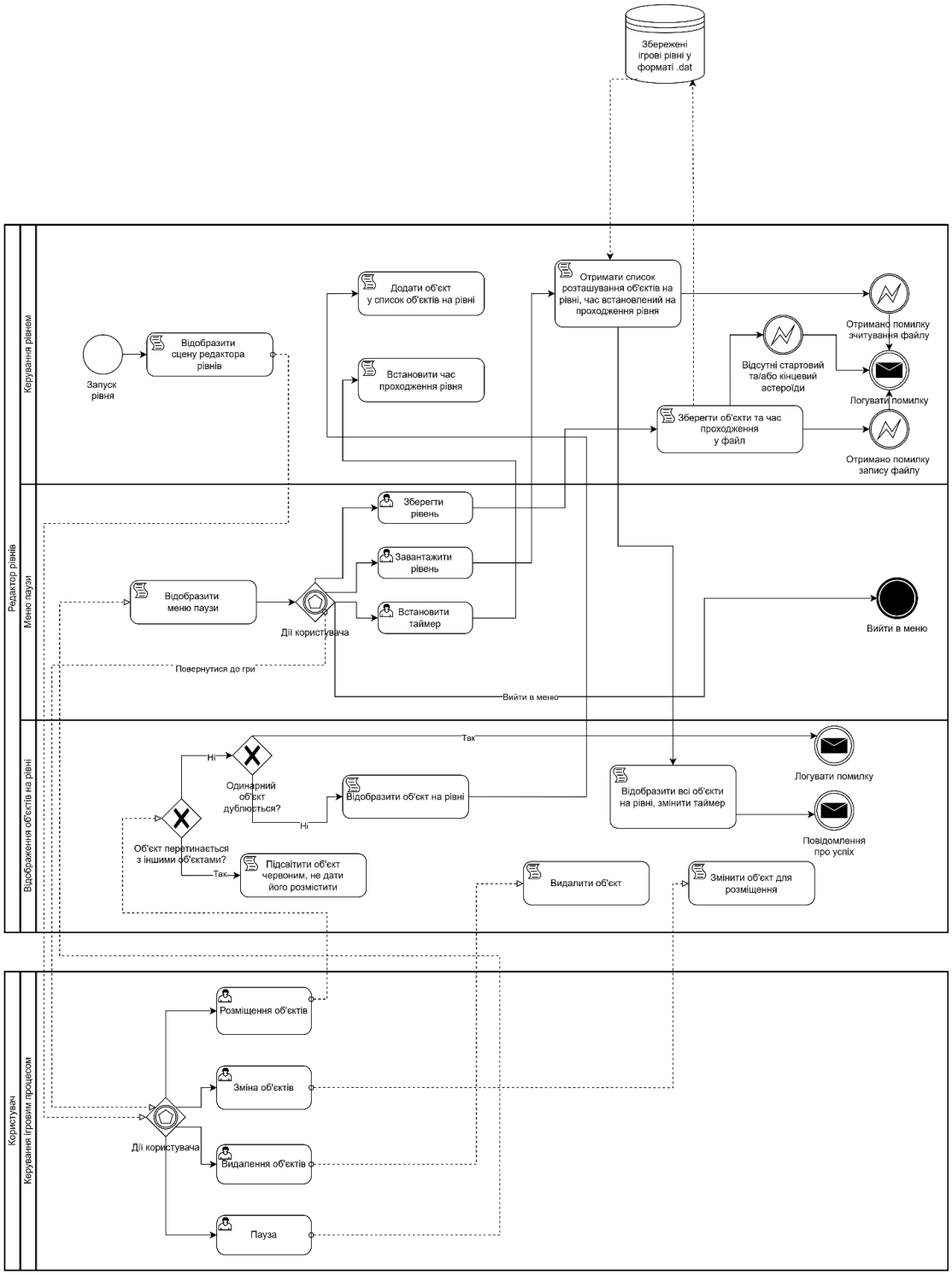


Рисунок 1.4 – Модель бізнес-процесу редактора рівнів

Процес редагування рівня починається з того, що гравець переходить до редактора рівня з головного меню гри. Це основний інтерфейс для роботи з рівнем. Тут гравець може виконувати наступні дії:

– розміщувати об'єкти: гравець обирає тип об'єкта (наприклад, початковий чи кінцевий астероїд, ворогів, бонуси) з меню вибору об'єктів та розміщує його на рівні. При розміщенні відбувається перевірка на колізії. Якщо об'єкт перетинається з іншим, розміщення забороняється, і об'єкт для розміщення виділяється червоним кольором. Також є обмеження на кількість об'єктів певного типу (початковий та кінцевий астероїди мусять бути лише одними на рівні).

– видаляти об'єкти: гравець може видаляти розміщені на рівні об'єкти.

– налаштовувати час рівня: гравець може встановити час, відведений на проходження даного рівня, в межах певного діапазону (від 60 до 180 секунд).

– зберігати рівень: гравець може зберегти рівень у обраному слоті збереження. Під час збереження відбувається перевірка на наявність необхідних об'єктів (початкового та кінцевого астероїда). Якщо рівень не відповідає вимогам, збереження стає неможливим. У разі помилки при збереженні, вона логується користувачу.

– очистити рівень: гравець може повністю очистити рівень від усіх розміщених об'єктів.

– поставити редактор на паузу: гравець може викликати меню паузи в редакторі рівня.

– повернутися до головного меню: дозволяє гравцю вийти з редактора рівня до головного меню гри.

– якщо гравець завантажує рівень, і виникає помилка при завантаженні, вона також логується користувачу.

Процес редагування рівня завершується, коли гравець виходить до головного меню з меню паузи в редакторі.

Висновки до розділу

У першому розділі було проведено переддипломне обстеження предметної області, сформульовано постановку задачі дипломного проєкту. Проведено детальний аналіз сучасного стану ігрової індустрії, зокрема жанру Action. Виявлено ключові проблеми наявних рішень. Було запропоновано концепцію гри, що поєднує просторову навігацію, ШІ на основі дерева прийняття рішень та можливість створення власних рівнів. Здійснено порівняльний аналіз існуючих аналогів та обгрунтовано переваги запропонованої розробки. Було змодельовано та описано основні бізнес-процеси у вигляді BPMN-діаграм, які охоплюють ігровий процес і редагування рівнів.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання з ключовими акторами та основними функціями знаходиться у графічному матеріалі, креслення 1.

В таблицях 2.1-2.19 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Ігровий процес
Use case ID	UC-01
Goals	Дозволити гравцю брати участь в ігровому процесі, проходячи доступні рівні (кампанії або користувацькі).
Actors	Гравець
Trigger	Гравець обирає та запускає будь-який доступний ігровий рівень.
Pre-conditions	Гравець увійшов у меню «Play».
Flow of Events	1. Гравець обирає рівень. 2. Рівень завантажується, і розпочинається ігровий процес. 3. Гравець здійснює дії в грі (Навігація по рівню, Використання зброї, Підбір бонусів). 4. Гра триває до моменту досягнення умови завершення рівня (Перемога або Поразка).
Extension	1. Можливість Поставити гру на паузу. 2. Під час гри відбувається зворотній відлік часу, при досяганні якого гравець вмирає
Post-Condition	Рівень завершено з певним результатом: перемога або поразка. Перехід до екрану перемоги або поразки з відповідною статистикою проходження рівня.

Таблиця 2.2 – Варіант використання UC-02

Use case name	Створення власних рівнів
Use case ID	UC-02

Продовження таблиці 2.2

Goals	Надати гравцю можливість розробляти, змінювати та зберігати власні ігрові рівні за допомогою вбудованого редактора.
Actors	Гравець
Trigger	Гравець увійшов у меню «Level editor»
Pre-conditions	Гравець перебуває в активній ігровій сесії на будь-якому рівні.
Flow of Events	1. Гравець переходить у меню редактора 2. Гравець використовує інструменти для Розміщення об'єктів та Видалення об'єктів. 3. Гравець може Керувати збереженням рівнів (зберігати прогрес, завантажувати існуючі рівні). 4. Після завершення змін, гравець зберігає рівень.
Extension	1. При збереженні може знадобитися Підтвердження вибору дії. 2. Система може обмежувати дії або надавати підказки при некоректному редагуванні.
Post-Condition	Створено або оновлено файл власного ігрового рівня гравця.

Таблиця 2.3 – Варіант використання UC-03

Use case name	Навігація по рівню
Use case ID	UC-03
Goals	Дозволити гравцю переміщувати свого персонажа або оглядати ігрове оточення в межах рівня.
Actors	Гравець
Trigger	Гравець використовує пристрої введення для керування рухом персонажа або камерою під час гри на рівні.
Pre-conditions	Гравець грає на будь-якому з рівнів
Flow of Events	1. Гравець використовує клавіші для руху (вперед, назад, в сторони, стрибки, прискорення). 2. Система відповідно змінює позицію та стан ігрового персонажа на рівні. 3. Гравець використовує мишу для керування напрямком погляду (камерою). 4. Система відповідно змінює огляд гравця.
Extension	Гравець може впасти до чорної діри й загинути.

Продовження таблиці 2.3

Post-Condition	Позиція та/або напрямок огляду гравця змінені.

Таблиця 2.4 – Варіант використання UC-04

Use case name	Використання зброї
Use case ID	UC-04
Goals	Дозволити гравцю застосовувати обрану зброю для атаки ворогів або їх спавнерів
Actors	Гравець
Trigger	Гравець використовує поточну зброю (натискає ліву кнопку комп'ютерної миші).
Pre-conditions	Гравець перебуває на рівні, має зброю
Flow of Events	1. Гравець застосовує зброю. 2. Система обробляє цю дію згідно з характеристиками поточної зброї. 3. Якщо атака вражає ціль (ворога або спавнер), система обчислює результат (пошкодження або знищення). 4. У разі успіху проти ворога може відбутися Знищення ворогів, що може призвести до Нарахування бонусів.
Extension	1. Недостатньо боєприпасів. 2. Зброя на перезарядці або має затримку між пострілами. 3. Атака не влучила в ціль, тоді , відобразити слід від попадання кулі на перешкоді. 4. Гравець може виконати Зміну типів озброєння перед використанням іншої зброї.
Post-Condition	Зброя була використана, можливі зміни в стані цілі (зменшення здоров'я) та гравця (витрата боєприпасів).

Таблиця 2.5 – Варіант використання UC-05

Use case name	Підбір бонусів
Use case ID	UC-05
Goals	Дозволити гравцю підбирати бонусні предмети, знайдені на рівні.
Actors	Гравець
Trigger	Гравець наближається до бонусного предмета на рівні

Продовження таблиці 2.5

Pre-conditions	Гравець грає на рівні, на поточному рівні присутні бонусні предмети.
Flow of Events	1. Гравець контактує з бонусним предметом (проходить через нього). 2. Система реєструє факт підбору бонусу. 3. До стану гравця застосовується відповідний ефект від бонусу (відновлення здоров'я/додаткова амуніція). 4. Бонусний предмет, як правило, видаляється з ігрового світу.
Extension	1. Гравець вже має максимальний ефект від цього бонусу, і підбір ігнорується. 2. Гравець не має відповідного типу озброєння для цього бонусу, і підбір ігнорується
Post-Condition	Бонусний предмет підібрано, відповідний ефект застосовано до гравця, бонусний предмет більше не присутній на рівні.

Таблиця 2.6 – Варіант використання UC-06

Use case name	Завершення і відображення результатів рівня
Use case ID	UC-06
Goals	Відобразити гравцю результати проходження рівня (перемога або поразка) та відповідну статистику.
Actors	Гравець
Trigger	Ігровий рівень досяг умови завершення (перемога за досягненням фінального астероїду або поразка за часом/здоров'ям/падінням).
Pre-conditions	Гравець грає на рівні (в рамках UC-01). Рівень досяг умови завершення.
Flow of Events	1. Система виявляє, що рівень досяг умови завершення (перемога або поразка). 2. Ігровий процес на поточному рівні припиняється. 3. Система збирає статистичні дані проходження рівня (набрані очки, час проходження/виживання тощо). 4. Система визначає результат рівня (перемога або поразка). 5. Система відображає відповідний екран завершення рівня ("Меню успішного завершення рівня" або "Меню програшу"). 6. На екрані відображаються зібрані статистичні дані та результат рівня. 7. Гравець ознайомлюється з результатами та статистикою. 8. Гравець обирає подальшу дію

Продовження таблиці 2.6

Extension	Зарахування проходження рівня, якщо це рівень кампанії
Post-Condition	Ігровий рівень завершено. Гравцю відображено екран результатів зі статистикою. Гравець готовий до вибору подальших дій поза межами пройденого рівня.

Таблиця 2.7 – Варіант використання UC-07

Use case name	Гра на рівнях кампанії
Use case ID	UC-07
Goals	Дозволити гравцю пройти заздалегідь створені розробниками рівні в рамках сюжетної кампанії.
Actors	Гравець
Trigger	Гравець обирає один з доступних рівнів стандартної кампанії в меню вибору рівнів.
Pre-conditions	Гравець перебуває у меню вибору рівнів. Доступний хоча б один рівень кампанії.
Flow of Events	1. Гравець обирає рівень кампанії. 2. Система завантажує обраний рівень кампанії. 3. Розпочинається ігровий процес на вибраному рівні (див. UC-01).
Extension	Успішне проходження рівня кампанії може призвести до Відкриття нових рівнів кампанії, якщо вони не були відкриті раніше.
Post-Condition	Розпочато ігровий процес на вибраному рівні кампанії, або система повернулася до меню вибору рівнів у випадку помилки завантаження.

Таблиця 2.8 – Варіант використання UC-08

Use case name	Гра на користувацьких рівнях
Use case ID	UC-08
Goals	Дозволити гравцю зіграти на рівнях, створених ним самим або за допомогою редактора рівнів.
Actors	Гравець

Продовження таблиці 2.8

Trigger	Гравець обирає один з доступних користувацьких рівнів в меню вибору рівнів.
Pre-conditions	Гравець перебуває у меню вибору рівнів. Існують збережені користувацькі рівні.
Flow of Events	1. Гравець обирає користувацький рівень. 2. Система завантажує обраний користувацький рівень. 3. Розпочинається ігровий процес на вибраному рівні (див. UC-01).
Extension	Можлива помилка при завантаженні користувацького рівня, якщо файл рівня пошкоджений або відсутній.
Post-Condition	Розпочато ігровий процес на вибраному користувацькому рівні, або система повернулася до меню вибору рівнів у випадку помилки завантаження.

Таблиця 2.9 – Варіант використання UC-09

Use case name	Поставити гру на паузу
Use case ID	UC-09
Goals	Дозволити гравцю тимчасово призупинити ігровий процес.
Actors	Гравець
Trigger	Гравець натискає клавішу, призначену для виклику меню паузи "ESC" під час ігрового процесу.
Pre-conditions	Гравець перебуває на рівні (в рамках UC-01).
Flow of Events	1. Гравець ініціює дію паузи. 2. Ігровий процес призупиняється. 3. На екрані відображається меню паузи.
Extension	З меню паузи гравець може вибрати: повернутися до гри, перезапустити рівень або вийти до головного меню.
Post-Condition	Ігровий процес призупинено, відображається меню паузи.

Таблиця 2.10 – Варіант використання UC-10

Use case name	Розміщення об'єктів
Use case ID	UC-10

Продовження таблиці 2.10

Goals	Дозволити гравцю додавати ігрові об'єкти на створюваний або рівень.
Actors	Гравець
Trigger	Гравець обирає тип об'єкта в меню вибору об'єктів редактора та натискає ліву кнопку миші на вільній ділянці ігрового поля в редакторі.
Pre-conditions	Гравець перебуває в меню створення/редагування рівня (в рамках UC-02). Гравець обрав тип об'єкта для розміщення.
Flow of Events	1. Гравець вказує місце для розміщення об'єкта. 2. Система перевіряє можливість розміщення (наявність колізій, обмеження кількості). 3. Якщо розміщення можливе, об'єкт додається на рівень. 4. Якщо розміщення неможливе, система візуально повідомляє про помилку, підсвітивши об’єкт червоним.
Extension	Обмеження на кількість деяких об'єктів (наприклад, початкова та кінцева точки). Візуальна індикація (червоний колір) при неможливості розміщення.
Post-Condition	Об'єкт розміщено на рівні (або система повідомила про неможливість розміщення).

Таблиця 2.11 – Варіант використання UC-11

Use case name	Видалення об'єктів
Use case ID	UC-11
Goals	Дозволити гравцю видаляти ігрові об'єкти зі створюваного або редагованого рівня.
Actors	Гравець
Trigger	Гравець натискає праву кнопку миші на розміщеному об'єкті в редакторі рівнів.
Pre-conditions	Гравець перебуває в меню створення/редагування рівня (в рамках UC-02). На рівні присутні об'єкти.
Flow of Events	1. Гравець вказує на об'єкт для видалення. 2. Система видаляє обраний об'єкт з рівня.
Extension	-
Post-Condition	Обраний об'єкт видалено з рівня.

Таблиця 2.12 – Варіант використання UC-12

Use case name	Керування збереженням рівнів
Use case ID	UC-12
Goals	Дозволити гравцю зберігати поточний стан створюваного рівня, завантажувати раніше збережені рівні.
Actors	Гравець
Trigger	Гравець обирає зберегти рівень в один з трьох доступних слотів, або завантажити доступний рівень
Pre-conditions	Гравець перебуває в меню створення/редагування рівня (в рамках UC-02).
Flow of Events	1. Гравець натискає на кнопку збереження або завантаження файлу 2. Гравець підтверджує свою дію 3. Гравець зберігає або завантажує рівень
Extension	Помилка збереження через некоректність рівня. Помилка завантаження через відсутність або пошкодження файлу.
Post-Condition	Рівень збережено, завантажено відповідно до вибору гравця (або система повідомила про помилку)

Таблиця 2.13 – Варіант використання UC-13

Use case name	Очищення об’єктів
Use case ID	UC-13
Goals	Очистити всі об’єкти на рівні
Actors	Гравець
Trigger	Гравець натиснув на кнопку «Очистити всі об’єкти»
Pre-conditions	Гравець перебуває в меню створення/редагування рівня (в рамках UC-02).
Flow of Events	1. Гравець натискає на кнопку очищення рівня 2. За підтвердженням, рівень очищується
Extension	-
Post-Condition	Дія або виконана (якщо підтверджено), або скасована (якщо відмовлено).

Таблиця 2.14 – Варіант використання UC-14

Use case name	Збільшення ігрового рахунку
Use case ID	UC-14
Goals	Збільшити ігровий рахунок гравця як винагороду за знищення ворогів
Actors	Гравець
Trigger	Гравець знищує ворога або його спавнер на рівні.
Pre-conditions	Ігровий процес активний. На рівні присутні вороги або спавнери.
Flow of Events	1. Гравець виконує дію, що призводить до отримання бонусу (знищення ворога або його спавнера). 2. Система реєструє подію. 3. Система додає відповідну кількість очок до рахунку гравця
Extension	Різна кількість очок за різні типи ворогів.
Post-Condition	Ігровий рахунок гравця збільшено

Таблиця 2.15 – Варіант використання UC-15

Use case name	Знищення ворогів
Use case ID	UC-15
Goals	Дозволити гравцю усувати противників на ігровому рівні.
Actors	Гравець
Trigger	Здоров'я ворога опускається до нуля внаслідок атаки гравця.
Pre-conditions	Вороги присутні на рівні. Гравець атакує ворога (в рамках UC-04).
Flow of Events	1. Гравець завдає шкоди ворогу. 2. Система віднімає очки здоров'я ворога. 3. Якщо здоров'я ворога стає <= 0, ворог знищується. 4. Система нараховує бонуси гравцю за знищення ворога (див. UC-14).
Extension	Різні типи ворогів мають різне здоров'я та отримують різну шкоду від різної зброї. Анімація знищення ворога (вибух) відіграється після їх смерті.
Post-Condition	Ворога знищено та видалено з ігрового світу. Гравцю нараховано бонуси.

Таблиця 2.16 – Варіант використання UC-16

Use case name	Зміна типів озброєння
Use case ID	UC-16
Goals	Дозволити гравцю перемикатися між доступними типами зброї.
Actors	Гравець
Trigger	Гравець ініціює команду зміни зброї (використовуючи прогортання колеса комп'ютерної миші) або підбирає нову зброю.
Pre-conditions	Гравець має в інвентарі більше одного типу озброєння.
Flow of Events	1. Гравець ініціює зміну зброї. 2. Система змінює поточний активний тип озброєння гравця. 3. Візуальне відображення зброї та інтерфейсу оновлюється відповідно до нового типу зброї.
Extension	-
Post-Condition	Активний тип озброєння гравця змінено.

Таблиця 2.17 – Варіант використання UC-17

Use case name	Відкриття нових рівнів
Use case ID	UC-17
Goals	Дозволити гравцю отримати доступ до нових рівнів кампанії після успішного проходження попередніх.
Actors	Гравець
Trigger	Успішне завершення рівня кампанії.
Pre-conditions	Гравець проходить рівень кампанії (в рамках UC-07).
Flow of Events	1. Гравець успішно завершує рівень кампанії. 2. Система перевіряє, чи є наступні рівні в кампанії, які ще не відкриті для гравця. 3. Якщо такі рівні існують, система відзначає їх як доступні для гравця.
Extension	-
Post-Condition	Нові рівні кампанії стають доступними для вибору гравцем.

Таблиця 2.18 – Варіант використання UC-18

Use case name	Завантаження рівня
Use case ID	UC-18
Goals	Підготувати ігровий світ та персонажа гравця для початку ігрового процесу на обраному рівні.
Actors	Гравець
Trigger	Гравець обирає рівень для гри (в рамках UC-01, UC-07, UC-08).
Pre-conditions	Гравець обрав доступний рівень.
Flow of Events	1. Система отримує запит на завантаження обраного рівня. 2. Система завантажує дані рівня з відповідного файлу. 3. Система налаштовує ігрове оточення (розміщує об'єкти, встановлює таймер). 4. Система розміщує персонажа гравця на початковій позиції. 5. Розпочинається ігровий процес.
Extension	Можлива помилка при завантаженні рівня, якщо файл рівня пошкоджений або відсутній.
Post-Condition	Ігровий рівень завантажено, ігровий процес розпочато, або система повідомила про помилку завантаження.

Таблиця 2.19 – Варіант використання UC-19

Use case name	Поява ворогів зі спавнерів
Use case ID	UC-19
Goals	Забезпечити динамічну появу ворожих сутностей у ігровому світі контрольованим чином, створюючи виклики для гравця.
Actors	Система
Trigger	Гравець наближається до зони активації спавнера ворогів.
Pre-conditions	Спавнер ворогів присутній на рівні та активний. Кількість живих ворогів, породжених цим спавнером, не перевищує встановлений ліміт.

Продовження таблиці 2.19

Flow of Events	1. Гравець входить в зону активації спавнера. 2. Система перевіряє умови для спавну (наявність спавнера, ліміт ворогів). 3. Якщо умови виконано, система породжує хвилю ворогів. 4. Вороги з'являються на визначених точках спавнера.
Extension	Вороги можуть з'являтися періодично хвилями, доки спавнер існує та не перевищено ліміт ворогів. Спавнер може бути знищений гравцем, припиняючи появу нових ворогів.
Post-Condition	На рівні з'явилася нова група (хвиля) ворогів, породжених спавнером, або система продовжила очікування умов для спавну.

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.20 наведено загальну модель вимог, а в таблиці 2.21 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.20 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Ігровий процес	FR-1	Високий	Високий
1.1	ШІ ворогів	FR-2	Високий	Високий
1.2	Поява ворогів	FR-3	Високий	Середній
1.3	Вбивство ворогів	FR-4	Високий	Високий
1.4	Система нарахування балів	FR-5	Середній	Низький
1.5	Система паузи	FR-6	Середній	Низький

Продовження таблиці 2.20

№	Назва	ID Вимоги	Пріоритети	Ризики
1.6	Відображення завершення рівня	FR-7	Високий	Середній
2	Притаманні ігрові механіки	FR-8	Високий	Високий
2.1	Реалізація переміщення персонажа по рівню	FR-9	Високий	Високий
2.2	Реалізація роботи озброєння	FR-10	Високий	Високий
2.3	Реалізація зміни типів озброєння	FR-11	Високий	Середній
2.4	Реалізація здоров’я персонажа	FR-12	Високий	Високий
3	Проходження кампанії	FR-13	Високий	Середній
3.1	Механізм послідовного відкриття рівнів кампанії	FR-14	Високий	Середній
4	Редактор рівнів	FR-15	Високий	Високий

Продовження таблиці 2.20

№	Назва	ID Вимоги	Пріоритети	Ризики
4.1	Візуалізація об'єктів у редакторі	FR-16	Високий	Високий
4.2	Налаштування часу рівня в редакторі	FR-17	Середній	Низький
4.3	Розміщення об’єктів у редакторі	FR-18	Високий	Високий
4.4	Видалення об’єктів у редакторі	FR-19	Високий	Середній
4.5	Валідація рівня перед збереженням	FR-20	Високий	Високий
4.5	Можливість збереження рівня	FR-21	Високий	Високий
4.6	Можливість редагування збереженого рівня	FR-22	Високий	Середній
4.7	Можливість повного очищення рівня	FR-23	Середній	Низький

Продовження таблиці 2.20

№	Назва	ID Вимоги	Пріоритети	Ризики
4.8	Можливість запуску для гри користувацьких рівнів	FR-24	Високий	Середній
5	Система збору бонусів	FR-25	Середній	Низький
5.1	Підбір аптечок та збільшення здоров'я (з обмеженням)	FR-26	Середній	Середній
5.2	Система управління зброєю (підбір нової зброї, додавання в інвентар, амуніція при підборі)	FR-27	Середній	Низький
5.3	Система управління амуніцією (підбір, додавання патронів, обмеження запасу)	FR-28	Середній	Низький

Таблиця 2.21 – Перелік функціональних вимог

Назва	Опис
FR-1	Забезпечення основного циклу взаємодії гравця з ігровим світом, включаючи переміщення, бій та взаємодію з рівнем.
FR-2	Реалізація штучного інтелекту для керування поведінкою ворожих сутностей у ігровому середовищі.
FR-3	Обробка механізму появи ворогів на рівні, з використанням спавнерів та обмеженням кількості ворогів на рівні.
FR-4	Обробка процесу знищення ворогів гравцем.
FR-5	Реалізація функціональності для підрахунку та відображення нарахованих гравцю балів
FR-6	Надання можливості тимчасової зупинки ігрового процесу з доступом до меню паузи.
FR-7	Відображення екрана з повідомленням про завершення рівня, інформуючи гравця про результат (перемога чи поразка).
FR-8	Визначення базових ігрових механік, пов'язаних з рухом персонажа, взаємодією з оточенням та боєм.
FR-9	Забезпечення можливості переміщення ігрового персонажа у тривимірному просторі рівня згідно з діями гравця.
FR-10	Реалізація механізму функціонування ігрового озброєння, включаючи механіку стрільби та взаємодії з цілями.
FR-11	Надання можливості гравцю перемикатися між різними типами доступної зброї під час гри.
FR-12	Реалізація системи здоров'я для персонажа гравця, включаючи отримання шкоди та можливе лікування/відновлення.
FR-13	Визначення загальної функціональності, пов'язаної з режимом проходження стандартних рівнів, створених розробниками.
FR-14	Реалізація механізму послідовного доступу до рівнів кампанії, де кожен наступний рівень відкривається після успішного проходження попереднього.

Продовження таблиці 2.21

Назва	Опис
FR-15	Визначення всього комплексу функцій, доступних користувачу для створення та модифікації власних ігрових рівнів.
FR-16	Забезпечення візуального відображення розміщених на рівні об'єктів у редакторі.
FR-17	Дозвіл встановлювати обмеження часу на проходження створюваного рівня в межах заданого діапазону.
FR-18	Надання функціональності для додавання ігрових об'єктів на рівень у редакторі.
FR-19	Дозвіл користувачу видаляти розміщені на рівні ігрові об'єкти.
FR-20	Реалізація перевірки цілісності та коректності структури рівня перед його збереженням (наявність ключових об'єктів).
FR-21	Надання можливості збереження поточного стану рівня, створеного або відредагованого користувачем.
FR-22	Дозвіл завантажувати раніше збережені користувацькі рівні для подальшого редагування.
FR-23	Надання функції повного видалення всіх об'єктів з поточного рівня в редакторі.
FR-24	Дозвіл гравцю запускати та проходити ігрові рівні, створені ним.
FR-25	Функціонування системи бонусів
FR-26	Реалізація механізму підбору аптечок для збільшення здоров’я
FR-27	Реалізація механізму підбору нового озброєння
FR-28	Реалізація механізму підбору амуніції до озброєння

	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06	UC-07	UC-08	UC-09	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18	UC-19
FR-1	X					X	X	X	X					X	X			X	
FR-2	X						X	X						X	X			X	X
FR-3	X						X	X											X
FR-4	X			X			X	X						X	X				X
FR-5	X			X		X	X	X						X	X				
FR-6	X						X	X	X										
FR-7	X					X	X	X										X	
FR-8	X		X	X	X		X	X							X	X			
FR-9	X		X				X	X										X	X
FR-10	X			X			X	X							X	X			X
FR-11	X			X			X	X								X			
FR-12	X				X		X	X							X				
FR-13	X					X	X										X		
FR-14	X					X	X										X		
FR-15		X						X		X	X	X	X						
FR-16		X					X	X		X								X	
FR-17		X					X	X				X						X	
FR-18	X	X					X	X		X					X			X	X
FR-19		X									X								
FR-20		X								X		X							
FR-21		X						X				X							
FR-22	X	X					X	X				X						X	
FR-23		X										X	X						
FR-24		X						X											
FR-25	X				X		X	X											
FR-26	X				X		X	X											
FR-27	X			X	X		X	X								X			
FR-28	X			X	X		X	X											

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Нефункціональними вимогами цього дипломного проєкту є:

Продуктивність: гра працює у 60 кадрів на секунду, що забезпечує стабільний ігровий процес.

Зручність використання: навігація по основних функціях не потребує більше 3-4 кліків у меню.

2.4 Аналіз системних вимог

Програмне забезпечення повинно функціонувати на персональних комп’ютерах, сумісних з операційною системою сімейства Windows.

Мінімальна конфігурація технічних засобів:

– ОС: Windows 10 (64 Bit);

– об‘єм ОЗП: 4 Гб;

– процесор: Intel Core 2 Duo;

Рекомендована конфігурація технічних засобів:

– ОС: Windows 11 (64 Bit);
– об‘єм ОЗП: 8 Гб;
– процесор: Intel Core i3;

2.5 Аналіз економічних показників програмного забезпечення

Для аналізу економічних показники ПЗ використаємо метод Use Case Point (UCP). Цей метод базується на аналізі варіантів використання (Use Cases) та акторів (Actors) системи.

Оцінка за методом UCP складається з кількох етапів. Спочатку визначаються невзважені актори (UAW) та невзважені варіанти використання (UUCW). Далі обчислюються невзважені Use Case Points (UUCP). Наступним кроком є визначення технічного та середовищного коефіцієнтів коригування (TCF і EF). Після цього розраховуються зважені Use Case Points (UCP), на основі яких оцінюються зусилля на розробку. Завершальний етап - розрахунок економічних показників.

Першим етапом є визначення та зважування акторів (UAW). Актори - це зовнішні сутності, які взаємодіють із системою. У дипломному проєкті основним актором є Користувач. Оскільки взаємодія відбувається через графічний інтерфейс, цього актора класифіковано як складного. Внесок цього актора в показник UAW визначено у таблиці 2.22.

Таблиця 2.22 – Таблиця акторів

Актор	Тип	Коефіцієнт	Кількість	Внесок в UAW
Користувач	Складний	3	1	3

Варіанти використання описують функції системи. Для аналізу використаємо діаграму варіантів використання (Рисунок 2.1) та детальний опис 19 варіантів використання (Таблиці 2.1-2.19).

Оцінимо складність кожного варіанта використання (Простий, Середній, Складний) на основі опису потоку подій та враховуючи включені (include) та розширені (extension) варіанти використання, як зазначено в таблицях 2.1-2.19. Вагові коефіцієнти для варіантів використання (відповідно до методики UCP): Простий - 5, Середній - 10, Складний - 15. Оцінимо складність варіантів використання у таблиці 2.23:

Таблиця 2.23 – Таблиця оцінки складності варіантів використання

Use Case ID	Назва варіанта використання	Оцінка складності	Коефіцієнт
UC-01	Гра на ігрових рівнях	Складний	15
UC-02	Створення власних рівнів	Складний	15
UC-03	Навігація по рівню	Простий	5
UC-04	Використання зброї	Середній	10
UC-05	Підбір бонусів	Простий	5
UC-06	Завершення і відображення результатів рівня	Середній	10
UC-07	Гра на рівнях кампанії	Простий	5
UC-08	Гра на користувацьких рівнях	Простий	5
UC-09	Поставити гру на паузу	Простий	5
UC-10	Розміщення об'єктів	Середній	10
UC-11	Видалення об'єктів	Простий	5
UC-12	Керування збереженням рівнів	Середній	10
UC-13	Очищення об’єктів	Простий	5
UC-14	Збільшення ігрового рахунку	Простий	5
UC-15	Знищення ворогів	Середній	10

Продовження таблиці 2.23

Use Case ID	Назва варіанта використання	Оцінка складності	Коефіцієнт
UC-16	Зміна типів озброєння	Простий	5
UC-17	Відкриття нових рівнів	Простий	5
UC-18	Завантаження рівня	Середній	10
UC-19	Поява ворогів зі спавнерів	Середній	10

Наступним етапом підрахуємо загальну кількість варіантів використання за складністю та розрахуємо UUCW у таблиці 2.24:

Таблиця 2.24 – Загальна складність варіантів використання

Складність варіанта використання	Коефіцієнт	Кількість варіантів використання	Внесок в UUCW
Простий	5	10	50
Середній	10	7	70
Складний	15	2	30
Всього		19	150

Наступним етапом є розрахунок невзважених Use Case Points (UUCP). Для визначення UUCP скористаємося формулою 2.1:

$$UUCP = UAW + UUCW \quad (2.1)$$

$$UUCP = 3 + 150 = 153$$

Наступним етапом є визначення коефіцієнтів коригування складності: технічного (TCF) та середовищного (EF). Для розрахунку зважених Use Case Points (UCP) необхідно врахувати вплив цих факторів.

Технічні фактори (TCF): Припустимо, що середній вплив кожного з 9 технічних факторів становить 3. TCF знайдемо за формулою 2.2

$$TFactor = 9 * 3 = 27.$$

$$TCF = 0.6 + (0.01 \times TFactor) \quad (2.2)$$

$$TCF = 0.6 + (0.01 \times 27) = 0.6 + 0.27 = 0.87$$

Фактори середовища (EF): Припустимо, що середній вплив кожного з 8 факторів середовища становить 3 (середній вплив за шкалою від 0 до 5). EF знайдемо за формулою 2.3

$$EFactor = 8 * 3 = 24.$$

$$EF = 1.4 + (-0.03 \times EFactor) \quad (2.3)$$

$$EF = 1.4 + (-0.03 \times 24) = 1.4 - 0.72 = 0.68$$

Наступним етапом є розрахунок зважених Use Case Points (UCP). Значення UCP визначається шляхом множення UUCP на коефіцієнти TCF та EF. Для цього спочатку обчислюється VAF за формулою 2.4, після чого використовується формула 2.5 для остаточного розрахунку UCP.

$$VAF\ (Value\ Adjustment\ Factor) = TCF * EF \quad (2.4)$$

$$VAF = 0.87 * 0.68 = 0.5916$$

$$UCP = UUCP \times VAF \quad (2.5)$$

$$UCP = 153 \times 0.5916 \approx 90.59$$

Округлимо до цілого числа: UCP ≈ 91

Передостаннім етапом є оцінка зусиль розробки (Effort). Вона виконується в людино-годинах або людино-місяцях на основі значення UCP. Для цього застосовується коефіцієнт продуктивності - кількість людино-годин на один UCP. Типові значення цього коефіцієнта становлять від 15 до 40 людино-годин/UCP. У межах дипломного проєкту для одного виконавця приймається значення 20 людино-годин/UCP. Effort знайдемо за формулою 2.6.

$$\text{Effort (люд.}-\text{год)} = \text{UCP} \times \text{Коефіцієнт продуктивності} \quad (2.6)$$

$$\text{Effort (люд.}-\text{год)} = 91 \times 20 = 1820\ \text{людино} - \text{годин}$$

Для переведення в людино-місяці скористаємося формулою 2.7:

$$Effort\ (\text{люд.}-\text{міс}) = \frac{Effort(\text{люд.}-\text{год})}{(\text{робочих годин на місяць})} \quad (2.7)$$

$$Effort\ (\text{люд.}-\text{міс}) = \frac{1820}{160} = 11.375\ \text{людино} - \text{місяців}$$

Оцінка зусиль розробки ≈ 11.38 людино-місяців

Останнім етапом є розрахунок економічних показників. На основі оцінених зусиль розробки визначається загальна вартість проєкту. Припускається, що орієнтовна вартість одного людино-місяця для виконання такого проєкту становить 1500 USD. Вартість проєкту знайдемо за формулою 2.8

$$\text{Вартість проєкту} = Effort\ (\text{люд.}-\text{міс}) * \text{Вартість людино} - \text{місяця} \quad (2.8)$$

$$\text{Вартість проєкту} = 11.38 * 1500 = 17070\ USD$$

Отже, оцінені зусилля розробки становлять приблизно 11.38 людино-місяців, а орієнтовна вартість проєкту - 17 070 доларів США.

2.6 Постановка завдання на розробку програмного забезпечення

Програмне забезпечення повинно бути реалізовано на мові C# з використанням рушія Unity та бути оптимізованим для запуску на ПК під управлінням ОС Windows.

Розробка призначена для забезпечення ігрового процесу гри в жанрі 3D Action з елементами просторової навігації для широкого кола користувачів.

Метою розробки є підвищення якості гри у жанрі 3D Action з елементами просторової навігації за рахунок створення штучного інтелекту, що формалізує поведінку противників на основі оригінального дерева прийняття рішень.

Мета досягається шляхом вирішення наступних задач:

- забезпечення функціонування ігрового процесу.
- забезпечення функціонування притаманних ігрових механік.
- забезпечення можливості проходження кампанії.
- забезпечення можливості створення користувацьких рівнів.
- забезпечення роботи системи збору бонусів.

Висновки до розділу

У даному розділі було детально розроблено вимоги до програмного забезпечення. Було описано основні варіанти використання програмного забезпечення, розроблено діаграму використання, наведено детальний опис 19 варіантів використання, які охоплюють весь функціонал гри.

На основі варіантів використання та завдань дипломного проєкту, було розроблено 28 функціональних вимог, також було створено матрицю трасування вимог. Було деталізовано нефункціональні вимоги, а саме вимоги щодо продуктивності та зручності використання гри. Було проаналізовано системні вимоги, представлено мінімальні та рекомендовані вимоги для забезпечення стабільного функціонування гри.

На основі методу метод Use Case Point, було проаналізовано економічні показники програмного забезпечення. Оцінені зусилля розробки становлять приблизно 11.38 людино-місяців, а орієнтовна вартість проєкту - 17 070 доларів США.

Було сформовано постановку завдання на розробку програмного забезпечення в рамках дипломного проєкту. За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

При проєктуванні та розробці програмного забезпечення гри було обрано та реалізовано Об’єктно-орієнтовану архітектуру (ООА) як основний підхід. В рамках ООА, ігрові об'єкти представлені у вигляді екземплярів відповідних класів. Кожен клас інкапсулює дані, що описують стан об'єкта, та поведінку, що визначають його дії та реакцію на події. Вибір ООА обґрунтований кількома ключовими факторами: її сумісністю з більшістю сучасних інструментів для створення ігор, включаючи ігровий рушій Unity та мову програмування C#; наявністю розгорнутої документації та великої активної спільноти розробників, що спрощує пошук рішень та обмін досвідом; а також зручною інтеграцією з підсистемами ігрового рушія, такими як фізичний рушій, системи рендерингу, інтерфейсів та анімацій.

Для подальшого підвищення структурної організованості програмного коду та забезпечення чіткого розділення відповідальності між різними частинами системи, частково впроваджено елементи архітектурного шаблону Model-View-Controller (MVC). Шаблон MVC передбачає розділення програми на три взаємопов'язані компоненти:

– модель (Model) - представляє дані та бізнес-логіку. У контексті гри це класи, що зберігають стан гравця, параметри ворогів, дані про рівень, ігрову статистику, тощо.

– представлення (View) - відповідає за відображення даних користувачеві. Це елементи інтерфейсу користувача (HUD, меню, елементи управління редактора) та візуальне представлення ігрових об'єктів на сцені.

– контролер (Controller) - обробляє введення від користувача та взаємодіє з Моделлю та Представленням, реагуючи на дії користувача та оновлюючи стан Моделі та візуальне Представлення.

Повне уявлення про архітектуру програмного забезпечення на рівні класів надає UML діаграма класів гри, що знаходиться у графічному матеріалі, креслення 2, аркуш 1. Ця діаграма візуалізує основні класи, які були розроблені в рамках Об'єктно-орієнтованої архітектури, а також показує зв'язки між ними.

Центральним базовим класом у системі є MonoBehaviour - стандартний клас Unity, від якого успадковуються всі компоненти, що мають бути прикріплені до ігрових об'єктів (GameObjects). Усі класи, що реалізують ігрову логіку, включаючи інтерфейси взаємодії, штучний інтелект ворогів, механіки гравця, інтерфейс користувача та обробку бонусів, прямо або опосередковано є підкласами MonoBehaviour.

Класи на діаграмі згруповано за функціоналом. Основну частину становлять компоненти ігрової логіки: MainPlayer, PlayerMovement, PlayerHealthManager, WeaponManager, LaserWeapon, WeaponScript, Enemy, UFOBehaviour, LaserTurretBehaviour, PlayerFallDetector, SpawnerScript, а також інтерфейси IDamageable, IEnemy для завдання шкоди ворогам та їх спавнерам.

Штучний інтелект реалізовано за допомогою дерева рішень, яке реалізовано за допомогою класів: DecisionNode, ConditionNode, ActionNode, з діями PatrolAction, ApproachPlayerAction, AttackAction, CirclePlayerAction, DieAction, які використовує клас UFOBehaviour.

Система бонусів представлена абстрактним класом Collectable та його нащадками TestDamageCollectable, MedkitCollectable, GunCollectable, VictoryCollectable, AmmoCollectable.

Інтерфейсна частина включає класи UIManager, EditorUIHandler, InfoMenu, MainMenu, PauseManager, Results, ScoreCount, ScoreManager, які керують меню, результатами, паузою, рахунком. Компоненти редактора рівнів: EditorCameraManager, EditorObjectPlacer, EditorObjectPicker, EditorObjectDeleter, EditorInputHandler, LevelEditor, LevelSelectManager, LevelProgressManager.

Збереження рівнів реалізують класи LevelData, MapTimer, GameLevelLoader, LevelSaveLoader, PlacedObjectInfo. Допоміжні об’єкти: AmmoCount, HealthCount, EffectsManager, ObjectData, ObjectRadius.

Типи даних: WeaponType (enum), SerializableQuaternion, SerializableVector3 для серіалізації положення й обертання об’єктів.

Також було розроблено діаграму класів з групуванням за пакетами - спрощене уявлення архітектури гри, де класи згруповані за функціями в логічні модулі. Діаграма знаходиться у графічному матеріалі, креслення 2, аркуш 2. Діаграма показує, які частини проєкту залежать одна від одної, хто з ким взаємодіє. Діаграма наочно демонструє, як пов’язані між собою гравець, вороги, дерево рішень, інтерфейс, редактор рівнів, зброя та менеджери. Діаграма містить наступні пакети:

– Player - відповідає за логіку гравця: рух, здоров’я, зброю та взаємодію з ігровим світом.

– Enemies - містить класи ворогів і їх поведінку.

– DecisionTree - містить абстрактні класи та їхні конкретні реалізації, що формують систему штучного інтелекту на основі дерева прийняття рішень.

– Collectables - реалізує предмети для підбирання.

– LevelEditor - забезпечує роботу редактора рівнів та запуск рівнів.

– UI - управляє всіма елементами інтерфейсу користувача.

– Weapons - реалізує різні види зброї та їх поведінку у грі.

– Visuals - відповідає за візуальні ефекти та анімації об’єктів.

– SingletonesManagers - керує глобальними станами гри, меню паузи та іншими менеджерами.

– MenuScripts - управляє роботою основних меню гри.

– Interfaces - містить інтерфейси для ворогів та об’єктів, які можуть отримувати шкоду.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

Перед підготовкою дипломного проєкту, було проаналізовано актуальні рушії на ринку, а саме: Unreal Engine 5, Godot Engine, Unity.

Unreal Engine 5 [18] - це потужний ігровий рушій, який відомий завдяки своїй фотореалістичній графіці, рушію Lumen для глобального освітлення та системі Nanite для роботи з моделями з мільйонами полігонів. Цей ігровий рушій надає найвищий рівень графіки на ринку, просунуті системи інструменти для створення кінематографічних сцен та потужну систему створення логіки без програмування під назвою Blueprint. З недоліків, Unreal Engine 5 має високі вимоги до заліза, що ускладнює як розробку гри, так і її подальший запуск для гравців. Проєкти мають велику вагу та складну структуру, і через велике ядро функцій розробка є набагато складнішим процесом. Через ці недоліки, Unreal Engine 5 не був обраним для розробки дипломного проєкту.

Godot Engine [19] - легкий, повністю відкритий рушій з власною мовою програмування GDScript, який поступово набирає популярність завдяки простоті використання і можливості повного контролю над проєктом. Серед його переваг є повністю відкритий код, мала вага рушія та швидкий запуск, простий у використанні UI-редактор, гнучка сцено-компонентна модель. Але є і недоліки, серед яких: обмежені можливості у сфері 3D-графіки, слабка система анімацій і фізики, обмеженість інструментів для створення штучного інтелекту. Попри свої переваги, Godot Engine був відкинутий при створенні дипломного проєкту через слабшу підтримку інструментів для створення 3D ігор та обмеженість у системах анімації, фізики й інструментах для створення ШІ ворогів.

Unity [20] - це рушій із відкритою структурою, який активно підтримує 3D графіку, фізику, анімації, системи частинок, користувацький інтерфейс, а також містить велику кількість вдубованих інструментів для взаємодії з цими елементами. Значною перевагою Unity є його широка підтримка як розробниками, так і спільнотою. З недоліків, Unity має відносно нижчу якість

візуалізації «з коробки» порівняно з Unreal Engine 5 та менш зручну підтримку великих проєктів.

У межах дипломного проєкту для реалізації ключових функціональних елементів, таких як система просторової навігації, штучний інтелект ворогів та редактор рівнів, було обрано ігровий рушій Unity. Цей рушій із відкритою архітектурою забезпечує широкі можливості для створення інтерактивних 3D-додатків, завдяки своїй кросплатформеності, гнучкій компонентній системі побудови об’єктів, а також вбудованим засобам керування анімаціями та системами частинок. Незважаючи на деякі обмеження, зокрема менш якісну графіку за замовчуванням порівняно з Unreal Engine 5 та обмежену зручність роботи з великими проєктами, ці аспекти не стали визначальними у контексті поставлених завдань, оскільки головна увага приділялася саме ігровій логіці, а не візуальній досконалості.

У процесі розробки дипломного проєкту було обрано інтегроване середовище розробки Visual Studio IDE [21] з огляду на його широку функціональність, стабільність роботи та глибоку інтеграцію з рушієм Unity. Visual Studio забезпечує повноцінну підтримку мови програмування C#, яка є основною для створення скриптів у Unity, а також містить розширені можливості для налагодження коду, підсвічування синтаксису, автодоповнення, інструменти аналізу коду та вбудовану систему керування пакетами NuGet [22].

В дипломному проєкті було використано два алгоритми ШІ для ворогів. Перший з них - машина скінченних станів. Цей алгоритм було використано для створенні ШІ найпростішого ворога в грі - турелі, яка перемикає стани спокою та нападу на гравця залежно від відстані до нього. Цей алгоритм був обраний завдяки його простоті реалізації, маленького навантаження на комп'ютер та передбачуваній поведінці. Такі вороги є досить легкими мішенями для гравця заради того, щоб навчити його основним механікам гри, а саме оминанню попадання у себе лазером та механіку стрільби по ворогам.

У процесі розробки поведінки для літаючих противників типу НЛО було застосовано більш просунутий підхід до штучного інтелекту, а саме - алгоритм дерева прийняття рішень.

3.3 Конструювання програмного забезпечення

3.3.1 Розробка алгоритму

Алгоритм дерева прийняття рішень ґрунтується на побудові ієрархічної структури, де кожен вузол виконує роль або перевірки умови, або ініціації конкретної дії. Основними абстрактними компонентами є DecisionNode, який виступає кореневим елементом дерева і відповідає за вибір подальшої гілки на основі умов, ConditionNode, який інкапсулює логіку перевірки певного стану гри (наприклад, чи виявлено гравця або чи живий НЛО), та ActionNode, який відповідає за виконання певної поведінки (атака, патрулювання, смерть тощо). У певні часові інтервали (кожну секунду) кореневий вузол починає аналіз ігрового стану, послідовно перевіряючи умови через підлеглі ConditionNode і, зрештою, обирає відповідний ActionNode. Останній запускає пов’язану корутину, що реалізує поведінкову дію НЛО. Такий підхід дозволяє не лише будувати складні гілки рішень, а й легко масштабувати та адаптувати логіку поведінки до нових ігрових ситуацій. Схему розробленого алгоритму дерева прийняття рішень для ворога НЛО можна побачити на рисунку 3.1:

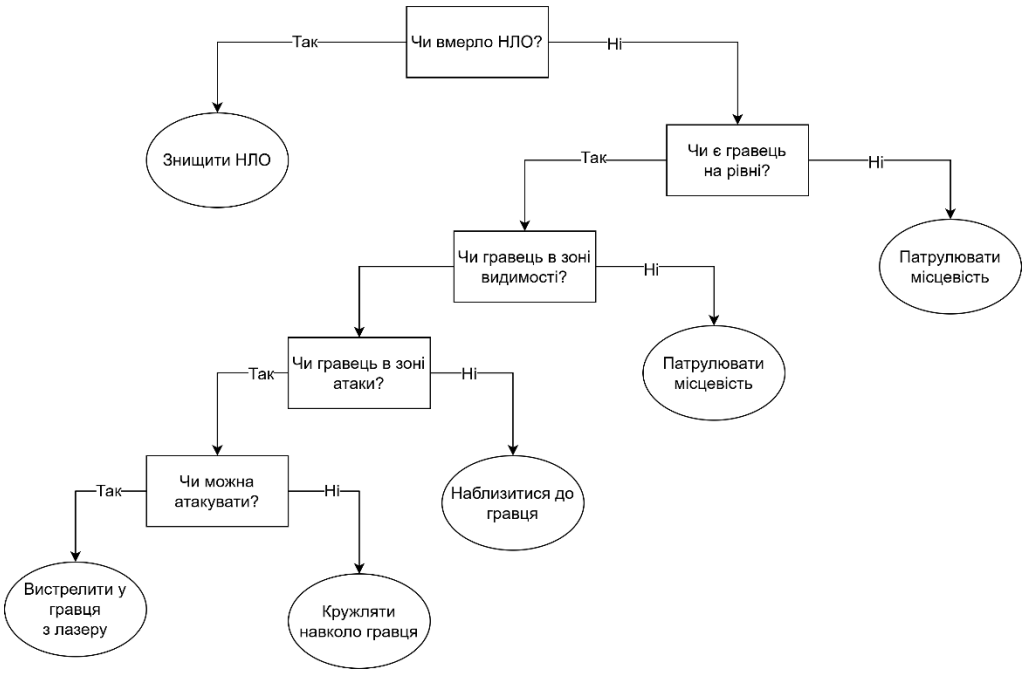


Рисунок 3.1 – Алгоритм дерева прийняття рішень для ворога НЛО

Реалізація описаного підходу відбувається на рівні кількох ключових класів. Головний з них - UFOBehaviour.cs, який є нащадком MonoBehaviour і реалізує інтерфейс IEnemy. У ньому зберігається посилання на кореневий вузол дерева рішень (rootDecisionNode), з якого й починається обробка поведінки. Метод Start() ініціалізує алгоритм, запускаючи корутину DecisionTreeCoroutine(), яка з інтервалом у секунду виконує логіку прийняття рішень: викликає метод MakeDecision() на кореневому вузлі, отримує результат у вигляді ActionNode і, якщо вибрана дія відрізняється від поточної, перезапускає відповідну корутину з виконанням нової поведінки через Execute().

Класи DecisionNode.cs, ConditionNode.cs та ActionNode.cs виступають базовими абстракціями для створення конкретних поведінкових елементів. Зокрема, ConditionNode реалізується у вигляді похідних класів на зразок IsPlayerDetectedDecision або IsDeadDecision, які здійснюють перевірку станів і повертають відповідні вузли в залежності від результату. ActionNode, своєю чергою, включає похідні реалізації таких дій, як PatrolAction (патрулювання навколо спавну), ApproachPlayerAction (наближення до гравця), AttackAction (запуск атаки), CirclePlayerAction (кружляння навколо гравця) та DieAction (ініціація смерті).

3.3.2 Опис структур даних

У процесі розробки редактора рівнів, реалізовано механізм збереження та завантаження рівнів гри із використанням серіалізації через BinaryFormatter. Дані про ігрові об'єкти та налаштування рівня перетворюються на байтову послідовність інформації та зберігаються у файл

Алгоритм збереження передбачає збір даних щодо об'єктів сцени, а саме їх тип, позицію та орієнтацію у просторі. Далі формується об'єкт типу LevelData, який включає список структур PlacedObjectData з параметрами об'єктів і додатковими налаштуваннями рівня, зокрема обмеженням часу на проходження рівня. За допомогою BinaryFormatter об'єкт серіалізується у бінарний файл. Завантаження відбувається у зворотному порядку: зчитування даних, десеріалізація, відтворення сцени.

Для підтримки збереження розташування об’єктів у просторі, використано серіалізовані структури SerializableVector3 і SerializableQuaternion. Клас LevelData об'єднує всі необхідні дані рівня для збереження та завантаження. Основні компоненти системи збереження рівня - класи LevelSaveLoadManager, LevelEditor і GameLevelLoader. Вони забезпечують збереження, редагування та відтворення рівня. Додатковий скрипт PlacedObjectInfo зберігає назву префабу для коректного розташування відповідних типів об'єктів на сцені при їх завантаженні.

3.3.3 Опис утиліт

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.1.

Таблиця 3.1 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Unity Editor	Головне середовище розробки програмного забезпечення, ігровий рушій
2	Visual Studio	IDE, яке широко використовується для розробки ігор у рушії Unity. Надає інструменти для написання, відлагодження та оптимізації коду на мові C#
3	Paint.net [23]	Програмне забезпечення необхідне для створення візуальної частини об’єктів, а саме їх текстур
4	Audacity [24]	Програмне забезпечення, необхідне для редагування аудіо файлів для звуків у грі

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Дипломний проєкт сфокусовано на розробку користувацької гри для однієї особи, що не отримує користувацьких даних, тому жодні вимоги до безпеки даних не висуваються.

Висновки до розділу

У третьому розділі детально описано архітектуру програмного забезпечення, основні компоненти, їхню ієрархічну структуру та логіку взаємодії. Побудовано UML-діаграму класів із поділом на логічні пакети.

Обґрунтовано вибір технологічного стеку: Unity як рушій з широким функціоналом для 2D- та 3D-ігор; мову C# як основний інструмент для реалізації ігрової логіки; Visual Studio як зручне інтегроване середовище розробки; Paint.NET і Audacity для створення та обробки мультимедійних ресурсів.

Описано роботу алгоритму дерева прийняття рішень, яке визначає поведінку ворогів НЛО. Побудовано схему роботи алгоритму.

Виконано опис механізмів збереження та завантаження рівнів гри з використанням бінарних файлів.

Ризики, пов’язані з безпекою даних, були відхилені з огляду на відсутність персоніфікованої інформації та мережевої взаємодії у рамках проєкту.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

4.1.1 Аналіз якості коду

Для початку було проведено оцінювання якості та складності програмного коду за допомогою вбудованого інструмента Visual Studio - Code Metrics Analysis [25], що дозволяє отримати узагальнене уявлення про підтримуваність і архітектурну організацію програмного проєкту. Застосовано основні метрики, серед яких індекс підтримуваності (Maintainability Index), цикломатична складність (Cyclomatic Complexity), глибина наслідування (Depth of Inheritance), зв'язаність класів (Class Coupling), а також кількість рядків сирцевого (Source) та виконуваного (Executable) коду. Результат аналізу можна побачити на рисунку 4.1.

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
Assembly-CSharp (Debug)	77	1 067	6	118	5 332	1 939
IEnemy	100	3	0	3	8	0
IDamagable	100	1	0	0	4	0
WeaponType	100	1	1	0	5	0
SerializableVector3	88	3	1	2	24	5
SerializableQuaternion	81	3	1	2	26	6
ObjectData	61	3	1	10	28	9
LevelData	100	1	1	3	6	0
DecisionNode	100	1	3	3	3	0
ActionNode	100	1	3	3	4	0
CirclePlayerAction	89	2	4	5	9	1
ApproachPlayerAction	89	2	4	5	9	1
CanAttackDecision	78	2	4	4	18	3
IsPlayerDetectedDecision	67	3	4	7	23	6
HasPlayerTargetDecision	76	2	4	6	18	3
PlayerInRangeDecision	67	3	4	7	23	6
AttackAction	82	2	4	5	10	2
PatrolAction	82	2	4	5	10	2
DieAction	82	2	4	5	10	2
IsDeadDecision	76	2	4	4	18	3
PlacedObjectInfo	100	1	5	1	4	0
PendulumRotatorY	76	2	5	8	27	10
PlayerFallDetector	81	5	5	8	24	6
PlayerHealthManager	82	18	5	4	54	18
PauseManager	82	18	5	11	73	28
ObjectRadius	62	52	5	23	283	120
PickingUpScript	77	3	5	6	12	3
PlayerMovement	72	11	5	13	91	28
RotateObject	89	1	5	5	8	1
ScoreCount	91	2	5	5	15	2
ScoreManager	92	8	5	3	25	8
AmmoCount	79	5	5	8	31	8

Рисунок 4.1 - Результати оцінювання якості програмного коду

Індекс підтримуваності (Maintainability Index) відображає зручність супроводу коду та базується на його складності, довжині й зв'язаності. Він вимірюється за шкалою від 0 до 100: значення понад 80 вказують на високу підтримуваність. У проєкті більшість модулів мають індекс 82-100, середнє значення - 77, що свідчить про середню загальну якість реалізації.

Цикломатична складність (Cyclomatic Complexity) відображає кількість незалежних логічних шляхів у програмі й прямо впливає на складність тестування. Її значення варіюються від 1 до 52, а сумарне значення становить 1067, що вказує на наявність як простих, так і складних компонентів.

Глибина наслідування (Depth of Inheritance), яка показує кількість рівнів ієрархії класів, перебуває в межах 1-6, що відповідає прийнятній структурній глибині й вказує на раціональне використання механізмів успадкування.

Зв’язаність класів (Class Coupling), тобто кількість зовнішніх типів, до яких звертається клас, змінюється від 0 до 23, а загальний показник становить 118, що свідчить про помірну взаємозалежність компонентів.

Проєкт також містить 5332 рядки сирцевого коду, з яких 1939 - виконувані інструкції, що демонструє помірний обсяг функціональної реалізації.

Враховуючи результати аналізу, можемо сказати, що проєкт готовий до подальшого розвитку й розширення без суттєвих ускладнень у підтримці чи модифікації.

4.1.1 Аналіз продуктивності гри

Наступним кроком стало дослідження продуктивності розробленої гри. Основною метрикою було обрано частоту кадрів на секунду (FPS). Для збору відповідних даних було використано безкоштовне програмне забезпечення FPSMonitor [26], за допомогою якого відображалися графіки зміни навантаження на ключові апаратні компоненти та значення FPS у режимі реального часу (рисунок 4.2).

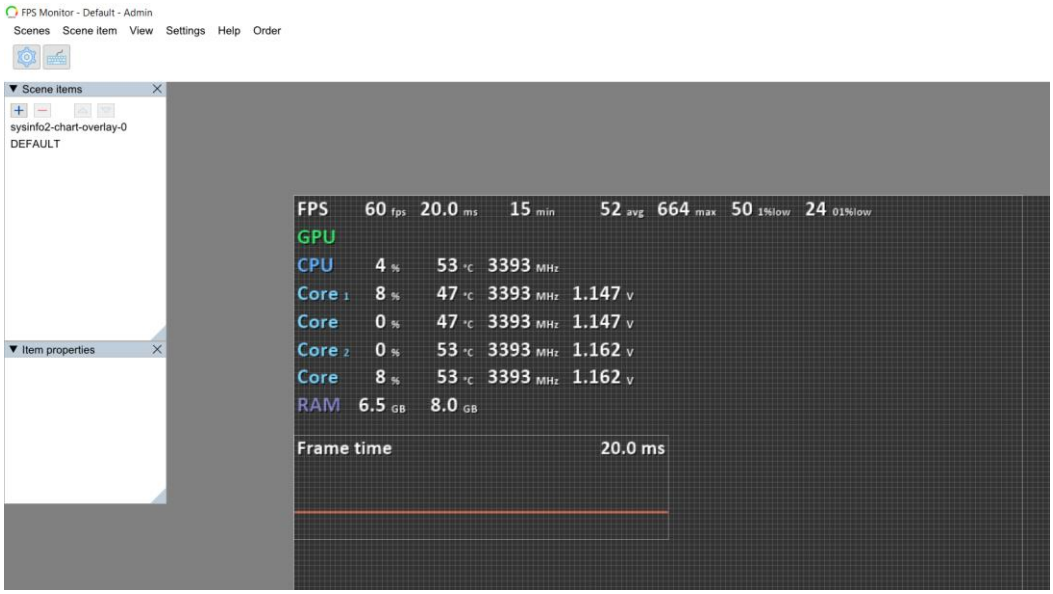


Рисунок 4.2 - Вікно програми FPSMonitor

Система, на якій проводилося тестування, має наступні характеристики: операційна система - Windows 10, процесор - Intel Core i3 із вбудованим графічним ядром, оперативна пам’ять - 8 ГБ DDR4.

На рисунку 4.3 представлено скріншот із першого рівня кампанії гри.

Рисунок 4.3 - Аналіз продуктивності гри на рівні кампанії

У ході тестування було зафіксовано стабільну частоту кадрів на рівні 60 FPS. Завантаження графічного процесора (GPU) сягало 100%, що є типовим для систем із вбудованим відеоядром, яке не має окремої відеопам’яті та використовує ресурси оперативної пам’яті. Аналогічне навантаження спостерігалося і на центральному процесорі (CPU): усі ядра працювали на

100%, а температура сягала 81-84°C. Це свідчить про інтенсивну роботу процесора, що пов’язана як із обчисленням ігрової логіки, так і з обробкою графіки за відсутності дискретного GPU. Обсяг використаної оперативної пам’яті становив 6.5 ГБ із наявних 8 ГБ, тобто система функціонувала в межах допустимих ресурсів, однак без значного резерву для багатозадачності. Час рендерингу одного кадру (frame time) залишався стабільним на рівні 16.9 мс, що відповідає 60 кадрам на секунду і підтверджує плавність та сталість відтворення зображення в грі.

На рисунку 4.4 проведено аналіз продуктивності користувацького рівня, в якому реалізовано велику кількість одночасно активних об’єктів типу «НЛО». У кадрі присутні 10 ворожих об’єктів, ще близько 10 - за межами камери, проте активні з точки зору ігрової логіки. Така кількість одночасно задіяних елементів впливає на загальне навантаження на систему.

Рисунок 4.4 - Аналіз продуктивності гри на користувацькому рівні

Частота кадрів коливається на рівні 58 кадрів на секунду, що свідчить про незначне зниження у порівнянні з попереднім рівнем, де фіксувалося стабільне значення в 60 FPS. Таке зменшення є очікуваним наслідком збільшеної кількості об'єктів, які потребують одночасної обробки логіки руху, зіткнень, анімацій та інших поведінкових сценаріїв.

Час рендерингу одного кадру коливається в межах 16.6-16.7 мс, що вказує на невелику нестабільність у генерації кадрів, пов’язану з великою кількістю ворогів на сцені. Тим не менш, поточний рівень frame time досі відповідає прийнятним показникам для плавного ігрового процесу, не створюючи помітних затримок чи ривків.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 - 4.10.

Таблиця 4.1 – Тест 1.1 Перевірка руху та навігації гравця

Початковий стан системи	Гравець знаходиться на початку рівня кампанії або користувацького рівня.
Вхідні дані	Клавіші W, A, S, D для руху; клавіша Space для стрибка; клавіша Shift для прискорення. Рухи мишею для керування камерою.
Опис проведення тесту	Гравець намагається рухатися вперед, назад, вліво, вправо. Гравець намагається стрибнути. Гравець намагається прискоритися. Гравець намагається керувати камерою. Гравець переміщується навколо перешкод та платформ.
Очікуваний результат	Персонаж гравця коректно рухається в усіх напрямках, стрибає, прискорюється, а камера коректно реагує на рухи мишею. Гравець може успішно переміщуватися по ігровому середовищу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Перевірка використання зброї - Стрільба

Початковий стан системи	Гравець знаходиться на рівні, має екіпіровану зброю з боєприпасами.
Вхідні дані	Натискання лівої кнопки комп’ютерної миші.
Опис проведення тесту	Гравець цілиться у ворога або елемент оточення та натискає ліву кнопку миші.
Очікуваний результат	Зброя стріляє. Якщо вражений ворог, ворог отримує пошкодження, чутно звук потрапляння у ціль. Якщо вражений об'єкт оточення, з'являється візуальний ефект сліду від кулі. Кількість боєприпасів зменшується. HUD оновлює кількість боєприпасів.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Перевірка використання зброї - Зміна зброї

Початковий стан системи	Гравець знаходиться на рівні та має принаймні два різні типи зброї.
Вхідні дані	Прокрутка колеса комп’ютерної миші.
Опис проведення тесту	Гравець використовує колесо миші для перемикання між доступними видами зброї.
Очікуваний результат	Екіпірована зброя змінюється на обрану. HUD оновлюється, відображаючи нову зброю та її боєприпаси.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Перевірка взаємодії з ворогом - Отримання шкоди від ворога

Початковий стан системи	Гравець знаходиться на рівні з активним ворогом (турель або НЛО).
Вхідні дані	Персонаж гравця знаходиться в зоні атаки та прямої видимості ворога.
Опис проведення тесту	Гравець дозволяє ворогу атакувати себе та не ховається від ворожого лазеру
Очікуваний результат	Здоров'я гравця зменшується. HUD оновлюється, відображаючи новий стан здоров'я. Екран світиться червоним. Якщо здоров'я досягає нуля, гравець помирає.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Перевірка взаємодії з ворогом - Знищення ворога

Початковий стан системи	Гравець знаходиться на рівні з активним ворогом. Гравець має зброю з боєприпасами.
Вхідні дані	Гравець стріляє у ворога.
Опис проведення тесту	Гравець кілька разів стріляє у ворога, доки його здоров'я не вичерпається.
Очікуваний результат	Ворог знищується, відіграється ефект вибуху ворога. Рахунок гравця збільшується.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Перевірка підбору бонусів - Аптечка

Початковий стан системи	Гравець знаходиться на рівні. На рівні присутній бонус-аптечка.
Вхідні дані	Персонаж гравця зіштовхується з аптечкою.
Опис проведення тесту	Гравець переміщує свого персонажа так, щоб підібрати аптечку.
Очікуваний результат	Здоров'я гравця збільшується. Аптечка зникає з рівня. HUD оновлює здоров'я. Якщо здоров'я вже було повним, воно залишається повним, а аптечка не підбирається.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Перевірка підбору бонусів - Боєприпаси

Початковий стан системи	Гравець знаходиться на рівні, має екіпіровану зброю. Присутній бонус боєприпасів для екіпірованої зброї.
Вхідні дані	Персонаж гравця зіштовхується з бонусом боєприпасів.
Опис проведення тесту	Гравець переміщує свого персонажа так, щоб підібрати бонус боєприпасів.
Очікуваний результат	Кількість боєприпасів для відповідної зброї збільшується. Бонус боєприпасів зникає. HUD оновлює кількість боєприпасів. Якщо гравець не має відповідної зброї, бонус не підбирається.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Перевірка завершення рівня - Перемога

Початковий стан системи	Гравець знаходиться на рівні.

Продовження таблиці 4.8

Вхідні дані	Гравець успішно добирається до кінцевого астероїда протягом ліміту часу.
Опис проведення тесту	Гравець проходить рівень та досягає кінцевої точки (фінального астероїда).
Очікуваний результат	Відображається екран "Перемога" ("You Won!"), що показує рахунок та витрачений час. Гравець може повернутися до головного меню. Якщо це був рівень кампанії, наступний рівень може бути розблокований.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Перевірка завершення рівня - Поразка

Початковий стан системи	Гравець знаходиться на рівні.
Вхідні дані	Здоров'я гравця досягає нуля через атаки ворогів, або гравець падає за межі ігрової зони.
Опис проведення тесту	Гравець гине від ворога або падає в чорну діру.
Очікуваний результат	Відображається екран "Поразка" ("You Died!"), що показує рахунок та час виживання. Гравець має опції перезапустити рівень або повернутися до головного меню.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Перевірка функціональності меню паузи

Початковий стан системи	Гравець активно грає на рівні.
Вхідні дані	Натискання клавіші "ESC". Натискання кнопок "Resume", "Restart" або "Menu" в меню паузи.
Опис проведення тесту	Гравець натискає "ESC" для відкриття меню паузи. Потім гравець намагається продовжити гру, перезапустити рівень та повернутися до головного меню за допомогою відповідних кнопок.
Очікуваний результат	Меню паузи з'являється при натисканні "ESC". Ігровий процес призупиняється. Кнопка "Resume" повертає до гри. Кнопка "Restart" перезавантажує поточний рівень з початку. Кнопка "Menu" переводить гравця до головного меню гри.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Редактор рівнів - Розміщення об'єкта

Початковий стан системи	Гравець знаходиться в Редакторі рівнів.
Вхідні дані	Вибір типу об'єкта з інтерфейсу редактора (за допомогою клавіш 0-9). Натискання лівої кнопки миші на сітці розміщення.
Опис проведення тесту	Гравець обирає об'єкт (наприклад, астероїд, спавнер ворогів) та натискає на дійсне місце на сітці для його розміщення. Спроба розмістити на недійсному місці (наприклад, накладаючи на інший об'єкт, або розміщуючи унікальний об'єкт, такий як стартовий або кінцевий астероїд, двічі).

Продовження таблиці 4.11

Очікуваний результат	Обраний об'єкт з'являється на сітці у вказаному місці, якщо воно дійсне. Візуальний попередній перегляд (тінь) показує область колізії та підсвічує дійсне (зеленим) або недійсне (червоним) розміщення. Повідомлення про помилку з'являється, якщо спробувати розмістити унікальний об'єкт, який вже існує, або якщо розміщення неможливе.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Редактор рівнів - Видалення об'єкта

Початковий стан системи	Гравець знаходиться в Редакторі рівнів. На сітці розміщено принаймні один об'єкт.
Вхідні дані	Натискання правої кнопки миші на розміщеному об'єкті.
Опис проведення тесту	Гравець наводить курсор миші на розміщений об'єкт та натискає праву кнопку миші.
Очікуваний результат	Обраний об'єкт видаляється з сітки.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Редактор рівнів - Збереження та завантаження користувацького рівня

Початковий стан системи	Гравець знаходиться в Редакторі рівнів. Рівень був створений або змінений (містить принаймні стартовий та фінішний астероїд).

Продовження таблиці 4.13

Вхідні дані	Натискання "ESC" для відкриття меню керування рівнями. Натискання кнопки "Save Slot". Згодом, натискання кнопки "Load Slot" для збереженого рівня.
Опис проведення тесту	Гравець створює рівень, переконуючись, що він має стартову та фінішну точки. Гравець зберігає рівень у слот. Потім гравець очищує редактор або завантажує інший рівень, а потім намагається завантажити раніше збережений рівень.
Очікуваний результат	Рівень успішно зберігається в обраний слот з повідомленням про підтвердження. Збережений рівень завантажується коректно, відновлюючи всі розміщені об'єкти та налаштування. Повідомлення про помилку з'являється, якщо рівень недійсний (відсутня стартова/фінішна точка).
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 1.14 Гра на створеному користувачем рівні

Початковий стан системи	Дійсний створений користувачем рівень збережений в одному зі слотів. Гравець знаходиться в меню "Вибір рівня".
Вхідні дані	Натискання кнопки "User Level", що відповідає збереженому рівню.
Опис проведення тесту	Гравець переходить на екран "Вибір рівня" та обирає для гри раніше створений та збережений користувацький рівень.

Продовження таблиці 4.14

Очікуваний результат	Обраний користувацький рівень завантажується, і починається ігровий процес відповідно до дизайну цього рівня. Всі розміщені об'єкти, вороги та бонуси функціонують належним чином.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15 – Тест 1.15 Перевірка радіусу дії ворогів

Початковий стан системи	Гравець починає рівень (кампанії або користувацький), який містить ворожу турель або спавнер НЛО
Вхідні дані	Персонаж гравця рухається в межах зони виявлення/атаки турелі або поблизу зони активації спавнера НЛО.
Опис проведення тесту	Гравець наближається до турелі, щоб спостерігати за її поведінкою. Гравець наближається до спавнера НЛО.
Очікуваний результат	Турель залишається неактивною, коли гравець поза зоною досяжності, і атакує, коли гравець знаходиться в зоні досяжності. НЛО з'являються зі спавнера, коли гравець входить в зону активації.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Наведемо опис контрольного прикладу використання редактора рівнів.

Перейшовши до редактора рівнів, ми можемо побачити площину встановлення об’єктів (рисунок 4.5).

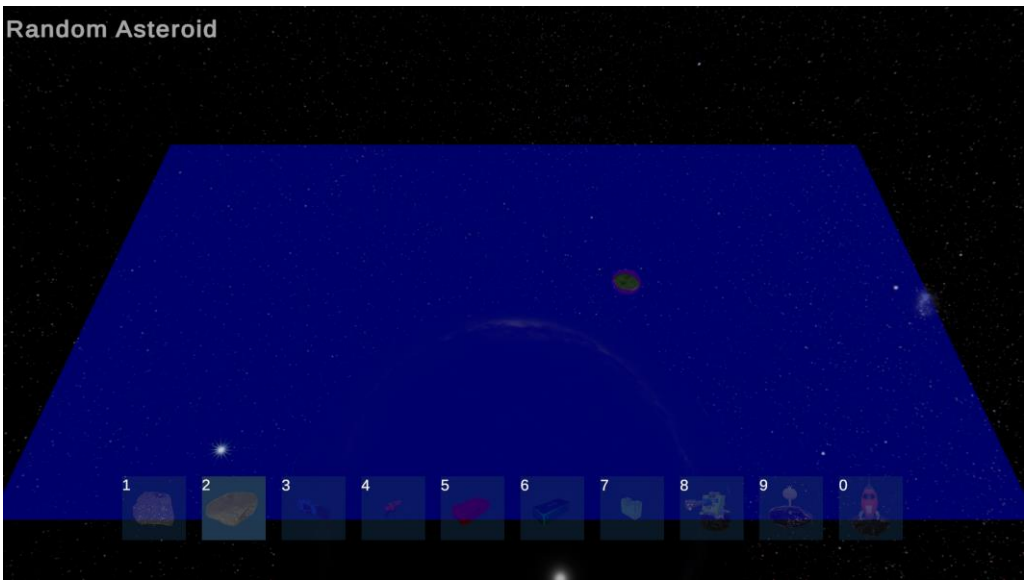


Рисунок 4.5 – Площина розміщення об’єктів редактора рівнів

Користувач може керувати камерою, наближаючи її до площини, використовуючи колесо комп’ютерної миші та може переміщувати камеру навколо площини за допомогою клавіш: W (вперед), A (вліво), S (назад), D (вправо). Переміщення камери обмежено для того, щоб користувач не міг відлетіти далеко від площини розміщення об'єктів, або перемістится під неї. Користувач може обирати об'єкти до розміщення на рівні, використовуючи клавіші: 1 для розміщення стартового астероїду, 2 для розміщення випадкового звичайного астероїду, 3 для розміщення астероїду з плазменною рушницею, 4 для розміщення астероїду з лазерним пістолетом-кулеметом, 5 для розміщення астероїду з амуніцією для лазерного пістолета-кулемета, 6 для розміщення астероїду з амуніцією для плазменної рушниці, 7 для розміщення астероїду з аптечкою, 8 для розміщення астероїду з лазерною турелю, 9 для розміщення спавнера НЛО, 0 для розміщення кінцевого астероїду. Вибір користувача відображається у вигляді комірок з іконками обраного астероїду, які можна побачити внизу екрану на рисунку 4.5. Також вибір користувача можна побачити у текстовому вигляді у лівому верхньому куті екрану. Наблизивши камеру, можна побачити тінь об'єкта для розміщення, яка показує його область колізії (рисунок 4.6).

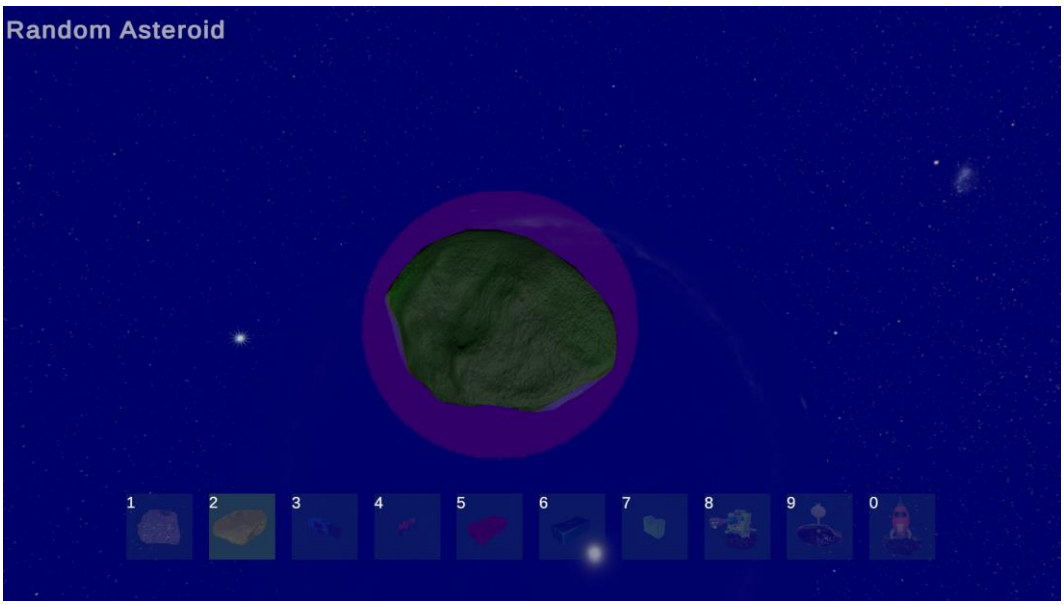


Рисунок 4.6 – Тінь об'єкта, що розміщується

Тінь об'єкта відповідає за його зайнятий простір. Якщо об'єкт можна розмістити, він підсвічується зеленим кольором, якщо його не можна розмістити - червоним (рисунок 4.7). Об'єкт не можна розмістити, якщо його тінь пересікається з іншим об'єктом, або екземпляр об'єкту є одиничним. Одиничними екземплярами об'єктів є початковим та кінцевий астроїди. Об'єкт розміщується за допомогою натискання на ліву кнопку комп'ютерної миші. Видалити об'єкт можна, якщо навестися на нього та натиснути праву кнопку комп'ютерної миші.

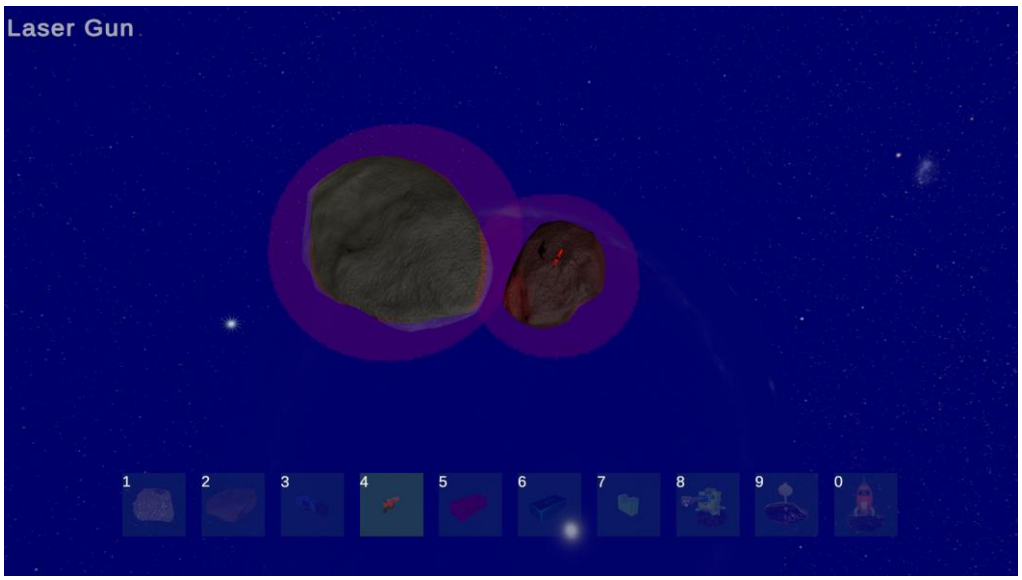


Рисунок 4.7 – Неможливість розмістити об'єкт

Якщо гравець намагається розмістити початковий або кінцевий астероїд двічі, гра виводить йому про це помилку та унеможливлює повторне розміщення цього об'єкту (рисунок 4.8).

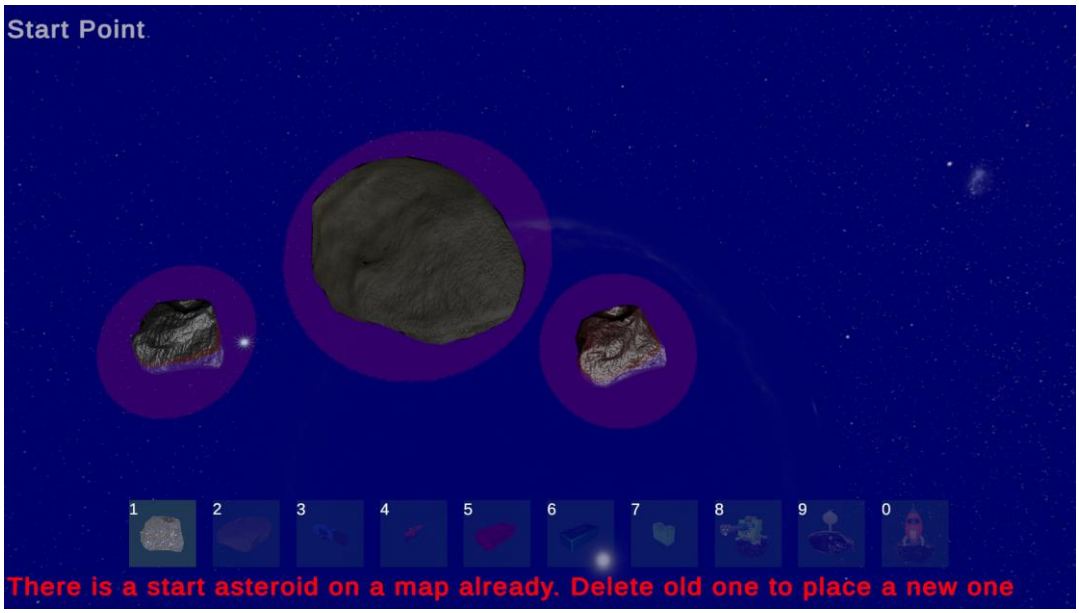


Рисунок 4.8 – Помилка про повторне розміщення стартового астероїда

Розміщуючи об'єкти ворогів, а саме лазерних турелей та спавнерів НЛО, гравець може побачити радіус їх дії для планування подальшої гри на рівні (рисунок 4.9).

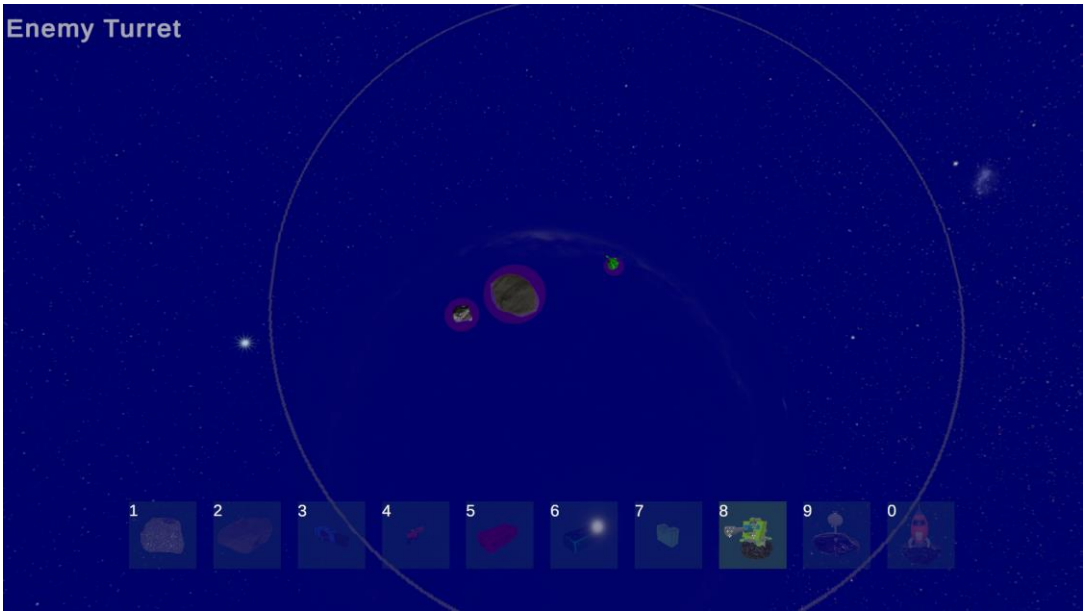


Рисунок 4.9 – Радіус дії лазерної турелі

При натисканні на клавішу “ESC”, гравець потрапляє до меню керування рівнями (рисунок 4.10).

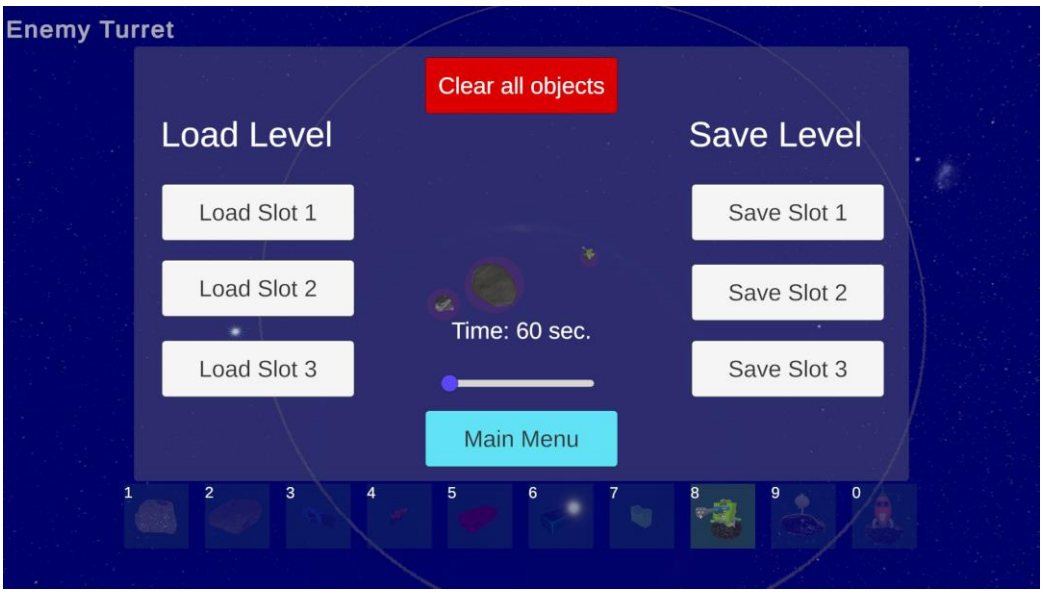


Рисунок 4.10 – Меню керування рівнями

Тут гравець може завантажити вже збережені рівні, або зберегти новий рівень. Також гравець може встановити час проходження рівня, який варіюється від 60 до 180 секунд. За бажанням, гравець може очистити всі об’єкти на рівні. При обиранні будь-якої дії, крім повернення до рівня редактора при повторному натисканні клавіші “ESC”, користувач отримує вікно підтвердження для запобігання втрати незбережених змін (рисунок 4.11).

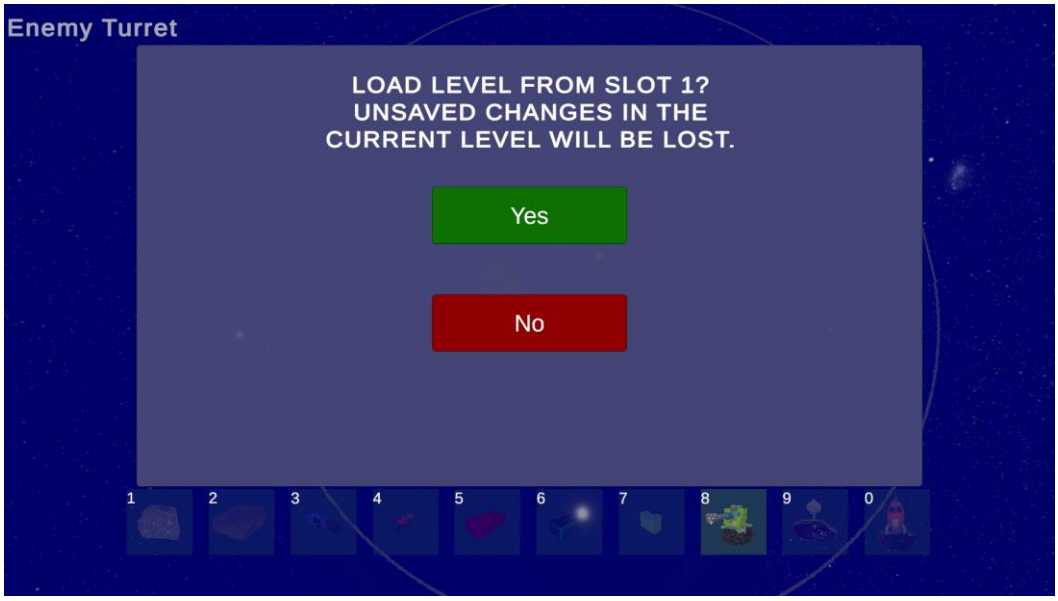


Рисунок 4.11 – Меню підтвердження дії

При збереженні рівня, він валідується на коректність (наявність стартового та фінального астероїда), та виводить помилку у разі

перешкоджання збереження рівня (рисунок 4.12) або повідомлення про успішне збереження рівня (рисунок 4.13).

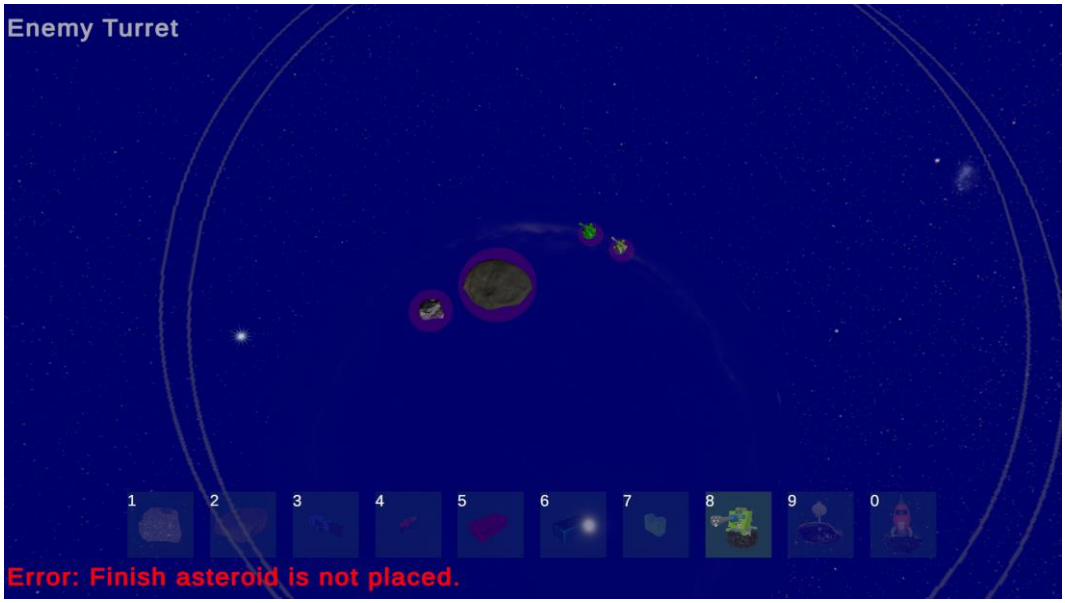


Рисунок 4.12 – Помилка збереження рівня

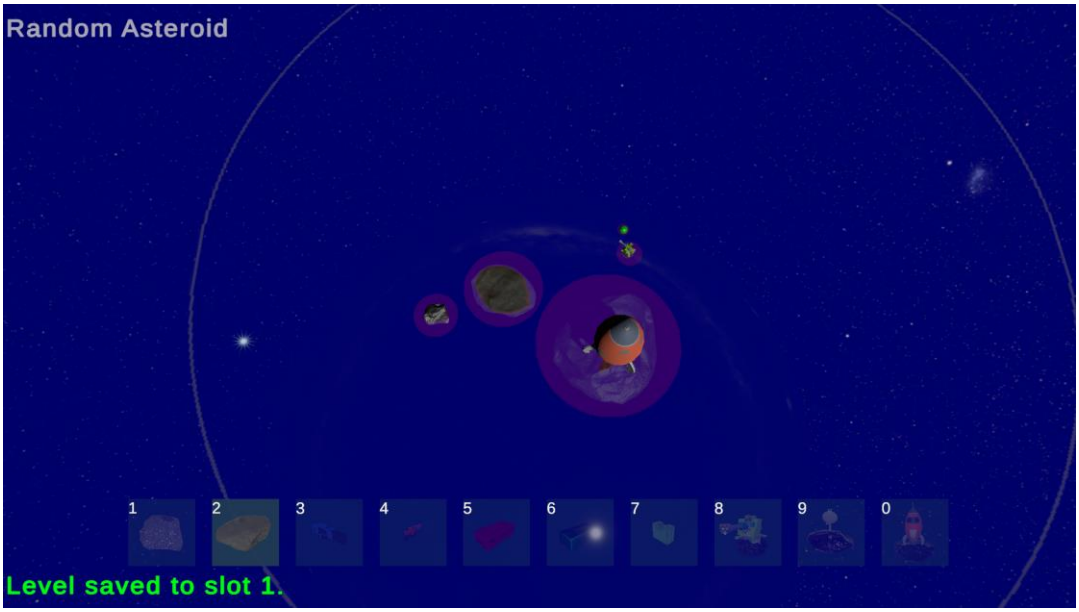


Рисунок 4.13 – Повідомлення про успішне збереження рівня

З меню керування рівнями (рисунок 4.10) можна перейти до головного меню, з якого можна перейти до меню гри на рівнях (рисунок 4.14).

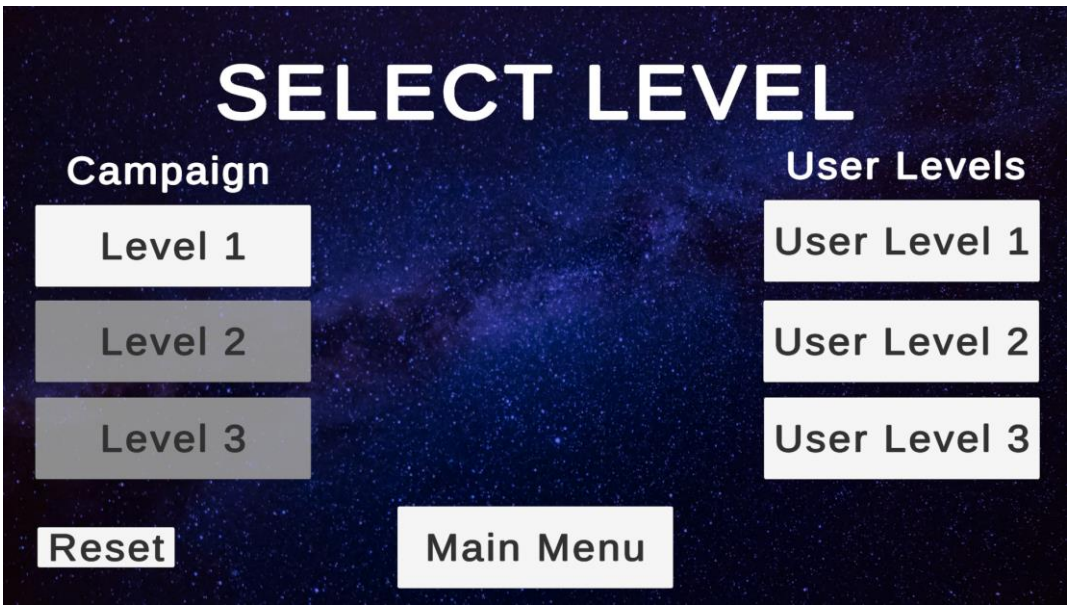


Рисунок 4.14 – Меню гри на рівнях

В меню гри на рівнях ми можемо обрати рівні кампанії, або можемо обрати доступні користувацькі рівні. Доступні рівні підсвічуються білим кольором, недоступні - сірим кольором.

Висновки до розділу

У четвертому розділі дипломного проєкту було здійснено аналіз якості програмного забезпечення та його тестування. Було визначено індекс підтримуваності, цикломатичну складність, глибину наслідування та зв’язаність класів. Отримані результати свідчать про достатню структурованість і підтримуваність коду. Додатково оцінено продуктивність програми. Результати свідчать про високий рівень продуктивності гри, FPS тримався на рівні 60 кадрів у секунду.

Було проведено ручне тестування згідно з «Програмою та методикою тестування». Було перевірено усі можливості ігрового процесу, а також усі можливості редактора рівнів.

Проведений контрольний приклад описує ключовий функціонал роботи редактора рівнів з усіма можливими варіантами подій. Опис доповнено ілюстраціями розробленої гри.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання проєкту скористаємося вбудованою функцією пакування гри для Unity Editor. Для цього необхідно зайти у вкладку “File” та обрати параметр “Build Settings” (рисунок 5.1). Далі треба зазначити цільову платформу - Windows (рисунок 5.2).

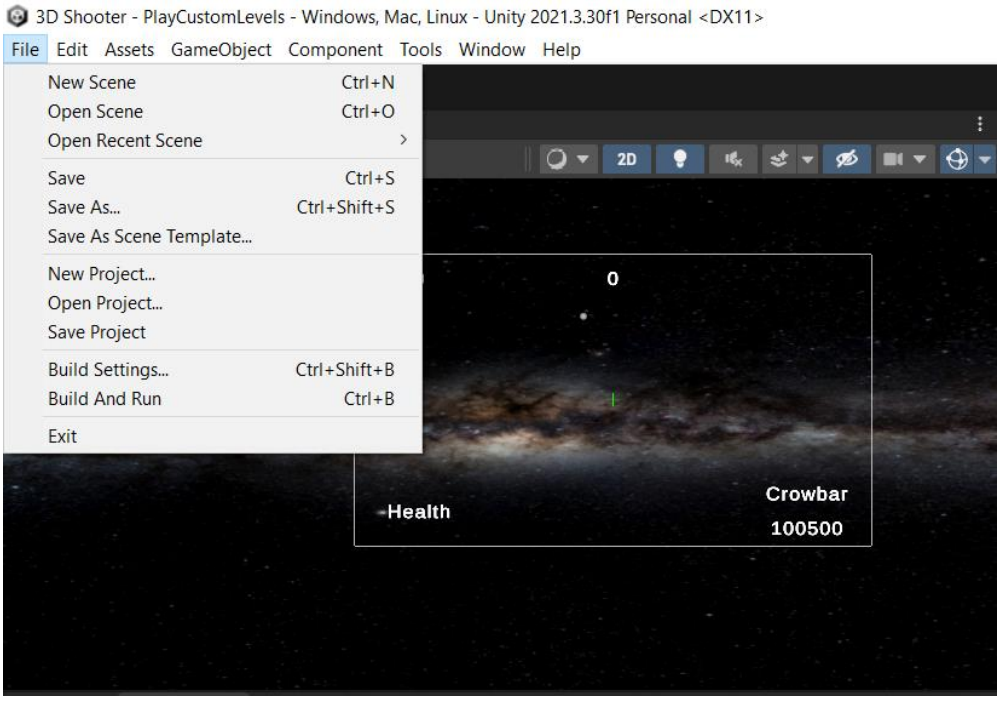


Рисунок 5.1 – Вікно редактора Unity з вкладкою «File»

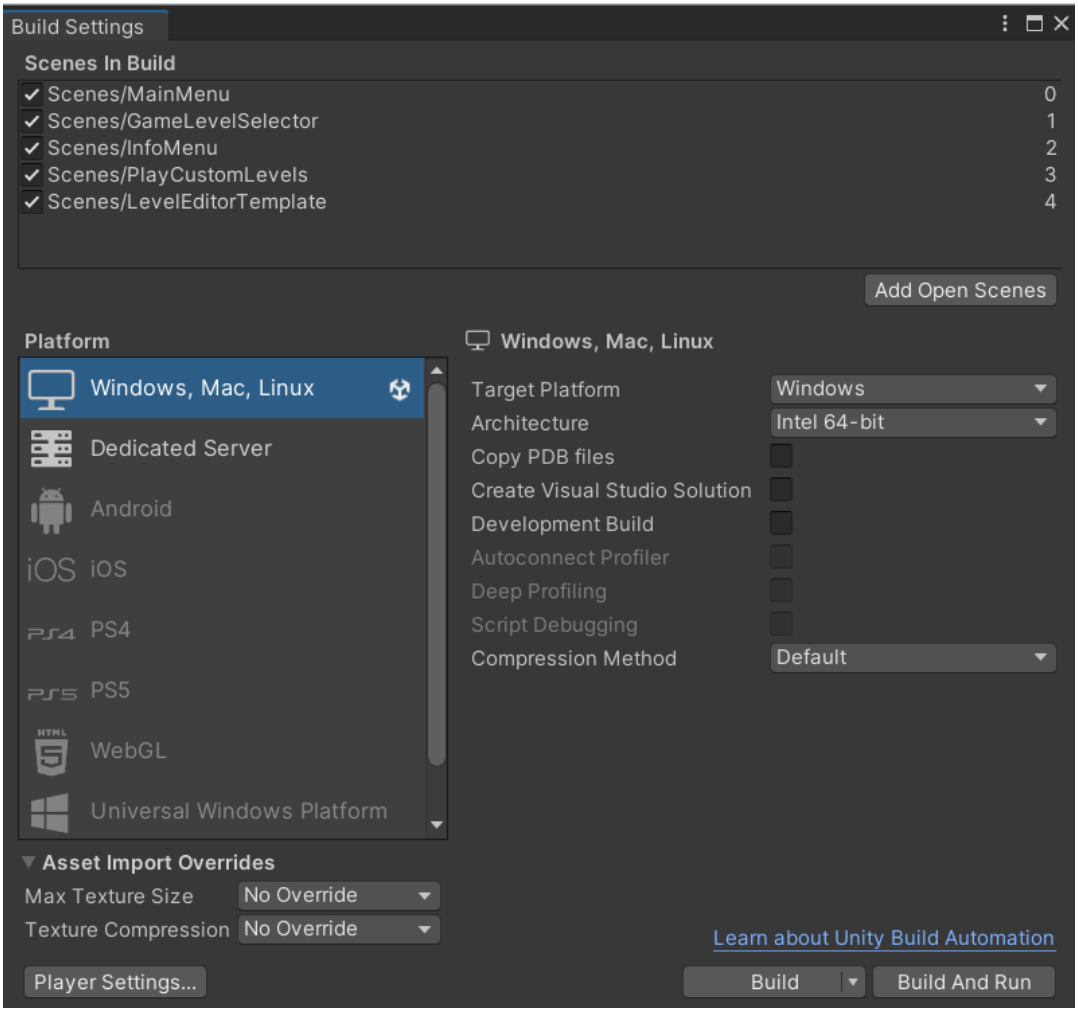


Рисунок 5.2 – Налаштування білду гри

На рисунку 5.2 можемо побачити налаштування сцени, які увійдуть до білду та послідовність самих сцен. Далі натискаємо кнопку “Build” та обираємо директорію, в яку буде запаковано наш білд (рисунок 5.3).

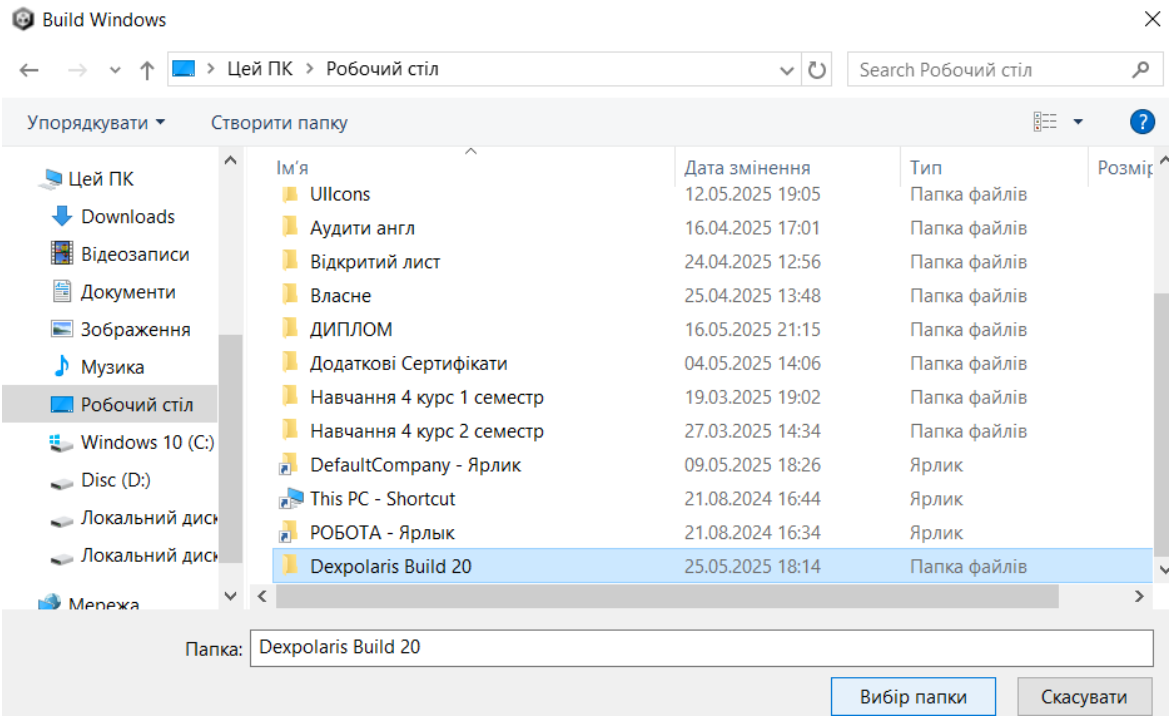


Рисунок 5.3 – Вибір папки директорії гри

Обравши папку для білду, почнеться процес пакування проєкту (рисунок 5.4). В перший раз процес пакування може зайняти кілька хвилин, в подальші рази він займає менше 30 секунд.

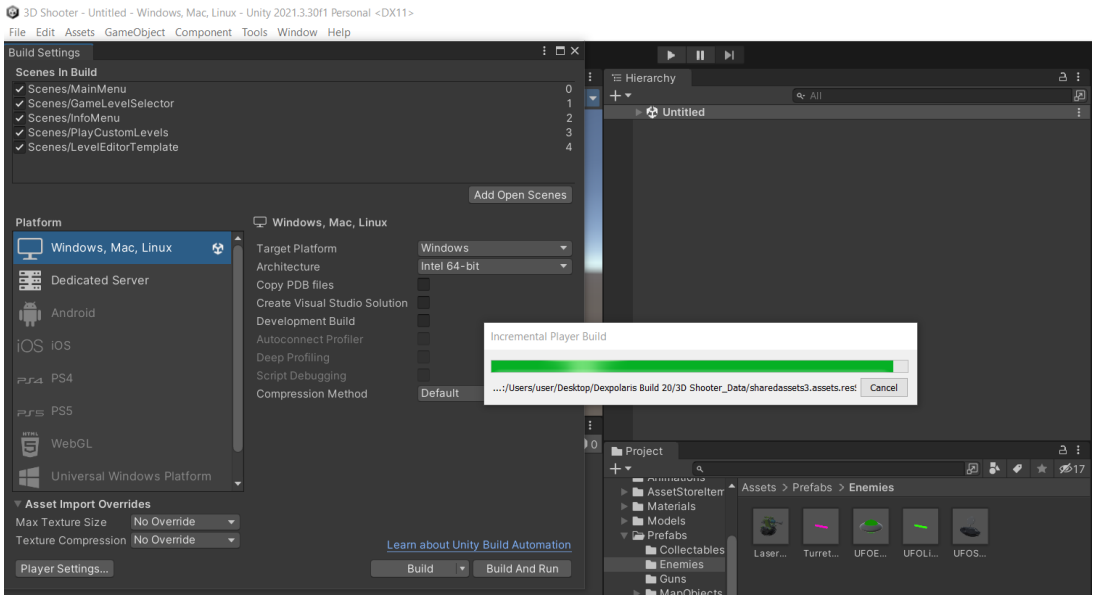


Рисунок 5.4 – Процес пакування проєкту

Після завершення пакування, в директорії, яку вказували як місце для пакування гри, з’являться усі системні файли гри, необхідні для її запуску (рисунок 5.5). Застосунок “3D Shooter.exe” є виконавчим файлом відеогри.

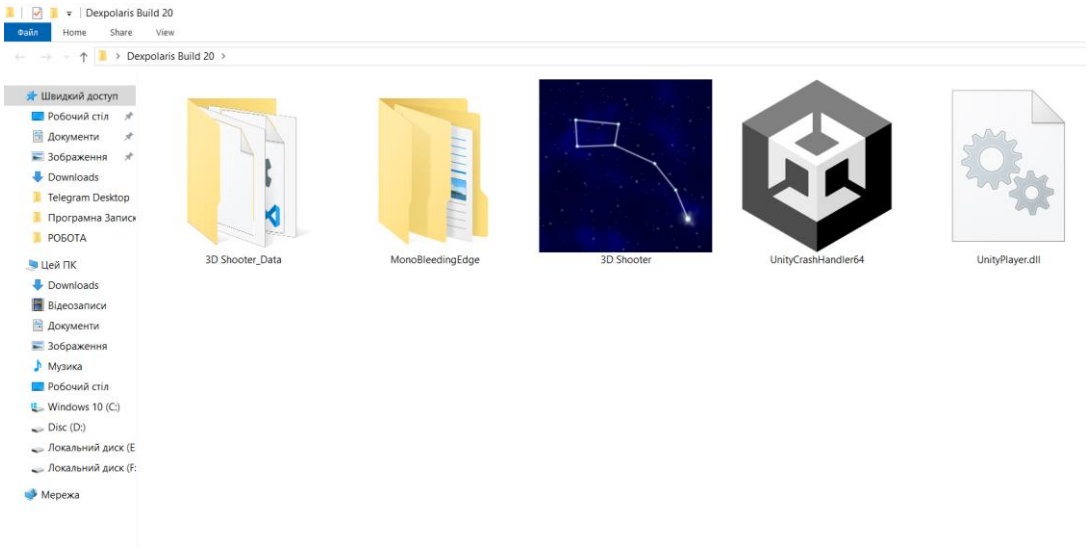


Рисунок 5.5 – Файли проєкту

5.2 Супровід програмного забезпечення

В контексті забезпечення ефективної підтримки та надання оновлень гри, було визначено стратегічний підхід, який передбачає використання Git [27] для керування версіями та публікацію гри на платформі itch.io [28]. Цей підхід спрямований на забезпечення зручного та ефективного механізму розгортання оновлень, а також сприяє простоті взаємодії з нашими користувачами.

Для початку, гра розміщується на GitHub у вигляді репозиторію (рисунок 5.6). Це включає всі вихідні файли, ресурси та код, що дозволяє ефективно керувати проектом та внести зміни за допомогою Git, який надає зручний інтерфейс для версіонування.

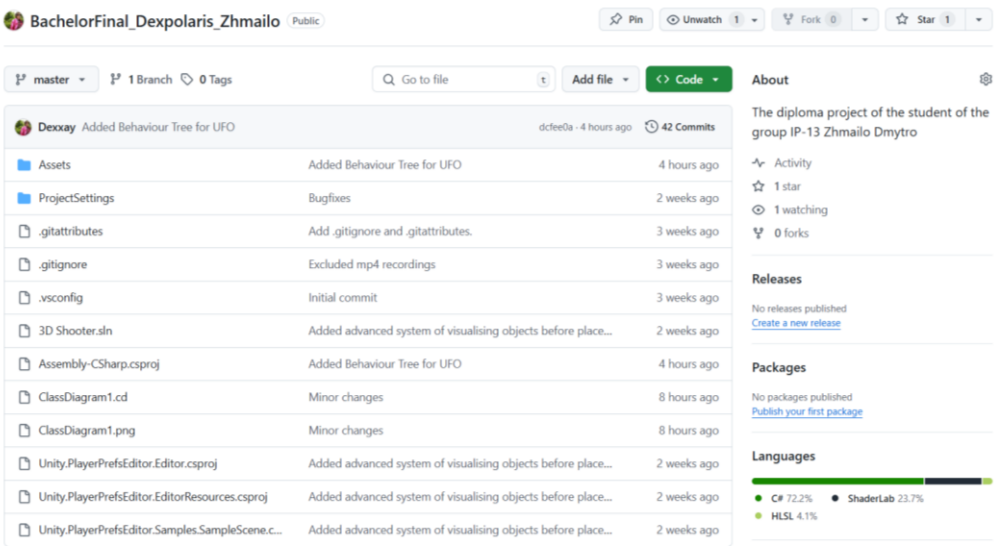


Рисунок 5.6 – Репозиторій гри на платформі GitHub

Після внесення змін у гру, буде оновлюватися версія гри та ставитись відповідні теги у репозиторії Git. Цей етап гарантує, що ми зможемо ефективно відстежувати зміни та швидко реагувати на виявлені помилки або впроваджувати нові функції.

Наступним етапом є публікація гри на платформі itch.io. itch.io - це онлайн-платформа для дистрибуції та продажу відеоігор, програмного забезпечення та інших творчих проектів (рисунок 5.7). Вона надає можливість незалежним розробникам та творчим особам самостійно публікувати свої творіння, встановлювати ціни або визначати форми безкоштовного доступу до свого контенту. Перевагою даної платформи є те, що публікація гри на itch.io є безкоштовною для розробника

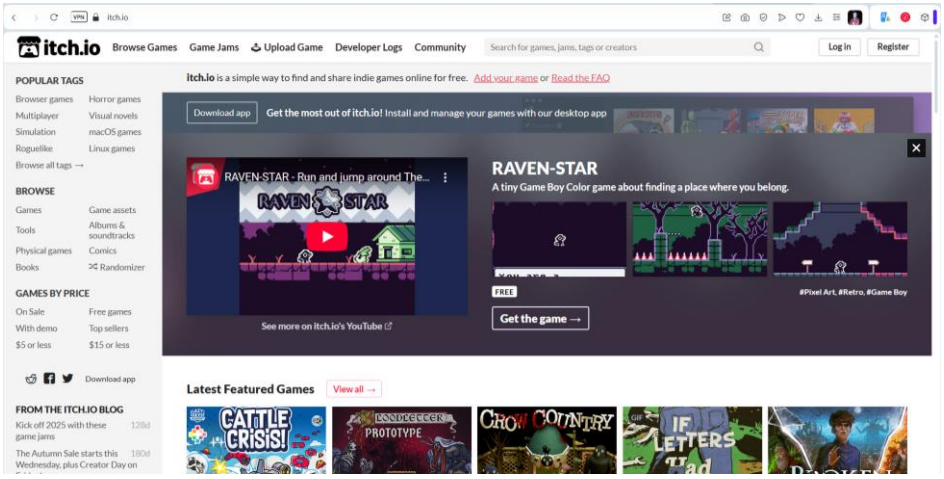


Рисунок 5.7 – Головна сторінка платформи itch.io

Перед публікацією нової версії гри, буде створюватися завантажувальний архів, який включає всі необхідні файли та ресурси. Потім буде створюватися новий реліз на itch.io, де буде вказуватися версія, опис зміни та посилання на завантажувальний архів. Це забезпечує користувачам легкий доступ до оновлень та нововведень гри.

Висновки до розділу

У п’ятому розділі було описано процес розгортання програмного забезпечення, що включав фінальне пакування гри в Unity Editor для створення виконавчих файлів гри, готових до дистрибуції.

Розглянуто стратегію підтримки проєкту, що охоплює оновлення, виправлення помилок та взаємодію з користувачами. Для ефективного керування версіями коду обрано систему Git, яка забезпечує повну історію змін. Розміщення гри реалізовано через безкоштовну платформу itch.io, що надає зручний доступ до продукту для широкої аудиторії.

Такий підхід дозволяє зручно керувати версіями гри, швидко доставляти оновлення користувачам і підтримувати зворотний зв’язок з аудиторією, що допомагає ефективніше розвивати гру.

ВИСНОВКИ

В ході виконання дипломного проєкту, було розроблено гру жанру 3D Action з елементами просторової навігації з функціоналом ігрової кампанії, інтегрованого редактору для створення користувацьких рівнів, бойових механік, системи бонусів та штучного інтелекту ворогів, побудованого на основі алгоритму дерева прийняття рішень.

Мета проєкту була досягнута повністю. Штучний інтелект ворогів на основі алгоритму дерева прийняття рішень підвищив якість ігрового процесу, шляхом створення цікавої поведінки ворогів НЛО, які заохочують гравця до більшої взаємодії з ворогами на ігрових рівнях.

Всі поставлені задачі в рамках дипломного проєктування виконані. Ігровий процес повністю реалізований від початку гри, і до її кінця, з усіма можливими розгалуження. Гравець має повний контроль над своїм персонажем, його озброєнням, може підбирати різні бонуси на рівні. Вороги з’являються на рівнях, взаємодіють з гравцем, використовуючи алгоритм дерева прийняття рішень. Створено три рівні кампанії, які послідовно відкриваються для гравця, забезпечено роботу редактора рівнів та можливість гри на власних рівнях.

У процесі роботи проведено детальний аналіз предметної області, сформульовано вимоги до програмного забезпечення, спроектовано його архітектуру. Програмне забезпечення пройшло тестування та аналіз якості коду, що підтвердило його стабільність та відповідність технічним вимогам. Було створено супровідну документація, зокрема інструкцію користувача, для спрощення освоєння гри для нових гравців.

Перспективними напрямками подальшої розробки є розширення функціоналу штучного інтелекту, додавання нових типів ворогів та бонусів, впровадження процедурної генерації рівнів, розробка режиму гри для кількох гравців, а також адаптація гри під мобільні платформи для розширення аудиторії користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Video Games in the 21st Century: The 2024 Economic Impact Report [Електронний ресурс]. - Entertainment Software Association, 2024. - Режим доступу: https://www.theesa.com/wp-content/uploads/2024/02/EIR_ESA_2024.pdf (дата звернення: 01.06.2025).

2. User-Generated Content and Editors in Video Games [Електронний ресурс]. - IEEE Conference on Games, 2022. - Режим доступу: https://ieee-cog.org/2022/assets/papers/paper_174.pdf (дата звернення: 01.06.2025).

3. The Father of the Video Game: The Ralph Baer Prototypes and Electronic Games [Електронний ресурс]. - Режим доступу: https://www.si.edu/spotlight/the-father-of-the-video-game-the-ralph-baer-prototypes-and-electronic-games/video-game-history (дата звернення: 16.05.2025).

4. Game Industry Trends [Електронний ресурс]. - Режим доступу: https://games.logrusit.com/en/news/game-industry-trends/ (дата звернення: 16.05.2025).

5. The importance of freedom in video games: Reader’s Feature [Електронний ресурс]. - Режим доступу: https://metro.co.uk/2019/05/12/the-importance-of-freedom-in-video-games-readers-feature-9499090/ (дата звернення: 16.05.2025).

6. AI Applications: AI & Video Games [Електронний ресурс]. - Режим доступу: https://ai.engineering.columbia.edu/ai-applications/ai-video-games/ (дата звернення: 16.05.2025).

7. Clustertruck [Електронний ресурс]. - Режим доступу: https://store.steampowered.com/app/397950/Clustertruck/ (дата звернення: 16.05.2025).

8. Зображення з гри Clustertruck [Електронний ресурс]. - Режим доступу: https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/397950/ss_bc11

6108727199df67c5afbbfad8d0661a14ac9d.600x338.jpg?t=1739397318 (дата звернення: 16.05.2025).

9. Roboquest [Електронний ресурс]. - Режим доступу: https://store.steampowered.com/app/692890/Roboquest/ (дата звернення: 01.06.2025).

10. Зображення з гри Roboquest [Електронний ресурс]. - Режим доступу: https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/692890/ss_b32ae626ffecd9fe70f7ea4ddbf34485662bb71d.600x338.jpg?t=1748353552 (дата звернення: 01.06.2025).

11. Jagdale D. Finite State Machine in Game Development [Електронний ресурс] // International Journal of Advanced Research in Science, Communication and Technology (IJARSCT). – 2021. – Vol. 10, Issue 1. – Режим доступу: https://www.researchgate.net/publication/355518086_Finite_State_Machine_in_Game_Development (дата звернення: 16.05.2025).

12. AI Decision Making [Електронний ресурс]. - Режим доступу: https://www.cs.umd.edu/class/spring2018/cmsc425/Lects/lect21-ai-dec-making.pdf (дата звернення: 16.05.2025).

13. Behavior Trees for AI: How They Work [Електронний ресурс]. - Режим доступу: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work (дата звернення: 16.05.2025).

14. Chaudhari V. Goal-Oriented Action Planning [Електронний ресурс]. - Режим доступу: https://medium.com/@vedantchaudhari/goal-oriented-action-planning-34035ed40d0b (дата звернення: 16.05.2025).

15. Stone P. Reinforcement Learning // Encyclopedia of Machine Learning and Data Mining. – Springer, 2017.

16. Game Programming Patterns [Електронний ресурс]. - Режим доступу: https://gameprogrammingpatterns.com (дата звернення: 16.05.2025).

17. Game Engine Popularity in 2024 [Електронний ресурс]. - Режим доступу: https://gamefromscratch.com/game-engine-popularity-in-2024/ (дата звернення: 16.05.2025).

18. Unreal Engine [Електронний ресурс]. - Режим доступу: https://www.unrealengine.com/en-US (дата звернення: 01.06.2025).

19. Godot Engine [Електронний ресурс]. - Режим доступу: https://godotengine.org (дата звернення: 01.06.2025).

20. Unity [Електронний ресурс]. - Режим доступу: https://unity.com (дата звернення: 01.06.2025).

21. Visual Studio: IDE and Code Editor from Microsoft [Електронний ресурс]. - Режим доступу: https://visualstudio.microsoft.com (дата звернення: 25.05.2025).

22. Unity Documentation [Електронний ресурс]. - Режим доступу: https://docs.unity.com (дата звернення: 25.05.2025).

23. Paint.NET [Електронний ресурс]. - Режим доступу: https://www.getpaint.net (дата звернення: 25.05.2025).

24. Audacity: Free, open source, cross-platform audio software [Електронний ресурс]. - Режим доступу: https://www.audacityteam.org (дата звернення: 25.05.2025)

25. Code metrics values - Visual Studio [Електронний ресурс]. - Режим доступу: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022 (дата звернення: 25.05.2025).

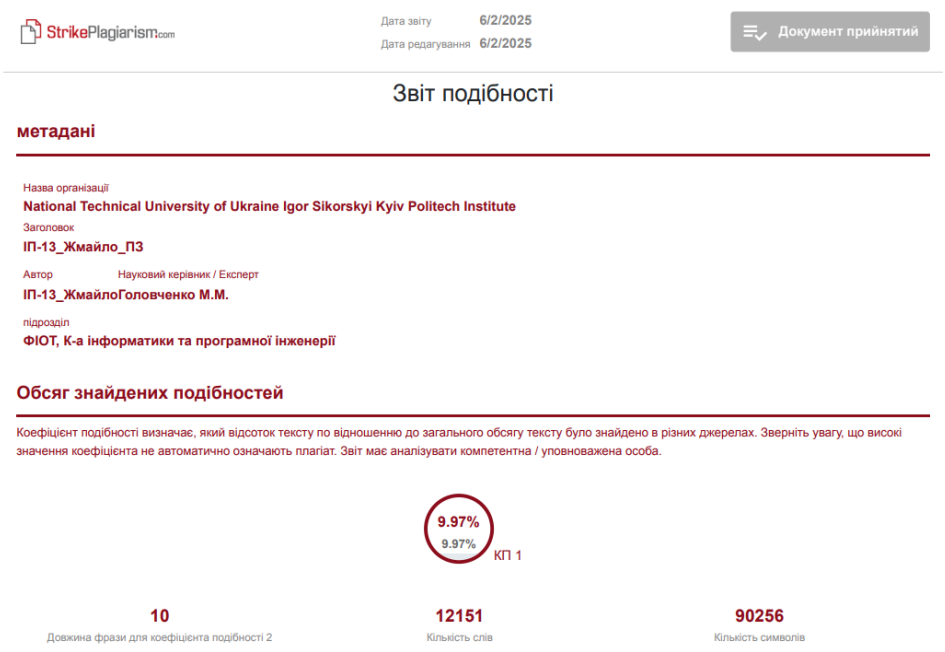
26. FPS Monitor - інструмент для моніторингу продуктивності ігор [Електронний ресурс]. - Режим доступу: https://fpsmon.com/en/ (дата звернення: 25.05.2025).

27. GitHub: платформа для розміщення та спільної розробки програмного коду [Електронний ресурс]. - Режим доступу: https://github.com (дата звернення: 25.05.2025).

28. Itch.io: платформа для розповсюдження інді-ігор [Електронний ресурс]. - Режим доступу: https://itch.io (дата звернення: 25.05.2025).

ДОДАТКИ

ДОДАТОК А ЗВІТ ПОДІБНОСТІ



StrikePlagiarism.com

Дата звіту 6/2/2025
Дата редагування 6/2/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-13_Жмайло_ПЗ
Автор Науковий керівник / Експерт
ІП-13_ЖмайлоГоловченко М.М.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

9.97%
9.97%
КП 1

10
Довжина фрази для коефіцієнта подібності 2

12151
Кількість слів

90256
Кількість символів

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ГРА У ЖАНРІ 3D ACTION З ЕЛЕМЕНТАМИ ПРОСТОРОВОЇ НАВІГАЦІЇ

Текст програми

КПІ.ІП-1311.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро ЖМАЙЛО

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/Dexxay/BachelorFinal_Dexpolaris_Zhmailo

Реалізація функціональної задачі “Забезпечення функціонування ігрового процесу”
Файл MainPlayer.cs

```
using UnityEngine;

public class MainPlayer : MonoBehaviour
{
    [SerializeField] private WeaponManager weaponManager;
    [SerializeField] private PlayerHealthManager playerHealthManager;
    [SerializeField] private PlayerMovement playerMovement;
    [SerializeField] private MouseMovement mouseMovement;
    [SerializeField] private ScoreManager scoreManager;
    [SerializeField] private CharacterController characterController;
    [SerializeField] private MapTimer mapTimer;

    public WeaponManager WeaponManager => weaponManager;
    public PlayerHealthManager PlayerHealthManager => playerHealthManager;
    public PlayerMovement PlayerMovement => playerMovement;
    public MouseMovement MouseMovement => mouseMovement;
    public ScoreManager ScoreManager => scoreManager;
    public CharacterController CharacterController => characterController;
    public MapTimer MapTimer => mapTimer;

    private void Start()
    {
        PlayerHealthManager.PlayerDied += OnPlayerDied;
        PauseManager.Instance.PauseStarted += OnPauseStarted;
        PauseManager.Instance.PauseEnded += OnPauseEnded;
    }


    private void OnPauseStarted()
    {
        scoreManager.enabled = false;
        playerMovement.enabled = false;
        mouseMovement.enabled = false;
        characterController.enabled = false;
        weaponManager.SelectedWeapon.gameObject.SetActive(false);
        weaponManager.enabled = false;
    }
```

```
    private void OnPauseEnded()
    {
        scoreManager.enabled = true;
        playerMovement.enabled = true;
        mouseMovement.enabled = true;
        characterController.enabled = true;
        weaponManager.SelectedWeapon.gameObject.SetActive(true);
        weaponManager.enabled = true;
    }


    private void OnPlayerDied()
    {
        scoreManager.enabled = false;
        playerMovement.enabled = false;
        mouseMovement.enabled = false;
        characterController.enabled = false;
        weaponManager.SelectedWeapon.gameObject.SetActive(false);
        weaponManager.enabled = false;
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;

        Time.timeScale = 0;
    }

    public void OnPlayerWon()
    {
        scoreManager.enabled = false;
        playerMovement.enabled = false;
        mouseMovement.enabled = false;
        characterController.enabled = false;
        weaponManager.SelectedWeapon.gameObject.SetActive(false);
        weaponManager.enabled = false;
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;

        Time.timeScale = 0;
    }
}
```

Файл MouseMovement.cs

```
using UnityEngine;

public class MouseMovement : MonoBehaviour
{
```

```
    [SerializeField] private float mouseXSensetivity = 500f;
    [SerializeField] private float mouseYSensetivity = 500f;
    [SerializeField] bool invertMouse = true;
    [Space]
    [SerializeField] private float topClamp = -90f;
    [SerializeField] private float bottomClamp = 90f;

    private float xRotation = 0f;
    private float yRotation = 0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;

        xRotation = Camera.main.transform.localEulerAngles.x;
        if (xRotation > 180) xRotation -= 360;

        yRotation = transform.localEulerAngles.y;
    }

    void Update()
    {
        float mouseX = Input.GetAxis("Mouse X") * mouseXSensetivity *
Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseYSensetivity *
Time.deltaTime;

        if (invertMouse)
        {
            mouseY *= -1;
        }

        xRotation += mouseY;
        xRotation = Mathf.Clamp(xRotation, topClamp, bottomClamp);

        yRotation += mouseX;

        Camera.main.transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        transform.localRotation = Quaternion.Euler(0f, yRotation, 0f);
    }

    public void ForceLookRotation(Quaternion cameraRotation, Quaternion
bodyRotation)
    {
        Camera.main.transform.rotation = cameraRotation;
```

```
        transform.rotation = bodyRotation;

        Vector3 camEuler = Camera.main.transform.localEulerAngles;
        xRotation = camEuler.x;
        if (xRotation > 180) xRotation -= 360;

        yRotation = transform.localEulerAngles.y;
    }
}
```

Файл PlayerMovement.cs

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    private CharacterController characterController;

    [Header("Налаштування руху")]
    [SerializeField] private float moveSpeed = 12f;
    [SerializeField] private float sprintMultiplier = 1.5f;
    [SerializeField] private float gravityForce = -20f;
    [SerializeField] private float jumpHeight = 3f;

    [Header("Перевірка на землю")]
    [SerializeField] private float groundDistance = 0.4f;
    [SerializeField] private Transform groundCheck;
    [SerializeField] private LayerMask groundMask;

    private bool isGrounded;
    private Vector3 velocity;

    private float coyoteTime = 0.2f;
    private float coyoteTimeCounter;

    private readonly KeyCode jumpKey = KeyCode.Space;
    private readonly KeyCode sprintKey = KeyCode.LeftShift;

    void Start()
    {
        characterController = GetComponent<CharacterController>();
    }

    void Update()
    {
        HandleMovement();
```

```
        HandleJumpAndGravity();
    }

    private void HandleMovement()
    {
        float xInput = Input.GetAxis("Horizontal");
        float zInput = Input.GetAxis("Vertical");

        Vector3 move = transform.right * xInput + transform.forward * zInput;
        move.y = 0f;

        float currentSpeed = moveSpeed;

        if (Input.GetKey(sprintKey))
        {
            currentSpeed *= sprintMultiplier;
        }

        Vector3 moveVector = move * currentSpeed;

        if (moveVector.magnitude > currentSpeed)
            moveVector = moveVector.normalized * currentSpeed;

        characterController.Move(moveVector * Time.deltaTime);
    }

    private void HandleJumpAndGravity()
    {
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance,
groundMask);

        if (isGrounded)
        {
            coyoteTimeCounter = coyoteTime;
            if (velocity.y < 0f)
            {
                velocity.y = -2f;
            }
        }
        else
        {
            coyoteTimeCounter -= Time.deltaTime;
        }

        if (Input.GetKeyDown(jumpKey) && coyoteTimeCounter > 0f)
```

```
        {
            velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravityForce);
            coyoteTimeCounter = 0f;
        }

        if (!isGrounded)
        {
            velocity.y += gravityForce * Time.deltaTime;
        }

        characterController.Move(velocity * Time.deltaTime);
    }


}
```

Реалізація функціональної задачі “Забезпечення функціонування притаманних ігрових механік”

Файл ActionNode.cs

```
using System.Collections;
using UnityEngine;

public abstract class ActionNode : ScriptableObject
{
    public abstract IEnumerator Execute(UFOBehaviour ufo);
}
```

Файл **DecisionNode.cs**

```
using UnityEngine;
public abstract class DecisionNode : ScriptableObject
{
    public abstract ActionNode MakeDecision(UFOBehaviour ufo);
}
```

Файл **UFOBehaviour.cs**

```
void Awake()
    {
        InitializeComponents();
    }

    void Start()
    {
        InitializeBehavior();
        StartDecisionMaking();
    }
```

```
    private void InitializeComponents()
    {
        rb = GetComponent<Rigidbody>();
        if (rb == null)
        {
            Debug.LogError("UFO_AI requires a Rigidbody component on " + gameObject.name);
            enabled = false;
            return;
        }
        rb.useGravity = false;

        if (laserWeapon == null) laserWeapon = GetComponentInChildren<LaserWeapon>();
        if (laserWeapon == null) Debug.LogWarning("LaserWeapon component not found on " + gameObject.name + " or its children. UFO will not be able to attack.");

        if (firePoint == null && laserWeapon != null) firePoint = laserWeapon.transform;
        else if (firePoint == null) firePoint = transform;

        currentHealth = maxHealth;
        spawnPosition = transform.position;
    }

    private void InitializeBehavior()
    {
        mainPlayerTarget = FindFirstObjectByType<MainPlayer>();
        if (mainPlayerTarget != null)
        {
            playerTransform = mainPlayerTarget.transform;
        }

        instanceCirclingDistance = circlingDistance + UnityEngine.Random.Range(-circlingDistanceVariation, circlingDistanceVariation);
        instanceCirclingDistance = Mathf.Max(1f, instanceCirclingDistance);
        instanceRelativeAltitude = relativeAltitude + UnityEngine.Random.Range(-relativeAltitudeVariation, relativeAltitudeVariation);
        instanceRelativeAltitude = Mathf.Max(2f, instanceRelativeAltitude);
        instanceMovementSpeed = movementSpeed + UnityEngine.Random.Range(-movementSpeedVariation, movementSpeedVariation);
        instanceMovementSpeed = Mathf.Max(1f, instanceMovementSpeed);

        SetNewPatrolDestination();
```

```
        lastAttackTime = -attackCooldown;
    }

    private void StartDecisionMaking()
    {
        if (rootDecisionNode == null)
        {
            Debug.LogWarning($"Root Decision Node is NOT assigned for {gameObject.name}. AI will default to Patrol and will not use the Decision Tree.");
            PatrolAction defaultPatrolAction = ScriptableObject.CreateInstance<PatrolAction>();
            currentAction = defaultPatrolAction;
            currentActionCoroutine = StartCoroutine(currentAction.Execute(this));
        }
        else
        {
            StartCoroutine(DecisionTreeCoroutine());
        }
    }

    private IEnumerator DecisionTreeCoroutine()
    {
        if (rootDecisionNode == null)
        {
            if (enableDebugLogs) Debug.LogError($"{gameObject.name}: DecisionTreeCoroutine stopping because rootDecisionNode is null.");
            yield break;
        }

        while (true)
        {
            if (isDead)
            {
                if (enableDebugLogs) Debug.Log($"{gameObject.name}: UFO is dead, stopping DecisionTreeCoroutine.");
                yield break;
            }

            UpdatePlayerTarget();

            ActionNode nextAction = rootDecisionNode.MakeDecision(this);

            if (nextAction == null)
            {
```

```
            if (enableDebugLogs) Debug.LogWarning($"{gameObject.name}: Decision Tree returned a NULL action. Sticking to current action or stopping.");
        }
        else if (nextAction != currentAction)
        {
            TransitionToAction(nextAction);
        }

        if (currentAction is DieAction && !isDead)
        {
            Die();
            yield break;
        }

        yield return new WaitForSeconds(decisionInterval);
    }
}

private void UpdatePlayerTarget()
{
    if (playerTransform == null && mainPlayerTarget == null)
    {
        mainPlayerTarget = FindFirstObjectByType<MainPlayer>();
        if (mainPlayerTarget != null)
        {
            playerTransform = mainPlayerTarget.transform;
        }
    }
}

private void TransitionToAction(ActionNode newAction)
{
    if (enableDebugLogs)
    {
        Debug.Log($"{gameObject.name} state: {(currentAction != null ? currentAction.name : "None")} -> {newAction.name}. " +
                  $"PlayerDist: {(playerTransform ? Vector3.Distance(transform.position, playerTransform.position).ToString("F1") : "N/A")}, " +
                  $"CanAttack: {CanAttack()}, Health: {currentHealth}");
    }

    if (currentActionCoroutine != null)
    {
        StopCoroutine(currentActionCoroutine);
```

```
            currentActionCoroutine = null;
        }

        currentAction = newAction;
        currentActionCoroutine = StartCoroutine(currentAction.Execute(this));
    }
```

Файл **LaserWeapon.cs**

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

public class LaserWeapon : MonoBehaviour
{
    [Header("Laser Settings")]
    [SerializeField] public float laserSpeed = 50f;
    [SerializeField] private int laserDamage = 25;
    [SerializeField] private float accuracy = 2.5f;
    [SerializeField] private float distanceTravelMultiplier = 2f;
    [SerializeField] private LineRenderer laserLinePrefab;
    [SerializeField] private LayerMask groundLayer;

    [Header("Effects")]
    [SerializeField] private ParticleSystem muzzleFlash;
    [SerializeField] private AudioSource audioSource;
    [SerializeField] private AudioClip shotSound;

    private List<GameObject> activeLasers = new();

    public void FireLaser(Vector3 startPosition, Vector3 targetPosition, MainPlayer
playerTarget)
    {
        if (laserLinePrefab == null)
        {
            Debug.LogWarning("LaserLinePrefab is not assigned in LaserWeapon!");
            return;
        }
        if (playerTarget == null)
        {
            Debug.LogWarning("PlayerTarget is null in LaserWeapon.FireLaser!");
            return;
        }


        if (muzzleFlash != null) muzzleFlash.Play();
```

```
        if (audioSource != null && shotSound != null) audioSource.PlayOneShot(shotSound);

        Vector3 direction = (targetPosition - startPosition).normalized;
        float distanceToTarget = Vector3.Distance(startPosition, targetPosition);
        Vector3 extendedTargetPosition = startPosition + direction * (distanceToTarget * distanceTravelMultiplier);

        GameObject laserGO = new GameObject("LaserBeam");

        LineRenderer lr = Instantiate(laserLinePrefab, laserGO.transform);
        lr.positionCount = 2;
        lr.SetPosition(0, startPosition);
        lr.SetPosition(1, startPosition);

        activeLasers.Add(laserGO);

        StartCoroutine(MoveLaser(lr, startPosition, extendedTargetPosition, laserGO, playerTarget));
    }

    private IEnumerator MoveLaser(LineRenderer lr, Vector3 start, Vector3 target, GameObject laserGO, MainPlayer playerTarget)
    {
        float elapsed = 0f;
        float totalDistance = Vector3.Distance(start, target);
        float travelTime = totalDistance / laserSpeed;
        bool hasHitPlayer = false;

        Color startColor = lr.startColor;
        startColor.a = 1f;
        lr.startColor = startColor;
        lr.endColor = startColor;

        lr.startWidth = 0.2f;
        lr.endWidth = 0.2f;

        while (elapsed < travelTime)
        {
            if (laserGO == null || lr == null)
            {
                yield break;
            }

            elapsed += Time.deltaTime;
```

```
        Vector3 currentPosition = Vector3.Lerp(start, target, elapsed / travelTime);
        lr.SetPosition(1, currentPosition);

        float alpha = Mathf.Lerp(1f, 0f, elapsed / travelTime);
        Color currentColor = lr.startColor;
        currentColor.a = alpha;
        lr.startColor = currentColor;
        lr.endColor = currentColor;

        float width = Mathf.Lerp(0.2f, 0f, elapsed / travelTime);
        lr.startWidth = width;
        lr.endWidth = width;

        Vector3 rayDir = (currentPosition - start).normalized;
        float rayDist = Vector3.Distance(start, currentPosition);

        RaycastHit hit;
        if (Physics.Raycast(start, rayDir, out hit, rayDist, groundLayer))
        {
            lr.SetPosition(1, hit.point);

            Vector3 position = hit.point + hit.normal * 0.01f;
            Quaternion rotation = Quaternion.LookRotation(hit.normal);

            if (EffectsManager.Instance != null)
            {
                GameObject bulletHole = EffectsManager.Instance.SpawnBulletHole(position, rotation);
                if (bulletHole != null) bulletHole.transform.SetParent(hit.transform, true);
            }
            else
            {
                Debug.LogWarning("EffectsManager.Instance is null. Cannot spawn bullet hole.");
            }

            Destroy(lr.gameObject);
            laserGO = null;
            break;
        }

        if (!hasHitPlayer && playerTarget != null && Vector3.Distance(currentPosition, playerTarget.transform.position) < accuracy)
        {
```

```
                ApplyDamageToTarget(playerTarget);
                hasHitPlayer = true;
                Destroy(lr.gameObject);
                laserGO = null;
                break;
            }

            yield return null;
        }

        if (laserGO != null)
        {
            activeLasers.Remove(laserGO);
            Destroy(laserGO);
        }
        else if (lr != null && lr.gameObject != null)
        {
            Destroy(lr.gameObject);
        }
    }


    public void ApplyDamageToTarget(MainPlayer playerTarget)
    {
        if (playerTarget != null && playerTarget.PlayerHealthManager != null)
        {
            playerTarget.PlayerHealthManager.ReduceHealth(laserDamage);
        }
        else
        {
            Debug.LogWarning("Cannot apply damage: playerTarget or
PlayerHealthManager is null.");
        }
    }

    public void DestroyAllLasers()
    {
        foreach (var laser in activeLasers)
        {
            if (laser != null)
            {
                foreach (Transform child in laser.transform)
                {
                    Destroy(child.gameObject);
                }
```

```
                Destroy(laser);
            }
        }
        activeLasers.Clear();
    }

    private void OnDestroy()
    {
        DestroyAllLasers();
    }
}
```

Реалізація функціональної задачі “Забезпечення можливості проходження кампанії”

Файл LevelProgressManager.cs

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

public class LevelProgressManager : MonoBehaviour
{
    private static LevelProgressManager instance;

    private const string MaxLevelIndexReachedKey = "MaxLevelIndexReached";
    public static LevelProgressManager Instance => instance;

    public List<string> campaignLevels = new List<string>()
    {
        "campaign_1",
        "campaign_2",
        "campaign_3",
    };

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else if (instance != this)
        {
            Destroy(this);
        }
```

```
        if (campaignLevels == null || campaignLevels.Count == 0)
        {
            Debug.LogError("Campaign levels list is empty in
LevelProgressManager!");
        }

        if (!PlayerPrefs.HasKey(MaxLevelIndexReachedKey))
        {
            PlayerPrefs.SetInt(MaxLevelIndexReachedKey, 0);
            PlayerPrefs.Save();
        }
    }

    public void UnlockNextLevel(string completedLevelName)
    {
        int maxLevelIndexReached = PlayerPrefs.GetInt(MaxLevelIndexReachedKey,
0);

        int completedLevelIndex = campaignLevels.IndexOf(completedLevelName);

        if (completedLevelIndex != -1)
        {
            int nextLevelIndexToUnlock = completedLevelIndex + 1;

            if (nextLevelIndexToUnlock >= campaignLevels.Count)
            {
                Debug.Log("Completed the last campaign level.");
            }
            else
            {
                if (nextLevelIndexToUnlock > maxLevelIndexReached)
                {
                    PlayerPrefs.SetInt(MaxLevelIndexReachedKey,
nextLevelIndexToUnlock);
                    PlayerPrefs.Save();
                    Debug.Log($"Level '{campaignLevels[nextLevelIndexToUnlock]}'
unlocked.");
                }
                else
                {
                    Debug.Log($"Level '{campaignLevels[nextLevelIndexToUnlock]}' is
already unlocked.");
                }
            }
        }
```

```
        else
        {
            Debug.LogWarning($"Completed level '{completedLevelName}' not found
in the campaign levels list.");
        }
    }

    public bool IsLevelUnlocked(string levelName)
    {
        if (campaignLevels == null || campaignLevels.Count == 0)
        {
            Debug.LogError("Campaign levels list is empty in
LevelProgressManager!");
            return false;
        }

        int levelIndex = campaignLevels.IndexOf(levelName);
        if (levelIndex == -1)
        {
            Debug.LogWarning($"Level '{levelName}' not found in the campaign levels
list.");
            return true;
        }

        int maxLevelIndexReached = PlayerPrefs.GetInt(MaxLevelIndexReachedKey,
0);

        return levelIndex <= maxLevelIndexReached;
    }

    public List<string> GetCampaignLevels()
    {
        return campaignLevels;
    }

    public int GetMaxLevelIndexReached()
    {
        return PlayerPrefs.GetInt(MaxLevelIndexReachedKey, 0);
    }
    public void ResetProgress()
    {
        PlayerPrefs.DeleteKey(MaxLevelIndexReachedKey);
        PlayerPrefs.SetInt(MaxLevelIndexReachedKey, 0);
        PlayerPrefs.Save();
        Scene scene = SceneManager.GetActiveScene();
```

```
            SceneManager.LoadScene(scene.name);
        }

    }
```

Реалізація функціональної задачі “Забезпечення можливості створення користувацьких рівнів”

Файл LevelEditor.cs

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

public class LevelEditor : MonoBehaviour
{
    [Header("Object Prefabs")]
    [SerializeField] private GameObject startAsteroidPrefab;
    [SerializeField] private List<GameObject> randomAsteroidPrefabs = new List<GameObject>();
    [SerializeField] private GameObject plasmaGunAsteroidPrefab;
    [SerializeField] private GameObject laserSMGAsteroidPrefab;
    [SerializeField] private GameObject ammo1AsteroidPrefab;
    [SerializeField] private GameObject ammo2AsteroidPrefab;
    [SerializeField] private GameObject healthPackPrefab;
    [SerializeField] private GameObject turretPrefab;
    [SerializeField] private GameObject ufoSpawnerPrefab;
    [SerializeField] private GameObject finishAsteroidPrefab;

    [Header("Editor Settings")]
    [SerializeField] private GameObject levelObjectsParent;
    [SerializeField] private float gridSpacing = 1f;
    [SerializeField] private float randomHeightRange = 0.5f;
    [SerializeField] private Collider editorPlaneCollider;
    [SerializeField] private string mainMenuSceneName = "MainMenu";

    [Header("Time Limit Settings")]
    [SerializeField] private float minSliderTimeLimit = 60f;
    [SerializeField] private float maxSliderTimeLimit = 180f;

    [HideInInspector] public GameObject startAsteroidInstance = null;
    [HideInInspector] public GameObject finishAsteroidInstance = null;

    private EditorCameraMovement cameraMovement;
    private EditorObjectPlacement objectPlacement;
    [SerializeField] private EditorUIHandler uiHandler;
```

```
    private LevelSaveLoadManager saveLoadManager;

    private string saveSlot1 = "level_slot_1.dat";
    private string saveSlot2 = "level_slot_2.dat";
    private string saveSlot3 = "level_slot_3.dat";

    private float currentLevelTimeLimit = 60f;

    public float GridSpacing { get { return gridSpacing; } }
    public float RandomHeightRange { get { return randomHeightRange; } }
    public Collider EditorPlaneCollider { get { return editorPlaneCollider; } }
    public EditorUIHandler UiHandler { get { return uiHandler; } }
    public GameObject LevelObjectsParent { get { return levelObjectsParent; } }
    public float MinSliderTimeLimit { get { return minSliderTimeLimit; } }
    public float MaxSliderTimeLimit { get { return maxSliderTimeLimit; } }


    void Awake()
    {
        cameraMovement = GetComponent<EditorCameraMovement>();
        objectPlacement = GetComponent<EditorObjectPlacement>();
        saveLoadManager = GetComponent<LevelSaveLoadManager>();

        if (cameraMovement == null) Debug.LogError("EditorCameraMovement not
found!");
        if (objectPlacement == null) Debug.LogError("EditorObjectPlacement not
found!");
        if (uiHandler == null) Debug.LogError("EditorUIHandler not assigned in the
Inspector!");
        if (saveLoadManager == null) Debug.LogError("LevelSaveLoadManager not
found!");

        if (uiHandler != null) uiHandler.Init(this);
        if (cameraMovement != null && editorPlaneCollider != null && Camera.main
!= null) cameraMovement.Init(Camera.main, editorPlaneCollider);
        else if (cameraMovement != null && editorPlaneCollider == null)
Debug.LogError("EditorPlaneCollider is not assigned for EditorCameraMovement
in the Inspector!");

        if (objectPlacement != null && Camera.main != null)
objectPlacement.Init(this, Camera.main);


        if (levelObjectsParent == null)
        {
```

```
            Debug.LogWarning("levelObjectsParent was not assigned in the Inspector.
Creating a new one.");
            levelObjectsParent = new GameObject("LevelObjectsParent");
        }
    }

    void Update()
    {
        if (uiHandler == null) return;

        bool isConfirmationDialogOpen = uiHandler.IsConfirmationDialogOpen();

        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (!isConfirmationDialogOpen)
            {
                uiHandler.TogglePauseMenu();
            }
        }

        if (uiHandler.IsPauseMenuOpen() || isConfirmationDialogOpen)
        {
            return;
        }

        if (cameraMovement != null) cameraMovement.HandleInput();
    }

    public float GetTimeLimit()
    {
        return currentLevelTimeLimit;
    }

    public void SetTimeLimit(float newTimeLimit)
    {
        currentLevelTimeLimit = Mathf.Clamp(newTimeLimit, minSliderTimeLimit,
maxSliderTimeLimit);
        if (uiHandler != null)
        {
            uiHandler.UpdateTimeLimitText(currentLevelTimeLimit);
        }
    }

    public void ShowMessage(string message, bool isError)
    {
```

```
        if (uiHandler == null)
        {
            Debug.LogError("uiHandler is not assigned. Cannot show message.");
            return;
        }

        if (isError)
        {
            uiHandler.SetRedColor();
        }
        else
        {
            uiHandler.SetGreenColor();
        }
        uiHandler.ShowError(message);
    }

    public void ClearMessage()
    {
        if (uiHandler != null)
        {
            uiHandler.ClearMessage();
        }
    }

    public void GoToMainMenuConfirmed()
    {
        Time.timeScale = 1f;
        if (string.IsNullOrEmpty(mainMenuSceneName))
        {
            Debug.LogError("mainMenuSceneName is not set in the Inspector. Cannot
load Main Menu. Loading scene with index 0 as a fallback.");
            SceneManager.LoadScene(0);
        }
        else
        {
            SceneManager.LoadScene(mainMenuSceneName);
        }
    }

    public void SaveLevelConfirmed(int slot)
    {
        if (saveLoadManager == null)
        {
            ShowMessage("Error: SaveLoadManager is not configured.", true);
```

```
            return;
        }
        if (levelObjectsParent == null)
        {
            ShowMessage("Error: levelObjectsParent is not configured.", true);
            return;
        }


        if (startAsteroidInstance == null)
        {
            ShowMessage("Error: Start asteroid is not placed.", true);
            return;
        }
        if (finishAsteroidInstance == null)
        {
            ShowMessage("Error: Finish asteroid is not placed.", true);
            return;
        }

        string filePath = GetSavePathBySlot(slot);
        if (string.IsNullOrEmpty(filePath))
        {
            ShowMessage($"Error: Could not determine save path for slot {slot}.", true);
            return;
        }
        saveLoadManager.SaveLevel(filePath,                levelObjectsParent,
currentLevelTimeLimit, startAsteroidInstance, finishAsteroidInstance);
        ShowMessage($"Level saved to slot {slot}.", false);
    }

    public void LoadLevelConfirmed(int slot)
    {
        if (saveLoadManager == null)
        {
            ShowMessage("Error: SaveLoadManager is not configured.", true);
            return;
        }
        if (levelObjectsParent == null)
        {
            ShowMessage("Error: levelObjectsParent is not configured.", true);
            return;
        }

        ClearAllObjectsConfirmedInternal();
```

```
        string filePath = GetSavePathBySlot(slot);
        if (string.IsNullOrEmpty(filePath))
        {
            ShowMessage($"Error: Could not determine load path for slot {slot}.", true);
            return;
        }

        LevelData loadedData = saveLoadManager.LoadLevelForEditor(filePath,
levelObjectsParent, this);
        if (loadedData != null)
        {
            currentLevelTimeLimit = loadedData.timeLimit;
            if (uiHandler != null)
uiHandler.UpdateTimeLimitText(currentLevelTimeLimit);

            startAsteroidInstance = null;
            finishAsteroidInstance = null;

            foreach (Transform child in levelObjectsParent.transform)
            {
                PlacedObjectInfo objInfo = child.GetComponent<PlacedObjectInfo>();
                if (objInfo != null &&
!string.IsNullOrEmpty(objInfo.originalPrefabName))
                {
                    if (startAsteroidPrefab != null && objInfo.originalPrefabName ==
startAsteroidPrefab.name)
                    {
                        startAsteroidInstance = child.gameObject;
                    }
                    if (finishAsteroidPrefab != null && objInfo.originalPrefabName ==
finishAsteroidPrefab.name)
                    {
                        finishAsteroidInstance = child.gameObject;
                    }
                }
            }
            ShowMessage($"Level loaded from slot {slot}.", false);
        }
        else
        {
            ShowMessage($"Error loading level from slot {slot}. File might not exist or
is corrupted.", true);
        }
    }
```

```
    public void DeleteLevelConfirmed(int slot)
    {
        if (saveLoadManager == null)
        {
            ShowMessage("Error: SaveLoadManager is not configured.", true);
            return;
        }
        string filePath = GetSavePathBySlot(slot);
        if (string.IsNullOrEmpty(filePath))
        {
            ShowMessage($"Error: Could not determine path for slot {slot} to delete.",
true);
            return;
        }

        if (saveLoadManager.DeleteLevel(filePath))
        {
            ShowMessage($"Level in slot {slot} deleted.", false);
        }
        else
        {
            ShowMessage($"Failed to delete level in slot {slot}. File may not exist.",
true);
        }
    }

    private void ClearAllObjectsConfirmedInternal()
    {
        if (levelObjectsParent == null)
        {
            Debug.LogWarning("levelObjectsParent is null. Cannot clear objects.");
            return;
        }

        foreach (Transform child in levelObjectsParent.transform)
        {
            Destroy(child.gameObject);
        }
        startAsteroidInstance = null;
        finishAsteroidInstance = null;
        ShowMessage("All objects cleared.", false);
    }

    public void ClearAllObjectsConfirmed()
    {
```

```
        ClearAllObjectsConfirmedInternal();
    }

    public GameObject GetPrefabByType(int objectType)
    {
        switch (objectType)
        {
            case 1: return startAsteroidPrefab;
            case 2:
                return randomAsteroidPrefabs != null && randomAsteroidPrefabs.Count
> 0 ? randomAsteroidPrefabs[0] : null;
            case 3: return plasmaGunAsteroidPrefab;
            case 4: return laserSMGAsteroidPrefab;
            case 5: return ammo1AsteroidPrefab;
            case 6: return ammo2AsteroidPrefab;
            case 7: return healthPackPrefab;
            case 8: return turretPrefab;
            case 9: return ufoSpawnerPrefab;
            case 0: return finishAsteroidPrefab;
            default:
                Debug.LogWarning($"Unknown object type: {objectType}");
                return null;
        }
    }

    public GameObject GetRandomAsteroidPrefab()
    {
        if (randomAsteroidPrefabs != null && randomAsteroidPrefabs.Count > 0)
        {
            return                randomAsteroidPrefabs[UnityEngine.Random.Range(0,
randomAsteroidPrefabs.Count)];
        }
        else
        {
            Debug.LogWarning("Random asteroid prefabs list is empty or null.");
            return null;
        }
    }

    public bool CheckIfCanPlace(Vector3 placementPosition, GameObject
prefabToSpawn, GameObject objectToIgnore = null)
    {
        if (LevelObjectsParent == null)
        {
```

```
            Debug.LogError("LevelObjectsParent in LevelEditor is not assigned during placement check.");
            return false;
        }

        ObjectRadius newObjectRadiusComponent = prefabToSpawn.GetComponent<ObjectRadius>();
        float newObjectRadius = newObjectRadiusComponent != null ? newObjectRadiusComponent.Radius : 1f;

        if (prefabToSpawn == startAsteroidPrefab && startAsteroidInstance != null && startAsteroidInstance != objectToIgnore)
        {
            ShowMessage("There is a start asteroid on a map already. Delete old one to place a new one", true);
            return false;
        }
        if (prefabToSpawn == finishAsteroidPrefab && finishAsteroidInstance != null && finishAsteroidInstance != objectToIgnore)
        {
            ShowMessage("There is a finish asteroid on a map already. Delete old one to place a new one", true);
            return false;
        }

        bool canPlace = true;

        foreach (Transform child in LevelObjectsParent.transform)
        {
            if (child.gameObject == objectToIgnore) continue;

            ObjectRadius existingObjectRadiusComponent = child.GetComponent<ObjectRadius>();
            if (existingObjectRadiusComponent != null)
            {
                float distanceXZ = Vector2.Distance(new Vector2(placementPosition.x, placementPosition.z), new Vector2(child.position.x, child.position.z));

                if (distanceXZ < newObjectRadius + existingObjectRadiusComponent.Radius)
                {
                    canPlace = false;
                    break;
                }
            }
```

```
        }

        return canPlace;
    }


    public GameObject GetPrefabByName(string prefabName)
    {
        if (string.IsNullOrEmpty(prefabName)) return null;

        if (startAsteroidPrefab != null && startAsteroidPrefab.name == prefabName)
return startAsteroidPrefab;
        if (finishAsteroidPrefab != null && finishAsteroidPrefab.name ==
prefabName) return finishAsteroidPrefab;
        if (randomAsteroidPrefabs != null)
        {
            foreach (var prefab in randomAsteroidPrefabs)
            {
                if (prefab != null && prefab.name == prefabName) return prefab;
            }
        }
        if (plasmaGunAsteroidPrefab != null && plasmaGunAsteroidPrefab.name ==
prefabName) return plasmaGunAsteroidPrefab;
        if (laserSMGAsteroidPrefab != null && laserSMGAsteroidPrefab.name ==
prefabName) return laserSMGAsteroidPrefab;
        if (ammo1AsteroidPrefab != null && ammo1AsteroidPrefab.name ==
prefabName) return ammo1AsteroidPrefab;
        if (ammo2AsteroidPrefab != null && ammo2AsteroidPrefab.name ==
prefabName) return ammo2AsteroidPrefab;
        if (healthPackPrefab != null && healthPackPrefab.name == prefabName)
return healthPackPrefab;
        if (turretPrefab != null && turretPrefab.name == prefabName) return
turretPrefab;
        if (ufoSpawnerPrefab != null && ufoSpawnerPrefab.name == prefabName)
return ufoSpawnerPrefab;

        Debug.LogWarning($"Prefab with name '{prefabName}' not found in
LevelEditor's list.");
        return null;
    }

    private string GetSavePathBySlot(int slot)
    {
        string fileName;
        switch (slot)
```

```
        {
            case 1: fileName = saveSlot1; break;
            case 2: fileName = saveSlot2; break;
            case 3: fileName = saveSlot3; break;
            default:
                Debug.LogError($"Invalid slot number: {slot}");
                return null;
        }
        return Path.Combine(Application.persistentDataPath, fileName);
    }
}
```

Реалізація функціональної задачі “Забезпечення роботи системи збору бонусів”

Файл Collectable.cs

```
using UnityEngine;

public abstract class Collectable : MonoBehaviour
{
    public abstract void OnBeingCollectedBy(MainPlayer character);

    void OnTriggerStay(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            MainPlayer player = other.GetComponent<MainPlayer>();
            if (player != null)
            {
                OnBeingCollectedBy(player);
            }
        }
    }
}
```

Файл **PickingUpScript.cs**

```
using UnityEngine;

public class PickingUpScript : MonoBehaviour
{
    [SerializeField] private MainPlayer owner;
    private void OnTriggerEnter(Collider other)
    {
        Debug.Log($"Entered object {other.name}");
        if (other.CompareTag("Collectable") && other.TryGetComponent(out
Collectable collectable))
```

```
        {
            collectable.OnBeingCollectedBy(owner);
        }
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ГРА У ЖАНРІ 3D ACTION З ЕЛЕМЕНТАМИ ПРОСТОРОВОЇ НАВІГАЦІЇ

Програма та методика тестування

КПІ.ІП-1311.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро ЖМАЙЛО

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблена гра для персональних комп’ютерів. Гра створена за допомогою мови програмування C# та ігрового рушія Unity. Гра розрахована на запуск на операційних системах сімейства Windows, починаючи з 10 версії.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи основних механік гри, а саме переміщення, системи озброєння, можливості проходження рівнів.
- перевірка правильності роботи притаманних ігрових механік, а саме появи ворогів на рівні, коректної роботи їх штучного інтелекту.
- перевірка коректності роботи системи проходження кампанії гри;
- перевірка коректності роботи редактора рівнів, системи збереження та завантаження рівнів.
- перевірка коректності роботи системи бонусів.
- перевірка сумісності гри з останніми версіями сучасних операційних систем сімейства Windows.
- перевірка продуктивності роботи гри.
- знаходження проблем, помилок і недоліків у коді з метою їх усунення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються наступні методи та підходи:

– Статичне тестування – на цьому етапі перевірявся весь код гри. Особлива увага приділялася аналізу коду на предмет дотримання стандартів програмування, виявлення потенційних помилок коду до його фактичного виконання.

– Динамічне тестування – застосовувалося в процесі роботи гри. У рамках динамічного тестування проводився аналіз продуктивності гри, зокрема моніторинг кількості кадрів в секунду (FPS).

– Мануальне тестування – тестування виконувалося без використання автоматизації. Тест-кейси розроблялися та виконувалися вручну для перевірки функціональності гри та її відповідності вимогам.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- програма FPSMonitor для моніторингу продуктивності (зокрема, кількості кадрів в секунду) під час ігрового процесу.
- Visual Studio Code Analyzer для статичного аналізу коду.

Порядок проведення тестування буде наступним:

- статичне тестування для аналізу коду гри під час її розробки.
- динамічне тестування гри під час ігрового процесу задля перевірки її продуктивності.
- мануальне тестування гри на відповідність функціональним вимогам.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ГРА У ЖАНРІ 3D ACTION З ЕЛЕМЕНТАМИ ПРОСТОРОВОЇ НАВІГАЦІЇ

Керівництво користувача

КПІ.ІП-1311.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро ЖМАЙЛО

Київ -2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Dexpolaris» - це комп’ютерна гра у жанрі 3D Action з елементами просторової навігації призначена для гри на персональних комп’ютерах. Інсталяційна версія включає в себе файли, потрібні для коректного функціонування гри та безпосередньо файл «3D Shooter.exe», який відповідає за запуск гри.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення повинно функціонувати на персональних комп’ютерах, сумісних з операційною системою сімейства Windows.

Мінімальна конфігурація технічних засобів:

- ОС: Windows 10 (64 Bit);
- об‘єм ОЗП: 4 Гб;
- процесор: Intel Core 2 Duo;

Рекомендована конфігурація технічних засобів:

- ОС: Windows 11 (64 Bit);
- об‘єм ОЗП: 8 Гб;
- процесор: Intel Core i3;

2.2 Завантаження застосунку

На даний момент гру можна встановити власноруч, використовуючи відповідний архівований файл гри. Для цього спершу необхідно завантажити архів з грою на персональний комп’ютер, а потім, використовуючи який-небудь архіватор, виконати розпаковку даного архіву. При розпаковці з’явиться папка з файлами гри, яка міститиме файл «3D Shooter.exe», який відповідає за запуск гри.

2.3 Перевірка коректної роботи

По завершення завантаження й розархівування файлів гри, у обраній директорії повинна з’явитися папка «Dexpolaris». У разі, якщо дана папка не з’явилася, або присутні не всі файли гри, то файли гри були пошкоджені при завантаженні. Інакше користувач має змогу запустити гру, відкривши файл «3D Shooter.exe», який знаходиться у папці «Dexpolaris». Після запуску повинно з’явитися основне меню гри.

3 ВИКОНАННЯ ПРОГРАМИ

Запускаємо гру та бачимо головне меню (рисунок 3.1).

Рисунок 3.1 – Головне меню гри

З головного меню гри можемо перейти до меню інформації про гру (рисунок 3.2), до меню редактора рівнів (рисунок 3.3), або до меню гри на рівнях (рисунок 3.12). Також можемо вийти з гри, натиснувши клавішу “Quit”.

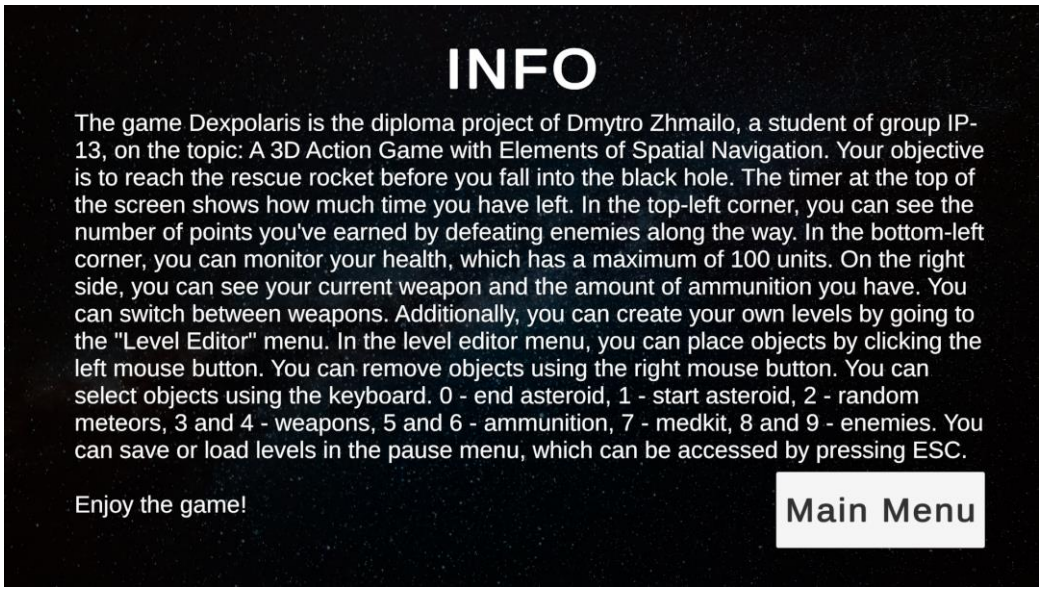


Рисунок 3.2 – Меню інформації про гру

Перейшовши до редактора рівнів, ми можемо побачити площину встановлення об’єктів (рисунок 3.3).

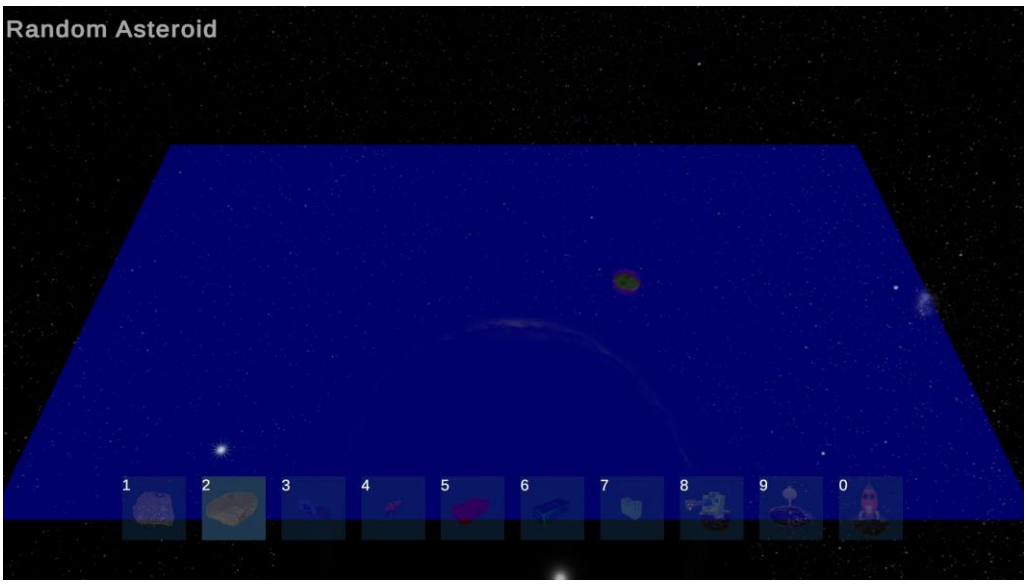


Рисунок 3.3 – Площина розміщення об’єктів редактора рівнів

Користувач може керувати камерою, наближаючи її до площини, використовуючи колесо комп’ютерної миші та може переміщувати камеру навколо площини за допомогою клавіш: W (вперед), A (вліво), S (назад), D (вправо). Переміщення камери обмежено для того, щоб користувач не міг відлетіти далеко від площини розміщення об'єктів, або перемістится під неї. Користувач може обирати об'єкти до розміщення на рівні, використовуючи клавіші: 1 для розміщення стартового астероїду, 2 для розміщення випадкового звичайного астероїду, 3 для розміщення астероїду з плазменною рушницею, 4 для розміщення астероїду з лазерним пістолетом-кулеметом, 5 для розміщення астероїду з амуніцією для лазерного пістолета-кулемета, 6 для розміщення астероїду з амуніцією для плазменної рушниці, 7 для розміщення астероїду з аптечкою, 8 для розміщення астероїду з лазерною турелю, 9 для розміщення спавнера НЛО, 0 для розміщення кінцевого астероїду. Вибір користувача відображається у вигляді комірок з іконками обраного астероїду, які можна побачити внизу екрану на рисунку 3.3. Також вибір користувача можна побачити у текстовому вигляді у лівому верхньому куті екрану. Наблизивши камеру, можна побачити тінь об'єкта для розміщення, яка показує його область колізії (рисунок 3.4).

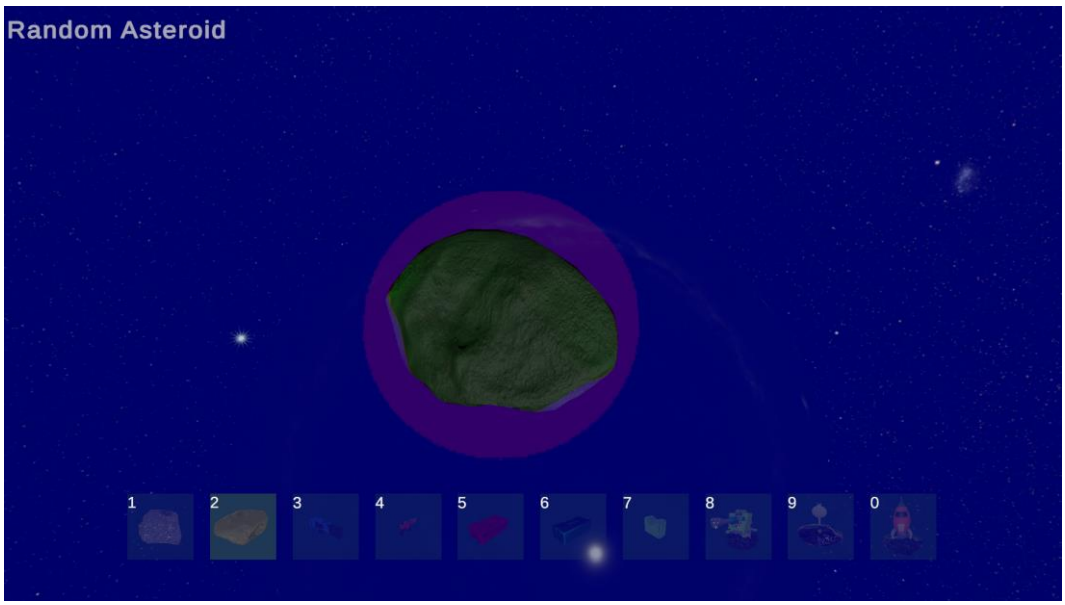


Рисунок 3.4 – Тінь об'єкта, що розміщується

Тінь об'єкта відповідає за його зайнятий простір. Якщо об'єкт можна розмістити, він підсвічується зеленим кольором, якщо його не можна розмістити -червоним (рисунок 3.5). Об'єкт не можна розмістити, якщо його тінь пересікається з іншим об'єктом, або екземпляр об'єкту є одиничним. Одиничними екземплярами об'єктів є початковим та кінцевий астроїди. Об'єкт розміщується за допомогою натискання на ліву кнопку комп'ютерної миші. Видалити об'єкт можна, якщо навестися на нього та натиснути праву кнопку комп'ютерної миші.

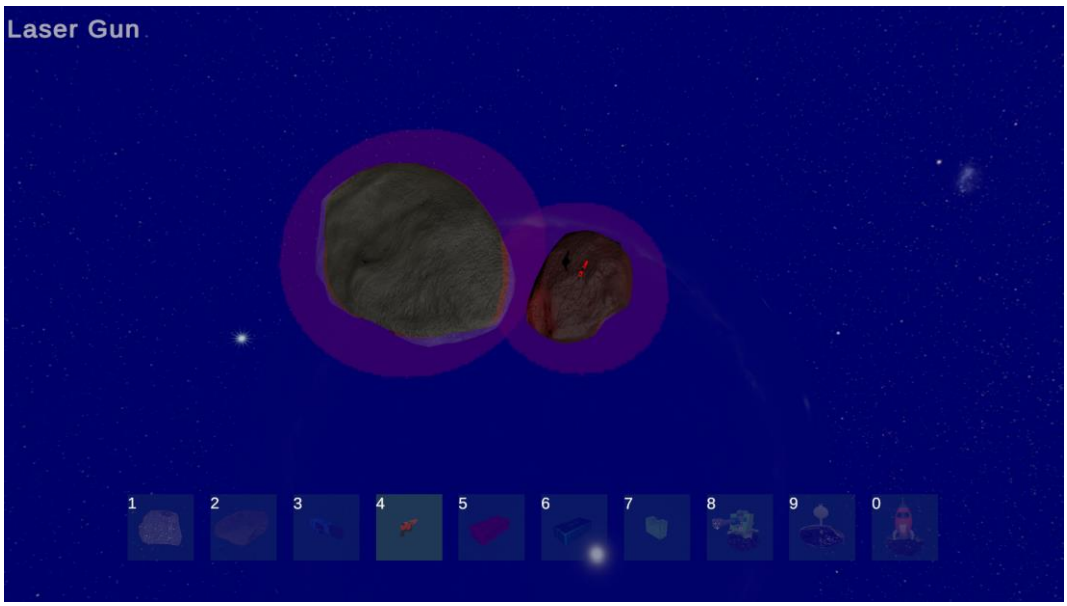


Рисунок 3.5 – Неможливість розмістити об'єкт

Якщо гравець намагається розмістити початковий або кінцевий астероїд двічі, гра виводить йому про це помилку та унеможливлює повторне розміщення цього об'єкту (рисунок 3.6).

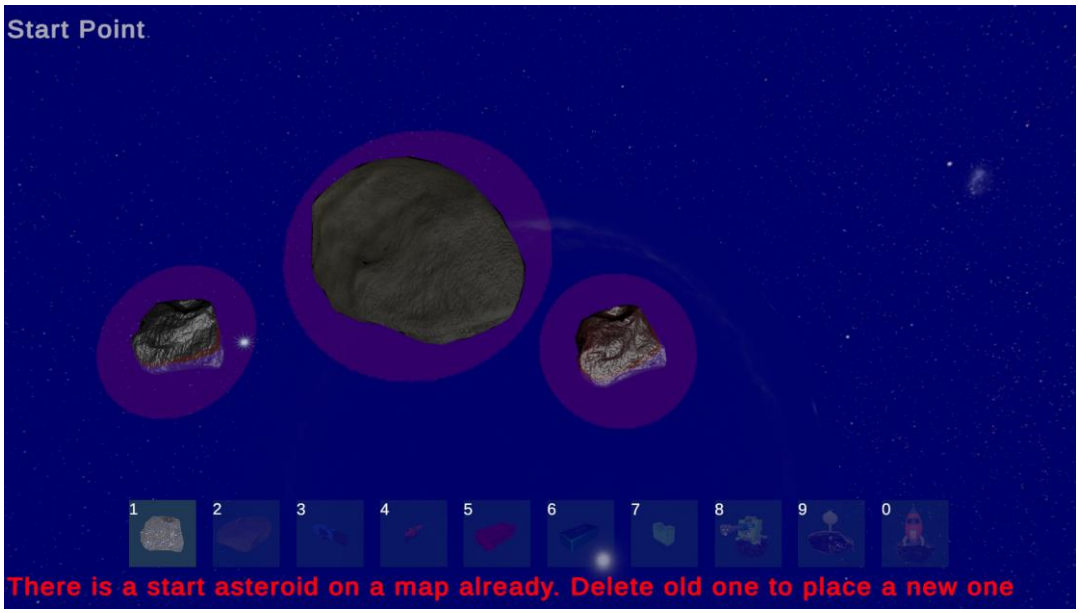


Рисунок 3.6 – Помилка про повторне розміщення стартового астероїда

Розміщуючи об'єкти ворогів, а саме лазерних турелей та спавнерів НЛО, гравець може побачити радіус їх дії для планування подальшої гри на рівні (рисунок 3.7).

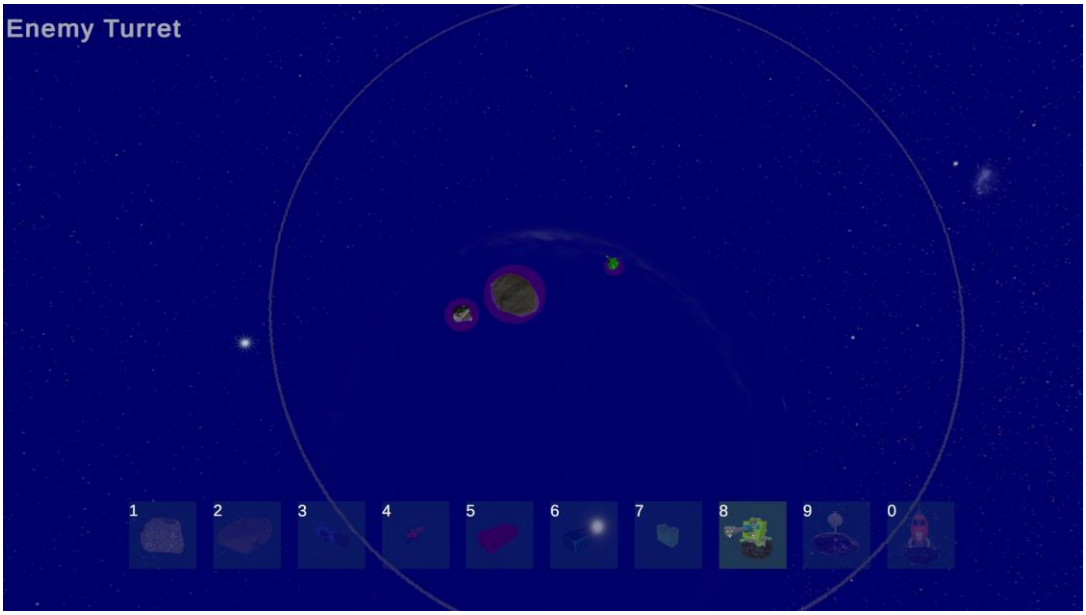


Рисунок 3.7 – Радіус дії лазерної турелі

При натисканні на клавішу “ESC”, гравець потрапляє до меню керування рівнями (рисунок 3.8).

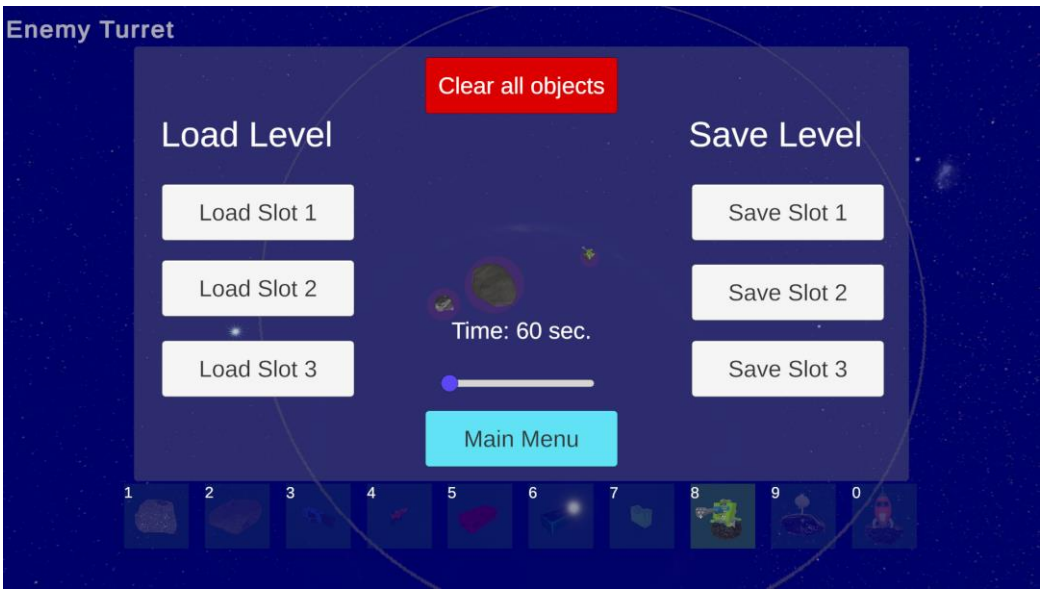


Рисунок 3.8 – Меню керування рівнями

Тут гравець може завантажити вже збережені рівні, або зберегти новий рівень. Також гравець може встановити час проходження рівня, який варіюється від 60 до 180 секунд. За бажанням, гравець може очистити всі об’єкти на рівні. При обиранні будь-якої дії, крім повернення до рівня редактора при повторному натисканні клавіші “ESC”, користувач отримує вікно підтвердження для запобігання втрати незбережених змін (рисунок 3.9).

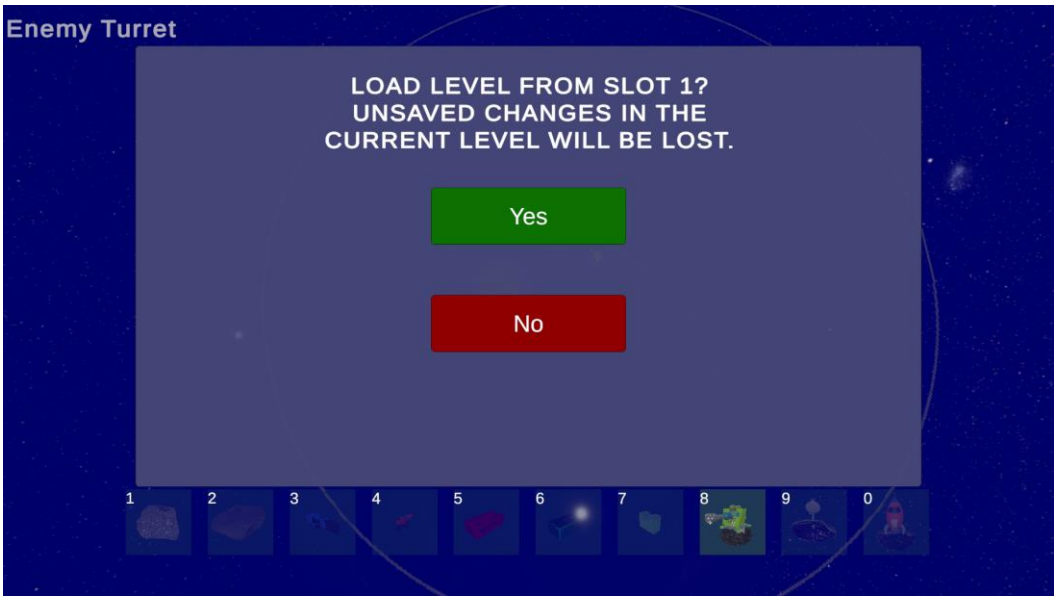


Рисунок 3.9 – Меню підтвердження дії

При збереженні рівня, він валідується на коректність (наявність стартового та фінального астероїда), та виводить помилку у разі

перешкоджання збереження рівня (рисунок 3.10) або повідомлення про успішне збереження рівня (рисунок 3.11).

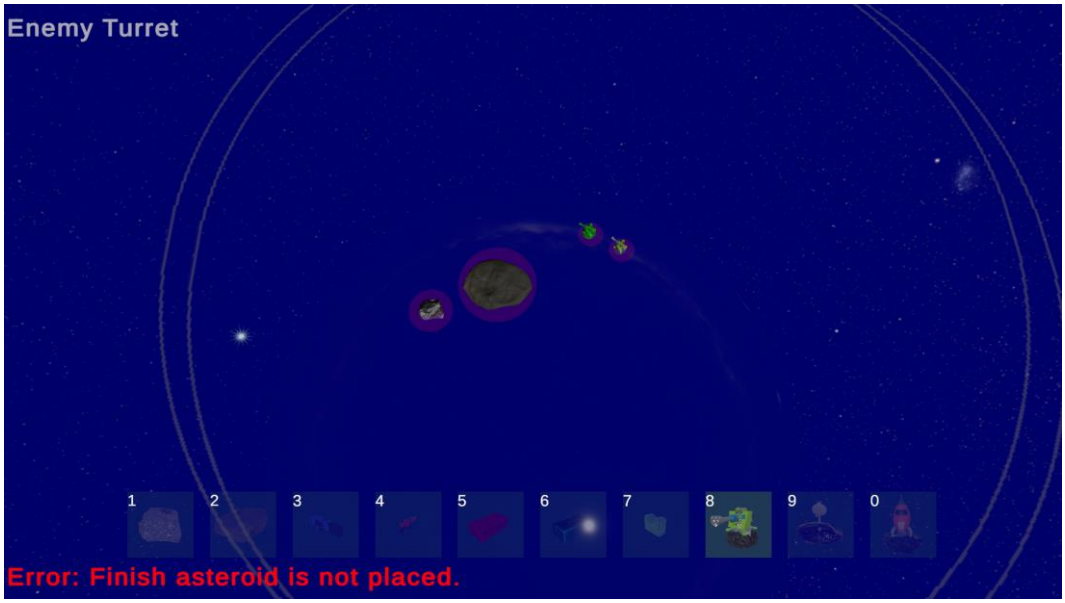


Рисунок 3.10 – Помилка збереження рівня

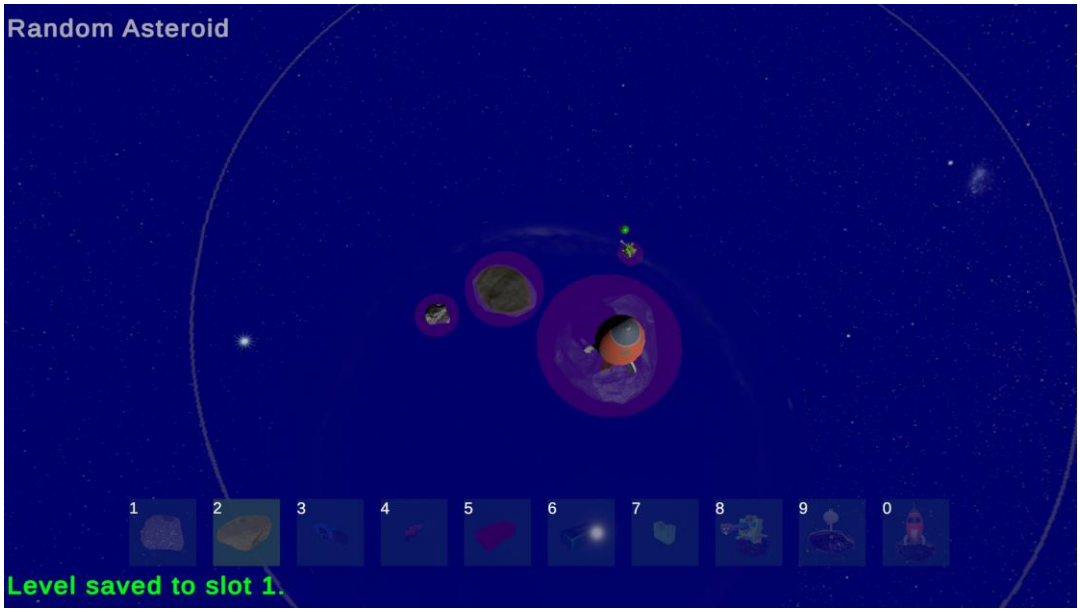


Рисунок 3.11 – Повідомлення про успішне збереження рівня

З меню керування рівнями (рисунок 3.8) можна перейти до головного меню (рисунок 3.1), з якого можна перейти до меню гри на рівнях (рисунок 3.12)

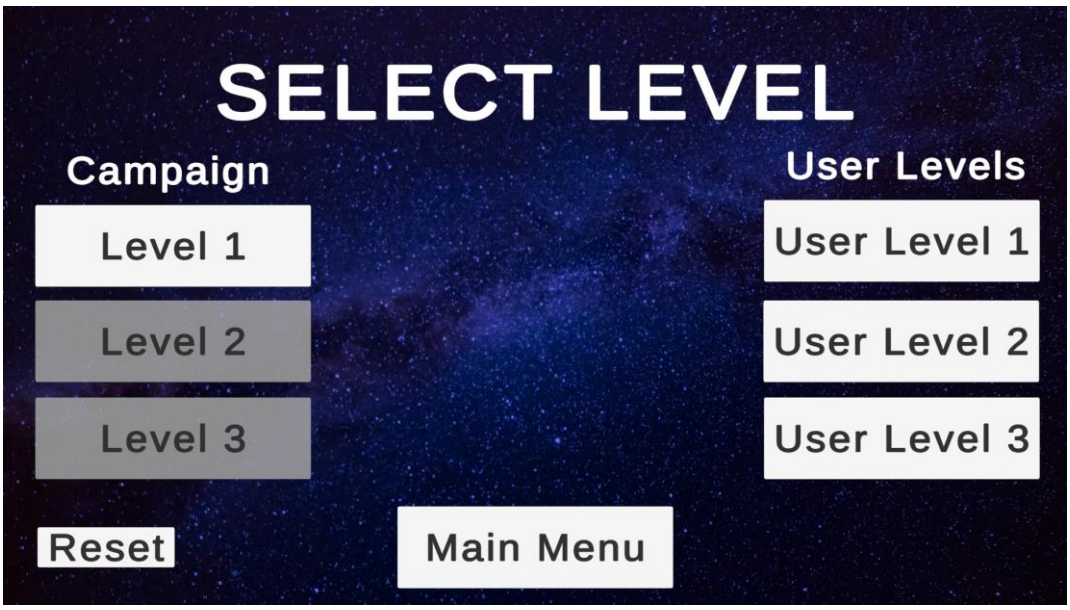


Рисунок 3.12 – Меню гри на рівнях

В меню гри на рівнях ми можемо обрати рівні кампанії, які відкриваються поступово одне за одним, або можемо обрати доступні користувацькі рівні. Доступні рівні підсвічуються білим кольором, недоступні -сірим кольором. Також можна скинути прогрес проходження рівнів кампанії, натиснувши кнопку “Reset”. Натиснувши на будь-який з доступних рівнів, ми потрапляємо безпосередньо до рівня гри з його ігровим процесом (рисунок 3.13).

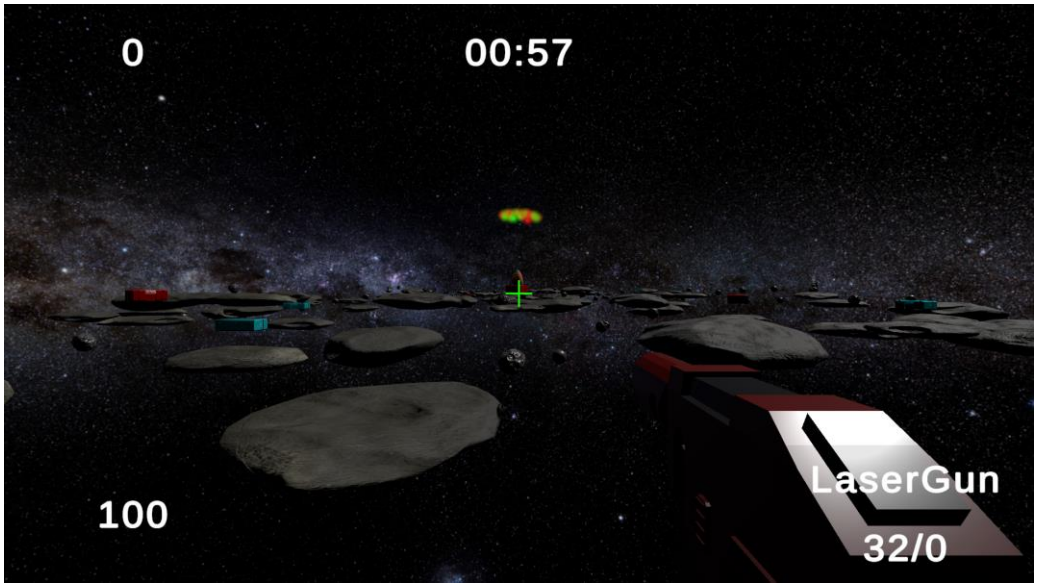


Рисунок 3.13 – Ігровий процес

При старті рівня, камера гравця завжди повернута до фінального астероїду. При появі гравця, ракета на фінальному астероїді випускає світло

маяка для легшого її знаходження. Світло повторюється знову, коли залишається 30 секунд до кінця рівня. Зліва знизу ми можемо побачити здоров'я гравця, справа -тип озброєння, кількість амуніції в магазині та взагалом. Змінити зброю можна, користуючись колесом комп'ютерної миші. Стріляти з зброї можна, натиснувши ліву кнопку комп'ютерної миші. Переміщуватися можна за допомогою клавіш керування: W (вперед), A (вліво), S (назад), D (вправо). Прискоритися можна, натиснувши клавішу “Shift”, підстрибнути можна, натиснувши клавішу “Space”. В лівому верхньому куті можна побачити кількість балів, нарахованих гравцю за знищення ворогів та їх спавнерів. Зверху можна побачити таймер, який веде зворотній відлік до часу падіння поясу астероїдів у чорну диру, відносне наближення якої можна візуально побачити знизу рівня (рисунок 3.14).

Рисунок 3.14 – Чорна діра, яка візуально стає ближчою

Суть гри - встигнути дійти до фінального астероїду й не загинути від ворогів або порожнечі космосу. При падінні або відсутності здоров'я, гравець отримує вікно програшу (рисунок 3.15).

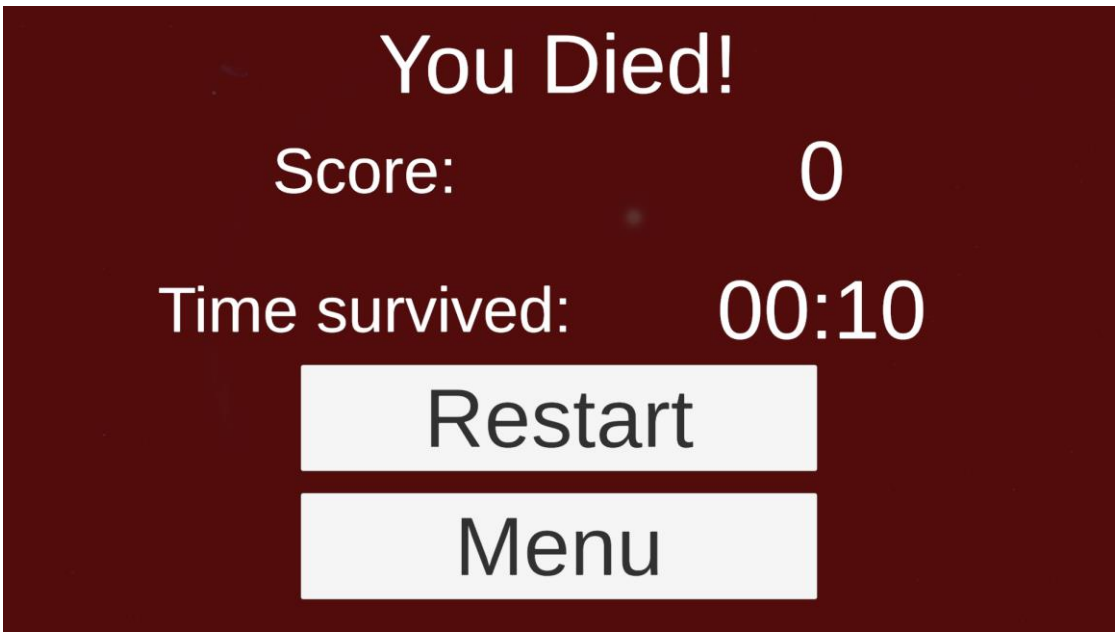


Рисунок 3.15 – Вікно програшу

Якщо гравець досягає фінального астероїду вчасно, він отримує вікно перемоги (рисунок 3.16)

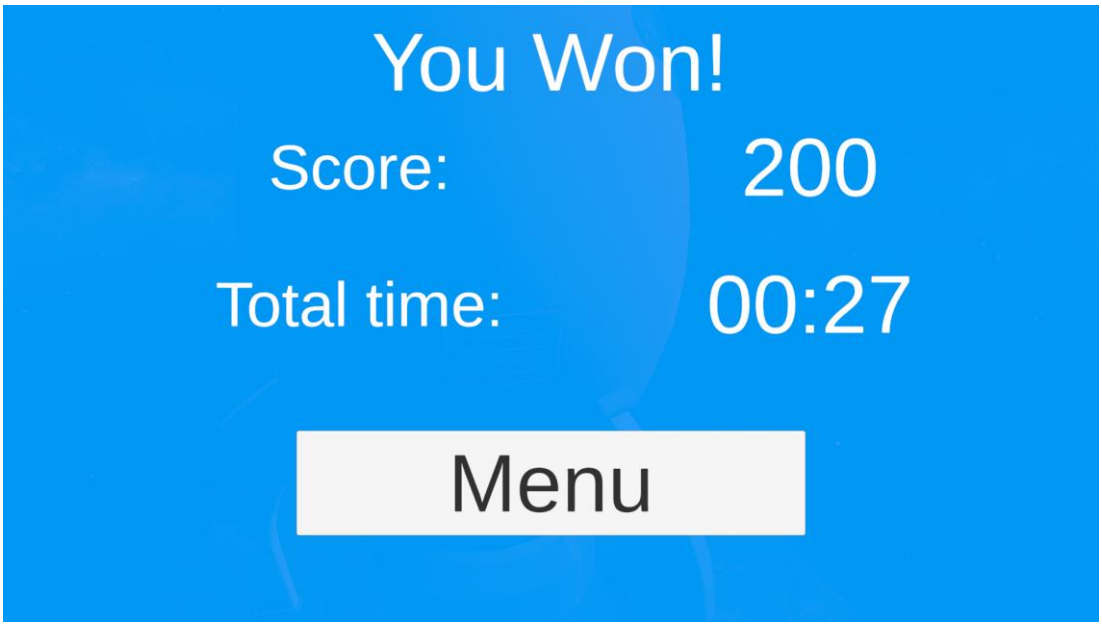


Рисунок 3.16 – Вікно перемоги

На шляху гравця трапляються численні бонуси, які він може підібрати, підійшовши до них (рисунок 3.17).

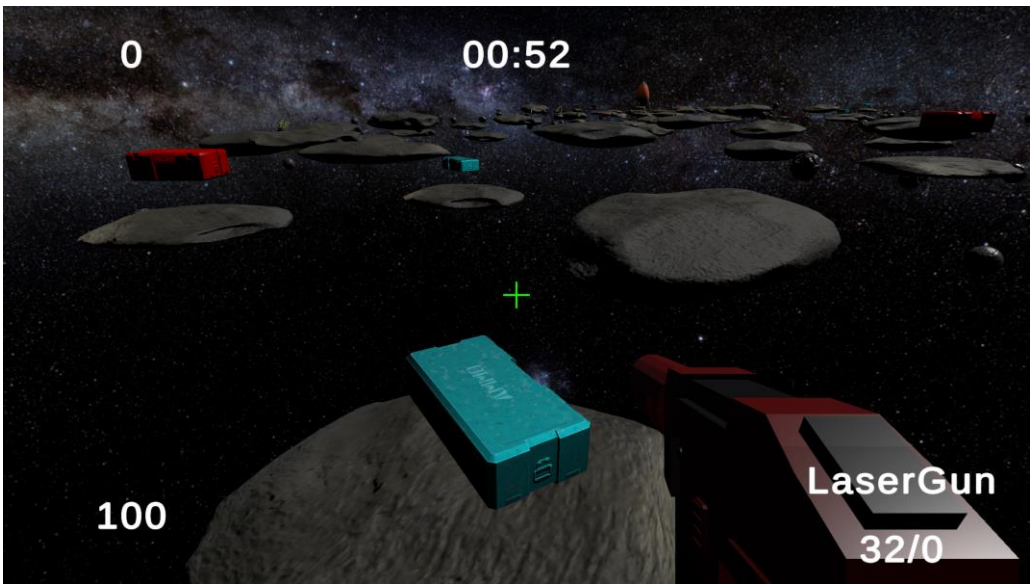


Рисунок 3.17 – Бонуси на рівнях

На шляху гравця можуть траплятися вороги: лазерні турелі та НЛО, які з'являються зі спавнерів (рисунок 3.18). Вороги атакують гравця лазерами. При знищенні ворогів, вони вибухають та нараховують бонусні бали гравцю. Вороги НЛО з'являються доти, доки присутні їх спавнери, які можна знищити за допомогою різних видів озброєння.

Рисунок 3.18 – Вороги на рівнях

Під час ігрового процесу, гравець може перейти до меню паузи, натиснувши кнопку “ESC” (рисунок 3.19).

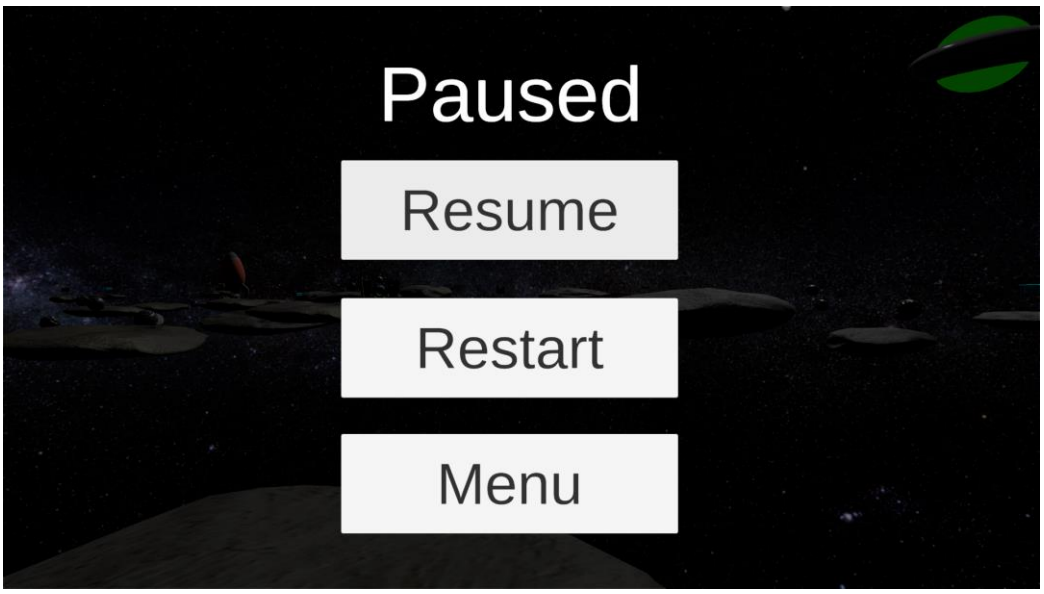


Рисунок 3.19 – Меню паузи

В меню паузи гравець може перезапустити рівень, повернутися до меню, або продовжити гру.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ГРА У ЖАНРІ 3D ACTION З ЕЛЕМЕНТАМИ ПРОСТОРОВОЇ НАВІГАЦІЇ

Графічний матеріал

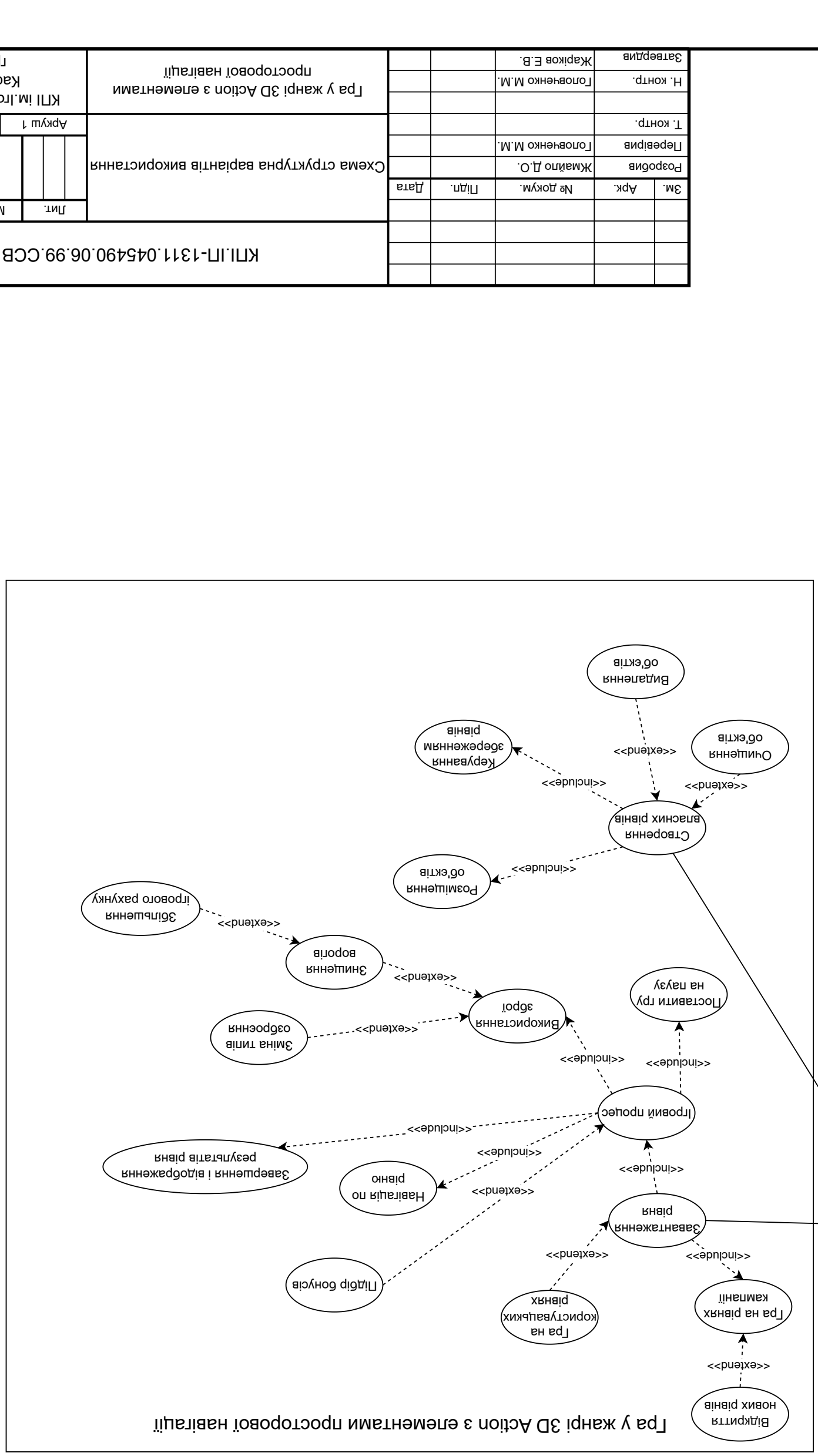
КПІ.ІП-1311.045490.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Дмитро ЖМАЙЛО

Київ – 2025

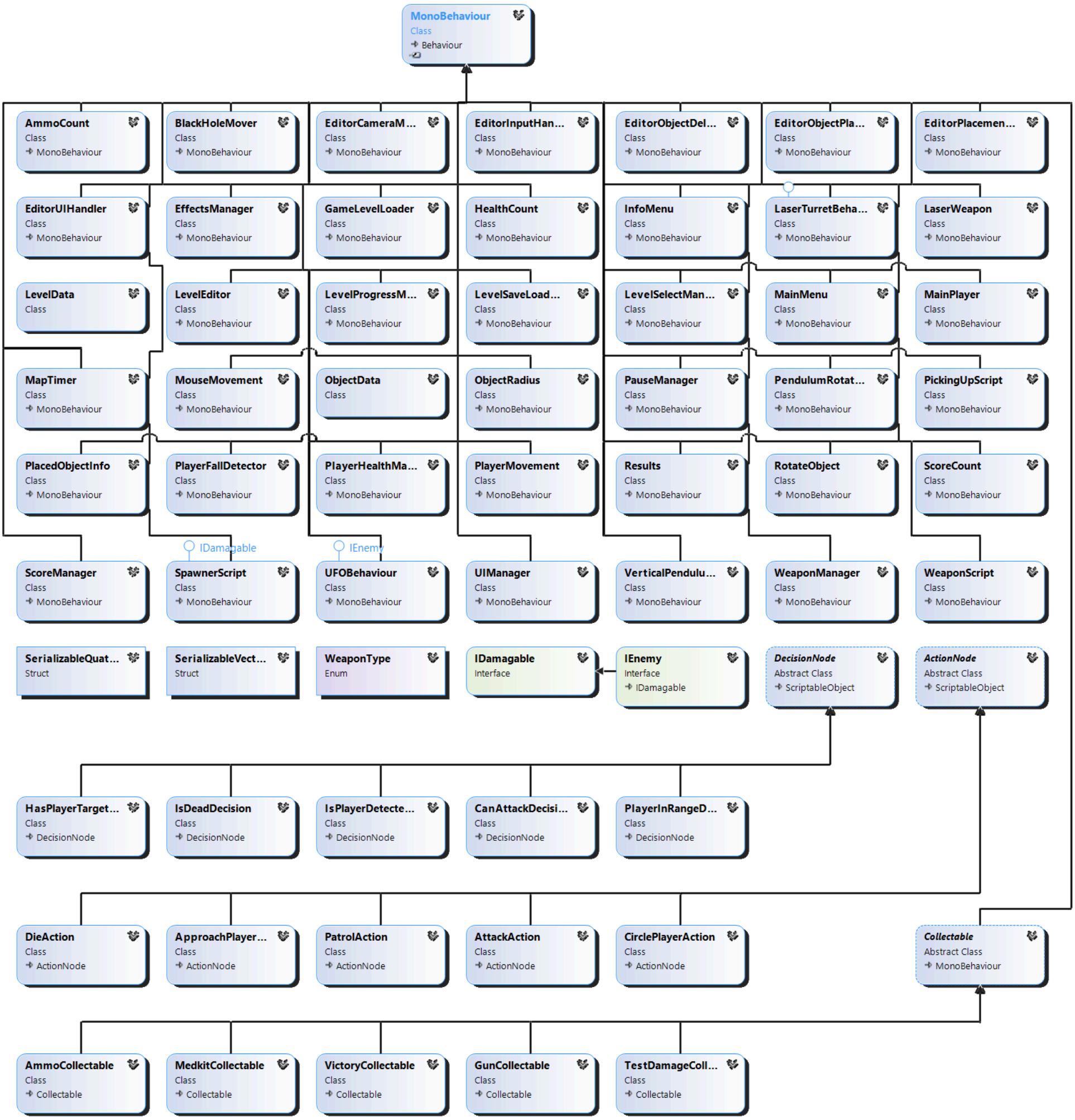


Гра у жанрі 3D Action з елементами простороової навігації

Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1311.045490.06.99.ССВ				
Розробив		Жмайло Д.О.							
Перевірив		Голованенко М.М.			Схема структура варіантів використання				
Т. контр.									
Н. контр.		Голованенко М.М.							
Затвердив		Жарікова Е.В.			Гра у жанрі 3D Action з елементами простороової навігації				
						Літ.	Маса	Масштаб	
						Аркуш 1		Аркушів 1	
						КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13			

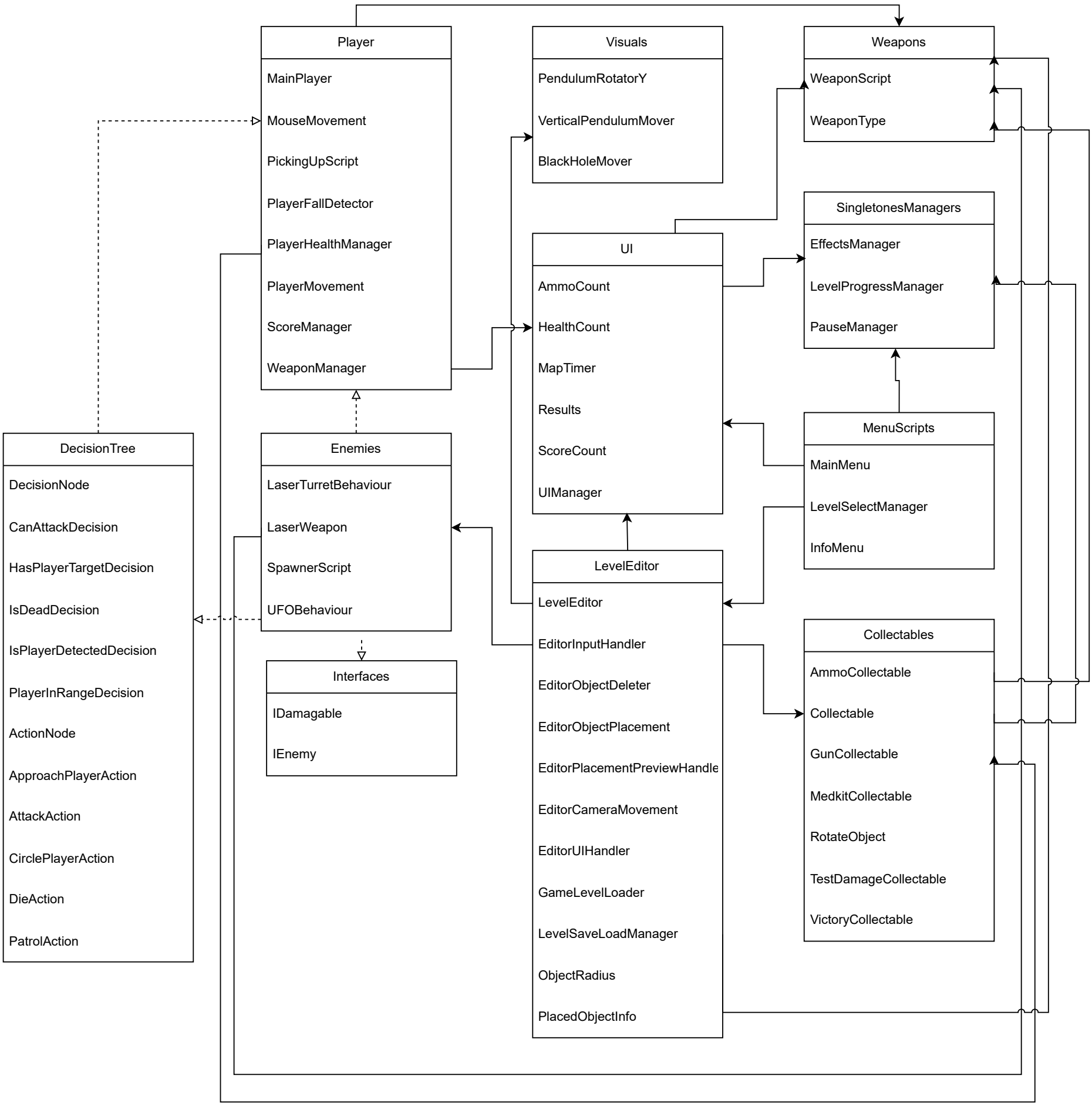
КПІ.ІП-1311.045490.06.99.ССВ

КПІ.ІП-1311.045490.06.99.ССК



					КПІ.ІП-1311.045490.06.99.ССК		
						Лит.	Маса	Масштаб
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна класів програмного забезпечення			
Розробив		Жмайло Д.О.				Аркуш 1	Аркушів 2	
Перевірив		Головченко М.М.			Гра у жанрі 3D Action з елементами просторової навігації	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13		
Т. контр.								
Н. контр.		Головченко М.М.						
Затвердив		Жаріков Е.В.						

КПІ.ІП-1311.045490.06.99.ССК



Зм.	Арк.	№ докум.	Підп.	Дата
Розробив		Жмайло Д.О.		
Перевірив		Головченко М.М.		
Т. контр.				
Н. контр.		Головченко М.М.		
Затвердив		Жаріков Е.В.		

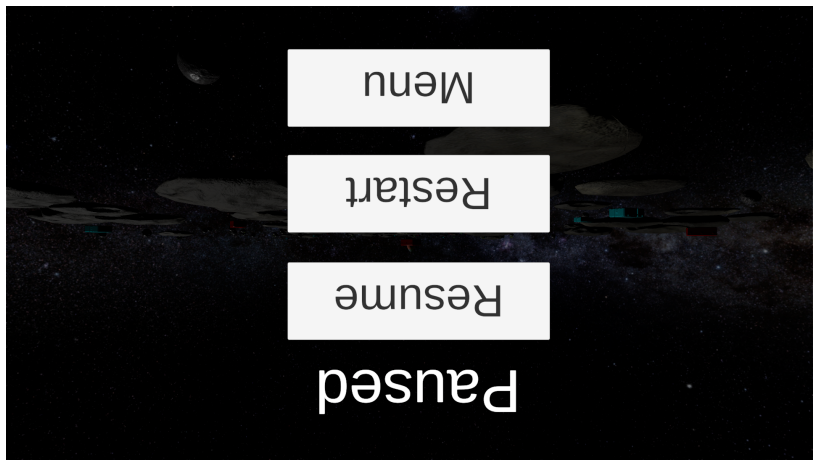
КПІ.ІП-1311.045490.06.99.ССК

Схема структурна класів програмного забезпечення

Лит.	Маса	Масштаб
Аркуш 2	Аркушів 2	

Гра у жанрі 3D Action з елементами просторової навігації

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-13



Зм.	Арк.	№ докум.	Підп.	Дата
Розробив		Жмайло Д.О.		
Перевірив		Голованенко М.М.		
Т. контр.				
Н. контр.		Голованенко М.М.		
Затвердив		Жарікова Е.В.		

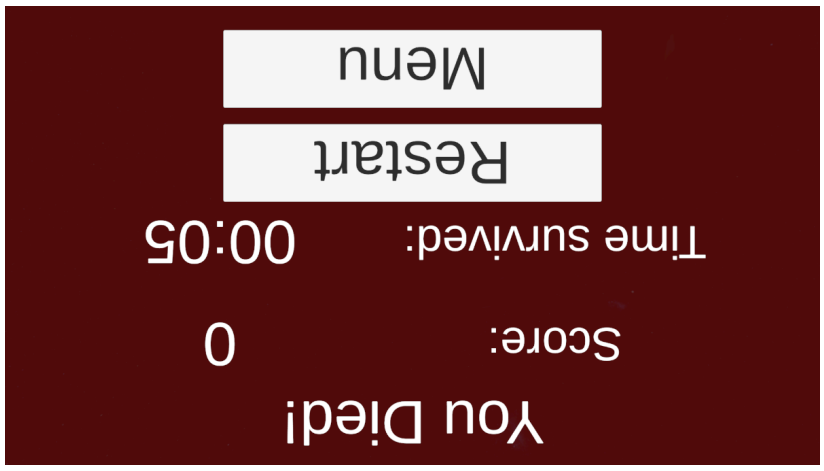
КПІ.ІП-1311.045490.06.99.КЕ

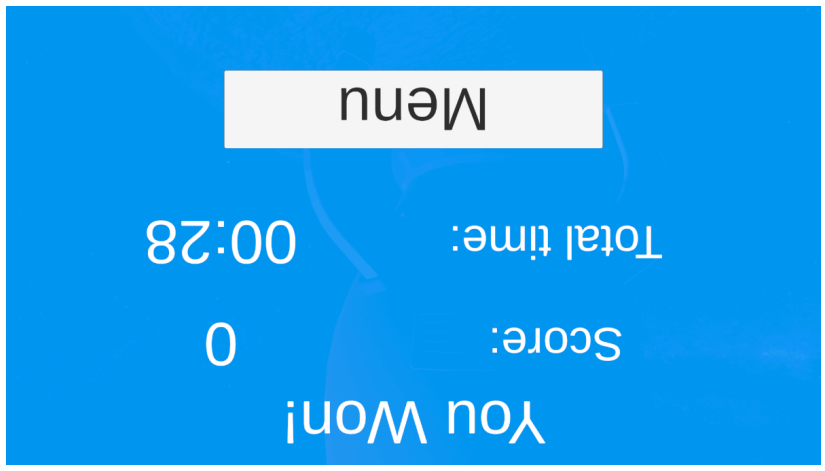
Крестенна вигляду екранних форм

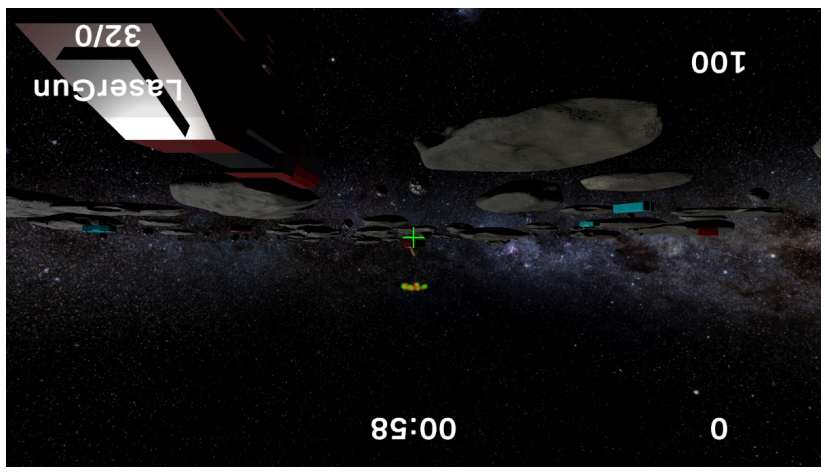
Літ.	Маса	Масштаб
Аркуш 1		Аркушів 1

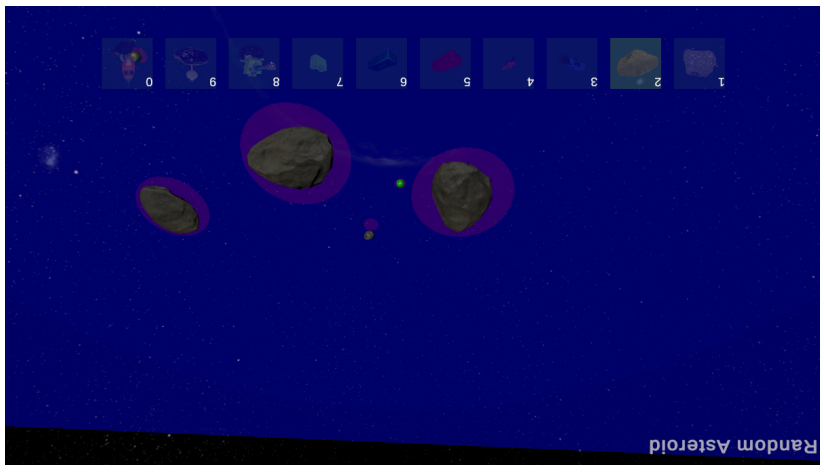
Гра у жанрі 3D Action з елементами пристосовоі навігації

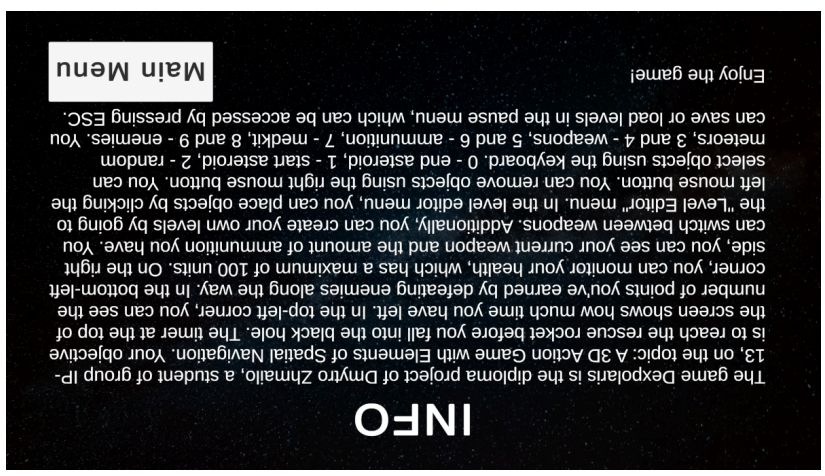
КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-13











КПІ.ІП-1311.045490.06.99.КЕ