МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-07

Дипломний проєкт

здобувача освіти денної форми навчання

РП.07.13.000.ДП

МЕЛЬНИКА МИКИТИ ОЛЕКСАНДРОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Інженерія програмного забезпечення»**

Група: **4РП-07**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту на тему:

Розробка ігрового застосунку у жанрі стратегія

Проєктний матеріал складається з пояснювальної записки на **74** сторінках та графічного (презентаційного) матеріалу на **13** аркушах (слайдах).

Дипломник _____ (Мельник М. О.)

Керівник _____ (Іванова Л. В.)

Консультанти:

з економічного розділу _____ (Іванченков В. С.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н. І.)

з нормоконтролю _____ (Петрашова В. І.)

старший консультант _____ (Кривченко Ю. В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю. В.)

Завідувач відділення _____ (Скорнякова О. В.)

Захист «17» 06 2024 р. Протокол ДКК № 1

Оцінка ДКК 4 (добре) / 80.

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітня програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І. В.*

" 15 " 01 *2024 року*

ЗАВДАННЯ

на дипломний проєкт

Здобувачеві освіти *Мельнику Микиті Олександровичу*

1. Тема проєкту *Розробка ігрового застосунку у жанрі стратегія*

Затверджена наказом по коледжу від " 02 " *листопада* *2023 р.*, наказ № *244-А2-ОД*

2 Термін здачі закінченого проєкту 10.06.2024

3. Вихідні дані до проєкту

1. Спроектувати сюжет ігрового застосунку.

2. Реалізувати графічний інтерфейс (GUI) ігрового застосунку

3. Спроектувати Level-дизайн ігрового застосунку

4. Спроектувати дизайн персонажів ігрового застосунку

5. Розробити ігровий застосунок засобами рушію Unity

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Аналіз предметної області. 2. Технології та засоби розробки (проєктування).

3. Проєктування дизайну гри. 4. Проєктування архітектури ігрового застосунку.

5. Розробка ігрового застосунку. 6. Тестування створеного ігрового застосунку.

7. Економічний розрахунок. 8. Аспекти охорона праці та техніки безпеки

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Презентація Power Point – 13 слайдів

(Використанні технології; Зовнішній вигляд веж; Зовнішній вигляд ворогів;

Вигляд мапи; Навігація по графичному інтерфейсу; Частина коду;

Початок гри; Геймплей; Кінець гри; Висновки)

6. Консультанти по проєкту, із зазначенням розділів проєкту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Іванова Л. В.*		
Економічний розділ	*Іванченков В. С.*		
Розділ охорони праці	*Чорновол Н. І.*		
Нормоконтроль	*Петрашова В. І.*		
Старший консультант	*Кривченко Ю. В.*		

7. Дата видачі завдання 24.10.2023

Керівник *Іванова Л. В.* (підпис)

Завдання прийняв до виконання *Мельник М. О.* (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проєкту (роботи)	Термін виконання етапів дипломного проєкту (роботи)	Відмітка про виконання
1	*Формування вступу*	*29.04.24*	Виконано
2	*Аналіз предметної області*	*10.05.24*	Виконано
3	*Підбір технічної літератури*	*19.05.24*	Виконано
4	*Вибір технологій та засобів розробки (проєктування)*	*20.05.24*	Виконано
5	*Проєктування дизайну ігрового застосунку*	*22.05.24*	Виконано
6	*Проєктування архітектури ігрового застосунку*	*24.05.24*	Виконано
7	*Розробка ігрового застосунку*	*27.05.24*	Виконано
8	*Тестування створеного ігрового застосунку*	*29.05.24*	Виконано
9	*Оформлення пояснювальної записки*	*31.05.24*	Виконано
10	*Оформлення графічної (презентаційної) частини*	*01.06.24*	Виконано
11	*Економічний розрахунок*	*02.06.24*	Виконано
12	*Опис охорони праці та техніки безпеки*	*09.06.24*	Виконано
13	*Аналіз результатів проєктування*	*13.06.24*	Виконано
14	*Підготовка доповіді для захисту*	*16.06.24*	Виконано

Дипломник (підпис)

Керівник (підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 07. 13 000. 00 ДП	Дипломний проєкт		
А4			РП 07. 13 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 07. 13 000. 00 ДП ПЗ		
Розроб.		Мельник М.О.		10.06	Розробка ігрового застосунку у жанрі стратегія	Літ.	Аркуш 4 / Аркушів 74
Перевірив		Іванова Л.В.		10.06		Н Д П	
Н. Контр.		Петрашова В.І.		10.06	ВСП "ОТФК ОНТУ" гр.4РП-07		
Затверд.		Кривченко Ю.В.		10.06			

ЗМІСТ

ВСТУП 7
1 ОСНОВНИЙ РОЗДІЛ 9
1.1 Аналіз предметної області 9
1.1.1 Порівняння наявних ігрових застосунків 9
1.1.2 Обрані технології ігробудування 14
1.2 Проєктування ігрового застосунку 17
1.2.1 Архітектура системи 18
1.2.2 Розробка ігрових механік 20
1.2.3 Дизайн інтерфейсу користувача 20
1.2.4 Структура даних гри 20
1.3 Покрокова розробка гри 21
1.3.1 Перші кроки 21
1.3.2 Створення противника 23
1.3.3 Рух противника 24
1.3.4 Створення веж 27
1.3.5 Система "Nodes" 33
1.3.6 Валюта та магазин 35
1.3.7 Система закінчення гри 37
1.3.8 Головне меню 38
1.3.9 Вибір рівня 39
1.4 Тестування гри 40
1.4.1 Методи тестування 41
1.4.2 План тестування 43
1.4.3 Аналіз результатів тестування 43
1.4.4 Початок тестування 44
2 ЕКОНОМІЧНИЙ РОЗДІЛ 50
2.1 Резюме 50
2.2 Визначення трудомісткості розробки програмного забезпечення 50

2.3 Розрахунок ціни програмного продукту 54
3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ 56
3.1 Вступ 56
3.2 Аналіз небезпечних та шкідливих чинників що впливають на працівника 56
3.3 Розробка заходів з охорони праці 57
3.3.1 Виробничі приміщення 57
3.3.2 Мікроклімат робочої зони працівників вентиляція 57
3.3.3 Освітлення робочого місця шум вібрація 58
3.3.4 Організація робочого місця користувача ПК 59
3.3.5 Санітарно-гігієнічні вимоги 59
3.3.6 Ергономіка 59
3.4 Пожежна безпека 60
3.4.1 Захист від статичної електрики 60
3.4.2 Захист від іонізуючих випромінювань 61
3.4.3 Захист від лазерного випромінювання 61
ВИСНОВКИ 62
ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 63
ДОДАТОК А. Програмний код основної логіки веб-застосунку 64
ДОДАТОК Б. Слайди мультимедійної презентації 68

ВСТУП

В сучасному світі індустрія відеоігор швидко розвивається і займає важливе місце в сфері розваг. Ігри жанру Tower Defense (TD) користуються великою популярністю серед різних вікових груп завдяки своїй стратегії, динамічності та можливості розвитку логічного мислення у гравців. Розробка гри в цьому жанрі дозволяє поєднати різні аспекти програмування, дизайну та геймплею, що робить цей проєкт надзвичайно цікавим та важливим з точки зору навчання і професійного розвитку.

Метою цього дипломного проєкту є розробка ігрового застосунку у жанрі Tower Defense.

Для досягнення цієї мети було поставлено наступні завдання:

1. Провести аналіз предметної області та вивчити існуючі аналоги.
2. Обрати технології та засоби розробки, які найкраще підходять для створення гри.
3. Розробити архітектуру гри, включаючи дизайн та механіки.
4. Реалізувати гру на основі обраної архітектури та дизайну.
5. Провести тестування гри та оцінити її ефективність.
6. Підготувати економічний розрахунок витрат на розробку та підтримку гри.
7. Розглянути аспекти охорони праці та техніки безпеки під час розробки гри.

В процесі роботи використовувались сучасні методи розробки програмного забезпечення, такі як об'єктно-орієнтоване програмування (ООП), методології Agile та Scrum. Для створення гри була використана платформа Unity, яка є потужним інструментом для розробки інтерактивних 3D-додатків та ігор. Мовами програмування для реалізації проекту обрані C# для скриптів у Unity та JavaScript для інтеграції з веб-середовищем, якщо це необхідно.

Розроблений ігровий застосунок може бути використаний як основа для подальшого вивчення та розвитку в сфері геймдизайну та програмування. Крім того, цей проєкт може стати комерційним продуктом, який буде цікавий широкій аудиторії гравців. Проєкт також демонструє можливості сучасних

технологій та підходів у розробці ігор, що є корисним досвідом для студентів та професіоналів в галузі програмного забезпечення.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз предметної області

1.1.1 Порівняння наявних ігрових застосунків

Ігри жанру Tower Defense (TD) мають давню історію і значну популярність серед гравців. Цей жанр поєднує в собі елементи стратегії, планування та тактики. Гравці повинні будувати захисні споруди та організовувати оборону від хвиль ворогів. Основна мета полягає у запобіганні досягнення ворогами кінцевої точки на карті.

Аналіз популярних ігор у жанрі Tower Defense

Перший аналог – Plants vs. Zombies.

Опис: Plants vs. Zombies – це популярна гра, де гравці використовують різні види рослин для захисту свого будинку від зомбі.

Переваги: Простий, але захоплюючий геймплей, велика кількість рівнів, унікальні вороги та оборонні споруди.

Недоліки: Може здатися занадто простою для досвідчених гравців [1].

На рис 1.1 зображено геймплей Plants vs. Zombies.

Рисунок 1.1. Геймплей Plants vs. Zombies

Другий аналог – Kingdom Rush.

Опис: Kingdom Rush – це класична гра в жанрі TD, де гравці захищають королівство від орд монстрів, використовуючи різноманітні захисні вежі та героїв.

Переваги: Висока стратегічна глибина, різноманітність ворогів і оборонних споруд, цікаві рівні та квести.

Недоліки: Може бути складною для новачків [2].

На рис 1.2 зображено геймплей Kingdom Rush.

Рисунок 1.2. Геймплей Kingdom Rush

Другий аналог – Bloons TD

Опис: Bloons TD – це гра, де гравці використовують різні види мавп та їх зброю для знищення хвиль повітряних кульок (bloons).

Переваги: Яскравий і веселий дизайн, різноманітність тактик і стратегій, постійні оновлення та додатки.

Недоліки: Деякі елементи гри можуть вимагати додаткових покупок для швидшого прогресу [3].

На рис 1.3 зображено геймплей Bloons TD.

Рисунок 1.3. Геймплей Bloons TD

Третій аналог – Arknights.

Опис: Arknights – це Гача гра, в якій користувач повинен використовувати різних персонажів для проходження різних рівнів.

Переваги: Досить приємний та різний візуальний стиль і велика кількість різних механік у персонажів.

Недоліки: Оскільки це гача, шанси на отримання певних персонажів мінімальні якщо не донатити [4].

На рис 1.4 зображено геймплей Arknights.

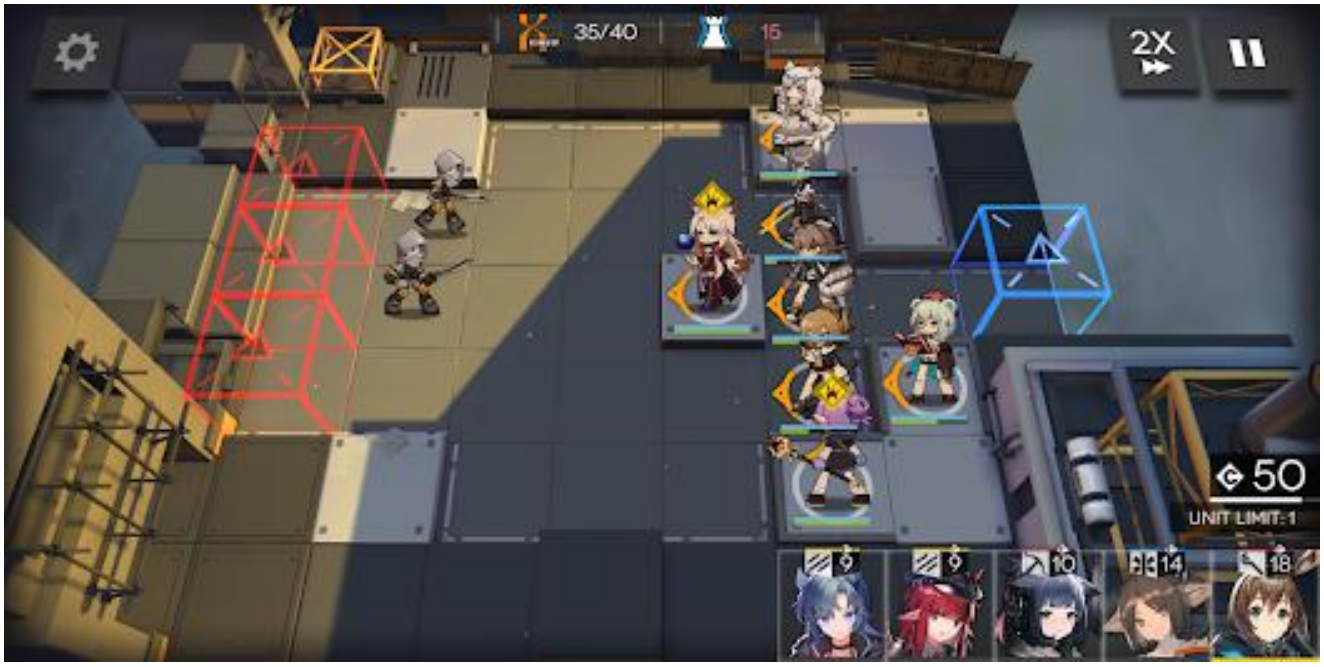


Рисунок 1.4. Геймплей Arknights

У таблиці 1.1. наведено порівняльну характеристику аналогів.

Таблиця 1.1. Порівняльна характеристика аналогів

Характеристика	**Plants vs. Zombies**	**Kingdom Rush**	**Bloons TD**	**Arknights**
Наявність реєстрації	Ні	Так	Так	Так
Наявність реклами	Так	Ні	Так	Так
Наявність української мови	Ні	Ні	Ні	Ні
Рівень складності	Легкий	Середній	Високий	Високий
Різноманітність ворогів	Висока	Висока	Висока	Висока
Різноманітність оборонних споруд	Висока	Висока	Висока	Висока

1.1.2 Обрані технології ігробудування

Для розробки ігор в жанрі TD зазвичай використовуються різні технології та інструменти. Найбільш популярні з них:

Перша технологія – Unity.

Опис: Потужна платформа для розробки ігор, що підтримує 2D та 3D графіку.

Переваги: Велика спільнота розробників, багатий набір інструментів і бібліотек, підтримка різних платформ (Windows, iOS, Android та інші).

Недоліки: Високі вимоги до апаратного забезпечення для складних проєктів [5].

Друга технологія – Unreal Engine.

Опис: Високопродуктивний рушій для розробки 3D ігор.

Переваги: Висока якість графіки, потужні інструменти для розробки, безкоштовний доступ для малих проєктів.

Недоліки: Складність у навчанні для новачків, великі вимоги до ресурсів комп'ютера [6].

Третя технологія – Godot Engine.

Опис: Відкритий рушій для розробки 2D та 3D ігор.

Переваги: Безкоштовний, гнучкий та легкий у навчанні, підтримка різних платформ.

Недоліки: Менша спільнота розробників у порівнянні з Unity та Unreal Engine [7].

У таблиці 1.1. наведено порівняльну характеристику аналогів.

Таблиця 1.1. Порівняльна характеристика аналогів

Характеристика	**Unity**	**Unreal Ungine**	**Godot Engine**
Підтримка платформ	Висока	Висока	Середня
Графіка	Висока	Дуже Висока	Середня
Продуктивність	Висока	Дуже Висока	Висока
Легкість у навчанні	Висока	Середня	Висока
Спільнота та підтримка	Велика	Велика	Менша
Вартість	Безкоштовна/Платна	Безкоштовна/Роялті	Бескоштовна
Система скриптингу	C#	C++, Blueprints	GDScript, C#
Інтуїтивність інтерфейсу	Висока	Середня	Висока
Магазин Ресурсів	Asset Store	Marketplace	Немає

Вибір платформи для розробки.

Для створення гри у жанрі Tower Defense обрано платформу Unity. Це одне з найпопулярніших середовищ для розробки 2D та 3D ігор, яке пропонує широкий спектр можливостей для реалізації різноманітних ігрових механік та візуальних ефектів.

Основні переваги Unity:

1. Підтримка багатоплатформності: дозволяє розгортати гру на різних платформах, таких як Windows, macOS, iOS, Android та інші.
2. Інтуїтивний інтерфейс: полегшує процес розробки, особливо для новачків.
3. Велика спільнота: доступ до численних ресурсів, навчальних матеріалів та підтримки від спільноти розробників.

4. Багатий набір інструментів: вбудовані інструменти для роботи з графікою, фізикою, анімаціями та звуком [5].

Мова програмування.

Для написання скриптів у Unity було обрано C#.

1. Ця мова програмування має такі переваги:
2. Висока продуктивність: забезпечує швидке виконання коду.
3. Об'єктно-орієнтований підхід: дозволяє створювати модульний та масштабований код.
4. Широка підтримка: доступ до великої кількості бібліотек та фреймворків, що значно полегшує розробку [8, 9, 10].

Інструменти для дизайну.

Для створення графічних елементів гри використано такі інструменти:

1. Adobe Photoshop: потужний редактор растрової графіки, який дозволяє створювати високоякісні текстури, спрайти та інші елементи гри.
2. Adobe Illustrator: редактор векторної графіки, який використовується для створення чітких та масштабованих графічних об'єктів.

Інструменти для анімації.

Для створення анімацій персонажів та об'єктів гри використано:

1. Spine: спеціалізований інструмент для створення 2D анімацій, який інтегрується з Unity та дозволяє створювати складні анімаційні ефекти.
2. Unity Animator: вбудований в Unity інструмент для створення анімаційних схем та контролю за анімаціями в реальному часі.

Інструменти для звуку.

Для роботи зі звуком та створення аудіоефектів використано:

Audacity: безкоштовний редактор аудіо, який дозволяє записувати, редагувати та змішувати звукові файли.

FMOD: платформа для інтеграції звукових ефектів у гру, яка забезпечує високу якість звуку та підтримує інтерактивні аудіо-ефекти.

Система контролю версій.

Для управління версіями коду та співпраці в команді використано Git з хостингом на GitHub. Це забезпечує:

1. Безпеку даних: зберігання коду на віддалених серверах.
2. Співпраця: можливість одночасної роботи над проєктом кількох розробників.
3. Відстеження змін: зручний інтерфейс для відстеження історії змін та виправлення помилок.

1.2 Проєктування ігрового застосунку

Процес проєктування застосунку є важливим етапом у розробці гри у жанрі Tower Defense. Він включає визначення архітектури системи, розробку ігрових механік, дизайну інтерфейсу користувача та структури даних.

На рисунку 1.5 зображено стек використовуємих технологій.

Рисунок 1.5. Стек використовуємих технологій

На рисунку 1.6 зображено навігацію по графічному інтерфейсу ігрового застосунку.

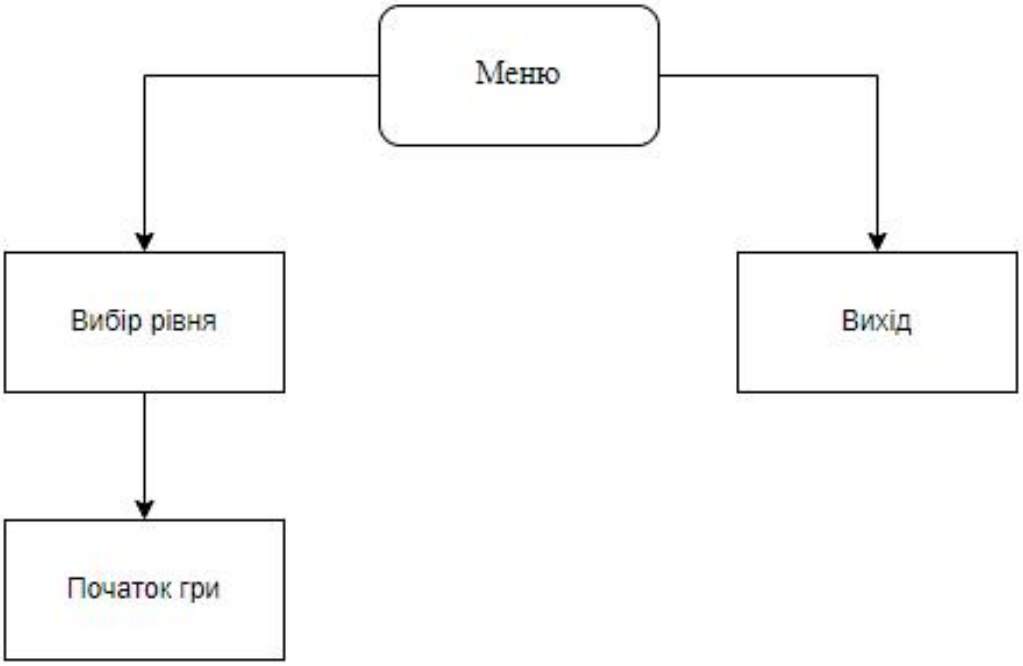


Рисунок 1.6. Навігація по ігровому застосунку

1.2.1 Архітектура системи

Архітектура гри в Unity 3D складається з асетів.

Асети в Unity 3D представляють собою цифрові файли, які використовуються для створення контенту в грі, таких як 3D-моделі, текстури, аудіо, анімації та сценарії. Вони можуть бути створені власноруч, завантажені з Інтернету або придбані в Unity Asset Store.

На рис. 1.7 зображено файлову архітектуру даного ігрового застосунку.

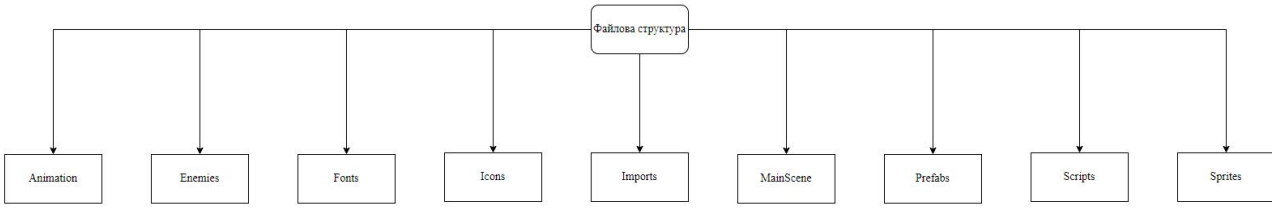


Рисунок 1.7. Файлова архітектура ігрового застосунку

На рис. 1.8 зображено фінальну файлову архітектуру в Unity3D.

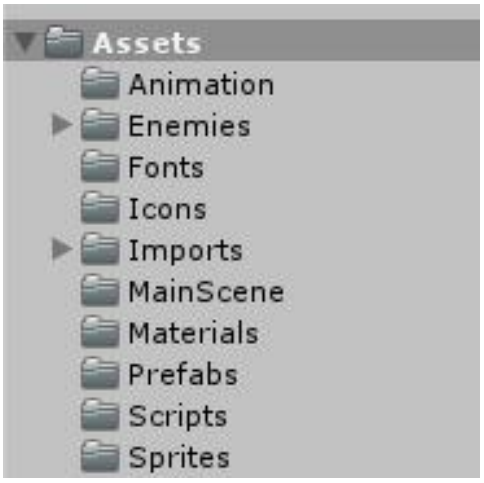


Рисунок 1.8. Фінальна файлова архітектуру в Unity3D

Розглянемо файлову архітектуру більш детально:

1. Animations: Ця папка призначена для зберігання анімацій, які застосовуються до 3D-моделей. Анімації можуть бути створені власноруч, завантажені з Інтернету або придбані в Unity Asset Store.
2. Enemies: Ця папка використовується для зберігання ворогів, які будуть присутні на рівнях, а також для зміни параметрів цих юнітів.
3. Fonts: Ця папка призначена для зберігання шрифтів, що використовуються в тексті. Шрифти можуть бути у форматах TTF, OTF або TrueType.
4. Icons: Ця папка використовується для зберігання іконок будівель для магазину. Іконки зберігаються у форматі PNG.
5. Imports: Ця папка призначена для зберігання моделей будівель у форматі FBX.
6. Materials: Ця папка використовується для зберігання матеріалів, які визначають зовнішній вигляд 3D-об'єктів. Матеріали включають текстури, колір, освітлення та інші властивості.
7. Prefabs: Ця папка призначена для зберігання префабів, які є шаблонами 3D-об'єктів. Префаби дозволяють швидко і легко створювати повторювані елементи у грі.
8. Scripts: Ця папка використовується для зберігання сценаріїв, які є фрагментами коду, що керують поведінкою 3D-об'єктів та ігровим процесом. Сценарії пишуться мовою C#.Розробка ігрових механік.

1.2.2 Розробка ігрових механік

Основні ігрові механіки для Tower Defense гри включають:

Розміщення веж:

1. Гравець має можливість розміщувати різні типи веж на визначених місцях карти.
2. Кожна вежа має свої характеристики (радіус атаки, швидкість стрільби, сила атаки).

Потік ворогів:

1. Вороги з'являються хвилями і рухаються по визначеному маршруту до кінцевої точки.
2. Кожен тип ворога має свої характеристики (швидкість, здоров'я, урон).

Оновлення веж: Гравець може покращувати вежі, збільшуючи їх потужність, швидкість стрільби та інші параметри.

Економіка гри: За знищення ворогів гравець отримує ресурси, які можна витратити на нові вежі або їх покращення.

Спеціальні вміння: Гравець має можливість використовувати спеціальні уміння, такі як атака по всій карті, уповільнення ворогів або миттєве знищення певної кількості ворогів.

1.2.3 Дизайн інтерфейсу користувача

Інтерфейс користувача повинен бути інтуїтивно зрозумілим та зручним для гравця.

Основні елементи інтерфейсу:

Панель управління: Включає кнопки для розміщення веж, використання спеціальних умінь, паузи гри та доступу до налаштувань.

Інформаційна панель: Відображає поточний рівень, кількість залишкових ворогів, ресурси гравця та здоров'я бази.

1.2.4 Структура даних гри

Структура даних для гри включає:

Дані про вежі: Описують типи веж, їх характеристики та рівні покращень.

Дані про ворогів: Описують типи ворогів, їх характеристики та особливості поведінки.

Дані про рівні: Включають інформацію про карти рівнів, розміщення точок для веж та маршрути руху ворогів.

Збереження прогресу: Містить дані про поточний прогрес гравця, досягнення, збереження гри та інші персональні налаштування.

1.3 Покрокова розробка гри

1.3.1 Перші кроки

Починаючи розробку гри на Unity, важливо спершу визначитися з тим, як вона виглядатиме і функціонуватиме. Ми вирішили створити 3D-візуалізацію для нашої Tower Defense (TD) гри. Основна ідея полягає в створенні карти, де є білі клітини для будівництва веж і червоні клітини, де будувати не можна. Для цього в лівій частині вікна обираємо 3D Object > Cube, створюючи необхідні клітини. Потім змінюємо їх колір, створюючи матеріали відповідних кольорів і перетягуючи їх на відповідні тайли.

На рисунку 1.9 зображено примітиви клітин.

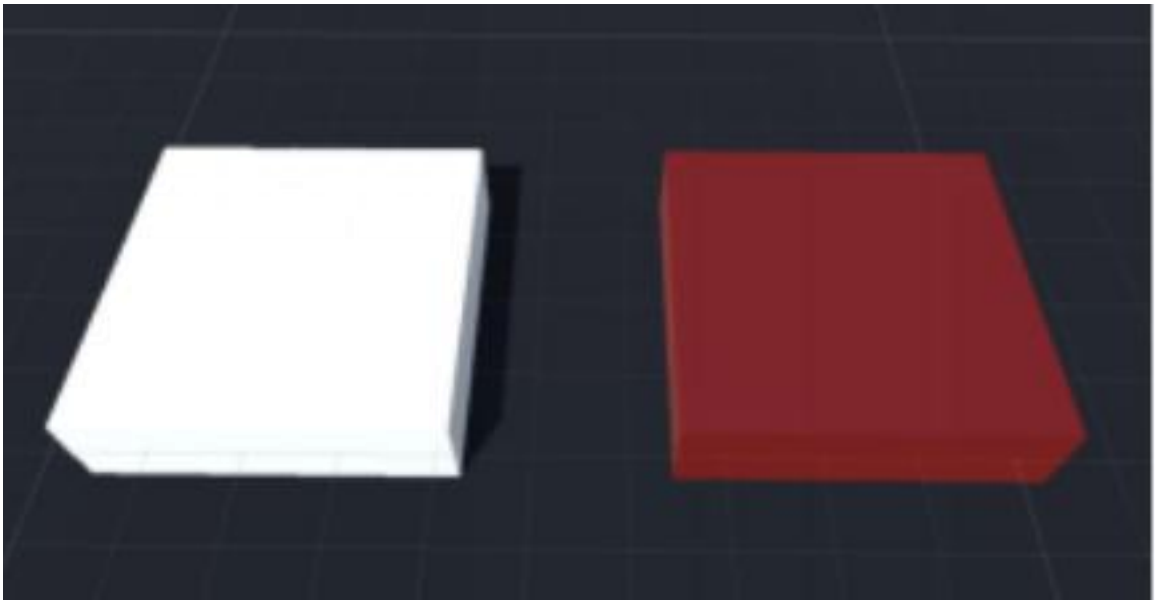


Рисунок 1.9. Примітиви клітин

Між клітинами має бути дорога, якою ходитимуть вороги. Дорогу створюємо за аналогією з червоними клітинами, але використовуємо сірий матеріал. Вона служить візуальним орієнтиром для маршруту, який долатимуть вороги. Також створюємо точки появи ворогів та кінцеву точку їх маршруту, позначивши їх кубами, збільшеними по вертикалі.

Зі створених тайлів формуємо базову карту. Вона не повинна бути великою, але повинна включати всі необхідні елементи. Далі створюємо точки маршруту (waypoints) для ворогів. Для цього створюємо порожній об'єкт, називаємо його waypoint, і розміщуємо його по всіх кутах дороги. Ці точки використовуватимуться для пересування ворогів.

На рисунку 1.10 зображено створення доріг для ворогів.

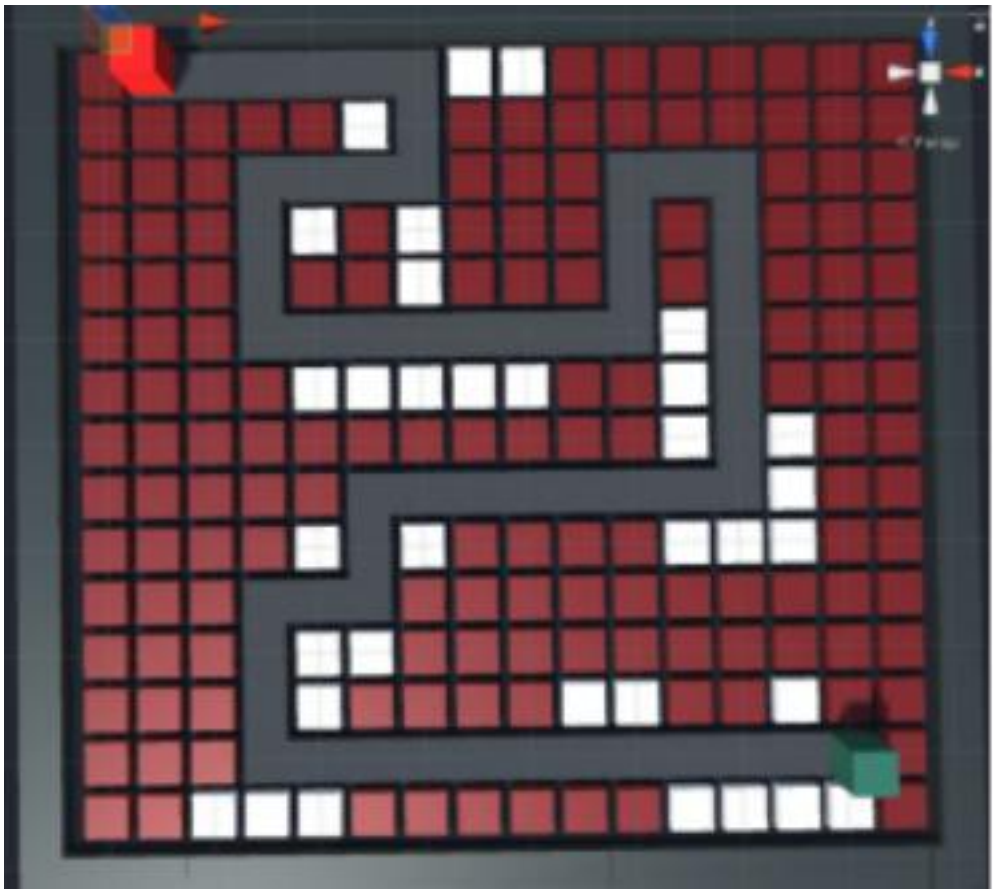


Рисунок 1.10. Створення доріг для ворогів

1.3.2 Створення противника

Далі ми створюємо ворогів, вони будуть з фіксованою швидкістю йти по точках з точки А.

Директива using UnityEngine; підключає простір імен UnityEngine, який містить базові класи та структури, необхідні для створення ігрових об'єктів та компонентів у Unity. Це включає роботу з ігровими об'єктами (GameObjects), компонентами (Components), трансформаціями (Transforms), фізикою, анімацією та багатьма іншими аспект ами.

Директива using UnityEngine.UI; підключає простір імен UnityEngine.UI, який містить класи та структури для роботи з елементами користувацького інтерфейсу (UI) у Unity. Це дозволяє вам створювати та керувати UI елементами, такими як кнопки, текстові поля, слайдери, зображення та інші компоненти інтерфейсу.

Нижче наведено код логіки з ворогами:

```
using UnityEngine;
using UnityEngine.UI;
public class Enemy : MonoBehaviour {
    public float startSpeed = 10f;
    [HideInInspector]
    public float speed;
    public float startHealth = 100;
    private float health;
    public int worth = 50;
    public GameObject deathEffect;
    [Header("Unity Stuff")]
    public Image healthBar;
    private bool isDead = false;
    void Start ()
    {
        speed = startSpeed;
        health = startHealth;
    }
    public void TakeDamage (float amount)
    {
        health -= amount;
        healthBar.fillAmount = health / startHealth;
        if (health <= 0 && !isDead)
        {
            Die();
        }
    }
    public void Slow (float pct)
    {
        speed = startSpeed * (1f - pct);
    }
```

```
    void Die ()
    {
        isDead = true;
        PlayerStats.Money += worth;
        GameObject effect = (GameObject)Instantiate(deathEffect, transform.positio
n, Quaternion.identity);
        Destroy(effect, 5f);
        WaveSpawner.EnemiesAlive--;
        Destroy(gameObject);
    }
}
```

Вороги будуть виглядати як кола різних кольорів, відрізнятимуться вони швидкістю та здоров'ям, сині противники будуть із середніми показниками, червоні матимуть багато здоров'я, але маленьку швидкість, а жовті будуть швидкими, але дуже слабкими за здоров'ям. З червоних випадатиме більше валюти, з синіх менше, а з жовтих майже не випадатиме.

На рисунку 1.11 зображено зовнішній вигляд ворогів.

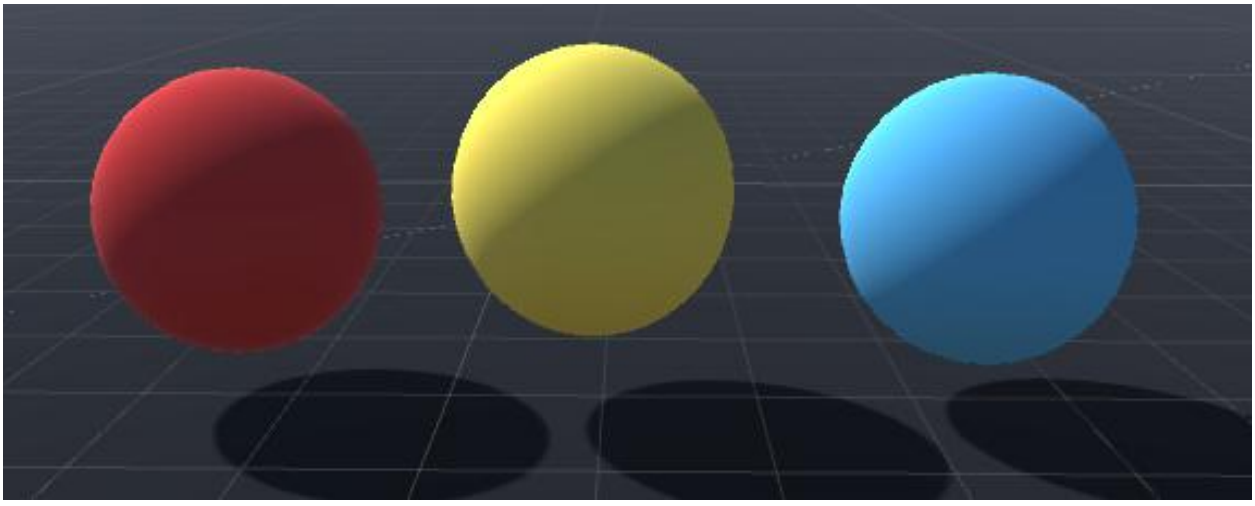


Рисунок 1.11. Зовнішній вигляд ворогів

1.3.3 Рух противника

Тепер ми повинні створити точки, якими ходитимуть противники

Тепер ми повинні створити точки, якими будуть ходити противники, ми створюємо порожній об'єкт, називаємо його waypoint.

На рисунку 1.12 зображено створення шляхів.

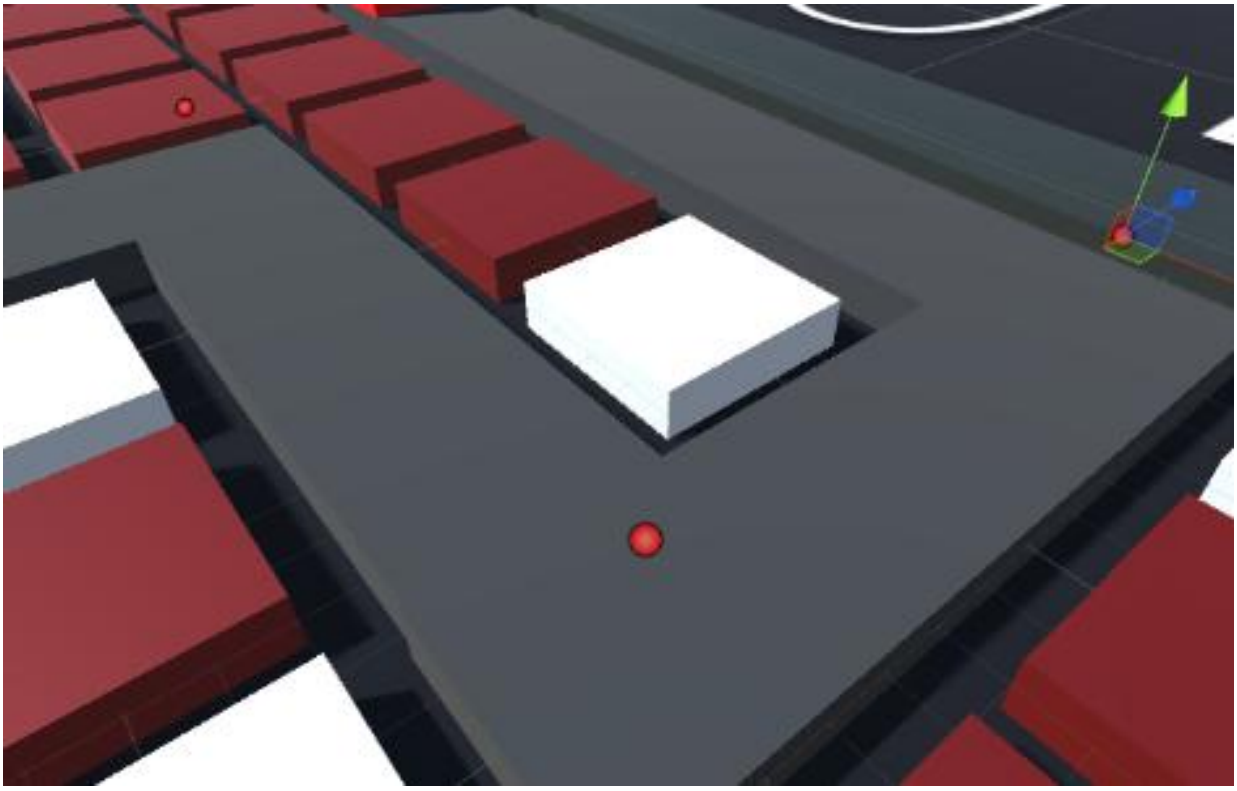


Рисунок 1.12. Створення шляхів

Ставимо цей об'єкт по всіх кутах дороги, пізніше ми прив'яжемо пересування супротивників до цих точок і вони слідуватимуть їм по черзі

Тепер ми створимо скрипт для роботи маркерів.

Нижче наведено код логіки для роботи маркерів:

```
using UnityEngine;
public class Waypoints : MonoBehaviour {
    public static Transform[] points;
    void Awake ()
    {
        points = new Transform[transform.childCount];
        for (int i = 0; i < points.Length; i++)
        {
            points[i] = transform.GetChild(i);
        }
    }
}
```

Для прив'язки скрипта до точок треба перетягнути його на префаб цих точок, так скрипт передасться всім точкам, які були створені.

На рис. 1.13 зображено Встановлення скрипту

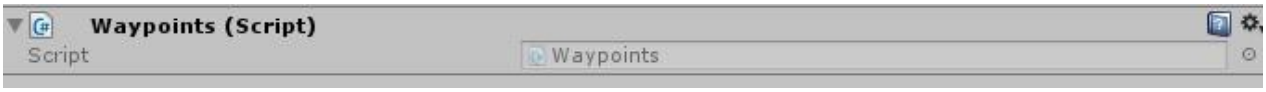


Рисунок 1.13 Встановлення скрипту

Далі ми створюємо скрипт, який спавнить противників у точці А.

Нижче наведено код логіки спавну противників.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class WaveSpawner : MonoBehaviour {
    public static int EnemiesAlive = 0;
    public Wave[] waves;
    public Transform spawnPoint;
    public float timeBetweenWaves = 5f;
    private float countdown = 2f;
    public Text waveCountdownText;
    public GameManager gameManager;
    private int waveIndex = 0;
    void Update ()
    {
        if (EnemiesAlive > 0)
        {
            return;
        }
        if (waveIndex == waves.Length)
        {
            gameManager.WinLevel();
            this.enabled = false;
        }

        if (countdown <= 0f)
        {
            StartCoroutine(SpawnWave());
            countdown = timeBetweenWaves;
            return;
        }
        countdown -= Time.deltaTime;
        countdown = Mathf.Clamp(countdown, 0f, Mathf.Infinity);
        waveCountdownText.text = string.Format("{0:00.00}", countdown);

\\ IEnumerator використовується для створення корутини, яка чекає 2 секунди перед виконанням наступного кроку. using System.Collections; потрібне, щоб забезпечити доступ до IEnumerator та інших колекційних класів
    }
    IEnumerator SpawnWave ()
    {
        PlayerStats.Rounds++;
        Wave wave = waves[waveIndex];
        EnemiesAlive = wave.count;
        for (int i = 0; i < wave.count; i++)
        {
            SpawnEnemy(wave.enemy);
            yield return new WaitForSeconds(1f / wave.rate);
        }
        waveIndex++;
    }
    void SpawnEnemy (GameObject enemy)
    {
        Instantiate(enemy, spawnPoint.position, spawnPoint.rotation);
    }
}
```

1.3.4 Створення веж

Далі ми створимо наші башти для захисту. Вони будуть 3 видів, кулеметна, ракетна, лазерна.

На рисунку 1.14 зображено типи веж.

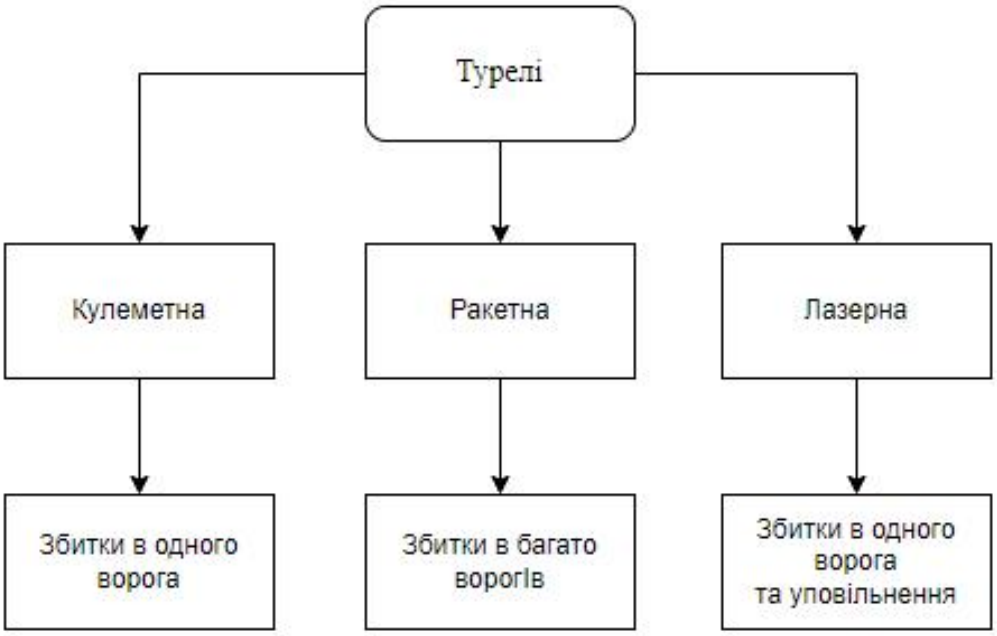


Рисунок 1.14. Типи веж

Почнемо з кулеметної. Для неї нам потрібно взяти текстуру з відкритих джерел або скористатися AssetStore для пошуку відповідної моделі. Сира турель без нанесення текстур виглядає так.

На рисунку 1.15 зображена кулеметна турель без матеріалів.

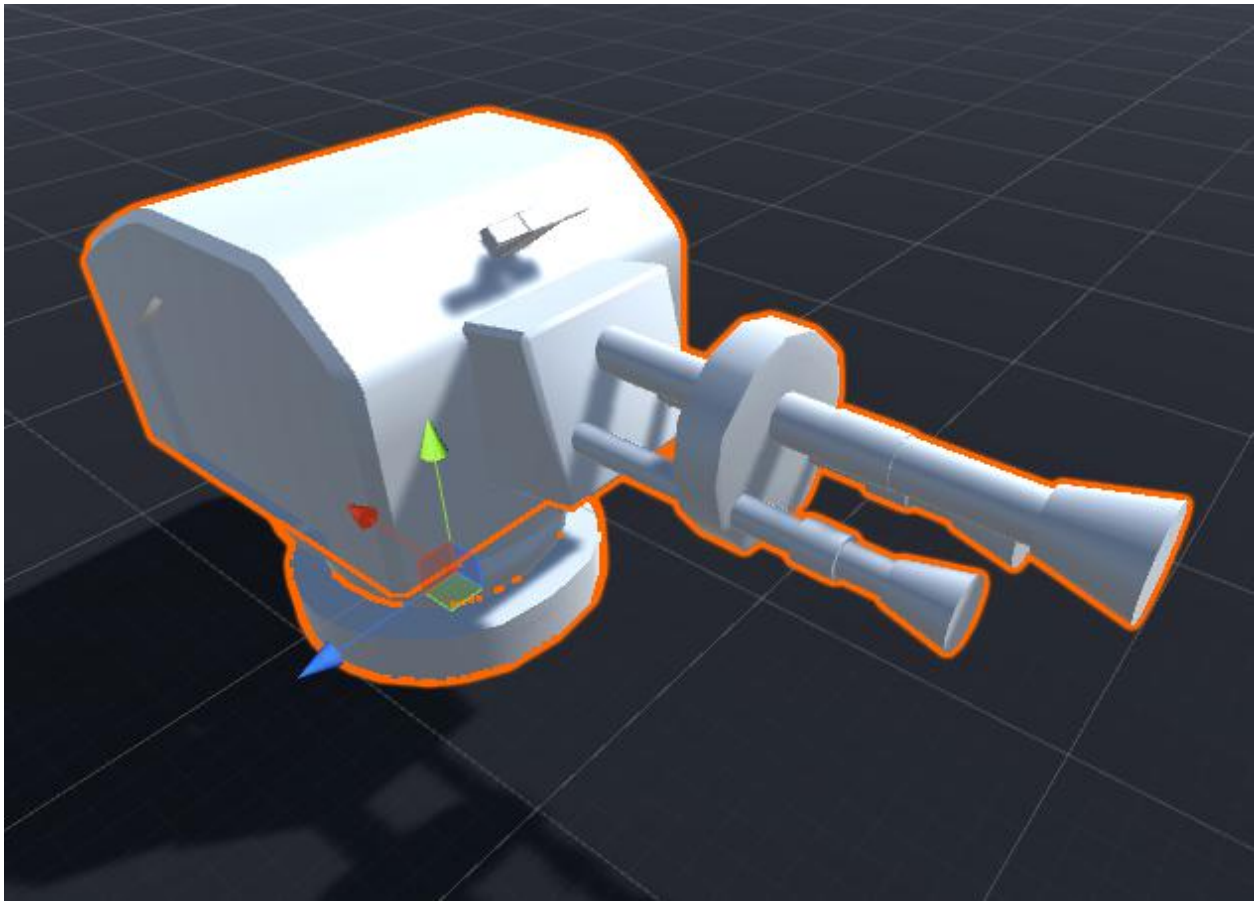


Рисунок 1.15. Кулеметна турель без матеріалів

На рисунку 1.16 зображена кулеметна турель з матеріалами.

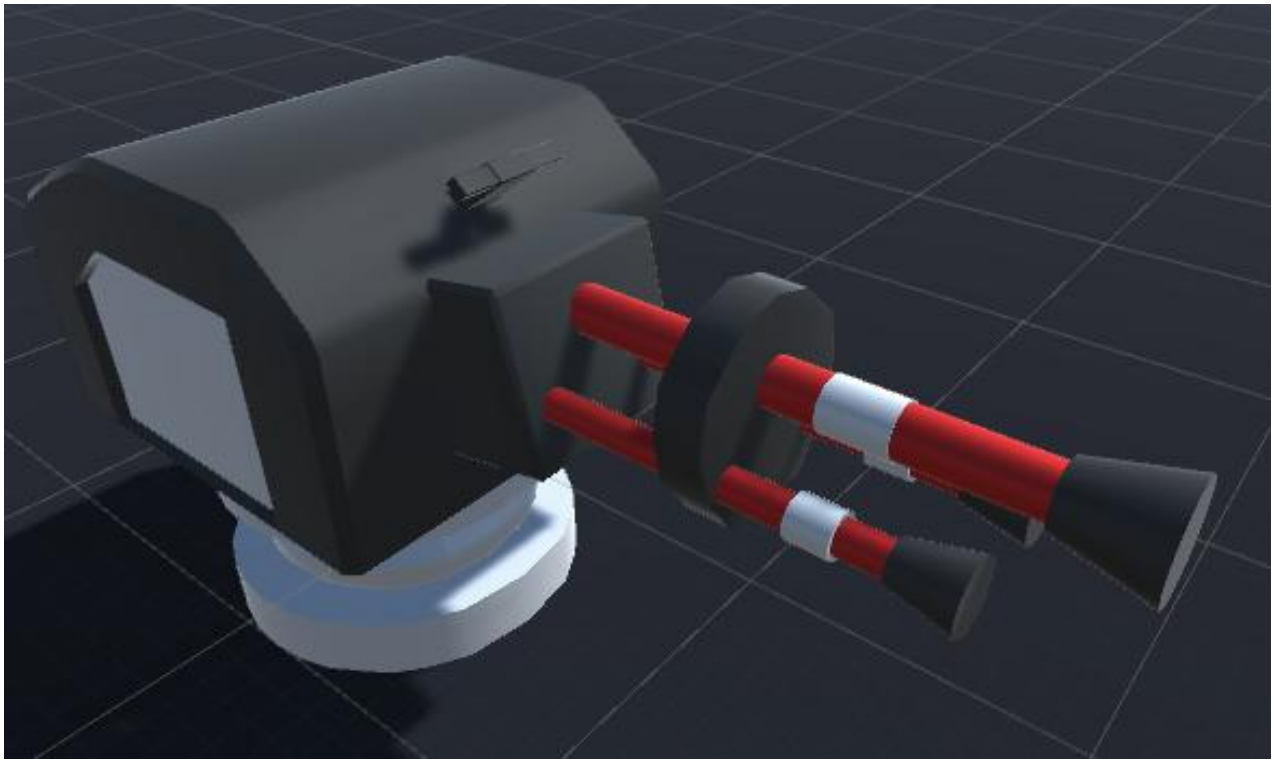


Рисунок 1.16. Кулеметна турель з матеріалами

Окремі деталі на ній можна за допомогою матеріалів можна перефарбовувати у різні кольори.

За підсумками наші будівлі виглядатимуть якось так

На рисунку 1.17 зображено ракетну та лазерну турелі.

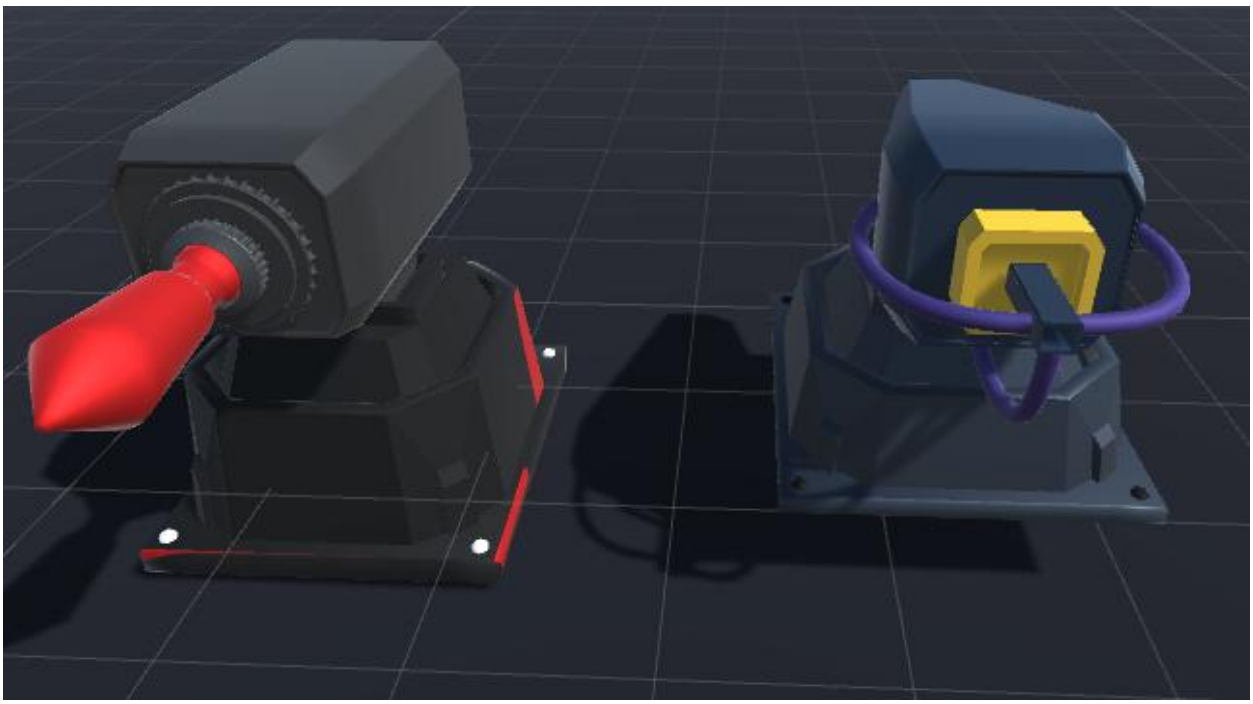


Рисунок 1.17. Ракетна та лазерна турелі

Розглянемо докладніше, чим відрізняються наші турелі та яку роль кожна з них відіграє в грі.

Кулеметна турель є базовою одиницею нашого арсеналу. Вона швидко стріляє, завдаючи невеликі пошкодження одному ворогу за раз. Її перевага полягає у високій швидкості вогню, що дозволяє ефективно знищувати менш потужних ворогів або значно послаблювати сильніших супротивників.

Ракетна турель діє по-іншому. Вона запускає снаряд, який вибухає при досягненні цілі, завдаючи шкоди всім ворогам у певному радіусі. Це робить її незамінною проти груп ворогів, дозволяючи завдавати масові пошкодження і швидко зменшувати чисельність противників на полі бою.

Лазерна турель має свої унікальні характеристики. Вона завдає значної шкоди одному ворогу, одночасно уповільнюючи його. Це корисно для

стратегічного знищення найбільш небезпечних ворогів або для зупинки швидких ворогів, даючи іншим турелям більше часу для нанесення шкоди.

Таким чином, кожен тип турелі має свої сильні та слабкі сторони, і їх правильне комбінування на полі бою дозволить гравцеві ефективно захищати свої позиції від різноманітних хвиль ворогів. Стратегічне розміщення і використання різних турелей стане ключем до успіху в нашій грі.

Нижче наведено код логіки турелей:

```
using UnityEngine;
using System.Collections;
public class Turret : MonoBehaviour {
    private Transform target;
    private Enemy targetEnemy;
    [Header("General")]
    public float range = 15f;
    [Header("Use Bullets (default)")]
    //Кулеметна турель
    public GameObject bulletPrefab;
    public float fireRate = 1f;
    private float fireCountdown = 0f;
    [Header("Use Laser")]
    public bool useLaser = false;
    public int damageOverTime = 30;
    public float slowAmount = .5f;
    public LineRenderer lineRenderer;
    public ParticleSystem impactEffect;
    public Light impactLight;
    [Header("Unity Setup Fields")]
    public string enemyTag = "Enemy";
    public Transform partToRotate;
    public float turnSpeed = 10f;
    public Transform firePoint;
    // Використовуйте це для ініціалізації
    void Start () {
        InvokeRepeating("UpdateTarget", 0f, 0.5f);
    }
    void UpdateTarget ()
    {
       GameObject[] enemies = GameObject.FindGameObjectsWithTag(enemyTag);
        float shortestDistance = Mathf.Infinity;
        GameObject nearestEnemy = null;
        foreach (GameObject enemy in enemies)
        {
            float distanceToEnemy = Vector3.Distance(transform.position, enemy.tra
nsform.position);
            if (distanceToEnemy < shortestDistance)
            {
                shortestDistance = distanceToEnemy;
                nearestEnemy = enemy;
            }
        }
        if (nearestEnemy != null && shortestDistance <= range)
        {
            target = nearestEnemy.transform;
```

```
                targetEnemy = nearestEnemy.GetComponent<Enemy>();
            } else
            {
                target = null;
            }
        }
        // Оновлення викликається один раз на кадр
        void Update () {
            if (target == null)
            {
                if (useLaser)
                {
                    if (lineRenderer.enabled)
                    {
                        lineRenderer.enabled = false;
                        impactEffect.Stop();
                        impactLight.enabled = false;
                    }
                }
                return;
            }
            LockOnTarget();
            if (useLaser)
            {
                Laser();
            } else
            {
                if (fireCountdown <= 0f)
                {
                    Shoot();
                    fireCountdown = 1f / fireRate;
                }
                fireCountdown -= Time.deltaTime;
            }
        }
        void LockOnTarget ()
        {
            Vector3 dir = target.position - transform.position;
            Quaternion lookRotation = Quaternion.LookRotation(dir);
            Vector3 rotation = Quaternion.Lerp(partToRotate.rotation, lookRotation, Ti
    me.deltaTime * turnSpeed).eulerAngles;
            partToRotate.rotation = Quaternion.Euler(0f, rotation.y, 0f);
        }
        void Laser ()
        {
            targetEnemy.TakeDamage(damageOverTime * Time.deltaTime);
            targetEnemy.Slow(slowAmount);
            if (!lineRenderer.enabled)
            {
                lineRenderer.enabled = true;
                impactEffect.Play();
                impactLight.enabled = true;
            }
            lineRenderer.SetPosition(0, firePoint.position);
            lineRenderer.SetPosition(1, target.position);
            Vector3 dir = firePoint.position - target.position;
            impactEffect.transform.position = target.position + dir.normalized;
            impactEffect.transform.rotation = Quaternion.LookRotation(dir);
        }
        void Shoot ()
        {
```

```
        GameObject bulletGO = (GameObject)Instantiate(bulletPrefab, firePoint.posi
tion, firePoint.rotation);
        Bullet bullet = bulletGO.GetComponent<Bullet>();
        if (bullet != null)
            bullet.Seek(target);
    }
    void OnDrawGizmosSelected ()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, range);
    }
}
```

Тепер ми переходимо до створення коду для наших снарядів, які використовуватимуть кулеметні та ракетні турелі. Це важливий етап, адже правильна реалізація снарядів забезпечить ефективну роботу турелей та вплине на загальну динаміку гри. Ми ретельно пропишемо логіку запуску, траєкторії польоту та влучання снарядів, щоб забезпечити реалістичність та задоволення від ігрового процесу. Цей крок включає детальне налаштування фізичних параметрів снарядів, таких як швидкість, радіус ураження та візуальні ефекти, які супроводжуватимуть їх політ і вибух.

Нижче наведено код створення снарядів:

```
using UnityEngine;
public class Bullet : MonoBehaviour {
    private Transform target;
    public float speed = 70f;
    public int damage = 50;
    public float explosionRadius = 0f;
    public GameObject impactEffect;
    public void Seek (Transform _target)
    {
        target = _target;
    }
    // Оновлення викликається один раз на кадр
    void Update () {
        if (target == null)
        {
            Destroy(gameObject);
            return;
        }
        Vector3 dir = target.position - transform.position;
        float distanceThisFrame = speed * Time.deltaTime;
        if (dir.magnitude <= distanceThisFrame)
        {
            HitTarget();
            return;
        }
        transform.Translate(dir.normalized * distanceThisFrame, Space.World);
        transform.LookAt(target);
    }
    void HitTarget ()
```

```
        {
            GameObject effectIns = (GameObject)Instantiate(impactEffect, transform.pos
    ition, transform.rotation);
            Destroy(effectIns, 5f);
            if (explosionRadius > 0f)
            {
                Explode();
            } else
            {
                Damage(target);
            }
            Destroy(gameObject);
        }
        void Explode ()
        {
            Collider[] colliders = Physics.OverlapSphere(transform.position, explosion
    Radius);
            foreach (Collider collider in colliders)
            {
                if (collider.tag == "Enemy")
                {
                    Damage(collider.transform);
                }
            }
        }
        void Damage (Transform enemy)
        {
            Enemy e = enemy.GetComponent<Enemy>();
            if (e != null)
            {
                e.TakeDamage(damage);
            }
        }
        void OnDrawGizmosSelected ()
        {
            Gizmos.color = Color.red;
            Gizmos.DrawWireSphere(transform.position, explosionRadius);
        }
    }
```

1.3.5 Система "Nodes"

Далі ми створюємо систему “Nodes”, на якій ми зможемо ставити, продавати і поліпшувати турелі.

Нижче наведено код логіки системи “Nodes”:

```
using UnityEngine;
using UnityEngine.EventSystems;
public class Node : MonoBehaviour
{
    public Color hoverColor;
    public Color notEnoughMoneyColor;
    public Vector3 positionOffset;
    [HideInInspector]
    public GameObject turret;
    [HideInInspector]
    public TurretBlueprint turretBlueprint;
    [HideInInspector]
```

```
    public bool isUpgraded = false;
    private Renderer rend;
    private Color startColor;
    BuildManager buildManager;
    void Start ()
    {
        rend = GetComponent<Renderer>();
        startColor = rend.material.color;
        buildManager = BuildManager.instance;
    }
    public Vector3 GetBuildPosition ()
    {
        return transform.position + positionOffset;
    }
    void OnMouseDown ()
    {
        if (EventSystem.current.IsPointerOverGameObject())
            return;
        if (turret != null)
        {
            buildManager.SelectNode(this);
            return;
        }
        if (!buildManager.CanBuild)
            return;
        BuildTurret(buildManager.GetTurretToBuild());
    }
    void BuildTurret (TurretBlueprint blueprint)
    {
        if (PlayerStats.Money < blueprint.cost)
        {
            Debug.Log("Not enough money to build that!");
            return;
        }
        PlayerStats.Money -= blueprint.cost;
        GameObject _turret = (GameObject)Instantiate(blueprint.prefab, GetBuildPos
ition(), Quaternion.identity);
        turret = _turret;
        turretBlueprint = blueprint;
        GameObject effect = (GameObject)Instantiate(buildManager.buildEffect, GetB
uildPosition(), Quaternion.identity);
        Destroy(effect, 5f);
        Debug.Log("Turret build!");
    }
    public void UpgradeTurret ()
    {
        if (PlayerStats.Money < turretBlueprint.upgradeCost)
        {
            Debug.Log("Not enough money to upgrade that!");
            return;
        }
        PlayerStats.Money -= turretBlueprint.upgradeCost;
        // Позбавтеся від старої башти
        Destroy(turret);
        // Побудуйте нову вежу
        GameObject _turret = (GameObject)Instantiate(turretBlueprint.upgradedPrefa
b, GetBuildPosition(), Quaternion.identity);
        turret = _turret;
        GameObject effect = (GameObject)Instantiate(buildManager.buildEffect, GetB
uildPosition(), Quaternion.identity);
        Destroy(effect, 5f);
```

```
            isUpgraded = true;
            Debug.Log("Turret upgraded!");
        }
        public void SellTurret ()
        {
            PlayerStats.Money += turretBlueprint.GetSellAmount();
            GameObject effect = (GameObject)Instantiate(buildManager.sellEffect, GetBu
ildPosition(), Quaternion.identity);
            Destroy(effect, 5f);
            Destroy(turret);
            turretBlueprint = null;
        }
        void OnMouseEnter ()
        {
            if (EventSystem.current.IsPointerOverGameObject())
                return;
            if (!buildManager.CanBuild)
                return;
            if (buildManager.HasMoney)
            {
                rend.material.color = hoverColor;
            } else
            {
                rend.material.color = notEnoughMoneyColor;
            }
        }
        void OnMouseExit ()
        {
            rend.material.color = startColor;
        }
}
```

1.3.6 Валюта та магазин

Далі ми зробимо важливу частину, це здоров'я та валюта. При проходженні ворогів у точку B, ми витрачатимемо здоров'я, а за валюти ми зможемо купувати та покращувати будівлі.

Нижче наведено код логіки системи здоров’я та валюти:

```
using UnityEngine;
using System.Collections;
public class PlayerStats : MonoBehaviour {
    public static int Money;
    public int startMoney = 400;
    public static int Lives;
    public int startLives = 5;
    public static int Rounds;
    void Start ()
    {
        Money = startMoney;
        Lives = startLives;
        Rounds = 0;
    }
}
```

У цьому скрипті у нас буде кількість грошей, яка доступна у нас зі старту та кількість здоров'я, далі ми посилатимемо на цей скрипт для підрахунку кількості пройдених хвиль.

Далі ми перейдемо до створення системи купівлі будівель, що дозволить гравцеві купувати і розміщувати різні споруди на карті, додаючи більше глибини та стратегічних можливостей до гри.

Нижче наведено код логіки системи купівель:

```
public GameObject buildEffect;
    public GameObject sellEffect;
    private TurretBlueprint turretToBuild;
    private Node selectedNode;
    public NodeUI nodeUI;
    public bool CanBuild { get { return turretToBuild != null; } }
    public bool HasMoney { get { return PlayerStats.Money >= turretToBuild.cost; }
 }
    public void SelectNode (Node node)
    {
        if (selectedNode == node)
        {
            DeselectNode();
            return;
        }
        selectedNode = node;
        turretToBuild = null;
        nodeUI.SetTarget(node);
    }
    public void DeselectNode()
    {
        selectedNode = null;
        nodeUI.Hide();
    }
    public void SelectTurretToBuild (TurretBlueprint turret)
    {
        turretToBuild = turret;
        DeselectNode();
    }
    public TurretBlueprint GetTurretToBuild ()
    {
        return turretToBuild;
    }
}
```

Далі ми зробимо систему покращень та продажу будівель. Вартість продажу ми робимо вдвічі меншою за покупку.

Нижче наведено код логіки системи покращень:

```
using UnityEngine;
using System.Collections;
[System.Serializable]
public class TurretBlueprint {
    public GameObject prefab;
```

```
    public int cost;
    public GameObject upgradedPrefab;
    public int upgradeCost;
    public int GetSellAmount ()
    {
        return cost / 2;
    }
}
```

Вона буде прив'язуватися до системи Nodes і перевіряти, чи є турель на клітці, чи можна її покращити або продати.

1.3.7 Система закінчення гри

Так виглядатиме скрипт, який обробляє програш гравця в грі, забезпечуючи відповідні дії та візуальні ефекти при досягненні цієї ситуації.

Нижче наведено код логіки закінчення гри:

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class GameOver : MonoBehaviour {
    public string menuSceneName = "MainMenu";
    public SceneFader sceneFader;
    public void Retry ()
    {
        sceneFader.FadeTo(SceneManager.GetActiveScene().name);
    }
    public void Menu ()
    {
        sceneFader.FadeTo(menuSceneName);
    }
}
```

Скрипт нижче буде в кінці гри робити перевірку на проходження рівня, якщо здоров'я користувача буде дорівнювати 0, він перекидатиме на екран програшу, а якщо гравець пройшов рівень, то даватиме вибір, пройти наступний або вийти в меню.

Нижче наведено код логіки перевірки програшу:

```
using UnityEngine;
using System.Collections;
public class GameManager : MonoBehaviour {
    public static bool GameIsOver;
    public GameObject gameOverUI;
    public GameObject completeLevelUI;
    void Start ()
    {
        GameIsOver = false;
    }
    // Оновлення викликається один раз на кадр
    void Update () {
        if (GameIsOver)
```

```
            return;
        if (PlayerStats.Lives <= 0)
        {
            EndGame();
        }
    }
    void EndGame ()
    {
        GameIsOver = true;
        gameOverUI.SetActive(true);
    }
    public void WinLevel ()
    {
        GameIsOver = true;
        completeLevelUI.SetActive(true);
    }
}
```

1.3.8 Головне меню

Для створення головного меню я скопіював перший рівень і видалив усе, крім камери і поставив у центр екрану турель, після цього я додав їй скрипт на поворот і поруч із нею поставив закопану в землю лазерну та ракетну турелі.

На рис. 1.18 зображено головне меню.

Рисунок 1.18 Головне меню

На турелі я зробив кнопку грати, а на дулі я прикріпив кнопку вийти

Ми створюємо скрипт головного меню, в якому він при натисканні кнопки "play" відкриває сцену вибору рівня, а при натисканні на "Exit" виходить із гри.

Нижче наведено код логіки виходу з гри:

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class MainMenu : MonoBehaviour {
    public string levelToLoad = "MainLevel";
    public SceneFader sceneFader;
    public void Play ()
    {
        sceneFader.FadeTo(levelToLoad);
    }
    public void Quit ()
    {
        Debug.Log("Exciting...");
        Application.Quit();
    }
}
```

1.3.9 Вибір рівня

Далі ми створюємо скрипт для вибору рівня. Він нам потрібний щоб прив'язувати до кнопок можливість вибору.

Нижче наведено код логіки вибору рівня:

```
using UnityEngine;
using UnityEngine.UI;
public class LevelSelector : MonoBehaviour {
    public SceneFader fader;
    public Button[] levelButtons;
    void Start ()
    {
        int levelReached = PlayerPrefs.GetInt("levelReached", 1);
        for (int i = 0; i < levelButtons.Length; i++)
        {
            if (i + 1 > levelReached)
                levelButtons[i].interactable = false;
        }
    }
    public void Select (string levelName)
    {
        fader.FadeTo(levelName);
    }
}
```

Після цього ми створюємо кнопку і даємо їй властивість "OnCLick" на вибір потрібного рівня

На рисунку 1.19 зображено створення вибору рівня натискання кнопки.

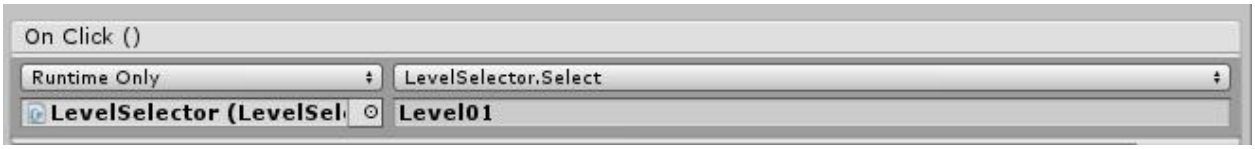


Рисунок 1.19. Створення вибору рівня натискання кнопки

Створюємо "title" та вписуємо туди "Select Level".

На рисунку 1.20 зображено меню вибору рівня.

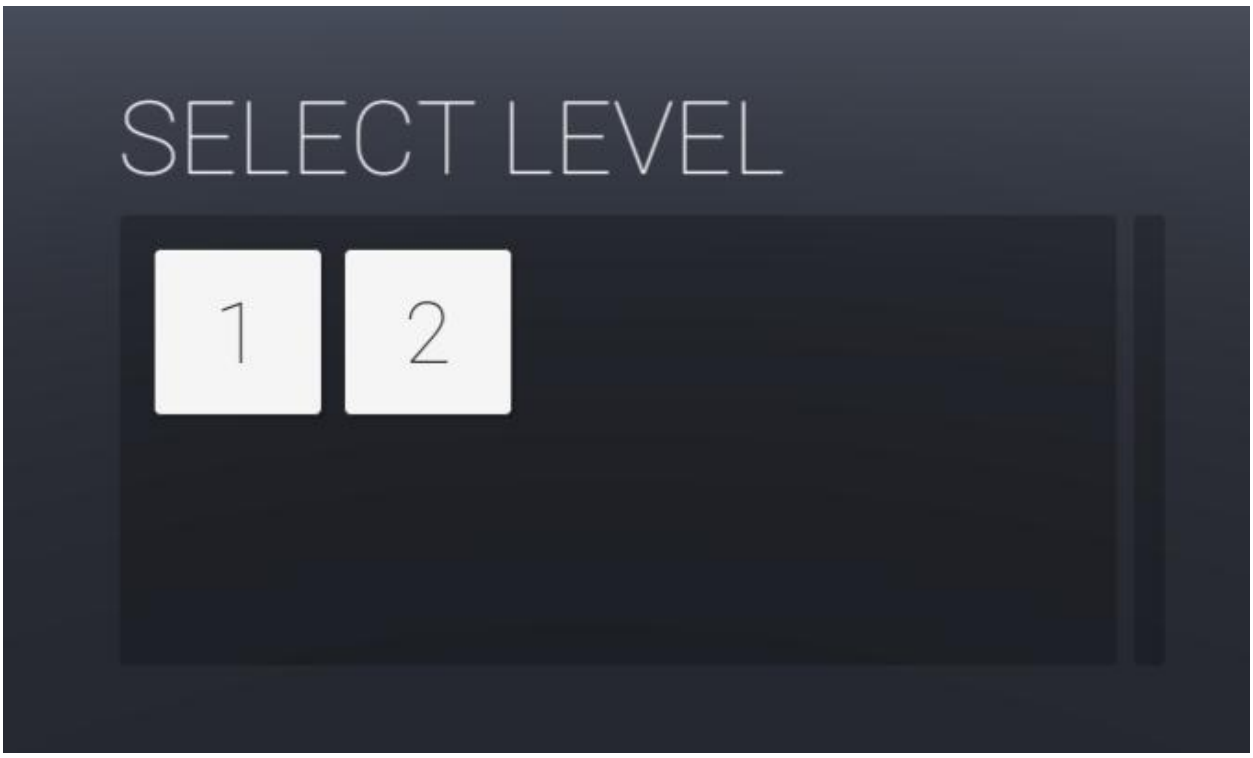


Рисунок 1.20. Меню вибору рівня

Так у нас з'явиться можливість обирати між нашими рівнями.

Далі ми компілюємо гру використовуючи параметр "build settings" у "File" У нас з'являється .Exe файл, в який ми зайдемо і відкриється наша гра.

1.4 Тестування гри

Тестування є невід'ємною частиною розробки гри, яка допомагає виявити та виправити помилки, забезпечити стабільність і продуктивність, а також покращити користувацький досвід. Розглянемо цей процес більш детально.

1.4.1 Методи тестування

Ручне тестування.

Ручне тестування проводиться безпосередньо тестувальниками, які запускають гру і перевіряють її функціональність, механіки та інтерфейс користувача.

Основні етапи включають:

1. Перевірку правильності спавну ворогів.
2. Тестування розміщення та роботи башт.
3. Перевірку руху ворогів по маршруту.
4. Оцінку системи здоров'я та валюти.
5. Тестування кінцевих умов гри (виграш/програш).

Автоматизоване тестування.

Автоматизоване тестування включає використання спеціальних програм і скриптів для автоматичної перевірки гри. Це дозволяє швидше і ефективніше виявляти помилки, які могли б залишитися непоміченими при ручному тестуванні.

Основні види автоматизованого тестування:

1. Юніт-тести для перевірки окремих модулів гри.
2. Інтеграційні тести для перевірки взаємодії між різними компонентами.
3. Стрес-тести для оцінки стабільності гри під високим навантаженням.

Так само є види тестування білої, сірої та чорної скриньки

Тестування білого, чорного і сірого ящика - це три основні методи тестування програмного забезпечення.

Продовжимо опис тестування черного і серого ящика:

Тестування білого ящика (White-box testing).

Опис: Тестування білого ящика включає в себе аналіз внутрішньої структури програми та коду.

Мета: Перевірка правильності логіки програми, виконання всіх можливих шляхів і умов.

Методи: Включає в себе тестування різних гілок коду, підходів до покриття коду, підрахунок цикломатичного числа, тестування викликів підпрограм.

Тестування сірого ящика (Grey-box testing).

Опис: Тестування сірого ящика поєднує в собі елементи як тестування білого, так і чорного ящика.

Мета: Перевірка функціональності та внутрішніх структур програми.

Методи: Тестування здійснюється на основі знань про внутрішній код програми, але не на рівні деталей, що використовується для тестування білого ящика.

Тестування чорного ящика (Black-box testing).

Опис: Тестування чорного ящика базується на специфікації програми без врахування внутрішньої структури або коду.

Мета: Перевірка функціональності програми, відповідності вимогам та очікуванням користувача.

Методи: Тестування здійснюється через введення тестових даних і перевірку вихідних результатів без знання внутрішніх деталей реалізації.

На рисунку 1.21 зображено метод тестування ящиків

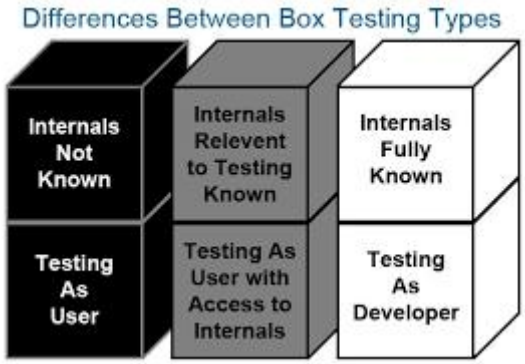


Рисунок 1.21. Метод тестування ящиків

1.4.2 План тестування

Визначення цілей тестування.

Основна мета тестування полягає у забезпеченні правильності роботи всіх ігрових механік, стабільності гри за різних умов і відповідності вимогам користувачів.

Розробка тестових сценаріїв.

Тестові сценарії описують конкретні дії, які необхідно виконати для перевірки певної функціональності гри. Наприклад:

1. Розміщення башти на білому тайлі та перевірка її роботи.
2. Спавн ворогів та їхній рух по маршруту.
3. Зменшення кількості життів при досягненні кінцевої точки ворогами.

Виконання тестів.

Тестувальники виконують розроблені тестові сценарії, фіксують результати та виявлені помилки. Якщо помилки знайдено, їх документують та передають розробникам для виправлення.

1.4.3 Аналіз результатів тестування

Виявлення та класифікація помилок.

Усі знайдені помилки класифікуються за критичністю та впливом на гру, що дозволяє пріоритезувати їхнє виправлення.

Виправлення помилок.

Розробники аналізують та виправляють виявлені помилки. Після цього проводиться повторне тестування для перевірки, чи були помилки успішно виправлені та чи не виникли нові проблеми.

Оцінка якості гри.

Після завершення всіх етапів тестування проводиться загальна оцінка якості гри, враховуючи стабільність, продуктивність, відсутність критичних помилок та відповідність вимогам.

1.4.4 Початок тестування

Перше що ми бачимо при вході в гру, це конфігуратор юніті, в якому можна вибрати роздільну здатність екрана, налаштування графіки та вибір монітора

На рисунку 1.22 зображено конфігуратор Unity.

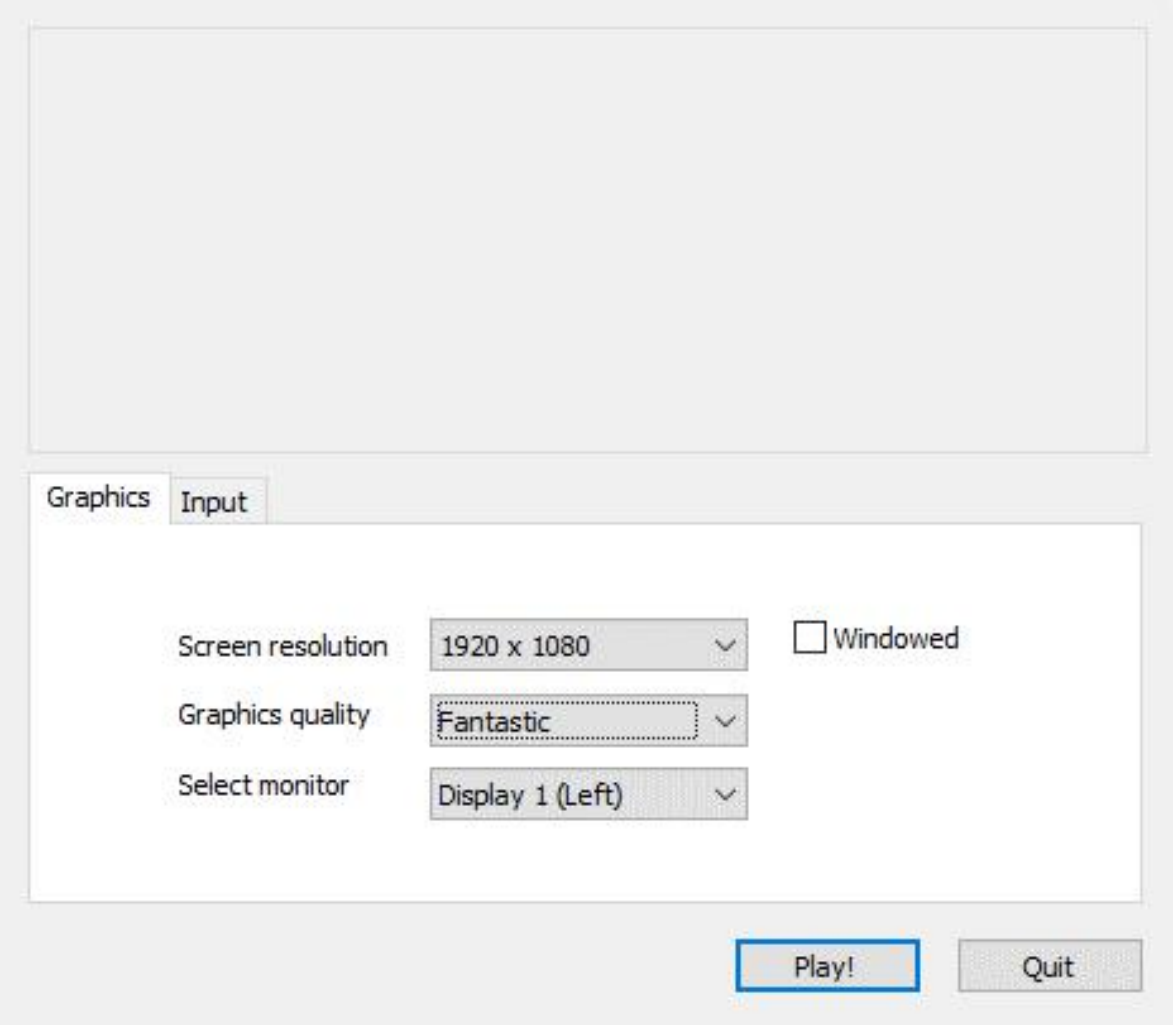


Рисунок 1.22. Конфігуратор Unity

Після вибору налаштувань та заходу у гру, ми бачимо головне меню, при натисканні на кнопку вихід, гра закривається. При натисканні на кнопку "Play" ми переходимо в меню вибору рівня.

На рис. 1.23 зображено головне меню.

Рисунок 1.23. Головне меню

На рисунку 1.24 зображено вибір рівня.

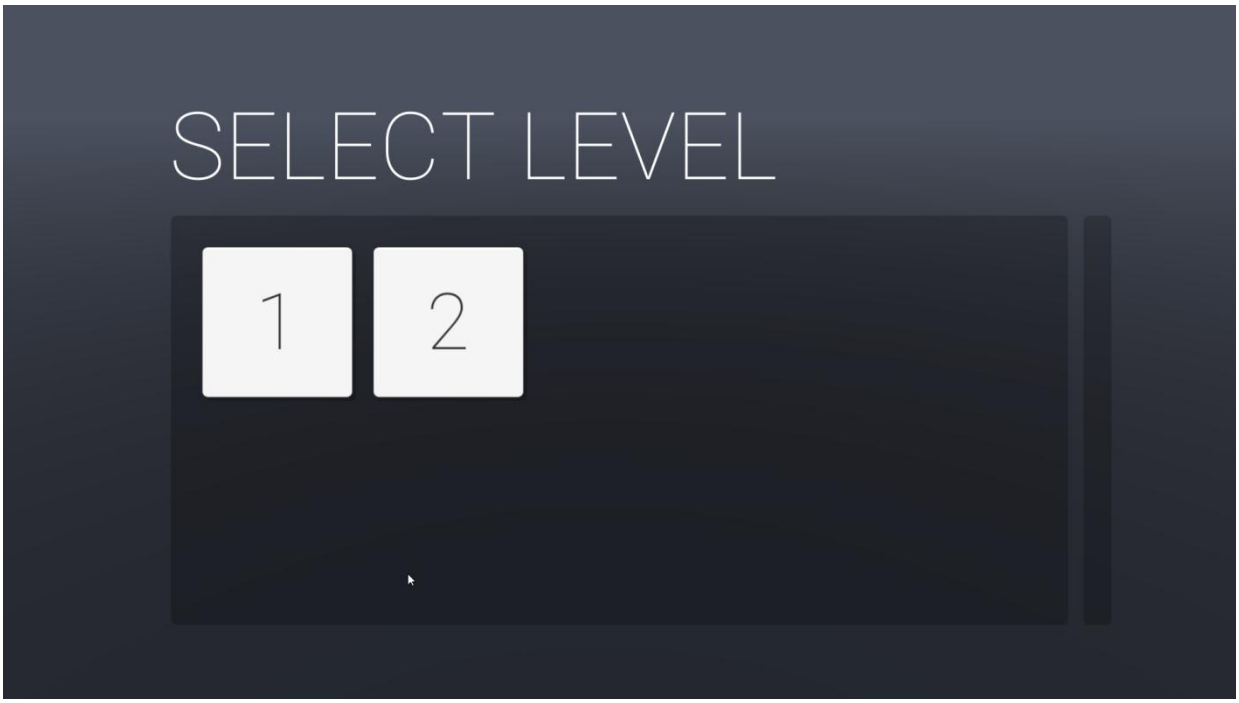


Рисунок 1.24. Вибір рівня

Коли ми входимо на рівень, наш персонаж з'являється на карті. У нашому розпорядженні є певна сума грошей і доступ до магазину. Ми можемо вибрати та придбати один із трьох типів будівель: кулеметну турель, ракетну турель або

лазерну турель. Крім того, у верхній частині екрана відображається кількість нашого здоров'я.

На рисунку 1.25 зображено початок рівня.

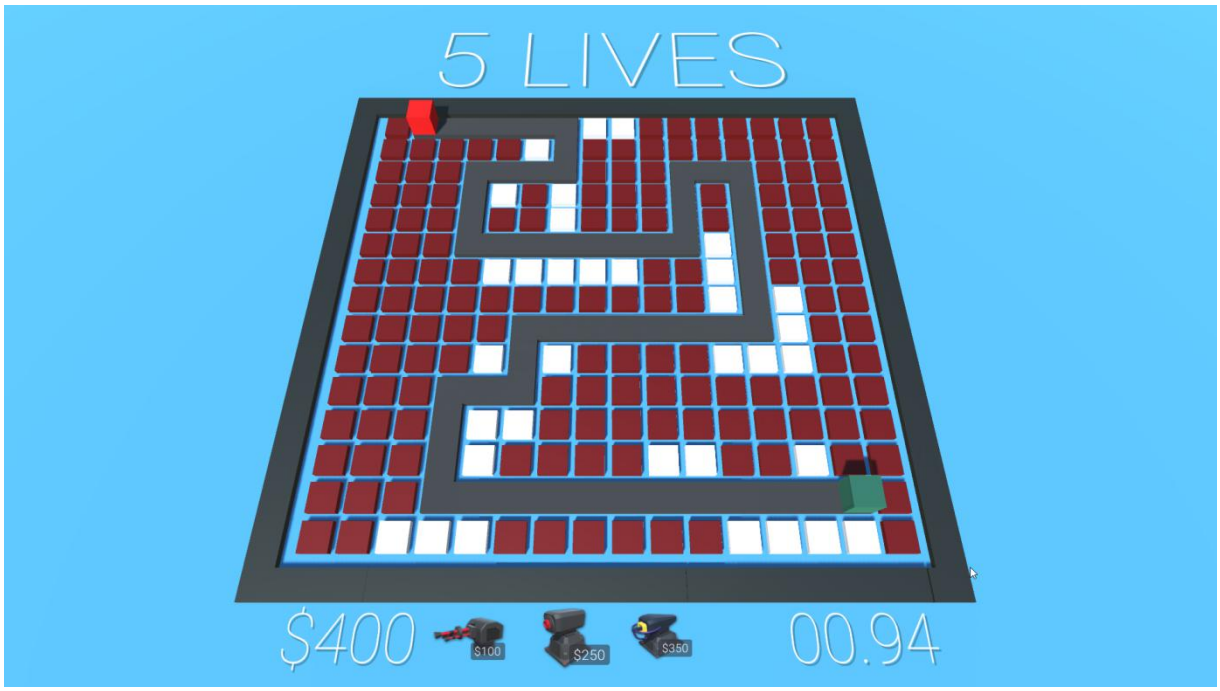


Рисунок 1.25. Початок рівня

Після початку раунду з'являються вороги, які рухаються за заздалегідь визначеним маршрутом. Для захисту від них ми розміщуємо турелі.

На рисунку 1.26 зображено початковий етап рівня.

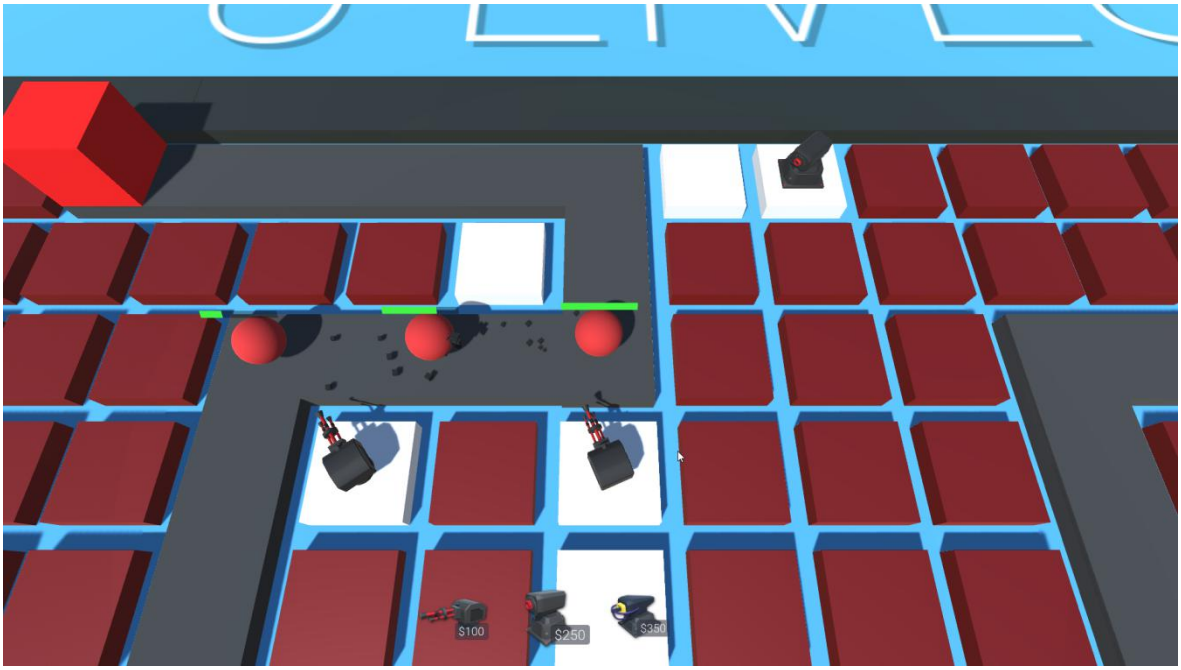


Рисунок 1.26. Початковий етап рівня

Кулеметна турель завдає значної шкоди одному противнику. Ракетна турель завдає середньої шкоди всім противникам у зоні ураження. Лазерна турель завдає невеликої шкоди, але уповільнює одного ворога.

На рисунку 1.27 зображено як виглядає лазер.

Рисунок 1.27. Як виглядає лазер

При натисканні на кнопку "Escape" гравець має можливість призупинити гру. У режимі паузи доступні опції для продовження гри, повторного проходження рівня та повернення до головного меню.

На рисунку 1.28 зображено меню паузи.

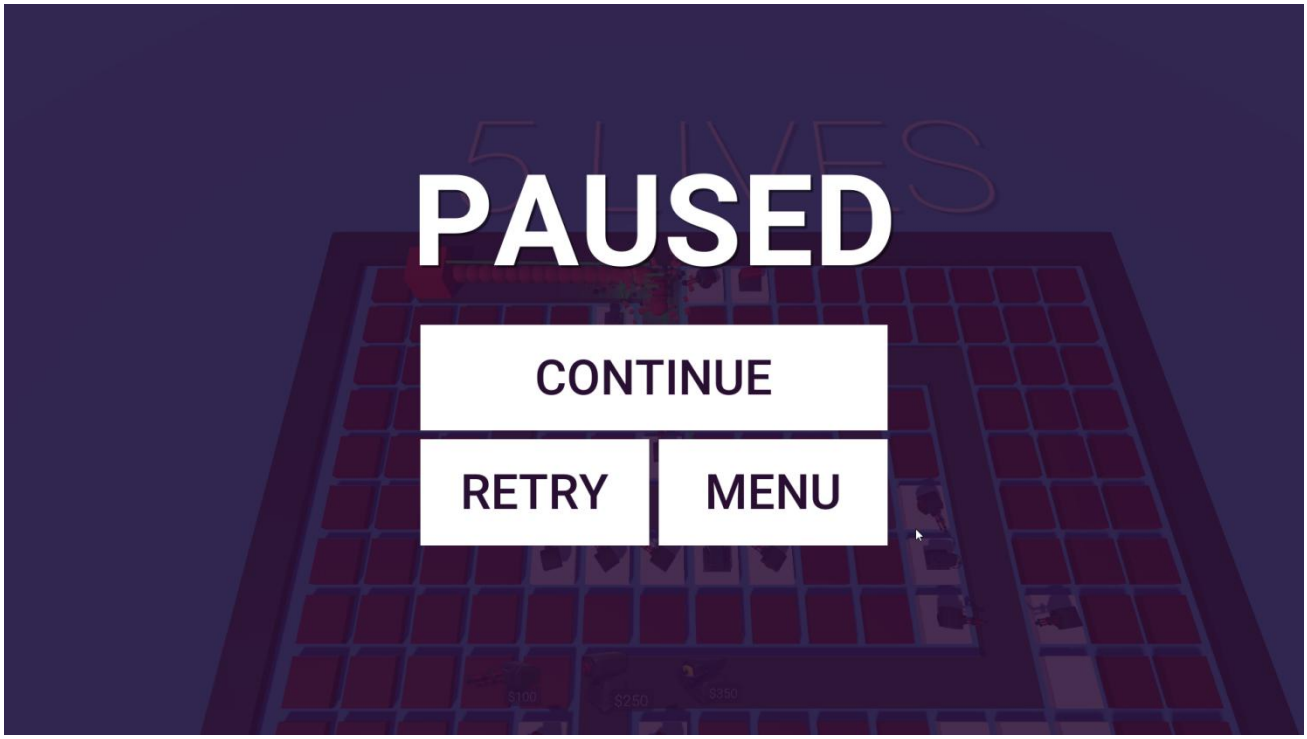


Рисунок 1.28. Меню паузи

Якщо противник досягне кінцевої точки маршруту, гравець втратить одне очко здоров'я.

На рисунку 1.29 зображена втрата здоров'я

Рисунок 1.29. Втрата Здоров'я

Якщо у гравця закінчується запас здоров'я, він зазнає поразки, після чого з'являється можливість розпочати гру знову або повернутися до головного меню.

На рис. 1.30 зображено кінець гри.

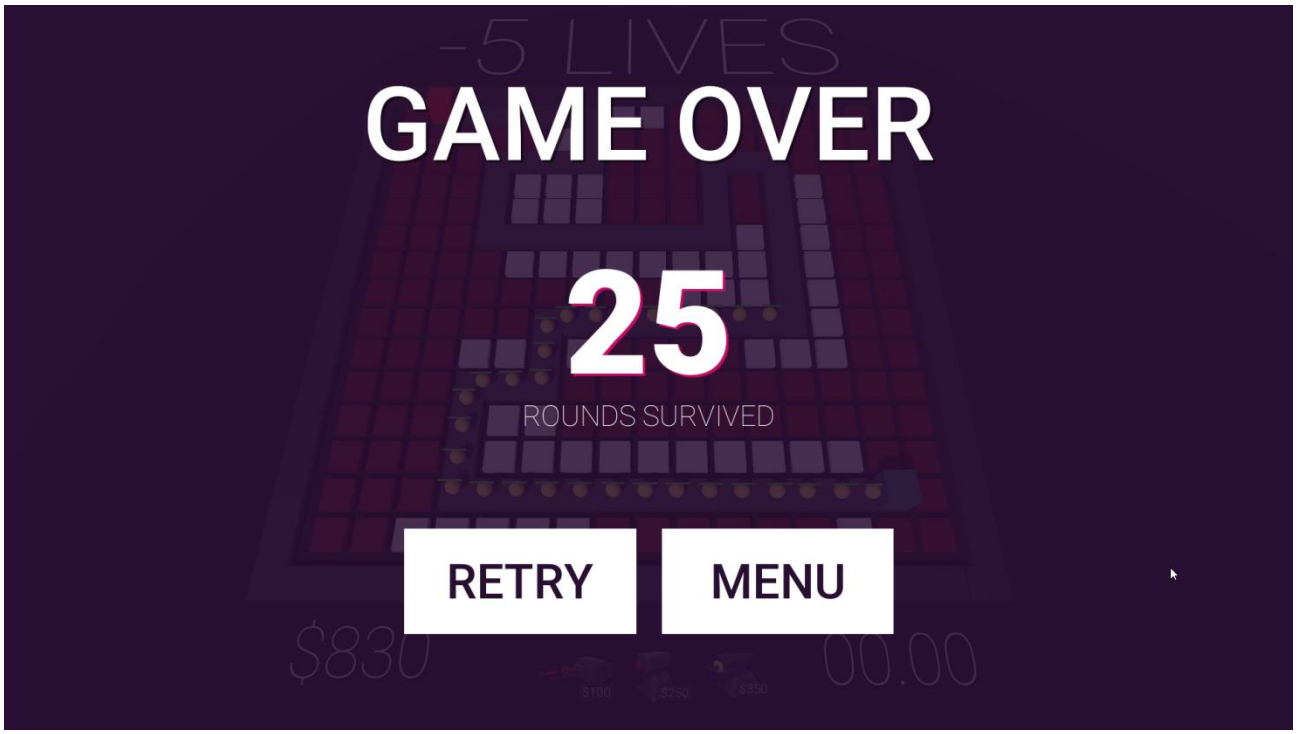


Рисунок 1.30. Кінець гри

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

У цьому дипломному проекті було розроблено та реалізовано початкову версію TD гри на програмному рушії Unity. Впроваджено основні механіки, необхідні для повноцінного ігрового процесу. Було виконано роботу над програмною частиною інтерфейсу та його взаємодією з ігровими елементами. Використання принципів об’єктно-орієнтованого програмування дозволило створити гнучку архітектуру, яка забезпечує можливість подальшої модифікації гри, додавання нових елементів, механік або режимів.

Оцінка якості програмного продукту з точки зору користувача визначається розміром оперативної пам'яті, необхідної на стадії функціонування, витратами машинного часу та пропускною спроможністю каналів передачі даних. Вона також включає визначення трудомісткості та вартості створення програмного продукту

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців та планових термінів, визначених ринковими умовами. Методом структурної аналогії за відповідними каталогами аналогів програмного забезпечення визначається обсяг програмних засобів у тисячах умовних машинних команд програм-аналогів.

Каталог аналогів

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Таблиця 2.1. Каталог аналогів програмного забезпечення

Найменування ПЗ	**Обсяг функції ПП = V_0, ум. машинних команд**
1. ПП СУБД	2500 – 9800
2. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800
3. ПП оптимізаційних розрахунків	1300 – 4200

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл. 2.2.

Таблиця 2.2. Трудомісткості на основі умовних машинних команд

Обсяг ПП, тис.умов.машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8):

Тар = 229 х 0,7 = 160.30 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{ТЗ} = T^a p \leftrightarrow L_1 \leftrightarrow K_H \qquad (2.1)$$

$$T_{ТП} = T^a p \leftrightarrow L_2 \leftrightarrow K_H \qquad (2.2)$$

$$T_{РП} = T^a p \leftrightarrow L_3 \leftrightarrow K_H \leftrightarrow K_T \qquad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.2);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.3.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.4).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	**Ступінь новизни**		
	А	**Б**	**В**
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	**Ступінь новизни**	**Значення $К_н$**
А	Принципово нові ПЗ	1,75 – 1,2
Б	ПЗ – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПЗ маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПЗ типовими програмами, %	**Значення Кт**
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання:

$$Ттз=Та*L1*Кн=160,3*0,12*0,8=15,38 \text{ (люд/годин)}$$

Трудомісткість розробки технічного проекту:

$$Ттп=Та*L2*Кн=160,3*0,11*0,8=14,11 \text{ (люд/годин)}$$

Трудомісткість розробки робочого проекту:

$$Трп=Та*L3*Кн*Кт=160,3*0,61*0,8*0,8=62,58 \text{ (люд/годин)}$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання Nтз=2 (стр), розробка ТП Nтп=16(стр), розробка робочого проекту Nрп=15 (стр), пояснювальна записка відповідно Nпз=51 (стр).

Розрахунок зведений у табл. 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	**Розрахунок, годин.**		
1	2	3	4
1.ТЗ	ТРтз=15,38	Ткк=0,7*Nтз= 0,7*2=1,4	Тнк=0,15*Nтз=0,15*2=0,30
2.Розробка ТП	ТРтп=14,11	Ткк=0,7*Nтп=0,7*16=11,2	Тнк=0,15*Nтп=0,15*16=2,4
3.Розробка РП	ТРрп=62,58	Ткк=0,7*Nрп=0,7*15=10,5	Тнк=0,15*Nрп=0,15*15=2,25
4.Розробка ПЗ	Тпз=1,5* *Nпз=1,5*51 =76,5	Ткк=0,7*Nтз=0,7*51=35,7	Тнк=0,15*Nпз=0,15*51 =7,65
Усього, в т.ч.:	225,99		
- на розробку	Тр=160,19		
- контроль керівника		Ткк= 54,2	
- нормоконтроль			Тнк=11,6

2.3 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПЗ. Розрахунок основної заробітної плати виконавців приведений у табл. 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2022» встановлено мінімальну заробітну плату у місячному розмірі з 1 квітня 2024 року - 8000 гривень; мінімальну погодинну тарифну ставку – 48 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	**Трудомісткість робіт, години**	**Погодинна тарифна ставка, грн.**	**Розрахунок, грн.**
1.Розробка ПП	160,19	48	7689,12
2.Контроль керівника	54,2	68,10	3691,02
3.Нормоконт-роль	11,6	68,10	789,96
Усього	-	-	Зо= 12170,10

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в табл. 2.8.

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	**Тип, модель**	**Кількість**	**Ціна одиниці, грн.**	**Вартість, грн.**
Папір	Лист А4	80	3,50	280
Разом	-	-	-	Вм і =280,0
Транспортно– заготівельні витрати (10%)				$Втр\_з = 0,1 \leftrightarrow Вм1 =$ 0,1х280.0=28,0
Усього				Вм=Вмі+Втр_з= 308,0

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в табл. 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	308,0	Вм (див. табл. 2.8)
2. Основна заробітна плата	12170,10	Зо (див. табл. 2.7)
3.Додаткова заробітна плата	1217,01	$Зд = 0,15 \leftrightarrow Зо$ =12170,10х0,1
4.Відрахування до єдиного фонду соціального внеску	2945,16	$Вє.с.в. = 0,22 \leftrightarrow (Зо + Зд)$ =0,22х (12170,10+ 1217,01)
5. Накладні витрати	7302,06	$Внак. = 0,6 \times Зо$ = 0,6 х 12170,10
6. Повна собівартість	23942,33	Спов= Вм +Зо + Зд +Вє.с.в. + Внак.= 308,0+12170,10+1217,01+2945,16+ 7302,06

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

П=(Сп*Р)/100= (23942,33*10)/100 = 2394,23грн (2.4)

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

Цо =Сп+П =23942,33+2394,23= 26336,56грн (2.5)

Податок на додану вартість визначаємо по наступній формулі:

ПДВ = 0.2 * Цо = 26336,56 *0,2 = 5267,31грн; (2.6)

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

Цр=Цо+ПДВ = 26336,56 + 5267,31 = 31603,87 грн (2.7)

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Вступ

Охорона праці як соціальний чинник відіграє на підприємстві важливу роль оскільки якими б важливими не були трудові здобутки вони не можуть компенсувати людині втраченого здоров'я а тим більше життя. Те і інше дається лише один раз. Необхідно пам'ятати що внаслідок нещасних випадків та аварій гинуть на виробництві не просто робітники та службовці на підготовку яких держава вкладає значні кошти а перш за все люди – годувальники сімей батьки та матері дітей. Незадовільний стан охорони праці відображається на економіці держави. Служба охорони праці створюється на підприємствах незалежно від форми власності та видів діяльності для виконання правових організаційно-технічних санітарно-гігієнічних соціально-економічних і лікувально-профілактичних заходів спрямованих на запобігання нещасних випадків професійних захворювань і аваріям в процесі праці. Основна мта всіх цих заходів – створити на підприємстві безпечні та здорові умови праці.

3.2 Аналіз небезпечних та шкідливих чинників що впливають на працівника

Оператори ПК і програмісти зіштовхуються із впливом таких фізично небезпечних і шкідливих виробничих факторів як підвищений рівень шуму підвищена температура зовнішнього середовища недостатня освітленість робочої зони електричний струм та інші. Тому на робочому місці програміста повинні бути створені умови для високопродуктивної праці. Перетворення і обробка інформації проводиться за допомогою ПК. Робота може кваліфікуватися як робота оператором ЕОМ.

3.3 Розробка заходів з охорони праці

3.3.1 Виробничі приміщення

При плануванні виробничого приміщення врахована санітарна характеристика виробничих процесів дотримуються норми корисної площі для працюючих а також нормативи площ для розташування устаткування що забезпечують безпечну роботу та зручне обслуговування устаткування. Об'ємно-планувальні рішення будівель та приміщень для роботи з ВДТ мають відповідати вимогам ДСанПіН 3.3.2.007-98. Розміщення робочих місць З ВДТ ЕОМ і ПЕОМ у підвальних приміщеннях на цокольних поверхах заборонено. Площа на одне робоче місце становить не менше ніж 60 м2 а об'єм – не менше ніж 200м3. Виробничі приміщення повинні обладнуватися шафами для зберігання документів полицями стелажами тумбами тощо з урахуванням вимог до плащі приміщення. У приміщеннях з ВДТ слід щоденно робити вологе прибирання. Приміщення повинні бути оснащені аптечками першої медичної допомоги.

3.3.2 Мікроклімат робочої зони працівників вентиляція

У виробничих приміщеннях на робочих місцях з ВДТ мають забезпечуватись оптимальні значення параметрів мікроклімату: температури відносної вологості й рухливості повітря (ДСанПіН 3.3.2.007-98).

У таблиці 3.1 наведено норми мікроклімату для приміщень з ВДТ ЕОМ та ПЕМ.

Таблиця 3.1. Норми мікроклімату для приміщень з ВДТ ЕОМ та ПЕМ

Пора року	**Категорія робіт**	**Температура повітря С не більше**	**Відносна вологість повітря %**	**Швидкість руху повітря м/с**
Холодна	Легка-1а	22-24	40-60	0.1
	Легка-1б	21-23	40-60	0.1
Тепла	Легка-1а	23-25	40-60	0.1
	Легка-1б	22-24	40-60	0.1

У таблиці 3.2 наведено рівні позитивних і негативних іонів у повітрі приміщень з ВДТ мають відповідати санітарно-гігієнічним нормам № 2152-80.

Таблиця 3.2. Рівні позитивних і негативних іонів у повітрі приміщень

Рівні	**Число іонів в 1 см3 повітря**	**Число іонів в 1 см3 повітря**
	n+	n-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимально допустимі	50000	50000

3.3.3 Освітлення робочого місця шум вібрація

Штучне освітлення в приміщеннях з робочими місцями обладнаними ВДТ має здійснюватись системою загального рівномірного освітлення. У виробничих та адміністративних приміщеннях у разі переважної роботи з документами допускається застосування системи комбінованого освітлення – крім системи загального освітлення додатково встановлюються світильники місцевого освітлення.Значення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500лк. Як джерела світла для штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛД. Допускається застосування ламп розжарювання у світильниках місцевого освітлення.

3.3.4 Організація робочого місця користувача ПК

Робочі місця слід так розташовувати відносно світових прорізів щоб природне світло падало збоку переважно зліва. При розміщенні робочих столів з ВДТ слід дотримуватися таких відстаней: між бічними поверхнями ВДТ -12м; від тильної поверхні одного ВДТ до екрану іншого – 25м.Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача що становить 600…700 мм але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.Клавіатуру розташовують на поверхні столу на відстані 100…300 мм від краю зверненого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5…150.При оснащенні робочого місця лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам СанПіН № 5804-91.ЕОМ ВДТ і ПК інше устаткування електропроводи та кабелі слід надійно заземляти та зануляти відповідно до вимог ПУЭ.

3.3.5 Санітарно-гігієнічні вимоги

Рівні шуму в приміщеннях де встановлено ВДТ не повинні перевищувати значень 50-60 дБ у діапазоні частот 31,5 – 8000 Гц відповідно до ГОСТ 12.1.003-83 ССБТ «Шум. Общие требования безопасности». Під час роботи з ВДТ рівні звукового тиску в октавних смугах із середньогеометричними частотами повинні відповідати вимогам ДСанПіН 3.3.2.007-98.Для зниження рівнів шуму слід проводити профілактичне обслуговування пристроїв вентиляції та іншого устаткування що є джерелом шуму встановлення шумозахисних екранів застосування звукопоглинальних матеріалів тощо.Основними джерелами електромагнітного випромінювання є монітори принтери джерела живлення що застосовуються в комп’ютерній техніці. Рівні електромагнітного випромінювання від ВДТ мають відповідати вимогам СанПіН 3.3.2.007-98.

3.3.6 Ергономіка

Для зменшення монотонності і забезпечення зручності в роботі з ПК передбачено робоче місце з оптимальними розмірами і розташуванням

елементів з урахуванням антропометричних характеристик користувача. У робочій позі користувача має бути забезпечена нейтральна позиція хребта передпліччя повинні знаходитися в паралельному до підлоги положенні або з невеликим нахилом до неї (10 градусів), стегна – в горизонтальному положенні гомілки – під прямим кутом до поверхні сидіння з опорою ступнями на підлогу або на підставку для ніг. Крісла для роботи з ПК мають бути підйомно-поворотні що дозволяє змінювати висоту і нахил сидіння підлокітники і висоту спинки.Для попередження втомлюваності очей важливими є такі чинники: зниження яскравості світлових відблисків на поверхні екрана збереження чіткості знаків на екрані достатня і рівномірна освітленість робочої поверхні правильна організація режимів зорової роботи.

3.4 Пожежна безпека

Приміщення з ПК повинні належати до категорії В пожежонебезпеки з допустимим навантаженням горючих матеріалів до 180 МДж/м2. У приміщенні слід розміщувати тільки речі та матеріали які необхідні для виконання робіт.Будівлі де розташовано приміщення з ПК обладнані засобами протипожежного захисту відповідно до вимог чинних нормативних документів і стандартів. Приміщення обладнуються вуглекислотними та порошковими вогнегасниками відповідно до норм передбачених ГОСТ 12.4.009-83 «Пожарная техника для защиты объектов. Основные виды. Размещение и обслуживание». Відстань від можливого осередку пожежі до вогнегасника має бути не більше 20 м. Загальна кількість вогнегасників визначається виходячи з площі приміщення але має бути не менше двох. У виробничих приміщеннях має бути передбачено не менше двох евакуаційних виходів що розміщуються на максимальній відстані один від одного.

3.4.1 Захист від статичної електрики

Захист від статичної електрики в офісі досягається шляхом проведення організаційних і технічних заходів. Системи заземлення і занулення мають бути

справними а їх опір має відповідати діючим нормам. Всі металеві частини електроустановок і корпуси ПК повинні бути заземлені.

3.4.2 Захист від іонізуючих випромінювань

Рівень іонізуючих випромінювань від ЕОМ не повинен перевищувати гранично допустимі норми які встановлені чинними нормативними документами. Приміщення в яких розташовані робочі місця з ВДТ мають бути забезпечені індивідуальними захисними екранами.

3.4.3 Захист від лазерного випромінювання

Лазерна техніка застосовується в офісі як джерело випромінювання у принтерах копіювальних пристроях та іншому устаткуванні. Основні вимоги безпеки стосовно лазерної техніки полягають у тому що устаткування має бути справним а робоче місце повинно бути організоване так щоб лазерний промінь не потрапляв у поле зору користувача.

ВИСНОВКИ

Протягом виконання дипломного проекту на тему "Розробка ігрового застосунку у жанрі стратегія" було досягнуто наступні результати:

Проведено детальний аналіз існуючих рішень у жанрі стратегічних ігор, що дозволило визначити найкращі практики та методики для створення якісного ігрового продукту. Вибір технології Unity був обґрунтований її можливостями забезпечення високої продуктивності та кроссплатформенності, що є ключовими для сучасних ігор

Розроблено архітектуру системи, яка включає створення ігрових механік, дизайн інтерфейсу користувача та структури даних. Основна увага приділялася створенню зручного та інтуїтивно зрозумілого інтерфейсу, що сприяє кращій взаємодії гравця з грою.

Створено ігровий застосунок на платформі Unity з використанням мови програмування C#. Unity надає інструменти для розробки, тестування та оптимізації ігрових механік, що дозволило досягти високої якості продукту.

Проведено ретельне тестування гри, включаючи виявлення та класифікацію помилок за критичністю та впливом на гру. Після виправлення помилок здійснювалося повторне тестування, що дозволило забезпечити стабільну роботу гри та високу якість користувацького досвіду.

В результаті виконаної роботи було створено повноцінний ігровий застосунок у жанрі стратегія, що відповідає сучасним вимогам до програмного забезпечення. Досягнуті результати підтверджують можливість подальшого розвитку та вдосконалення гри з метою її комерційного запуску.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Plants vs. Zombies. [Веб-сайт]. URL: https://play.google.com/store/apps/details?id=com.ea.game.pvzfree_row&hl=en/.
2. Kingdom Rush Tower Defense TD. [Веб-сайт]. URL: https://play.google.com/store/apps/details?id=com.ironhidegames.android.kingdomrush&hl=en/.
3. Bloons TD 6. [Веб-сайт]. URL: https://play.google.com/store/apps/details?id=com.YoStarEN.Arknights&hl=en/.
4. Arknights. [Веб-сайт]. URL: https://play.google.com/store/apps/details?id=com.YoStarEN.Arknights&hl=en/.
5. Unity 3D Docs. [Веб-сайт]. URL: https://docs.unity3d.com/Manual/index.html/.
6. Unreal Engine Docs. [Веб-сайт]. URL: https://docs.unrealengine.com/4.27/en-US/.
7. GODOT Docs. [Веб-сайт]. URL: https://docs.godotengine.org/en/stable/index.html/.
8. C# Docs. [Веб-сайт]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/.
9. М. В. Добролюбова. Процедурна складова мови C# (навч. посібник). – «КПІ Ігоря Сікорського», 2021.
10. Huw Collingbourne. The Little Book Of C# Programming. – «Paperback», 2019 (англ. мовою).

ДОДАТОК А. Програмний код основної логіки веб-застосунку

```
// BuildManager.cs

using UnityEngine;

public class BuildManager : MonoBehaviour {

    public static BuildManager instance;

    void Awake ()
    {
        if (instance != null)
        {
            Debug.LogError("More than one BuildManager in scene!");
            return;
        }
        instance = this;
    }

    public GameObject buildEffect;
    public GameObject sellEffect;

    private TurretBlueprint turretToBuild;
    private Node selectedNode;

    public NodeUI nodeUI;

    public bool CanBuild { get { return turretToBuild != null; } }
    public bool HasMoney { get { return PlayerStats.Money >= turretToBuild.cost; } }

    public void SelectNode (Node node)
    {
        if (selectedNode == node)
        {
            DeselectNode();
            return;
        }

        selectedNode = node;
        turretToBuild = null;

        nodeUI.SetTarget(node);
    }

    public void DeselectNode()
    {
        selectedNode = null;
        nodeUI.Hide();
    }

    public void SelectTurretToBuild (TurretBlueprint turret)
    {
        turretToBuild = turret;
        DeselectNode();
    }
```

```
    public TurretBlueprint GetTurretToBuild ()
    {
        return turretToBuild;
    }

}

// Bullets.cs

using UnityEngine;

public class Bullet : MonoBehaviour {

    private Transform target;

    public float speed = 70f;

    public int damage = 50;

    public float explosionRadius = 0f;
    public GameObject impactEffect;

    public void Seek (Transform _target)
    {
        target = _target;
    }

    // Оновлення викликається один раз на кадр
    void Update () {

        if (target == null)
        {
            Destroy(gameObject);
            return;
        }

        Vector3 dir = target.position - transform.position;
        float distanceThisFrame = speed * Time.deltaTime;

        if (dir.magnitude <= distanceThisFrame)
        {
            HitTarget();
            return;
        }

        transform.Translate(dir.normalized * distanceThisFrame, Space.World);
        transform.LookAt(target);

    }

    void HitTarget ()
    {
        GameObject effectIns = (GameObject)Instantiate(impactEffect, transform.position,
 transform.rotation);
        Destroy(effectIns, 5f);

        if (explosionRadius > 0f)
        {
            Explode();
        } else
        {
            Damage(target);
```

```
        }

        Destroy(gameObject);
    }

    void Explode ()
    {
        Collider[] colliders = Physics.OverlapSphere(transform.position, explosionRadius)
;
        foreach (Collider collider in colliders)
        {
            if (collider.tag == "Enemy")
            {
                Damage(collider.transform);
            }
        }
    }

    void Damage (Transform enemy)
    {
        Enemy e = enemy.GetComponent<Enemy>();

        if (e != null)
        {
            e.TakeDamage(damage);
        }
    }

    void OnDrawGizmosSelected ()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, explosionRadius);
    }
}

// CameraController.cs

using UnityEngine;

public class CameraController : MonoBehaviour {

    public float panSpeed = 30f;
    public float panBorderThickness = 10f;

    public float scrollSpeed = 5f;
    public float minY = 10f;
    public float maxY = 80f;

    // Оновлення викликається один раз на кадр
    void Update () {

        if (GameManager.GameIsOver)
        {
            this.enabled = false;
            return;
        }

        if (Input.GetKey("w") || Input.mousePosition.y >= Screen.height -
 panBorderThickness)
        {
            transform.Translate(Vector3.forward * panSpeed * Time.deltaTime, Space.World)
;
```

```
        }
        if (Input.GetKey("s") || Input.mousePosition.y <= panBorderThickness)
        {
            transform.Translate(Vector3.back * panSpeed * Time.deltaTime, Space.World);
        }
        if (Input.GetKey("d") || Input.mousePosition.x >= Screen.width -
 panBorderThickness)
        {
            transform.Translate(Vector3.right * panSpeed * Time.deltaTime, Space.World);
        }
        if (Input.GetKey("a") || Input.mousePosition.x <= panBorderThickness)
        {
            transform.Translate(Vector3.left * panSpeed * Time.deltaTime, Space.World);
        }

        float scroll = Input.GetAxis("Mouse ScrollWheel");

        Vector3 pos = transform.position;

        pos.y -= scroll * 1000 * scrollSpeed * Time.deltaTime;
        pos.y = Mathf.Clamp(pos.y, minY, maxY);

        transform.position = pos;

    }
}

// CompleteLevel.cs

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

public class CompleteLevel : MonoBehaviour {

    public string menuSceneName = "MainMenu";

    public string nextLevel = "Level02";
    public int levelToUnlock = 2;

    public SceneFader sceneFader;

    public void Continue ()
    {
        PlayerPrefs.SetInt("levelReached", levelToUnlock);
        sceneFader.FadeTo(nextLevel);
    }

    public void Menu ()
    {
        sceneFader.FadeTo(menuSceneName);
    }

}
```

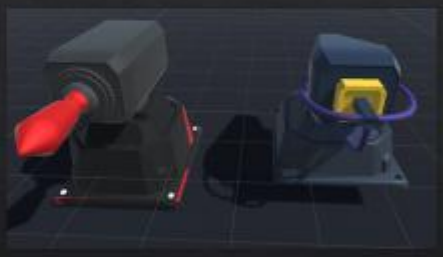
ДОДАТОК Б. Слайди мультимедійної презентації

Розробка гри у жанрі TD

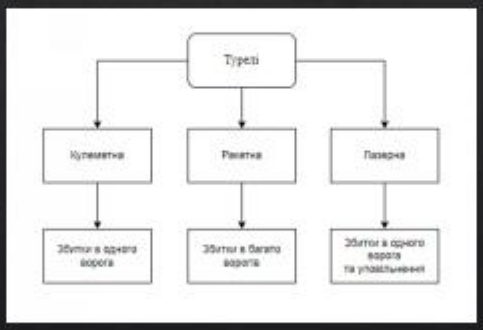
Мельник Микита, гр.
4РП-07, ОТФК ОНТУ

Використовані технології

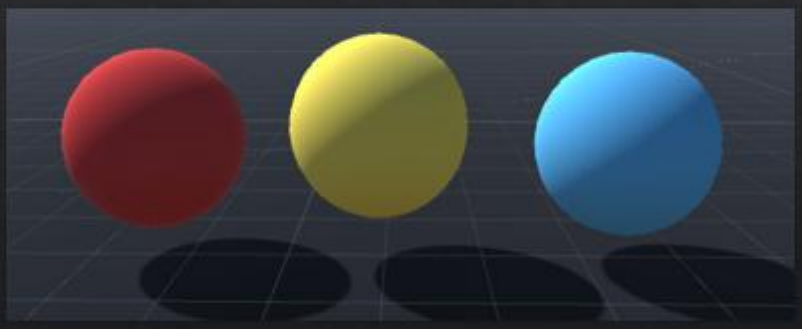
Зовнішній вигляд веж



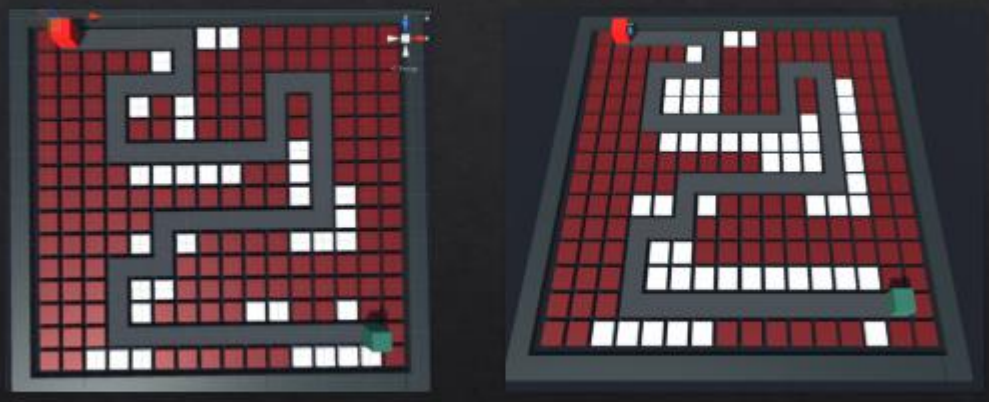
Види веж



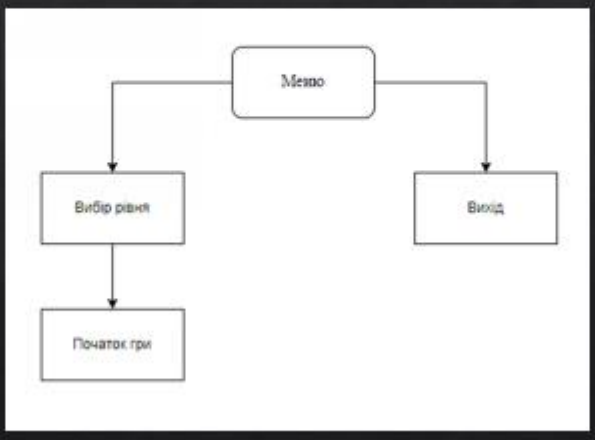
Зовнішній вигляд ворогів



Вигляд мапи



Навігація по графичному інтерфейсу



Частина коду

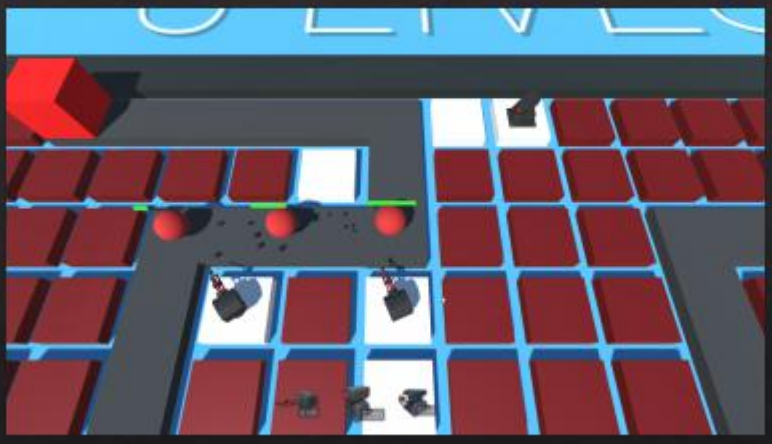
«CameraController»

```
using UnityEngine;

public class CameraController : MonoBehaviour {

    public float panSpeed = 30f;
    public float panBorderThickness = 10f;

    public float scrollSpeed = 5f;
    public float minY = 10f;
    public float maxY = 80f;

    // Оновлення викликається один раз на кадр
    void Update () {

        if (GameManager.GameIsOver)
        {
            this.enabled = false;
            return;
        }

        if (Input.GetKey("w") || Input.mousePosition.y >= S
creen.height - panBorderThickness)
        {
            transform.Translate(Vector3.forward * panSpeed
* Time.deltaTime, Space.World);
        }

        if (Input.GetKey("s") || Input.mousePosition.y <= panBorder
Thickness)
        {
            transform.Translate(Vector3.back * panSpeed * T
ime.deltaTime, Space.World);
        }
        if (Input.GetKey("d") || Input.mousePosition.x >= S
creen.width - panBorderThickness)
        {
            transform.Translate(Vector3.right * panSpeed *
Time.deltaTime, Space.World);
        }
        if (Input.GetKey("a") || Input.mousePosition.x <= p
anBorderThickness)
        {
            transform.Translate(Vector3.left * panSpeed * T
ime.deltaTime, Space.World);
        }

        float scroll = Input.GetAxis("Mouse ScrollWheel");

        Vector3 pos = transform.position;

        pos.y -
= scroll * 1000 * scrollSpeed * Time.deltaTime;
        pos.y = Mathf.Clamp(pos.y, minY, maxY);

        transform.position = pos;

    }
}
```

Початок гри

Геймплей

Геймплей



Кінець гри

Висновки

У результаті виконаної роботи створено повноцінний ігровий застосунок у жанрі стратегія, що відповідає сучасним вимогам до програмного забезпечення. Досягнуті результати підтверджують можливість подальшого розвитку та вдосконалення гри для її комерційного запуску.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Мельника Микити Олександровича

(прізвище, ім`я та по батькові)

Спеціальність: *121 «Інженерія програмного забезпечення» ;*

Освітня програма: *«Розробка програмного забезпечення»*

Тема дипломного проекту: *Розробка ігрового застосунку у жанрі стратегія*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 74 сторінки. У пояснювальній записці описано етапи розробки ігрового застосунку у жанрі стратегія засобами рушію Unity. Графічна частина складається з окремих слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра.*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувач освіти Мельник Микита поступово та послідовно виконував всі етапи, проявляв ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Мельник Микита під час роботи над дипломним проектом вивчив достатньо багато літературних та інтернет-джерел за даною тематикою.*

Вважаю, що теоретична підготовка дипломника достатня і він готовий до захисту проекту.

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувач освіти Мельник Микита показав вміння організовано працювати над поставленим завданням, застосовувати знання у галузі програмування та математики, розробляти, встановлювати та налаштовувати спеціалізоване програмне забезпечення, оформлювати слайди та складати презентації, користуючись сучасними комп'ютерними програмними засобами, такими як MS VS, Unity 3D, MS PowerPoint, MS Visio та ін.*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проекту *Іванова Лілія Вікторівна*

Місце роботи і посада керівника дипломного проекту *ВСП «Одеський технічний фаховий коледж ОНТУ», викладач спецдисциплін циклової комісії комп'ютерної техніки та програмної інженерії*

Підпис

«10» червня 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Мельника Микити Олександровича

(прізвище, ім`я та по батькові)

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Іванова Лілія Вікторівна*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка ігрового застосунку у жанрі стратегія*

Обсяг розрахунково-пояснювальної записки *73* сторінок

Обсяг графічної (презентаційної) частини *11* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений проблемі створенню ігор у жанрі стратегія та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 11 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скриншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Реалізовано ігровий застосунок у жанрі стратегія, що містить рівні, головного героя, ворогів та зброю.*

У якості ігрового рушію обрано сучасну технологію – Unity 3D.

Ігровий застосунок використовує 3D режим в геймплеї.

д) основні недоліки дипломного проекту

Графічний інтерфейс ігрового застосунку надає лише базові можливості, такі як запуску рівня, але не надає інших можливостей, наприклад, зміни мови.

В основному розділі пояснювальної записки поверхнево описано етапи розробки. В деяких частинах пояснювальної записки присутні помилки оформлення.

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Добре*

Загальна оцінка *Добре*

Прізвище, ім'я, по батькові рецензента *к.т.н. Кірєєв Ігор Анатолійович*

Місце роботи і посада рецензента *Державний університет інтелектуальних технологій і зв'язку, доцент каф. інформаційної безпеки та передачі даних*

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ
КАДРІВ ДУІТЗ

ВІДДІЛ КАДРІВ

Підпис:

«14» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016357551

Дата перевірки:
13.06.2024 17:00:00 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.06.2024 17:06:37 EEST

ID користувача:
100011688

Назва документа: 4РП-07_Мельник_М

Кількість сторінок: 45 Кількість слів: 5417 Кількість символів: 42070 Розмір файлу: 3.64 MB ID файлу: 1016161871

22%
Схожість

Найбільша схожість: 5.46% з Інтернет-джерелом (https://repozitorij.foi.unizg.hr/islandora/object/foi%3A5951/datastrea..

22% Джерела з Інтернету | 515 ..... Сторінка 47

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 11

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Мельник Микита Олександрович,
здобувач освіти гр. 4РП-07, та

Іванова Лілія Вікторівна,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка ігрового застосунку у жанрі стратегія» (автор роботи – Мельник М.О., керівник роботи – Іванова Л.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Мельник М.О. /

Керівник _____ / Іванова Л.В. /

«10» червня 2024 р.