

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ДВОМІРНОЇ КОМП'ЮТЕРНОЇ ГРИ
SOFTWARE IMPLEMENTATION OF A 2D COMPUTER GAME

Виконав(ла): студент(ка) 4 курсу, групи КНТ-151

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

АНДРЕЄВ Р.А.

(ПРІЗВИЩЕ та ініціали)

Керівник КОЛПАКОВА Т.О.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

АНДРЕЄВА Руслана Андрійовича
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація двовірної комп'ютерної гри.
Software Implementation of a 2D Computer Game

керівник проєкту (роботи) к.т.н., доцент, КОЛПАКОВА Тетяна Олексіївна,
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року
 3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення. 3. Конструювання програмного забезпечення. 4. Керівництво користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КОЛПАКОВА Т.О., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Руслан АНДРЕЄВ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна КОЛПАКОВА
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 74 с., 4 табл., 30 рис., 3 дод., 25 джерел.

2D ГРА, КОМП'ЮТЕРНА ПРОГРАМА, ІГРОВИЙ ЗАСТОСУНОК, IT-ТЕХНОЛОГІЇ, GAME ENGINE, C#.

Об'єкт дослідження – процес розробки двомірної комп'ютерної гри.

Предмет дослідження – засоби розробки ігрової програми для комп'ютера.

Мета роботи – програмна реалізація двомірної комп'ютерної гри з використанням сучасних інструментів програмування.

Матеріали, методи та технічні засоби: середовище розробки Microsoft Visual Studio, мова програмування C#, ігровий рушій Unity, персональний комп'ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 10.

Галузь використання – проєкт можна використати як базу для подальшого розвитку більш складних ігор.

Результати. Розроблено двомірну комп'ютерну гру що реалізує базові елементи ігрового процесу: управління персонажем, обробку подій, відображення графіки.

Висновки. Розроблено функціональний прототип двомірної комп'ютерної гри на мові програмування C# та на ігровому рушії Unity.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 74 pages, 4 tables, 30 figures, 3 appendixes, 25 sources.

2D GAME, COMPUTER PROGRAM, GAME APPLICATION, IT TECHNOLOGY, GAME ENGINE, C#.

The object of research is the process of developing a two-dimensional computer game.

The subject of research is the means of developing a game program for a computer.

The purpose of the work is the software implementation of a two-dimensional computer game using modern programming tools.

Materials, methods and technical means: Microsoft Visual Studio development environment, C# programming language, Unity game engine, personal computer with an Intel Core i5 processor running the Microsoft Windows 10 operating system.

Scope of use - the project can be used as a basis for further development of more complex games.

Results. A two-dimensional computer game has been developed that implements the basic elements of the game process: character control, event processing, graphics display.

Conclusions. A functional prototype of a two-dimensional computer game has been developed in the C# programming language and the Unity game engine.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.2 Історія жанру Roguelike.....	10
1.3 Аналіз існуючих ігор у жанрі Roguelike.....	11
1.3.1 Brotato.....	11
1.3.2 Vampire Survivors.....	12
1.3.3 Enter the Gungeon.....	13
1.3.4 Порівняння існуючих аналогів ігор у жанрі Roguelike.....	14
1.3.5 Висновки за розділом 1.....	15
2 Проєктування програмного забезпечення.....	16
2.1 Вимоги до системи.....	16
2.2 Архітектура розроблюваної системи.....	16
2.3 Вибір рушія для розробки.....	18
2.3.2 GameMaker Studio.....	20
2.3.3 Godot Engine.....	23
2.4 Особливості Unity як основної платформи розробки.....	27
2.5 Вибір IDE.....	30
2.5.1 Microsoft Visual Studio.....	30
2.5.2 JetBrains Rider.....	31
2.6 Висновки за розділом 2.....	33
3 Конструювання програмного забезпечення.....	34
3.1 Система дій, реакцій і відгуків у грі.....	34
3.1.1 Дія.....	34
3.1.2 Реакція.....	35
3.1.3 Відгук.....	35
3.2 Компоненти проєкту.....	36
3.3 Опис алгоритму створення хвиль ворогів.....	40

3.4 Висновки за розділом 3	41
4 Керівництво користувача	42
4.1 Системні вимоги.....	42
4.2 Функціональні можливості застосунку	42
4.3 Інструкція з експлуатації.....	42
4.4 Принцип роботи	43
4.5 Висновки за розділом 4	46
Висновки	47
Перелік джерел посилання	49
Додаток А Технічне завдання	51
Додаток Б Текст програми	55
Додаток В Слайди презентації.....	68

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CPU	– Central Processing Unit;
HDD	– Hard Disk Drive;
IDE	– Integrated Development Environment;
MVC	– Model-View-Controller;
SDK	– Software Development Kit;
UML	– Unified Modeling Language;
БД	– база даних;
ЛКМ	– ліва клавіша миші;
ОС	– операційна система;
ПК	– персональний комп'ютер.

ВСТУП

На нинішньому етапі розвитку інформаційних технологій комп'ютерні ігри стали не лише засобом розваги, але і потужним інструментом навчання, розвитку логічного мислення, візуалізації складних процесів і моделювання взаємодії у віртуальному середовищі. Особливе місце серед них посідають двовимірні ігри, які, незважаючи на простішу графіку, залишаються актуальними завдяки доступності, легкості у реалізації та можливості швидкого прототипування ігрових ідей.

Розробка двовимірних ігор дозволяє програмісту ознайомитися з основами гейм-дизайну, алгоритмами управління персонажем, побудовою графічного інтерфейсу та інтерактивною взаємодією. Такі проєкти мають велике значення як навчальний інструмент в освітньому процесі, та як стартова платформа для входження у сферу розробки комп'ютерних ігор.

Метою даної дипломної роботи є створення повноцінного програмного прототипу двовимірної гри з використанням ігрового рушія Unity та сучасних засобів програмування.

Актуальність теми зумовлена широким використанням ігор у різних сферах – від освіти до комерційних продуктів – а також потребою у фахівцях, здатних проєктувати та реалізовувати ігрові додатки на сучасному рівні.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення жанру Roguelike

Roguelike – жанр комп'ютерних ігор, характерними особливостями якого є випадково згенеровані рівні, покроковість і незворотність у смерті персонажа – у разі його загибелі гравець не може завантажити гру, і має почати її проходження спочатку.

1.2 Історія жанру Roguelike

Жанр «roguelike» сходить до гри 1980 року Rogue, темою якою було дослідження підземель [1]. Хоча до неї виходили подібні ігри, саме Rogue стала зразком для незліченних наслідувань. Подібні ігри, що поширювалися у вигляді відкритого коду, були вкрай популярні в середовищі американських програмістів у 1980-х – 1990-х роках. За наявності безлічі варіантів і відмінностей між окремими іграми основні принципи геймплею roguelike залишалися незмінними.

Термін «roguelike» почали використовувати близько 1993 року в Uneset, щоб об'єднати обговорення «однокористувацьких фентезійних рольових комп'ютерних ігор, дія яких зазвичай відбувається в підземеллі, з простим графічним інтерфейсом». Через брак чіткішого визначення, як основні приклади того, що цей термін мав представляти, використовувалися ігри Rogue [1], Nethack [2]. Оскільки Rogue була попередницею всіх інших ігор цього набору, «roguelike» спочатку сприймався як «найменше з усіх можливих зол».

З часом стало очевидно, що ігри об'єднувало щось більше, ніж їхня тематика чи графічне представлення. Набір спільних ігрових елементів зацікавив розробників-любителів, які вивели жанр на новий рівень. Взявши з цих оригінальних ігор те, що їм сподобалося, вони створили похідні, допрацьовуючи механіку та впроваджуючи нові сюжети.

У певний момент інші розробники ігор побачили потенціал, який мав деякі особливості цього жанру, і також зрозуміли, що деякі елементи традиційного формату не подобаються більш широкій аудиторії. Вони взяли особливості цього жанру та адаптували їх до іншого типу ігор: 3D-шутерів від першої особи, екшн-рольових ігор і т.д.

Через ці зміни зросла плутанина щодо значення терміну «roguelike» і згодом з'явилася низка нових термінів, які пов'язані з жанром «roguelike», а саме: roguelite, roguelike-like та action roguelike.

На сьогодні жанр ролевих ігор живий як ніколи, ролеві ігри в традиційному стилі продовжують розвиватися, і в той же час аспекти формули ролевих ігор продовжують проникати в інші жанри, створюючи захопливі та веселі ігри [3].

1.3 Аналіз існуючих ігор у жанрі Roguelike

Аналіз було проведено за наступними критеріями: візуальна частина, керування персонажем, інтерфейс, музичний супровід.

1.3.1 Brotato

Brotato – арений шутер з елементами roguelike (рис. 1.1.), де користувач грає за озброєну картоплину, що дає відсіч ордам прибульців. Візуальна частина та візуальні ефекти приємні оку. Управління персонажем відбувається завдяки клавішам з клавіатури WASD, а стрільба за допомогою лівої клавіші миші (ЛКМ). Музичний супровід багатий на різні композиції. Користувацький інтерфейс комфортний та зручний [4].



Рисунок 1.1 – Brotato [4]

1.3.2 Vampire Survivors

Vampire Survivors – казуальна гра у стилі готичного хорору з елементами спрощеного roguelike (рис. 1.2). Гравець керує одним із героїв якими треба атакувати ворогів, і намагається вижити якомога довше в умовах постійно зростаючої хвилі монстрів. Також у цій грі є багато прихованих деталей і секретів, які відкриваються з часом. Вона кросплатформна, підтримує як клавіатуру так і сенсорні екрани та геймпади. Управління на комп'ютері здійснюється за допомогою клавіш клавіатури WASD та за допомогою ЛКМ. Хоч графіка і виглядає дуже спрощено та застаріло, але це не заважає їй бути приємною оку. Інтерфейс дуже зрозумілий і зручний. Гра має велику кількість музикальних композицій [5].



Рисунок 1.2 – Vampire Survivors [5]

1.3.3 Enter the Gungeon

Enter the Gungeon – гра з елементами roguelike (рис. 1.3), цілю якої є вийти за одне життя з підземелля, і в якій гравець грає за одного з запропонованих героїв. У кожного героя є характеристики, а також пасивні і активні здібності, завдяки яким гравець може підлаштовуватись під власний стиль ігри. Управління здійснюється завдяки клавішам WASD, ЛКМ та пробілу, натиснувши який гравець активує здібність персонажа. Графіка, анімації, візуальні ефекти у грі дуже гарні та добре пропрацьовані. Музикальний супровід дуже великий та різноманітний [6].



Рисунок 1.3 – Enter the Gungeon [6]

1.3.4 Порівняння існуючих аналогів ігор у жанрі Roguelike

В п.1.3.3 було проаналізовано вже випущені у світ аналоги ігор у жанрі Roguelike. Згідно з критеріями, які були визначені на початку попереднього пункту, у таблиці 1.1 показано порівняння з оцінками кожної характеристики по шкалі від 1 до 5.

Таблиця 1.1 – Порівняльна таблиця

Назва гри	Візуальна складова	Керування	Інтерфейс	Музикальний супровід
Brotato	4	5	4	5
Vampire Survivors	2	5	5	5
Enter the Gungeon	5	5	5	5

1.3.5 Висновки за розділом 1

В розділі було проведено аналіз предметної області. Було описано історію становлення жанру ігор Roguelike, та характеристики, що дозволяють віднести гру до цього жанру. Також було виконано аналіз деяких вже випущених у світ аналогів Roguelike ігор та їх порівняння за такими критеріями, як візуальна частина, керування персонажем, інтерфейс користувача та музикальний супровід.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В цьому розділі описано проєктування програмного забезпечення.

Розглянуто наступні питання: вимоги до системи; архітектура розроблюваної системи; вибір та аналіз ігрового рушія, на якому проводиться розробка ігрового додатку; вибір та аналіз інтегрованого середовища розробки (IDE) для програмування.

2.1 Вимоги до системи

Розроблювана гра у жанрі Roguelike має бути зручною та зрозумілою під час використання, геймплей повинен бути цікавим, і також вона повинна бути оптимізованою.

Гра повинна мати такий функціонал:

- пересування гравця по локації за осями X та Y;
- логіка програшу;
- генерація безкінечних хвиль супротивників кількість яких зростає в геометричній прогресії;
- меню навігації.

2.2 Архітектура розроблюваної системи

Для того, щоб реалізувати цей проєкт, було прийнято рішення використати підхід Model-View-Controller (MVC), завдяки якому можна розділити логіку програми (model), графічне представлення (view) та обробку подій (controller).

Модель відповідає за зберігання даних та логіку їх обробки, включаючи запити до бази даних (БД) і перевірку їхньої правильності. Вона не залежить ні від представлення (не займається відображенням даних), ні від контролера (не взаємодіє напряду з користувачем). Основна її роль –

забезпечити доступ до даних і керування ними. Конструкція моделі передбачає реакцію на зовнішні запити з можливістю змінювати свій стан. За потреби в модель можуть бути інтегровані механізми сповіщення інших компонентів (наприклад, патерн «спостерігач»). Завдяки незалежності від візуального шару, одну й ту саму модель можна відображати через різні інтерфейси без зміни її внутрішньої структури.

Представлення відповідає за виведення інформації користувачу, отримуючи дані з моделі. Воно лише відображає ці дані, не обробляючи безпосередньо дії користувача.

Контролер виступає посередником між користувачем і програмною системою. Він сприймає введення користувача, інтерпретує його, та ініціює відповідні зміни в моделі або оновлення подання. Саме контролер координує взаємодію між іншими компонентами для забезпечення коректної роботи програми [7].

На рисунку 2.1 наведено схему, на якій показано, як працює MVC.

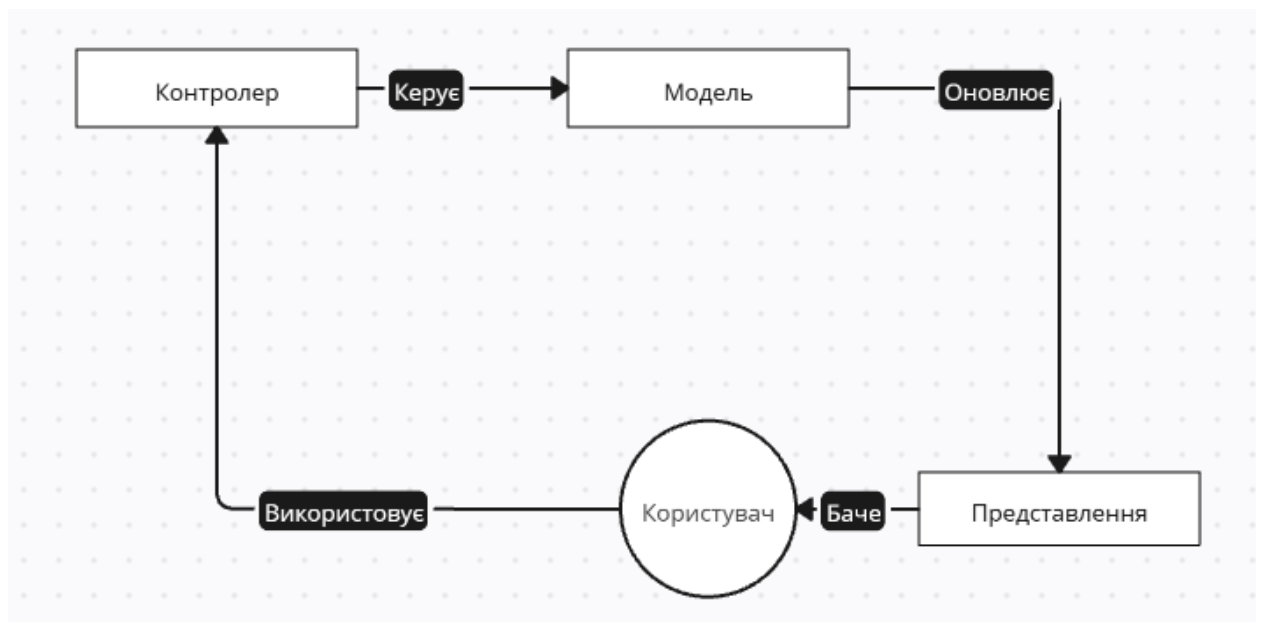


Рисунок 2.1 – Схема роботи MVC

На рисунку 2.2 наведена діаграма класів, яка була розроблена завдяки безкоштовному вебдодатку Creately [8] з використанням мови UML [9]. На

ній видно, який клас за яким закріплено, і як вони взаємодіють. Кожен клас у розроблюваній системі виконує одну з ролей: модель, контролер або представлення. Такий поділ забезпечує реалізацію архітектурного шаблону MVC. Головним класом є Player.

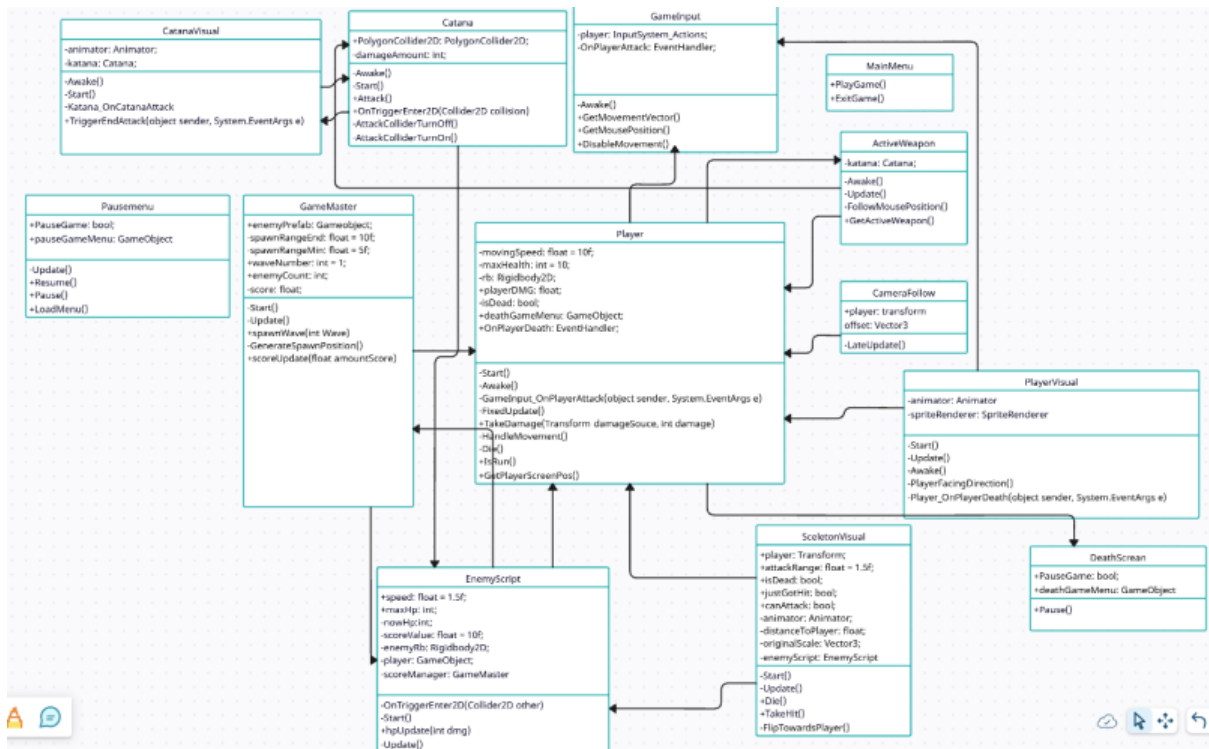


Рисунок 2.2 – UML діаграма класів

2.3 Вибір рушія для розробки

У цьому розділі здійснено аналіз існуючих ігрових рушіїв з метою визначення найліпшого варіанту для розробки 2D гри. Серед найбільш популярних ігрових рушіїв для розроблення додатку для порівняльного аналізу було виокремлено найбільш популярні: Unity, GameMaker, Godot Engine.

Ігровий рушій (англ. Game engine) – це програмний комплекс, центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її внутрішньої структури [10].

2.3.1 Unity

Unity – кросплатформне середовище розробки комп’ютерних ігор, яке було створене американською компанією Unity Technologies. Unity дає змогу створювати додатки, які функціонують на понад 25 різних платформах, зокрема персональних комп’ютерах (ПК), ігрових консолях, мобільних пристроях та інших. Реліз Unity відбувся у 2005 року, і відтоді відбувається неперервний розвиток.

Редактор Unity вирізняється інтуїтивно зрозумілим інтерфейсом із підтримкою технології Drag&Drop. Його інтерфейс складається з декількох функціональних вікон, що забезпечує зручність у процесі розробки та дозволяє здійснювати налагодження гри безпосередньо в редакторі [11]. Для написання скриптів використовується мова програмування C#. Фізичні обчислення в Unity забезпечуються за допомогою двох вбудованих рушіїв: PhysX від NVIDIA – для 3D фізики, та Box2D – для 2D-симуляцій.

Проекти в Unity поділяються на сцени (рівні), кожна з яких є окремим файлом із власним набором об’єктів, компонентів, сценаріїв та налаштувань. Сцени можуть містити як візуальні об’єкти, так і порожні невидимі об’єкти (пустушки), які використовуються як контейнери або логічні одиниці. Усі об’єкти сцени формуються з компонентів, до яких прикріплюються файли зі скриптами, що реалізують логіку взаємодії.

Кожен об’єкт має назву, а також може бути позначений тегом або належати до певного шару (Layer). Незалежно від типу, усі об’єкти містять обов’язковий компонент Transform, який зберігає координати положення, обертання та масштабу тривимірному просторі.

Unity також підтримує моделювання фізики твердих тіл, тканини та систему Ragdoll (імітація «ганчіркової ляльки») [12]. У редакторі реалізовано ієрархію об’єктів – дочірні об’єкти автоматично наслідують зміни позиції, повороту та масштабу батьківських.

Скрипти додаються до об'єктів як окремі компоненти. Для створення анімацій можна скористатися вбудованим інструментом Animator, або ж імпортувати заздалегідь готові анімації з 2D-редакторів разом із моделями, після чого розбивати їх на окремі кліпи.

Після компілювання проєкту створюється виконуваний файл (.exe) для Windows, а в окремій директорії розміщуються всі супутні дані гри – рівні, текстури, звуки, анімації та бібліотеки, що підключаються динамічно [13].

Переваги Unity:

- рушій безкоштовний;
- низький поріг входження;
- інтуїтивно зрозумілий інтерфейс;
- кросплатформність;
- потужний редактор;
- візуалізація змін у реальному часі;
- компонентно-модульна архітектура.

Головним недоліком Unity є обмежені можливості візуального редактора під час роботи з об'єктами, що містять велику кількість компонентів, що ускладнює керування складними структурами в ієрархії сцени.

На рисунку 2.3 зображено інтерфейс Unity.

2.3.2 GameMaker Studio

GameMaker Studio є одним із найпоширеніших ігрових рушіїв, який забезпечує можливість створення застосунків для широкого спектра платформ – від персональних комп'ютерів до мобільних пристроїв, ігрових консолей та вебсередовищ.

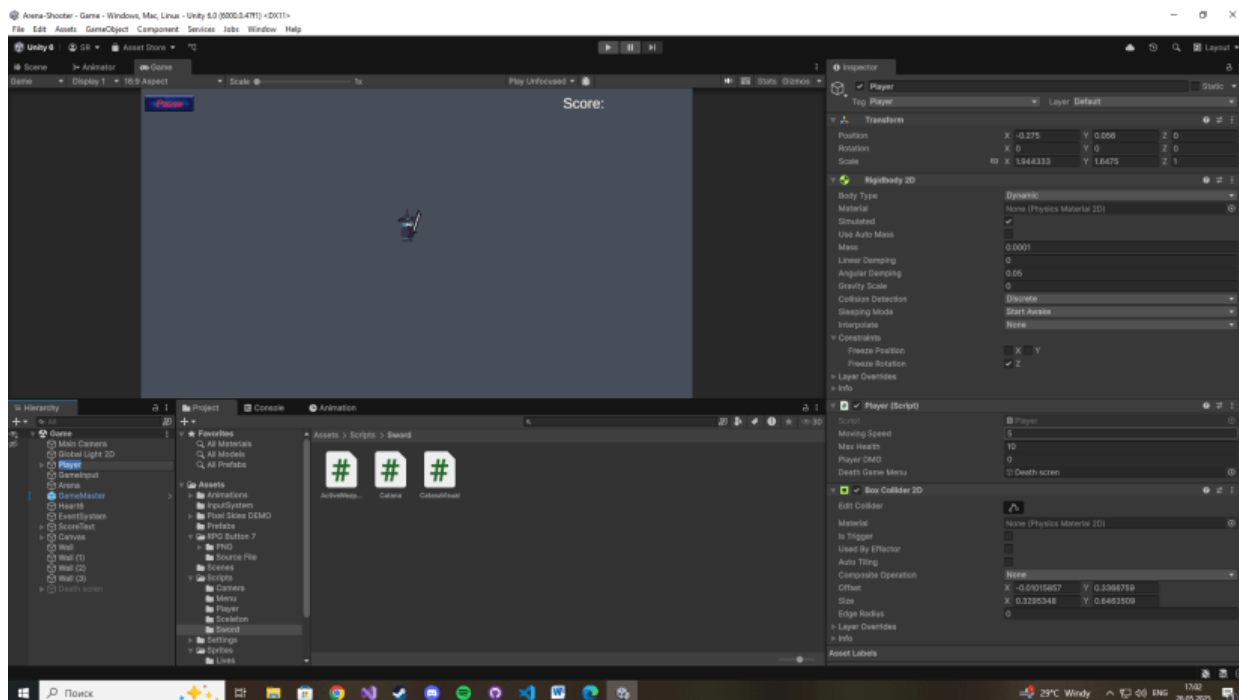


Рисунок 2.3 – Інтерфейс Unity

GameMaker надає можливість створення розширень для численних платформ із використанням відповідних мов програмування. Підтримуються такі формати розширень: GML (універсальний), DLL (Windows, Windows Phone, XboxOne), JavaScript (HTML5), SO (Linux, Tizen), DYLIB (macOS), PRX (PS5), а також спеціальні заглушки для iOS та Android. Доступна функція проксі-розширень, що дозволяє використовувати однакові імена функцій у кодї з прив'язкою до відповідних бібліотек кожної платформи.

GameMaker підтримує роботу з кодуванням Base64, JSON, MD5, розпаковуванням ZIP-архівів, обробкою файлів .ini, текстових та бінарних файлів, а також операціями з каталогами. Крім цього, рушій дозволяє реалізовувати мережеву взаємодію: багатокористувацькі ігри через UDP, TCP, Bluetooth, HTTP-запити, завантаження файлів, а також інтеграцію зі Steam API та Facebook.

З версії 1.1.1086 додано підтримку шейдерів – потужного інструмента для візуальних ефектів і рендерингових трансформацій [14]. Підтримуються вершинні та фрагментні шейдери.

Окрім стандартних елементів керування у вигляді миші та клавіатури реалізовано підтримку геймпадів, джойстиків, сенсорних екранів і датчиків нахилу на мобільних пристроях.

Рушій містить широкий набір математичних функцій для роботи з векторами, скалярами, тригонометричними обчисленнями, логарифмами, інтерполяцією, нормалізацією векторів, а також функціями для обчислення скалярних добутків. Також він містить інтегрований фізичний рушій Box2D [15] з повним набором інструментів для роботи з ним.

На відміну від попередньої версії GameMaker, сучасний GameMaker компілює код, а не інтерпретує його, що значно покращує продуктивність, хоч і виключає можливість динамічного виконання коду під час роботи. Додатково доступний модуль YoYoCompiler, який трансліює GML у C++ та оптимізує виконання. Зібрані за допомогою цього компілятора проекти демонструють високу продуктивність, навіть при великій кількості об'єктів на сцені. Однак підтримка YoYoCompiler обмежена і підтримується не всіма платформами.

GameMaker підтримує популярні сервіси монетизації, такі як AdMob, Google, Analytics, Google Play Licensing. Також передбачено інтеграцію з системами контролю версій – GIT, SVN, Mercurial.

Останні версії цього рушія оснащені розширеним модулем відлагодження (The Debug Module), який дозволяє ставити точку зупину, відстежувати виконання скриптів, переглядати значення змінних, вміст буферів та оцінювати продуктивність кожного фрагменту коду [16].

Переваги GameMaker Studio:

- підтримує розробку під різні платформи, зокрема: Windows, macOS, Linux, Opera GX, Android, IOS, Windows Phone, Tizen, Xbox та Playstation;
- має доступну підтримку зовнішніх бібліотек і розширень, включно з тими, що написані на різних мовах програмування;

- має гнучку систему ліцензування, причому базова версія GameMaker Studio є повністю безкоштовною;
- має інтеграцію з популярними системами контролю версій;
- підтримує платформи дистрибуції, такі як Steam, Google Play та App Store.

Головним недоліком є складна інтеграція з зовнішніми системами.

На рисунку 2.4 зображено інтерфейс GameMaker Studio.

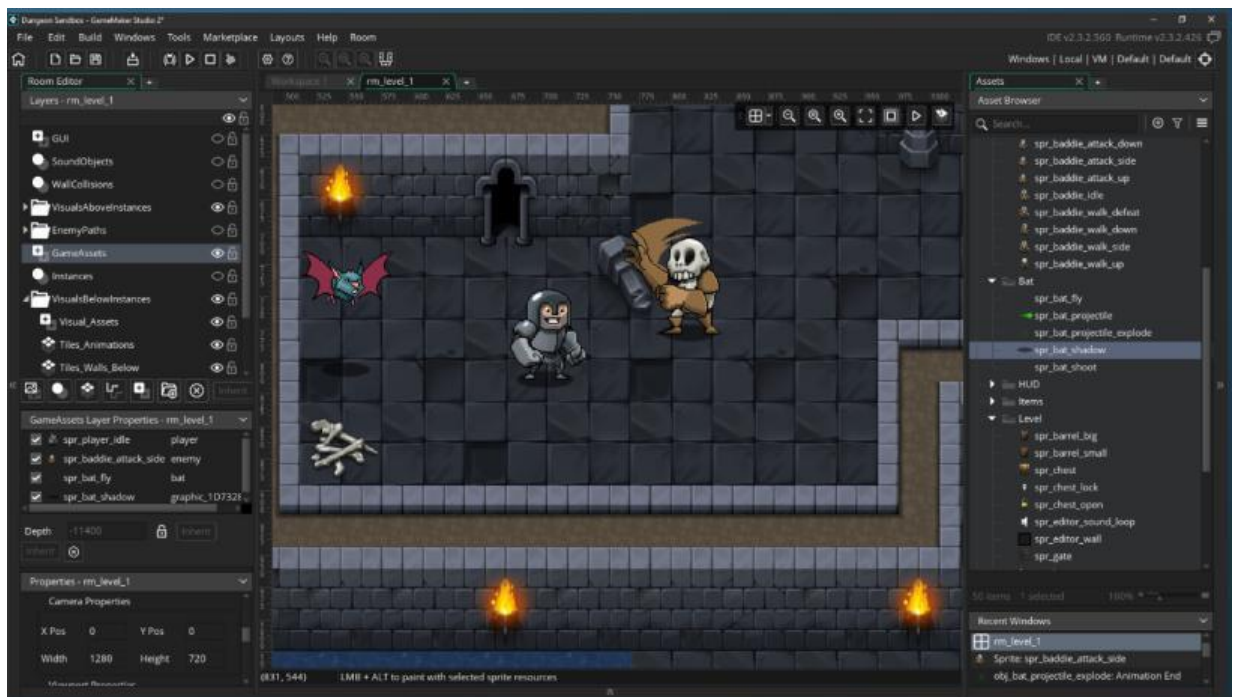


Рисунок 2.4 – Інтерфейс GameMaker Studio

2.3.3 Godot Engine

Godot Engine – це відкритий мультиплатформний ігровий рушій для створення 2D та 3D, що розповсюджується під ліцензією MIT [17]. Його середовище розробки працює на Android, HTML5, Linux, macOS, Windows, BSD та Haiku, а також підтримує експорт проєктів для ПК, ігрових консолей, мобільних пристроїв і вебплатформ.

Головна мета Godot – забезпечити максимально інтегроване і самодостатнє середовище для створення ігор. Воно дозволяє розробникам

працювати «з нуля», без необхідності залучати сторонні інструменти, окрім тих, що потрібні для створення контенту (наприклад, графіку або звуку). Програмування можна виконувати як у вбудованому редакторі, так і в зовнішніх IDE, наприклад Visual Studio Code.

Архітектура рушія базується на дереві спадкових сцен [18], де кожен елемент може стати окремою повноцінною сценою. Це дозволяє гнучко змінювати структуру проєкту, масштабувати її, і працювати зі складними структурами на основі простих абстракцій.

Всі ресурси гри – скрипти, графіка, сцени – зберігаються у вигляді звичайних файлів у папці проєкту, без складних внутрішніх баз даних. Прості ресурси зберігаються у вигляді текстових файлів, що полегшує їх обробку і дозволяє комфортно використовувати системи контролю версій, особливо в командній розробці.

Ігрова логіка в Godot пишеться на власній скриптовій мові програмування GDScript, яка є високорівневою і динамічно типізованою, схожою на Python. Відмінність полягає в обов'язковому оголошенні змінних через ключове слово «var», та оптимізації під систему сцен і вузлів.

Godot включає вбудований редактор скриптів з автодоповненням, автоформатуванням, підсвіткою синтаксису, швидким доступом до повного програмного інтерфейсу (application programming interface, API) рушія. Підтримується також робота з зовнішніми IDE, як Visual Studio Code, із можливістю відлагодження.

Графічна система рушія базується на OpenGL ES 3.0, а починаючи з версії 4.0, додано підтримку Vulkan. Для створення шейдерів [14] застосовується спрощена мова, яка є підмножиною GLSL, з можливістю використання вершинних (vertex) і фрагментних (fragment) шейдерів, а також візуального редактора шейдерів.

Godot має окрему 2D-графічну підсистему, яку можна використовувати незалежно від 3D. Вона підтримує GUI, роботу зі спрайтами, тайлову графіку (звичайну, ізометричну та власного формату),

parallax scrolling, 2D-освітлення, тіні, систему часточок. Розробники можуть комбінувати 2D і 3D через вузол Viewpoint.

У середовищі є вбудована анімаційна система, що підтримує скелетну анімацію [19], анімацію форм і створення кат-сцен. Завдяки вузловій архітектурі можна анімувати будь-які параметри гри.

Фізичний рушій [20], розроблений з нуля, оптимізований для 2D та 3D, підтримує рейкастинг, виявлення зіткнень, динаміку твердих тіл, з'єднання між ними, кінематичний контролер персонажа та базовий контролер транспорту з фізикою підвіски.

Після завершення розробки, проєкт можна експортувати на різні платформи: ПК, мобільні пристрої, консолі і веб. Для кожної платформи можна налаштовувати особливі параметри – зберігання даних, захист, стискування текстур, роздільну здатність тощо. Підтримуються Windows, macOS, X11 (Linux, BSD), Android, iOS, HTML5, а також можливий ручний експорт на інші системи через компіляцію рушія під відповідний пакет інструментів (Software development kit, SDK). Завдяки мінімальній залежності від сторонніх бібліотек цей процес досить простий [21].

Переваги Godot Engine:

- безкоштовна версія;
- кросплатформність;
- інтуїтивна вузлова архітектура;
- легке керування версіями;
- вбудоване середовище розробки.

Головним недоліком цього рушія є значно менша кількість ресурсів і документації, ніж у Unity чи GameMaker Studio.

На рисунку 2.5 зображено інтерфейс Godot Engine.

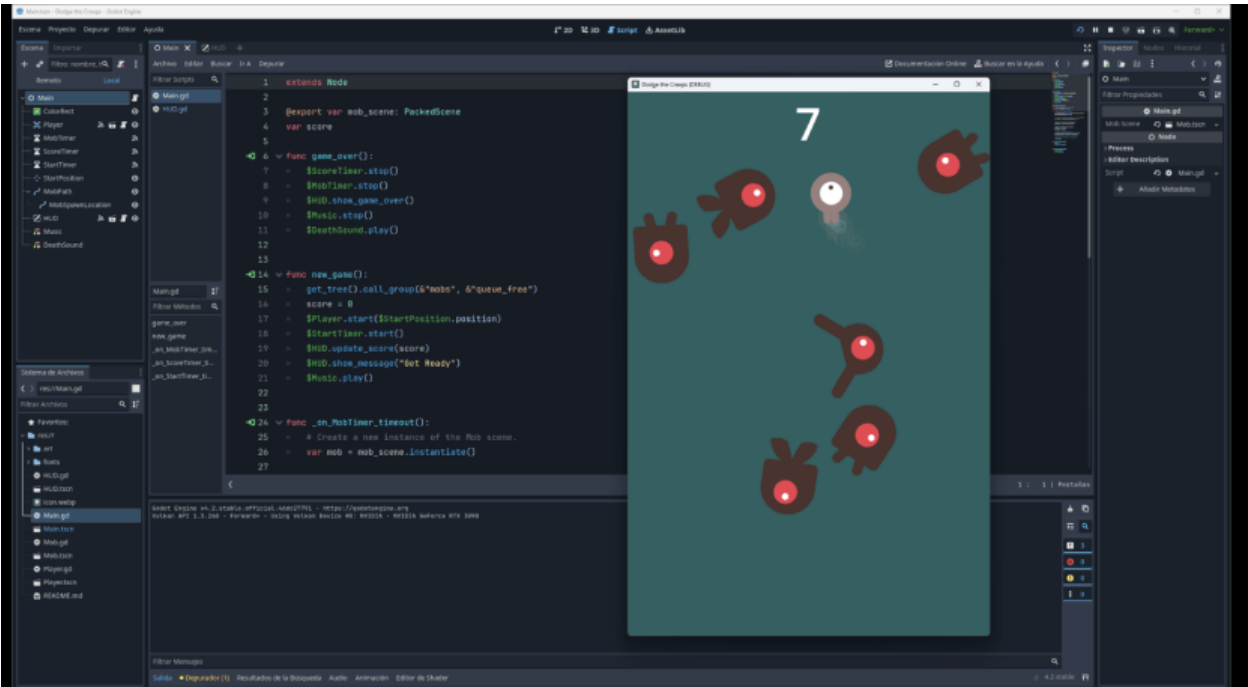


Рисунок 2.5 – Інтерфейс Godot Engine

У таблиці 2.1 наведено порівняння проаналізованих ігрових рушіїв Unity, GameMaker Studio, Godot Engine, з оцінкою ключових характеристик за шкалою від 0 до 5.

Таблиця 2.1 – порівняння ігрових рушіїв Unity, GameMaker Studio, Godot Engine

Характеристика	Unity	GameMaker Studio	Godot Engine
Детальне технічне супроводження	5	4	2
Повноцінні можливості з «коробки»	5	5	4
Підлаштування під різні пристрої та екрани	5		
Підлаштування під різні пристрої та екрани	5	4	5
Розширена функціональність	4	3	5
Продуктивність та ефективність роботи	5	4	5

Продовження таблиці 2.1

Характеристика	Unity	GameMaker Studio	Godot Engine
Здатність до зростання складності проєкту	5	3	5
Легкість освоєння новачками	5	4	2
Кросплатформність	5	5	5
Вільне застосування без ліцензійних витрат	5	4	5

У результаті проведеного порівняльного аналізу було встановлено, що найбільш оптимальним варіантом для реалізації мого проєкту є рушій Unity. Він вирізняється доступністю для новачків, наявністю безкоштовної версії, розвинутими засобами для створення 2D-графіки та анімацій та повною технічною документацією. Godot Engine поступається Unity в тому, що у нього дуже мало доступної документації. Як було зазначено вище, його розроблювала не комерційна компанія чи організація, а програмісти-любители, і, хоча співтовариство шанувальників цього рушія росте з кожним днем, все одно через недостатність інформації почати розроблювати проєкти на ньому дуже важко. Хоча GameMaker Studio демонструє хороші показники за більшістю параметрів, його можливості у сфері 2D-візуалізації поступаються Unity у функціоналі, і через це він не дозволяє повністю задовольнити вимоги проєкту. Саме тому Unity обрано як основний інструмент для реалізації поставленої задачі.

2.4 Особливості Unity як основної платформи розробки

У рамках обраного інструменту розробки програмного забезпечення – ігрового рушія Unity, архітектура майбутньої програми базуватиметься на спеціалізованих модулях. В основі проєкту лежать сцени, які є окремими файлами, що формують незалежні ігрові простори з власними об'єктами,

сценаріями та налаштуваннями. Сцени можуть містити як повноцінні візуальні елементи, так і логічні блоки без графічного представлення, тобто так звані «пусті» об'єкти. На рисунку 2.6 зображена ієрархія об'єктів сцени.

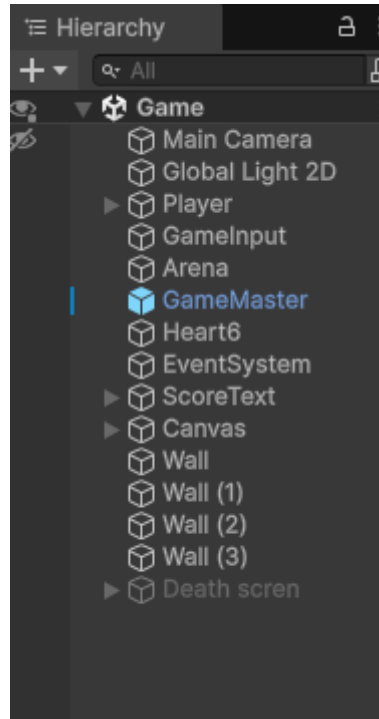


Рисунок 2.6 – Ієрархія об'єктів сцени

Кожен об'єкт у сцені складається з компонентів, які визначають його поведінку та властивості. Скрипти, що описують логіку, взаємодіють саме з цими компонентами. У кожного об'єкта є назва (яка не обов'язково має бути унікальною), теги для класифікації та шари, які відповідають за рендеринг. Одним з базових елементів кожного об'єкта є Transform – компонент, який задає положення, орієнтацію та масштаб у просторі.

Unity дозволяє застосовувати колайдери для реалізації фізичних взаємодій, наприклад коробковий (box), сферичний (sphere), капсульний (capsule). Компоненти об'єктів зображено на рисунку 2.7.

Ієрархія об'єктів в редакторі підтримує систему батьківських і дочірніх елементів, у якій зміни позиції чи масштабу «батьків» автоматично поширюються на всі вкладені об'єкти. Скрипти додаються до об'єктів у вигляді компонентів, що забезпечує модульний підхід до побудови логіки.

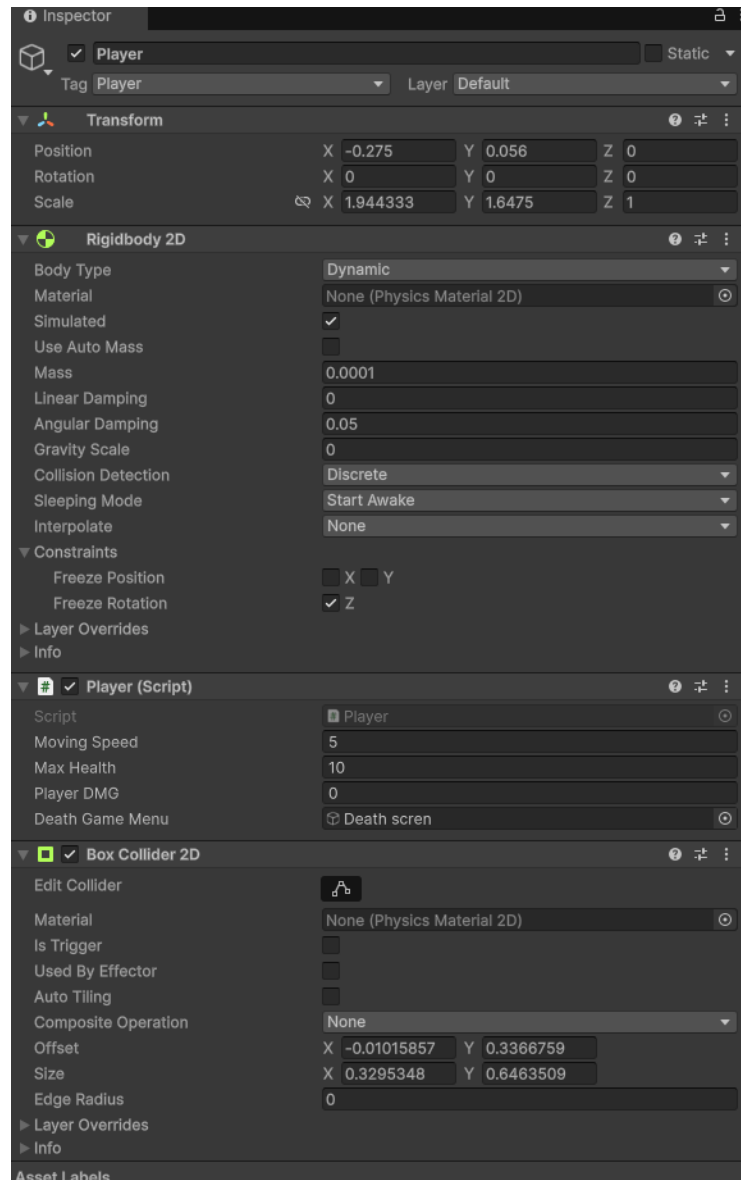


Рисунок 2.7 – Компоненти об'єктів

Рушій підтримує написання власних шейдерів, анімаційні сцени можна створювати як всередині Unity, так і імпортувати ззовні, після чого обробляти їх у редакторі.

Загалом, рушій Unity надає широкий спектр інструментів, необхідних для створення ігор різного типу: фізичне модулювання, рендеринг з динамічним освітленням і тінями, застосування карт нормалей тощо. На відміну від багатьох альтернатив, ключовою перевагою Unity є зручне візуальне середовище розробки. Вбудований редактор не тільки забезпечує

інструменти для моделювання та тестування, а й значно прискорює розробку прототипів.

Також варто зазначити модульну побудову об'єктів у Unity: замість класичного наслідування, як у деяких рушіях, тут використовується композиція – об'єкти створюються шляхом об'єднання функціональних елементів. Такий підхід значно спрощує проектування та дозволяє ефективно працювати з прототипами.

2.5 Вибір IDE

2.5.1 Microsoft Visual Studio

Microsoft Visual Studio є потужним середовищем розробки, що має вбудований редактор коду з підтримкою IntelliSense – інтелектуальної підказки при написанні коду, а також базові інструменти для зручного рефакторингу. Внутрішній відлагоджувач (debugger) дозволяє виконувати як покрокову перевірку вихідного коду, так і працювати на рівні машинних інструкцій.

Серед інших інструментів, що інтегровані у Microsoft Visual Studio, є візуальний редактор форм, який спрощує побудову інтерфейсів користувача, редактор HTML/вебконтенту, конструктор класів та редактор для проектування баз даних.

Однією з головних переваг Microsoft Visual Studio є його гнучкість у розширенні функціоналу – середовище підтримує інтеграцію сторонніх розширень і модулів. Це дозволяє користувачам підключати додаткові інструменти на будь-якому етапі розробки: від інтеграції з системами контролю версій (як наприклад Subversion або Visual SourceSafe), до розширених редакторів, які допомагають працювати з новими мовами чи специфічними прикладними середовищами [22].

2.5.2 JetBrains Rider

JetBrains Rider – це потужне середовище розробки, орієнтоване на створення програм для платформ .NET Core, Xamarin та Unity, яке доступне для Windows, macOS та Linux. Редактор підтримує широку гаму мов, які використовуються в .NET-екосистемі: C#, VB.NET, F#. А також працює з ASP.NET Razor, Javascript, TypeScript, HTML, CSS, SCSS, XAML, XML, JSON і SQL.

Інтелектуальні функції редагування включають кілька типів автодоповнення, шаблони для спрощення написання коду, постфіксні вставки, можливість редагування одночасно в кількох місцях, а також навігацію по ієрархії класів і методів завдяки зручним індикаторам. Крім того, середовище автоматично додає відсутні простори імен, дужки, підсвічує логічні блоки коду, а також забезпечує зручний перехід до рефакторингу, генерації чи навігації всього в кілька натискань.

Rider містить понад 2200 аналізаторів коду, які виявляють помилки, неефективні конструкції або невідповідність, а також понад 1000 автоматизованих способів виправлення таких проблем. Інструмент глобального аналізу проєкту дозволяє знаходити помилки в будь-якому файлі рішення – навіть якщо він наразі не відкритий у вікні редактора.

Для розробників, які працюють з Unity, Rider пропонує глибоку інтеграцію: у верхній панелі розташовані кнопки Play, Pause, Step, що синхронізуються з відповідними елементами редактора Unity. Стан підключення до Unity відображається через спеціальну іконку, а всі зміни в коді автоматично передаються до Unity у режимі редагування.

Ще однією важливою перевагою є можливість декомпіляції зовнішніх бібліотек – користувачі можуть переглядати, налагоджувати та аналізувати навіть зворотно скомпільований код, включно з переходами по функціях, вставлення точок зупинки та переглядом змінних.

Rider також має унікальні інструменти для роботи з Unity-проектами: спеціалізовані інспекції, які виявляють неефективні шаблони коду і пропонують рішення – наприклад, використання кешування, перевантажених методів чи більш оптимального API. Усе це значно покращує продуктивність розробки й дозволяє створювати ефективніші ігрові рішення [23].

Нижче наведено таблицю порівняння функцій Microsoft Visual Studio та JetBrains Rider.

Таблиця 2.2 – Порівняння вище описаних середовищ розробки.

Характеристика	Microsoft Visual Studio	Godot Engine
Підтримка кількох ОС	+	+
Доступність безоплатного варіанту використання	+	-
Повноцінне та структуроване довідкове забезпечення	+	+
Сумісність з ігровим процесом Unity через панель керування	+	-
Повноцінна робота з об'єктною моделлю Unity	+	+
Підтримка інтеграції з системами відстеження змін у коді	+	+
Зручний та інтуїтивно зрозумілий інтерфейс	+	-
Висока швидкодія та продуктивність середовища	+	-
Можливість перегляду та налагодження стороннього скомпільованого коду	-	+

У підсумку можна стверджувати, що серед вище описаних середовищ розробки, Microsoft Visual Studio є найбільш придатним для ефективної та зручної роботи над моїм проєктом, зокрема завдяки налагодженій інтеграції з редактором Unity.

2.6 Висновки за розділом 2

У цьому розділі було розглянуте завдання, пов'язане з проєктуванням програмного забезпечення. В результаті проєктування було визначено архітектурні принципи та обрано фреймворки, які будуть задіяні під час розробки.

Були сформульовані функціональні вимоги. Зокрема створено схематичне зображення, яке ілюструє принцип роботи патерну MVC (Model-View-Controller), що забезпечує логічне розділення системи на дані та інтерфейс.

Також було побудовано діаграму класів, яка візуалізує застосування архітектурної моделі MVC у структурі майбутнього додатку. На її основі визначено основні програмні класи.

Було проведено детальний аналіз трьох ігрових рушіїв: Unity, Godot Engine та GameMaker Studio. На основі цього аналізу складено порівняльну таблицю, що дозволяє оцінити сильні та слабкі сторони кожного варіанту. За результатами аналізу, для реалізації ідеї обрано рушій Unity з огляду на його простоту у вивченні, доступну безкоштовну версію, потужну 2D-графіку та наявність якісної документації.

Окрім цього, здійснено аналіз інструментів розробки – інтегрованих середовищ програмування. Було створено таблицю, яка порівнює різні IDE за ключовими параметрами. Після аналізу Microsoft Visual Studio та JetBrains Rider, остаточний вибір було зроблено на користь Microsoft Visual Studio завдяки глибокій інтеграції з редактором Unity, підтримці роботи з ігровими об'єктами, та завдяки високій продуктивності.

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглянуто процес створення програмної частини ігрового застосунку. Основну увагу приділено наступним аспектам: побудова ігрової логіки, структура проекту, реалізація алгоритму створення хвиль супротивників.

3.1 Система дій, реакцій і відгуків у грі

Ігровий процес (англ. Gameplay) – це характер і спосіб взаємодії користувача з відеогрою. Цей процес формується на основі визначених механік: правил, цілей і методів їх досягнення, що передбачені розробником. Разом із сюжетною складовою та технічною реалізацією, ігрова механіка становить один із ключових компонентів будь-якого ігрового продукту [24].

У центрі цього процесу зазвичай лежить циклічна система: виконання дії (Дія) – реакція системи (Реакція) – відповідний зворотний сигнал (Відгук), який впливає на подальші рішення гравця.

3.1.1 Дія

Дія (англ. Action) – це будь-яке введення, яке гравець здійснює за допомогою пристроїв керування (контролерів). У межах мого проекту така взаємодія реалізована через клавіатуру.

Гравець взаємодіє з головною героїнею (відьма), яка пересувається у 2D просторі за осями X та Y використовуючи клавіші WASD на клавіатурі, та б'є мечем при натисканні ЛКМ. Якщо підійти до ворога близько, то у відьми знизиться її максимальне здоров'я, і якщо воно впаде до 0, то вона помре і на екран висвітиться меню програшу, на якому буде показано ваш результат [25].

3.1.2 Реакція

Реакція (англ. Reaction) – це відповідь ігрової логіки на введення з боку користувача, яка вже сформована, але ще не представлена гравцю у візуальному або звуковому вигляді.

Під час натискання клавіш клавіатури WASD, об'єкт «відьма» змінює своє положення у 2D просторі. Меч (катана) зроблена також окремим об'єктом. І хоча вона прив'язана до об'єкта «відьма», при натисканні ЛКМ він також змінює своє положення у просторі.

3.1.3 Відгук

Відгук (англ. Feedback) – це інформація, яку гравець отримує від гри у відповідь на свої дії. Вона має дві основні форми:

- явна – чітко показує, як гра реагує на взаємодію користувача;
- неявна – виконує фонову чи допоміжну функцію, наприклад, через підказки або інтерфейс.

Існує кілька типів зворотнього зв'язку, які можуть комбінуватись:

- візуальний – все що відображається на екрані;
- аудіальний – звукові ефекти або музика, що супроводжують події.
- ігровий – дії самої гри у відповідь на користувацький ввід;
- взаємодія з персонажами – реакції неігрових героїв (NPC);
- прогресивний – зміни, що демонструють досягнення гравця чи розвиток подій;
- емоційний – переживання або емоції, які викликає ігровий процес;
- змістовий – спосіб донесення основної ідеї гри або конкретного завдання;
- інформативний – дані, що допомагають краще орієнтуватися в грі.

Об'єкти «відьма» і меч анімовані. При натисканні обраних клавіш змінюється анімація відьми зі спокою (Idle) на біг (Run). Об'єкт меч також анімований, у нього є два стани: Idle (спокій) та Attack (атака).

При програші, коли об'єкт «відьма» вмирає, один раз відбувається її анімація смерті (Death), та з'являється вікно програшу з поточним рахунком очок, які були набрані за поточне проходження. У вікні програшу є дві кнопки, а саме: кнопка повернення до меню, щоб ще раз запустити гру, і кнопка повного виходу з гри, при натисканні якої гра повністю закривається.

3.2 Компоненти проєкту

Проєкт має модульну структуру, що включає кілька ключових елементів: скрипти (які визначають логіку взаємодії об'єктів у межах ігрової сцени), анімації, параметри сцени (границі ігрового поля), а також елементи користувацького інтерфейсу, через які гравець взаємодіє з грою.

Застосування архітектурного підходу MVC дало змогу організувати розробку так, щоб програмна логіка функціонувала незалежно від графічного оформлення. Це означає, що зовнішній вигляд гри – моделі, елементи інтерфейсу тощо – можна змінювати без необхідності втручання в основний код.

У таблиці 3.1 представлено перелік скриптів, що реалізують основну механіку ігрового процесу, засновану на послідовності «Дія – Реакція – Відгук».

Таблиця 3.1 – Перелік та опис скриптів.

Перелік	Опис
Game Master.cs	Відповідає за створення хвиль ворогів, відстежує кількість ворогів на сцені, ініціює появу нових хвиль після знищення попередніх, веде підрахунок очок +
GameInput.cs	Відповідає за користувацький ввід у грі. Отримує, інтерпретує та передає дії гравця, такі як рух, атаку та позицію миші. +
Camera Follow.cs	Відповідає за автоматичне стеження камери за ігровим об'єктом.+
DeathMenu.cs	Реалізує механіку зупинки гри та відображення меню смерті гравця, коли настає його поразка.
MainMenu.cs	Відповідає за обробку взаємодії гравця з головним меню гри. Реалізує базову навігацію між сценами, запуск гри та вихід із неї.
PauseMenu.cs	Реалізує просте меню паузи в грі, яке дозволяє гравцю зупинити або відновити гру. Також відповідає за навігацію між сценами.
Player.cs	Відповідає за керування гравцем у грі: за його рух, здоров'я, атаки та обробку смерті.
PlayerVisual.cs	Відповідає за візуальне відображення гравця: анімації руху, смерті та атаки, а також за напрямок у який дивиться персонаж
EnemyScript.cs	Реалізує поведінку ворога в грі: знаходження об'єкта персонажа, отримання пошкоджень і взаємодію з гравцем при дотику.

Продовження таблиці 3.1

PlayerVisual.cs	Відповідає за візуальне відображення гравця: анімації руху, смерті та атаки, а також за напрямок у який дивиться персонаж
EnemyScript.cs	Реалізує поведінку ворога в грі: знаходження об'єкта персонажа, отримання пошкоджень і взаємодію з гравцем при дотику.
SkeletonVisual.cs	Керує візуальними аспектами ворога-скелета: анімаціями (біг, атака, смерть) та напрямком погляду відносно позиції об'єкта гравця.
ActiveWeapon.cs	Відповідає за керування активною зброєю гравця (катаною). Забезпечує орієнтацію зброї у напрямку курсора миші на екрані.
Catana.cs	Відповідає за атаку гравця за допомогою катани, увімкнення/вимкнення колайдера для реєстрації влучень. Надсилає події про атаку і нанесення шкоди ворогам при зіткненні з ними.
CatanaVisual.cs	Синхронізує логіку атаки скрипта Catana з анімацією, візуалізує удар, керує моментом активації/деактивації колайдера катани.

На рисунку 3.1 представлено загальну файлову структуру проєкту.

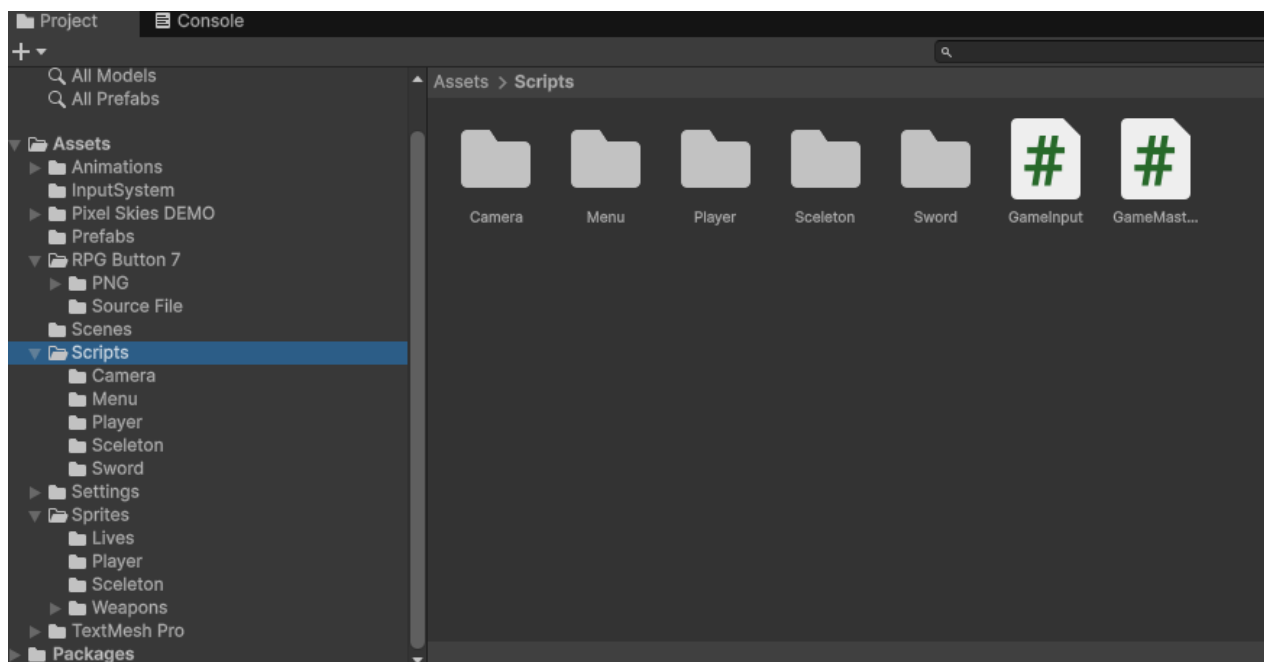


Рисунок 3.1 – Файлова структура проєкту

На рисунку 3.2 показано ієрархічну структуру об'єктів, розміщених на сцені.

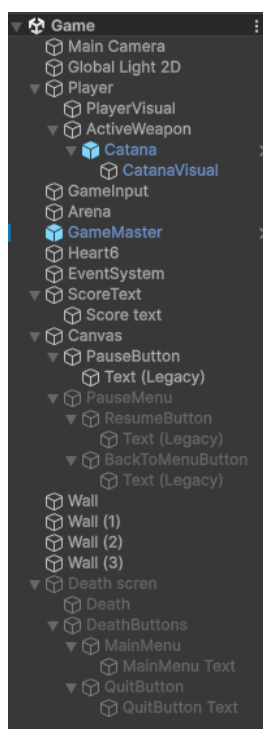


Рисунок 3.2 – Ієрархічна структура об'єктів на сцені

3.3 Опис алгоритму створення хвиль ворогів

Алгоритм створення хвиль ворогів базується на поступовому збільшенні кількості ворогів у кожній хвилі з випадковим розміщенням їх у просторі. Він працює так:

- підрахунок ворогів на сцені – кожний кадр підраховується поточна кількість ворогів на сцені;
- перехід до нової хвилі – якщо ворогів не залишилось (їхня кількість дорівнює 0), збільшується номер хвилі на 1, і викликається генерація нової хвилі;
- генерація хвилі – нова хвиля складається з ворогів у кількості, що дорівнює номеру хвилі;
- розміщення ворогів – кожен ворог з'являється у випадковій позиції в межах заданої області, але з урахуванням мінімальної відстані від центру, щоб уникнути занадто близької появи до гравця.

На рисунку 3.3 зображено алгоритм створення хвиль ворогів.

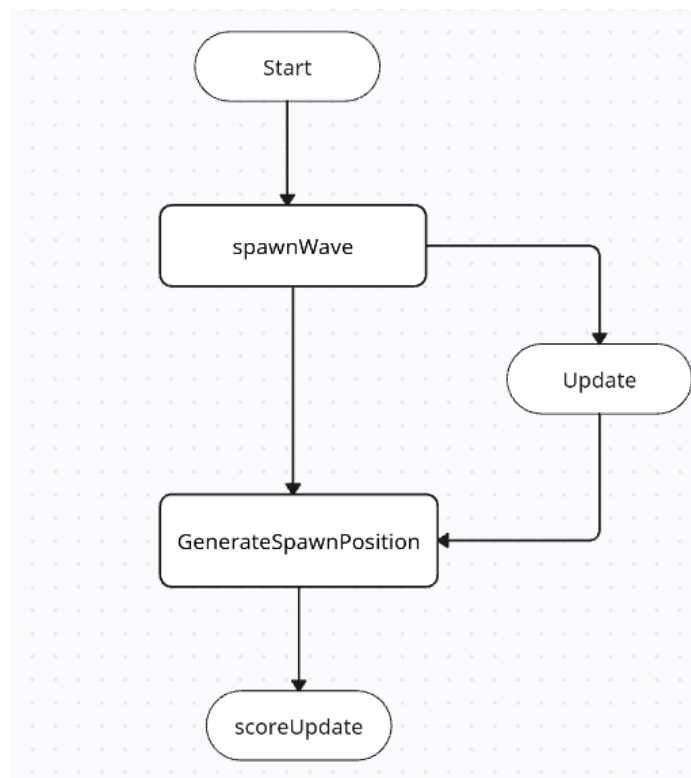


Рисунок 3.3 – Алгоритм створення хвиль ворогів

3.4 Висновки за розділом 3

В даному розділі було виконано формування концепції програмного забезпечення, що дозволило чітко окреслити, який функціонал потрібно реалізувати.

Проведено аналітичне дослідження ігрового процесу – визначено основні етапи, які проходить гравець у ході гри. Відповідно до цього аналізу була розроблена структура ігрового процесу майбутнього застосунку.

Створено логічну архітектуру проєкту. До її складу входять скрипти, які відповідають за логіку взаємодії ігрових об'єктів на сцені, елементи сцени, а також інтерфейс користувача, який забезпечує зв'язок між гравцем і грою. Це дозволило конкретизувати перелік компонентів, які потребують реалізації.

Сформовано алгоритм генерації хвиль супротивників. Його логіка полягає у тому, що на початку гри створюється перша хвиля ворогів, кількість яких дорівнює номеру хвилі. У процесі гри відбувається моніторинг кількості активних ворогів. Якщо всі вороги знищені, номер хвилі збільшується, і генерується нова хвиля. Координати для спавну обираються випадковим чином, при цьому використовується перевірка на мінімальну відстань від центру, щоб уникнути появи ворогів надто близько до гравця.

4 КЕРІВНИЦТВО КОРИСТУВАЧА

У цьому розділі наведено системні вимоги, функціональні можливості розробленого застосунку, інструкція з експлуатації, та опис його основних принципів роботи.

4.1 Системні вимоги

Для роботи програми необхідний персональний комп'ютер з операційною системою Windows 10.

4.2 Функціональні можливості застосунку

Застосунок призначений для розваги та веселого проведення часу.

Застосунок має такий функціонал:

- завантаження, запуск та вихід із програми;
- відображення головного меню;
- перехід на сцену зі грою;
- запуск ігрової сесії, під час якої гравець керує головним персонажем з метою пережити якомога більше хвиль ворогів. Упродовж цього ігрового процесу на екрані відображаються такі елементи: лічильник зароблених очок, кнопка паузи гри;
- відображення екрану програшу при смерті головного персонажа з такими кнопками: кнопка виходу у меню, щоб запустити ігрову сесію спочатку, кнопка повного виходу з гри.

4.3 Інструкція з експлуатації

Для запуску програми, потрібно завантажити ZIP-архів, розпакувати усі файли у вказану папку, запустити .exe файл з іконкою гри.

4.4 Принцип роботи

Щоб розпочати роботу з ігровим застосунком, потрібно його запустити, двічі клацнувши .exe файл з іконкою гри. На рисунку 4.1 представлено головне меню програми англійською мовою, яка була обрана як мова за замовчуванням.



Рисунок 4.1 – Головне меню додатку

Для того, щоб почати ігрову сесію треба натиснути кнопку START.

Після натискання кнопки START запускається ігрова сцена, зображена на рисунку 4.2.

На ігровій сцені здійснюється керування ігровим персонажем за допомогою клавіш WASD. Мета гри – пережити якомога більше хвиль супротивників, та набрати якомога більше очок.

Очки нараховуються за вбивство кожного ворога за схемою: 1 ворог = 10 очок. На рисунку 4.3 показано розміщення лічильника кількості очок за вбивство ворогів і інтерфейсі.

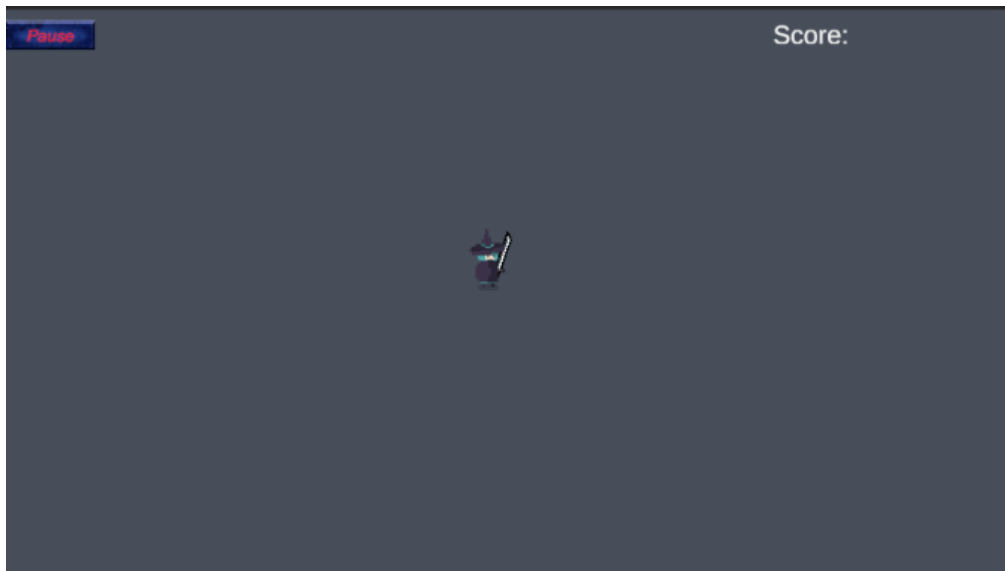


Рисунок 4.2 – Ігрова сцена

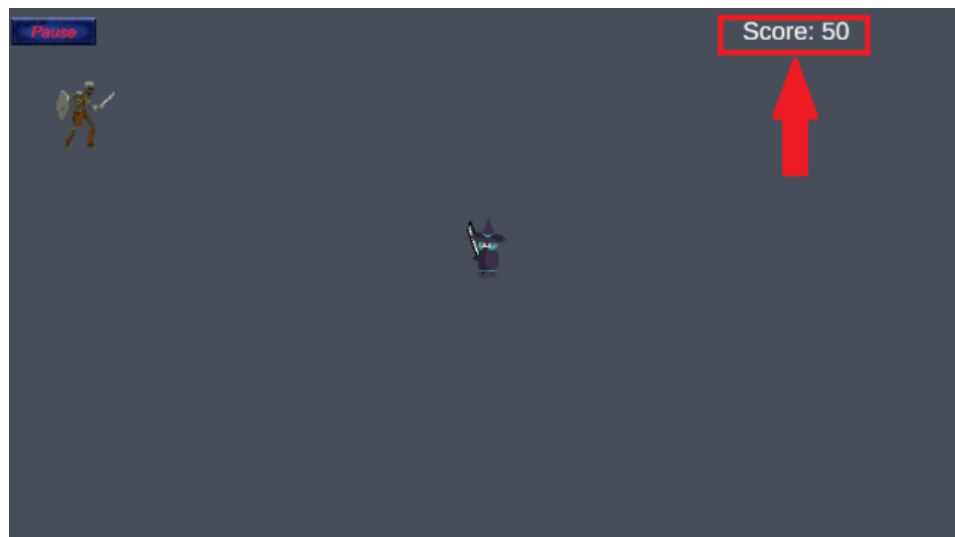


Рисунок 4.3 – Кількість набраних очок за знищення ворогів

При втраті всіх очок життя (за замовчуванням персонаж має 10) програється анімація смерті персонажа, сцену буде затемнено, і з'явиться екран програшу на якому в центрі екрану буде написано «Death», а також з'являться дві кнопки: кнопка MainMenu та Quit. На цьому екрані також буде видно кількість очок, яку гравець встиг заробити за поточне проходження (рис. 4.4).

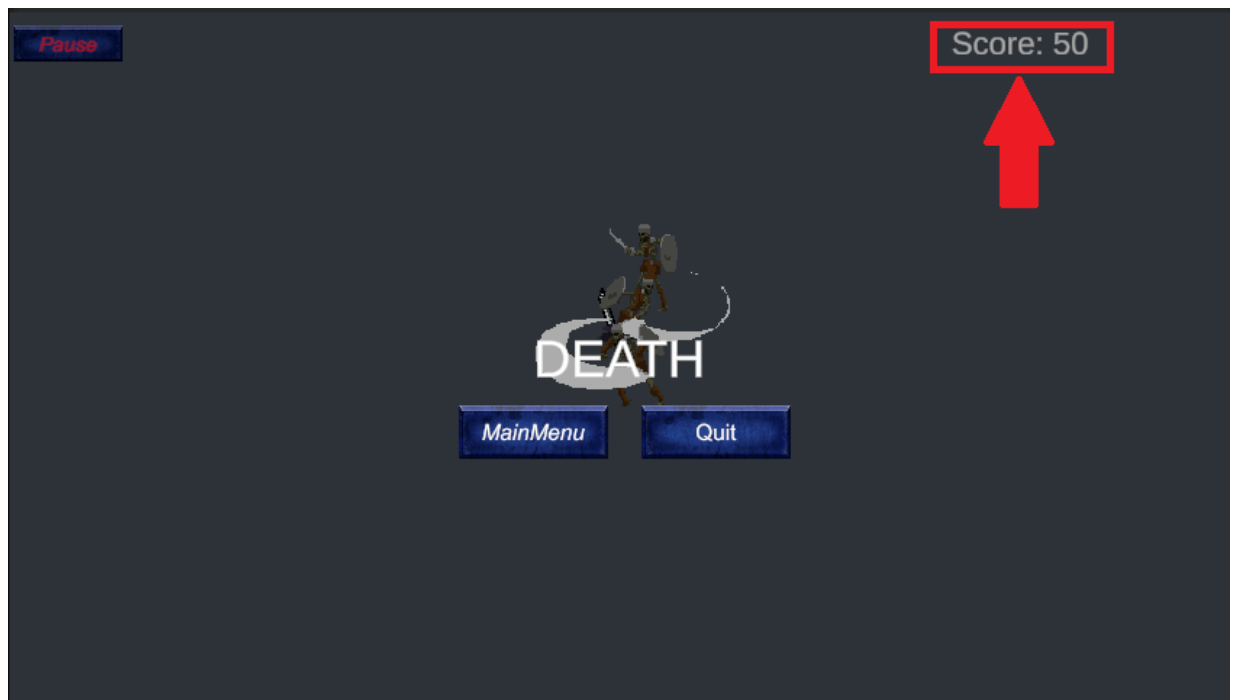


Рисунок 4.4 – Екран програшу

При натисканні кнопки *Pause* відбудеться зупинка сцени і з'являться дві кнопки посеред екрану: кнопка *Continue* і *Menu*. При натисканні кнопки *Continue* гра вийде з паузи та продовжиться, а при натисканні кнопки *Menu* відбудеться зміна сцени на головне меню (рис. 4.5).



Рисунок 4.5 – Меню паузи

4.5 Висновки за розділом 4

У цьому розділі виконано підготовку користувацької документації до розроблюваного застосунку, що забезпечує чітке розуміння принципів взаємодії з ігровим додатком для кінцевого користувача.

Проведено аналіз системних вимог, необхідних для коректної роботи гри на ПК під керуванням операційної системи (ОС) Windows 10. Розглянуто ключові параметри, що впливають на стабільність та продуктивність програмного забезпечення.

Окремо наведено покрокову інструкцію з запуску програми та роз'яснено порядок взаємодії користувача з усіма основними елементами інтерфейсу.

ВИСНОВКИ

В результаті виконання дипломної роботи було розглянуто ігрові застосунки жанру Roguelike. Було проведено аналіз вже існуючих аналогів цього жанру.

Виконано огляд сучасних ігрових рушіїв для створення ігрових проєктів відповідного жанру, з детальним описом їх можливостей та переваг. Перевагами не дуже розповсюдженого аналогу Godot Engine є:

- інтуїтивна вузлова архітектура;
- вбудоване середовище розробки.

Перевагами аналогу GameMaker Studio є:

- доступна підтримка зовнішніх бібліотек і розширень, включно з тими, що написані на різних мовах програмування;
- інтеграції з популярними системами контролю версій.

Але вони також мають недоліки, як то:

- мало ресурсів і документації у Godot Engine;
- складна інтеграція з зовнішніми системами у GameMaker Studio.

У результаті проведення аналізу за визначеними критеріями було обрано ігровий рушій Unity, що вирізняється простотою у вивченні, наявністю безкоштовної версії, розвиненою системою 2D-візуалізації та доступною й якісною документацією.

Також було розглянуто сучасні інструменти розробки програмного коду. Для реалізації проєкту було обрано Microsoft Visual Studio як найбільш сумісне середовище з Unity та одне з найзручніших у роботі.

У межах роботи було проаналізовано основні складові ігрового процесу, зокрема такі елементи, як «Дія», «Реакція», «Відгук», що становлять основу ігрової взаємодії.

Розглянуто структуру проєкту та архітектурні рішення, які лягли в основу його побудови. В якості основної архітектурної моделі було

використано шаблон MVC, що забезпечує чіткій поділ логіки, візуального представлення та обробки введення користувача.

Описано реалізацію алгоритму для генерації нескінченних хвиль ворогів, який забезпечує поступове зростання складності гри.

Подано характеристику функціональних можливостей розробленої системи та умови її використання. Надано покрокову інструкцію з встановлення та запуску програмного продукту, а також пояснення щодо взаємодії з основними елементами ігрового процесу.

В рамках роботи було реалізовано ігровий проєкт в жанрі Roguelike, що відповідає усім поставленим функціональним та технічним вимогам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rogue [Electronic resource]. – Access mode: <https://store.steampowered.com/app/1443430/Rogue/>.
2. Harris J. Exploring Roguelike Games / John Harris. – CRC Press, 2021. – 556 p
3. Temple of the roguelike [Electronic resource]. – Access mode: <https://blog.roguetemple.com/history/>.
4. Brotato [Electronic resource]. – Access mode: <https://store.steampowered.com/app/1942280/Brotato/>.
5. Vampire Survivors [Electronic resource]. – Access mode: https://store.steampowered.com/app/1794680/Vampire_Survivors/.
6. Enter the Gungeon [Electronic resource]. – Access mode: https://store.steampowered.com/app/311690/Enter_the_Gungeon/.
7. Фримен А. ASP.NET MVC 3 Framework з прикладами на C# для професіоналів / Адам Фримен, Стивен Сандерсон. – Вільямс, 2012. – 672 с.
8. Creately [Electronic resource]. – Access mode: <https://app.creately.com/d/start/dashboard>.
9. Miles R. Learning UML 2.0: A Pragmatic Introduction to UML / Russ Miles, Kim Hamilton. – O`Reilly, 2006. – 283 p.
10. Gregory J. Game Engine Architecture / Jason Gregory. – CRC Press, 2018. – 1218 p.
11. Редактор рівнів [Electronic resource]. – Access mode: https://wiki.frictionalgames.com/page/HPL2/Tools/Level_Editor.
12. Ragdoll-фізика [Electronic resource]. – Access mode: <https://docs.unity3d.com/6000.0/Documentation/Manual/ragdoll-physics-section.html>.
13. Ігровий рушій Unity [Electronic resource]. – Access mode: <https://unity.com/products/unity-engine>.

14. Шейдер [Electronic resource]. – Access mode: <https://unity.com/blog/engine-platform/shader-variants-optimization-troubleshooting-tips>.
15. Box2D [Electronic resource]. – Access mode: <https://support.unity.com/hc/en-us/articles/360015178512-Determinism-with-2D-Physics>.
16. Ігровий рушій GameMaker Studio [Electronic resource]. – Access mode: <https://gamemaker.io/en>.
17. Ліцензія MIT [Electronic resource]. – Access mode: <https://opensource.org/license/mit>.
18. Дерево (структура даних) [Electronic resource]. – Access mode: <https://www.wscubetech.com/resources/dsa/tree-data-structure>.
19. Скелетна анімація [Electronic resource]. – Access mode: <https://learnopengl.com/Guest-Articles/2020/Skeletal-Animation>.
20. Фізичний рушій [Electronic resource]. – Access mode: <https://docs.unity3d.com/6000.1/Documentation/Manual/PhysicsSection.html>.
21. Ігровий рушій Godot Engine [Electronic resource]. – Access mode: <https://godotengine.org/>.
22. IDE Visual Studio [Електронний ресурс]. – Режим доступа: <https://visualstudio.microsoft.com/ru/#vs-section>.
23. IDE JetBrains Rider [Electronic resource]. – Access mode: <https://www.jetbrains.com/rider/>.
24. Rouse R. Game Design: Theory and Practice, Second Edition / Richard Rouse. – Jones & Bartlett Learning, 2004. - 704 p.
25. Ігровий процес [Electronic resource Electronic resource]. – Access mode: <https://www.merriam-webster.com/dictionary/gameplay>.

ДОДАТОК А
Технічне завдання

Вступ

Розробка ігрового додатку зумовлена актуальною темою про широке використання ігор у різних сферах – від освіти до комерційних продуктів – а також проблемою у потребі фахівців, здатних проєктувати та реалізовувати ігрові додатки на сучасному рівні.

Метою роботи є розроблення програмного продукту, що реалізує функціональність двовимірної комп'ютерної гри з використанням сучасного ігрового рушія, а також створення повноцінної ігрової логіки, графічного інтерфейсу користувача, анімації, та індивідуального ігрового контенту відповідно до обраного жанру – Roguelike.

А.1 Підстава для розробки

Ігровий додаток розробляється на підставі завдання на дипломну кваліфікаційну роботу затвердженого наказом № 209 від 28 квітня 2025 року Національного Університету «Запорізька політехніка» на тему «Програмна реалізація двовимірної комп'ютерної гри».

А.2 Призначення розробки

Готовий ігровий додаток повинен надавати користувачу базовий ігровий досвід та приємні емоції від гри.

А.3 Основні вимоги до розроблюваного проєкту

А.3.1 Вимоги надійності

Програма повинна стабільно працювати протягом тривалого часу без неочікуваних збоїв або аварійного завершення процесу.

Алгоритми гри повинні бути оптимізовані таким чином, щоб забезпечити стабільну роботу навіть при великій кількості одночасно активних об'єктів на сцені.

Класи, що відповідають за логіку гравця, ворогів, зброї та GUI, мають працювати синхронізовано без порушення логіки гри при багатопоточному або циклічному виклику методів.

А.3.2 Мінімальна конфігурація обладнання для функціонування гри

Для забезпечення коректної роботи двовимірної гри необхідно дотримуватись певних технічних вимог до апаратного забезпечення, зокрема:

- оперативна пам'ять – не менше 4 Гб;
- жорсткий диск (HDD) – не менше 64 Гб;
- процесор (CPU) – не менше 2 ГГц;
- операційна система (ОС) – Windows 10;
- відеокарта – не менше 1050 TI.

А.3.3 Умови маркування та оформлення пакування

Збереження програми можливе на зовнішніх носіях (USB-накопичувачах, CD/DVD-дисках), а також у внутрішній пам'яті комп'ютера.

А.3.4 Умови транспортування та зберігання

Порядок зберігання та транспортування програмного забезпечення визначається характеристиками фізичного носія, на якому воно записане, і не відрізняється від стандартних вимог до обігу цифрових даних.

A.3.5 Вимоги до документації

Додаток має подаватися разом із «Керівництвом користувача».

ДОДАТОК Б
Текст програми

Б.1 Текст файла GameManager.cs

```

using System;
using System.Numerics;
using System.Reflection;
using TMPro;
using Unity.Mathematics;
using UnityEngine;

public class GameManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    [SerializeField] private float spawnRangeEnd = 10;
    [SerializeField] private float spawnRangeMin = 5;
    [SerializeField] public int waveNumber = 1;
    public int enemyCount;

    private float score;
    public TextMeshProUGUI scoreText;
    void Start()
    {
        spawnWave(waveNumber);
        score = 0;
        scoreText.text = "Score: " + score;
    }
    void Update()
    {
        enemyCount = FindObjectsOfType<EnemyScript>(FindObjectsSortMode.None).Length;

        if (enemyCount == 0)
        {
            waveNumber++;
            spawnWave(waveNumber);

            //if (waveNumber >= 4)
            //{
            //    Debug.Log("Congrats");
            //    //Stop this game!
            //}
            //else
            //{
            //}

        }
    }

    public void spawnWave(int Wave)
    {
        for (int i = 0; i < Wave; i++)
        {
            Instantiate(enemyPrefab, GenerateSpawnPosition(),
            enemyPrefab.transform.rotation);
        }
    }

    private UnityEngine.Vector3 GenerateSpawnPosition()
    {
        float spawnPosX = UnityEngine.Random.Range(-spawnRangeEnd,
spawnRangeEnd);
        float spawnPosY = UnityEngine.Random.Range(-spawnRangeEnd,
spawnRangeEnd);
    }
}

```

```

        UnityEngine.Debug.Log("=====координаты
изменения=====");
        UnityEngine.Debug.Log(spawnPosX);
        UnityEngine.Debug.Log(spawnPosY);
        if (spawnPosX < spawnRangeMin && spawnPosX > 0)
            spawnPosX = spawnPosX + spawnRangeMin;
        else if (spawnPosX > -spawnRangeMin && spawnPosX < 0)
            spawnPosX = spawnPosX - spawnRangeMin;
        if (spawnPosY < spawnRangeMin && spawnPosY > 0)
            spawnPosY = spawnPosY + spawnRangeMin;
        else if (spawnPosY > -spawnRangeMin && spawnPosY < 0)
            spawnPosY = spawnPosY - spawnRangeMin;
        UnityEngine.Debug.Log("+++++++координаты после изменения+++++++");
        UnityEngine.Debug.Log(spawnPosX);
        UnityEngine.Debug.Log(spawnPosY);
        UnityEngine.Vector2 randomPos = new UnityEngine.Vector3(spawnPosX,
spawnPosY, 0);
        return randomPos;
    }
    public void scoreUpdate(float amountScore)
    {
        score += amountScore;
        scoreText.text = "Score: " + score;
    }
}

```

Б.2 Текст файлу GameInput.cs

```

using System;
using UnityEngine;
using UnityEngine.InputSystem;

public class GameInput : MonoBehaviour
{
    public static GameInput Instance { get; private set; }

    private InputSystem_Actions player;

    public event EventHandler OnPlayerAttack;

    private void Awake()
    {
        Instance = this;

        player = new InputSystem_Actions();
        player.Enable();

        player.Combat.Attack.started += PlayerAttack_started;
    }

    private void PlayerAttack_started(InputAction.CallbackContext obj)
    {
        OnPlayerAttack?.Invoke(this, EventArgs.Empty);
    }

    public Vector2 GetMovementVector()
    {
        Vector2 inputVector = player.Player.Move.ReadValue<Vector2>();

        return inputVector;
    }
}

```

```

    public Vector3 GetMousePosition()
    {
        Vector3 mousePos = Mouse.current.position.ReadValue();
        return mousePos;
    }

    public void DisableMovement()
    {
        player.Disable();
    }
}

```

Б.3 Текст файла CameraFollow.cs

```

using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    public Transform player;
    public Vector3 offset = new Vector3(0, 0, -10);

    void LateUpdate()
    {
        if (player != null)
        {
            transform.position = player.position + offset;
        }
    }
}

```

Б.4 Текст файла DeathMenu.cs

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class DeathScreen : MonoBehaviour
{
    public bool PauseGame;
    public GameObject deathGameMenu;

    public void Pause()
    {
        deathGameMenu.SetActive(true);
        Time.timeScale = 0f;
        PauseGame = true;
    }
}

```

Б.5 Текст файла MainMenu.cs

```

using UnityEngine;
using UnityEngine.SceneManagement;

```

```

public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex +
1);
        Time.timeScale = 1f;
    }

    public void ExitGame()
    {
        Debug.Log("❖❖❖❖ ❖❖❖❖❖❖❖❖❖❖");
        Application.Quit();
    }
    public void MainMenuLoad()
    {
        SceneManager.LoadScene("MainMenu");
    }
}

```

Б.6 Текст файлу PauseMenu.cs

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public bool PauseGame;
    public GameObject pauseGameMenu;

    private void Update()
    {
        if(Input.GetKeyDown(KeyCode.Escape))
        {
            if(PauseGame)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseGameMenu.SetActive(false);
        Time.timeScale = 1f;
        PauseGame = false;
    }

    public void Pause()
    {
        pauseGameMenu.SetActive(true);
        Time.timeScale = 0f;
        PauseGame = true;
    }

    public void LoadMenu()
    {

```

```

        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }
}

```

Б.7 Текст файла Player.cs

```

using System;
using UnityEngine;

public class Player : MonoBehaviour
{
    public static Player Instance { get; private set; }

    [SerializeField] private float movingSpeed = 10f;
    [SerializeField] private int maxHealth = 10;

    private Rigidbody2D _rb;

    private float minMoveSpeed = 0.1f;

    private bool isRunning = false;

    public float playerDMG;

    private int currHealth;

    private bool isDead = false;

    public event EventHandler OnPlayerDeath;

    public GameObject deathGameMenu;

    private void Start()
    {
        currHealth = maxHealth;
        GameInput.Instance.OnPlayerAttack += GameInput_OnPlayerAttack;
    }

    private void GameInput_OnPlayerAttack(object sender, System.EventArgs e)
    {
        ActiveWeapon.Instance.GetActiveWeapon().Attack();
    }

    private void Awake()
    {
        Instance = this;
        _rb = GetComponent<Rigidbody2D>();
    }

    private void FixedUpdate()
    {
        HandleMovement();
    }

    public void TakeDamage(Transform damageSource, int damage)
    {
        currHealth = Mathf.Max(0, currHealth - damage);
        Debug.Log(currHealth);
        if (currHealth <= 0)
        {

```

```

        OnPlayerDeath?.Invoke(this, EventArgs.Empty);
        GameInput.Instance.DisableMovement();
        Die();
    }
}

private void HandleMovement()
{
    Vector2 inputVector = GameInput.Instance.GetMovementVector();
    inputVector = inputVector.normalized;
    _rb.MovePosition(_rb.position + inputVector * (movingSpeed *
Time.fixedDeltaTime));

    if (Mathf.Abs(inputVector.x) > minMoveSpeed ||
Mathf.Abs(inputVector.y) > minMoveSpeed)
    {
        isRunning = true;
    }
    else
    {
        isRunning = false;
    }
}

private void Die()
{
    if (isDead) return; // ?????? ? ???? ????
    isDead = true;

    // ?????? ???? ???? ????
    GetComponent<Collider2D>().enabled = false;
    _rb.linearVelocity = Vector2.zero;
    _rb.bodyType = RigidbodyType2D.Kinematic;
    deathGameMenu.SetActive(true);
    //Time.timeScale = 0f;
}

public bool isAlive() => isRunning;

public bool IsRun()
{
    return isRunning;
}

public Vector3 GetPlayerScreenPos()
{
    Vector3 playerScreenPos =
Camera.main.WorldToScreenPoint(transform.position);
    return playerScreenPos;
}
}

```

Б.8 Текст файла PlayerVisual.cs

```

using UnityEngine;

public class PlayerVisual : MonoBehaviour
{
    private Animator _animator;
    private SpriteRenderer _spriteRenderer;
}

```

```

private const string IS_RUN = "IsRun";
private const string IS_DIE = "Die";

private void Awake()
{
    _animator = GetComponent<Animator>();
    _spriteRenderer = GetComponent<SpriteRenderer>();
}

private void Start()
{
    Player.Instance.OnPlayerDeath += Player_OnPlayerDeath;
}

private void Player_OnPlayerDeath(object sender, System.EventArgs e)
{
    _animator.SetBool(IS_DIE, true);
}

private void Update()
{
    _animator.SetBool(IS_RUN, Player.Instance.IsRun());
    if (Player.Instance.isAlive())
        PlayerFacingDirection();
}

private void PlayerFacingDirection()
{
    Vector3 mousePos = GameInput.Instance.GetMousePosition();
    Vector3 playerPosition = Player.Instance.GetPlayerScreenPos();
    if (mousePos.x < playerPosition.x)
    {
        _spriteRenderer.flipX = true;
    }
    else
    {
        _spriteRenderer.flipX = false;
    }
}
}

```

Б.9 Текст файла EnemyScript.cs

```

using UnityEngine;

public class EnemyScript : MonoBehaviour
{
    [SerializeField] public float speed = 1.5f;
    [SerializeField] public int maxHP;
    private int nowHP;

    private Rigidbody2D enemyRb;
    private GameObject player;
    private GameMaster scoreManager;
    private float scoreValue = 10f;

    private void OnTriggerEnter2D(Collider2D other)
    {

```

```

        if (other.CompareTag("Player"))
        {
            Player player = other.GetComponent<Player>();
            if (player != null)
            {
                player.TakeDamage(transform, 1); // урон 1
                Debug.Log(" Скелет ударил игрока!");
            }
        }
    }

    void Start()
    {
        enemyRb = GetComponent<Rigidbody2D>();
        player = GameObject.Find("Player");
        nowHP = maxHP;
        scoreManager = FindAnyObjectByType<GameMaster>();
    }

    public void hpUpdate(int dmg)
    {
        nowHP -= dmg;
        Debug.Log("Enemy take damage: " + dmg + " | nowHp: " + nowHP);

        if (nowHP <= 0)
        {
            nowHP = 0;
            Debug.Log("Skeleton died");
            scoreManager?.scoreUpdate(scoreValue);
            Destroy(gameObject);
        }
    }

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

        Vector3 playerVector = (player.transform.position -
transform.position).normalized;
        transform.position += playerVector * speed * Time.deltaTime;
    }
}

```

Б.10 Текст файла SkeletonVisual.cs

```

using UnityEngine;

public class SkeletonVisual : MonoBehaviour
{
    [Header("Настройки")]
    public Transform player;
    public float attackRange = 1.5f;

    [Header("Состояния")]
    public bool isDead = false;
    public bool justGotHit = false;
    public bool canAttack = true;

    private Animator animator;
    private float distanceToPlayer;
    private Vector3 originalScale;
}

```

```

private EnemyScript enemyScript;

void Start()
{
    enemyScript = GetComponent<EnemyScript>();
    animator = GetComponent<Animator>();
    originalScale = transform.localScale;

    if (player == null)
    {
        GameObject foundPlayer =
        GameObject.FindGameObjectWithTag("Player");
        if (foundPlayer != null)
            player = foundPlayer.transform;
    }

    void Update()
    {
        if (isDead)
        {
            animator.SetBool("IsDie", true);
            return;
        }

        if (player != null)
        {
            distanceToPlayer = Vector2.Distance(transform.position,
            player.position);
            float speed = distanceToPlayer > attackRange ? 1f : 0f;

            //параметры движения
            animator.SetBool("IsRunning", speed > 0.1f);
            animator.SetFloat("ChasingSpeed", speed);

            // Атака
            if (canAttack && distanceToPlayer <= attackRange)
            {
                animator.SetTrigger("Attack");
            }

            FlipTowardsPlayer();
        }

        // Урон
        if (justGotHit)
        {
            animator.SetTrigger("TakeHit");
            justGotHit = false; // Сброс, чтобы не зацикливалось
        }

        // Вызывается из другого скрипта при смерти
        public void Die()
        {
            isDead = true;
            animator.SetBool("IsDie", true);
        }

        // Получение урона
        public void TakeHit()
        {
            justGotHit = true;
        }
    }
}

```

```

private void FlipTowardsPlayer()
{
    if (player == null) return;

    float direction = player.position.x - transform.position.x;

    if (direction > 0)
    {
        // Смотрим вправо – нормальный масштаб
        transform.localScale = new Vector3(Mathf.Abs(originalScale.x),
originalScale.y, originalScale.z);
    }
    else if (direction < 0)
    {
        // Смотрим влево – зеркалим по X
        transform.localScale = new Vector3(-Mathf.Abs(originalScale.x),
originalScale.y, originalScale.z);
    }
}
}

```

Б.11 Текст файлу ActiveWeapon.cs

```

using UnityEngine;

public class ActiveWeapon : MonoBehaviour
{
    public static ActiveWeapon Instance { get; private set; }
    [SerializeField] private Catana katana;

    private void Awake()
    {
        Instance = this;
    }

    private void Update()
    {
        if (Player.Instance.isAlive())
            FollowMousePosition();
    }

    public Catana GetActiveWeapon()
    {
        return katana;
    }

    private void FollowMousePosition()
    {
        Vector3 mousePos = GameInput.Instance.GetMousePosition();
        Vector3 playerPosition = Player.Instance.GetPlayerScreenPos();
        if (mousePos.x < playerPosition.x)
        {
            transform.rotation = Quaternion.Euler(0, 180, 0);
        }
        else
        {
            transform.rotation = Quaternion.Euler(0, 0, 0);
        }
    }
}

```

```
}
```

Б.12 Текст файлу Catana.cs

```
using System;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.EventSystems;

public class Catana : MonoBehaviour
{
    public PolygonCollider2D PolygonCollider2D;

    public event EventHandler OnCatanaAttack;

    [SerializeField] private int damageAmount = 5;

    public void Awake()
    {
        PolygonCollider2D = GetComponent<PolygonCollider2D>();
    }

    private void Start()
    {
        AttackColliderTurnOff();
    }

    public void Attack()
    {
        AttackColliderTurnOn();
        OnCatanaAttack?.Invoke(this, EventArgs.Empty);
    }

    public void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.transform.TryGetComponent(out EnemyScript enemyScript))
        {
            enemyScript.hpUpdate(damageAmount);
            Debug.Log("Урон нанесен");
        }
        else
        {
            Debug.Log("Урона нет");
        }
    }

    public void AttackColliderTurnOff()
    {
        PolygonCollider2D.enabled = false;
    }

    private void AttackColliderTurnOn()
    {
        PolygonCollider2D.enabled = true;
    }
}
```

Б.13 Текст файла CatanaVisual.cs

```
using UnityEngine;

public class CatanaVisual : MonoBehaviour
{
    private Animator animator;

    [SerializeField] private Catana katana;

    private const string ATTACK = "Attack";

    private void Awake()
    {
        animator = GetComponent<Animator>();
    }

    private void Start()
    {
        katana.OnCatanaAttack += Katana_OnCatanaAttack;
    }

    private void Katana_OnCatanaAttack(object sender, System.EventArgs e)
    {
        animator.SetTrigger(ATTACK);
    }

    public void TriggerEndAttack()
    {
        katana.AttackColliderTurnOff();
    }
}
```

ДОДАТОК В
Слайди презентації

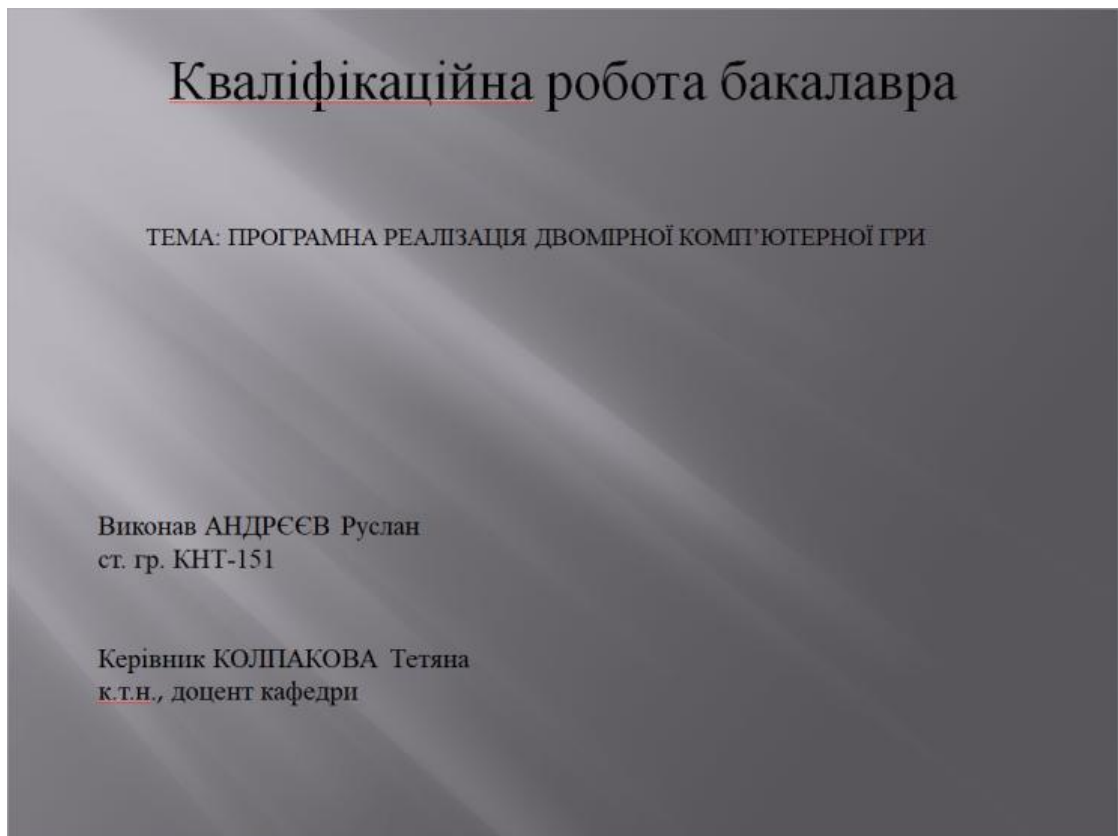


Рисунок В.1 – Титульний слайд

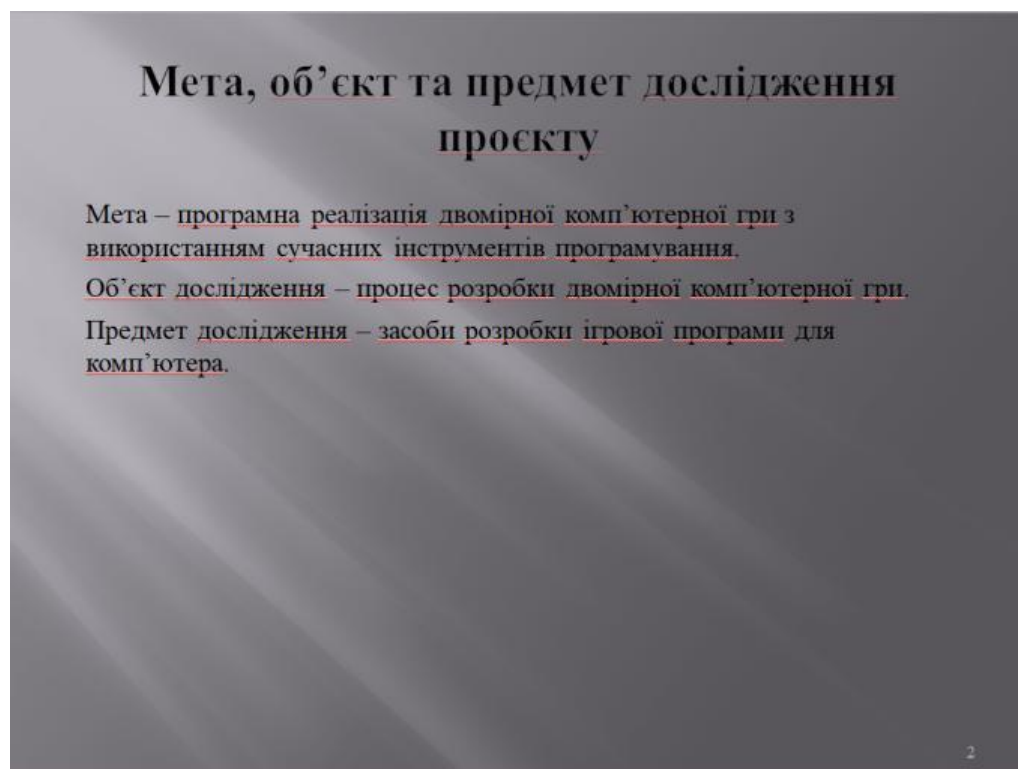


Рисунок В.2 – Мета, об'єкт та предмет дослідження проекту

Завдання до роботи

- ▣ Визначити потреби і критерії до програмного забезпечення;
- ▣ Спроектувати логічну структуру гри;
- ▣ Розробити дизайн користувацького інтерфейсу;
- ▣ Підібрати технологічні засоби для реалізації проєкту;
- ▣ Створити ігрової сцени та функціональні модулі;
- ▣ Розробити користувацький посібник.

3

Рисунок В.3 – Завдання до роботи

Ключові риси жанру RogueLike

Риси	Опис
Процедурна генерація рівнів	кожне проходження гри створюється динамічно, що забезпечує унікальний досвід при кожному запуску.
Постійна смерть (permadeath)	смерть персонажа веде до повного перезапуску гри, що підвищує складність і стратегічність.
Відсутність збережень	збереження прогресу, як правило, неможливе; прогрес — це навички гравця, а не накопичення ресурсів.
<u>Покрокова</u> або реальна <u>бойова</u> система	у класичних roguelike — покрокова, у сучасних — часто реального часу з <u>елементами bullet hell</u> .
Складні підземелля	зазвичай багато поверхів, коридори, кімнати, вороги і пастки.
Ресурси з обмеженням	обмежена кількість життя, боєприпасів, предметів, що стимулює тактичне мислення.
Сильний елемент випадковості	не лише в рівнях, а й у предметах, ворогах, подіях.
Різноманітність предметів та апгрейдів	гравець знаходить нове спорядження чи здібності у випадковому порядку
Повторюваність ігрового процесу (replayability)	гра орієнтована на багаторазове проходження.
<u>Мінімалістична</u> сюжетна подача	історія <u>грає</u> <u>другорядну</u> роль, на перше місце виходить геймплей.

4

Рисунок В.4 – Ключові риси жанру Roguelike

Порівняльна таблиця аналогів

Назва гри	Візуальна складова	Керування	Інтерфейс	Музикальний супровід
<u>Brotato</u>	4	5	4	5
Vampire Survivors	2	5	5	5
Enter the Gungeon	5	5	5	5

6

Рисунок В.5 – Порівняльна таблиця аналогів

Вибір рушія для розробки додатку

Порівняння рушіїв Unity, GameMaker Studio, Godot Engine

Характеристика	Unity	<u>GameMaker Studio</u>	Godot Engine
Детальне технічне супроводження	5	4	2
Повноцінні можливості з «коробки»	5	5	4
Гнучкість під різні пристрої та екрани	5	4	5
Розширена функціональність	4	3	5
Продуктивність та ефективність роботи	5	4	5
Здатність до зростання складності проєкту	5	3	5
Легкість освоєння новачками	5	4	2
Кросплатформність	5	5	5
Вільне застосування без ліцензійних витрат	5	4	5

7

Рисунок В.6 – Вибір рушія для розробки додатку

Порівняння середовищ розробки (IDE)

Характеристика	Microsoft Visual Studio	Jetbrains Rider
Підтримка кількох ОС	+	+
Доступність безплатного варіанту використання	+	-
Повноцінне та структуроване довідкове забезпечення	+	+
Сумісність з ігровим процесом Unity через панель керування	+	-
Повноцінна робота з об'єктною моделлю Unity	+	+
Підтримка інтеграції з системами відстеження змін у коді	+	+
Зручний та інтуїтивно зрозумілий інтерфейс	+	-
Висока швидкодія та продуктивність середовища	+	-
Можливість перегляду та налагодження стороннього скомпільованого коду	-	+

8

Рисунок В.7 – Порівняння середовищ розробки (IDE)

Архітектура системи

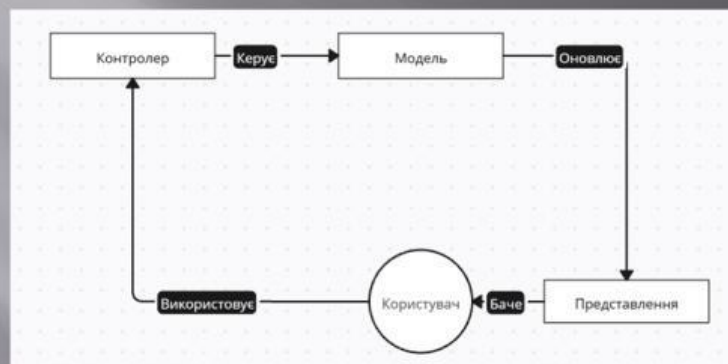
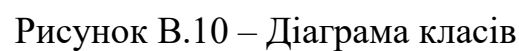
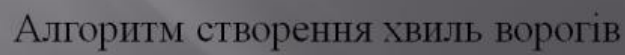


Схема роботи MVC

9

Рисунок В.8 – Архітектура системи



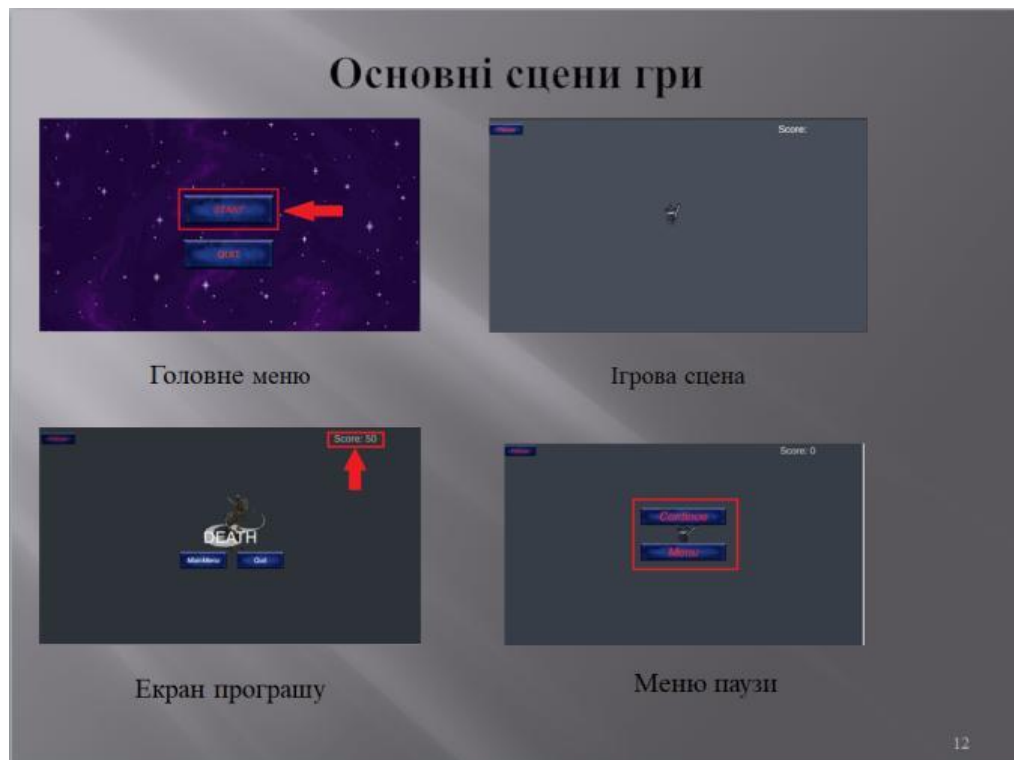


Рисунок В.11 – Основні сцени додатку

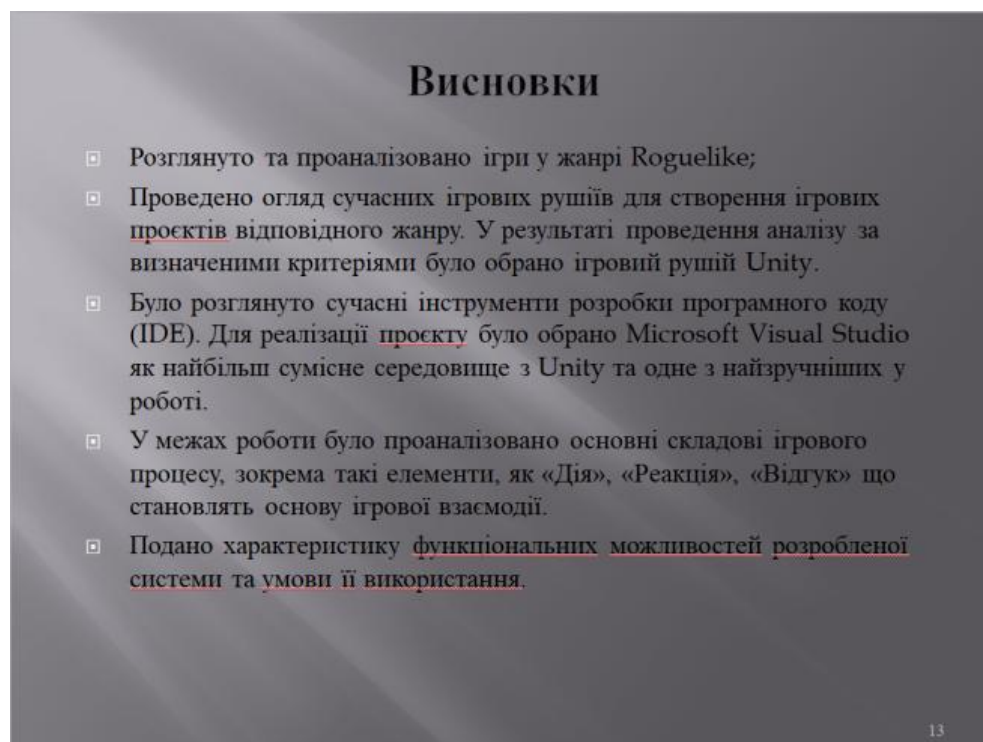


Рисунок В.12 – Висновки