СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування

# ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр

спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)

спеціалізація „Інженерія програмного забезпечення”

на тему „Розробка 2D гри платформера на базі unity”


Виконав: студент групи ІПЗ-20д ______ ( підпис ) О.П.Клименко (ініціали і прізвище)

Керівник ______ ( підпис ) В.О.Лифар (ініціали і прізвище)

Завідувач кафедри ______ ( підпис ) О.І. Захожай (ініціали і прізвище)

Рецензент________________


Київ – 2024

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет ) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування

Освітній ступінь бакалавр

спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”

(назва спеціалізації)

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри** ПМ,
____________Захожай О.І
“____” ____________2024 року

**З А В Д А Н Н Я**

НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Клименко Олег Павлович
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка 2D гри платформера на базі unity

Керівник роботи ___ Лифар Володимир Олексійович Доц., д.т.н ___________,
( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “06”травня 2024року №171/15.15-С

2. Строк подання студентом роботи 20 травня 2024

3. Вихідні дані до роботи Об'єктом даної роботи є процес розробки 2D гри платформера найефективнішими засобами

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Аналітичний огляд, з висвітленням наступних питань:що собою являють сучасні комп'ютерні ігри та їх класифікація, основні етапи розробки ігор. розгляд додатків-аналогів. Основна частина, в якої розыбрали: вібір додатків та інструментів для реалізації,проектування та реалізація проекту. Висновки. Перелік використаних джерел. код готового проекту

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)

**6. Консультанти розділів роботи**

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

7. Дата видачі завдання 30 березня 2024 року.

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів виконання кваліфікаційної випускної роботи | Строк виконання етапів | Примітка |
|---|---|---|---|
| 1 | Одержання завдання на виконання роботи | 30.03.24 | |
| 2 | Укладання і погодження з керівником плану і етапів виконання роботи | 06.04.24 | |
| 3 | Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі» | 13.04.24 | |
| 4 | Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху | 20.04.24 | |
| 5 | Укладання та тестування програмного продукту | 27.04.24 | |
| 6 | Укладання, оформлення та погодження пояснювальної записки з керівником | 04.05.24 | |
| 7 | Здача готової пояснювальної записки на кафедру | 25.05.24 | |
| 8 | Укладання доповіді і презентації | 30.05.24 | |

**Студент** ________ Клименко О.П
( підпис ) (ініціали і прізвище)

**Керівник роботи** ________ Лифар В.О.
( підпис ) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-20д Клименко О.П.

Науковий керівник

Доцент, д.т.н.________________________________ Лифар В.О.

Оцінка наукового керівника: ______________________

Рецензент ________________________________________________________

ПІБ, місто роботи, посада

Оцінка рецензента: ___________________________________

Кінцева оцінка за результатами захисту:

______________________________________________________________________

Голова ЕК
Професор кафедри ІТП
д.т.н. ____________________ Меняйленко О.С.

підпис

# РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Гра в жанрі 2D платформер на базі ігрового двигуна UNITY» містить: 54 основних сторінок, 39 додаткових, рисунки, 14 інформаційних джерел.

**Об’єкт дослідження** – гра в жанрі 2D платформер.

**Предмет дослідження** – технології розробки 2D ігор.

**Мета кваліфікаційної роботи** – розробити 2D гру у жанрі платформер з використанням визначених у ході дослідження найефективніших інструментів та програмного забезпечення.

**Методи дослідження** – аналіз, порівняння, обробка літературних джерел та проектування.

Розроблений прототип гри є цінним матеріалом, який може бути використаний для багатьох цілей. Він може служити основою для подальшого розвитку та вдосконалення гри. Крім того, прототип можна використовувати для демонстрації потенціалу та можливостей цієї розробки для широкої публіки.

Для створення комп’ютерної гри використані інструмента та забезпечення які були обрані у ході дослідження як найбільш ефективні в порівнянні з їх аналогами. Для створення візуальної частини використовувалися готові безкоштовні ассети та спрайти з магазину Unity та інших подібних безіменних ресурсів.

# ЗМІСТ

**ВСТУП** ..... **5**

**РОЗДІЛ 1**
**АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ** ..... **7**

1.1. Дослідження поняття комп’ютерних ігор ..... 7

1.2. Класифікація комп’ютерних ігор ..... 8

1.3. Етапи створення комп’ютерних ігор ..... 11

1.4. Огляд та аналіз додатків-аналогів ..... 12

**ВИСНОВКИ ДО РОЗДІЛУ 1** ..... **17**

**РОЗДІЛ 2**
**ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ** ..... **18**

2.1 Мова програмування C# ..... 18

2.2. Аналіз існуючих ігрових движків ..... 19

2.2.1. CryEngine ..... 20

2.2.2. Unity ..... 22

2.2.3. Unreal Engine ..... 24

2.3. Середовище розробки Microsoft Visual Studio ..... 26

**ВИСНОВКИ ДО РОЗДІЛУ 2** ..... **30**

**РОЗДІЛ 3**
**РЕАЛІЗАЦІЯ ПРОЕКТУ** ..... **31**

3.1. Концепція та сценарій гри ..... 31

3.2. Цільова аудиторія ..... 31

3.3. Графічне оформлення ..... 32

3.3.1. Спрайти головного персонажу ..... 33

3.4. Етап проектування гри ..... 36

3.4.1 Фізичні властивості об’єктів ..... 37

3.4.2. Рух об’єктів ..... 39

3.4.3. Анімація об’єктів ..... 43

3.4.4. Звукові ефекти ..... 46

3.4.5. Розробка меню початку, паузи та кінця гри ..... 47

3.5. Аналіз отриманого результату ..... 51

**ВИСНОВКИ ДО РОЗДІЛУ 3** ..... **53**

**ВИСНОВКИ РОБОТИ** ..... **55**

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** ..... **57**

**ДОДАТКИ** ..... **58**

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ** ..... **58**

**ПРОГРАМНИЙ КОД** ..... **59**

# ВСТУП

У сучасному світі комп’ютерні ігри стали не лише популярною формою розваг, але й важливим компонентом розвитку культури та технологій. Їх вплив на суспільство стає все більш помітним, оскільки вони відкривають нові можливості та заохочують інновації.

Відеоігри займають значну частину світового ринку розваг. Відеоігри стали доступнішими для широкого кола користувачів завдяки стрімкому розвитку комп’ютерних технологій, а особливо поширенню Інтернету, на широкому спектрі пристроїв, від стаціонарних ПК до смартфонів.

З кожним роком ігрова індустрія вдосконалюється за допомогою нових ідей і технологій. Ігри стали більше, ніж просто засіб розваги; вони також дозволяють людям співпрацювати, працювати разом і навіть змагатися один з одним.

Геймдизайнер та розробники постійно працюють над створенням захопливих ігор, які не лише допомагають гравцям розважатися, але й допомагають їм розвивати різні навички. Розробники ігор змушені використовувати найновіші технології для створення ще більш захопливих і інноваційних ігор, які мають значний вплив на культуру та суспільство в цілому завдяки постійному залучення до ігрового світу.

Крім того, важливо зазначити, що комп’ютерні ігри є не лише засобом для розваг, але й платформою для віртуального спілкування, створюючи міжнародні спільноти гравців, які працюють разом, змагаються та обмінюються досвідом. Таким чином, ігрова індустрія продовжує змінювати те, як ми розважаємося та спілкуємося, вносячи значний внесок у сучасну культуру.

**Актуальність** теми «Розробка 2D гри платформера на базі unity» ґрунтується на тому, що ігрова індустрія сьогодні перебуває в стані стрімкого зростання та розвитку. Комп’ютерні ігри стають все більш популярними серед різноманітних груп користувачів. Це викликає значну потребу в розробці високоякісних, захоплюючих ігор. Наведена кваліфікаційна робота відповідає поточним тенденціям розвитку ігрової індустрії та пропонує можливості для подальшого дослідження та розвитку в галузі розробки комп’ютерних ігор.

**Об’єкт дослідження** – 2D гра платформер

**Мета роботи** – створити гру 2D платформер на основі Unity. Ця гра може служити основою для наступних проектів, пов’язаних із цією темою, або початком нового 2D проекту в іншому жанрі.

**Основні завдання дослідження** визначені відповідно до поставленої мети роботи:

– Аналіз предметної області;

– Аналіз сучасного ринку ігор;

– Дослідження основних етапів розробки;

– Огляд програмних середовищ для розробки;

– Створення концепції гри;

– Реалізація проекту з використанням даних цього дослідження ;

– Аналіз отриманого результату;

**Методи досягнення мети**:

– обробка літературних джерел;

– порівняльний аналіз;

– проектування.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1. Дослідження поняття комп’ютерних ігор

Комп'ютерна гра є програмним забезпеченням, розробленим для надання ігрового досвіду користувачам. Вона складається з багатьох компонентів, які створюють віртуальний світ для гравців, включаючи графіку, геймплей, звуковий супровід та інші компоненти. Комп’ютерні ігри включають широкий спектр жанрів і форматів, від екшн стратегій до пригодницьких симуляторів. Вони також доступні для різних платформ, включаючи мобільні пристрої, ігрові консолі та персональні комп’ютери.

Комп’ютерні ігри, як і традиційні ігри, є симуляцією реальності, в якій гравець взаємодіє з віртуальним світом. З іншого боку, комп’ютерні ігри відрізняються від традиційних ігор обмеженнями щодо простору, часу та можливостей. Наприклад, у комп’ютерних іграх гравець може бути обмежений простором, який визначається програмним кодом гри, а також часом, який контролюється часом гри або обмеженнями розробника.

Відмінність, яка відрізняє комп’ютерні ігри від інших, полягає в тому, що візуальні ефекти цих ігор створюються розробниками за допомогою їх творчої роботи, а не гравця. Використовуючи інтерфейс гри, гравець взаємодіє з віртуальним світом, а не безпосередньо з реальними об’єктами.

З часом з’явилися випадки, коли відомі ігрові серії випускають додаткові матеріали, які розширюють світ гри. Додаткові рівні, історії персонажів або навіть різноманітні DLC.

Крім того, ігри можуть бути використані як навчальний матеріал або як інструмент в наукових дослідженнях. Такі ігри можуть надати інтерактивний спосіб вивчення певної теми або моделювати складні процеси, які потрібно досліджувати.

Отже, комп'ютерні ігри, незалежно від форми та призначення, стають важливим елементом культури та суспільства, впливаючи на розваги, навчання та наукові дослідження.

### 1.2. Класифікація комп'ютерних ігор

Аналогічно до літератури і музики, відеоігри можна класифікувати за жанрами, які базуються на загальних аспектах ігрового процесу та цілях, які він має досягти. З огляду на величезний вибір комп'ютерних ігор, їх можна групувати за різними критеріями, такими як завдання, тип геймплею чи атмосфера.

Способом організації різноманітності ігрового досвіду є класифікація відеоігор на жанри, яка може включати різні піджанри. Гра може одночасно належати до кількох жанрів, оскільки вона може поєднувати різні елементи ігрового процесу.

У процесі розвитку комп'ютерних ігор виникла прийнятна класифікація, яка включає у себе дуже багату кількість жанрів, але деякі з них можна вважати найбільш популярними:

– Adventure (пригода) - жанр комп'ютерних ігор, у якому головним пріоритетом є пошук, розв'язання головоломок і розвиток історії. Щоб просуватися по сюжету, гравці повинні взаємодіяти з ігровим світом, знаходити предмети, вирішувати завдання та спілкуватися з персонажами;

– RPG (RolePlaying Game) - це жанр комп'ютерних ігор, у яких гравець грає за вигаданого персонажа та бачить ігровий світ через його очі. Гравець створює персонажа, приймає рішення, які впливають на розвиток ігри, і виконує завдання відповідно до вигаданого сценарію;

– Arcade (аркада) - це тип комп'ютерної гри, яка використовує прості, швидкі та динамічні механіки. Зазвичай гравці повинні швидко реагувати на події в грі за допомогою простих керувань;

– Shooter (Шутер) - це жанр комп'ютерної гри, в якій гравець керує персонажем, який бореться з ворогами, зазвичай за допомогою вогнепальної

зброї. Основною метою є перемога над суперниками та виживання в ігровому середовищі;

– Educational (навчальна гра) - це жанр комп'ютерної гри, мета якої полягає в тому, щоб гравці набули певних навичок або знань. Цей тип ігор може включати інтерактивні завдання, головоломки, тести та інші активності, які мають на меті розвивати розумові або практичні навички гравців. Вони можуть охоплювати різні теми, від математики та мов до наукових дисциплін або управлінських навичок;

– Strategy (стратегія) - це жанр комп'ютерної гри, в якій гравець приймає стратегічні рішення, щоб досягти цілей. Для успішного виконання завдань у цьому жанрі необхідно планувати дії, керувати ресурсами та взаємодіяти з ігровим середовищем. Головна мета полягає в розробці та реалізації стратегічного плану, щоб перемогти противника або досягти ігрової мети;

– Simulator (симулятор) - це жанр комп'ютерних ігор, спрямованих на імітацію реальних або уявних подій. Гравець має можливість відчути себе персонажем або керувати певними процесами чи об'єктами в ігровому середовищі. Симулятори можуть мати різні теми, від авіасимуляторів і автомобільних гонок до симуляцій ведення господарства або навіть космічних польотів. Симулятори мають на мету дати гравцям можливість відчути реалістичність і імерсивність ігрового досвіду у вибраній сфері;

– Platformer (Платформер) - це жанр комп'ютерної гри, в якій основною метою є пересування головного персонажа по різних рівнях чи платформах. Зазвичай гравець керує персонажем, який може стрибати, бігти та виконувати інші дії, щоб подолати перешкоди, збирати предмети та досягати цілей. У цьому жанрі можна знайти широкий спектр ігрових елементів, включаючи головоломки та бойові сцени.

Основним критерієм поділу жанрів у іграх є те, що гравець робить найчастіше під час гри. Звичайно, ігри можна класифікувати в різні групи залежно від того, який основний акцент робиться на певних типах дій. Загалом

ігри поділяються на три основні категорії: ігри контролю, ігри дій і ігри інформації(Таблиця 1.1).

| | Ігри інформації | Ігри дії | Ігри контролю |
|---|---|---|---|
| Опис | Ігри, спрямовані на передачу гравцеві будь якої інформації. | Ігри, де гравець може взаємодіяти з ігровим середовищем та виконувати дії й приймати рішення, які впливають на події в грі. | Ігри, які дозволяють гравцю контролювати обставини або обмежувати його вибір дій |
| Мета | Передати корисну або цікаву для думок інформацію гравцю | Розвивати навички взаємодії зі світом гри | Розвивати або перевіряти навички управління та контролю |
| Навички | Логічне мислення, увага, концентрація, стратегічне планування | Реакція, координація рухів, прийняття рішень у різних ситуаціях | Логічне мислення, увага, концентрація, стратегічне планування |
| Цільова аудиторія | Люди, які цікавляться певною темою та бажають дізнатися більше про неї | Люди, які люблять діяти та взаємодіяти зі світом гри | Люди, які люблять контролювати ситуацію та вирішувати головоломки |
| Приклади ігор | Симулятори, енциклопедії, квести,рольові ігри, пригоди, візуальні новели. | Екшн-ігри, гонки, ігри-пригоди, платформери. | Всі різновиди стратегій, економічні ігри, варгейми, шахи, головоломки. |

Таблиця 1.1 – Жанрова класифікація комп’ютерних ігор

Важливо додати, що розвага гравця не завжди є метою гри.

Залежно від кількості гравців усі ігри можна класифікувати як одиночні, так і мультиплеєрні. За допомогою цієї системи класифікації можна визначити режими, які будуть доступні в грі.

Одиночна гра(singleplayer) - це тип гри, в якій гравець грає самостійно, без участі інших гравців. Зазвичай гравцеві протистоїть комп'ютерний інтелект або він працює сам на досягнення кінцевої мети гри, такої як проходження

рівнів, збір ресурсів або підвищення своїх навичок. Часто ці цілі можуть комбінуватися між собою.

Мультиплеєр - це режим гри, який призначений для одночасної гри більше ніж однієї людини. Багато ігор комбінують одиночну гру та мультиплеєр, дозволяючи гравцям вибирати між грою в одиночному режимі або з друзями у мережі.

Візуально комп’ютерні ігри можна розділити на наступні категорії:

– 2D ігри — це ігри, в яких всі елементи та об’єкти, включаючи спрайти та фонові елементи, зображені у двовимірній графіці;

– 3D ігри - це ігри, в яких всі елементи та об’єкти відображаються в тривимірних моделях у. Це надає іграм реалістичний вигляд і дозволяє гравцям сприймати гру як більш просторову та іммерсивну;

– Текстові ігри - це ігри, у яких використовується текстовий інтерфейс для взаємодії з гравцем. Зазвичай вони використовуються для розповіді історії, вирішення головоломок і прийняття рішень, зазвичай гравець вводить команди або вибирає опції з текстового меню.

Типи ігрових платформ:

– ПК;

– ігрові приставки і консолі;

– мобільні телефони.

### 1.3. Етапи створення комп’ютерних ігор

Розробник повинен пройти всі етапи від ідеї до розробки, щоб створити якісну та успішну комп'ютерну гру.

Процес створення комп’ютерних ігор складається з наступних етапів:

1. Ідея: розробка ідеї гри, яка включає жанр, сюжет, основні механіки та характеристики.

2. Проектування: створення повного опису гри, який включає графіку, звук, інтерфейс користувача, механіку, архітектуру програмного забезпечення та інші компоненти.

3. Розробка: створення графічних елементів, анімації, звуку, програмного коду та інших елементів гри.

4. Тестування: перевірка відповідності стандартам якості, оцінка геймплею та випробування помилок.

5. Випуск: гра буде випущена на ринку або доступна для завантаження в Інтернет-магазині.

6. Підтримка та оновлення: Гравці отримують технічну підтримку, виправлення помилок, патчі та оновлення, щоб покращити гру.

### 1.4. Огляд та аналіз додатків-аналогів

Створення власної гри передбачає дослідження та вивчення подібних комп'ютерних ігор жанру платформер. Це дозволяє оцінити наявні ігри у цьому жанрі, оцінити їхні характеристики та успіх, а також визначити потенційні недоліки або прогалини, які можуть бути усунені у власному проекті.

Платформери відомі своїм впливом на багатьох гравців. Для когось Super Mario Bros. стала першою грою, хтось обожнював швидкість Соніка, а інші випробовували свої навички в Crash Bandicoot. Цей жанр все ще популярний, і на ПК є чим вибирати. У ході використання кваліфікаційної роботи розглянемо деякі проекти.

Першою для аналізу обрано гру Rain world(Рис.1.2)

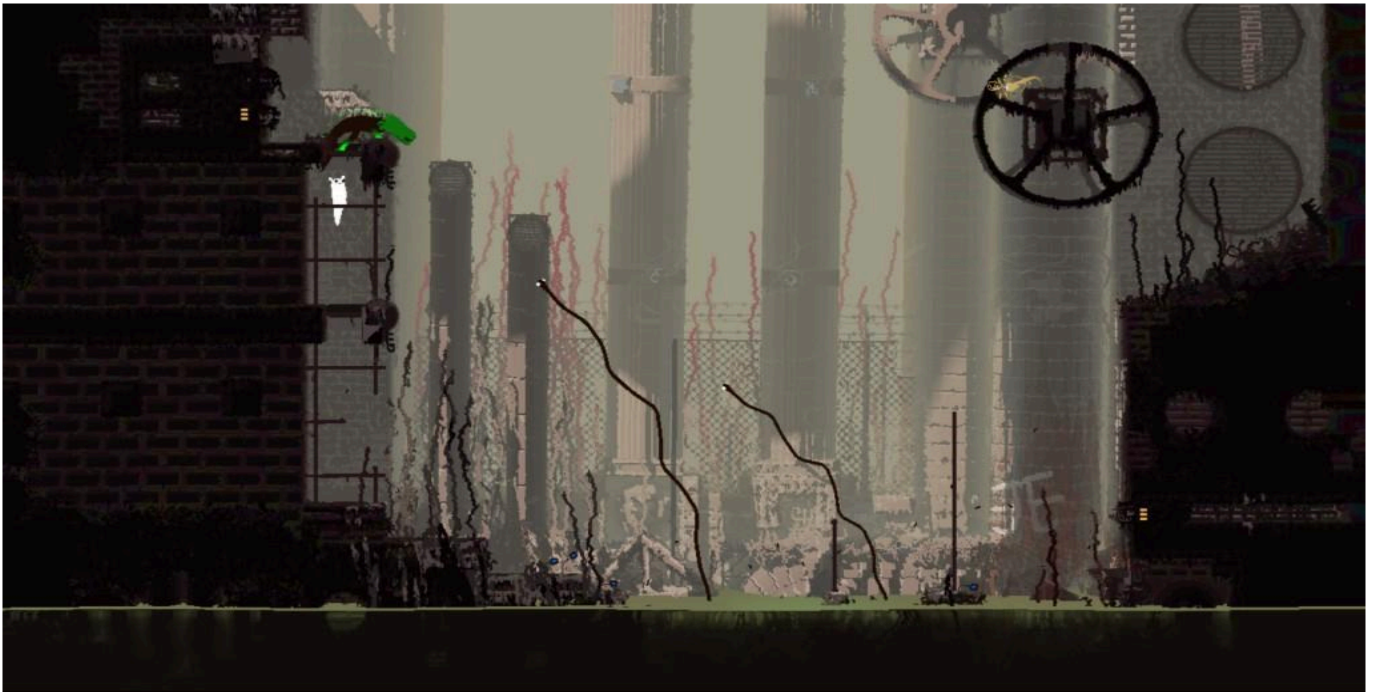

Рис. 1.2. Знімок екрана з гри Rain world

"Rain World" - це інді-гра в жанрі пригодницького платформера, розроблена студією Videocult та випущена Adult Swim Games у 2017 році. Гра відома своєю високою складністю, унікальним візуальним стилем і атмосферою виживання в постапокаліптичному світі.

**Ігровий процес:**

"Rain World" занурює гравців у ролі маленького істоти, відомої як Slugcat, що намагається вижити в небезпечному середовищі після катастрофічної події. Основні елементи ігрового процесу включають:

**Основні аспекти ігрового процесу:**

Виживання: Гравець повинен знайти їжу і уникати хижаків, щоб вижити. Ресурси обмежені, і правильне планування є ключем до виживання.

Дослідження: Світ "Rain World" великий і нелінійний, що спонукає до дослідження і пошуку шляхів через різні локації.

Цикли дощу: Гра має цикли дощу, які змінюють ігровий процес. Дощі дуже небезпечні, тому гравець повинен знайти безпечне місце до їх початку.

Складні вороги: Світ наповнений різними хижаками, кожен з яких має свої поведінкові патерни, що додає стратегічного елементу в ухиленні і боротьбі з ними.

**Візуальний стиль та музика:**

"Rain World" має унікальний візуальний стиль, який поєднує піксельну графіку з атмосферними фонами, створюючи похмурий та загадковий світ. Анімація рухів персонажів та істот ретельно пророблена, додаючи грі реалістичності та “живості”. Саундтрек, підкреслює напруженість і атмосферність світу гри.

**Сюжет:**

Сюжет "Rain World" мінімалістичний і подається через візуальні підказки та оточення. Гравець кіт-слимак, який втратив свою родину під час катастрофічної події і тепер намагається знайти їх у цьому небезпечному світі. Історія подається нелінійно, і гравець самостійно збирає фрагменти історії, досліджуючи світ.

**Критика та успіх**:

"Rain World" отримала змішані відгуки від критиків. Позитивно оцінювались її унікальний стиль, атмосферність та складність. Однак, деякі рецензенти критикували високу складність і часом фруструючий геймплей. Незважаючи на це, гра здобула культовий статус серед гравців, які цінують виклик і унікальність.

**Висновок:**

"Rain World" - це гра, яка виділяється своєю атмосферою, унікальним художнім стилем і викликом, який вона ставить перед гравцем. Вона показує, як інді-гра може запропонувати унікальний досвід, який відрізняється від традиційних комерційних проектів.

Наступною грою для аналізу  обрано Hollow Knight(Рис. 1.3).

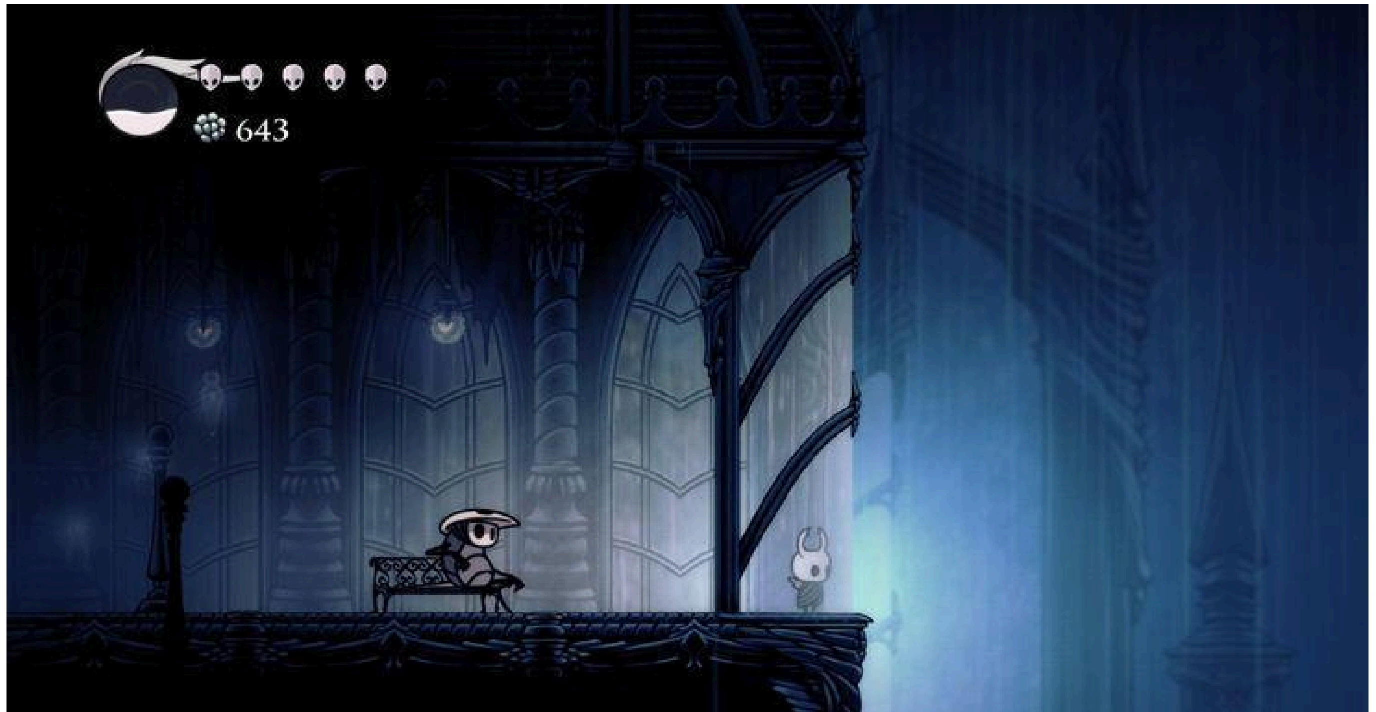


Рис. 1.3. Знімок екрана з гри Hollow Knight

"Hollow Knight" - це інді-гра у жанрі метроїдванія, розроблена та видана студією Team Cherry. Випущена у 2017 році, гра швидко здобула популярність завдяки своїй глибокій ігровій механіці, атмосферному світу та приголомшливому візуальному стилю.

**Ігровий процес**

"Hollow Knight" пропонує гравцям дослідити великий та відкритий світ, наповнений різноманітними локаціями, ворогами та секретами. Головний герой, Безіменний лицар, мандрує підземним королівством Галлоунест, досліджуючи його та борючись з різними істотами.

**Основні аспекти ігрового процесу:**

Дослідження: Гра ставить великий акцент на дослідженні. Кожна нова область пропонує унікальні виклики та головоломки.

Бої: Гравці стикаються з багатьма ворогами та босами, кожен з яких вимагає індивідуального підходу.

Розвиток персонажа: Гравці можуть покращувати свого героя, знаходячи нові здатності та предмети.

Система амулетів: Гравці можуть використовувати амулети, які надають різні бонуси та здатності.

**Візуальний стиль та музика**

"Hollow Knight" вирізняється своїм унікальним візуальним стилем, що поєднує в собі казкові та готичні елементи. Гра має ручну анімацію, яка додає їй особливої чарівності. Саундтрек, написаний Крістофером Ларкіним, ідеально доповнює атмосферу гри, підкреслюючи як спокійні, так і напружені моменти.

**Сюжет**

Сюжет гри розповідає про падіння королівства Галлоунест та спроби головного героя розкрити його таємниці. Хоча історія не подається лінійно, гравці можуть збирати фрагменти інформації з розмов з NPC, предметів та описів локацій, що створює глибоке та захопливе наративне тло.

**Критика та успіх**

"Hollow Knight" отримала високі оцінки від критиків та гравців за її геймплей, атмосферу та художнє виконання. Вона також здобула кілька нагород і стала однією з найпопулярніших інді-ігор свого часу. Гра має високий рівень складності, що приваблює гравців, які шукають викликів.

**Висновок**

"Hollow Knight" - це яскравий приклад того, як інді-гра може стати справжнім хітом, поєднуючи чудовий геймплей, унікальний стиль та захоплюючий світ. Гра залишається популярною і сьогодні, надихаючи інших розробників та тішачи своїх фанатів.

## ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі було проведено аналіз предметної області, пов'язаної з комп'ютерними іграми. Вивчення концепції комп’ютерних ігор показало, наскільки вони важливі та цінні в сучасному світі. Класифікація комп’ютерних ігор дозволила розподілити їх на різні категорії залежно від їх основних характеристик і жанрів.

Розглянуто основні етапи створення комп’ютерних ігор. Встановлено, що процес розробки гри складається з наступних етапів: аналізу, планування, проектування, розробки, тестування та релізу. Досліджено детально кожен із цих етапів і визначено, як вони пов’язані один з одним і як вони необхідні для успішної розробки гри. У результаті цього аналізу були визначені основні етапи та процедури, які необхідно враховувати під час створення комп’ютерних ігор.

Огляд і аналіз додатків-аналогів були проведені з метою виявлення подібних програмних продуктів на ринку та визначення їх переваг і недоліків. Це дозволило зробити висновки про потенційні можливості для подальшого дослідження та розробки, а також отримати уявлення про поточні тенденції та інновації в галузі комп’ютерних ігор. Згідно з оглядом, головним пріоритетом для платформерів є зручність керування, інтерфейсу та дизайну. У цьому жанрі ігри зосереджені на ігровому процесі, але й не позбавлені сюжету та гарного візуалу та атмосфери.

# РОЗДІЛ 2
# ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

## 2.1 Мова програмування C#

Під час розробки програмного забезпечення вибір мови програмування визначається вимогами проекту. Однією з популярних мов програмування є C# (C-Sharp), розроблена компанією Microsoft. C# є об’єктно-орієнтованою мовою програмування, що має в своєму арсеналі потужні функції та можливості, що дозволяють розробникам створювати різноманітне програмне забезпечення.

Хоча C# належить до сімейства мов C/C++, він має певні особливості. Вона ґрунтується на об’єктно-орієнтованому підході, який дозволяє розробникам моделювати взаємодію та фізичні об’єкти, які існують у реальному світі. Розробка та розуміння коду полегшуються за допомогою чистого синтаксису мови, а автоматичне управління пам’яттю полегшує роботу з пам’яттю та зменшує ймовірність помилок.

Синтаксис C#, який є простим і зрозумілим, є однією з найбільш привабливих характеристик мови, що робить його простішим для новачків у програмуванні та сприяє зменшенню кількості помилок у коді. Крім того, підтримка автоматичного управління пам'яттю в C# покращує використання ресурсів і знижує ймовірність утечек пам'яти.

Мова програмування C# пропонує можливість кросплатформеного розвитку завдяки проекту .NET Core. Ця характеристика дозволяє запускати код, написаний на C#, на різних операційних системах без необхідності значних модифікацій вихідного коду. Такий підхід відкриває перспективи для створення кросплатформенних додатків, що можуть функціонувати на різних операційних системах, таких як Windows, Linux та macOS,без додаткового зусиль. Також, C#

може використовуватися для розробки мобільних додатків для операційних систем iOS та Android за допомогою фреймворка Xamarin.

Розробка в галузі ігор є ще одним напрямом, де мова програмування C# виявляється особливо ефективною. Платформа розробки ігор Unity, яка є однією з провідних у цій галузі, широко використовує мову C# як основну. Це дозволяє створювати розробникам складні та цікаві ігри, використовуючи різноманіття функціональних можливостей та бібліотек, доступних у мові C#.

Поміж стандартних функціональних можливостей та бібліотек мови C#, існує обширна громада розробників, яка активно займається розширенням цієї мови та створенням власних наборів інструментів. Цей процес сприяє інтенсивному обміну досвідом та інформацією, а також дає доступ до значної кількості готових рішень та прикладів коду, що значно полегшує процес розробки програмного забезпечення.

Мова C# є універсальним інструментом для розробки програмного забезпечення в багатьох галузях, включаючи ігровий сектор, завдяки своїй високій абстракції, зрозумілому синтаксису та широкому діапазону функцій. Розробники можуть створювати високоякісні ігри та програми з меншими затратами та часом, використовуючи мову C#. Це прискорює процес розробки та допомагає досягти цілей.

### 2.2. Аналіз існуючих ігрових движків

Ігрові движки є важливими інструментами для розробки ігор, які надають розробникам можливість ефективно використовувати свої знання та ресурси для створення як 2D, так і 3D ігор. Ці засоби надають широкий спектр функцій, які спрощують завдання, такі як програмування геймплею, робота з графікою, фізикою, звуком, штучним інтелектом та іншими аспектами гри.

Ігрові движки надають базову структуру, на яку розробники можуть покластися для створення власної гри. Вони можуть використовувати різні функції та інструменти, включаючи введення, рендеринг, сценарії, виявлення зіткнень і штучний інтелект, серед інших.

Ігрові двигуни дозволяють розробникам здійснювати ефективне використання вже готових компонентів, замість того, щоб витрачати час на їх створення з нуля. Це дозволяє зосередитися на унікальних аспектах гри, таких як дизайн персонажів, текстури, фізика, геймплей та інші компоненти. Використання ігрових двигунів зменшує витрати на розробку ігор, гарантує якість готового продукту та прискорює процес розробки.

При виборі ігрового движка розробники ігор мають ретельно враховувати свої потреби і мету проекту, а також оцінювати доступні варіанти і їх можливості. Кожен ігровий движок має свої властивості, переваги й обмеження, тому важливо знайти той, що належним чином відповідає вимогам проекту.

Для аналізу були взяті одні з найпопулярніших ігрові двигуни, а саме: CryEngine, Unity та Unreal Engine.

### 2.2.1. CryEngine

CryEngine є ігровим рушієм, розробленим німецькою компанією Crytek(Рис. 2.1). Вперше він був використаний у серії відеогр Far Cry.

Цей рушій відомий своєю вражаючою графікою та розширеними функціями, які дають розробникам ігор багато можливостей для творчості. Він має потужні редактори фізики, рівнів, анімації та штучний інтелект, що дозволяє втілювати складні механізми та взаємодію у своїх іграх. Розширення функцій і можливостей рушія CryEngine дозволяє розробникам використовувати сучасну 3D-графіку та масштабовані обчислення. Цей рушій має велику кількість вбудованих інструментів, тому ви можете створювати ігри самостійно, не використовуючи додаткових програм. Досі ніхто не зміг повністю використовувати всі можливості CryEngine, що робить його чудовим інструментом для розробки ігор нового покоління. Параметрична скелетна анімація, процедурне деформування руху та система індивідуальних персонажів є частиною вбудованої системи анімації.

Особливості програми CryEngine:

– Можливість створення онлайн ігор;

– Підтримка платформ для ПК, Xbox і PlayStation;

– Використання передових графічних технологій;

– Імпорт моделей і текстур з графічних редакторів Maya і 3ds Max;

– Програмування місій у вільному форматі;

– Можливості оптимізації гри під GPU-рендеринг;

– Зручний режим тестування ігрового процесу і меню ігор;

– Використання передового штучного інтелекту;

– Створення великих закритих і відкритих локацій;

– Наявність всіх необхідних інструментів для повноцінної розробки без використання сторонніх програм;

– Користувальницький інтерфейс рушія реалізований англійською мовою.

Переваги:

– Графічна якість: CryEngine славиться своєю вражаючою графікою та реалістичними візуальними ефектами. Він забезпечує високу деталізацію, реалістичне освітлення і фотореалістичні текстури, що дозволяє створювати неймовірно реалістичні ігрові світи.

– Фізична модель: CryEngine має потужну фізичну модель, яка реалістично відтворює рухи об'єктів, взаємодію об'єктів з навколишнім середовищем та передає різні фізичні ефекти.

– Інструменти розробки: CryEngine надає розширений набір інструментів для розробки ігрових проектів. Він має зручний візуальний редактор, що дозволяє швидше створювати та налаштовувати ігрові об'єкти, механіки та інтерактивні елементи.

Недоліки:

– Вимоги до апаратного забезпечення: Один з недоліків CryENGINE полягає у високих вимогах до обчислювальної потужності комп'ютера, що може

ускладнити доступність розробки для користувачів з менш потужними системами.

– Складність в освоєнні: CryENGINE має складну структуру та інтерфейс, що може вимагати часу та зусиль для освоєння. Необхідний досвід у розробці ігор або вивчення документації та посібників для ефективного використання цього движка.

– Ліцензійні витрати: Використання CryENGINE пов'язане з ліцензійними витратами, особливо для комерційних проектів, що може стати чинником обмеження для незалежних розробників або студій з обмеженими фінансовими ресурсами.

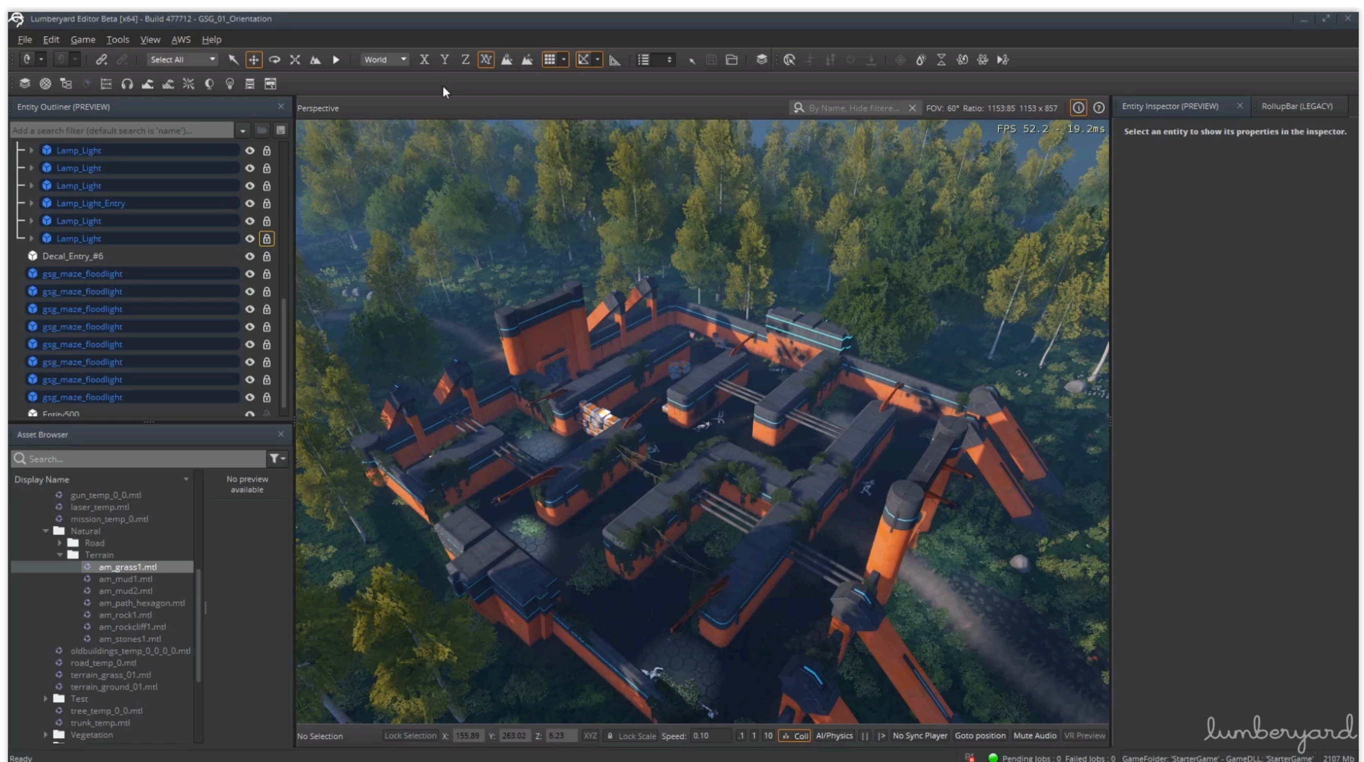


Рис. 2.1. Інтерфейс ігрового двигуна CryEngine

### 2.2.2. Unity

Unity є інтегрованим середовищем розробки та ігровим двигуном, яке використовується для створення ігор, віртуальної реальності (VR), доповненої реальності (AR) і інтерактивних додатків. Це складне рішення надає розробникам широкий набір інструментів і функцій, щоб допомогти їм створити професійні ігри з різноманітними геймплейними механіками та захопливими візуальними ефектами.(Рис. 2.2).

Мета ігрового движка полягає в тому, щоб розробники ігор могли використовувати його для створення ігор будь-якого рівня складності, навіть для початківців. Крім того, Unity Technologies стала одним із найпотужніших ігрових двигунів, зосереджуючись на розвитку 3D у реальному часі, розширюючи своє покриття в інші сфери.

Одними з найважливіших елементів є гнучкість і широта. Двигун має велику кількість плагінів і розширень, що дозволяє розробникам використовувати більш спеціалізовані інструменти для своїх проектів.

Інтерфейс Unity є інтуїтивно зрозумілим та добре організованим, що сприяє роботі розробників і зменшує час, необхідний для освоєння інструментів та процесу розробки. Крім того, Unity має потужні функції для роботи з графікою, анімацією, фізикою, штучним інтелектом та звуком, що дозволяє створювати вражаючі візуальні та аудіо елементи для ігор.

Переваги:

– Багатоплатформність - Unity надає можливість розробляти ігри для різних платформ, включаючи комп'ютери, мобільні пристрої та консолі.

– Розширюваність - У Unity існує значна кількість плагінів та розширень, що дозволяють розширити функціонал двигуна та використовувати спеціалізовані інструменти.

– Інтуїтивний інтерфейс - Unity пропонує добре організований та легкий у використанні інтерфейс, що спрощує процес розробки.

– Потужність - Unity забезпечує потужні функції для роботи з графікою, анімацією, фізикою, штучним інтелектом та звуком.

Недоліки:

– Високі витрати - Деякі функції та додаткові плагіни Unity можуть призвести до додаткових фінансових витрат.

– Обмеження продуктивності - У певних випадках Unity може бути менш продуктивним порівняно з іншими ігровими двигунами при обробці великих обсягів даних або складних графічних ефектів.

– Залежність від розробників - Unity постійно оновлюється та розвивається, що може вимагати від розробників постійного оновлення та ознайомлення з останніми версіями та функціональністю.

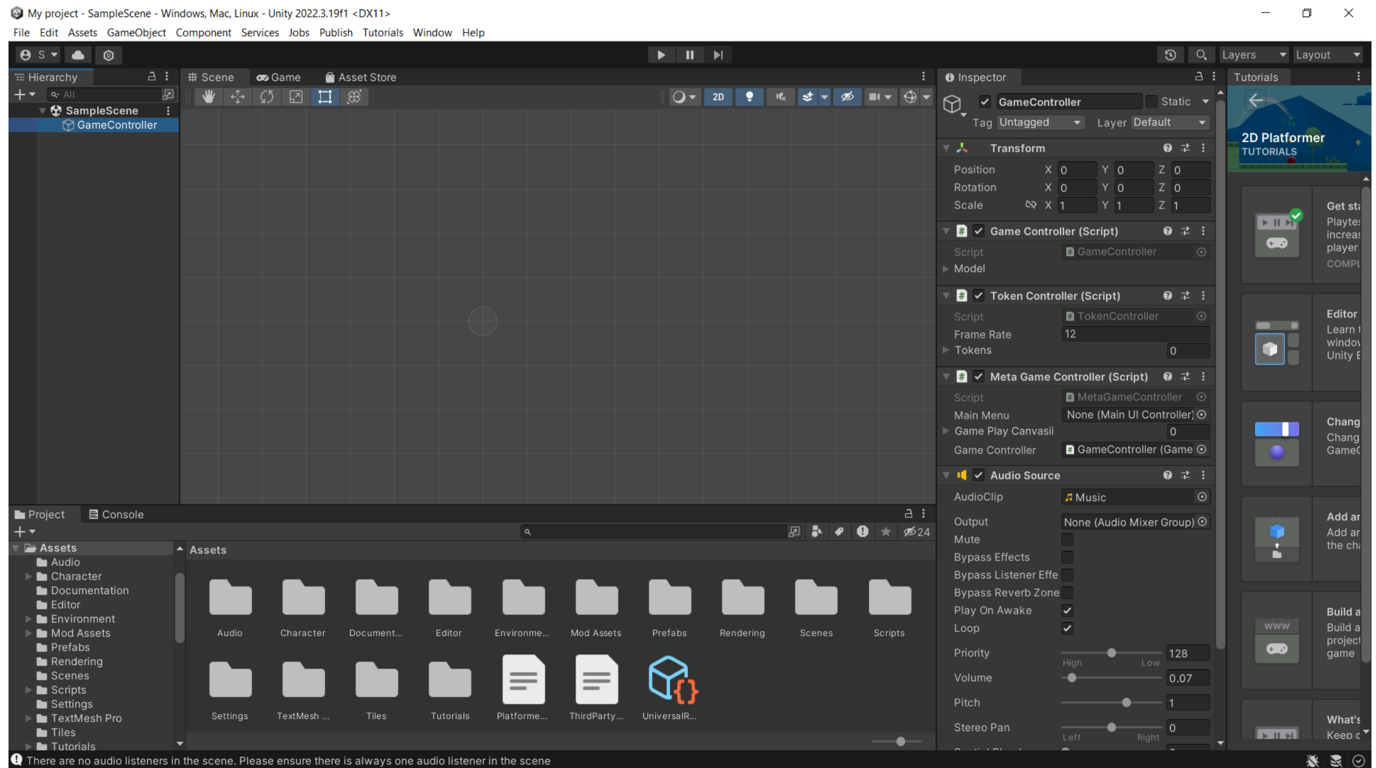


Рис. 2.2 Інтерфейс ігрового двигуна Unity

### 2.2.3. Unreal Engine

З метою спрощення завдань розробників, у 1996 році було створено Unreal Engine — ігровий двигун, початково спрямований на створення ігор у від першої особи(Рис. 2.3). Згодом його використання розширилося, що стало можливим завдяки простоті розробки та відсутності потреби в освоєнні складної мови програмування C++.

На ранніх стадіях використання Unreal Engine було платним. Проте у 2015 році розробники вирішили випустити безкоштовну пробну версію, яка отримала значну популярність. Це пояснюється можливістю створювати складні ігри, які не можна було розробити за допомогою інших безкоштовних двигунів.

Рухій Unreal Engine, який побудований на C++, дозволяє створювати ігри для різних операційних систем і платформ, таких як Microsoft Windows, Linux, Mac OS і Mac OS X, а також для консолей Xbox, Xbox 360, PlayStation 2,

PlayStation Portable, PlayStation 3, Dreamcast і Nintendo GameCube. У грудні 2009 року Марк Рейн представив роботу рушія Unreal Engine 3 на iPod Touch і iPhone 3GS. У березні 2010 року також була показана робота рушія на комунікаторі Palm Pre, який працює на платформі webOS.

Характеристики створення ігор на Unreal Engine:

– Рушій Unreal Engine має модульну систему компонентів, що дозволяє переносити ігри та інші розробки з однієї платформи на іншу.

– Будучи розробленим на мові програмування C++, Unreal Engine дозволяє створювати ігри для різних платформ, таких як Windows, Mac OS і Linux.

– Недавно розробники Unreal Engine активно працюють над створенням кейсів для платформи Android, що дозволяє створювати ігри для мобільних пристроїв.

Переваги Unreal Engine:

– Легкість використання. Цей рушій простий у роботі і доступний навіть для новачків, адже його навігація та налаштування подібні до інших програмних засобів.

– Розширені можливості. Unreal Engine дозволяє розробникам створювати свої продукти без обмежень, оскільки підтримує велику кількість функцій.

– Універсальність та доступність. Використовується як досвідченими розробниками, так і новачками, завдяки своїй універсальності.

– Активна спільнота користувачів, яка надає навчальні матеріали та допомагає вирішувати проблеми.

Вбудована система візуального скриптингу, що спрощує побудову ігрової логіки.

Умовно безкоштовне використання: плата за Unreal Engine стягується тільки в разі успіху гри.

Мультиплатформеність, яка дозволяє створювати ігри для різних пристроїв та платформ.

Недоліки Unreal Engine:

Високі вимоги до апаратного забезпечення, що може вимагати потужного комп'ютера для ефективної роботи.

Великий обсяг проекту, що може вплинути на обсяг файлів гри та вимагати більшої потужності обробки ЦП та обсягу пам'яті.

Рис. 2.3. Інтерфейс середовища розробки Unreal Engine

Результати дослідження засобів розробки програмного забезпечення показали, що середовище програмування Unity є найкращим для створення комп'ютерних ігор у стилі 2D платформер. Зрозумілий набір інструментів, зручний інтерфейс і простий синтаксис сприяють ефективній роботі. Крім того, освітня документація та безкоштовна ліцензія роблять Unity доступним і привабливим для розробників будь-якого рівня кваліфікації.

### 2.3. Середовище розробки Microsoft Visual Studio

Для досягнення успіху на ринку та забезпечення високої продуктивності будь-який розробник програмного забезпечення повинен володіти навичкою написання коду в короткий термін. Вибір правильного редактора коду або

інтегрованого середовища розробки (IDE) є важливим для створення та підтримки високоякісного програмного забезпечення. Розробники повинні вибирати найкращих редакторів, щоб досягти своїх цілей, оскільки обсяг коду та різноманітність мов програмування постійно зростають.

У середовищі Unity програмування виконується за допомогою інтегрованого середовища розробки Microsoft Visual Studio, яке може бути розширене додатковим функціоналом для підтримки Unity. Microsoft Visual Studio — це комплексне інтегроване середовище розробки програмного забезпечення, розроблене компанією Microsoft(Рис. 2.4). Це середовище дозволяє розробляти різноманітні програмні продукти, включаючи консольні програми, додатки з графічним інтерфейсом, такі як віконні додатки Windows Forms, а також веб-додатки та інші типи програм.

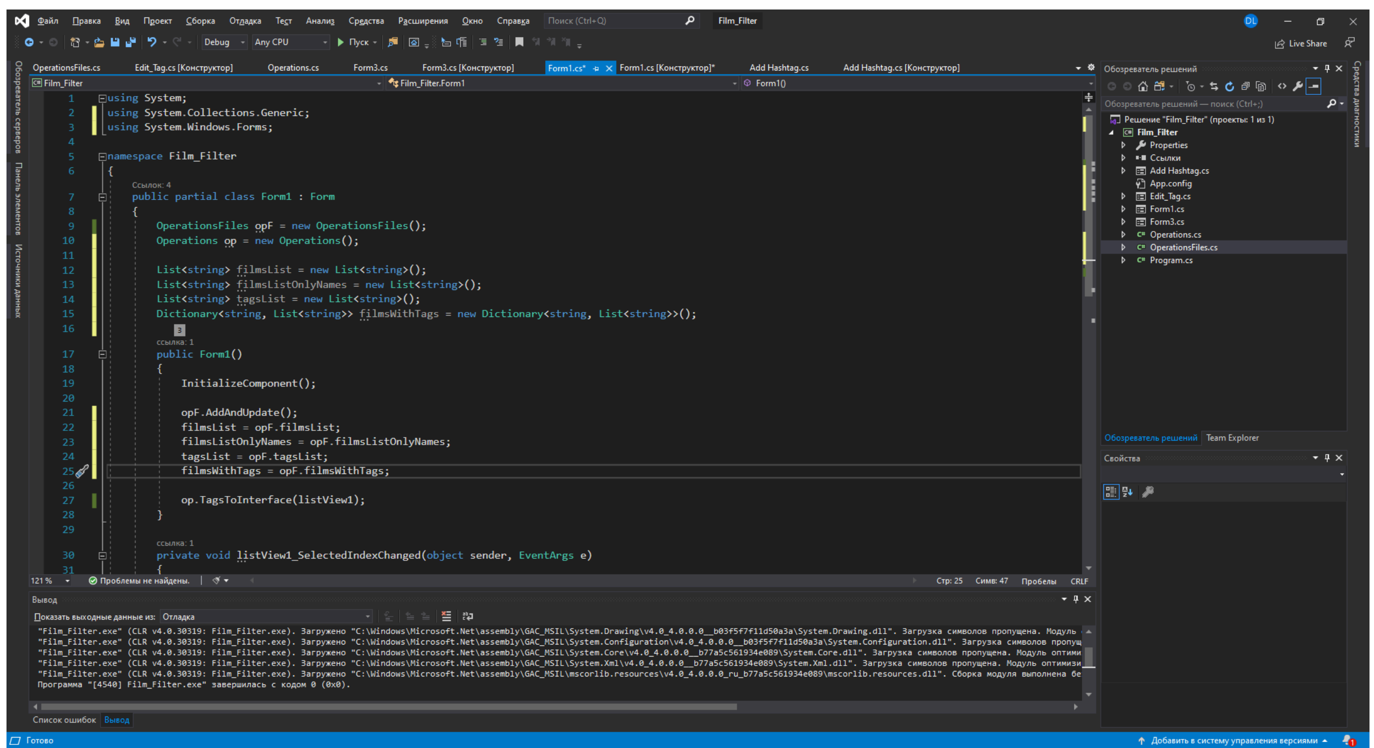


Рис. 2.4. Інтерфейс Visual Studio

Здатність підтримувати широкий спектр мов програмування, таких як C++, C#, Visual Basic, F# і Python, є однією з основних функцій Microsoft Visual Studio. Крім того, це інтегроване середовище розробки має багато інструментів, таких як відладчик, система контролю версій, дизайнер інтерфейсу користувача, візуальний редактор коду та система контролю версій..

Microsoft Visual Studio надає розширені можливості для командної розробки, включаючи спільну роботу над проектами, інтегровану зворотну зв'язок та інструменти збірки та розгортання програм. Крім того, воно має розширювану архітектуру, що дозволяє встановлювати сторонні плагіни та розширення для покращення його функціональності.

Середовище розробки Microsoft Visual Studio відзначається своєю потужністю та високою ефективністю для професійної розробки програмного забезпечення. Це надійне інструментальне середовище надає розробникам зручний і продуктивний інструментарій для роботи над проектами, включаючи їх створення, відлагодження та управління.

Основні переваги та можливості Microsoft Visual Studio охоплюють наступне:

– Розширена підтримка мов програмування: Visual Studio підтримує багато мов програмування, таких як C++, C#, Visual Basic, F#, Python та багато інших, що дозволяє розробникам вибрати мову, яка найкраще підходить для їхніх потреб і навичок..

– Інтегровані інструменти розробки: Вбудовані інструменти Visual Studio, такі як візуальний редактор коду, відлагоджувач, система контролю версій, дизайнер інтерфейсу користувача, профілер продуктивності та інші, сприяють ефективній розробці програмного забезпечення.

– Розширюваність і плагіни: Visual Studio дозволяє розширити свою функціональність за допомогою різних плагінів і розширень, що надають додаткові можливості для розробки. Розробники можуть встановлювати сторонні плагіни для підтримки конкретних технологій або фреймворків.

– Інтеграція з іншими продуктами Microsoft: Visual Studio добре працює з такими продуктами, як Azure, SQL Server і SharePoint. Під час розробки ці продукти взаємодіють легше завдяки цій інтеграції..

– Підтримка командної розробки: Visual Studio забезпечує розширені можливості для командної розробки, включаючи системи контролю версій,

спільну роботу над проектами та засоби збірки та розгортання програм, що полегшує спільну роботу над проектами в команді.

Незважаючи на численні переваги, Microsoft Visual Studio також має свої недоліки:

– Великі вимоги до системи: Visual Studio може вимагати значних обчислювальних ресурсів та дискового простору. Це може стати проблемою для менш потужних комп'ютерів або віртуальних середовищ.

– Вартість: Деякі версії Visual Studio, особливо при комерційному використанні, мають високу вартість ліцензування. Це може ускладнювати доступність для незалежних розробників з обмеженими фінансовими ресурсами.

– Великий розмір інсталяційного пакету: Інсталяційний пакет Visual Studio може бути досить великим, особливо з урахуванням включених функціональних компонентів і підтримуваних мов програмування.

– Visual Studio є дуже потужним та функціональним, що може призвести до складності у використанні, особливо для новачків. Інтерфейс може здатися перевантаженим і важким для освоєння.

– Хоча є версії Visual Studio для macOS, вони не такі потужні та функціональні, як версії для Windows. Це може створити проблеми для розробників, які працюють на різних платформах.

– Хоча оновлення зазвичай приносять нові функції та виправлення, часті оновлення можуть дратувати користувачів, оскільки часто вимагають перезавантаження системи або можуть призводити до непередбачених проблем із сумісністю.

## ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було розглянута інструментальна база для програмної реалізації ігрового проекту. Мова програмування C# була вибрана через її широке використання та потужні можливості, які дозволяють ефективно розробляти ігровий функціонал.

Проведено аналіз таких ігрових двигунів, як CryEngine, Unity та Unreal Engine, виявлено їхні особливості, переваги та недоліки. Цей аналіз послужив підґрунтям для обрання Unity як ігрового двигуна для даного проекту.

Також було досліджено та проаналізовано середовище розробки Microsoft Visual Studio, яке є одним із найпопулярніших інструментів для програмної розробки. Його потужність та багатий функціонал дозволяють ефективно працювати над проектами та забезпечувати високу якість коду.

На основі аналізу було вирішено, що використання мови програмування C#, ігрового двигуна Unity та середовища розробки Microsoft Visual Studio є правильним вибором для реалізації проекту гри у жанрі 2D платформер. Ці інструменти полегшують роботу розробників і досягають бажаного результату під час виконання проекту..

# РОЗДІЛ 3
# РЕАЛІЗАЦІЯ ПРОЕКТУ

## 3.1. Концепція та сценарій гри

Для цього проекту було обрано жанр платформер. Подолання різноманітних перешкод і використання стрибків між платформами для пересування в просторі є основними ознаками цього жанру. Традиційна структура геймплею передбачає завершення одного рівня і перехід до наступного, який зазвичай є більш складним або відрізняється за деякими параметрами. Ігри цього жанру не дуже складні в реалізації, але вони можуть зацікавити гравців протягом тривалого періоду часу та зацікавити широке коло аудиторії.

Головним персонажем сюжету є лицар-дракон, який живе у вигаданому світі. Він має крила, які дозволяють йому літати в небі на кілька секунд. Крім того, він має здатність завдавати шкоди вогняним диханням.

Основне завдання полягає в подоланні різноманітних перешкод і продовжувати рухатися вперед. Головний герой зіткнеться з численними перешкодами, які намагатимуться завдати йому шкоди або навіть вбити його. У міру того, як грок продовжує свою подорож, він стикається з більшою кількістю небезпечних ситуацій і стає все важче зупинитися. Коли користувач грає в гру, його здатність керувати головним персонажем розвивається, що дозволяє йому проходити більше рівнів і створювати нові власні рекорди.

## 3.2. Цільова аудиторія

Під час розробки гри важливим є правильне визначення цільової аудиторії, оскільки це дозволяє краще зрозуміти потреби, інтереси та бажання гравців, на яких спрямована гра. Це необхідно для створення продукту, який відповідає очікуванням цільової аудиторії та гарантує, що вона буде задоволена грою. Коли гра адаптована до цільової аудиторії, вона стає більш привабливою для гравців, що відображається на її комерційному успіху та репутації серед геймерської спільноти.

У маркетингу та просуванні гри визначення цільової аудиторії є важливим

етапом, оскільки це дозволяє адаптувати рекламні кампанії та стратегії до цільової аудиторії. Розробники можуть створити продукт, який відповідає очікуванням гравців і допомагає їм насолоджуватися грою.

Цільовою аудиторією гри є гравці платформерів, які мають інтерес до двовимірних ігор. Вони включають як початківців, так і досвідчених гравців, які насолоджуються викликами цих ігор. Платформери можуть привернути увагу різних груп гравців, включаючи дорослих і підлітків.

### 3.3. Графічне оформлення

Візуальна атмосфера будь-якої комп’ютерної гри є важливою, і графічне оформлення часто визначає її. Ми розглянемо основні терміни геймдизайну, такі як "спрайт" і "тайл", перш ніж розпочати ознайомлення з графічної частини.

Тайл - це малий повторюваний фрагмент, що використовується для створення великих зображень, і цей процес називається "тайлова графіка".

Спрайт — це графічне зображення об’єкта, яке використовується в двовимірних іграх і може містити кілька зображень для анімації об’єкта.

У візуальному складовому прототипі використовується готовий набір двовимірних спрайтів, який можна знайти в офіційному магазині Unity Asset Store.Цей набір містить широкий спектр різних жанрів. Було обрано текстури та спрайти, які відповідають ідеям гри. Редагування спрайтів можна виконати за допомогою функції Sprite Editor(Рис. 3.1), яка доступна в середовищі розробки Unity.

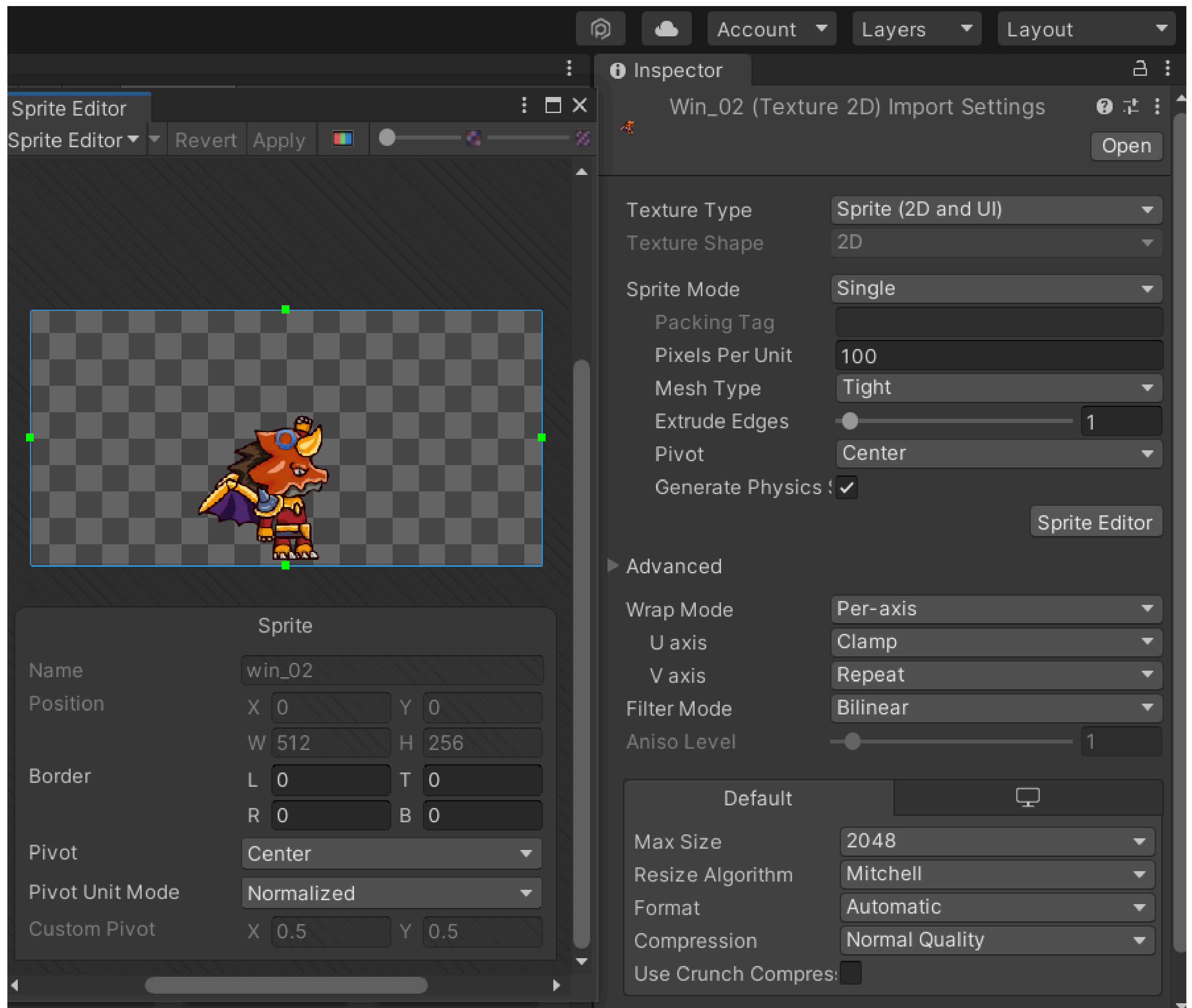


Рис. 3.1 функція sprite editor

За допомогою функції Sprite Editor можна відкривати великі спрайт-об'єкти та розбивати їх на складові частини для подальшої анімації.

### 3.3.1. Спрайти головного персонажу

Головний герой був зображений у вигляді спрайта дракона(Рис. 3.2), який складався з різних спрайтів, які відображали різні дії, наприклад стояння, біг, стрибки, пошкодження та смерть. Крім того, був завантажений спрайт, який дозволяв боротися з ворогами за допомогою вогняного дихання(Рис. 3.3).

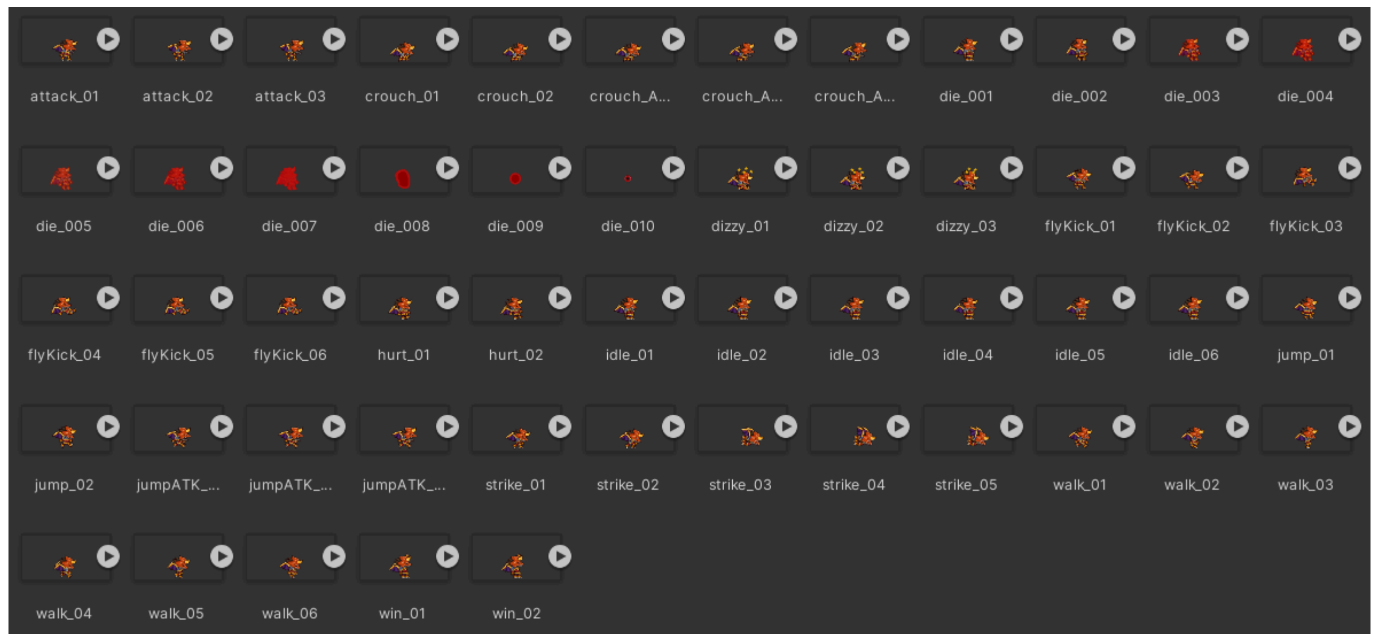


Рис. 3.2. Спрайти головного персонажу

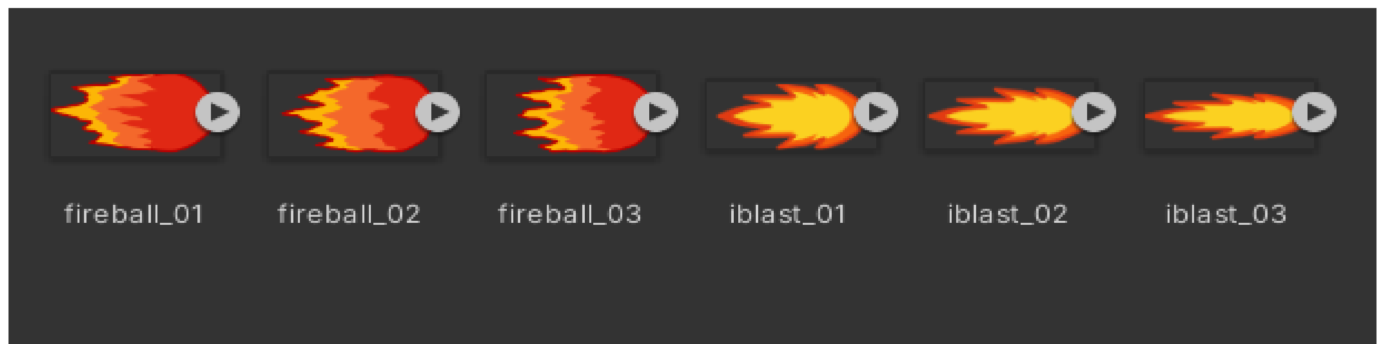


Рис. 3.3. Спрайти вогняного шару

Вибір спрайтів для фонового зображення є наступним кроком у реалізації гри. Крім того, для створення графічного інтерфейсу гри також були використані візуальні елементи.

Для використання в якості фонового зображення були вибрані спрайти, які доступні в офіційному магазині Unity Asset Store(Рис. 3.4).

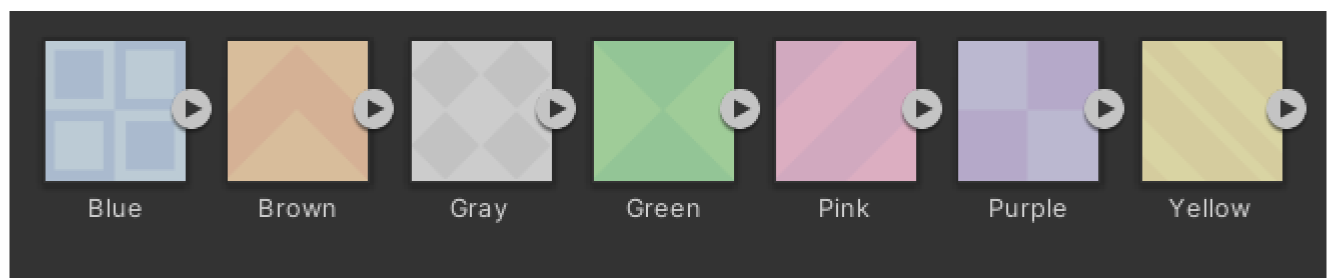


Рис. 3.4 Спрайти фонового зображення

Для відображення візуальних елементів гри було використано спрайт платформ у вигляді цегляної кладки(Рис. 3.5), по яким буде стрибати та бігати

ігровий персонаж.

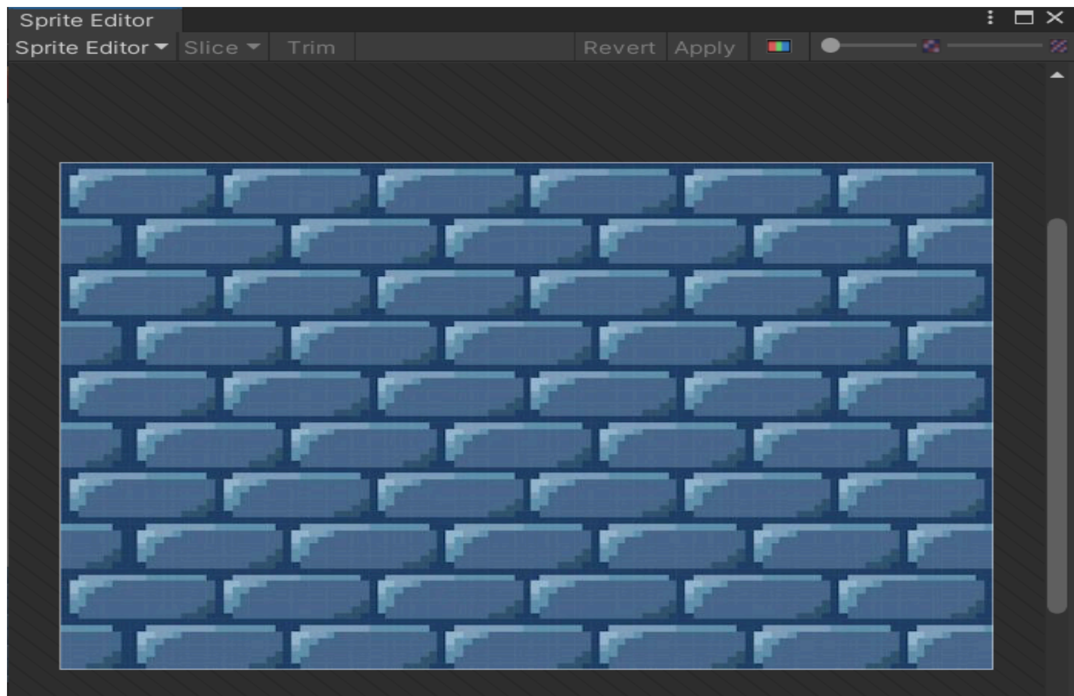


Рис. 3.5 Спрайт платформи

Для відображення стін, що обмежують рух персонажа, були обрані спрайти дерев'яних стін(Рис. 3.6).

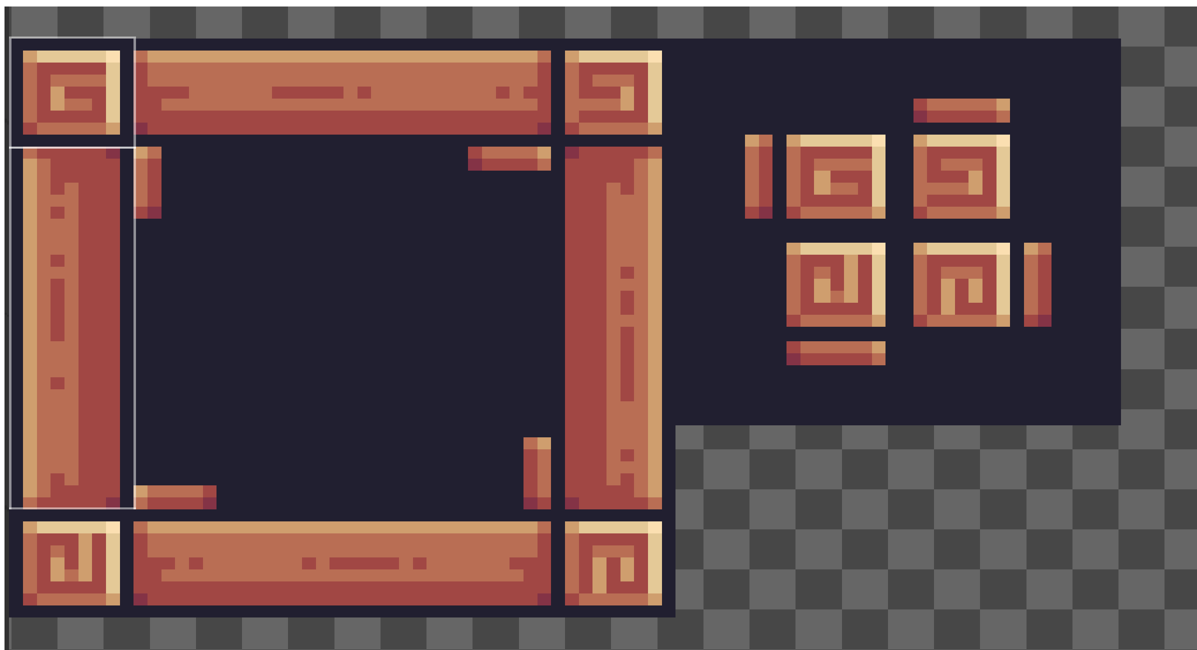

Рис. 3.6 Спрайти стін

Для візуального подання ворога був обраний спрайт лицаря, який атакує ігрока(Рис. 3.7). Подібно до головного персонажа, ворог також буде анімованим. Тому для відтворення анімації отримання шкоди та смерті використовуються спрайти. Крім того, лицар, так само як і головний герой, володіє здатністю випускати вогняні снаряди для завдання ушкоджень.

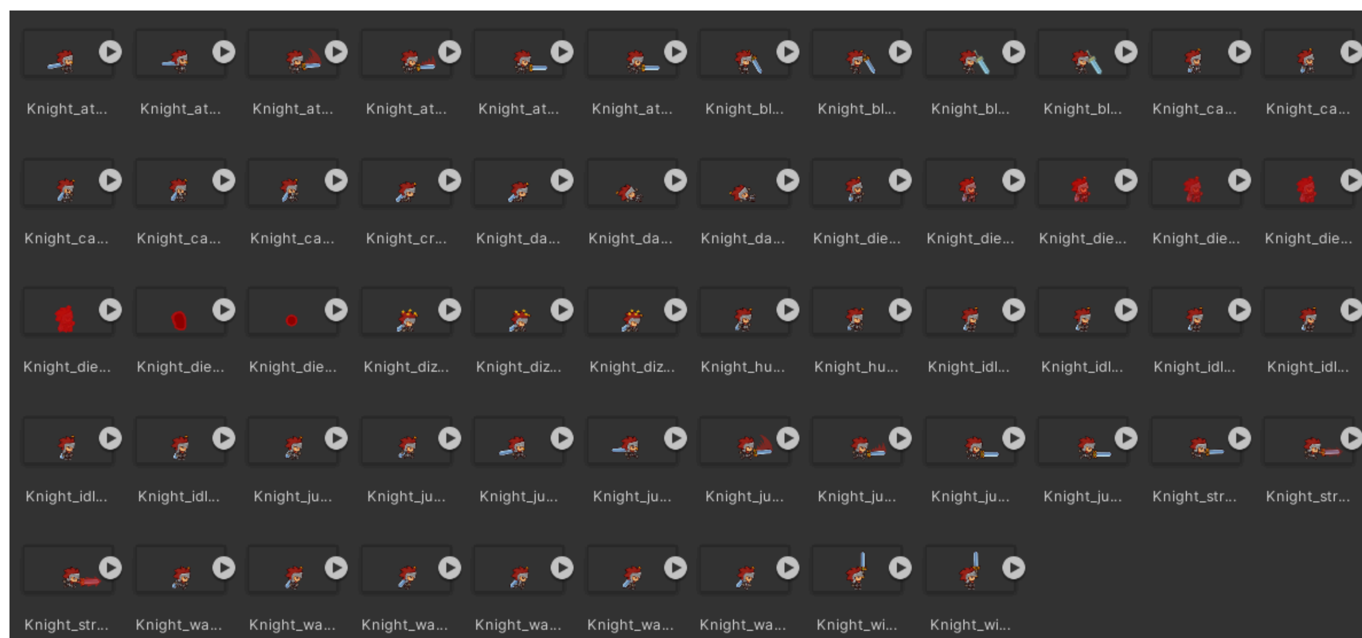


Рис. 3.7 Спрайти ворога

### 3.4. Етап проектування гри

Для розробки комп'ютерної гри використовувалися ігровий двигун Unity та середовище програмування Microsoft Visual Studio. При запуску Unity відкривається вікно проекту(Рис. 3.8), яке містить різні панелі, такі як вікно Scene, вікно Animator і вікно Game. Вікно Scene використовується для створення композиції рівня та додавання нових об'єктів, вікно Animator дає можливість створювати та редагувати анімації об'єктів, а вікно Game показує перспективу з камери, що демонструє вигляд гри на поточний момент.

Ліворуч розміщене вікно Hierarchy, в якому відображаються всі об'єкти, які використовуються у поточній сцені.

Праворуч розташоване вікно Inspector, яке відображає всі поточні властивості обраного об'єкта та дає доступ до їх налаштування.

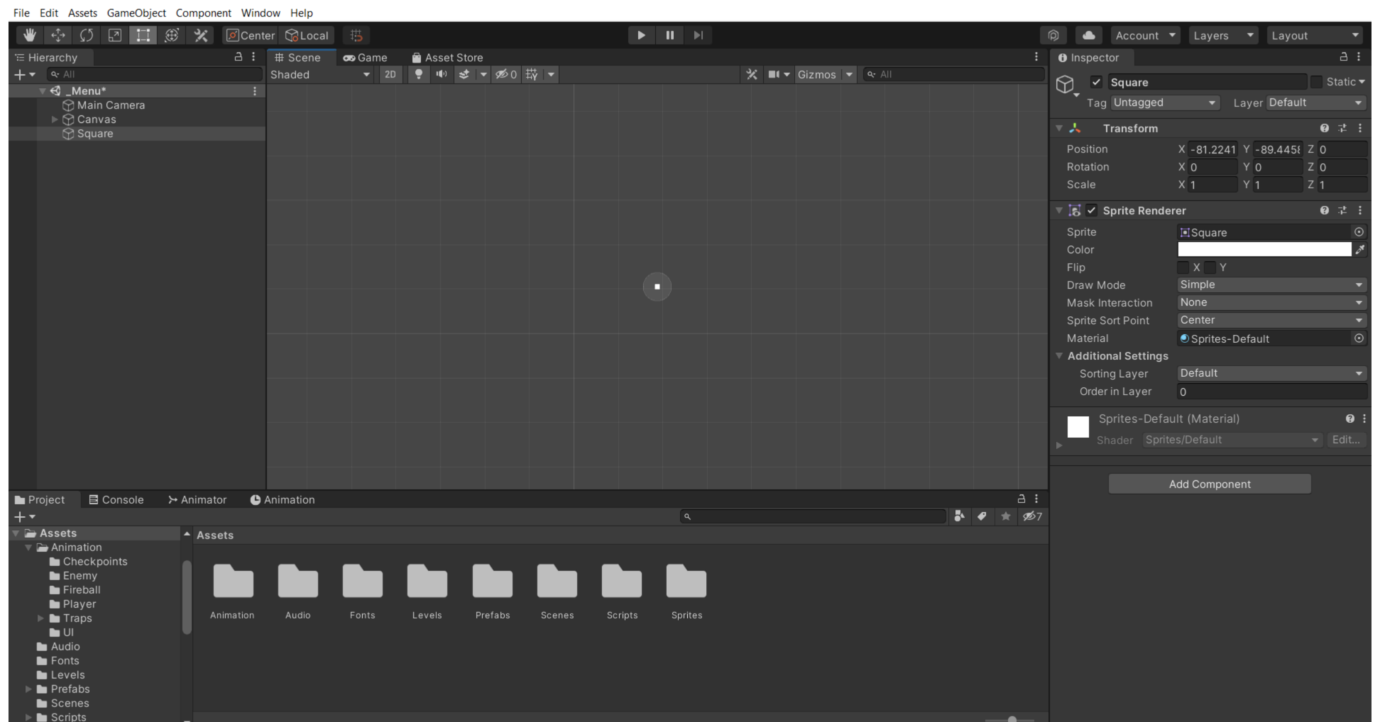


Рис. 3.8 Вікно проекту Unity

### 3.4.1 Фізичні властивості об’єктів

Надання об’єктам фізичних властивостей є першим кроком у створенні прототипу гри. По-перше, на сцену потрібно додати платформу, по якій рухатиметься головний герой, а також надати їй фізичні характеристики. Оскільки програма не вважатиме платформи фізичними об’єктами, гравець просто буде нескінченно падати у просторі.

Спочатку потрібно додати сам спрайт платформи, який був створений раніше. Щоб це зробити, просто перетягніть його з папки у вікно проекту. Після цього спрайт повинен з’явитися серед усіх об’єктів, які вже включені до гри. Далі ви повинні додати платформу, яку вибрали, до поточної сцени. Це можна зробити двома способами: або перетягнути його з вікна Project до вікна Scene, або спочатку створити окремий порожній об’єкт у вікні Hierarchy, а потім застосувати спрайт платформи до нього.

На створену платформу спочатку додається компонент Box Collider 2D, який надає їй фізичну форму куба, за допомогою функціоналу Unity. Це прямокутник у локальному координатному просторі спрайту з визначеним положенням, шириною та висотою(Рис. 3.9).

У Unity є два види колайдерів: один для двомірних об’єктів, а інший для трьохмірних об’єктів. Об’єкти повинні мати двовимірні колайдери, оскільки ви створюєте гру двох вимірів. Щоб додати колайдер до об'єкта, спочатку потрібно виділити його у вікні Hierarchy. Потім у вікні Інспектора потрібно натиснути кнопку «Add component». Кнопка «Edit collider» дозволяє змінити розміри колайдера.

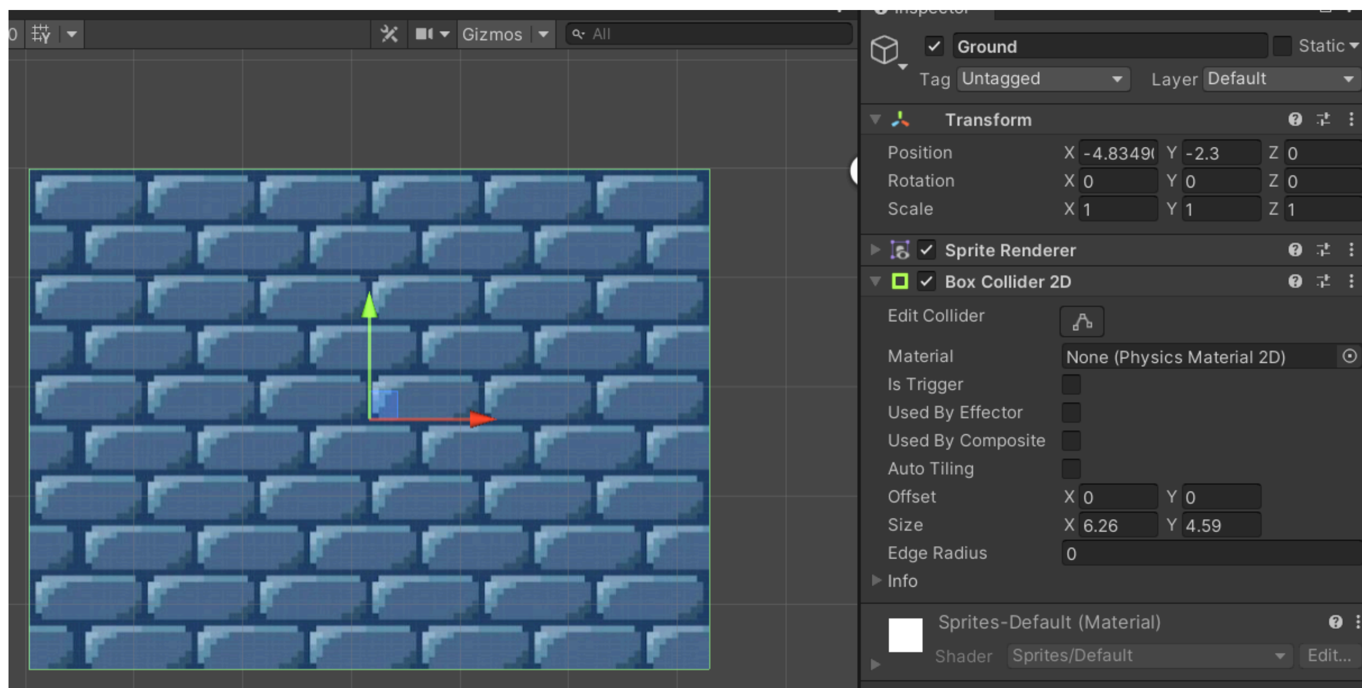


Рис. 3.9 Платформа з 2D коллайдером у вікні Scene

Наступним кроком є створення головного героя гри. Для цього також додається Box Collider 2D, щоб надати фізичні властивості об’єкту, а також обраний спрайт героя. Наступним кроком є включення компоненту Rigidbody 2D(Рис. 3.10).

Компонент Rigidbody 2D у Unity є одним з ключових елементів для моделювання фізичної поведінки об'єктів у двовимірному просторі гри. Він надає можливість об'єктам рухатись, взаємодіяти з фізичним середовищем та реагувати на сили, що діють на них.

Крім цього, компонент Rigidbody 2D працює в парі з коллайдерами, що дозволяє об'єктам взаємодіяти з іншими об'єктами, виявляти зіткнення та реагувати на них.

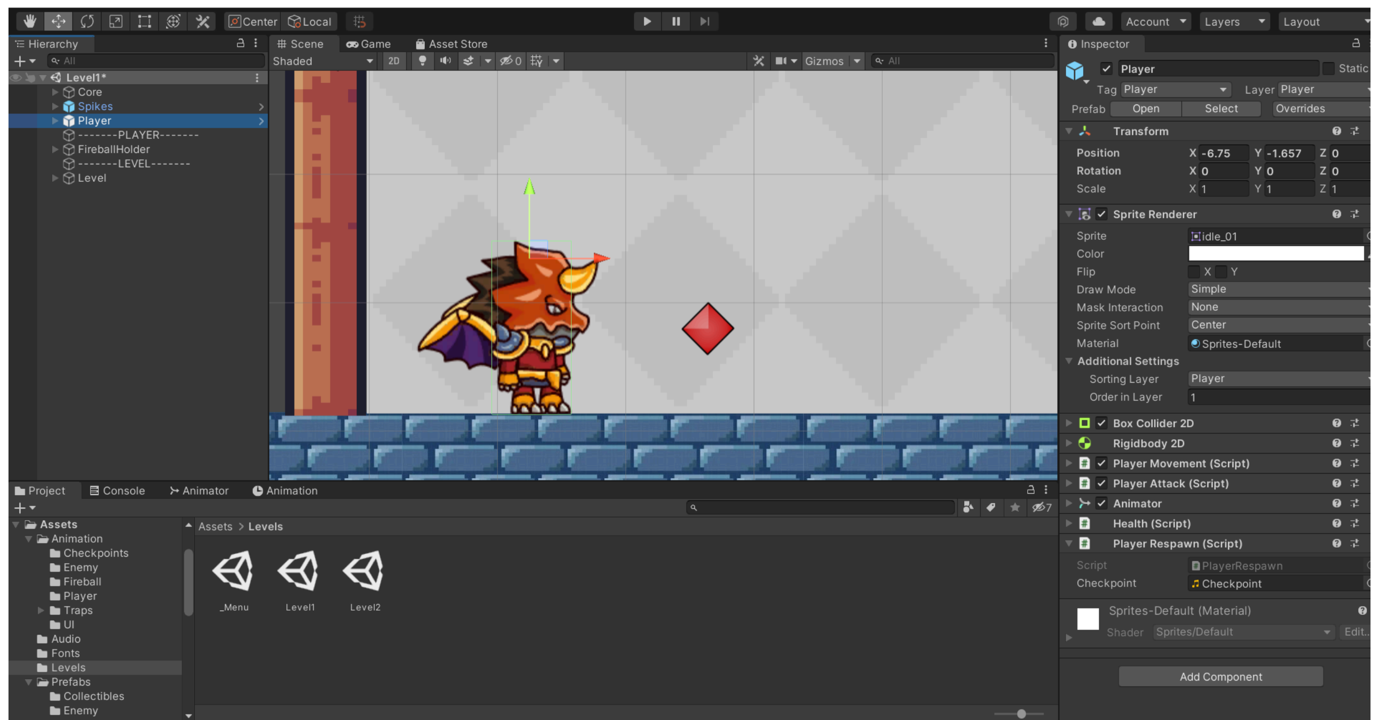


Рис. 3.10 Головний персонаж у вікні Scene

### 3.4.2. Рух об'єктів

Для того, щоб головний герой міг рухатися та виконувати різні дії, потрібно написати скрипт на мові програмування C#, який буде мати назву PlayerMovement. Це можна зробити двома способами: у вікні Project його можна створити або безпосередньо на потрібному об'єкті у вікні Інспектора. Відкриття скрипту автоматично виконує запуск Microsoft Visual Studio. У коді вже будуть підключені основні бібліотеки та створено типовий метод оновлення.

Наступним кроком у розробці ігрового персонажа є написання скрипту для його руху та стрибків. Спочатку визначаються змінні(Рис. 3.11).

```
//заданні змінні
private Rigidbody2D body;
private Animator anim;
private BoxCollider2D boxCollider;
private float wallJumpCooldown;
private float horizontalInput;
```

Рис. 3.11 Змінні для скрипта PlayerMovement

Змінна body буде зберігати об'єкт класу Rigidbody2D, який використовується для моделювання фізичних властивостей об'єкта.

Змінна anim буде отримувати значення класу Animator, який використовується для анімації дій, таких як біг, атака та інші(Рис. 3.12).

```
//параметри аніматора
anim.SetBool("run", horizontalInput != 0);
anim.SetBool("grounded", isGrounded());
```

Рис. 3.12 Встановлення параметрів класу Animator

Змінна boxCollider буде мати значення класу BoxCollider2D, який використовується для надання фізичних властивостей об'єкту.

Змінна horizontalInput буде використовуватися для керування рухом персонажа. Для цього використовується клас Input, який відповідає за отримання введення, та його метод GetAxis(), що визначає горизонтальний рух персонажа. Метод GetAxis("Horizontal") повертає значення -1, якщо персонаж рухається вліво, та 1, якщо рухається вправо. Отримане значення цього методу присвоюється змінній horizontalInput.

Для того, щоб головний герой мав можливість дивитися в обидві сторони - вправо і вліво, в залежності від напрямку руху, написан такий код(Рис. 3.13).

```
//розворот спрайта ігрока ліворуч/праворуч
if (horizontalInput > 0.01f)
    transform.localScale = Vector3.one;
else if (horizontalInput < -0.01f)
    transform.localScale = new Vector3(-1, 1, 1);
```

Рис. 3.13 Код для повороту героя в обидві сторони

Якщо значення змінної horizontalInput перевищує 0.01f (що вказує на рух вправо), тоді метод transform.localScale, що регулює масштаб об'єкту, встановлюється на значення 1, 1, 1 по відповідним осям x, y, z, що спрямовує головного героя вправо. Аналогічно, якщо значення змінної horizontalInput менше -0.01f, метод transform.localScale встановлюється на значення -1, 1, 1, що спрямовує персонажа вліво.

Наступним етапом буде надання можливості стрибати герою. Для цього використовується клас Input та його метод GetKeyDown, який перевіряє, чи була натиснута клавіша пробілу (Space). Якщо так, то викликається метод Jump(), який відповідає за виконання стрибка.

Метод Jump() включає послідовність перевірок та дій, пов'язаних з різними умовами стрибка:

– Перш ніж виконати стрибок, перевіряється, чи виконується будь-яка з умов, при яких стрибок не потрібен. У такому випадку виконання методу завершується, і стрибок не відбувається.

– Перевіряється, чи гравець знаходиться на стіні. У цьому випадку викликається метод WallJump(), який виконує стрибок від стіни.

– Перевіряється, чи гравець перебуває на землі (за допомогою isGrounded()). Якщо це так, гравець здійснює звичайний стрибок, змінюючи його вертикальну швидкість (body.velocity.y) на значення jumpPower.

– Перевіряється значення coyoteCounter. Цей лічильник відповідає за короткий час, коли гравець може здійснити стрибок після зникнення контакту з землею. Якщо значення coyoteCounter більше 0, гравець виконує звичайний стрибок.

– Якщо значення coyoteCounter менше або дорівнює 0, перевіряється умова if (jumpCounter > 0). jumpCounter відповідає за кількість доступних додаткових стрибків. Якщо jumpCounter більше 0, гравець виконує стрибок (подвійний стрибок) і зменшує значення jumpCounter на 1.

Також, для визначення того, чи перебуває персонаж на землі чи ні, потрібно створити порожній дочірній об'єкт у героя, який буде називатися isGrounded. Це необхідно, щоб уникнути можливості безкінечного вертикального руху гравця при постійному натисканні кнопки стрибка.

```
    //Стрибок
    if (Input.GetKeyDown(KeyCode.Space))
        Jump();

    //Регульована висота стрибка
    if (Input.GetKeyUp(KeyCode.Space) && body.velocity.y > 0)
        body.velocity = new Vector2(body.velocity.x, body.velocity.y / 2);

    if (onWall())
    {
        body.gravityScale = 0;
        body.velocity = Vector2.zero;
    }
    else
    {
        body.gravityScale = 7;
        body.velocity = new Vector2(horizontalInput * speed, body.velocity.y);

        if (isGrounded())
        {
            coyoteCounter = coyoteTime; //Скидання coyote counter на землі
            jumpCounter = extraJumps; //Скидання jump counter до додаткового значення стрибка
        }
        else
            coyoteCounter -= Time.deltaTime; //Початок зменшення coyote counter не на землі
    }
}
```

Рис. 3.14.1 Код для реалізації стрибків(перча частина)

```
private void Jump()
{
    if (coyoteCounter <= 0 && !onWall() && jumpCounter <= 0) return;
    //Якщо coyote counter дорівнює 0 або менше, і він не стоїть на стіні та не має додаткових стрибків, нічого не робіть

    SoundManager.instance.PlaySound(jumpSound);

    if (onWall())
        WallJump();
    else
    {
        if (isGrounded())
            body.velocity = new Vector2(body.velocity.x, jumpPower);
        else
        {
            //Якщо не на землі та coyote counter більше 0, виконайте звичайний стрибок
            if (coyoteCounter > 0)
                body.velocity = new Vector2(body.velocity.x, jumpPower);
            else
            {
                if (jumpCounter > 0) //Якщо є додаткові стрибки, тоді стрибок та зменшує jump counter
                {
                    body.velocity = new Vector2(body.velocity.x, jumpPower);
                    jumpCounter--;
                }
            }
        }

        //Скидання coyote counter до 0 для выдключення подвыйного стрибка
        coyoteCounter = 0;
    }
}
```

рис. 3.14.2. Код для реалізації стрибків(друга частина)

### 3.4.3. Анімація об’єктів

Наступний крок - створення анімацій для об'єктів, зокрема для головного персонажа. Планується створити декілька анімацій, таких як біг, стрибок, смерть, отримання пошкоджень, атака та спокійний стан. Для цього будуть використані вже обрані та завантажені спрайти з офіційного сайту. За допомогою функції Sprite Editor буде проведено розділення великого спрайту астронавта.

Далі будуть додані скрипти для всіх цих анімацій до проекту. Потім буде створений Animator Controller, який буде приєднаний до персонажа. Цей контролер визначатиме правила та зв'язки для анімацій конкретного об'єкта.

Для створення окремих анімацій перейдемо до вікна Animation(Рис. 3.15.1) та створимо новий анімаційний кліп для кожного типу анімації, вибравши відповідний об'єкт. Потім впорядковуємо всі кадри анімації у цьому вікні, де також налаштуємо швидкість анімації.

Після створення всіх необхідних анімаційних кліпів перейдемо до вікна Animator(Рис. 3.15.2), де створимо зв'язки та налаштуємо умови переходів між різними анімаціями, що були створені раніше.

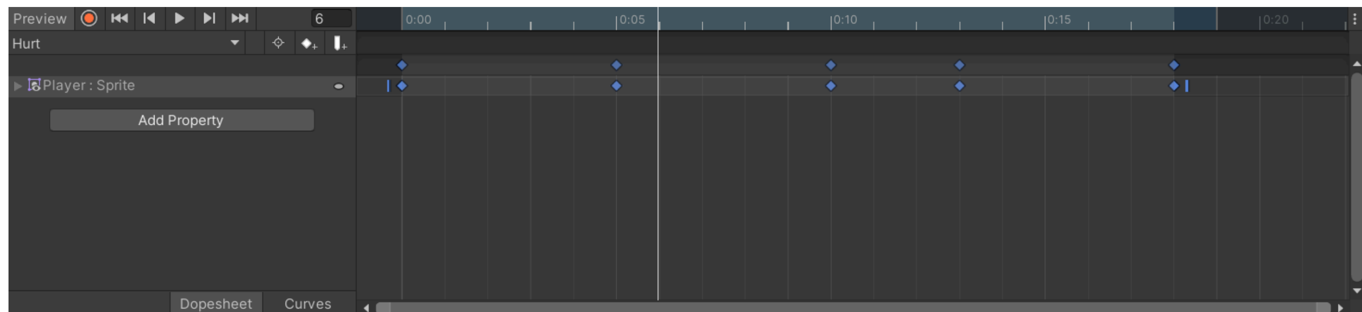


Рис. 3.15.1 Створення анімації

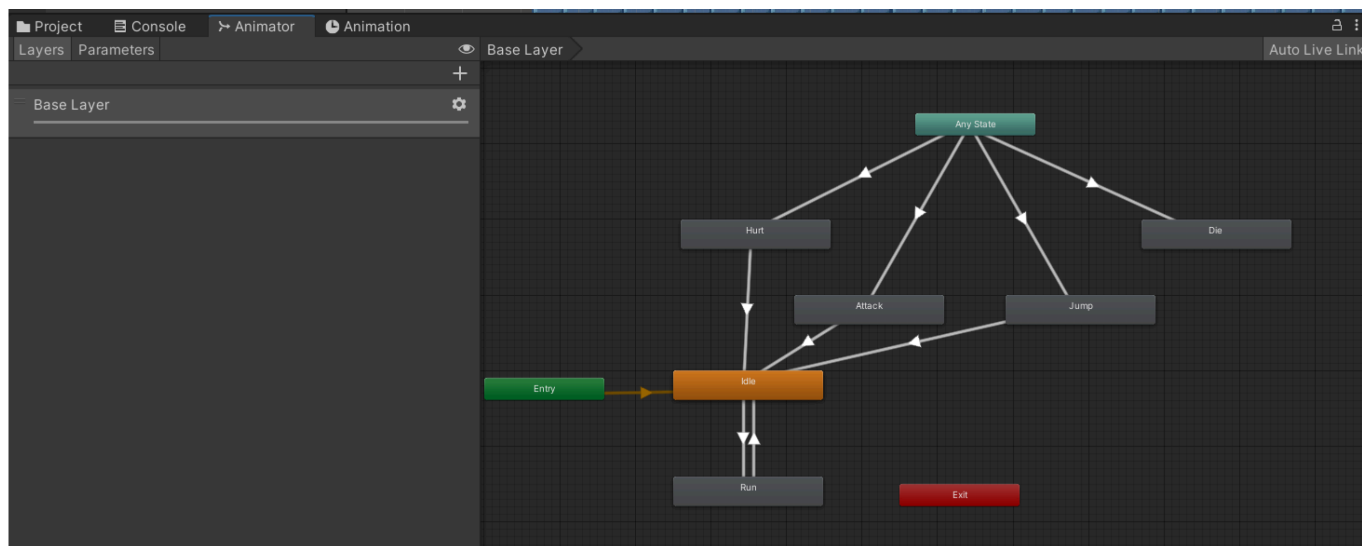


Рис. 3.15.2 Налаштування роботи анімацій

Після завершення створення всіх анімацій настає час прив'язати їх до конкретних клавіш клавіатури, кнопок миші, або інших пристроїв вводу, таких як геймпади або консолі. Ця функціональність відома як система введення, яка є основою взаємодії у реальному часі в ігрових проектах.

Прив'язка різних дій до логіки коду та активація різних пристроїв вводу можлива за допомогою візуального інтерфейсу вікна Input Manager(Рис. 3.16). Цей інструмент дозволяє налаштовувати, які клавіші або кнопки відповідають за певні дії в грі, і забезпечує зручний спосіб налаштування різних варіантів керування.

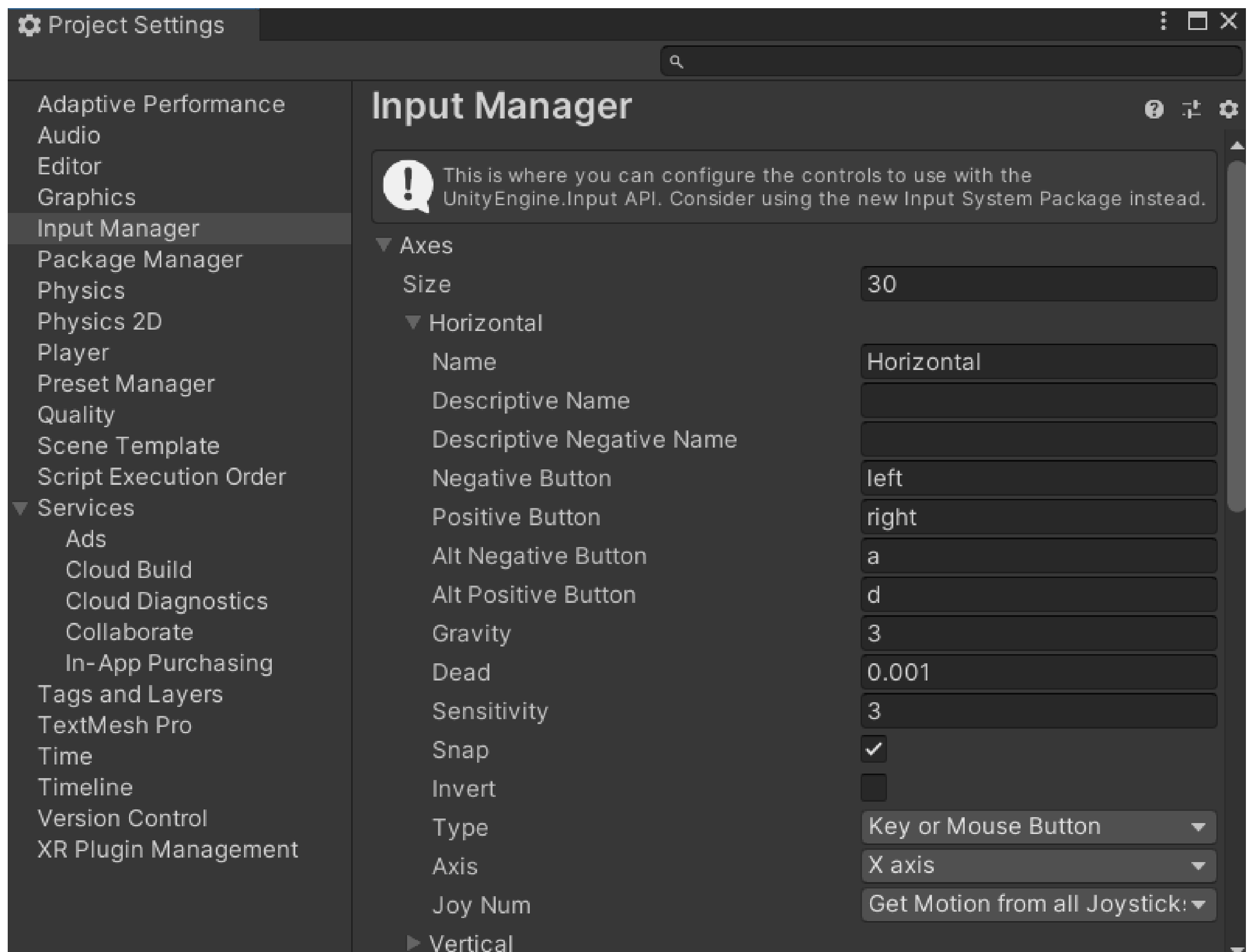


Рис. 3.16 Панель налаштування введення для руху по горизонтальній осі

Кожна вісь в Input Manager пов'язана з двома кнопками на джойстику, миші або клавіатурі. Поле "Name" використовується для ідентифікації осі у скрипті. "Positive button" - це кнопка, яка збільшує значення осі у позитивному напрямку. "Alt Positive Button" - альтернативна кнопка, яка збільшує значення осі у протилежному напрямку. Параметр "Gravity" визначає швидкість, з якою вісь повертається до нейтрального положення, коли кнопки не натиснуті, вимірюється в одиницях за секунду. "Dead" - це розмір мертвої зони для аналогових пристроїв; всі значення в цьому діапазоні вважаються нейтральними. "Sensitivity" - це швидкість, з якою вісь рухатиметься до цільового значення, вимірюється в одиницях за секунду. Ця функція працює лише для цифрових пристроїв. "Type" - це тип введення, який контролює дану вісь.

### 3.4.4. Звукові ефекти

Звукові ефекти є життєво важливими для гри, оскільки вони покращують геймплей, передають інформацію гравцеві, впливають на його настрій і гарантують задоволення від гри. Вони створюють аудіальне середовище, яке зацікавлює гравця, додаючи реалізму та сприяючи взаємодії з віртуальним світом. Звукові ефекти також можуть створювати різні емоційні стани, такі як занепокоєння, радість, смуток або страх, що відповідає концепції гри. Звук, який добре підібраний, може підвищити емоції та враження гравця від гри.

У Unity можна відтворювати звук за допомогою компонентів AudioSource(Рис. 3.17) та AudioClip. Відтворення звукових ефектів контролюється компонентом AudioSource, а аудіо дані містяться в AudioClip.

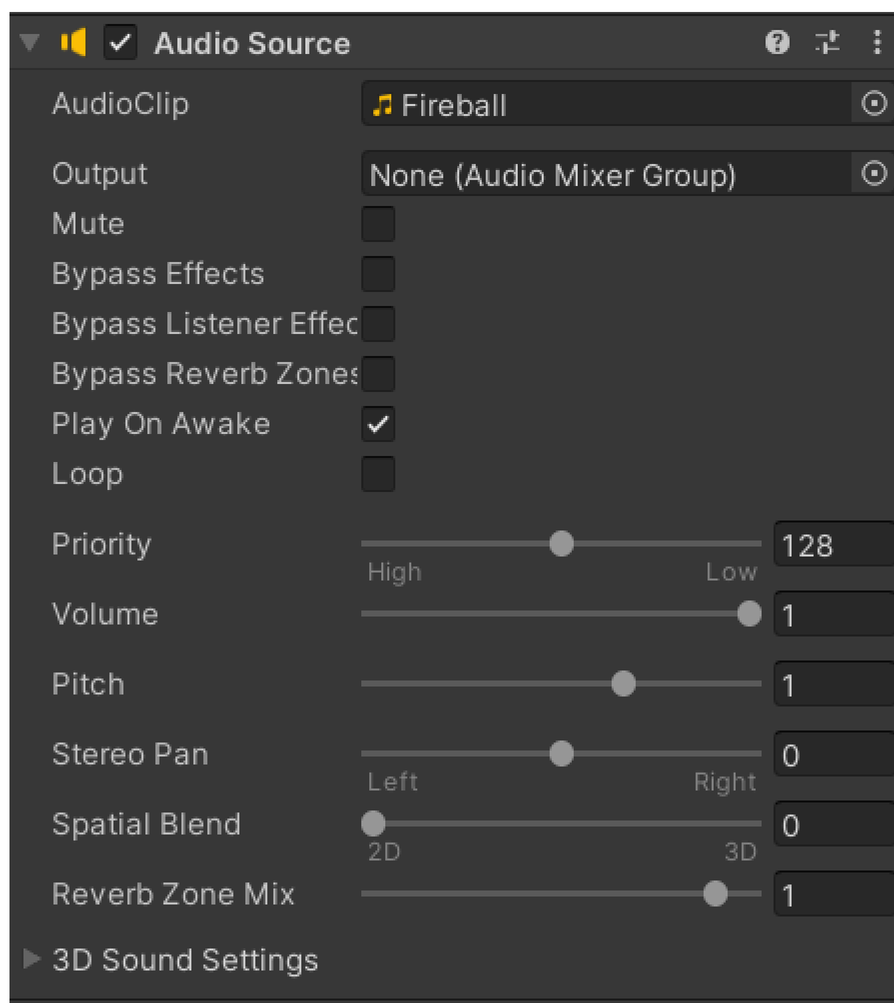


Рис. 3.17 Налаштування звукових ефектів

Компонент AudioSource пропонує широкий вибір налаштувань, що дозволяє змінювати звукові ефекти в грі. Наприклад, можна налаштувати AudioClip для відтворення, регулювати гучність звуку, змінювати панораму,

щоб розділити звук між правим і лівим каналами, змінювати ефекти просторового звучання, включаючи віддалення та приближення, встановлювати затримки між відтворенням звуків та інші налаштування.

У свою чергу AudioClip містить аудіодані, такі як звукова хвиля та її характеристики. Можна використовувати вбудовані звукові ресурси, створювати нові AudioClip або імпортувати власні звукові файли.

Щоб реалізувати різноманітний звуковий дизайн у грі, використання компонентів AudioSource та AudioClip поглиблює відчуття віртуального світу гри.

Звукові ефекти супроводжують кожну взаємодію в грі. Наприклад, головний герой відтворює звуки стрибка, ушкодження, атаки та смерті. Крім того, для створення атмосфери гри додано фонову музику.

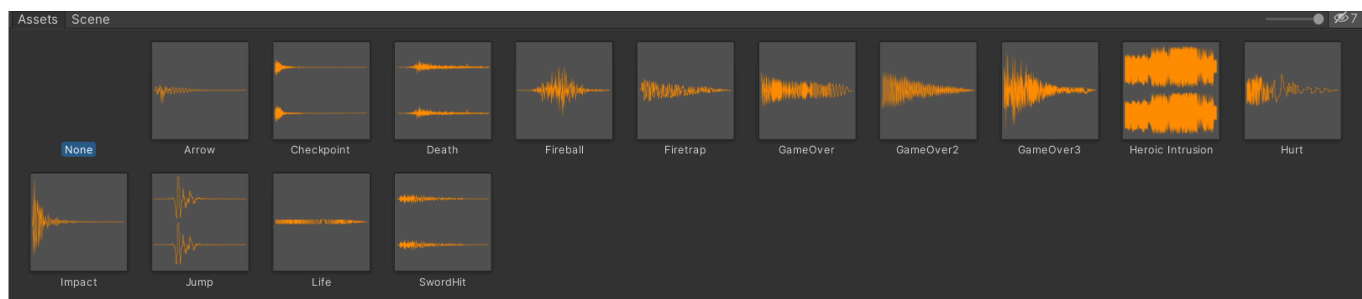


Рис. 3.18 Звукові файли гри

### 3.4.5. Розробка меню початку, паузи та кінця гри

Початкове меню гри визначає перше враження гравця та те, як він сприймає гру. Створення привабливого, зрозумілого та зручного інтерфейсу, який привертає увагу гравця та зацікавлює його, має вирішальне значення.

При проектуванні стартового меню потрібно враховувати багато речей. По-перше, дизайн повинен бути красивим і відповідати загальному стилю гри. Після цього інтерфейс повинен бути простим у використанні та зрозумілим, використовуючи прості символи та піктограми. Розташування кнопок повинно бути таким, щоб вони були легко доступними та зручними для натискання.

По-третє, гравець має мати можливість легко орієнтуватися у стартовому меню, яке містить кнопки «Грати», «Налаштування», «Вийти» та інші важливі функції.

Основні елементи, такі як основні кнопки, які виконують функції, містяться в головному меню гри(Рис. 3.19):

– Play – почати гру;

– Settings – налаштувати гру;

– Quit – вийти з гри.

Рис. 3.19 Початкове меню гри

Під час гри гравець має можливість поставити гру на паузу зі збереженням поточного стану гри. За допомогою скрипта задаються умови, за яких можлива пауза у грі. Нижче наведено код реалізації даної можливості(Рис. 3.20).

```
public void PauseGame(bool status)
{
    //Якщо статус == true пазуа | якщо статус == false відміна пазуи
    pauseScreen.SetActive(status);

    //Коли статус паузи true змінює timescale на 0 (час зупиняеться)
    //Коли статус паузи false повертає до  1 (час рухаеться нормально)
    if (status)
        Time.timeScale = 0;
    else
        Time.timeScale = 1;
}
Ссылок: 0
public void SoundVolume()
{
    SoundManager.instance.ChangeSoundVolume(0.2f);
}
Ссылок: 0
public void MusicVolume()
{
    SoundManager.instance.ChangeMusicVolume(0.2f);
}
```

Рис. 3.20 Код для реалізації паузи гри

Функція PauseGame може активувати або деактивувати об’єкт PauseScreen, який є екраном паузи в грі. Якщо параметр статус дорівнює справжньому, екран паузи буде видимим, а якщо він дорівнює несправжньому, він буде прихованим. Після цього, в залежності від значення статусу, налаштовується часова шкала. Щоб зупинити час у грі, калібр часу встановлюється на 0, якщо статус рівний true. Якщо ж статус рівний неправдивий, шкала часу встановлюється на 1. Це дозволяє грі проходити нормально.

Це дозволяє контролювати стан паузи в грі, включаючи активацію або деактивацію екрану паузи та зупинення або відновлення часової шкали відповідно до стану паузи.

Гра завершується, коли головний персонаж вмирає, і з’являється меню, яке означає кінець гри(Рис. 3.21). У цьому меню гравець може розпочати гру знову (почати знову), повернутися до початкового меню (повернутися до меню) або вийти.

Рис. 3.21.1 Меню кінця гри

```
//Активація меню кінця гри
Ссылок: 1
public void GameOver()
{
    gameOverScreen.SetActive(true);
    SoundManager.instance.PlaySound(gameOverSound);
}

//Рестарт
Ссылок: 0
public void Restart()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}

//Головне меню
Ссылок: 0
public void MainMenu()
{
    SceneManager.LoadScene(0);
}

//Вийти
Ссылок: 0
public void Quit()
{
    Application.Quit(); //Закрити гру (працює тільки у готовому білді)

#if UNITY_EDITOR
    UnityEditor.EditorApplication.isPlaying = false;
#endif
}
```

Рис. 3.21.2. Код для реалізації кінцевого меню

Екран кінця гри активується за допомогою методу GameOver. Крім того, він може відтворювати звук завершення гри за допомогою методу PlaySound. Викликаючи метод LoadScene(), метод Restart() перезавантажує поточний рівень. Головне меню гри доступне за допомогою методу MainMenu().

Якщо використовується в редакторі, метод Quit() виходить з гри або з режиму гри.

### 3.5. Аналіз отриманого результату

У кінечному результаті отримано, гра складається з трьох рівнів, що вказує на те, що гравець поступово підвищує складність гри. В цій грі головний герой повинен подолати різноманітні перешкоди, наприклад уникнення стріл, шипів і інших ворожих предметів розміщених в різних місцях рівнів. Не забувайте, що головний герой також повинен перемогти ворожого нпс лицаря, який може нанести ушкодження вогняними шарами з великої дистанції, поки ви намагаєтесь подолати інші перешкоди(рис. 3.22.1 - 3.22.3)

Рис. 3.22.1. Перший рівень гри

Рис. 3.22.2. Другий рівень гри

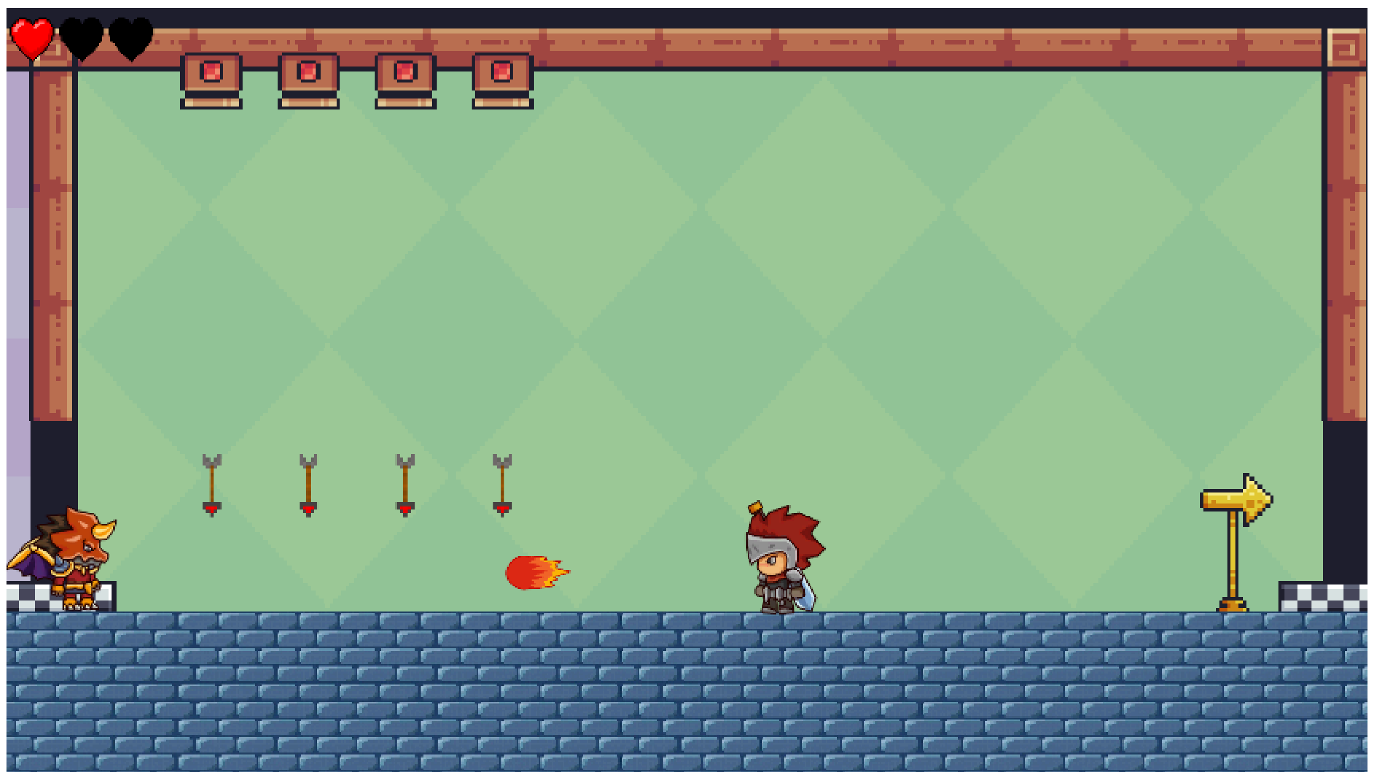

Рис. 3.22.3. Третій рівень гри

Після завершення процесу налаштування ігрових сцен, анімацій, звуків та інших компонентів починається процес створення збірки гри. Налаштування збірки дозволяють вибрати цільову платформу, наприклад Windows, MacOS або

Linux. У результаті вибору платформи створюється збірка гри, яка включає всі ресурси, скрипти та файли, необхідні для виконання гри.

Збірку гри, яку ви створили, можна поділитися з іншими користувачами за допомогою різних способів розповсюдження, наприклад, завантажуючи її на платформи постачальників контенту або розповсюджуючи через Інтернет. Це дозволяє гравцям насолоджуватися грою на своїх пристроях, а також ділитися цим задоволенням з іншими людьми.

Для подальшого розвитку та покращення гри можна розглянути наступні етапи:

– Додавання нових рівнів, перешкод, ворогів, вмінь головного героя;

– Поліпшення графічної якості гри та звукових ефектів;

– Створення кросплатформених версій гри;

– Розповсюдження на таких платформах, як Steam, App Store, Play Store.

– Збір відгуків користувачів з метою аналізу та покращення гри.

## ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі було розглянуто реалізація проекту відповідно до визначеної теми. Сценарій і перша концепція гри послужили основою для подальшої розробки. Визначення цільової аудиторії дозволило зосередитися на потребах і бажаннях гравців.

Привабливе середовище для гравців було ретельно розроблено, включаючи спрайти для головного героя, інтерфейс, ворогів та звукове супроводження.

Розробка фізичних характеристик об’єктів, їх руху, анімації та звукових ефектів були частиною етапу проектування гри. Для забезпечення плавного та реалістичного геймплею ці елементи були ретельно досліджені та реалізовані.

Окремо розглядалося створення меню для початку, паузи та завершення гри. Відповідні екрани та функції були розроблені, щоб гравці могли легко взаємодіяти з грою та змінювати її налаштування.

Аналіз отриманих результатів свідчить про успішне завершення проекту, який відповідає цілям і вимогам. Цільова аудиторія може отримати захоплюючий досвід від гри. Розширення геймплею, додавання нових рівнів і завдань, покращення графіки та оптимізація продукту для різних платформ є декількома потенційними шляхами розвитку та покращення гри.

## ВИСНОВКИ РОБОТИ

Розвиток комп’ютерних технологій продовжує розширювати можливості розробників ігор і породжувати нові тенденції в цій галузі.

Крім того, широкий спектр користувачів отримує доступ до індустрії ігор. Ігри складають різноманітну аудиторію, яка включає дітей, підлітків, дорослих і навіть літніх людей, які цікавляться іграми як засобом розваги, відпочинку, соціалізації, або чогось іншого.

У зв’язку з зростанням популярності відеоігор і їх значним прибутком ця галузь привертає увагу все більшої кількості талановитих розробників і інвесторів. Для створення високоякісних ігор потрібна креативна команда ентузіастів, які можуть створити чудовий геймплей, захоплюючі сюжети та незабутні емоції для гравців.

Враховуючи все сказане, можна зробити висновок, що сфера комп’ютерних ігор у жанрі 2D платформерів, розроблених на базі ігрового двигуна Unity, має значний потенціал для розвитку. Розробка таких ігор дозволяє задовольнити потреби різноманітної аудиторії гравців, одночасно розширюючи ринок ігор. Завдяки покращенню технологій, зростанню зацікавленості в іграх і доступності для розробки ігрових проектів ця галузь приваблює молодих спеціалістів і підприємців. Сучасні ігри є важливою частиною суспільства, оскільки вони не тільки допомагають розслабитися, але й виконують важливу функцію візуалізації інформації, соціалізації та навчання.

У процесі роботи було проведено аналіз предметної області, щоб дослідити основні елементи комп’ютерних ігор, такі як їхні ідеї, класифікація та основні етапи створення. Аналіз ефективності інструментів і технологій, використовуваних для розробки ігрових додатків, дозволив визначити найкращий набір інструментів для успішного завершення проекту, який є актуальним і працездатним.

Дослідження та аналіз подібних додатків дозволили зрозуміти їхні особливості та технічні рішення, які використовуються в сучасних іграх, що призвело до покращення розуміння та технічної компетентності. У результаті

цього аналізу були визначені як переваги, так і недоліки цих додатків. Результати цього аналізу були використані для подальшої розробки ігрового додатку.

Вибір мови програмування C# та ігрового двигуна Unity був зумовлений широким спектром можливостей, які пропонує проект. Зручний і ефективний інструментарій, який надає вибраний ігровий двигун Unity, дозволяє створювати графічне оформлення, фізичні характеристики об’єктів, їх рух і анімацію, а також створювати звукові ефекти. Вивчення також включає вивчення середовища розробки Microsoft Visual Studio, яке було обрано для подальшої реалізації проекту, щоб забезпечити ефективний розвиток і реалізацію проекту.

У процесі проектування біла розроблено концепцію та сценарій гри, а також визначено цільову аудиторію. Оформлення гри включало спрайти головного героя, ігрового оточення, інтерфейс і ворогів. Крім того, було розглянуто різні етапи проектування гри, такі як визначення фізичних характеристик об’єктів, їх руху, анімації та звукових ефектів. Для створення зручного користувацького інтерфейсу було розроблено меню для початку гри, паузи та завершення.

Після аналізу результатів можна сказати, що проект був успішно завершений і що його цілі були досягнуті. Гра має потенціал для подальшого розвитку. Можливість додати нові рівні, механіки гри та розширити геймплей є дуже привабливими перспективами. Розробка гри дала можливість розширити знання та навички в галузі розробки комп’ютерних ігор, зокрема щодо 2D платформерів і використання ігрового двигуна Unity.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

**1.** Братвейт, Б., & Шрайбер, І. (2009). "Виклики для дизайнерів ігор". Видавництво Cengage Learning.

**2.** Paris Buttfield-Addison. Unity Game Development Essentials. – O'Reilly Media. – 2019. – 216 s.

**3.** Мур, М. (2013). "Кулінарна книга розробки ігор на Unity 2D". Видавництво Packt Publishing Ltd.

**4.** Чепу, М., & Чепу, С. (2015). "Розробка ігор на Unity 5 для iOS, Android, Windows Phone, Tizen та інших платформ". Видавництво "Пітер".

**5.** Книга "Unity 2017 2D Game Development Projects: Create three interactive and engaging 2D games with Unity 2017" by Lauren S. Ferro та Francesco Sapio (Packt Publishing, 2018).

**6.** [Електронний ресурс] Steam – гра **Rain World**

**7.** [Електронний ресурс] Steam – гра **Hollow Knight**

**8.** [Електронний ресурс] Вікіпедія – **Комп’ютерна гра.**

**9.** [Електронний ресурс] UAPLAY – **Жанри комп’ютерних ігор**

**10.** [Електронний ресурс] Informatics – **Visual Studio**

**11.** [Електронний ресурс] Report.if – **Ігровий движок Unreal Engine**

**12.** [Електронний ресурс] Igrodom — **Ігровий движок CryEngine**

**13.** [Електронний ресурс] Unity Learn — **Ігровий движок Unity**

**14.** [Електронний ресурс] Docs.unity3d – **Box Collider 2D**

# ДОДАТКИ

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

3D – Тривимірне

2D – Двовимірне

Геймплей – Термін, яким називають особливості взаємодії людини з відеогрою.

ПК – Персональний комп'ютер

Мультиплеєр – Багатокористувацька гра

## ПРОГРАМНИЙ КОД

## 1. ГОЛОВНЕ МЕНЮ ГРИ

---

### 1.1 Кнопка початку гри

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

public class Menu : MonoBehaviour
{
   public int sceneNumber;
   public void Transition()
   {
      SceneManager.LoadScene(sceneNumber);
   }
}
```

---

### 1.2 Кнопка виходу з гри

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

public class Menu2 : MonoBehaviour
```

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

---

# 2. ПІДКОНТРОЛЬНИЙ ІГРОВИЙ ПЕРСОНАЖ

---

## 2.1 Атака ігрового персонажа

```
using UnityEngine;

public class PlayerAttack : MonoBehaviour
{
    [SerializeField] private float attackCooldown;
    [SerializeField] private Transform firePoint;
    [SerializeField] private GameObject[] fireballs;
    [SerializeField] private AudioClip fireballSound;

    private Animator anim;
    private PlayerMovement playerMovement;
    private float cooldownTimer = Mathf.Infinity;

    private void Awake()
    {
```

```
        anim = GetComponent<Animator>();

        playerMovement = GetComponent<PlayerMovement>();

    }


    private void Update()

    {

            if (Input.GetMouseButton(0) && cooldownTimer > attackCooldown &&
playerMovement.canAttack()

            && Time.timeScale > 0)

            Attack();


        cooldownTimer += Time.deltaTime;

    }


    private void Attack()

    {

        SoundManager.instance.PlaySound(fireballSound);

        anim.SetTrigger("attack");

        cooldownTimer = 0;


        fireballs[FindFireball()].transform.position = firePoint.position;

fireballs[FindFireball()].GetComponent<Projectile>().SetDirection(Mathf.Sign(transform.l
ocalScale.x));

    }

    private int FindFireball()

    {
```

```
            for (int i = 0; i < fireballs.Length; i++)
            {
                if (!fireballs[i].activeInHierarchy)
                    return i;
            }
            return 0;
        }
    }
```

---

**2.2 Рух ігрового персонажа**

```
using UnityEngine;


public class PlayerMovement : MonoBehaviour
{
    [Header("Movement Parameters")]
    [SerializeField] private float speed;
    [SerializeField] private float jumpPower;


    [Header("Coyote Time")]
    [SerializeField] private float coyoteTime; //How much time the player can hang in the air before jumping
    private float coyoteCounter; //How much time passed since the player ran off the edge


    [Header("Multiple Jumps")]
    [SerializeField] private int extraJumps;
```

```
private int jumpCounter;

[Header("Wall Jumping")]
[SerializeField] private float wallJumpX; //Horizontal wall jump force
[SerializeField] private float wallJumpY; //Vertical wall jump force

[Header("Layers")]
[SerializeField] private LayerMask groundLayer;
[SerializeField] private LayerMask wallLayer;

[Header("Sounds")]
[SerializeField] private AudioClip jumpSound;

private Rigidbody2D body;
private Animator anim;
private BoxCollider2D boxCollider;
private float wallJumpCooldown;
private float horizontalInput;

private void Awake()
{
   //Grab references for rigidbody and animator from object
   body = GetComponent<Rigidbody2D>();
   anim = GetComponent<Animator>();
   boxCollider = GetComponent<BoxCollider2D>();
```

```
}

private void Update()
{
    horizontalInput = Input.GetAxis("Horizontal");

    //Flip player when moving left-right
    if (horizontalInput > 0.01f)
        transform.localScale = Vector3.one;
    else if (horizontalInput < -0.01f)
        transform.localScale = new Vector3(-1, 1, 1);

    //Set animator parameters
    anim.SetBool("run", horizontalInput != 0);
    anim.SetBool("grounded", isGrounded());

    //Jump
    if (Input.GetKeyDown(KeyCode.Space))
        Jump();

    //Adjustable jump height
    if (Input.GetKeyUp(KeyCode.Space) && body.velocity.y > 0)
        body.velocity = new Vector2(body.velocity.x, body.velocity.y / 2);

    if (onWall())
```

```
            {
                body.gravityScale = 0;
                body.velocity = Vector2.zero;
            }
            else
            {
                body.gravityScale = 7;
                body.velocity = new Vector2(horizontalInput * speed, body.velocity.y);

                if (isGrounded())
                {
                    coyoteCounter = coyoteTime; //Reset coyote counter when on the ground
                    jumpCounter = extraJumps; //Reset jump counter to extra jump value
                }
                else
                    coyoteCounter -= Time.deltaTime; //Start decreasing coyote counter when not on the ground
            }
        }

        private void Jump()
        {
            if (coyoteCounter <= 0 && !onWall() && jumpCounter <= 0) return;
                //If coyote counter is 0 or less and not on the wall and don't have any extra jumps don't do anything
```

```
            SoundManager.instance.PlaySound(jumpSound);

            if (onWall())
                WallJump();
            else
            {
                if (isGrounded())
                    body.velocity = new Vector2(body.velocity.x, jumpPower);
                else
                {
                    //If not on the ground and coyote counter bigger than 0 do a normal jump
                    if (coyoteCounter > 0)
                        body.velocity = new Vector2(body.velocity.x, jumpPower);
                    else
                    {
                        if (jumpCounter > 0) //If we have extra jumps then jump and decrease the jump counter
                        {
                            body.velocity = new Vector2(body.velocity.x, jumpPower);
                            jumpCounter--;
                        }
                    }
                }

                //Reset coyote counter to 0 to avoid double jumps
                coyoteCounter = 0;
```

```
        }
    }

    private void WallJump()
    {
        body.AddForce(new Vector2(-Mathf.Sign(transform.localScale.x) * wallJumpX, wallJumpY));
        wallJumpCooldown = 0;
    }


    private bool isGrounded()
    {
            RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center, boxCollider.bounds.size, 0, Vector2.down, 0.1f, groundLayer);
        return raycastHit.collider != null;
    }
    private bool onWall()
    {
            RaycastHit2D raycastHit = Physics2D.BoxCast(boxCollider.bounds.center, boxCollider.bounds.size, 0, new Vector2(transform.localScale.x, 0), 0.1f, wallLayer);
        return raycastHit.collider != null;
    }
    public bool canAttack()
    {
        return horizontalInput == 0 && isGrounded() && !onWall();
    }
```

```
}
```

---

**2.3 Респавн ігрока після смерті**

```
using UnityEngine;

public class PlayerRespawn : MonoBehaviour
{
    [SerializeField] private AudioClip checkpoint;
    private Transform currentCheckpoint;
    private Health playerHealth;
    private UIManager uiManager;

    private void Awake()
    {
        playerHealth = GetComponent<Health>();
        uiManager = FindObjectOfType<UIManager>();
    }

    public void RespawnCheck()
    {
        if (currentCheckpoint == null)
        {
            uiManager.GameOver();
            return;
        }
```

```
            playerHealth.Respawn(); //Restore player health and reset animation
              transform.position = currentCheckpoint.position; //Move player to checkpoint location

            //Move the camera to the checkpoint's room

Camera.main.GetComponent<CameraController>().MoveToNewRoom(currentCheckpoint.parent);
        }
        private void OnTriggerEnter2D(Collider2D collision)
        {
            if (collision.gameObject.tag == "Checkpoint")
            {
                currentCheckpoint = collision.transform;
                SoundManager.instance.PlaySound(checkpoint);
                collision.GetComponent<Collider2D>().enabled = false;
                collision.GetComponent<Animator>().SetTrigger("activate");
            }
        }
    }
```

---

**2.4 Снаряд атаки ігрока**

```
using UnityEngine;

public class Projectile : MonoBehaviour
{
```

```
[SerializeField] private float speed;
private float direction;
private bool hit;
private float lifetime;

private Animator anim;
private BoxCollider2D boxCollider;

private void Awake()
{
    anim = GetComponent<Animator>();
    boxCollider = GetComponent<BoxCollider2D>();
}
private void Update()
{
    if (hit) return;
    float movementSpeed = speed * Time.deltaTime * direction;
    transform.Translate(movementSpeed, 0, 0);

    lifetime += Time.deltaTime;
    if (lifetime > 5) gameObject.SetActive(false);
}

private void OnTriggerEnter2D(Collider2D collision)
{
```

```
        hit = true;
        boxCollider.enabled = false;
        anim.SetTrigger("explode");

        if (collision.tag == "Enemy")
            collision.GetComponent<Health>()?.TakeDamage(1);
    }
    public void SetDirection(float _direction)
    {
        lifetime = 0;
        direction = _direction;
        gameObject.SetActive(true);
        hit = false;
        boxCollider.enabled = true;

        float localScaleX = transform.localScale.x;
        if (Mathf.Sign(localScaleX) != _direction)
            localScaleX = -localScaleX;

        transform.localScale = new Vector3(localScaleX, transform.localScale.y, transform.localScale.z);
    }
    private void Deactivate()
    {
        gameObject.SetActive(false);
    }
```

```
}
```

---

# 3. МЕХАНІКА ЗДОРОВ’Я

---

## 3.1 Здоров’я ігрока

---

```
using UnityEngine;
using System.Collections;

public class Health : MonoBehaviour
{
    [Header ("Health")]
    [SerializeField] private float startingHealth;
    public float currentHealth { get; private set; }
    private Animator anim;
    private bool dead;

    [Header("iFrames")]
    [SerializeField] private float iFramesDuration;
    [SerializeField] private int numberOfFlashes;
    private SpriteRenderer spriteRend;

    [Header("Components")]
    [SerializeField] private Behaviour[] components;
    private bool invulnerable;
```

```
[Header("Death Sound")]
[SerializeField] private AudioClip deathSound;
[SerializeField] private AudioClip hurtSound;

private void Awake()
{
    currentHealth = startingHealth;
    anim = GetComponent<Animator>();
    spriteRend = GetComponent<SpriteRenderer>();
}
public void TakeDamage(float _damage)
{
    if (invulnerable) return;
    currentHealth = Mathf.Clamp(currentHealth - _damage, 0, startingHealth);

    if (currentHealth > 0)
    {
        anim.SetTrigger("hurt");
        StartCoroutine(Invunerability());
        SoundManager.instance.PlaySound(hurtSound);
    }
    else
    {
        if (!dead)
```

```
        {
            //Deactivate all attached component classes
            foreach (Behaviour component in components)
                component.enabled = false;

            anim.SetBool("grounded", true);
            anim.SetTrigger("die");

            dead = true;
            SoundManager.instance.PlaySound(deathSound);
        }
    }
}
public void AddHealth(float _value)
{
    currentHealth = Mathf.Clamp(currentHealth + _value, 0, startingHealth);
}
private IEnumerator Invunerability()
{
    invulnerable = true;
    Physics2D.IgnoreLayerCollision(10, 11, true);
    for (int i = 0; i < numberOfFlashes; i++)
    {
        spriteRend.color = new Color(1, 0, 0, 0.5f);
        yield return new WaitForSeconds(iFramesDuration / (numberOfFlashes * 2));
```

```
            spriteRend.color = Color.white;
            yield return new WaitForSeconds(iFramesDuration / (numberOfFlashes * 2));
        }
        Physics2D.IgnoreLayerCollision(10, 11, false);
        invulnerable = false;
    }
    private void Deactivate()
    {
        gameObject.SetActive(false);
    }

    //Respawn
    public void Respawn()
    {
        AddHealth(startingHealth);
        anim.ResetTrigger("die");
        anim.Play("Idle");
        StartCoroutine(Invunerability());
        dead = false;

        //Activate all attached component classes
        foreach (Behaviour component in components)
            component.enabled = true;
    }
}
```

---

## 3.2 Відображення здоров’я

---

```
using UnityEngine;
using UnityEngine.UI;


public class Healthbar : MonoBehaviour
{
   [SerializeField] private Health playerHealth;
   [SerializeField] private Image totalhealthBar;
   [SerializeField] private Image currenthealthBar;


   private void Start()
   {
      totalhealthBar.fillAmount = playerHealth.currentHealth / 10;
   }
   private void Update()
   {
      currenthealthBar.fillAmount = playerHealth.currentHealth / 10;
   }
}
```

---

## 3.3 Збір здоров’я

```
using UnityEngine;
```

```
public class HealthCollectible : MonoBehaviour
{
    [SerializeField] private float healthValue;
    [SerializeField] private AudioClip pickupSound;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            SoundManager.instance.PlaySound(pickupSound);
            collision.GetComponent<Health>().AddHealth(healthValue);
            gameObject.SetActive(false);
        }
    }
}
```

---

## 4. Ворог та загрози

---

### 4.1 Атака врага

```
using UnityEngine;

public class RangedEnemy : MonoBehaviour
{
    [Header("Attack Parameters")]
```

```
[SerializeField] private float attackCooldown;
[SerializeField] private float range;
[SerializeField] private int damage;

[Header("Ranged Attack")]
[SerializeField] private Transform firepoint;
[SerializeField] private GameObject[] fireballs;

[Header("Collider Parameters")]
[SerializeField] private float colliderDistance;
[SerializeField] private BoxCollider2D boxCollider;

[Header("Player Layer")]
[SerializeField] private LayerMask playerLayer;
private float cooldownTimer = Mathf.Infinity;

[Header("Fireball Sound")]
[SerializeField] private AudioClip fireballSound;

//References
private Animator anim;
private EnemyPatrol enemyPatrol;

private void Awake()
{
```

```
    anim = GetComponent<Animator>();
    enemyPatrol = GetComponentInParent<EnemyPatrol>();
}

private void Update()
{
    cooldownTimer += Time.deltaTime;

    //Attack only when player in sight?
    if (PlayerInSight())
    {
        if (cooldownTimer >= attackCooldown)
        {
            cooldownTimer = 0;
            anim.SetTrigger("rangedAttack");
        }
    }

    if (enemyPatrol != null)
        enemyPatrol.enabled = !PlayerInSight();
}

private void RangedAttack()
{
    SoundManager.instance.PlaySound(fireballSound);
```

```
            cooldownTimer = 0;

            fireballs[FindFireball()].transform.position = firepoint.position;

fireballs[FindFireball()].GetComponent<EnemyProjectile>().ActivateProjectile();

        }

        private int FindFireball()

        {

            for (int i = 0; i < fireballs.Length; i++)

            {

                if (!fireballs[i].activeInHierarchy)

                    return i;

            }

            return 0;

        }



        private bool PlayerInSight()

        {

            RaycastHit2D hit =

                    Physics2D.BoxCast(boxCollider.bounds.center + transform.right * range * transform.localScale.x * colliderDistance,

                    new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y, boxCollider.bounds.size.z),

                0, Vector2.left, 0, playerLayer);



            return hit.collider != null;

        }

        private void OnDrawGizmos()
```

```
    {
        Gizmos.color = Color.red;
        Gizmos.DrawWireCube(boxCollider.bounds.center + transform.right * range * transform.localScale.x * colliderDistance,
            new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y, boxCollider.bounds.size.z));
    }
}
```

---

**4.2 Снаряд атаки ворога**

```
using UnityEngine;


public class EnemyFireballHolder : MonoBehaviour
{
    [SerializeField] private Transform enemy;


    private void Update()
    {
        transform.localScale = enemy.localScale;
    }
}
```

---

**4.3 Пастка зі стрілами**

```
using UnityEngine;


public class ArrowTrap : MonoBehaviour
```

```
{
    [SerializeField] private float attackCooldown;
    [SerializeField] private Transform firePoint;
    [SerializeField] private GameObject[] arrows;
    private float cooldownTimer;

    [Header("SFX")]
    [SerializeField] private AudioClip arrowSound;

    private void Attack()
    {
        cooldownTimer = 0;

        SoundManager.instance.PlaySound(arrowSound);
        arrows[FindArrow()].transform.position = firePoint.position;
        arrows[FindArrow()].GetComponent<EnemyProjectile>().ActivateProjectile();
    }
    private int FindArrow()
    {
        for (int i = 0; i < arrows.Length; i++)
        {
            if (!arrows[i].activeInHierarchy)
                return i;
        }
        return 0;
```

```
    }
    private void Update()
    {
        cooldownTimer += Time.deltaTime;

        if (cooldownTimer >= attackCooldown)
            Attack();
    }
}
```

---

**4.4 Вогнева пастка**

```
using UnityEngine;
using System.Collections;

public class Firetrap : MonoBehaviour
{
    [SerializeField] private float damage;

    [Header("Firetrap Timers")]
    [SerializeField] private float activationDelay;
    [SerializeField] private float activeTime;
    private Animator anim;
    private SpriteRenderer spriteRend;

    [Header("SFX")]
```

```
[SerializeField] private AudioClip firetrapSound;

private bool triggered; //when the trap gets triggered
private bool active; //when the trap is active and can hurt the player

private Health playerHealth;

private void Awake()
{
    anim = GetComponent<Animator>();
    spriteRend = GetComponent<SpriteRenderer>();
}

private void Update()
{
    if (playerHealth != null && active)
        playerHealth.TakeDamage(damage);
}

private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player")
    {
        playerHealth = collision.GetComponent<Health>();
```

```
            if (!triggered)
                StartCoroutine(ActivateFiretrap());


            if (active)
                collision.GetComponent<Health>().TakeDamage(damage);
        }
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.tag == "Player")
            playerHealth = null;
    }
    private IEnumerator ActivateFiretrap()
    {
        //turn the sprite red to notify the player and trigger the trap
        triggered = true;
        spriteRend.color = Color.red;


        //Wait for delay, activate trap, turn on animation, return color back to normal
        yield return new WaitForSeconds(activationDelay);
        SoundManager.instance.PlaySound(firetrapSound);
        spriteRend.color = Color.white; //turn the sprite back to its initial color
        active = true;
        anim.SetBool("activated", true);
```

```
        //Wait until X seconds, deactivate trap and reset all variables and animator
        yield return new WaitForSeconds(activeTime);
        active = false;
        triggered = false;
        anim.SetBool("activated", false);
    }
}
```

---

**4.5 Ворожий снаряд**

```
using UnityEngine;


public class EnemyProjectile : EnemyDamage
{
    [SerializeField] private float speed;
    [SerializeField] private float resetTime;
    private float lifetime;
    private Animator anim;
    private BoxCollider2D coll;


    private bool hit;


    private void Awake()
    {
        anim = GetComponent<Animator>();
        coll = GetComponent<BoxCollider2D>();
```

```
}

public void ActivateProjectile()
{
    hit = false;
    lifetime = 0;
    gameObject.SetActive(true);
    coll.enabled = true;
}
private void Update()
{
    if (hit) return;
    float movementSpeed = speed * Time.deltaTime;
    transform.Translate(movementSpeed, 0, 0);

    lifetime += Time.deltaTime;
    if (lifetime > resetTime)
        gameObject.SetActive(false);
}

private void OnTriggerEnter2D(Collider2D collision)
{
    hit = true;
    base.OnTriggerEnter2D(collision); //Execute logic from parent script first
    coll.enabled = false;
```

```
        if (anim != null)
            anim.SetTrigger("explode"); //When the object is a fireball explode it
        else
            gameObject.SetActive(false); //When this hits any object deactivate arrow
    }
    private void Deactivate()
    {
        gameObject.SetActive(false);
    }
}
```

---

## 5. КІМНАТИ

---

### 5.1 Кімната

```
using UnityEngine;

public class Room : MonoBehaviour
{
    [SerializeField] private GameObject[] enemies;
    private Vector3[] initialPosition;

    private void Awake()
    {
        //Save the initial positions of the enemies
```

```
        initialPosition = new Vector3[enemies.Length];
        for (int i = 0; i < enemies.Length; i++)
        {
            if(enemies[i] != null)
                initialPosition[i] = enemies[i].transform.position;
        }

        //Deactivate rooms
        if (transform.GetSiblingIndex() != 0)
            ActivateRoom(false);
    }
    public void ActivateRoom(bool _status)
    {
        //Activate/deactivate enemies
        for (int i = 0; i < enemies.Length; i++)
        {
            if (enemies[i] != null)
            {
                enemies[i].SetActive(_status);
                enemies[i].transform.position = initialPosition[i];
            }
        }
    }
}
```

**5.2 Двері**

```
using UnityEngine;


public class Door : MonoBehaviour
{
    [SerializeField] private Transform previousRoom;
    [SerializeField] private Transform nextRoom;
    [SerializeField] private CameraController cam;


    private void Awake()
    {
        cam = Camera.main.GetComponent<CameraController>();
    }


    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            if (collision.transform.position.x < transform.position.x)
            {
                cam.MoveToNewRoom(nextRoom);
                nextRoom.GetComponent<Room>().ActivateRoom(true);
                previousRoom.GetComponent<Room>().ActivateRoom(false);
            }
            else
```

```
            {
                cam.MoveToNewRoom(previousRoom);
                previousRoom.GetComponent<Room>().ActivateRoom(true);
                nextRoom.GetComponent<Room>().ActivateRoom(false);
            }
        }
    }
}
```

---

# 6. ЯДРО

---

## 6.1 Камера

```
using UnityEngine;

public class CameraController : MonoBehaviour
{
    //Room camera
    [SerializeField] private float speed;
    private float currentPosX;
    private Vector3 velocity = Vector3.zero;

    //Follow player
    [SerializeField] private Transform player;
    [SerializeField] private float aheadDistance;
```

```
    [SerializeField] private float cameraSpeed;

    private float lookAhead;


    private void Update()

    {

        //Room camera

        transform.position = Vector3.SmoothDamp(transform.position, new Vector3(currentPosX, transform.position.y, transform.position.z), ref velocity, speed);


        //Follow player

        //transform.position = new Vector3(player.position.x + lookAhead, transform.position.y, transform.position.z);

        //lookAhead = Mathf.Lerp(lookAhead, (aheadDistance * player.localScale.x), Time.deltaTime * cameraSpeed);

    }


    public void MoveToNewRoom(Transform _newRoom)

    {

        print("here");

        currentPosX = _newRoom.position.x;

    }

}
```

---

**6.2 Менеджер завантажень**

```
using UnityEngine;

using UnityEngine.SceneManagement;
```

```
public class LoadingManager : MonoBehaviour
{
    public static LoadingManager instance { get; private set; }

    private void Awake()
    {
        //Keep this object even when we go to new scene
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        //Destroy duplicate gameobjects
        else if (instance != null && instance != this)
            Destroy(gameObject);
    }

    public void LoadCurrentLevel()
    {
        int currentLevel = PlayerPrefs.GetInt("currentLevel", 1);
        SceneManager.LoadScene(currentLevel);
    }
    public void Restart()
    {
        //SceneManager.LoadScene(currentLevel);
```

```
    }
}
```

---

**6.3 Звуковий менеджер**

```
using UnityEngine;

public class SoundManager : MonoBehaviour
{
    public static SoundManager instance { get; private set; }
    private AudioSource soundSource;
    private AudioSource musicSource;

    private void Awake()
    {
        soundSource = GetComponent<AudioSource>();
        musicSource = transform.GetChild(0).GetComponent<AudioSource>();

        //Keep this object even when we go to new scene
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        //Destroy duplicate gameobjects
        else if (instance != null && instance != this)
```

```
        Destroy(gameObject);

    //Assign initial volumes
    ChangeMusicVolume(0);
    ChangeSoundVolume(0);
}
public void PlaySound(AudioClip _sound)
{
    soundSource.PlayOneShot(_sound);
}

public void ChangeSoundVolume(float _change)
{
    ChangeSourceVolume(1, "soundVolume", _change, soundSource);
}
public void ChangeMusicVolume(float _change)
{
    ChangeSourceVolume(0.3f, "musicVolume", _change, musicSource);
}

private void ChangeSourceVolume(float baseVolume, string volumeName, float change, AudioSource source)
{
    //Get initial value of volume and change it
    float currentVolume = PlayerPrefs.GetFloat(volumeName, 1);
    currentVolume += change;
```

```
        //Check if we reached the maximum or minimum value
        if (currentVolume > 1)
            currentVolume = 0;
        else if (currentVolume < 0)
            currentVolume = 1;

        //Assign final value
        float finalVolume = currentVolume * baseVolume;
        source.volume = finalVolume;

        //Save final value to player prefs
        PlayerPrefs.SetFloat(volumeName, currentVolume);
    }
}
```

---