Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

Створення 2D гри

Кваліфікаційна робота

Рівень вищої освіти - перший (бакалаврський)

Виконав:
студент 4 курсу, 407 групи
Березюк Максим Олексійович
Керівник:
кандидат фізико-математичних наук
Перцов А.С.

До захисту допущено
на засіданні кафедри
математичного моделювання
протокол № _ від _____ 2023р.
Зав. кафедри _____ проф. Черевко І.М.

Чернівці – 2024

Анотація

У даній роботі розглядається процес створення 2D гри в середовищі Unity з використанням мови програмування C#. Описані основні етапи розробки, включаючи створення та обробку графіки у Photoshop. Обговорюються технічні аспекти кодування, анімації та інтеграції графічних ресурсів у середовищі Unity.

Ключові слова:Unity, C#, 2D гра, Photoshop,розробка ігор, графіка, анімація, інтеграція.

Annotation

This article examines the process of creating a 2D game in the Unity environment using the C# programming language. The main development stages are described, including the creation and processing of graphics in Photoshop . Technical aspects of coding, animation, and the integration of graphical assets into Unity are discussed. The material is intended for beginner developers and those interested in the gaming industry.

Keywords: Unity, C#, 2D game, Photoshop, game development, graphics, animation, integration..

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____Березюк М.О.
(підпис)

ЗМІСТ

ВСТУП

Актуальність теми створення 2D-ігор у сучасних середовищах розробки ігор , таких як Unity, незаперечна. Зі зростаючою популярністю мобільних і інді - ігор існує високий попит на розробників , які можуть створювати високоякісні і захоплюючі ігри з використанням новітніх інструментів.Unity - одна з найпопулярніших платформ для розробки ігор завдяки зручності , широкому функціоналу та підтримці різних мов програмування, включаючи C#.

Практичне значення цієї роботи полягає у створенні 2D гри, яку можна буде розвивати та вдосконалювати для майбутнього релізу. Цей проект демонструє застосування різних інструментів і технологій, таких як Photoshop для створення графіки, та C# для програмування логіки гри.

Мета даної роботи полягає в тому, щоб розробити 2D гру в Unity, яка буде готова до подальшого розвитку та вдосконалення для релізу. Це дозволить отримати практичні навички в розробці ігор, створенні графіки та інтеграції різних елементів у єдиний проект, що сприятиме професійному зростанню та розширенню портфоліо.

Структура роботи. Кваліфікаційна робота складається зі вступу, двох розділів, висновків, списку використаних джерел та додатків.

-описати концепцію та функціонал 2D платформера;

-дослідити схожі проекти для визначення ключових елементів успіху;

-описати використання інструменту Photoshop для створення графіки;

-виявити особливості програмування гри на мові C# у середовищі Unity;

-розробити прототип 2D гри, включаючи основні механіки та інтеграцію графічних елементів, та описати процес розробки гри.

Розділ 1 Опис використаних Технологій

1.1 Середовище Unity

Unity — це потужний і гнучкий кросплатформний ігровий рушій, який використовується для створення ігор та інтерактивних додатків, що працюють на різних платформах, включаючи Windows, macOS, Linux, iOS, Android, а також ігрові консолі та VR/AR пристрої. Unity дозволяє розробникам створювати як 2D, так і 3D проекти, пропонуючи широкий спектр інструментів та можливостей.

Основна перевага Unity полягає у його візуальному редакторі, який дозволяє легко перетягувати об'єкти у світі гри, налаштовувати їх властивості та створювати складні ігрові сцени без безпосереднього написання коду. Редактор підтримує різноманітні формати асетів, такі як 3D моделі, текстури, анімації та звуки, що спрощує інтеграцію зовнішніх ресурсів.

Unity має вбудовану підтримку фізики через фізичний движок, що дозволяє створювати реалістичну взаємодію об'єктів у грі. Також рушій включає потужну систему анімації, яка допомагає анімувати персонажів та інші елементи ігрового світу.

Для розробки логіки ігор в Unity використовується мова програмування C#, що робить платформу доступною для широкого кола розробників. Ця мова забезпечує гнучкість та потужні можливості для створення складних ігрових механік.

Unity також пропонує багатий набір інструментів для оптимізації та поліпшення продуктивності ігор, включаючи профайлер, систему віртуальної реальності та інструменти для мобільної оптимізації. Це дозволяє розробникам створювати високопродуктивні ігри, оптимізовані для різноманітного обладнання.

Крім того, спільнота Unity є однією з найбільших та найактивніших серед ігрових розробників, надаючи доступ до величезної кількості ресурсів, які включають навчальні матеріали, готові асети та плагіни, що ще більше спрощує процес розробки ігор.

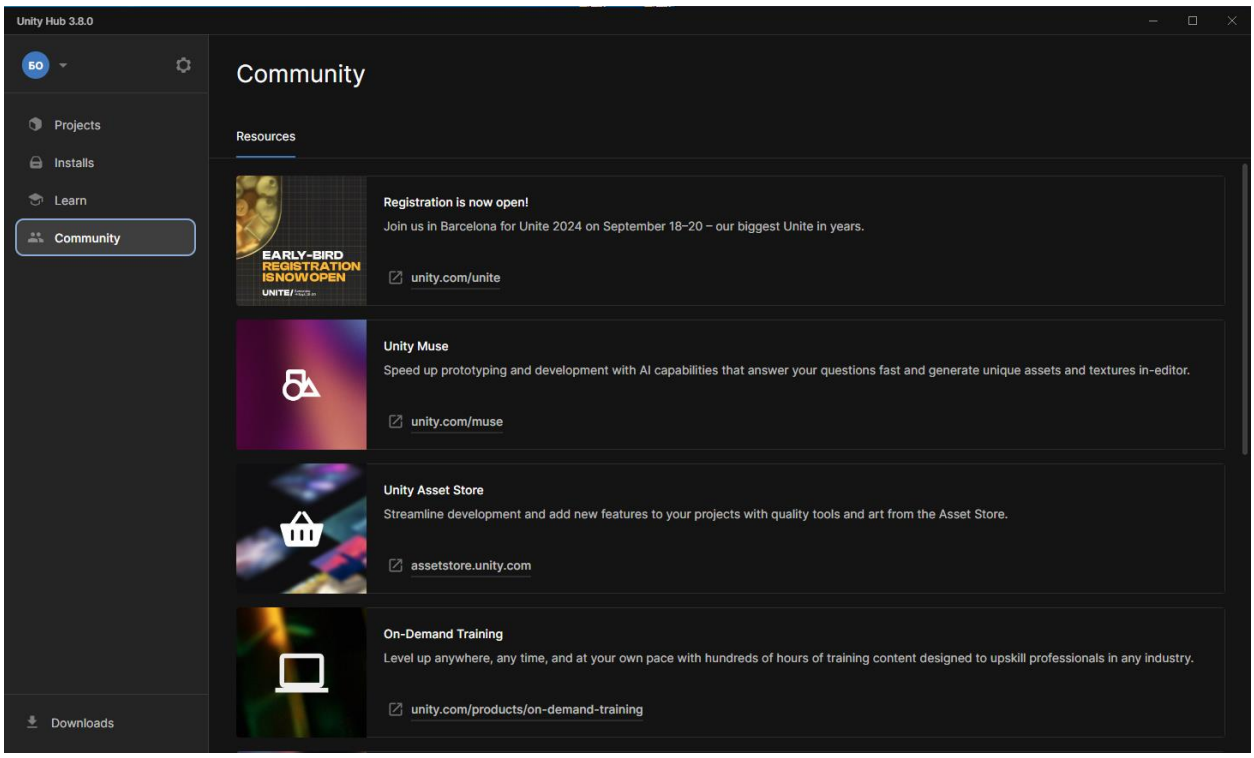

Рис. 1.1 Unity Hub

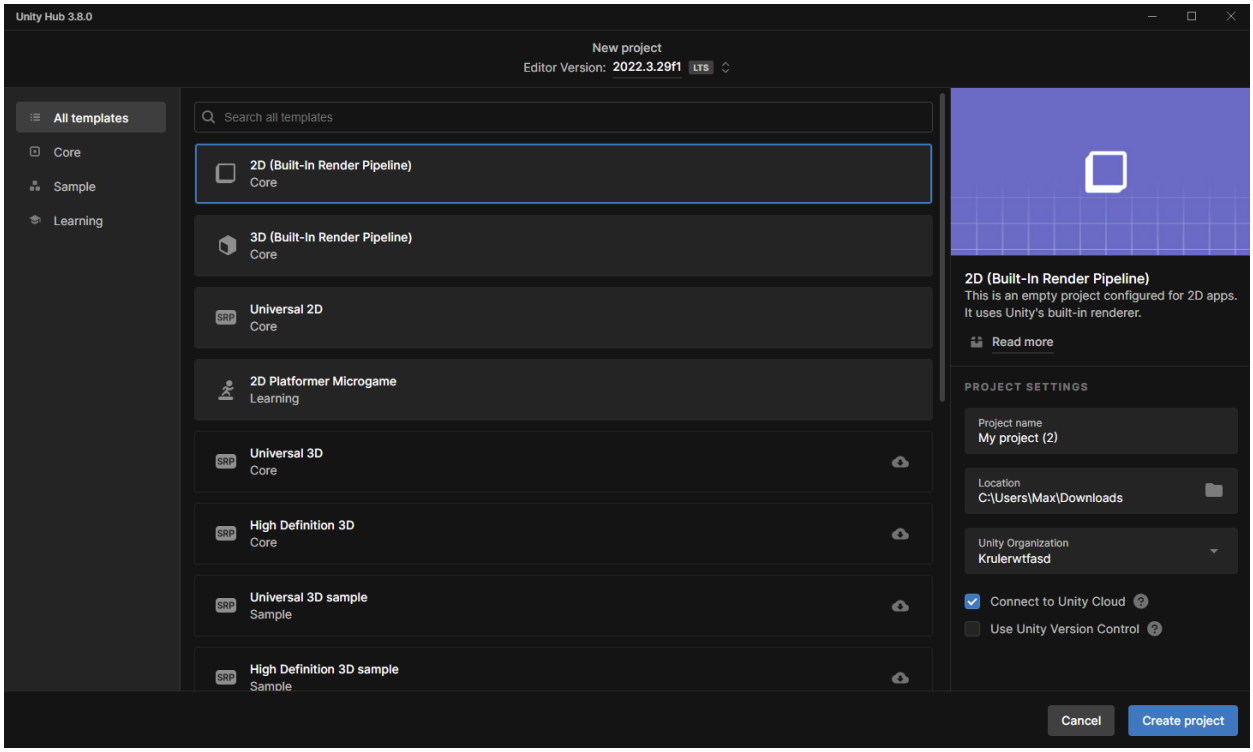

Рис. 1.2 Unity Hub

1.2 Недоліки Unity

Unity — це дуже популярний ігровий рушій, але, як і будь-яка технологія, він не позбавлений певних недоліків. Один із значних викликів полягає у високих вимогах до системних ресурсів. Розробники часто

зіштовхуються з тим, що їхні проекти в Unity потребують значної кількості оперативної пам'яті та обчислювальної потужності, особливо при роботі з великими і складними сценами. Це може вплинути на продуктивність, особливо на менш потужних системах.

Крім того, хоча Unity підтримує багатоплатформенність, існують випадки, коли перенесення проектів між різними платформами може бути непростим процесом. Це часто пов'язано з версійними конфліктами або несумісністю залежностей і плагінів. Оновлення Unity також можуть призводити до несподіваних проблем у вже існуючих проектах, вимагаючи додаткових зусиль для виправлення помилок або адаптації коду під нову версію рушія.

Фінансовий бік використання Unity також може стати викликом для незалежних розробників або малого бізнесу. Хоча базова версія рушія є безкоштовною, повнофункціональна версія Unity та деякі додаткові інструменти та плагіни можуть вимагати платної підписки або одноразової покупки.

Щодо документації, то хоча вона досить обширна і підтримується спільнотою, іноді вона може бути неповною або не надто зрозумілою, особливо у випадку з розробкою складних функціональностей або роботою з новими функціями рушія.

Також попри можливості оптимізації для мобільних пристроїв, деякі розробники можуть зіткнутися з проблемами, пов'язаними з досягненням високої продуктивності на обмежених мобільних платформах, що може вимагати значних зусиль для фінтюнінгу проекту під обмеження цих пристроїв.

1.3 Робота з середовищем Unity

Розпочати роботу в Unity можна з встановлення самого рушія. Після завантаження та інсталяції Unity через офіційний вебсайт або Unity Hub, ви

зможете створити новий проект або відкрити існуючий. Unity Hub допомагає керувати версіями рушія, проектами та додатковими модулями для різних платформ.

Першим кроком після створення нового проекту є ознайомлення з основними компонентами інтерфейсу Unity:

- Scene View дозволяє візуально маніпулювати об'єктами в 3D або 2D просторі.

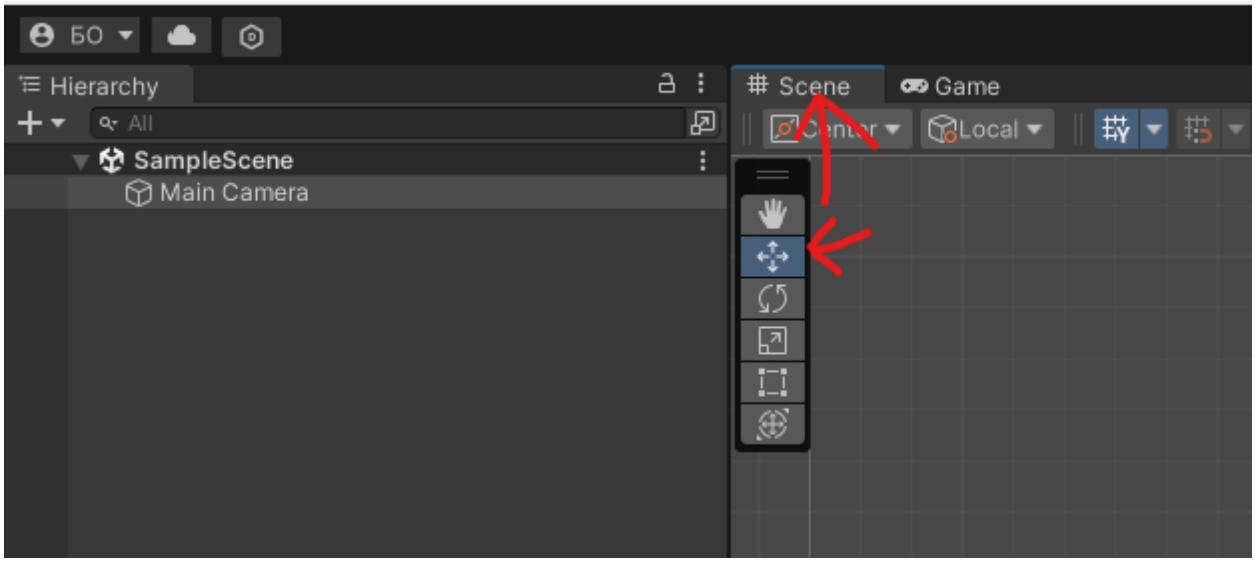

Рис. 1.3 Scene View

- Game View показує, як ваша гра буде виглядати в реальному часі при запуску.

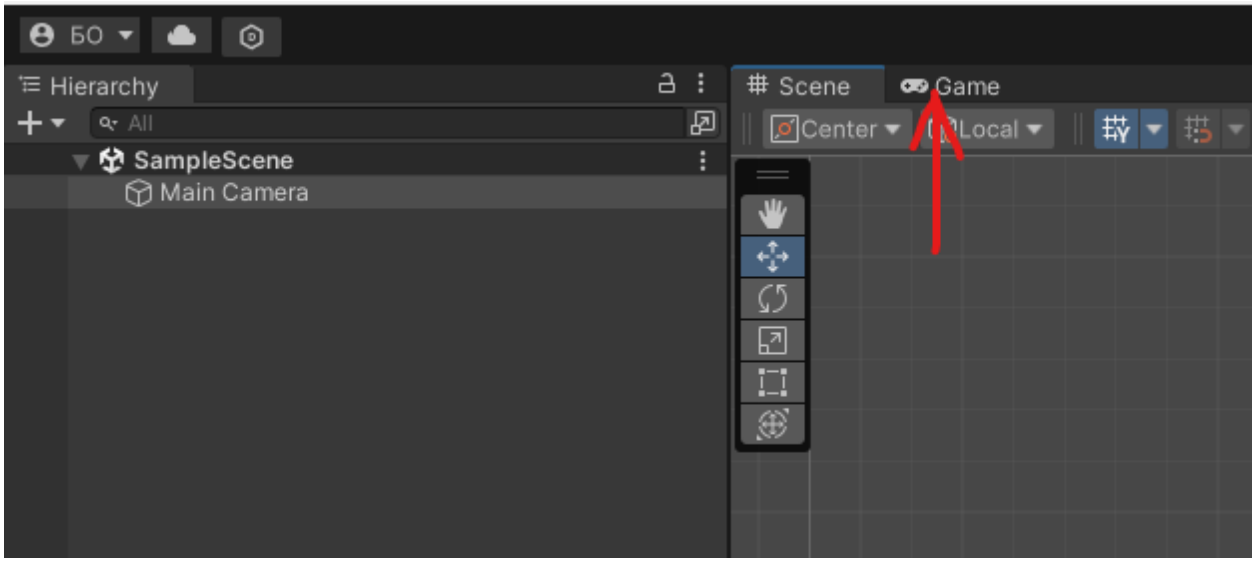

Рис. 1.4 Game View

- Hierarchy є панеллю, де відображаються всі об'єкти у сцені, дозволяючи легко керувати їх структурою.

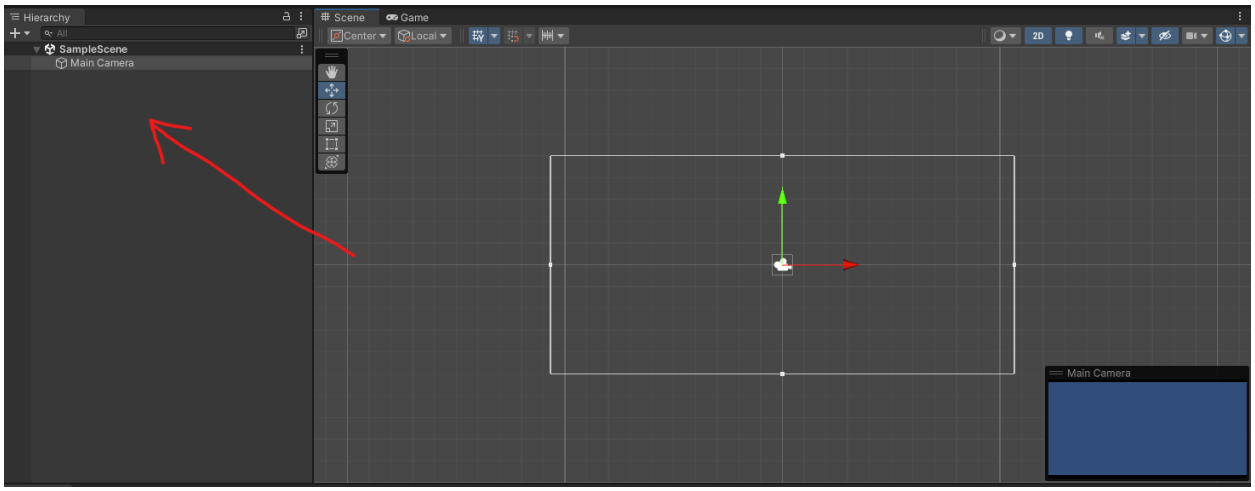

Рис. 1.5 Hierarchy

- Inspector використовується для редагування властивостей обраного об'єкта.

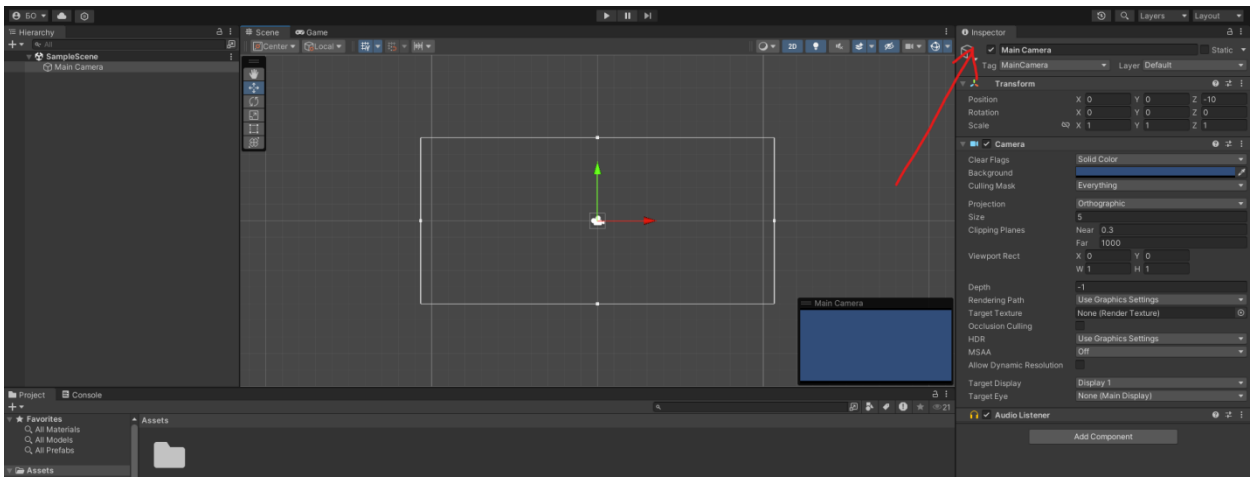

Рис. 1.6 Inspector

- Project Window організовує всі файли, пов'язані з проектом, дозволяючи швидко знаходити та використовувати ресурси.

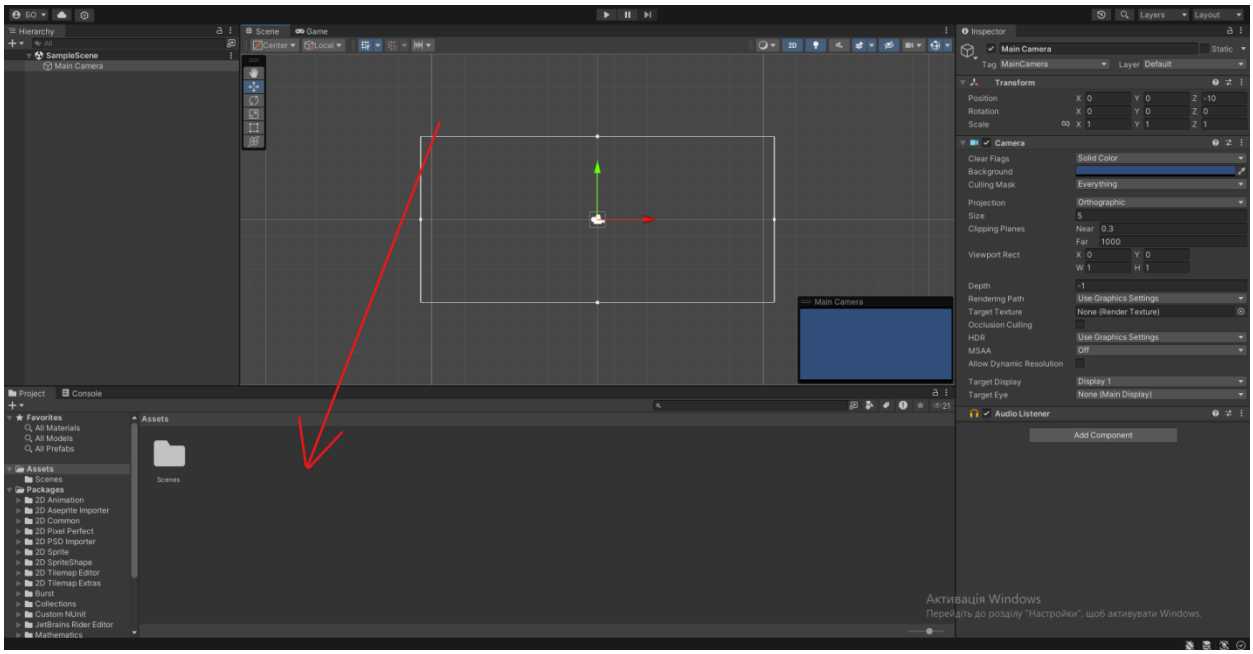

Рис. 1.7 Project Window

Щоб почати розробку, спершу потрібно створити різні об'єкти в ігровій сцені. В Unity це можуть бути 3D моделі, спрайти, камери, світла та інші компоненти. Кожен об'єкт можна налаштовувати за допомогою компонентів, таких як рендерери, колайдери, скрипти тощо, які додаються через вкладку Inspector.

Unity також дозволяє створювати скрипти на мові C#, які можна прикріплювати до об'єктів, щоб надати їм індивідуальну поведінку. Програмування в Unity орієнтоване на об'єкти та їх взаємодії, що робить розробку інтуїтивною та доступною.

Для перевірки та відладки ігрових сценаріїв використовують вікно Game View, де можна запускати і тестувати ігру в режимі реального часу. Це ключовий інструмент для швидкого тестування змін та вдосконалення ігрового процесу.

1.4 Unity запуск коду : компоненти сценарію

Unity використовує компонентно-орієнтований підхід до розробки, де поведінка об'єктів визначається за допомогою скриптів, що додаються до них як компоненти. Скрипти в Unity пишуться на мові C# і є ключовими для додавання інтерактивності та динаміки в ігри або інші програми. Ось основні кроки та концепції, що відносяться до запуску коду у Unity через скрипти:

Створення скрипта

Для створення нового скрипта:

Клацніть правою кнопкою миші в області Project Window.

Виберіть "Create" > "C# Script".

Назвіть скрипт і натисніть Enter.

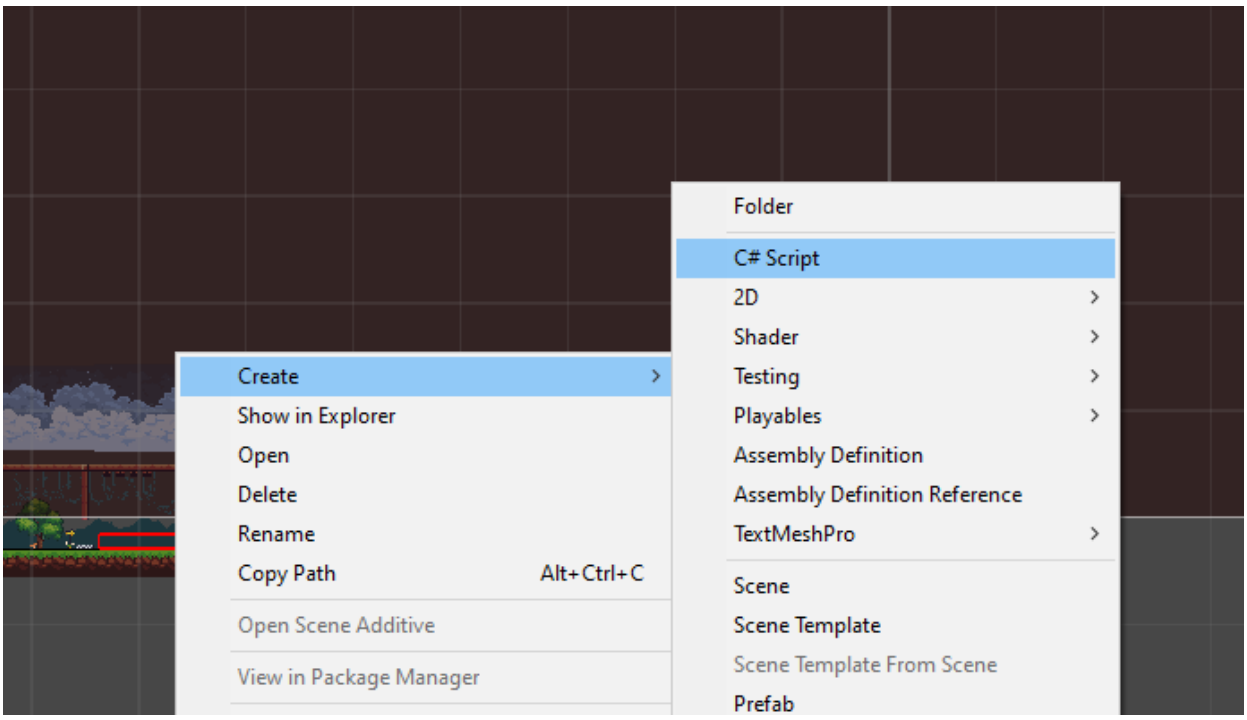

Рис. 1.8 Створення скрипта

Після створення, двічі клацніть на скрипті, щоб відкрити його в редакторі коду (наприклад, Visual Studio), який інтегрований з Unity.

Структура скрипта

Базовий скрипт містить два методи:

Start(): виконується один раз при старті скрипта. Використовується для ініціалізації змінних, налаштувань початкового стану об'єктів та інших задач, що вимагають одноразового виконання.

Update(): виконується щоразу, коли обробляється новий кадр. Це місце для коду, що реагує на вхідні дані від користувача, зміни в ігровому світі чи інші події, що мають відбуватись упродовж часу.

Додавання скрипта до об'єкта

Щоб скрипт почав виконуватися, його потрібно прикріпити до об'єкта у сцені:

1)Перетягніть скрипт з Project Window на об'єкт у Hierarchy Window.

2)Або ж виберіть об'єкт в Hierarchy, перейдіть до вкладки Inspector і перетягніть скрипт у розділ компонентів об'єкта.

Інтеракція скрипта з компонентами

Скрипти можуть взаємодіяти з іншими компонентами на тому ж об'єкті (наприклад, з Rigidbody для фізичних обчислень або з MeshRenderer для зміни візуальних характеристик). Це можна зробити за допомогою методу GetComponent. Наприклад:

```
void Start() {
   Rigidbody rb = GetComponent<Rigidbody>();
   rb.mass = 5; // Змінюємо масу Rigidbody
}
```

Запуск і тестування

Після прикріплення скрипта до об'єкта, запустіть сцену, натиснувши на кнопку Play у Unity. Тепер Unity буде виконувати код у скриптах, реагуючи на зміни у сцені та вводи користувач

1.5 Графіка

Unity підтримує різноманітні графічні ресурси, які можна використовувати для створення візуального вмісту в іграх та інших інтерактивних додатках. Ось деякі з основних типів графічних ресурсів, що підтримуються Unity:

1. Текстури

Текстури - це зображення, яке використовується для прикріплення до поверхонь об'єктів у вашій грі. Вони додають деталізацію, текстуру та кольоровість об'єктам. Текстури можуть бути застосовані до широкого спектру об'єктів, від стін будівель до поверхонь персонажів. Unity підтримує різні типи текстур, включаючи звичайні текстури, текстури нормалей для відтворення деталізації, текстури висоти для симуляції 3D ефектів, текстури емісії для випромінювання світла та інші.

2. Моделі

Моделі - це 3D об'єкти, які представляють собою геометричні форми і структури. Вони використовуються для створення об'єктів, персонажів та інших елементів у вашій грі. Моделі можуть бути створені у спеціалізованих програмах для моделювання, таких як Blender, Maya або 3ds Max, а потім імпортовані в Unity. Unity підтримує різні формати файлів для моделей, включаючи FBX, OBJ та деякі інші.

3. Спрайти

Спрайти - це 2D графічні об'єкти, які використовуються для створення ігор у 2D просторі. Вони широко використовуються для створення фонів, персонажів, предметів і інших елементів інтерфейсу. У Unity спрайти можуть бути представлені як звичайні текстури, але їх також можна анімувати та керувати через спеціалізовані компоненти, такі як Sprite Renderer.

4. Анімації

Анімації - це послідовність зображень або станів, які змінюються з часом. У Unity ви можете створювати анімації для як 2D, так і 3D об'єктів. Ви можете створити анімації руху, анімації зміни кольору, анімації зміни форми та багато іншого. Анімації використовуються для створення рухомих ефектів,

таких як ходьба персонажа, рух об'єктів у середовищі, ефекти вибухів та багато іншого.

5. Матеріали

Матеріали - це компоненти, які визначають візуальні властивості об'єктів у вашій грі. Вони використовуються для надання об'єктам кольору, текстури, відбиття світла, прозорості та інших властивостей. Матеріали дозволяють вам керувати тим, як ваші об'єкти виглядають у грі, і можуть бути застосовані до будь-якого об'єкта у вашій сцені.

6. Ефекти частинок

Ефекти частинок - це спеціальні графічні ефекти, які використовуються для створення реалістичних атмосферних ефектів у вашій грі. Вони використовуються для створення ефектів, таких як вогненні кулі, дим, дощ, вибухи та багато іншого. Ефекти частинок додають атмосферу та динаміку до вашої гри, роблячи її більш захоплюючою для гравців.

Ці різноманітні графічні ресурси дозволяють вам створити вражаючий візуальний контент для вашої гри в Unity.

Розділ 2 Програмна частина

2.1 Постановка задачі

Мета гри - розробити 2D гру в Unity, яка буде готова до подальшого розвитку та вдосконалення для релізу. Для того аби виконати задачу потрібно:

Створити основну сцена: main;

Створити префаби(шаблони) які будуть використовуватись у грі(спрайти або префаби беремо з Asset Store або створюємо самі за допомогою Photoshop) ;

Прописуємо логіку для головного героя(Player),ворога(Enemy),пасток(Traps),а також для підбираємих предметів(colectible items);

Додати звуки для гри(кроки,супровідна музика,вистріли та стрибка);

2.2 Опис гри

Гра має тільки одну сцену level1 яка відображає саму гру. Також є меню паузи.

При запуску гри перше що побачить користувач,це буде головна сцена level1 та почує фонову музику. Спочатку ми додаємо спрайт, далі ми додаємо на нього створену нами текстуру головного героя(Player).Та при натискані клавіші “Esc” в нас зявиться меню пазуи, де ми зможемо зменшити музику(Music) або зменшити гучність всього звуку(Volume) також там є кнопка Продовжити (“Resume”) та Вийти(”Quit”).

За допомогою Unity Asset Store та Photoshop було створено задні слої фону(WOODS , BUSH , Background1-4 , Space background та Brown) , земля(Ground) , вороги(enemy) , пасток(Traps).

Меню пазуи сховане з самого початку.Для його реалізації мені потрібно було в скрипті”UIManager” зрозуміти як зупинити гру при виклику самого меню.Для цього я створив умовний оператор(if) в якому прописав що при виклику меню паузи timeScale=0 , це значить що час зупиняється при вкилику цього меню.Для того аби в паузі була активна музика та я міг користуватись кнопками , я створив їх як окремий обєкт та вже на них додав звук при перелистувані та скріпти такі як “SoundManager,VolumeText та UIManager”.

Також юуло реалізоване меню смерті. Воно зявляється тільки в тому випадку коли гравець не дійшов до чек-поїнта(точки збереження).Воно створенно на основі меню паузи тому вони майже ідентичні.

Рис. 2.1 Меню смерті

Рис. 2.2 Меню паузи

Також я реалізував подвійний стрибок,чіпляння за стіни,атаку(вистріли) для свого персонажу.

-Для реалізації чіпляння я в скрипті “PlayerMovment” створив перевірку чи мій персонаж на землі чи він в повітрі. Та якщо він стикається з фізичним обєктом які не є ворогом або пасткою то він чіпляється за нього та може відстрибнути.

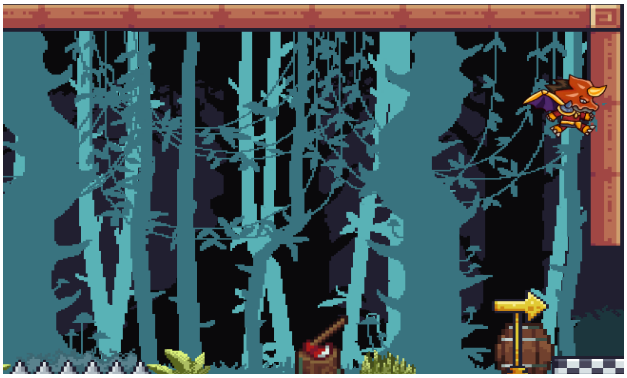

Рис. 2.3 Чіпляння за стіну

-Подвійний стрибок також був реалізований у скрипті “PlayerMovment”.

Рис. 2.4 Чіпляння за стіну

-Атака, а точніше вистріли були реалізовані у скрипті “PlayerAttack”. Для того аби не було такого що персонаж просто ходить та

стріляє. Я встановив активацію при нажатті на клавішу та додав таймер на атаку. Також додав швидкість з якою буде пересуватись наша вогняна куля.

Рис. 2.4 Атака

Але найважче в даному етапі для мене були анімації. Для того аби використовувати анімації в 2D грі нам потрібні заготовлені ассети під кожен рух. Для цього я звертаюсь за допомогою в Aset Store . Це сайт від розробників Unity де люди діляться або продають свої заготовки . Саме там я найшов свого персонажа та деяких ворогів. Після того як ми імпортували їх до себе в проект нам потрібно в Animator це все анімувати. Там ми підставляємо покадрово кожен наш спрайт. Якщо в нас є декілька анімацій для 1 персонажа то нам потрібно буде в Animtor-і надати структури нашим анімаціям.

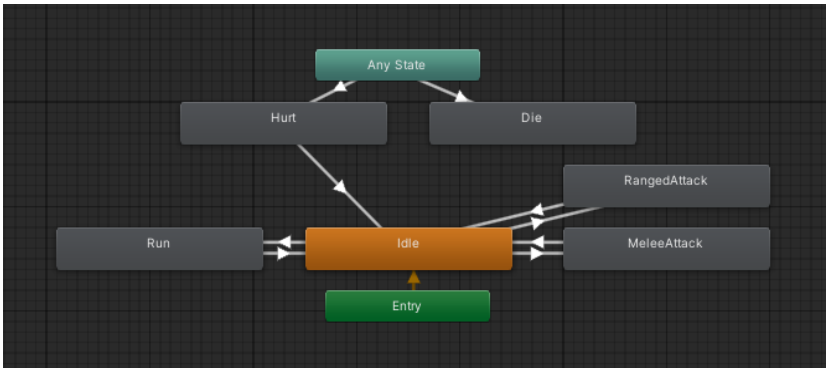

Рис. 2.5 Структура Animator для Player

В подальшому нам це знадобиться аби ці анімації(саме назви ,адже кожен блочок це окремо створена анімація) викликати у коді , для привязки під якусь дію або тригер.

Для ворога(Enemy) та пасток я реалізував патрулювання,атаку при заході в область,присліджування гравця, автоматичні регулярні вистріли та активація при взаємодії.

-Патрулювання.До даного персонажа я додав пару скриптів . А саме так і як Health,Melee Enemy,EnemyDamage. Перший скрипт “Health” нам потрібен для того аби у нашого ворога було здоров'я, а також в ньому використовується анімація смерті. Другий скрипт”Melee Enemy” нам потрібен саме для того щоб завдати рамки патрулювання(Рис 2.6) вони позначені як жовті таки сфери(в самій грі їх не видно).Також ми можемо замітити червоний прямокутник перед ворогом(також не видно) це область атаки . І третій скрипт”EnemyDamage” там ми прописуємо код, для того, щоб при зіткненні з головним героєм Player-у завдавалась шкода.

Рис. 2.6 Melee Enemy

-Атака при заході в область.Це новий тип Enemy, який стоїть на місті та стріляє в головного героя коли той заходить в поле видимості. Я його створив на основі Melee Enemy. Єдині відмінності від Melee це колір ,та замість ближньої атаки мечем , він стріляє в певній області.

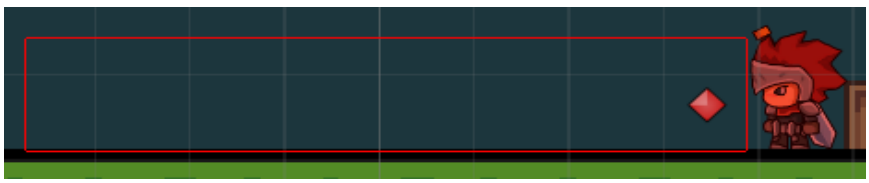

Рис. 2.7 Ranged Enemy

Тепер перейдемо до пасток. Я реалізував п'ять пасток а саме : Spikes,Firetrap,ArrowTrap,Saw,Spikehead.

-Spikes - це є звичайна пастка. Коли ви стаєте на неї вона завдає вам шкоду.

Рис. 2.8 Spikes

-Firetrap - це пастка тригер. Тобто коли головний герой взаємодіє з нею а саме торкається її, то через 0,25 секунд з'являється вогонь який завдає шкоду.

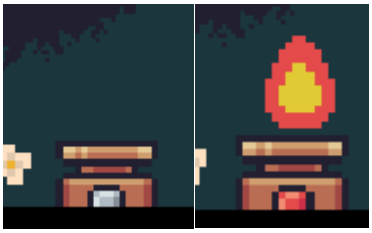

Рис. 2.9 Firetrap

-ArrowTrap- це пастка яка систематично випускає стріли з інтервалом 1 секунда.Для стріл я створив окремі об'єкти які з'являються з певної точки.

Рис. 2.10 ArrowTrap

-Saw - це пастка яка рухаєься по осі X в певній області .

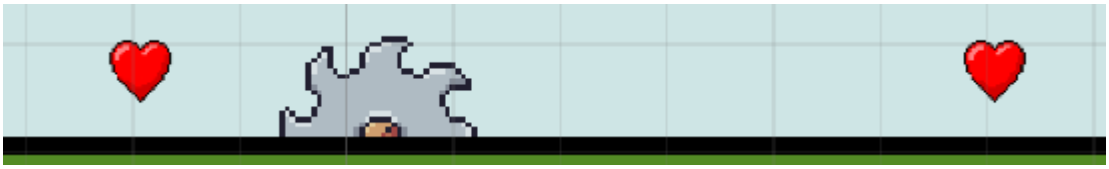

Рис. 2.11 Saw

-Spikehead - це пастка яка рухається за героєм якщо той перетнув певну лінію. Вони рухаються за ним до поки той не зазнає шкоди або не обійде.Їхнє поле зору це чотири напрямки верх низ вліво та вправо. Це була найважча реалізована пастка.Для цього мені було потрібно розрахувати всі 4 траєкторії руху. Для того аби перевірити знаходження головного героя на цих траєкторіях я заходив на різні форуми аби знайти потрібну мені частинку кода:

```
private void CheckForPlayer()
{
    CalculateDirections();

    for (int i = 0; i < directions.Length; i++)
    {
        Debug.DrawRay(transform.position, directions[i], Color.red);
        RaycastHit2D hit = Physics2D.Raycast(transform.position, directions[i], range, playerLayer);

        if (hit.collider != null && !attacking)
        {
            attacking = true;
            destination = directions[i];
            checkTimer = 0;
        }
    }
}
```

Рис. 2.12 Spikehead

Також в моєму проекті реалізовані ще 2 механіки це Checkpoint(місце збереження) та підбираємі серця для відновлення основного здоров'я(Colectibles)

-Checkpoint - це точка збереження після смерті. Тобто якщо ви помрете у грі після того як активували Checkpoint то ви з'явитесь на його місті та гра не закінчиться.Для його реалізації мені було потрібно взаємодіяти з скриптом “Health” це аби при відродженні здоров'я було повне.Також в цьому же скрипті я прописав що при втраті всіх сердець повинна програватись анімація смерті. І на основі цього коду я зробив і пару строчок для відродження на точці збереження.Для того аби камера не лишалась на місті смерті , я додав функцію “MoveToNewRoom”. Там описано шо якщо Checkpoint був активований і герой помер то проходить перевірка який послідній Checkpoint був активований і в ту кімнату переміщається камера.Також для того аби не можна було активувати його ще раз я додав опцію що при активації Checkpoint з нього спадає 2DColider ,і він стає неактивним.

Рис. 2.13 Checkpoint

-Colectibles - це предмети які можна підібрати. В даній грі я це реалізував як сердечка для відновлення основного здоров'я. Для цього я створив новий скрипт “HealthCollectible” і я накинув BoxColider для тригеру. Так як раніше я вже реалізував подібний код , тільки там наносилась шкода . Я його відредагував під новий предмет у грі.

Рис. 2.14 HealthCollectible

Ну і в кінці я реалізував так звані двері-проходи між кімнатами.Вони були зроблені завдяки невидимому колайдеру на який я накинув скрипт в якому розписав щоб при зіткненні Player та Door камеру перенаправляло на наступну кімнату(кожна кімнату я описав як окремий елемент тому я просто вказував номер кімнати в яку потрібно перенести камеру).

Висновки

Розробка 2D гри в середовищі Unity з використанням C# та Photoshop є важливим кроком для будь-якого розробника, який бажає створювати сучасні та якісні ігри. В процесі роботи над проектом були розглянуті різні аспекти розробки ігор, включаючи створення графіки, програмування логіки та інтеграцію різних елементів. Завдяки цій роботі вдалося створити гру, яка має потенціал для подальшого розвитку та вдосконалення для релізу. Отримані знання та навички сприятимуть професійному зростанню та розширенню портфоліо.

Список використаної літератури

1. Unity – [Електронний ресурс] – https://docs.unity3d.com/Manual/2DFeature.html
2. Unity 2D aniamtion – [Електронний ресурс] – https://docs.unity3d.com/Manual/com.unity.2d.animation.html
3. Unity 2D Sprite – [Електронний ресурс] – https://docs.unity3d.com/Manual/com.unity.2d.sprite.html
4. Asset Store – [Електронний ресурс] – https://assetstore.unity.com/
5. Музика та звуки – [Електронний ресурс] – https://99sounds.org/sounds/
6. C# – [режим доступу] – https://dou.ua/users/ilya-gromislav/topics/
7. Visual Studio – [Електронний ресурс] – https://learn.microsoft.com/ru-ru/visualstudio/gamedev/unity/get-started/visual-studio-tools-for-unity?pivots=windows

Додаток LoadingManager

```
using UnityEngine;
using UnityEngine.SceneManagement;

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

    private void Awake()
    {
        // Зберігати цей об'єкт навіть коли ми переходимо на нову сцену
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        // Знищити дублікати об'єктів
        else if (instance != null && instance != this)
            Destroy(gameObject);
    }

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

    public void Restart()
    {
        // Перезавантажити поточний рівень
        //SceneManager.LoadScene(currentLevel);
    }
}
```

Додаток CameraController

```
using UnityEngine;


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

    // Слідування за гравцем
    [SerializeField] private Transform player;
    [SerializeField] private float aheadDistance;
    [SerializeField] private float cameraSpeed;
    private float lookAhead;

    private void Update()
    {
        // Камера кімнати
        transform.position = Vector3.SmoothDamp(transform.position, new Vector3(currentPosX, transform.position.y, transform.position.z), ref velocity, speed);

        // Слідування за гравцем
        // 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;
    }
}
```

Додаток SoundManager

```
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>();

        // Зберігати цей об'єкт навіть коли ми переходимо на нову сцену
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        // Знищити дублікати об'єктів
        else if (instance != null && instance != this)
            Destroy(gameObject);

        // Призначити початкові рівні гучності
        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)
    {
        // Отримати початкове значення гучності і змінити його
        float currentVolume = PlayerPrefs.GetFloat(volumeName, 1);
```

```
        currentVolume += change;

        // Перевірити, чи досягли ми максимального або мінімального значення
        if (currentVolume > 1)
            currentVolume = 0;
        else if (currentVolume < 0)
            currentVolume = 1;

        // Призначити кінцеве значення
        float finalVolume = currentVolume * baseVolume;
        source.volume = finalVolume;

        // Зберегти кінцеве значення у PlayerPrefs
        PlayerPrefs.SetFloat(volumeName, currentVolume);
    }
}
```

Додаток MeleeEnemy

```
using UnityEngine;

public class MeleeEnemy : MonoBehaviour
{
    [Header("Параметри атаки")]
    [SerializeField] private float attackCooldown;
    [SerializeField] private float range;
    [SerializeField] private int damage;

    [Header("Параметри колайдера")]
    [SerializeField] private float colliderDistance;
    [SerializeField] private BoxCollider2D boxCollider;

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

    // Посилання
    private Animator anim;
    private Health playerHealth;
    private EnemyPatrol enemyPatrol;

    private void Awake()
    {
        anim = GetComponent<Animator>();
        enemyPatrol = GetComponentInParent<EnemyPatrol>();
    }

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

        // Атакувати тільки коли гравець в полі зору
        if (PlayerInSight())
        {
            if (cooldownTimer >= attackCooldown)
            {
                cooldownTimer = 0;
                anim.SetTrigger("meleeAttack");
            }
        }

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

```
    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);

        if (hit.collider != null)
            playerHealth = hit.transform.GetComponent<Health>();

        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));
    }

    private void DamagePlayer()
    {
        if (PlayerInSight())
            playerHealth.TakeDamage(damage);
    }
}
```

Додаток RangedEnemy

```
using UnityEngine;

public class RangedEnemy : MonoBehaviour
{
    [Header("Параметри атаки")]
    [SerializeField] private float attackCooldown; // Час між атаками
    [SerializeField] private float range; // Дальність атаки
    [SerializeField] private int damage; // Урон від атаки

    [Header("Дальня атака")]
    [SerializeField] private Transform firepoint; // Точка, з якої випускається снаряд
    [SerializeField] private GameObject[] fireballs; // Масив снарядів

    [Header("Параметри колайдера")]
    [SerializeField] private float colliderDistance; // Дистанція для перевірки зіткнень
    [SerializeField] private BoxCollider2D boxCollider; // Колайдер для перевірки зіткнень

    [Header("Шар гравця")]
    [SerializeField] private LayerMask playerLayer; // Шар, на якому знаходиться гравець
    private float cooldownTimer = Mathf.Infinity; // Таймер для відстеження часу між атаками

    [Header("Звук снаряда")]
    [SerializeField] private AudioClip fireballSound; // Звук, що відтворюється при стрільбі

    // Посилання
    private Animator anim; // Аніматор для керування анімаціями
    private EnemyPatrol enemyPatrol; // Компонент патрулювання ворога

    private void Awake()
    {
        anim = GetComponent<Animator>();
        enemyPatrol = GetComponentInParent<EnemyPatrol>();
    }
```

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

        // Атакувати тільки коли гравець в полі зору
        if (PlayerInSight())
        {
            if (cooldownTimer >= attackCooldown)
            {
                cooldownTimer = 0;
                anim.SetTrigger("rangedAttack");
            }
        }

        // Вимикати патрулювання, коли гравець в полі зору
        if (enemyPatrol != null)
            enemyPatrol.enabled = !PlayerInSight();
    }

    private void RangedAttack()
    {
        SoundManager.instance.PlaySound(fireballSound);
        cooldownTimer = 0;
        // Знайти і активувати снаряд
        fireballs[FindFireball()].transform.position = firepoint.position;
        fireballs[FindFireball()].GetComponent<EnemyProjectile>().ActivateProjectile();
    }

    // Знайти перший неактивний снаряд у масиві
    private int FindFireball()
    {
        for (int i = 0; i < fireballs.Length; i++)
        {
            if (!fireballs[i].activeInHierarchy)
                return i;
        }
        return 0; // Повернути перший снаряд, якщо всі інші зайняті
    }

    private bool PlayerInSight()
    {
        // Перевірити, чи є гравець в полі зору за допомогою BoxCast
        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()
    {
        // Намалювати гізмо для візуалізації поля зору
        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));
    }
}
```

Додаток EnemyPatrol

```
using UnityEngine;

public class EnemyPatrol : MonoBehaviour
{
    [Header("Точки патрулювання")]
    [SerializeField] private Transform leftEdge; // Ліва межа патрулювання
    [SerializeField] private Transform rightEdge; // Права межа патрулювання

    [Header("Ворог")]
    [SerializeField] private Transform enemy; // Трансформ ворога

    [Header("Параметри руху")]
    [SerializeField] private float speed; // Швидкість руху
    private Vector3 initScale; // Початковий масштаб ворога
    private bool movingLeft; // Чи рухається ворог вліво

    [Header("Поведінка під час бездіяльності")]
    [SerializeField] private float idleDuration; // Тривалість бездіяльності
    private float idleTimer; // Таймер для відстеження часу бездіяльності

    [Header("Аніматор ворога")]
    [SerializeField] private Animator anim; // Аніматор для керування анімаціями

    private void Awake()
    {
        initScale = enemy.localScale; // Зберегти початковий масштаб ворога
    }

    private void OnDisable()
    {
        anim.SetBool("moving", false); // Зупинити анімацію руху, коли об'єкт вимкнено
    }

    private void Update()
    {
        if (movingLeft)
        {
            if (enemy.position.x >= leftEdge.position.x)
                MoveInDirection(-1); // Рухатись вліво
            else
                DirectionChange(); // Змінити напрямок
        }
        else
        {
            if (enemy.position.x <= rightEdge.position.x)
                MoveInDirection(1); // Рухатись вправо
            else
                DirectionChange(); // Змінити напрямок
        }
    }

    private void DirectionChange()
    {
        anim.SetBool("moving", false); // Зупинити анімацію руху
        idleTimer += Time.deltaTime; // Збільшити таймер бездіяльності

        if (idleTimer > idleDuration)
            movingLeft = !movingLeft; // Змінити напрямок руху після закінчення бездіяльності
    }

    private void MoveInDirection(int _direction)
    {
```

```
        idleTimer = 0; // Скинути таймер бездіяльності
        anim.SetBool("moving", true); // Запустити анімацію руху

        // Повернути ворога в напрямку руху
        enemy.localScale = new Vector3(Mathf.Abs(initScale.x) * _direction,
            initScale.y, initScale.z);

        // Рухатись в заданому напрямку
        enemy.position = new Vector3(enemy.position.x + Time.deltaTime * _direction * speed,
            enemy.position.y, enemy.position.z);
    }
}
```

Додаток EnemyFireballHolder

```
using UnityEngine;



public class EnemyFireballHolder : MonoBehaviour

{

    [SerializeField] private Transform enemy;



    private void Update()

    {

        transform.localScale = enemy.localScale;

    }

}
```

Додаток Health

```
using UnityEngine;
using System.Collections;

public class Health : MonoBehaviour
{
    [Header("Здоров'я")]
    [SerializeField] private float startingHealth; // Початкове здоров'я
    public float currentHealth { get; private set; } // Поточне здоров'я
    private Animator anim; // Аніматор для керування анімаціями
    private bool dead; // Чи мертвий персонаж

    [Header("Невразливість")]
    [SerializeField] private float iFramesDuration; // Тривалість невразливості після
отримання шкоди
    [SerializeField] private int numberOfFlashes; // Кількість миготінь під час
невразливості
    private SpriteRenderer spriteRend; // Спрайт рендерер для зміни кольору під час
миготіння

    [Header("Компоненти")]
    [SerializeField] private Behaviour[] components; // Компоненти, які потрібно
деактивувати при смерті
    private bool invulnerable; // Чи невразливий персонаж

    [Header("Звук смерті")]
    [SerializeField] private AudioClip deathSound; // Звук смерті
```

```
    [SerializeField] private AudioClip hurtSound; // Звук при отриманні шкоди

    private void Awake()
    {
        currentHealth = startingHealth; // Встановити поточне здоров'я
        anim = GetComponent<Animator>(); // Отримати компонент аніматора
        spriteRend = GetComponent<SpriteRenderer>(); // Отримати компонент спрайт рендерера
    }

    public void TakeDamage(float _damage)
    {
        if (invulnerable) return; // Якщо персонаж невразливий, не отримувати шкоду

        currentHealth = Mathf.Clamp(currentHealth - _damage, 0, startingHealth); // Зменшити здоров'я

        if (currentHealth > 0)
        {
            anim.SetTrigger("hurt"); // Запустити анімацію отримання шкоди
            StartCoroutine(Invulnerability()); // Почати корутину невразливості
            SoundManager.instance.PlaySound(hurtSound); // Відтворити звук отримання шкоди
        }
        else
        {
            if (!dead)
            {
                // Деактивувати всі прикріплені компоненти
                foreach (Behaviour component in components)
                    component.enabled = false;

                anim.SetBool("grounded", true); // Встановити анімацію знаходження на землі
                anim.SetTrigger("die"); // Запустити анімацію смерті

                dead = true; // Встановити статус персонажа як мертвого
                SoundManager.instance.PlaySound(deathSound); // Відтворити звук смерті
            }
        }
    }

    public void AddHealth(float _value)
    {
        currentHealth = Mathf.Clamp(currentHealth + _value, 0, startingHealth); // Збільшити здоров'я
    }

    private IEnumerator Invulnerability()
    {
        invulnerable = true; // Встановити невразливість
        Physics2D.IgnoreLayerCollision(10, 11, true); // Ігнорувати зіткнення з певними шарами

        for (int i = 0; i < numberOfFlashes; i++)
        {
            spriteRend.color = new Color(1, 0, 0, 0.5f); // Змінити колір спрайта на червоний з прозорістю
            yield return new WaitForSeconds(iFramesDuration / (numberOfFlashes * 2));
            spriteRend.color = Color.white; // Відновити колір спрайта
            yield return new WaitForSeconds(iFramesDuration / (numberOfFlashes * 2));
        }

        Physics2D.IgnoreLayerCollision(10, 11, false); // Відновити зіткнення з певними шарами
```

```
        invulnerable = false; // Вимкнути невразливість
    }

    private void Deactivate()
    {
        gameObject.SetActive(false); // Деактивувати об'єкт
    }

    // Відродження
    public void Respawn()
    {
        AddHealth(startingHealth); // Відновити здоров'я
        anim.ResetTrigger("die"); // Скинути тригер смерті
        anim.Play("Idle"); // Запустити анімацію бездіяльності
        StartCoroutine(Invulnerability()); // Почати корутину невразливості
        dead = false; // Встановити статус персонажа як живого

        // Активувати всі прикріплені компоненти
        foreach (Behaviour component in components)
            component.enabled = true;
    }
}
```

Додаток Healthbar

```
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;

    }

}
```

Додаток HealthCollectible

```
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);

        }

    }

}
```

Додаток PlayerMovement

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    [Header("Параметри руху")]
    [SerializeField] private float speed; // Швидкість руху
    [SerializeField] private float jumpPower; // Сила стрибка

    [Header("Час польоту")]
    [SerializeField] private float coyoteTime; // Час, протягом якого гравець може
залишатись у повітрі перед стрибком
    private float coyoteCounter; // Час, який пройшов з моменту, коли гравець зійшов
з краю

    [Header("Множинні стрибки")]
    [SerializeField] private int extraJumps; // Додаткові стрибки
    private int jumpCounter; // Лічильник стрибків

    [Header("Стрибки від стіни")]
    [SerializeField] private float wallJumpX; // Горизонтальна сила стрибка від стіни
    [SerializeField] private float wallJumpY; // Вертикальна сила стрибка від стіни

    [Header("Шари")]
    [SerializeField] private LayerMask groundLayer; // Шар землі
    [SerializeField] private LayerMask wallLayer; // Шар стін

    [Header("Звуки")]
    [SerializeField] private AudioClip jumpSound; // Звук стрибка

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

```
    private void Awake()
    {
        // Отримання компонентів Rigidbody та Animator з об'єкта
        body = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        boxCollider = GetComponent<BoxCollider2D>();
    }

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

        // Переворот гравця при русі вліво або вправо
        if (horizontalInput > 0.01f)
            transform.localScale = Vector3.one;
        else if (horizontalInput < -0.01f)
            transform.localScale = new Vector3(-1, 1, 1);

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

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

        // Регулювання висоти 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 таймера "чояте таймера" при
знаходженні на землі
                jumpCounter = extraJumps; // Скидання лічильника стрибків до
додаткових стрибків
            }
            else
                coyoteCounter -= Time.deltaTime; // Зменшення таймера "чояте
таймера", коли гравець не на землі
        }
    }

    private void Jump()
    {
        if (coyoteCounter <= 0 && !onWall() && jumpCounter <= 0) return;
        // Якщо таймер "чояте таймера" дорівнює або менший за 0, гравець не на стіні
і немає додаткових стрибків, не виконувати дію

        SoundManager.instance.PlaySound(jumpSound);

        if (onWall())
            WallJump();
        else
        {
            if (isGrounded())
```

```
                body.velocity = new Vector2(body.velocity.x, jumpPower);
            else
            {
                // Якщо гравець не на землі і таймер "чояте таймера" більший за 0, виконати звичайний стрибок
                if (coyoteCounter > 0)
                    body.velocity = new Vector2(body.velocity.x, jumpPower);
                else
                {
                    if (jumpCounter > 0) // Якщо є додаткові стрибки, виконати стрибок і зменшити лічильник стрибків
                    {
                        body.velocity = new Vector2(body.velocity.x, jumpPower);
                        jumpCounter--;
                    }
                }
            }

            // Скинути таймер "чояте таймера" до 0, щоб уникнути подвійних стрибків
            coyoteCounter = 0;
        }
    }

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

    private bool isGrounded()
    {
        RaycastHit2D   raycastHit   =   Physics2D.BoxCast(boxCollider.bounds.center, boxCollider.bounds.size,  0,  Vector2.down,  0.1f,  groundLayer);  //  Перевірка,  чи знаходиться гравець на землі
        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); // Перевірка, чи знаходиться гравець на стіні
        return raycastHit.collider != null;
    }

    public bool canAttack()
    {
        return horizontalInput == 0 && isGrounded() && !onWall(); // Перевірка, чи може гравець атакувати
    }
}
```

Додаток PlayerAttack

```
using UnityEngine;

public class PlayerAttack : MonoBehaviour
{
    [SerializeField] private float attackCooldown; // Період атаки
    [SerializeField] private Transform firePoint; // Точка викиду вогняних куль
    [SerializeField] private GameObject[] fireballs; // Масив вогняних куль
    [SerializeField] private AudioClip fireballSound; // Звук вогняної кулі

    private Animator anim; // Аніматор гравця
    private PlayerMovement playerMovement; // Клас руху гравця
    private float cooldownTimer = Mathf.Infinity; // Таймер перезарядки атаки
```

```
    private void Awake()
    {
        anim = GetComponent<Animator>(); // Отримання компоненту аніматора
        playerMovement = GetComponent<PlayerMovement>(); // Отримання компоненту руху
гравця
    }

    private void Update()
    {
        // Виконання атаки, якщо гравець натиснув ліву кнопку миші, таймер перезарядки
атаки дорівнює періоду атаки,
        // гравець може атакувати, і час відтворення більше 0
        if (Input.GetMouseButton(0) && cooldownTimer > attackCooldown &&
playerMovement.canAttack() && Time.timeScale > 0)
            Attack();

        cooldownTimer += Time.deltaTime; // Збільшення таймера перезарядки
    }

    private void Attack()
    {
        SoundManager.instance.PlaySound(fireballSound); // Відтворення звуку вогняної
кулі
        anim.SetTrigger("attack"); // Запуск анімації атаки
        cooldownTimer = 0; // Скидання таймера перезарядки атаки

        // Встановлення позиції вогняної кулі та її напрямку
        fireballs[FindFireball()].transform.position = firePoint.position;

fireballs[FindFireball()].GetComponent<Projectile>().SetDirection(Mathf.Sign(transfo
rm.localScale.x));
    }

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

Додаток Projectile

```
using UnityEngine;

public class Projectile : MonoBehaviour
{
    [SerializeField] private float speed; // Швидкість руху
    private float direction; // Напрямок руху
    private bool hit; // Чи попав у ціль
    private float lifetime; // Тривалість існування

    private Animator anim; // Аніматор
    private BoxCollider2D boxCollider; // Коллайдер

    private void Awake()
    {
        anim = GetComponent<Animator>(); // Отримання компоненту аніматора
        boxCollider = GetComponent<BoxCollider2D>(); // Отримання компоненту
коллайдера
    }
    private void Update()
    {
        if (hit) return; // Якщо попав у ціль, не виконувати подальші дії
```

```
        float movementSpeed = speed * Time.deltaTime * direction; // Обчислення швидкості руху
        transform.Translate(movementSpeed, 0, 0); // Рух в напрямку

        lifetime += Time.deltaTime; // Збільшення тривалості існування
        if (lifetime > 5) gameObject.SetActive(false); // Деактивація, якщо тривалість існування більше 5 секунд
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        hit = true; // Встановлення попадання у ціль
        boxCollider.enabled = false; // Вимкнення коллайдера
        anim.SetTrigger("explode"); // Запуск анімації вибуху

        if (collision.tag == "Enemy") // Якщо ціль - ворог
            collision.GetComponent<Health>()?.TakeDamage(1); // Завдання ворогові 1 одиниці пошкодження
    }

    public void SetDirection(float _direction)
    {
        lifetime = 0; // Скидання тривалості існування
        direction = _direction; // Встановлення напрямку руху
        gameObject.SetActive(true); // Активація
        hit = false; // Скидання попадання у ціль
        boxCollider.enabled = true; // Включення коллайдера

        // Зміна напрямку об'єкту
        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); // Деактивація
    }
}
```

Додаток PlayerRespawn

```
using UnityEngine;

public class PlayerRespawn : MonoBehaviour
{
    [SerializeField] private AudioClip checkpoint; // Звук чекпоінту
    private Transform currentCheckpoint; // Поточний чекпоінт
    private Health playerHealth; // Здоров'я гравця
    private UIManager uiManager; // Менеджер інтерфейсу

    private void Awake()
    {
        playerHealth = GetComponent<Health>(); // Отримання компоненту здоров'я гравця
        uiManager = FindObjectOfType<UIManager>(); // Пошук компоненту менеджера інтерфейсу
    }

    public void RespawnCheck()
    {
        if (currentCheckpoint == null) // Якщо поточний чекпоінт не задано
        {
            uiManager.GameOver(); // Виведення повідомлення про кінець гри
            return;
```

```
        }

        playerHealth.Respawn(); // Відновлення здоров'я гравця та скидання анімації
        transform.position = currentCheckpoint.position; // Переміщення гравця до
поточного чекпоінту

        // Переміщення камери до кімнати з поточним чекпоінтом

Camera.main.GetComponent<CameraController>().MoveToNewRoom(currentCheckpoint.parent)
;
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Checkpoint") // Якщо стикаємося з об'єктом-
чекпоінтом
        {
            currentCheckpoint = collision.transform; // Встановлення поточного
чекпоінту
            SoundManager.instance.PlaySound(checkpoint); // Відтворення звуку
чекпоінту
            collision.GetComponent<Collider2D>().enabled = false; // Вимкнення
колайдера чекпоінту
            collision.GetComponent<Animator>().SetTrigger("activate"); // Запуск
анімації активації чекпоінту
        }
    }
}
```

Додаток Door

```
using UnityEngine;

public class Door : MonoBehaviour
{
    [SerializeField] private Transform previousRoom; // Попередня кімната
    [SerializeField] private Transform nextRoom; // Наступна кімната
    private CameraController cam; // Контролер камери

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

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player") // Якщо гравець зіштовхується з дверима
        {
            if (collision.transform.position.x < transform.position.x) // Якщо
гравець зліва від дверей
            {
                cam.MoveToNewRoom(nextRoom); // Переміщення камери до наступної
кімнати
                nextRoom.GetComponent<Room>().ActivateRoom(true); // Активація
наступної кімнати
                previousRoom.GetComponent<Room>().ActivateRoom(false); //
Деактивація попередньої кімнати
            }
            else // Якщо гравець справа від дверей
            {
                cam.MoveToNewRoom(previousRoom); // Переміщення камери до попередньої
кімнати
                previousRoom.GetComponent<Room>().ActivateRoom(true); // Активація
попередньої кімнати
                nextRoom.GetComponent<Room>().ActivateRoom(false); // Деактивація
наступної кімнати
            }
```

```
        }
    }
}
```

Додаток Room

```
using UnityEngine;

public class Room : MonoBehaviour
{
    [SerializeField] private GameObject[] enemies; // Масив ворогів
    private Vector3[] initialPosition; // Початкові позиції ворогів

    private void Awake()
    {
        // Збереження початкових позицій ворогів
        initialPosition = new Vector3[enemies.Length];
        for (int i = 0; i < enemies.Length; i++)
        {
            if(enemies[i] != null)
                initialPosition[i] = enemies[i].transform.position;
        }

        // Деактивація кімнат
        if (transform.GetSiblingIndex() != 0)
            ActivateRoom(false); // Деактивація кімнати
    }

    public void ActivateRoom(bool _status)
    {
        // Активація/деактивація ворогів
        for (int i = 0; i < enemies.Length; i++)
        {
            if (enemies[i] != null)
            {
                enemies[i].SetActive(_status); // Встановлення статусу активності
                enemies[i].transform.position  =  initialPosition[i];  //  Повернення
ворогів на початкові позиції
            }
        }
    }
}
```

Додаток Spikehead

```
using UnityEngine;

public class Spikehead : EnemyDamage
{
    [Header("SpikeHead Attributes")]
    [SerializeField] private float speed; // Швидкість руху
    [SerializeField] private float range; // Дальність видимості
    [SerializeField] private float checkDelay; // Затримка перевірки
    [SerializeField] private LayerMask playerLayer; // Шар гравця
    private Vector3[] directions = new Vector3[4]; // Масив напрямків руху
    private Vector3 destination; // Цільова точка
    private float checkTimer; // Таймер перевірки
    private bool attacking; // Статус атаки

    [Header("SFX")]
    [SerializeField] private AudioClip impactSound; // Звук удару

    private void OnEnable()
    {
        Stop(); // Зупинити
    }

    private void Update()
    {
```

```
        // Рух спайкхеда до цільової точки лише при атакі
        if (attacking)
            transform.Translate(destination * Time.deltaTime * speed);
        else
        {
            checkTimer += Time.deltaTime;
            if (checkTimer > checkDelay)
                CheckForPlayer(); // Перевірка наявності гравця
        }
    }

    private void CheckForPlayer()
    {
        CalculateDirections(); // Обчислення напрямків руху

        // Перевірка, чи спайкхед бачить гравця у всіх чотирьох напрямках
        for (int i = 0; i < directions.Length; i++)
        {
            Debug.DrawRay(transform.position, directions[i], Color.red);
            RaycastHit2D hit = Physics2D.Raycast(transform.position, directions[i],
range, playerLayer);

            if (hit.collider != null && !attacking)
            {
                attacking = true;
                destination = directions[i];
                checkTimer = 0;
            }
        }
    }

    private void CalculateDirections()
    {
        directions[0] = transform.right * range; // Вправо
        directions[1] = -transform.right * range; // Вліво
        directions[2] = transform.up * range; // Вгору
        directions[3] = -transform.up * range; // Вниз
    }

    private void Stop()
    {
        destination = transform.position; // Встановити цільову точку як поточну
позицію, щоб ворог не рухався
        attacking = false;
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        SoundManager.instance.PlaySound(impactSound); // Відтворення звуку удару
        base.OnTriggerEnter2D(collision); // Виклик методу базового класу
        Stop(); // Зупинити спайкхеда після зіткнення з чимось
    }
}
```

Додаток Firetrap

```
using UnityEngine;
using System.Collections;

public class Firetrap : MonoBehaviour
{
    [SerializeField] private float damage; // Пошкодження, яке завдає пастка

    [Header("Firetrap Timers")]
    [SerializeField] private float activationDelay; // Затримка активації
    [SerializeField] private float activeTime; // Тривалість активності
    private Animator anim;
    private SpriteRenderer spriteRend;
```

```
    [Header("SFX")]
    [SerializeField] private AudioClip firetrapSound; // Звук пастки

    private bool triggered; // Статус спрацьованої пастки
    private bool active; // Статус активної пастки, яка може завдати шкоди гравцю

    private Health playerHealth; // Здоров'я гравця

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

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

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            playerHealth = collision.GetComponent<Health>(); // Отримати компонент здоров'я гравця

            if (!triggered)
                StartCoroutine(ActivateFiretrap()); // Активувати пастку, якщо вона не активна

            if (active)
                collision.GetComponent<Health>().TakeDamage(damage);   //   Завдати шкоду гравцю, якщо пастка активна
        }
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.tag == "Player")
            playerHealth = null; // Обнулити здоров'я гравця при виході з області пастки
    }
    private IEnumerator ActivateFiretrap()
    {
        // Перемкнути колір спрайта на червоний, щоб повідомити гравця та спрацювати пастку
        triggered = true;
        spriteRend.color = Color.red;

        // Зачекати деякий час, активувати пастку, включити анімацію та повернути колір назад
        yield return new WaitForSeconds(activationDelay);
        SoundManager.instance.PlaySound(firetrapSound); // Відтворити звук пастки
        spriteRend.color = Color.white; // Повернути спрайт в початковий колір
        active = true; // Активувати пастку
        anim.SetBool("activated", true); // Включити анімацію

        // Зачекати деякий час, деактивувати пастку та скинути всі змінні та аніматор
        yield return new WaitForSeconds(activeTime);
        active = false; // Деактивувати пастку
        triggered = false; // Скинути статус спрацьованої пастки
        anim.SetBool("activated", false); // Вимкнути анімацію
    }
}
```

Додаток EnemyProjectile

```
using UnityEngine;

public class EnemyProjectile : EnemyDamage
{
    [SerializeField] private float speed; // Швидкість снаряду
    [SerializeField] private float resetTime; // Час життя снаряду до його вимкнення
    private float lifetime; // Час, який снаряд існує
    private Animator anim; // Аніматор снаряду
    private BoxCollider2D coll; // Колайдер снаряду

    private bool hit; // Прапорець, що показує, чи влучив снаряд у ворожого об'єкта

    private void Awake()
    {
        anim = GetComponent<Animator>(); // Отримуємо посилання на компонент аніматора
        coll = GetComponent<BoxCollider2D>(); // Отримуємо посилання на компонент
колайдера
    }

    public void ActivateProjectile()
    {
        hit = false; // При активації снаряду знімаємо прапорець влучення
        lifetime = 0; // Скидаємо час життя
        gameObject.SetActive(true); // Активуємо об'єкт снаряду
        coll.enabled = true; // Активуємо колайдер
    }
    private void Update()
    {
        if (hit) return; // Якщо снаряд вже влучив у ворожий об'єкт, вихід з методу

        float movementSpeed = speed * Time.deltaTime; // Вираховуємо швидкість руху
снаряду
        transform.Translate(movementSpeed, 0, 0); // Зміщуємо снаряд вперед

        lifetime += Time.deltaTime; // Збільшуємо час життя снаряду

        if (lifetime > resetTime) // Якщо снаряд живе довше за встановлений час
            gameObject.SetActive(false); // Вимикаємо снаряд
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        hit = true; // Встановлюємо прапорець влучення, оскільки снаряд влучив
        base.OnTriggerEnter2D(collision);   //   Викликаємо   логіку   влучення   в
батьківському класі

        coll.enabled = false; // Вимикаємо колайдер снаряду

        if (anim != null) // Якщо є аніматор у снаряду
            anim.SetTrigger("explode"); // Відтворюємо анімацію вибуху
        else
            gameObject.SetActive(false); // Вимикаємо снаряд, якщо немає аніматора
    }
    private void Deactivate()
    {
        gameObject.SetActive(false); // Вимикаємо снаряд
    }
}
```

Додаток EnemyDamage

```
using UnityEngine;
```

```
public class EnemyDamage : MonoBehaviour
{
    [SerializeField] protected float damage;

    protected void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
            collision.GetComponent<Health>()?.TakeDamage(damage);
    }
}
```

Додаток Enemy_Sideways

```
using UnityEngine;

public class Enemy_Sideways : MonoBehaviour
{
    [SerializeField] private float movementDistance;
    [SerializeField] private float speed;
    [SerializeField] private float damage;
    private bool movingLeft;
    private float leftEdge;
    private float rightEdge;

    private void Awake()
    {
        leftEdge = transform.position.x - movementDistance;
        rightEdge = transform.position.x + movementDistance;
    }

    private void Update()
    {
        if (movingLeft)
        {
```

```
            if (transform.position.x > leftEdge)
            {
                transform.position = new Vector3(transform.position.x - speed *
Time.deltaTime, transform.position.y, transform.position.z);
            }
            else
                movingLeft = false;
        }
        else
        {
            if (transform.position.x < rightEdge)
            {
                transform.position = new Vector3(transform.position.x + speed *
Time.deltaTime, transform.position.y, transform.position.z);
            }
            else
                movingLeft = true;
        }
    }

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

Додаток ArrowTrap

```
using UnityEngine;

public class ArrowTrap : MonoBehaviour
{
    [SerializeField] private float attackCooldown; // Час між пострілами
    [SerializeField] private Transform firePoint; // Точка, з якої випускаються стріли
    [SerializeField] private GameObject[] arrows; // Масив стріл

    private float cooldownTimer; // Таймер відновлення атаки
```

```
    [Header("SFX")]
    [SerializeField] private AudioClip arrowSound; // Звук стріли

    private void Attack()
    {
        cooldownTimer = 0; // Скидаємо таймер відновлення атаки

        SoundManager.instance.PlaySound(arrowSound); // Відтворюємо звук стріли
        arrows[FindArrow()].transform.position = firePoint.position; // Встановлюємо
позицію стріли
        arrows[FindArrow()].GetComponent<EnemyProjectile>().ActivateProjectile(); //
Активуємо стрілу
    }
    private int FindArrow()
    {
        for (int i = 0; i < arrows.Length; i++)
        {
            if (!arrows[i].activeInHierarchy) // Шукаємо неактивну стрілу
                return i; // Повертаємо її індекс
        }
        return 0; // Якщо всі стріли активні, повертаємо першу стрілу
    }
    private void Update()
    {
        cooldownTimer += Time.deltaTime; // Збільшуємо таймер відновлення атаки

        if (cooldownTimer >= attackCooldown) // Якщо таймер досягнув встановленого
значення
            Attack(); // Виконуємо атаку
    }
}
```

Додаток UIManager

```
using UnityEngine;

using UnityEngine.SceneManagement;



public class UIManager : MonoBehaviour

{

    [Header("Game Over")]

    [SerializeField] private GameObject gameOverScreen;

    [SerializeField] private AudioClip gameOverSound;



    [Header("Pause")]

    [SerializeField] private GameObject pauseScreen;



    private void Awake()

    {

        // Deactivate the game over and pause screens initially

        gameOverScreen.SetActive(false);

        pauseScreen.SetActive(false);
```

```
    }

    private void Update()
    {
        // Toggle pause screen on Escape key press
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            // If the pause screen is already active, unpause; otherwise, pause the
game
            PauseGame(!pauseScreen.activeInHierarchy);
        }
    }

    #region Game Over
    // Activate the game over screen
    public void GameOver()
    {
        gameOverScreen.SetActive(true);
        SoundManager.instance.PlaySound(gameOverSound);
    }

    // Restart the level
    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    // Load the main menu scene
    public void MainMenu()
    {
        SceneManager.LoadScene(0);
    }
```

```
    // Quit the game
    public void Quit()
    {
        Application.Quit(); // Quits the game (only works in build)

#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false; // Exits play mode (will only be executed in the editor)
#endif
    }
    #endregion

    #region Pause
    // Toggle pause state of the game
    public void PauseGame(bool status)
    {
        // Activate or deactivate the pause screen based on the status parameter
        pauseScreen.SetActive(status);

        // When the game is paused, set timescale to 0 (time stops); otherwise, set it back to 1 (normal time)
        Time.timeScale = status ? 0 : 1;
    }

    // Adjust the sound volume
    public void SoundVolume()
    {
        SoundManager.instance.ChangeSoundVolume(0.2f);
    }

    // Adjust the music volume
    public void MusicVolume()
    {
```

```
        SoundManager.instance.ChangeMusicVolume(0.2f);

    }

    #endregion

}
```

Додаток MenuManager

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuManager : MonoBehaviour
{
    [SerializeField] private RectTransform arrow; // Стрілка, яка відображає поточне
положення в меню
    [SerializeField] private RectTransform[] buttons; // Масив кнопок меню
    [SerializeField] private AudioClip changeSound; // Звук перемикання між пунктами
меню
    [SerializeField] private AudioClip interactSound; // Звук взаємодії з обраним
пунктом меню
    private int currentPosition; // Поточне положення в меню

    private void Awake()
    {
        ChangePosition(0); // Встановлення початкового положення в меню
    }
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.UpArrow)) // Переміщення вгору
            ChangePosition(-1);
        else if (Input.GetKeyDown(KeyCode.DownArrow)) // Переміщення вниз
            ChangePosition(1);

        if (Input.GetKeyDown(KeyCode.KeypadEnter) || Input.GetButtonDown("Submit"))
// Взаємодія з пунктом меню
            Interact();
    }

    public void ChangePosition(int _change)
    {
        currentPosition += _change;

        if (_change != 0)
            SoundManager.instance.PlaySound(changeSound);   //   Програвання   звуку
перемикання

        if (currentPosition < 0)
            currentPosition = buttons.Length - 1;
        else if (currentPosition > buttons.Length - 1)
            currentPosition = 0;

        AssignPosition(); // Встановлення положення стрілки
    }
    private void AssignPosition()
    {
        arrow.position           =           new           Vector3(arrow.position.x,
buttons[currentPosition].position.y);  //  Прив'язка  стрілки  до  позиції  поточного
пункту меню
    }
    private void Interact()
    {
        SoundManager.instance.PlaySound(interactSound);    //    Програвання    звуку
взаємодії
```

```
        if (currentPosition == 0)
        {
            // Початок гри
            SceneManager.LoadScene(PlayerPrefs.GetInt("level", 1));
        }
        else if (currentPosition == 1)
        {
            // Відкриття налаштувань
        }
        else if (currentPosition == 2)
        {
            // Відкриття кредитів
        }
        else if (currentPosition == 3)
            Application.Quit(); // Вихід з гри
    }
}
```

Додаток SelectionArrow

```
using UnityEngine;
using UnityEngine.UI;

public class SelectionArrow : MonoBehaviour
{
    [SerializeField] private RectTransform[] buttons; // Масив кнопок меню
    [SerializeField] private AudioClip changeSound; // Звук перемикання між пунктами
меню
    [SerializeField] private AudioClip interactSound; // Звук взаємодії з обраним
пунктом меню
    private RectTransform arrow; // Стрілка вибору
    private int currentPosition; // Поточне положення стрілки

    private void Awake()
    {
        arrow = GetComponent<RectTransform>(); // Отримання RectTransform компонента
    }
    private void OnEnable()
    {
        currentPosition = 0; // Встановлення початкового положення
        ChangePosition(0);
    }
    private void Update()
    {
        // Зміна положення стрілки вибору
        if (Input.GetKeyDown(KeyCode.UpArrow) || Input.GetKeyDown(KeyCode.W))
            ChangePosition(-1);
        if (Input.GetKeyDown(KeyCode.DownArrow) || Input.GetKeyDown(KeyCode.S))
            ChangePosition(1);

        // Взаємодія з поточним варіантом
        if (Input.GetKeyDown(KeyCode.KeypadEnter) || Input.GetKeyDown(KeyCode.E))
            Interact();
    }

    private void ChangePosition(int _change)
    {
        currentPosition += _change;

        if (_change != 0)
            SoundManager.instance.PlaySound(changeSound);   //   Програвання   звуку
перемикання

        if (currentPosition < 0)
            currentPosition = buttons.Length - 1;
        else if (currentPosition > buttons.Length - 1)
            currentPosition = 0;
```

```
        AssignPosition(); // Встановлення нового положення стрілки
    }
    private void AssignPosition()
    {
        // Встановлення Y-позиції поточного варіанту для стрілки (рух вгору та вниз)
        arrow.position = new Vector3(arrow.position.x,
buttons[currentPosition].position.y);
    }
    private void Interact()
    {
        SoundManager.instance.PlaySound(interactSound); // Програвання звуку
взаємодії

        // Отримання компонента кнопки для поточного варіанту та виклик його функції
        buttons[currentPosition].GetComponent<Button>().onClick.Invoke();
    }
}
```

Додаток VolumeText

```
using UnityEngine;

using UnityEngine.UI;



public class VolumeText : MonoBehaviour

{

    [SerializeField] private string volumeName;

    [SerializeField] private string textIntro; //Sound:  or Music:

    private Text txt;



    private void Awake()

    {

        txt = GetComponent<Text>();

    }

    private void Update()

    {

        UpdateVolume();

    }

    private void UpdateVolume()

    {

        float volumeValue = PlayerPrefs.GetFloat(volumeName) * 100;

        txt.text = textIntro + volumeValue.ToString();

    }

}
```