НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних Наук

(назва кафедри)

Голуб Б. Л.

(підпис) (ПІБ)

"___" _____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмна система ігрових механік у жанрі платформер»

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми

д.е.н., професор Руденський Р.А.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

д. ф. Назаренко В. А.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав Бушний А.А.

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Комп'ютерних наук_____

(назва кафедри)

___к.т.н., доцент_____Голуб Б.Л.

(науковий ступінь, вчене звання) (підпис) (ПІБ)

"_16_" _грудня___ 2024 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Бушному Андрію Андрійовичу

Спеціальність 122 – «Комп'ютерні науки»

ОП «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Програмна система ігрових механік у жанрі платформер» затверджена наказом ректора НУБіП України від 16.12.2024 № _2246_С_

2. Термін подання завершеної роботи на кафедру_____2025.05.25_____
(рік, місяць, число)

3. Вихідні дані до роботи: гра-платформер з моторошною атмосферою

4. Перелік питань, що розглядаються:

- Аналіз проблемної області
- Моделювання предметної області
- Проєктування програмної системи
- Впровадження та експлуатація системи

Дата видачі завдання "_16_"_грудня_____ 2024 р.

Керівник бакалаврської кваліфікаційної роботи _____ / Назаренко В.А. /

(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:_____ / Бушний А.А. /

(підпис) (прізвище та ініціал

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GDD – Game Design Document, документ проектування гри.

PC – Personal Computer.

UML – Unified Modelling Language.

CSV – Comma Separated Values.

СУБД – Система управління базами даних.

SFX – Sound Efects

ВСТУП

Актуальність теми. Ігрова індустрія демонструє стрімкий розвиток: лише за останнє десятиліття доходи від продажу відеоігор зросли більш ніж удвічі, перевищивши 200 мільярдів доларів США у 2023 році. Паралельно з комерційним зростанням значно розширилося коло гравців і жанрів, зокрема зберігається стабільна популярність 2D- та 2.5D-платформерів. Цей жанр відзначається простотою управління та високим потенціалом для художнього вираження, особливо у поєднанні з сучасною 3D-графікою.

Сучасні ігрові рушії, зокрема Unity, відкривають широкі можливості для розробки подібних проєктів навіть індивідуальними розробниками. Unity надає інструменти для створення фізики, рендерингу, анімації та зручного програмування логіки взаємодії. Це робить платформу ідеальною для реалізації ігор, орієнтованих на занурення, наративність та атмосферу.

Метою кваліфікаційної роботи є створення демо-версії 2.5D платформера з базовим набором геймплейних механік, орієнтованої на одинарну гру з високим рівнем атмосферності.

Для досягнення цієї мети були поставлені наступні завдання: сформулювати технічне завдання та вимоги до ігрової системи; проаналізувати існуючі ігри жанру платформер; розробити архітектуру гри з використанням об'єктно-орієнтованих принципів; реалізувати рух, зіткнення, чекпоінти, анімацію та елементи UI; створити демо-рівень для тестування; забезпечити кросплатформеність та збереження конфігурацій.

Об’єктом дослідження є процес розробки інструментального ігрового середовища на базі рушія Unity. Предметом дослідження виступає архітектура та реалізація ключових механік платформера в рамках сучасного геймдизайну.

У роботі застосовано методи системного та функціонального аналізу, об’єктно-орієнтованого програмування, побудови UML-діаграм, а також моделювання ігрової логіки. Основним середовищем реалізації виступає Unity із застосуванням C# як мови розробки та нативних інструментів рушія: Tilemap, Animator, Physics2D, Audio Mixer, ScriptableObject.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Почнемо з визначення того, що таке комп'ютерна гра. Комп'ютерна гра є інтерактивним цифровим розважальним продуктом, створеним для відтворення віртуальної реальності і надання користувачам можливості взаємодіяти з ним. Вона поєднує в собі елементи геймплею, графіки, звуку та інтерактивності, щоб забезпечити цікавий та захоплюючий досвід користувачеві.

Комп'ютерні ігри можуть мати різноманітні жанри, включаючи пригоди, екшн, стратегію, головоломки, рольові ігри та багато інших. Вони можуть бути одиночними (для одного гравця) або мультиплеєрними (для кількох гравців), дозволяючи спільну гру з друзями або іншими гравцями з усього світу через Інтернет.

Комп'ютерні ігри є не тільки джерелом розваги, але і важливим інструментом у галузі освіти, тренування та симуляцій. Вони використовуються для навчання, розвитку навичок та творчого мислення, а також для стимулювання соціальної взаємодії та співпраці.

В цілому, комп'ютерні ігри є важливою частиною сучасної культури та розвитку технологій. Вони надають гравцям унікальну можливість погрузитися в світ фантазії, випробувати свої навички та взаємодіяти з цифровим середовищем, що розширює межі реальності.

Платформер (англ. *platformer*) — жанр відеоігор, де основним елементом геймплею є переміщення персонажа через різноманітні платформи, перешкоди та ворогів, часто з використанням стрибків, лазання або інших способів взаємодії з ігровим середовищем.В основному, платформери націлені на випробування вправності гравця, гравець повинен точно керувати персонажем уникати пасток та перешкод.

Піджанри:

- Класичні платформери: Проста механіка стрибків, чітко розділені рівні, часто з ретро-графікою.Прикладами є Super Mario Bros. ,Sonic the Hedgehog.

- Метроїдавнії: Відзнаками є не характерна для платформерів нелінійність, імітація невеликого відкритиого світу, та здобуття нових навичок для доступу до раніше не досяжних зон.Прикладами є Hollow Knight ,Metroid Dread , Ori.
- Платформери-головоломки: Акцент на розв’язання складних завдань, часто з фізико-орієнтованими механіками(механіки пов’язані з рухом предметів в світі гри).Прикладами є Inside, Limbo.
- Екшн-платформер: Основними відзнаками є швидкісний геймплей, бої з ворогами, арсенал зброї.Приклажами є Cuphead, Dead Cells
- 2.5D платформери: Основними відзнакам є 3D-графіка з обмеженим рухом по двовимірній площині, можуть містити у собі деталі з платформерів інших жанрів

Для розробки комп’ютерної гри для одного гравця у жанрі платформер необхідно виконати наступні задачі:

- сформулювати основну ідею гри
- визначити основні елементи геймплею
- описати основні механіки
- описати вимоги до візуального оформлення гри
- визначити цільову платформу, на яку буде розроблятися гра
- визначити цільову аудиторію гри
- визначити технічні вимоги гри

Проєкт має на меті створити платформер у стилі Little Nightmares/Inside, де гравець досліджує темне, незрозуміле середовище, керуючи персонажем з видом

збоку. У грі немає діалогів чи явного сюжету — лише ігровий світ, що мовчки розповідає історію.

Геймплей - це термін, який описує спосіб, спосіб взаємодії гравця з віртуальним світом гри та системою правил, які визначають його дії та досвід гри. Геймплей включає в себе всі аспекти гри, такі як цілі, завдання, механіки, взаємодія з об'єктами, системи розвитку персонажа та багато іншого. Він визначає характер гри, викликає емоції та надає гравцеві можливість відчути задоволення від гри.

Основним елементом геймплею є система руху персонажа по платформах. Гравець може рухатися в 3Д просторі , але з видом збоку маючи можливість взаємодіяти з платформами.

Механіка - це правила, системи та інструменти, що використовуються у грі для створення взаємодії та обмежень, які впливають на геймплей. Механіки описують спосіб функціонування гри, включаючи рух, взаємодію з об'єктами, стрілянину, скакання, розблокування нових рівнів або властивостей персонажа та багато іншого. Вони визначають правила, за якими гра працює та як гравець може впливати на її світ. Механіки гри формують основу геймплею і є ключовими елементами, що роблять гру унікальною та цікавою[5].

Основні механіки гри включають в себе:

- взаємодія з платформами- гравець має можливість прогресувати по грі за допомогою взаємоміі з ними , для руху далі

Вимоги до візуального оформлення гри є важливою частиною створення іммерсивного та привабливого ігрового досвіду. Основні вимоги до візуального оформлення гри включають:

- естетика і атмосфера -гра повинна мати чітко визначений стиль і атмосферу, які відповідають жанру і настрою гри
- графічні ефекти - використання високоякісних графічних ефектів, таких як освітлення, тіні, частинки, водяні ефекти, дим, вогонь тощо, може значно

покращити візуальний вигляд гри і забезпечити реалістичність або ефектність сцен.

- моделі і текстури - об'єкти та персонажі в грі повинні мати деталізовані 3D-моделі, які відповідають їхнім функціям і характеристикам.
- колірна палітра - вибір кольорів повинен відповідати настрою гри і передавати потрібні емоції.
- інтерфейс користувача - візуальний оформлення інтерфейсу гри повинно бути зрозумілим, привабливим та функціональним. Меню, кнопки, іконки та інші елементи інтерфейсу повинні бути зручними для використання та відповідати загальному стилю гри.

Врахування цих вимог до візуального оформлення гри допоможе створити привабливу і незабутню графіку, яка сприятиме глибокому зануренню гравця в ігровий світ.

При визначенні цільової платформи для розробки гри важливо враховувати різні фактори, такі як цільова аудиторія, характеристики платформи, доступні ресурси та потенційний ринок. Вибір цільової платформи може вплинути на дизайн, функціональність і процес розробки гри.

В даному випадку, обраною цільовою платформою є PC (персональний комп'ютер), це надає широкі можливості і гнучкість для реалізації складних графічних ігор. Це дозволяє звернути увагу на деталізацію графіки та ігровий процес.

Мінімальні технічні вимоги визначають базові характеристики, які необхідні для запуску гри і забезпечення її роботи без серйозних проблем.

Рекомендовані технічні вимоги, зазвичай, вищі за мінімальні. Вони вказують на оптимальні характеристики, які забезпечать більш високу якість графіки, плавний геймплей та загальний комфорт при грі.

Крім апаратних вимог, технічні вимоги можуть також включати програмні вимоги, наприклад, певну версію драйверів графічної карти або необхідність наявності певного програмного забезпечення, такого як DirectX або OpenGL.

Під час системного аналізу було визначено функціональні та нефункціональні вимоги до програмного продукту.

Функціональні вимоги:

- геймплей -створення системи руху та складних марщрутів, що викликатиме цікавість у гравця.
- управління - Інтуїтивно зрозумілий та легкий для освоєння інтерфейс управління грою.
- візуальна сторона - Привабливий та естетичний дизайн рівнів, об'єктів та інтерфейсу користувача. Використання візуальних ефектів, які допоможуть підкреслити головоломки та створити атмосферу гри.
- звуковий супровід - Наявність музики та звукових ефектів, які доповнюють атмосферу гри та створюють емоційну зв'язок з гравцем.

Нефункціональні вимоги:

- продуктивність - Гра повинна працювати плавно та без затримок
- стабільність та надійність - Гра повинна працювати без відмов та критичних помилок. Повинна бути відсутність ситуацій, що можуть призвести до втрати прогресу гравця або пошкодження файлів.
- інтерфейс користувача - Інтуїтивно зрозумілий та зручний інтерфейс, який дозволяє гравцям легко орієнтуватися та взаємодіяти з грою, відсутній інтерфейс в процесі гри для занурення в світ
- ефективність використання ресурсів - Оптимізація використання апаратних ресурсів, таких як процесор, пам'ять та графічний прискорювач, для забезпечення оптимальної продуктивності гри.

1.2 Аналіз наявних рішень

Little Nightmares

Гра для одного гравця орієнтована на емоційне переживання, у ній поєднані стелс, платформінг та загадки. Ви граєте за дитину у світі гігантів яка потрапила на величезний дивний круїзний корабель. З усіма дорослими щось не так, вони переслідують гравця. Гра є фантстичною та має свій сюжет хоча гравцю його ніхто не розповідає напряму, це робить сам ігровий світ .Візуальний стиль похмурий, геймплей — повільний, але насичений інтерактивністю.

Рис. 1.1 Little Nightmares.

Inside

2.5D платформер для одного гравця із кінематографічним стилем. Без слів розповідає глибоку історію про постапокаліптичний світ в якому є тільки правила

та ніякої індивідуальності. Основу складає взаємодія з перешкодами, об’єктами, фізичні головоломки, які поступово змінюються механічно. Гравець проходить крізь атмосферні сцени, не знаючи, що чекає далі.

Рис. 1.2 Inside.

Trine

2.5D гра для одного гравця з фантастичним стилем , яка розповідає про пригоди 3 героїв:лицаря, мага, лучниці. Основа механіка гри це вміння героїв, за допомогою них потрібно проходити рівні з стрибками , головоломки. Кожен з героїв вміє щось унікальне , та на основі цього побудовані рівні. Стиль фентезі

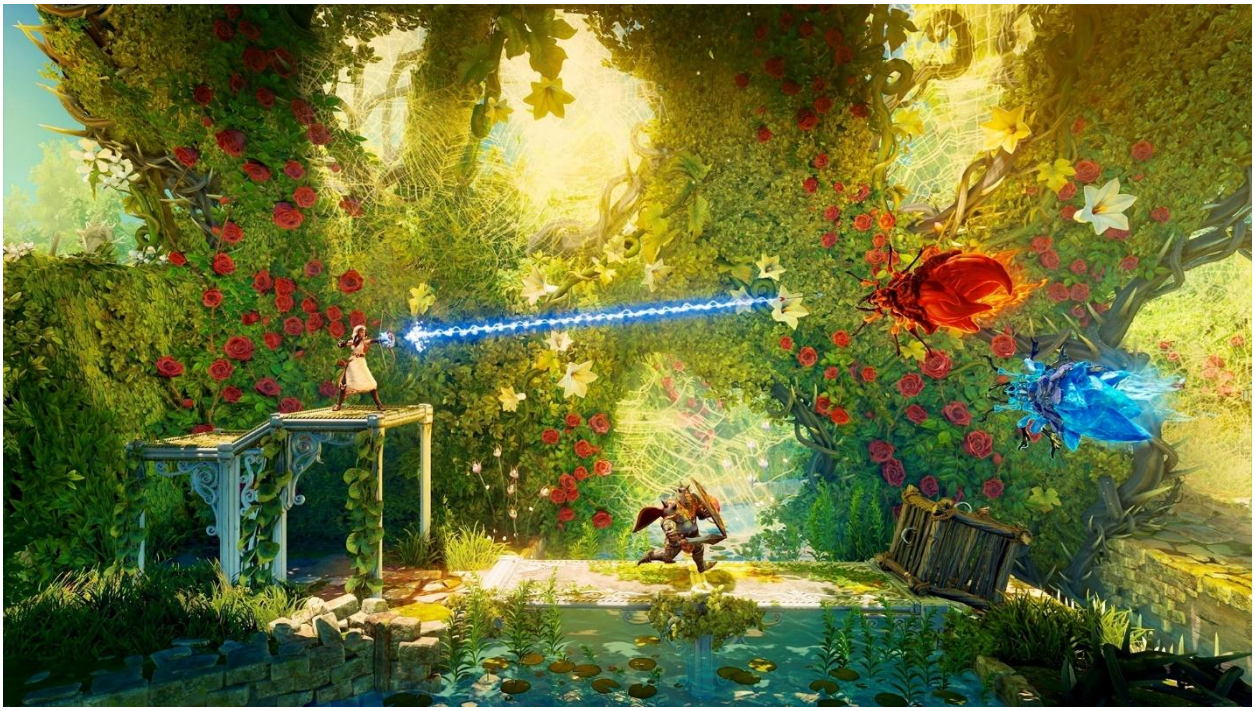


Рис. 1.3 Trine.

Порівняння

Таблиця 1.1

Назва	**Стиль**	**Механіки**	**Атмосфера**
Little Nightmares	Темний 2.5D	Платформінг, стелс	Напруга, страх
Inside	2.5D	Платформінг, фізика	Моторошність
Trine	2.5D	Платформінг, головоломки	Казковість, мрійливість

Завдяки різноманітності геймплею, естетиці та ідей, кожна гра пропонує неповторний досвід.

В розроблюваному проекті немає власної оригінальної ігрової механіки, але використовуються ідеї та елементи з аналогічних ігор, для того щоб поєднати їх таким чином, щоб створити новий і унікальний досвід для гравців.

1.3 Моделювання предметної області

1.3.1 Загальні відомості про етапи розробки комп’ютерних ігор.

У процесі розробки комп'ютерних ігор існує кілька важливих етапів, що допомагають забезпечити успішну та ефективну реалізацію проекту. Кожен з цих етапів відіграє свою роль у процесі розробки та сприяє створенню якісної гри.

Перший етап розробки комп'ютерної гри - це створення ідеї та розробка GDD [9]. Ідея визначає загальну концепцію гри, тематику та мету. Вона може бути інноваційною або базуватися на вже існуючих жанрах. Після визначення ідеї, розробляється GDD - документ, що визначає основні аспекти гри, такі як механіка, інтерфейс, система рівнів та сценарій гри. GDD є основним джерелом інформації для всіх учасників розробки та визначає загальну візію та цілі проекту.

Наступним етапом є прототипування, на якому створюється пробний варіант гри, який дозволяє перевірити основні механіки та ідеї, внести корективи та визначити найбільш ефективні рішення. Прототип допомагає уточнити геймплей, зрозуміти потенційні проблеми та виявити можливості для поліпшення. Він може бути в простому форматі, сконцентрованим на ключових аспектах гри, але варіант прототипу може відрізнятися залежно від потреб проекту.

Після прототипування, йде MVP - це мінімально функціональний продукт, який містить базовий набір функцій та може бути випущений для перевірки гри на реальних користувачах[10]. MVP дозволяє оцінити реакцію аудиторії, отримати фідбек та визначити, які аспекти гри потребують подальшого вдосконалення. Цей етап допомагає сконцентруватися на основних елементах гри та впевнитися, що вона викликає зацікавлення у гравців.

Альфа[11]. На етапі альфа-версії гри реалізуються основні функції та механіки, а також додаються перші рівні або завдання. Гра проходить внутрішнє тестування та оптимізацію. Важливим аспектом альфа-версії є залучення тестерів, які надають фідбек щодо геймплею, балансу, помилок та загального враження від гри.

Бета[12]. На етапі бета-версії гра набуває більшого обсягу та готова до тестування на широкій аудиторії. Випускається публічна бета-версія гри, в якій гравці мають можливість випробувати її та надати фідбек. Цей етап допомагає виявити та виправити проблеми, покращити гру та залучити нових гравців до проекту.

Передостаннім етапом розробки комп’ютерної гри є реліз, який передбачає остаточну підготовку гри до комерційного випуску. На цьому етапі вирішуються останні проблеми та помилки, проводяться остаточні тести та оптимізація гри. Після цього гра готова до випуску на ринок та до взаємодії зі широкою аудиторією гравців[13].

Після релізу розпочинається етап післярелізного супроводу гри. На цьому етапі здійснюється моніторинг реакції гравців, вирішення проблем та випуск оновлень та доповнень для поліпшення гри. Також важливо взаємодіяти з гравцями через спеціалізовані форуми, соціальні мережі та інші канали зв'язку, щоб отримати зворотний зв'язок та зрозуміти потреби іншого користувача. Це дозволяє забезпечити підтримку та розвиток гри після релізу.

1.3.2 Загальні відомості про процеси моделювання для предметної області.

Одним з найважливіших етапів у розробці гри є моделювання, яке дозволяє розробникам створювати абстрактні моделі гри та її компонентів для подальшого аналізу, проектування та реалізації.

Для моделювання комп'ютерних ігор використовуються різноманітні засоби, серед яких важливе місце посідає мова моделювання UML.

UML є стандартною мовою моделювання, розробленою з метою візуалізації, специфікації, конструювання та документування програмних систем[14]. Ця мова надає розробникам інструменти та термінологію, що дозволяють зрозуміло та однозначно комунікувати між собою та зі зацікавленими сторонами.

UML складається з набору графічних діаграм, які допомагають уявити, описати та аналізувати структуру та поведінку системи. Діаграми UML можуть

бути використані на різних етапах життєвого циклу розробки програмного забезпечення, починаючи з аналізу та визначення вимог, до проектування та реалізації. Деякі з ключових діаграм UML включають:

Діаграма прецедентів (Use Case Diagram) - Використовується для ідентифікації функціональних можливостей системи та опису взаємодії між акторами (користувачами) та системою.

Діаграми діяльності (Activity Diagrams) - Використовуються для моделювання послідовностей дій або процесів у системі, де кожна дія представлена вузлом, а переходи між вузлами показують потік виконання.

Діаграми компонентів (Component Diagrams) - Використовуються для моделювання архітектури системи та відображення компонентів, інтерфейсів та залежностей між ними.

1.3.3 Діаграма прецедентів.

Діаграма прецедентів є однією з основних діаграм UML і використовується для моделювання функціональних вимог до системи з точки зору її користувачів[15]. Вона допомагає ідентифікувати основних акторів (користувачі або зовнішні системи) та функціональні можливості (прецеденти), які система повинна забезпечувати.

У діаграмі прецедентів актори представлені зовнішніми сутностями, які взаємодіють з системою. Прецеденти відображають окремі функції або операції, які можуть бути виконані користувачами або зовнішніми системами. Вони описуються назвою, описом та взаємодією з акторами.

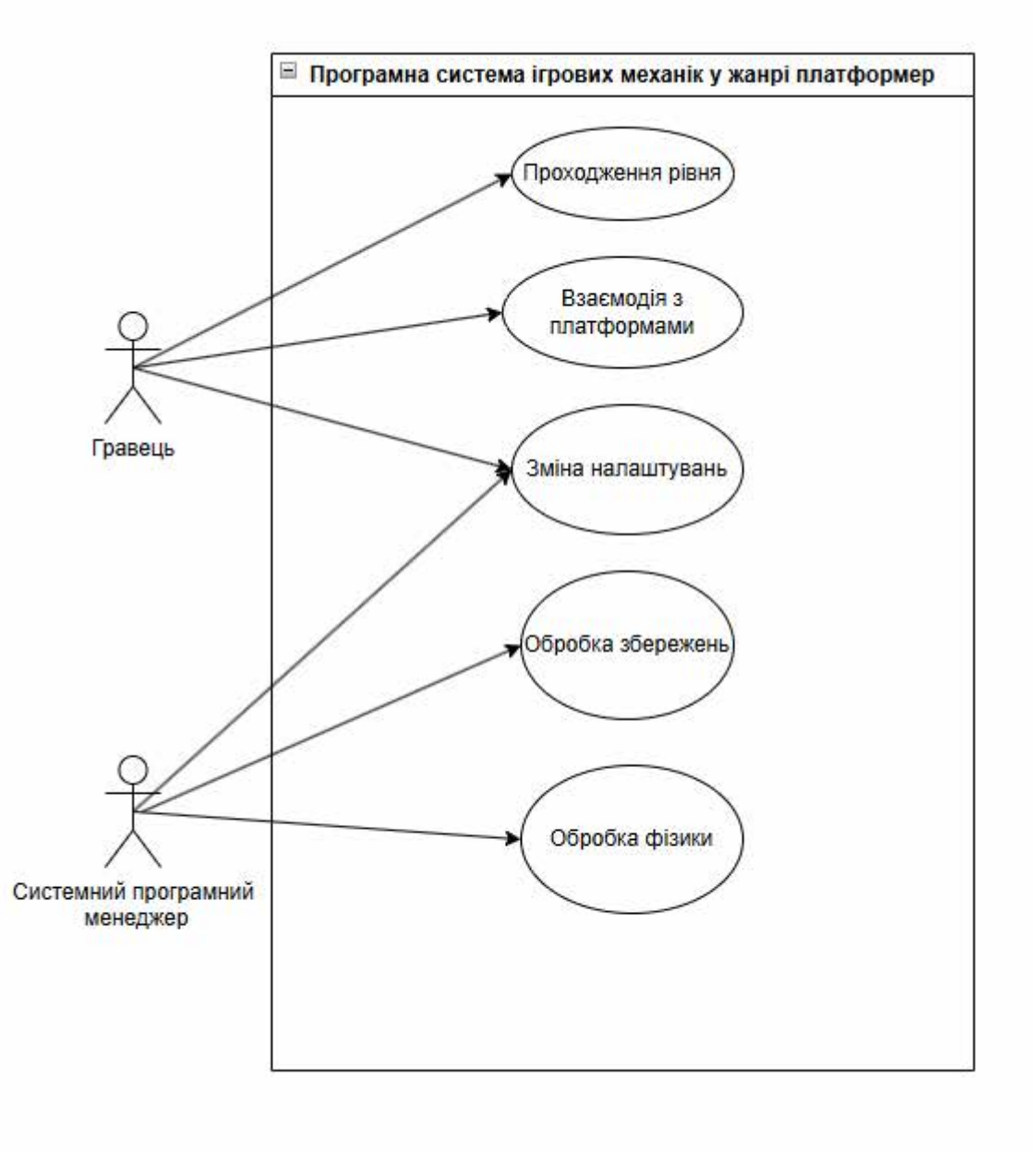


Рис 1.4 Діаграма прецедентів

1.3.4 Діаграма діяльності.

Щоб краще розуміти послідовність дій, рішень та взаємодій між різними об'єктами та акторами у конкретному сценарії, використовують діаграму діяльності[16]. Суть діаграми діяльності полягає у візуалізації послідовності та логіки виконання дій, яка допомагає аналізувати та розробляти процеси в системі.

Основними компонентами діаграми діяльності є наступні елементи:

- Початок та кінець -Початок вказує на момент створення потоку, а кінець показує його завершення.
- Дія - є основним елементом діаграми діяльності і представляє конкретну дію або операцію, яку виконує потік.
- Блок рішення - елемент використовується для вибору між декількома альтернативними шляхами або гілками потоку. Він представляється у вигляді ромба і включає умови, за яких потік обирає певну гілку.
- Зв'язки між елементами - вказують на послідовність виконання дій у діаграмі діяльності. Вони з'єднують різні елементи діаграми і показують напрямок руху потоку від початку до кінця.

Дивлячись на зображену на Рис. 1.5 діаграму діяльності для гри, можна зрозуміти загальний потік роботи системи. Починаючи з запуску гри користувачем, відбувається послідовний набір дій, які включають в себе вибір головного меню, завантаження рівня та проходження геймплею.

Після запуску гри, користувачу надається доступ до головного меню, де він може вибрати, почати нову гру або продовжити гру. В залежності від вибору, система завантажує відповідний рівень: початок рівня для нової гри або чекпоінт, на якому гравець закінчив попередню ігрову сесію.

Під час проходження рівня, гравець взаємодіє з об'єктами та проходить перешкоди. Якщо гравець успішно проходить до фіналу рівня, він отримує доступ до наступного рівня.

Якщо гравець входить до кімнати наступного рівня, система автоматично зберігає прогрес. Після цього подальший розвиток гри залежить від дій користувача. Він може продовжувати проходити рівень та просуватися по грі, вручну зберегти прогрес серед проходження рівня або просто вийти з гри.

Звертаючи увагу на діаграму діяльності, можна простежити послідовність дій та взаємозв'язки між компонентами гри. Вона надає візуальне уявлення про

загальну структуру та потік роботи системи, що допомагає розробникам у вдосконаленні геймплею та покращенні користувацького досвіду.

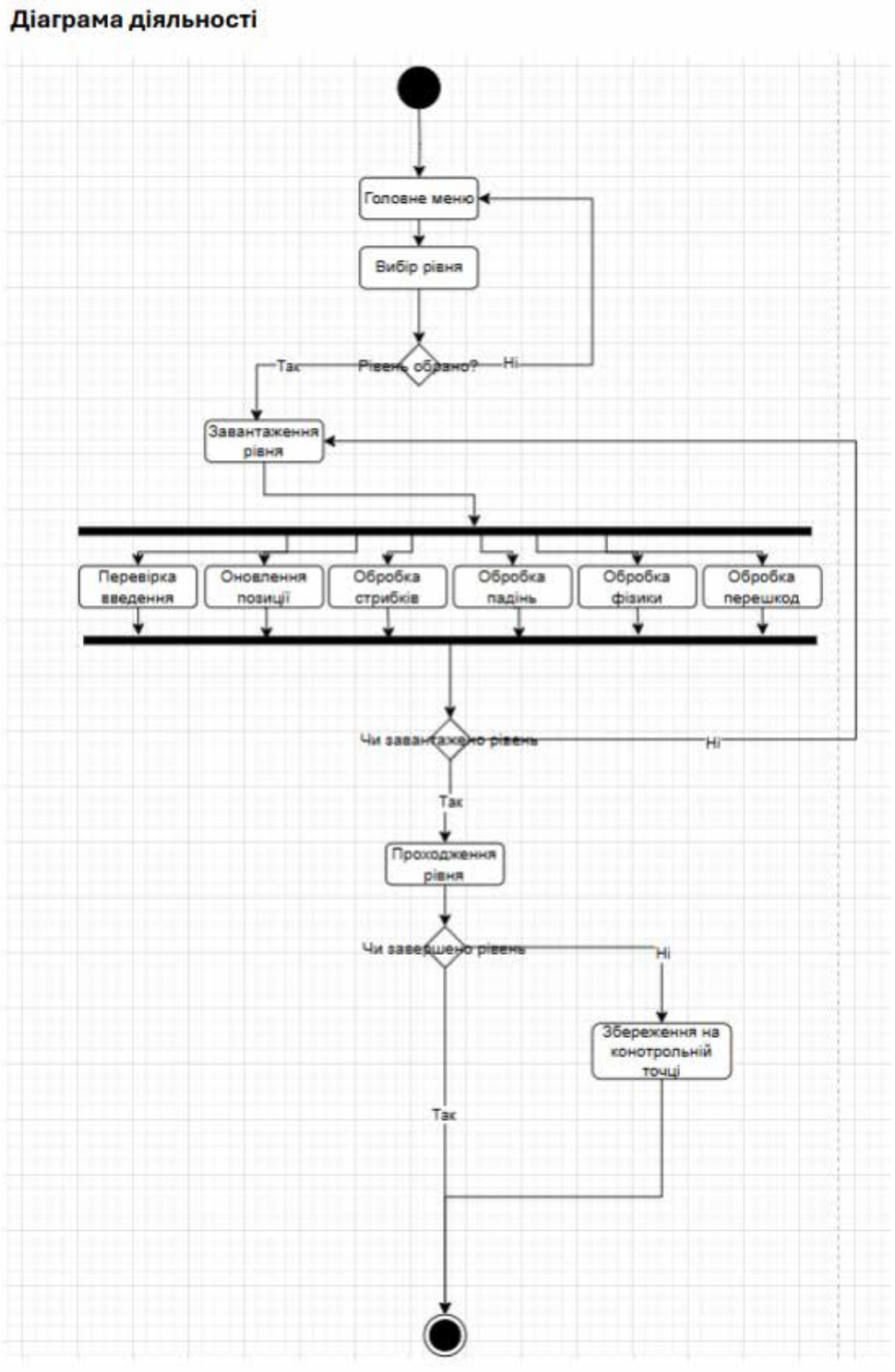


Рис. 1.5 Діаграма діяльності

1.3.5 Діаграма послідовності.

Діаграма, яка показує послідовність дій в тому чи іншому процесі або в системі в цілому. Лінія життя йде згори до низу, показуючи послідовність.

Приклад окремого процесу:

Рух гравця: як ми бачимо на рис 1.6, починається рух з введення команд до контролера, після чого система отримує сигнал від контролера, та оновлює позицію, після чого виводить зміни положення на екран гравця. Трохи нижче показано цикл руху представленого вище, та зупинки персонажа, якщо гравець не надає нових команд.

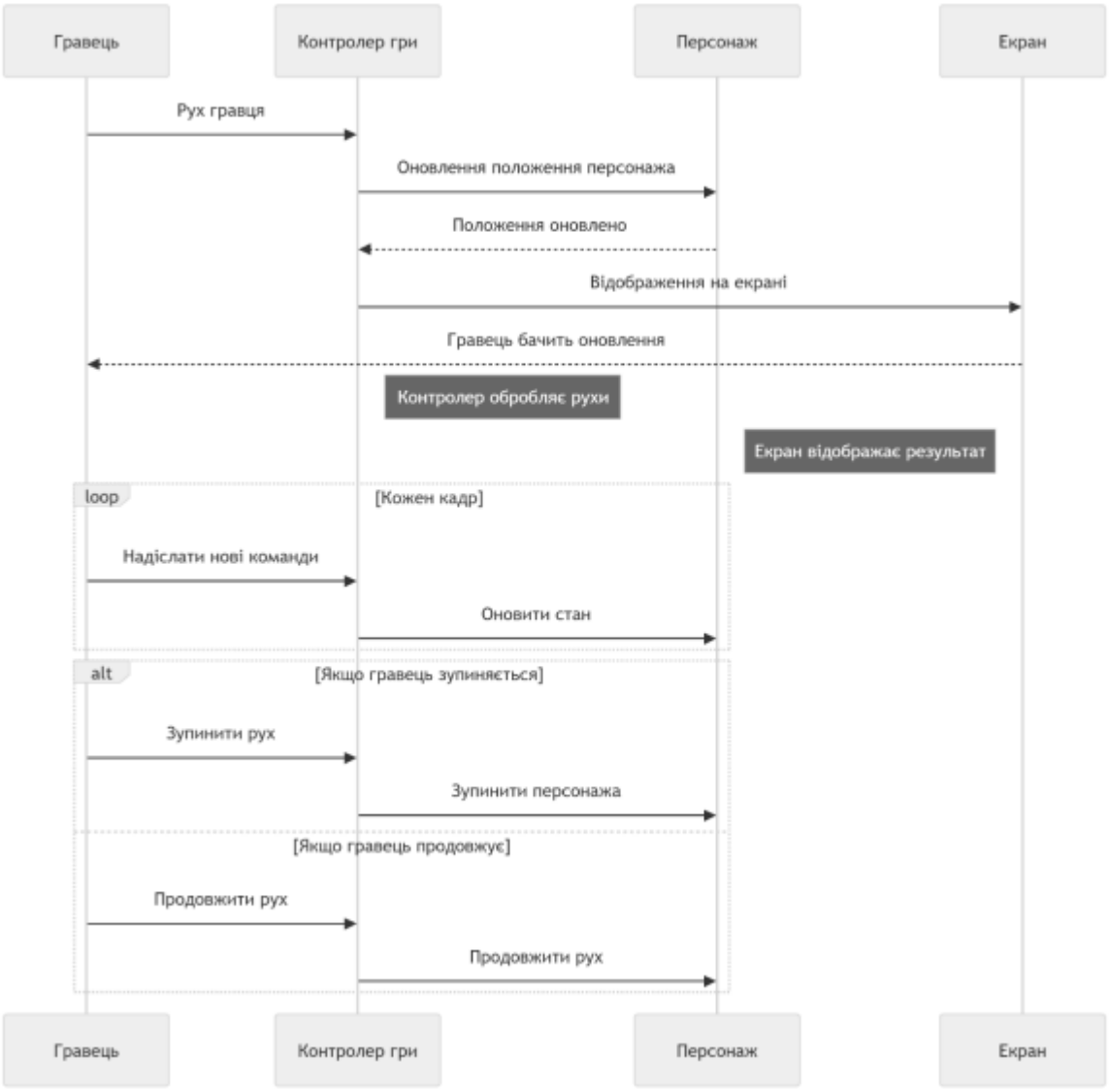


Рис. 1.6 Діаграма послідовності для руху гравця.

2 ІНФОРМАЦІЙНЕ ПРОГРЕМНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних або логічна схема - це абстрактна модель, яка виражає структуру та взаємозв'язки даних в певній предметній області. Вона не залежить від конкретної технології зберігання даних і не визначає фізичні аспекти реалізації. Замість цього, логічна модель даних використовує абстрактні структури, такі як реляційні таблиці, об'єктно-орієнтовані класи або XML-теги, для опису структури інформації та її взаємозв'язків[22].

Логічні моделі даних подають абстрактну структуру області інформації. Вони часто мають схематичний характер і найтиповіше використовуються у бізнес-процесах, які прагнуть захопити речі, що мають важливе для організації значення, та як вони відносяться одна до одної. Одного разу перевірена та схвалена, логічна модель даних може стати основою фізичної моделі даних і сформувати дизайн бази даних.

Побудова логічної структури даних має наступні причини:

- Розширює загальне розуміння елементів бізнес-даних і вимог, сприяючи кращій комунікації та взаєморозумінню між учасниками проекту.
- Виступає основою для проектування бази даних, допомагаючи визначити структуру, взаємозв'язки та обмеження даних.
- Забезпечує уникнення надмірності даних, що допомагає уникнути неузгодженості даних і бізнес-транзакцій.
- Сприяє повторному використанню та обміну даними, завдяки стандартизованій структурі та формату даних.
- Знижує час і вартість розробки та підтримки, прискорюючи процес проектування та забезпечуючи ефективну розробку бази даних.

- Підтверджує логічну модель процесів і допомагає аналізувати вплив змін на дані та бізнес-процеси.

Логічна модель , не зовісім коректний спосіб для показу даних ігрової системи платформера в Unity, так як даних , в простому розумінні слова в грі не багато. Фізична модель буде трохи коректніша, але не повністю відображає дані.

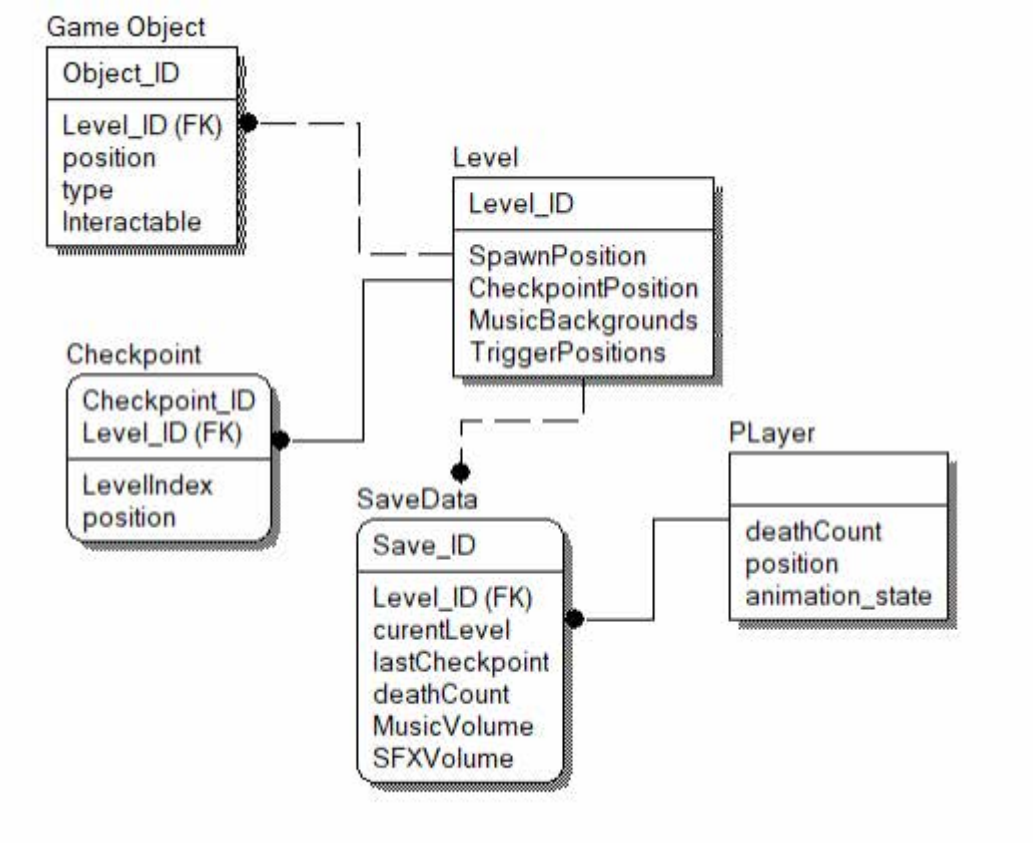


Рис. 2.1 Логічна модель даних

На рисунку 2.1 наведено модель даних , що відображає структуру, яка використовується для збереження прогресу гравця в грі.

Модель містить таблиці, які зберігають дані про чекпоінти, налаштування гравця, кількість смертей, позицію.

2.2 Дані в Unity

Вхідні та вихідні дані в Unity відіграють важливу роль у взаємодії з грою та зовнішнім середовищем. Вони дозволяють обмінюватись інформацією між різними компонентами системи, включаючи об'єкти гри, сцени, матеріали, анімації та інше.

Вхідні дані (input data) відображають вхідні сигнали, які гра отримує від користувача чи зовнішніх джерел. Це можуть бути рухи миші, натискання клавіш, введення з клавіатури, сигнали контролера або інші події, що відбуваються у реальному часі. Вхідні дані використовуються для керування рухом персонажів, взаємодії з об'єктами, виконання дій та активації різних функцій гри.

Вихідні дані (output data) представляють результати обчислень, стан об'єктів гри та іншу інформацію, яка передається з гри до зовнішнього середовища або інших компонентів системи. Це можуть бути графічні дані, звукові ефекти, статус гравця, результати виконання завдань, повідомлення та інші види інформації, які потрібні для відображення або обробки з боку зовнішніх систем або модулів гри.

У Unity вхідні та вихідні дані взаємодіють за допомогою різних механізмів, таких як обробники подій, системи введення. Це дозволяє гнучко керувати потоком даних у грі та забезпечувати взаємодію з користувачем та зовнішніми системами для досягнення бажаних результатів у відтворенні геймплею та взаємодії з грою.

2.3 Вибір системи управління базами даних

Застосування систем управління базами даних (СУБД) в ігровій індустрії є невід'ємною складовою процесу розробки та управління ігровими даними. СУБД дозволяють розробникам ефективно зберігати, керувати та отримувати доступ до великої кількості ігрової інформації, що включає в себе дані про персонажів, об'єкти, рівні, налаштування гри та багато іншого.

Одним із важливих аспектів використання СУБД в ігровій індустрії є можливість забезпечити постійне зберігання і доступ до гри у випадку перерв або відключень. Крім того, СУБД забезпечують можливість швидкого пошуку, сортування та фільтрації ігрової інформації, що відіграє важливу роль в оптимізації ігрового досвіду.

Багато популярних мультиплеєрних ігор сьогодні мають власну базу даних, яка виконує важливі функції для успішної гри. Ці бази даних відіграють критичну роль у зберіганні, обробці та управлінні великим обсягом ігрових даних.

Одна з головних функцій баз даних у мультиплеєрних іграх - це зберігання профілів гравців. Кожен гравець має свій особистий профіль з унікальними параметрами, які можуть включати досягнення, рівні, налаштування гри та іншу інформацію. База даних дозволяє зберігати ці дані і забезпечує можливість гравцям звертатися до своїх профілів та відстежувати свій прогрес у грі.

Декілька популярних мультиплеєрних ігор, які мають власну базу даних, включають "Fortnite" від Epic Games, "PlayerUnknown's Battlegrounds" (PUBG) від PUBG Corporation та "Overwatch" від Blizzard Entertainment. У всіх цих іграх база даних використовується для зберігання ігрових профілів, статистики гравців та інших важливих даних, які забезпечують функціональність та якість геймплею.

Player Prefs система вбудована в Unity, яка дозволяє зберігати налаштування гравця на диск комп’ютера , система потрібна для збереження найпростіших даних, параметри налаштувань поточний рівень, дуже корисно для швидкого використання найпростіших даних

JSON - це унікальний формат, для збереження даних, який дуже зручно використовувати в невеликих проектах, з одним гравцем . Ось деякі переваги та недоліки JSON:

Переваги:

- Простота використання: JSON має простий синтаксис та безліч можливостей для збереження даних.
- Легкість установки: JSON не потребує складної настройки та може використовуватися легко та просто .

Недоліки:

- Менша потужність: JSON не призначений для великих проектів з високим навантаженням, оскільки він може бути повільним при роботі з великими обсягами даних.

MySQL - це потужна та широко використовувана реляційна СУБД, яка використовує клієнт-серверну архітектуру[20]. Ось деякі переваги та недоліки MySQL:

Переваги:

- Висока продуктивність: MySQL здатний ефективно обробляти великі обсяги даних та забезпечувати високу продуктивність навіть при великому навантаженні.
- Розширені можливості: MySQL підтримує багато функцій та розширень, включаючи тригери, процедури та індекси, що дозволяє більш гнучке та потужне управління даними.
- Широка підтримка спільнотою: MySQL має велику та активну спільноту розробників, яка надає багато корисних матеріалів та допомогу.

Недоліки:

- Складність настройки: MySQL вимагає більшої складності для настройки та конфігурації, зокрема, потребує окремого сервера для роботи.
- Великі вимоги до ресурсів: MySQL може вимагати більше ресурсів комп'ютера та обладнання, що робить його важким у використанні на менш потужних системах.

Висновок: У випадку однокористувацької гри з низьким навантаженням, JSON в комбінації з Player Prefs є раціональним вибором для управління даними в проекті.

2.4 Реалізація моделі даних

Фізичний рівень інформаційної бази системи відображається у вигляді моделі, яка показує структуру та характеристики даних, як показано на рисунку 2.2. Ця модель представляє базу даних системи і включає поля, сутності, атрибути та типи даних, що будуть присутні у кінцевій базі даних. Це дозволяє чітко визначити, яким чином інформація буде представлена і організована в системі.

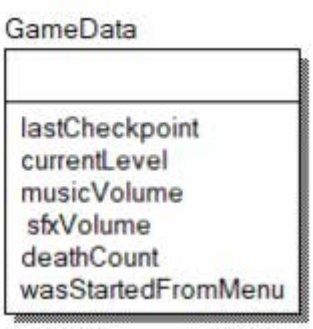
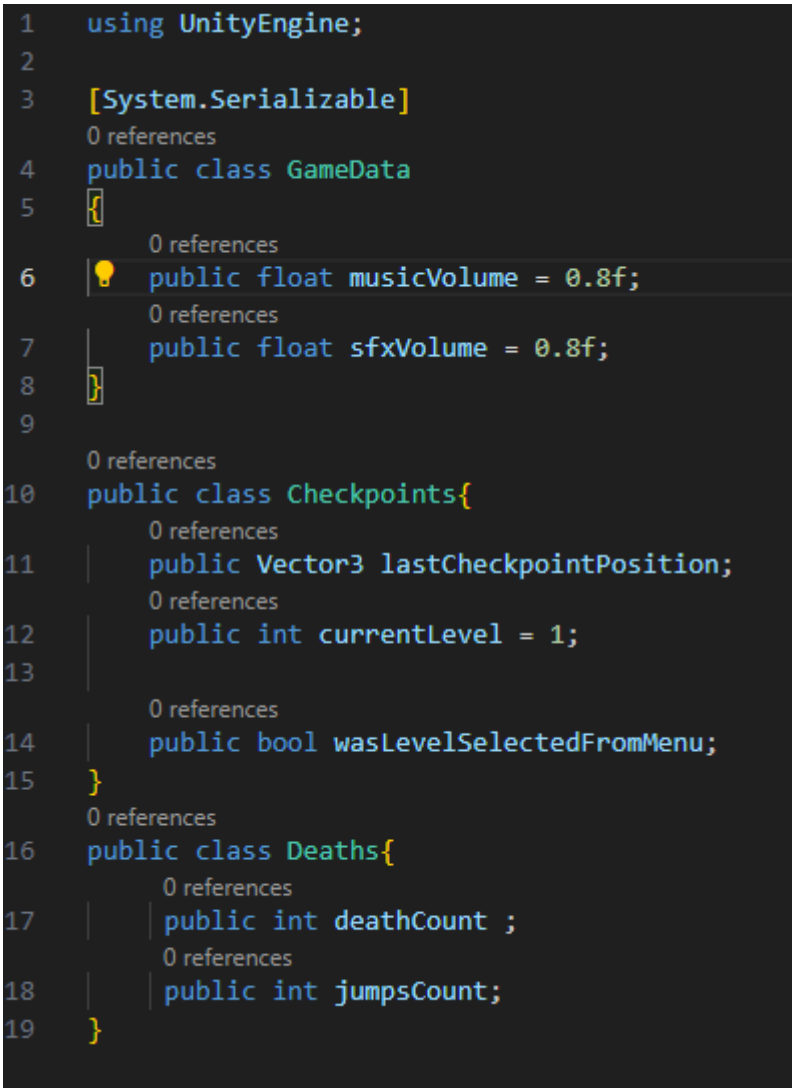


Рис. 2.2 Фізична модель даних

Фізчина модель не може відобразити взаємодію даних коректно в проекті, через специфіку проекту тому на рисунку 2.3 можна побачити коректнішу візуалізацію даних. Створено 3 типи даних для збереження в файлах JSON.



```
using UnityEngine;

[System.Serializable]
0 references
public class GameData
{
    0 references
    public float musicVolume = 0.8f;
    0 references
    public float sfxVolume = 0.8f;
}

0 references
public class Checkpoints{
    0 references
    public Vector3 lastCheckpointPosition;
    0 references
    public int currentLevel = 1;

    0 references
    public bool wasLevelSelectedFromMenu;
}
0 references
public class Deaths{
    0 references
    public int deathCount ;
    0 references
    public int jumpsCount;
}
```

Рис. 2.3 Створення типів даних в скрипті Unity

GameData для музики(використовується для класів які взаємодіють з Audio Mixer), Checkpoints дані для чекпоінтів та відстеження рівня(важливі дані для багатьох класів, для обробки їх),

Deaths для підрахунку смертей та стрибків(просто для статистики)

3 ПРИКЛАДНЕ ПРОГРЕМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організація програмного забезпечення

Розробка комп'ютерної гри для одного гравця у жанрі платформер ведеться однією особою. Така організаційна структура програмного забезпечення має свої переваги та особливості.

Однолюдна розробка дозволяє забезпечити високу ступінь контролю та впливу на весь процес розробки гри. Розробник, який веде проект, виконує всі основні ролі і відповідальності, пов'язані з розробкою гри. Він виступає як проектний менеджер, програміст, дизайнер і тестувальник одночасно. Така організаційна структура дозволяє забезпечити єдність бачення та контроль над усіма аспектами розробки, що сприяє швидкому прийняттю рішень та знижує ризик змішування комунікацій.

Однак, розробка однією людиною також може мати свої виклики. Особа, яка веде розробку, повинна мати широкий спектр навичок і знань в різних областях, що вимагає додаткових зусиль для оволодіння різноманітними аспектами гри. Важливо забезпечити організацію робочого процесу та відповідний контроль якості, щоб забезпечити успішне завершення проекту. Однолюдна розробка, нажаль дуже затяжний процес , бо один розробник виконує всі завдання самосітйно.

Організаційна структура для команди з одного розробника може включати такі елементи:

- Проектний менеджмент: Розробник відіграє роль проектного менеджера, встановлюючи мети, плануючи та контролюючи процес розробки гри. Він відповідає за розподіл завдань, управління ресурсами та визначення пріоритетів.

- Аналіз та проектування: Розробник проводить аналіз предметної області та вимог до гри, щоб визначити її функціональність та основні характеристики. Він також розробляє архітектуру гри, сценарії, механіку та інтерфейс.
- Програмування: Розробник виконує написання програмного коду, реалізуючи всі функції та механіки гри. Він забезпечує оптимізацію коду, обробку помилок та взаємодію з різними системами гри.
- Дизайн: Розробник відповідає за створення графічного дизайну, анімації та звукового супроводу гри. Він розробляє візуальні ефекти, персонажів, рівні та об'єкти гри, створюючи затяжний та захоплюючий геймплей.
- Тестування: Розробник здійснює перевірку та тестування гри, виявляючи та виправляючи помилки та недоліки. Він переконується, що гра працює належним чином та надає задоволення гравцям.

Під час розробки комп'ютерних ігор, розробники стикаються з важливим вибором - використати готові ігрові рушії, або ж написати власний з нуля.

Готові ігрові рушії, є потужними інструментами для розробки комп'ютерних ігор. Вони надають розробникам готовий набір функціональності, інструментів і ресурсів для створення ігор. Основні переваги використання ігрових рушіїв полягають у швидкості розробки, підтримці кросплатформності та наявності розширених функцій, таких як фізика, штучний інтелект та графічний рушій.

Одним із найпопулярніших ігрових рушіїв є Unity[21]. Він має велику спільноту розробників, багато ресурсів для вивчення і дозволяє розробляти ігри для різних платформ, включаючи комп'ютери, консолі та мобільні пристрої. Іншим варіантом є Unreal Engine, який володіє потужним графічним рушієм і надає можливості для реалістичної візуалізації та фотореалістичного рендерингу. Крім цього, існують інші ігрові рушії, такі як Godot, CryEngine та Construct, які також мають свої унікальні особливості та переваги.

Так як знадобляться 3D моделі, нам також буде потрібне середовище для створення моделей. На ринку присутні багато середовищ, таких як Cinema4D, Maya, Plasticity, ZBrush, але найбільше виділяється Blender3D. Середовище має в собі всі інструменти, такі як аніматор, скульптінг, рігінг, скріпти, малювання текстур. Серед всього софту вибір впав на Blender.

Написання власного двигуна для розробки комп'ютерних ігор - це складний та витратний процес, який вимагає великого обсягу знань та навичок у різних областях. Однак, власний ігровий двигун може надати розробникам більшу гнучкість, контроль та можливість реалізувати унікальні ідеї.

3.2 Вибір програмних засобів реалізації

3.2.1 Опис вимог до ігрового рушія.

Одна з найважливіших аспектів розробки гри - це вибір інструментарію, що використовується під час розробки. Складність розробки гри полягає в тому, що вона включає в себе багато різних елементів, таких як геймплей, графіка, фізика, штучний інтелект та інші. Кожен з цих аспектів потребує спеціалізованого інструментарію, який забезпечує розробникам необхідні функціональні можливості та зручність у роботі.

При виборі інструментарію для розробки гри, розробник повинен враховувати такі фактори, як його досвід, доступність, потужність та підтримка. Наявність потужного інструментарію з високим рівнем функціональності та гнучкості дозволяє розробнику реалізувати свої ідеї та створити захоплюючу гру. Підтримка інструментарію забезпечує розробнику можливість отримати допомогу та вирішити проблеми, що виникають під час розробки.

3.2.2 Обґрунтування вибору рушія Unity та Blender3D.

З урахуванням вищевказаних вимог, було обрано Unity як основний інструментарій для розробки програмного забезпечення для гри. Unity є потужним та простим, який надає розробникам широкий спектр функціональних можливостей для створення захоплюючих ігрових досвідів.

Unity відомий своєю універсальністю, розробляти можна і на мобільні платформи, і на PC. Рушій є основою багатьох ігр, через свою комфортність та гнучкість. Графічно Unity гірший за Unreal Engine 5, але може визвати конкуренцію.

Unity пропонує безліч особливостей та переваг для розробників ігор. Ось деякі з них:

- графічна потужність – Unity має вражаючі графічні можливості, завдяки чому розробники можуть створювати світи, деталізовані персонажі та захоплюючі візуальні ефекти.
- візуальне програмування – Unity надає потужну систему візуального програмування Visual Scripting, система нодів. Це дозволяє розробникам створювати складні інтерактивні системи, геймплейні механіки та взаємодію об'єктів без необхідності писати код. Це робить процес розробки більш доступним та швидким, забезпечуючи широкі можливості для творчості.
- мультиплатформеність – Unity підтримує різні платформи, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність. Це дозволяє розробникам створювати ігри, які можуть бути запущені на різних пристроях, забезпечуючи більш широку аудиторію гравців.
- розширені інструменти розробки – Unity надає широкий набір інструментів для розробки ігор, включаючи редактор рівнів, систему фізики, анімації, звуку, штучного інтелекту та багато інших. Ці інструменти дозволяють розробникам ефективно працювати над усіма аспектами гри і забезпечувати її якість та функціональність.
- Велика спільнота розробників: Unity має активну та велику спільноту розробників, яка надає підтримку, поради та ресурси для новачків і професіоналів у галузі розробки ігор. Це створює сприятливу середу для

обміну знаннями та взаємодопомоги, що допомагає розробникам вирішувати проблеми та досягати кращих результатів.

Visual Scripting- це візуальний інструмент для програмування в Unity. Він надає швидку інтерактивну розробку, що дозволяє створювати логіку гри без прямого писання коду. Visual Scripting корисні у таких ситуаціях:

- прототипування та швидка розробка: Завдяки візуальному підходу, Visual Scripting дозволяють швидко створювати та тестувати ідеї без необхідності в написанні коду. Вони особливо корисні на ранніх стадіях розробки, коли потрібно швидко перевірити та валідувати концепції геймплею.

C# є потужною та універсальною мовою програмування, яка широко використовується в ігровій індустрії. Вона надає розробникам гнучкість та повний контроль над реалізацією геймплею, штучного інтелекту, фізики, графіки та багатьох інших аспектів гри[3].

C# використовується, коли потрібно створити складну, оптимізовану та розширювану логіку гри. Він надає розробникам повний контроль над кожним аспектом гри і дозволяє реалізувати високошвидкісні обчислення, оптимізовані алгоритми та складні системи. Використання C# особливо корисне в таких випадках:

- реалізація складних алгоритмів: Якщо вам потрібно створити складну логіку, що вимагає оптимізації та високої продуктивності, то використання C# дозволяє вам написати ефективний код, який виконується швидше за використання Visual Scripting.
- розробка спеціалізованих функціональних можливостей: Якщо вам потрібно створити унікальні функції або системи, які не входять до стандартного набору функціональності Unity, використання C# дозволяє вам розширити можливості двигуна та реалізувати специфічні потреби вашої гри.

- інтеграція зі зовнішніми бібліотеками та сервісами: C# надає можливість легко інтегрувати зовнішні бібліотеки та сервіси у вашу гру. Це особливо корисно, якщо ви плануєте використовувати сторонні API, штучний інтелект, фізичні симуляції та інші функції, що необхідні для вашого проекту.

Отже, використання C# або Visual Scripting залежить від потреб проекту. C# надає більше контролю, оптимізації та можливостей розширення, тоді як Visual Scripting дозволяють швидше прототипувати через інтерактивну розробку. Здебільшого, комбінація обох підходів використовується для досягнення найкращих результатів у розробці гри.

3.2.3 Набір програмних модулів та компонентів Unity.

Unity має широкій набір програмних модулів, від керування музикою до анімацій та ефектів.

Unity Engine – забезпечує все для функціонування гри та сцен в рушії, фізика та логіка Unity.

Animator – модуль для створення анімацій та циклів анімацій, для обробки подій та запуску анімацій при цих подіях. Дуже зручний модуль, для вза’модії з анімаціямі, налаштуванням анімацій та тригерів анімацій.

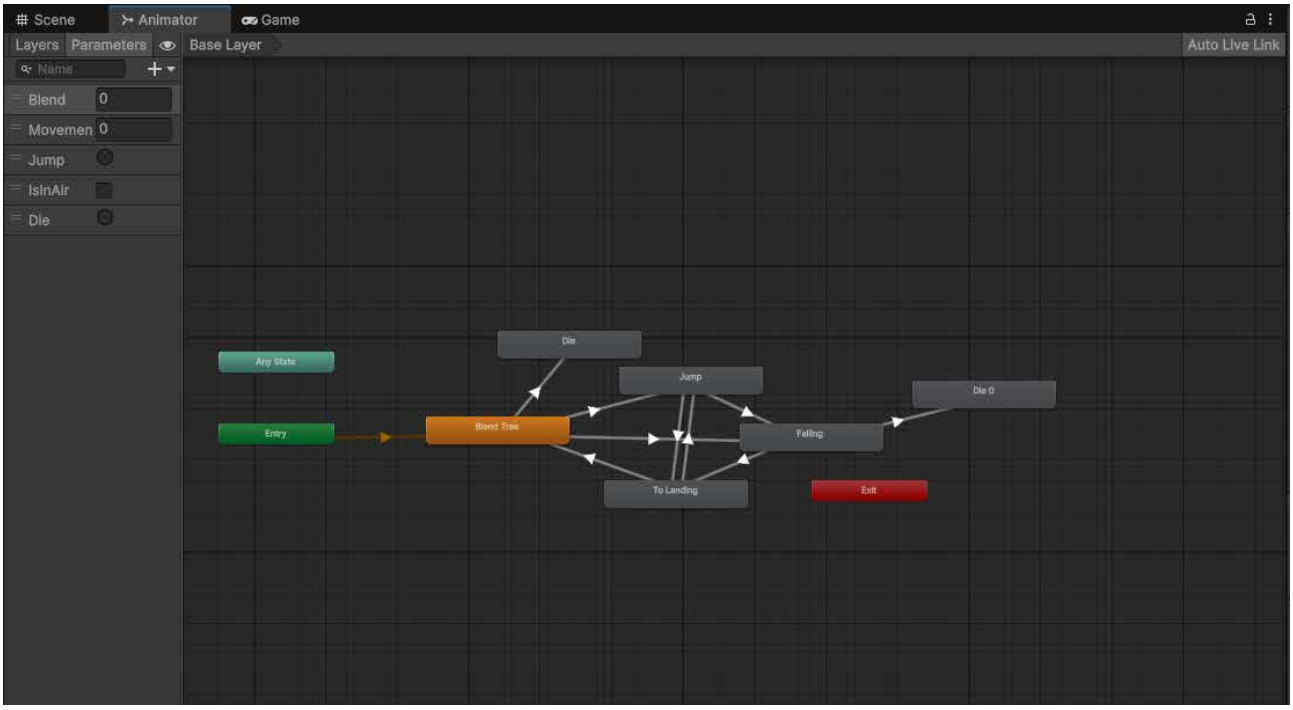


Рис. 3.1 Модуль Animator

Rendering Engine– надає можливість додати ефекти та текстури для більш гарної та аутентичної картинки

AudioMixer – модуль для керування музикою та звуками в цілому.

3.3 Класи

Класи або скріпти – логіка проекту, в скриптах пишеться логіка для елементів взаємодії, музики, логіки даних, збережень.

Основні класи

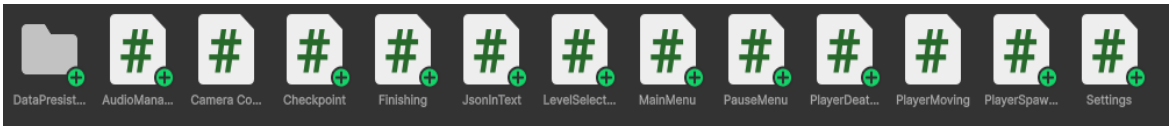


Рис. 3.2 Класи

PlayerMoving – відповідає за систему руху гравця, використовуючи Vector для запису позиції гравця, при натисканні на W показник X в Vector3 змінюється в негативну сторону, та для кожної з клавіш руху еквівалентно. Модель персонажа

повертається за направленням натискання. Також в скрипті викликається аніматор , для коректних анімацій бігу.

```
void Update()
{
    float h = Input.GetAxis("Horizontal");
    float v = Input.GetAxis("Vertical");
    Vector3 directionVector = new Vector3(-v , 0 , h);
    if(directionVector.magnitude> Mathf.Abs(0.05f))
    transform.rotation = Quaternion.Lerp(transform.rotation,Quaternion.LookRotation(directionVector),Time.deltaTime * rotationSpeed);

    animator.SetFloat("Movementspees",Vector3.ClampMagnitude(directionVector,1).magnitude);
    Vector3 moveDir = Vector3.ClampMagnitude(directionVector,1) * speed;
    rigidbody.linearVelocity = new Vector3(moveDir.x,rigidbody.linearVelocity.y,moveDir.z);
    rigidbody.angularVelocity = Vector3.zero;

    if (Input.GetKeyDown(KeyCode.Space)){
        Jump();
    }
    if(Physics.CheckSphere(groundCheckerTransform.position,0.3f,  notPlayerLayer)){
    animator.SetBool("IsInAir", false);
    }
    else {
        animator.SetBool("IsInAir",true);
    }
}
```

Рис. 3.3 Фрагмент коду для руху з класа PlayerMoving

```
void Jump()
{

    if(Physics.Raycast(groundCheckerTransform.position,Vector3.down, 0.3f, notPlayerLayer)){
        animator.SetTrigger("Jump");
        rigidbody.AddForce(Vector3.up* jumpForce,ForceMode.Impulse);
    Deaths data = SaveSystem.LoadDeaths();
    data.jumpsCount += 1;
    SaveSystem.SaveDeaths(data);
    }

}
```

Рис. 3.4 Фрагмент коду для стрибків з класа Player Moving

CameraControl – клас руху камери, камера рухається суворо по 2 осям та слідує за гравцем , не може змінювати третю. Також використовує Vector3 з показниками X,Y,X.

AudioManager -для керування аудіо в Unity, та пізніше для керування налаштуваннями музики.

Checkpoint – клас який надається Pivot,Pivot стає чекпоінтом , та гравець буде з’являтися на ньому якщо помре. Координати беруться з файлу JSON

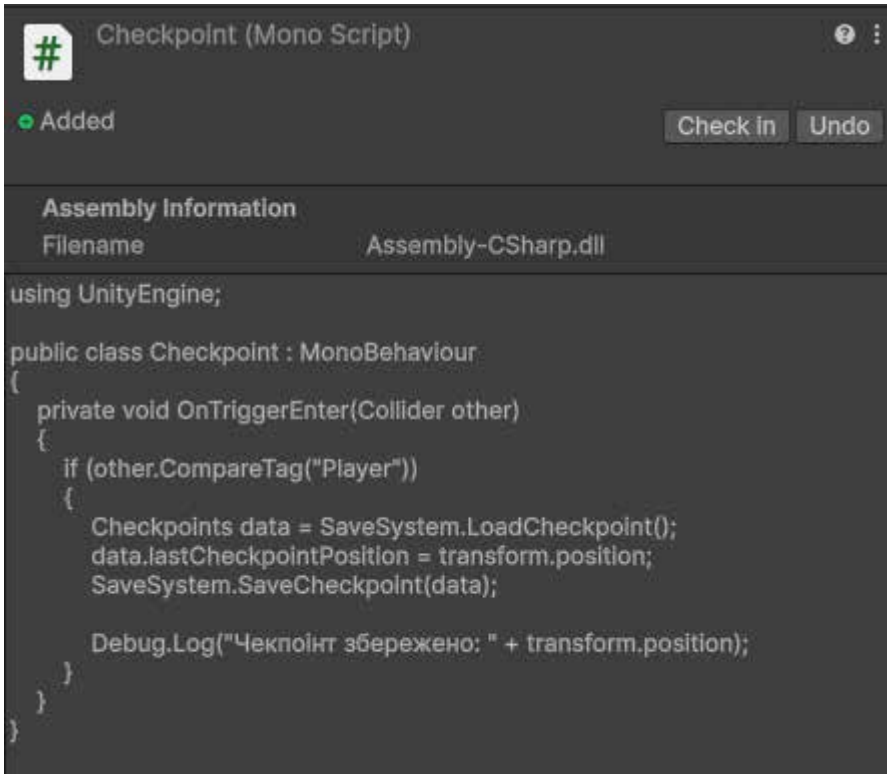


Рис. 3.5 Клас Checkpoint

MainMenu та PauseMenu – для керування меню. MainMenu надає логіку кнопкам, пускає в гру, дає виконати налаштування, чи вийти з гри.

PauseMenu зупиняє час в грі, та дає можливість повернутися в головне меню, чи налаштувати звук.Використовує Unity.SceneManager та Unity.UI для керування інтерфейсом та сценами, відправляти лані дофайлів та гри.

SaveSystem – клас для збереження та вивантаження з/в JSON файли. Зберігається в кілька файлів, розбитих на категорії, чекпоінти , збереження музки, статистика.

```
5 references
private static string savePath = Path.Combine(Application.persistentDataPath, "save.json");
3 references
private static string savePathMusic = Path.Combine(Application.persistentDataPath, "saveMusic.json");
3 references
private static string saveCheckpoint = Path.Combine(Application.persistentDataPath, "savecheckpoint.json");
3 references
private static string saveDeaths = Path.Combine(Application.persistentDataPath, "savedeaths.json");

0 references
public static void Save(GameData data)
{
    string json = JsonUtility.ToJson(data);
    File.WriteAllText(savePath, json);
}
```

Рис. 3.6 Система збережень

Player.log	22.05.2025 13:50	Текстовый докум...	4 КБ
Player-prev.log	22.04.2025 00:20	Текстовый докум...	3 КБ
savecheckpoint.json	23.05.2025 13:19	Исходный файл J...	1 КБ
savedeaths.json	22.05.2025 13:50	Исходный файл J...	1 КБ

Рис. 3.7 Файли збережень

C: > Users > andys > AppData > LocalLow > Bushnyi > SomeGame > {} savecheckpoint.json > wasLevelSelectedFromMenu

```
{"lastCheckpointPosition":{"x":0.364300012588501,"y":12.0,"z":125.05999755859375},
"currentLevel":1,
"wasLevelSelectedFromMenu":false}
```

Рис. 3.8 Файл savecheckpoint.json

PlayerSpawner – визначає де був гравець востаннє, та гравець з’являться на чекпоінті якщо проходив його, або на початку рівня. Координати беруться з файлу де записувався останній чекпоінт, і там з’являється персонаж.

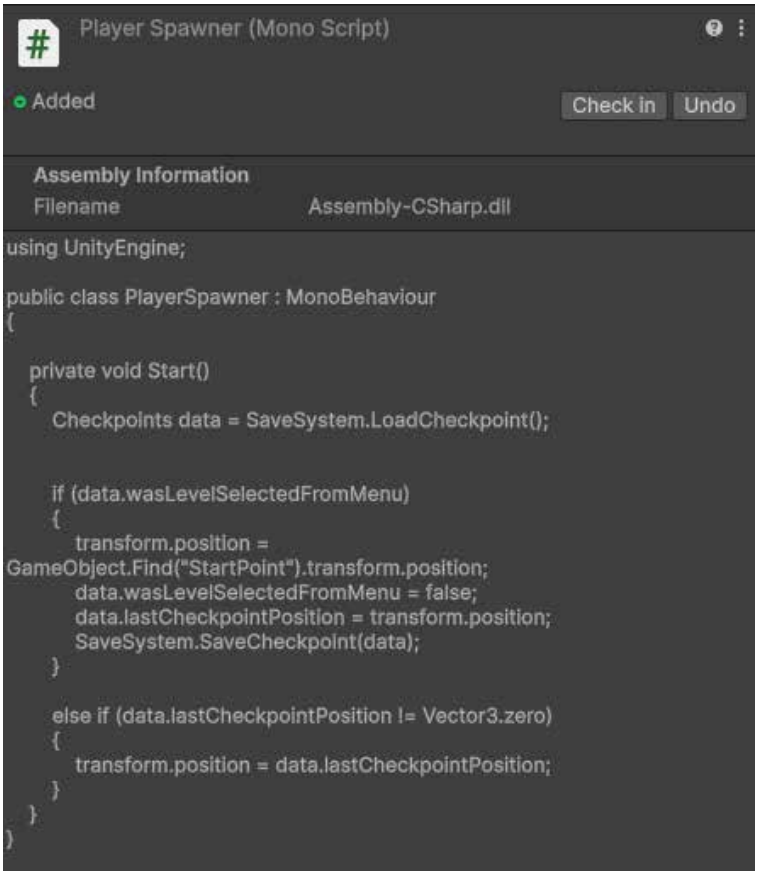


Рис. 3.9 Фрагмент коду класа Player Spawner

Finishing – клас який перевіряє чи фінішував гравець, якщо так то відправляє його на наступний рівень якщо такий є, а якщо ні то в головне меню.

Finishing (Mono Script)

Added Check in Undo

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class Finishing : MonoBehaviour
{

    public Text endText;
    public float delayBeforeMenu = 3f;

    private bool hasEnded = false;

    void Start()
    {
        if (endText != null)
            endText.enabled = false;
    }

    void OnTriggerEnter(Collider other)
    {
        if (!hasEnded && other.CompareTag("Player"))
        {
            hasEnded = true;
            if (endText != null)
                endText.enabled = true;

            endText.text = "Thanks for playing demo...";
            Invoke("ReturnToMenu", delayBeforeMenu);
        }
    }
    void ReturnToMenu()
    {
        SceneManager.LoadScene("Menu");
    }
```

Рис. 3.10 Фрагмент класа Finishing

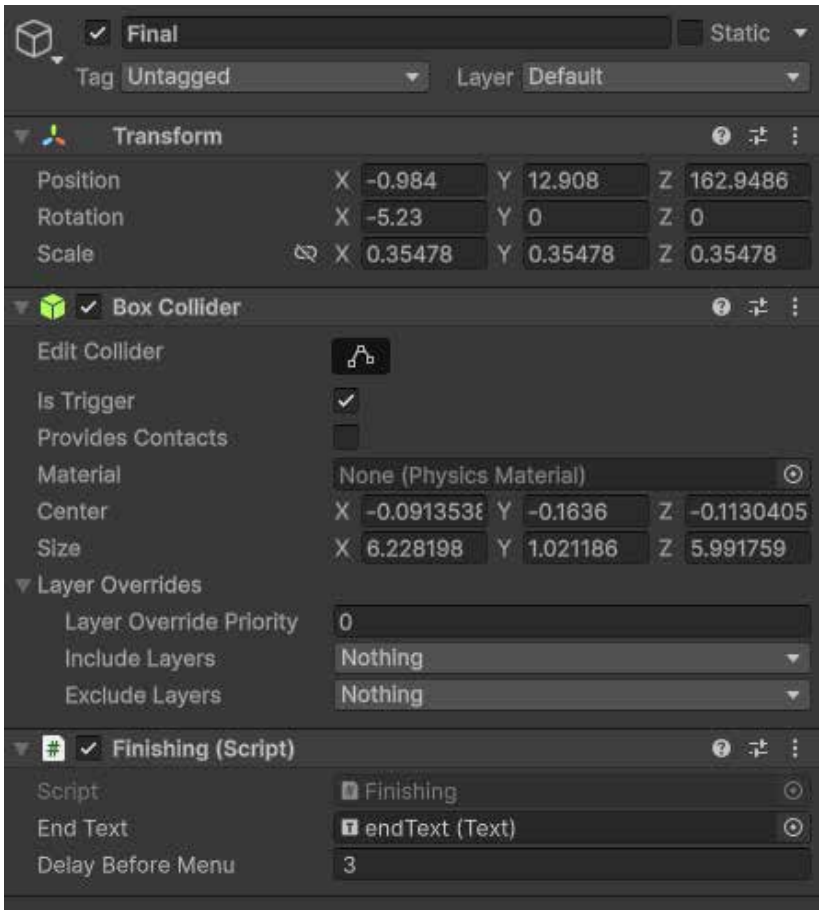


Рис.3.11 Компоненти Empty об’єкта

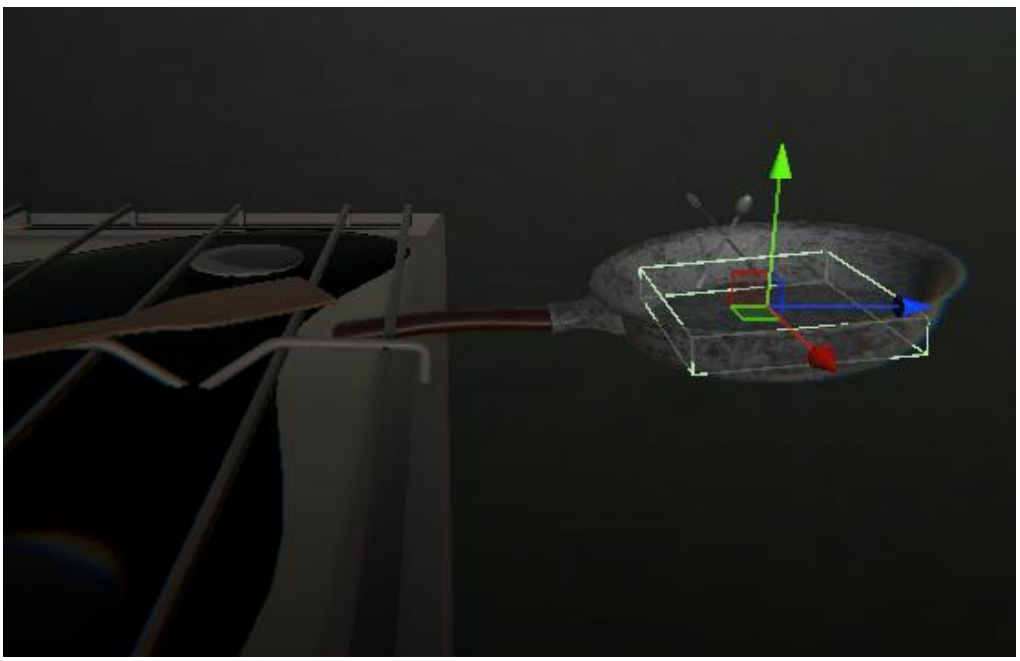


Рис.3.12 Візуалізація блока колайдера фінішу в грі

PlayerDeathControl – рахує кількість смертей гравця, за записує в статистичний файл JSON.Також перевіряє чи не стоїть персонаж на поверхні с тегом “Dead”.

```
private void OnCollisionEnter(Collision collision)
{
    if (IsInDeathLayer(collision.gameObject))Die();
}


private void OnTriggerEnter(Collider other)
{
    if (IsInDeathLayer(other.gameObject)) Die();
}

    private bool IsInDeathLayer(GameObject obj)
{
    return deathLayer == (deathLayer | (1 << obj.layer));
}


private void Die()
{
    if (isDead) return;

    isDead = true;
    Deaths data = SaveSystem.LoadDeaths();
    data.deathCount += 1;
    SaveSystem.SaveDeaths(data);

    if (movementScript != null)
        movementScript.enabled = false;


    if (animator != null)
        animator.SetTrigger("Die");



    Invoke(nameof(Respawn), deathDelay);
}


private void Respawn()
{



SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);


}


}
```

Рис. 3.13 Фрагмент коду класу PlayerDeathControl

Основий функціонал класу PlayerDeathControl реалізовано через теги. Якщо гравець наступає на поверхню або предмет з тегом “Dead” , то в персонажа починається анімація смерті, смерь записується в JSON файл, а персонаж спавниться на чекпоінті, або на початку рівня, залежить від того де помер персонаж.

3.4 Реалізація програмних модулів

Цей розділ включає детальний опис програмних модулів, які використовуються в системі гри для досягнення її функціональності та взаємодії з користувачем.

Рис. 3.14 Діаграма послідовності для стрибка

Як видно з рисунку 3.14 , стрибок дозвляє змінювати координати персонажа,по вертикалі, змінюючи своє положення в ігровому світі.

Логіка не дуже складна, але анімації стрибку річ складніша.Для цього ми використовуємо аніматор. При настисканні Space персонаж змінює свої координати X,Y,Z та одночасно з цим запускає анімацію стрибка(Попередньо зроблену в програмному середовищі для розробки 3D Blender)

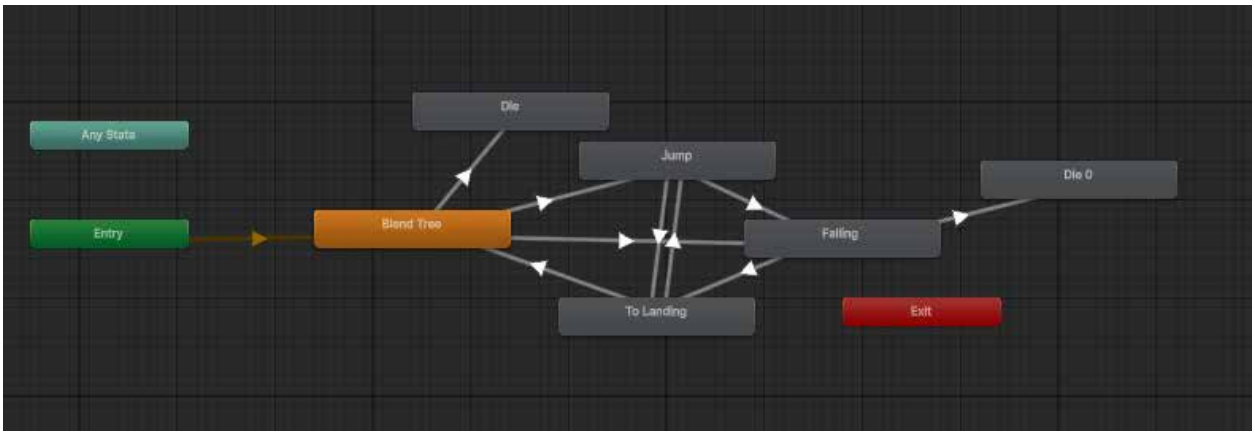


Рис.3.15 Дерево станів анімацій персонажа

Анімації,для плавності переходять один в одну, щоб персонаж не ковзав по карті, або під час стрибку не рухав ногами ніби він бігає.Зі спокійного стану, при натисканні Space запусскається анімація стрибку, яка переходить в анімацію падіння якщо персонаж в повітрі, якщо персонаж торкнувся землі(поверхні з певним тегом), то запускається анімація приземлення, з подальш переходом в стан спокою. Також оброблено нестандартні ситуації коли після преземлення персонаж знову падає.

Анімація смерті може настати з двох станів, зі стану руху, або з падіння, смерть наступає якщо персонаж наступає на поверхню з тегом “Dead”.

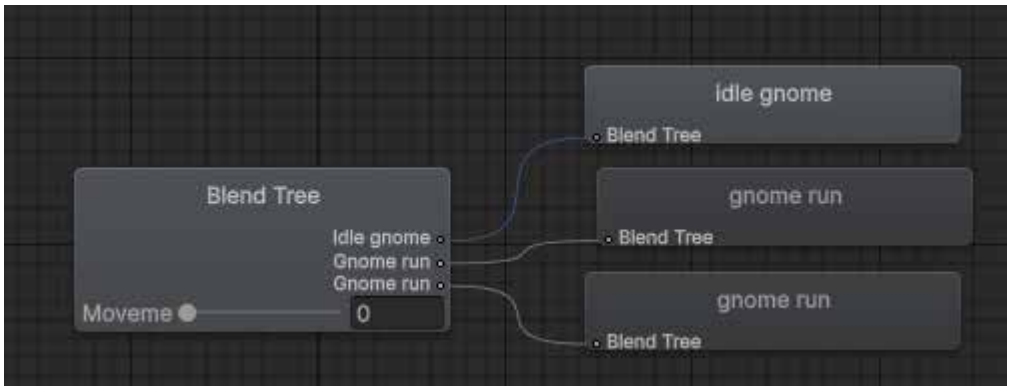


Рис. 3.16 Дерево анімації руху

Для руху зроблено 3 стани для більш плавного старту з переходом в повну швидкість.

Audio Mixer – модуль для роботи з музикою та звуками.В нашій ситуаціх використовуєтсья для керування музикою та для інтеграції музики в гру.

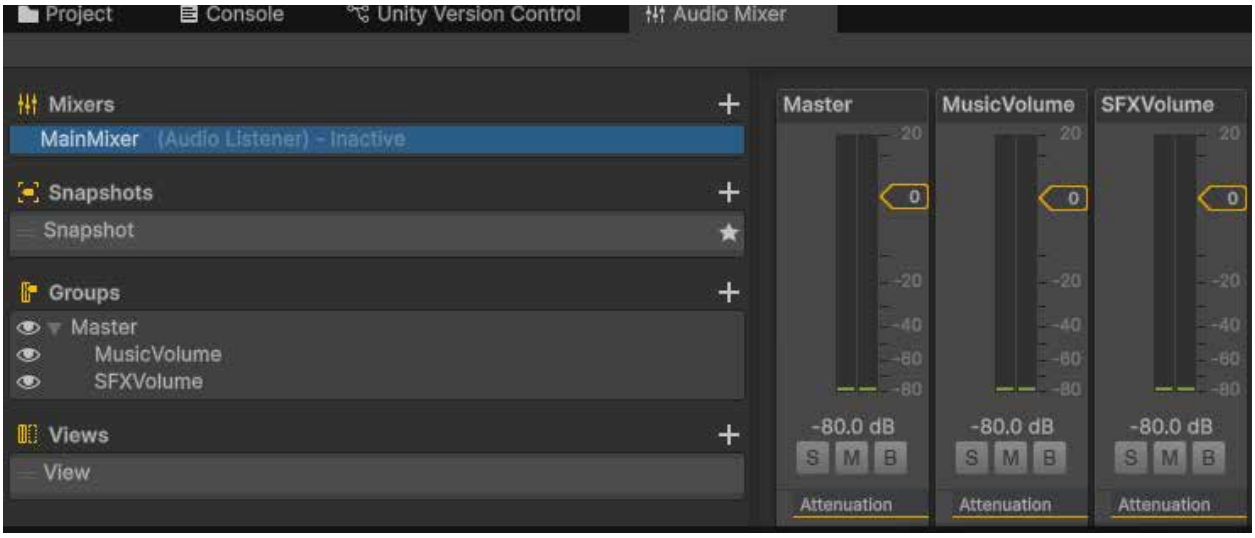


Рис.3.17 Керування звуками

Передбачено 3 параметри:Master,Music,SFX.

Master – для керування рівнем гучності в цілому.

SFX- для керування гучності звуків світу.

Music- для курування рівнем гучності музики

Використовуються в скриптах для керування і збереження налаштувань гучностію.

Unity Renderer - модуль для додання ефектів картинці, створення блюру і тд. До сцени було додано пост-ефекти для надання картинці трох моторошнішого вигляду.

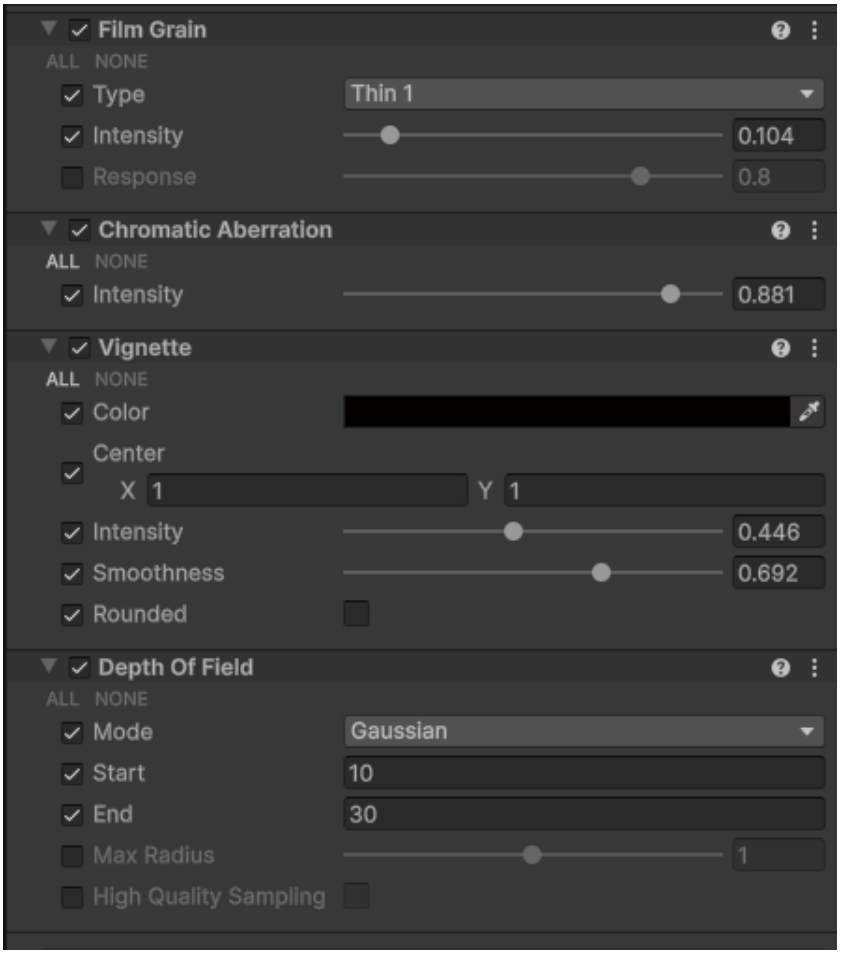


Рис.3.18 Налаштування пост-ефектів

Unity Engine – модуль використовувася для імпорту моделей створених до цього в Blender, імпортувалися модель(Mesh) з текстурам окремо, пізніше текстури примінялися на Mesh .На рисунку 3.17 бачимо що імпорт пройшов чудово.

Об’єкти зі сцени на рисунку 3.18 імпортуться окремо, через особливість імпорту в Unity.В блендер вісь Z , це вертикальна вісь, а X та Y горизонтальні, а в Unity Y цце вертикальна вісь, тому при імпорті можуть бути деякі складності, моделі можуть бути перевернутими або деформованими. Імпорт відбувався в форматі FBX, рідному для Unity, з перевернутими спеціально для цього осями в Blender. Текстури експортувалися з Blender окремо в форматі JPG.

Рис.3.19 Модель персонажа

Рис. 3.20 Сцена для імпорту в Unit

За допомогою Unity EnginePеалізуються системи колайдерів та руху.Colider-це компонент у фізичній системі гри, який визначає форму об'єкта для зіткнень.За допомогою них задається можливість торкатися об’єктів.Колайдери в Unity мають кілька форм, BoxColider, CapsuleColider, MeshColider. Кожен з них використовується в проекті.

Приклади:BoxColider використовується для поверхонь , стола наприклад по якому йде персонаж.

CapsuleColider використовується для колайдинга персонажа, а MeshColider універсальний колайдер, для всього , має більше полігонів .

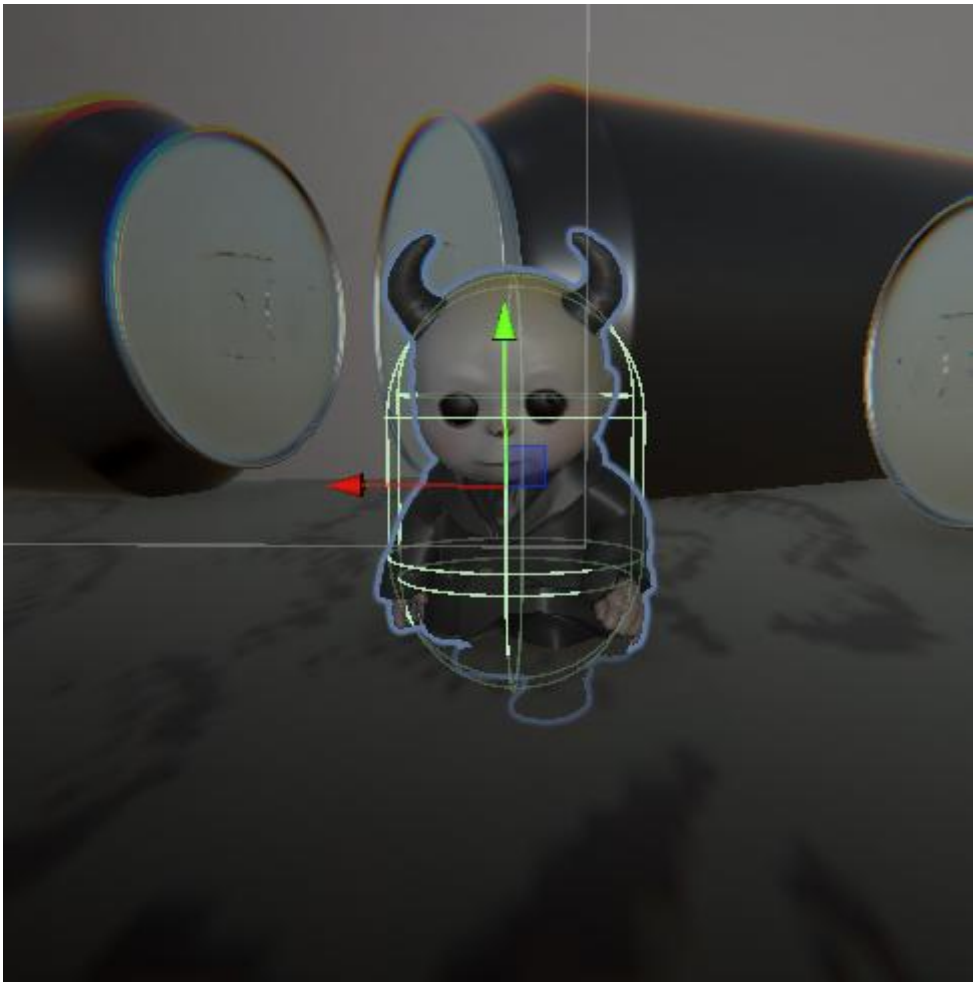


Рис. 3.21 Приклад CapsuleColider

Тут використано 2 CapsuleColider, щоб персонаж коректно взаємодіяв з поверхнями , та не запускалися анімації які заважають іншим.

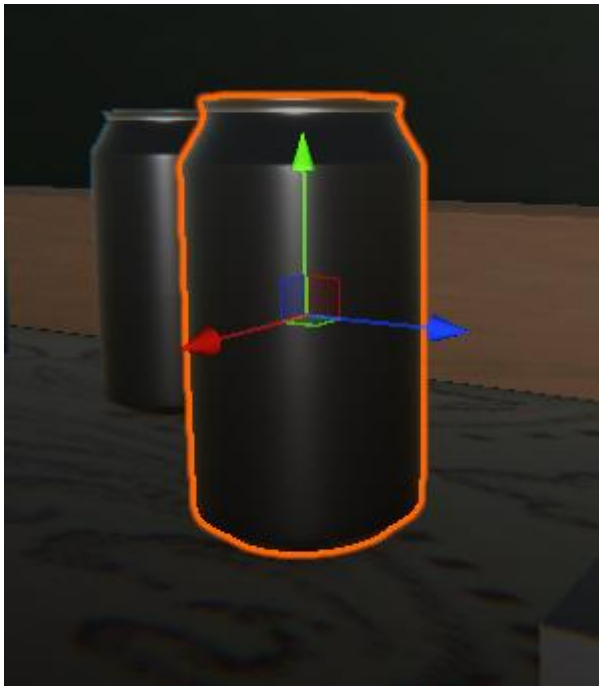


Рис. 3.22 MeshColider

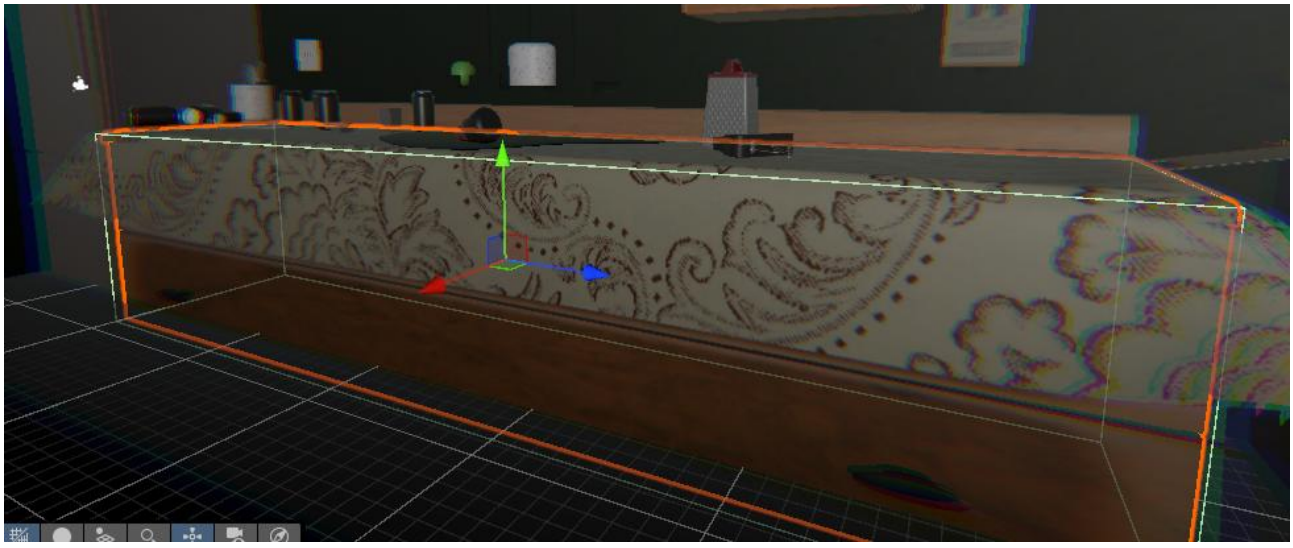


Рис. 3.23 BoxColider

3.5 Результат роботи

Проект , зі всіма його едементами та модулям готовий до запуску, модулі працюють коректно , налаштування проекту виконані.

При запуску гри відкривається головне меню, через яке можно запустити гру або вбрати рівень. Можно вибрати нову гру, чи продовжити з чекпоінта, збереженого в файлі. Як видно на рисунку 3.25 ,також надається можливість

переглянути статистику, кількість стрибків, скільки разів ігровий персонаж програвав анімацію смерті, або інакше кажучи помирав.

Також доступне меню налаштувань , в якому можна зробити звук тихіше або гучніше, або зовсім вимкнути . Після проходження елементів , на рівні присутній чекпоінт , який збереже місце спавну після смерті. Звідси вже починається фінальний відрізок.

Рис. 3.23 Головне меню

Натискаючи “New Game”, гра почнеться спочатку.

Рис. 3.24 Поява напочатку рівня

Після появи , цілю є дойти до фінішу, на шляху будуть кілька елементів з платформінгом(елементи рівня, де геймплей орієнований на вміння гравця).

Гра , не має сюжету, але орієнтуючись на ігри приклади, видно що сюжет, який розповідається напряму не потрібний, світ надає сюжет, та кожен інтерпритує світ по своєму, догадуючись або не догадуючись до деталей.

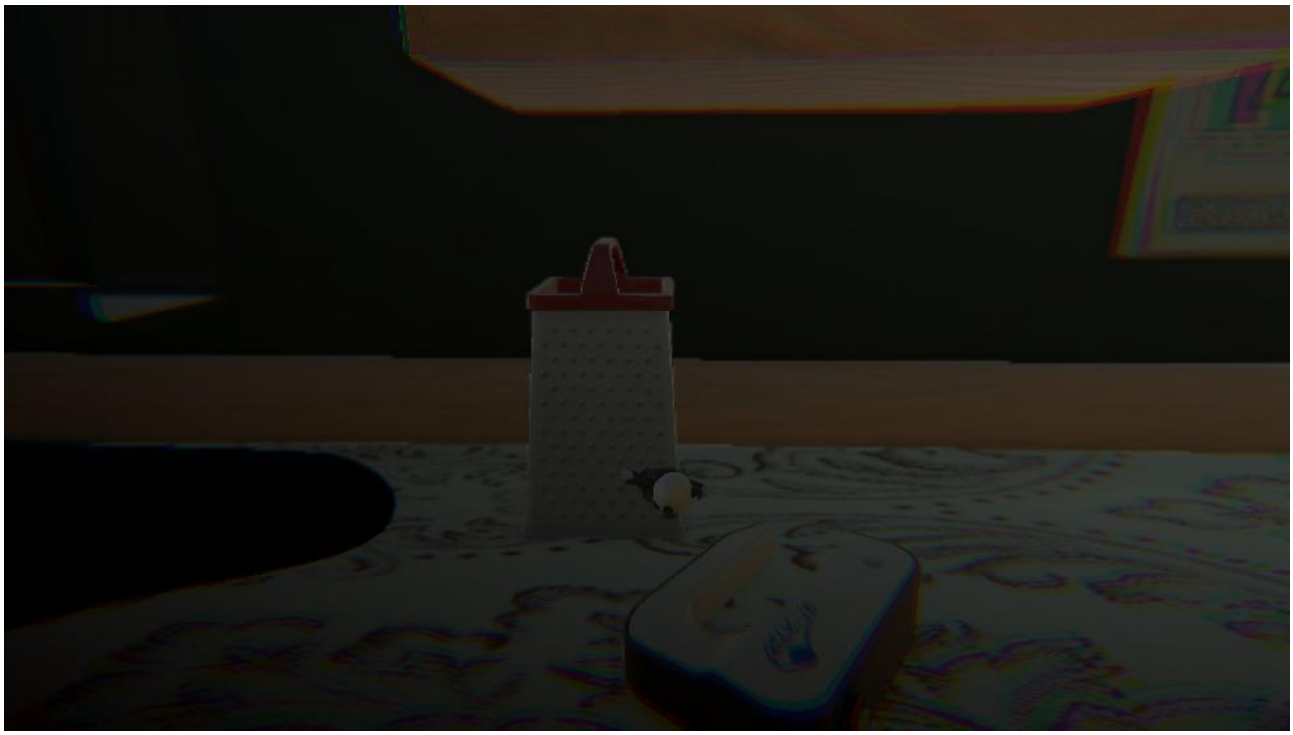


Рис. 3.25 Після чекпоінту

У фінальному відрізку, потрібно просто дойти до фінішу, з невеликими перешкодами. Гравець досягає фінішу та переходить далі

Рис. 3.26 Фінал рівня

Серед функціоналу , звісно присутнє меню паузи, в якому передбачі налаштування музики та звуку в цілому.

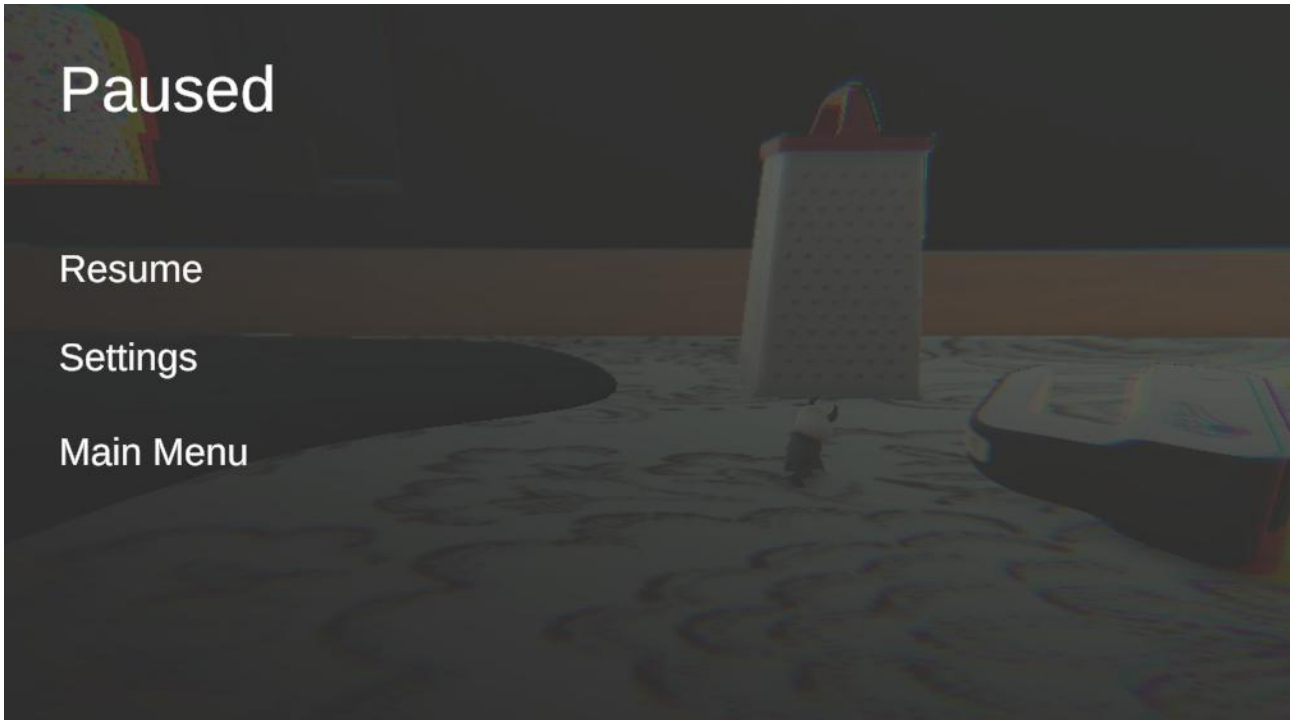


Рис. 3.27 Меню паузи

Рис. 3.28 Меню налаштувань в меню паузи

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного і програмного забезпечення

Так як застосунок розроблюється під операційну систему Windows ми будемо розглядати вимоги до персональних комп’ютерів та ноутбуків.

Вимоги до апаратного забезпечення:

- Процесор: Quad-core Intel або AMD, 2.5 GHz або швидше.
- Оперативна пам’ять: не менше 8 GB RAM.
- Графічна карта: Сумісна з DirectX 11 або 12 відеокарта.
- Місце на диску: 5 Гб.

Вимоги до програмного забезпечення:

- Операційна система: Windows 10 64-bit версія 1909 ревізія .1350або вище, або версія 2004 та 20H2 ревізія .789 або вище.

Склад інсталяційного пакету

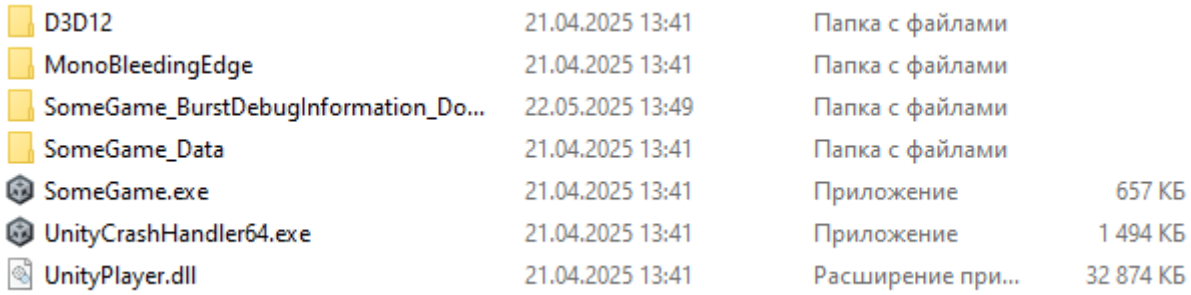


Рис. 4.1 Вміст інсталяційного пакету

Виконуваний файл гри (Game Executable) - Це головний файл, який відповідає за запуск гри.

Директорія "SomeGame_Data" - Ця папка містить необхідні файли та компоненти самого ігрового движка Unity , які потрібні для роботи гри. Вона містить бібліотеки, модулі та інші компоненти.

Конфігураційні файли (Configuration Files): Файли, які містять налаштування та параметри гри, такі як налаштування графіки, звуку, керування, мови тощо. Зазвичай це файли з розширенням .ini або .cfg.

Ресурсні файли (Resource Files) - Файли, які містять ресурси, використовувані в грі, такі як текстури, моделі персонажів і об'єктів, звукові ефекти, музика тощо.

Директорія - Папка, яка містить усі ресурси гри, такі як текстури, моделі, анімації, звукові ефекти тощо. Це одна з найважливіших папок, оскільки вона містить всі активи, які використовуються під час гри.

Директорія "Config" - Папка, яка містить конфігураційні файли гри. Ці файли містять налаштування графіки, звуку, керування та інші параметри гри.

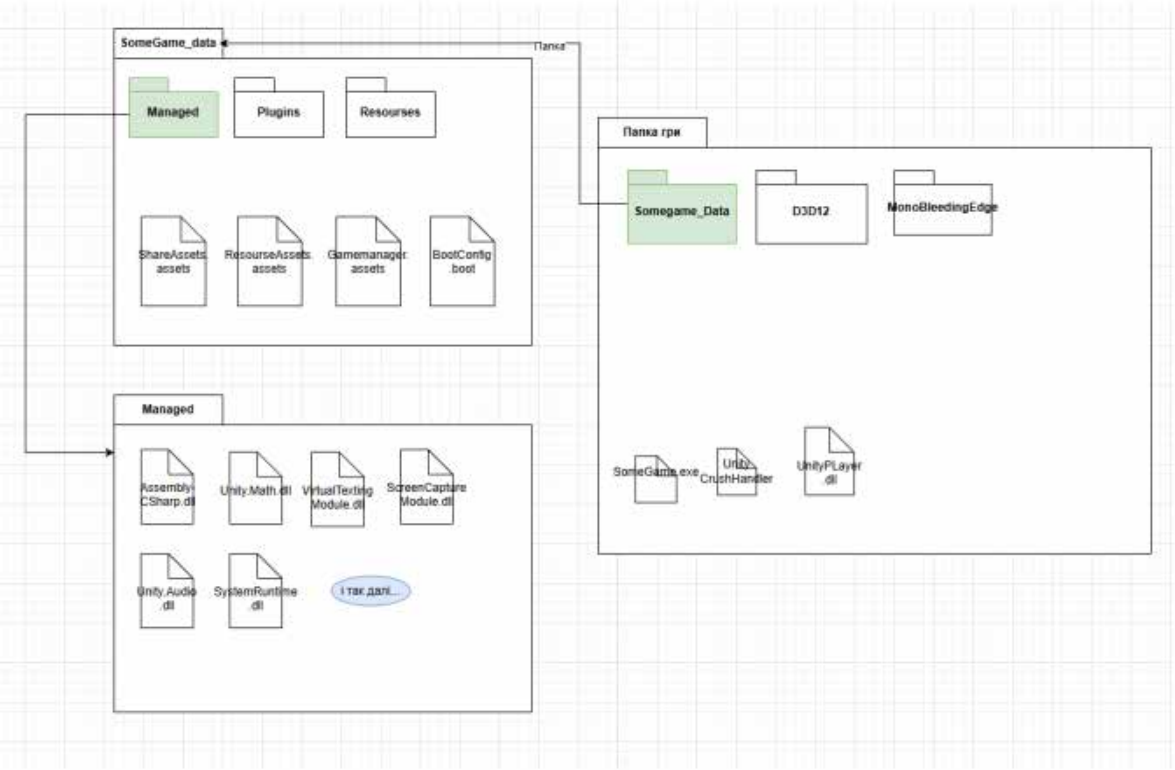


Рис. 4.2 Діаграма розміщення

4.2 Інструкції впровадження та експлуатації системи

Перш ніж поширювати проект серед користувачів, його потрібно належним чином запакувати. Пакування гарантує, що весь код і вміст є актуальним і має належний формат для запуску на потрібній цільовій платформі.

Перш ніж пакувати гру, спершу потрібно встановити сцену за замовчуванням, яка буде завантажуватися під час запуску запакованої гри. Якщо не встановити мапу і використовувати порожній проект, ми побачимо лише чорний екран під час запуску запакованої гри.

Щоб встановити сцену гри за замовчуванням, обираємо у головному меню редактора пункт File > BuildProfiles > Сцени, як показано на рисунку 4.2.

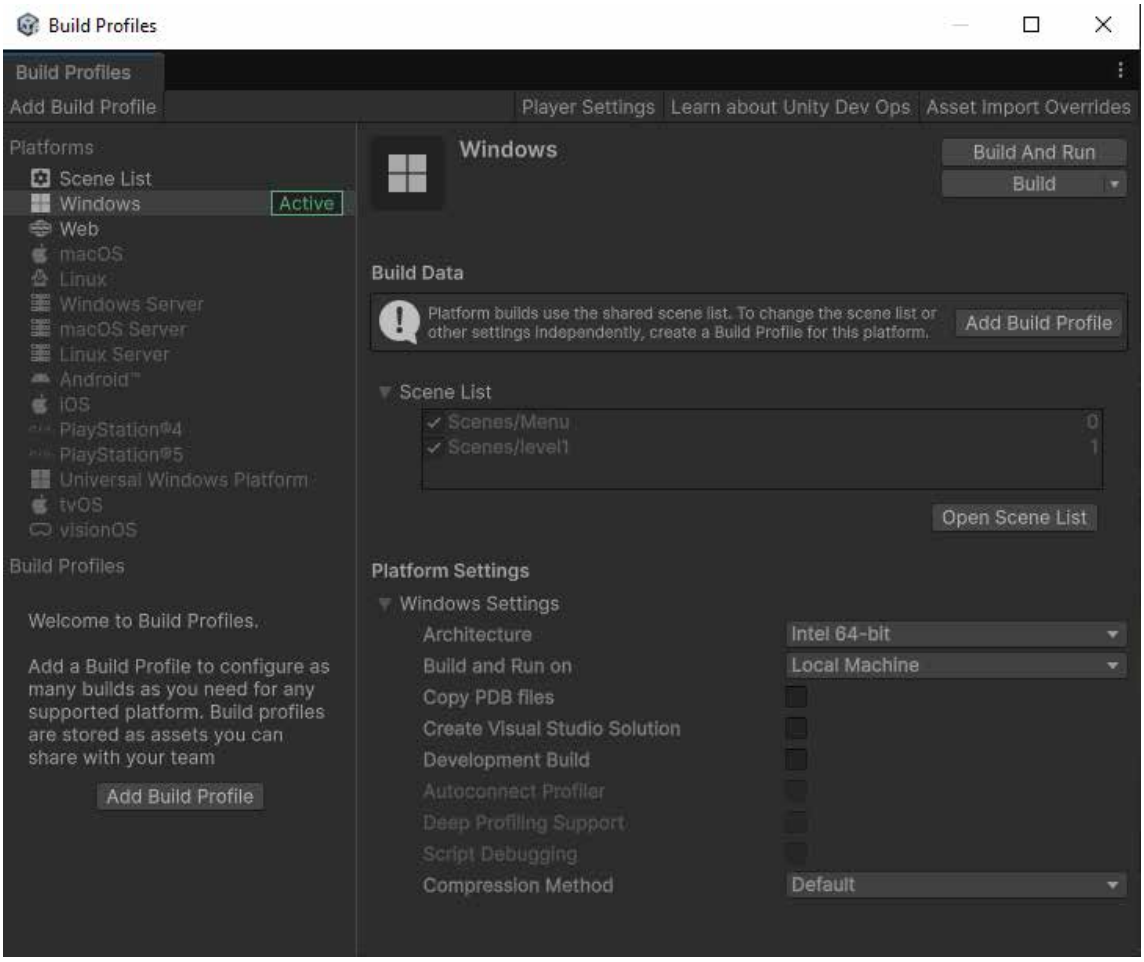


Рис. 4.3 Налаштування сцен для пакування проекту

У вікні Scene List вибираємо порядок запуску сцен, 0- сцена за замовчанням(початкова), головне меню ,1 – сцена першого рівня , сцену запускатимуться та пакуватимуться за порядком від 0 і зростаючи далі.

Після налаштування сцен , можна налаштувати пакування проекту, дати назву, вказати автора , змінити іконку гри, також вибрати директорію для збереження гри, в яку занесуться файли потрібні для функціонування гри .

4.3 Тестування роботи системи

4.3.1 Тестування комп’ютерних ігор.

Тестування комп'ютерних ігор є невід'ємною частиною процесу розробки ігрового продукту. Воно має на меті виявлення помилок, недоліків, а також перевірку функціональності та якості гри перед її випуском на ринок. Тестування включає в себе різні етапи і методи, які сприяють поліпшенню ігрового досвіду та забезпеченню високої якості гри.

Один з перших етапів тестування - це тестування функціональності. На цьому етапі перевіряється, чи працюють всі функції гри, включаючи головні механіки, системи управління, інтерфейс та інші елементи геймплею. Тестувальники виконують різні дії та досліджують можливі шляхи взаємодії з грою, щоб виявити будь-які проблеми або некоректну роботу.

Тестування також включає в себе пошук та виправлення помилок, відомих як "баги". Тестувальники активно перевіряють гру на наявність помилок, реєструють їх і подають звіти розробникам. Розробники в свою чергу виправляють ці помилки та проводять повторне тестування, щоб переконатися, що проблема була вирішена.

У процесі розробки гри, тестування є неперервним ітеративним процесом. Воно проводиться на різних етапах розробки, починаючи від ранніх прототипів до

фінальної версії. Кожне тестування допомагає виявити проблеми та забезпечити якість гри, що відповідає очікуванням гравців.

Тестування на Unity проекті відбувається вручну, в класах передбачені перевірки виведення в консоль чи відбудлася умова чи дія. Наприклад, на рисунку 3.19 перевірка запису координат чекпоінту.

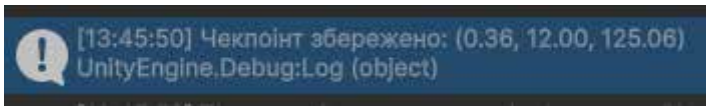


Рис. 3.21 Виведення перевірки в консоль

4.3.2 Тестування проекту за допомогою інструментів розробника.

Unity Profiler – інструмент тестування вбудований в Unity , оцінює продуктивність роботи програми .

Він дозволяє розробникам отримувати цінну інформацію про продуктивність гри та виявляти можливі проблеми, що впливають на її швидкодію та якість.

Під час тестування з використанням Unity Profiler, розробники мають можливість:

- Запускати гру в режимі профілювання, що дозволяє збирати детальні дані про використання процесора, пам'яті, графічного процесора та інших ресурсів.
- Аналізувати зібрані дані в реальному часі або після закінчення тестування. Unity Profiler надає графіки, діаграми та інші візуальні засоби відображення даних, що допомагають виявляти проблеми та шукати способи їх вирішення.
- Виявляти непотрібні ресурсоємні операції та вузькі місця в грі, що дозволяє оптимізувати її швидкодію та забезпечити плавний геймплей.

- Відстежувати використання пам'яті та ідентифікувати можливі проблеми з утриманням пам'яті, такі як витоки пам'яті або надмірне використання ресурсів.
- Тестувати гру на різних платформах та пристроях для забезпечення оптимальної продуктивності та сумісності.

Тестування з використанням Unity Profilerдопомагає покращувати якість та продуктивність гри, забезпечує ефективне використання ресурсів та створює більш задовільний результат для гравців. Використання цього інструменту дозволяє швидко виявляти та вирішувати проблеми, що можуть виникнути під час розробки та експлуатації гри[1]. Вікно Unity Profiler можна побачити на рисунку 3.23.

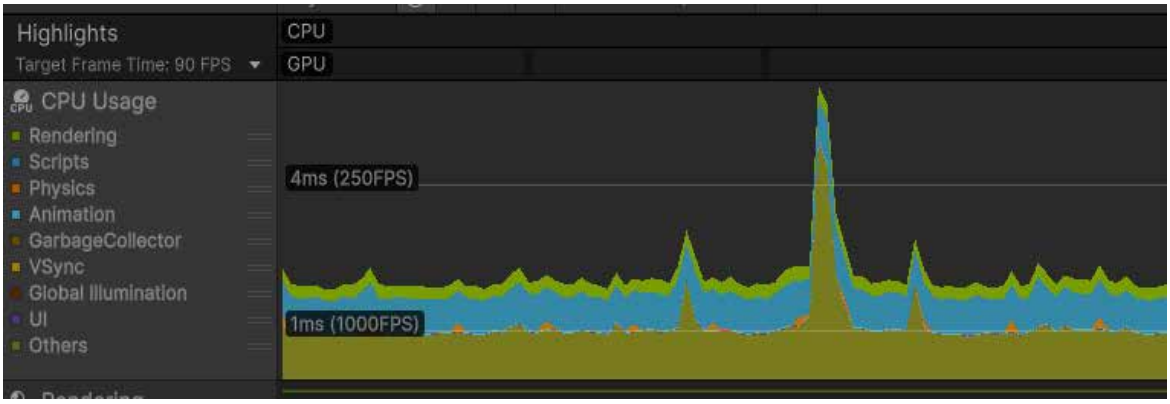


Рис. 3.22 Продутивність гри в FPS

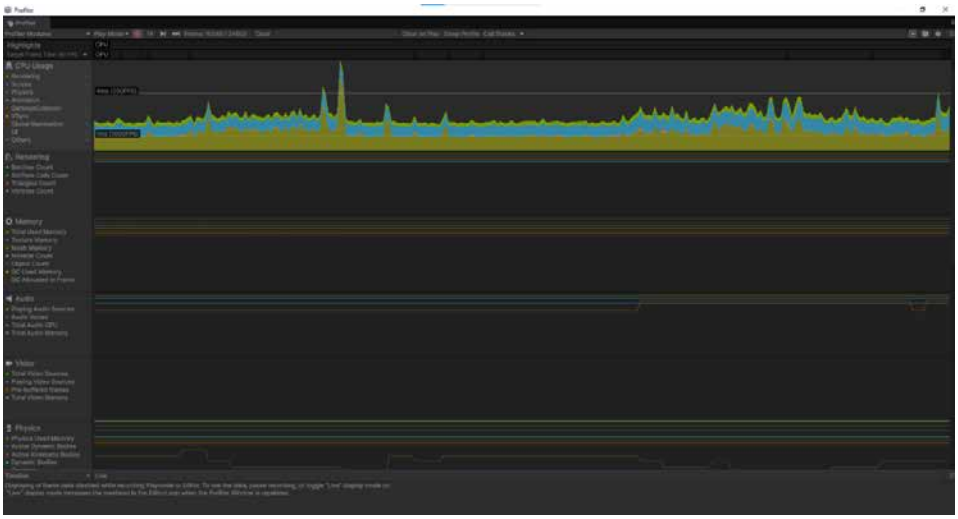


Рис.3.23 Повне вікно Unity Profiler

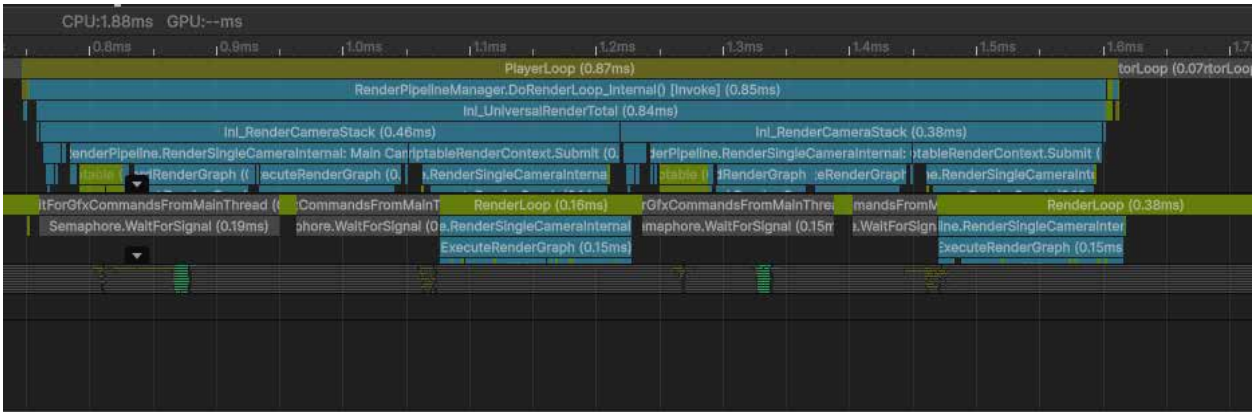


Рис 3.24 Покадрова продуктивність в Unity Profiler

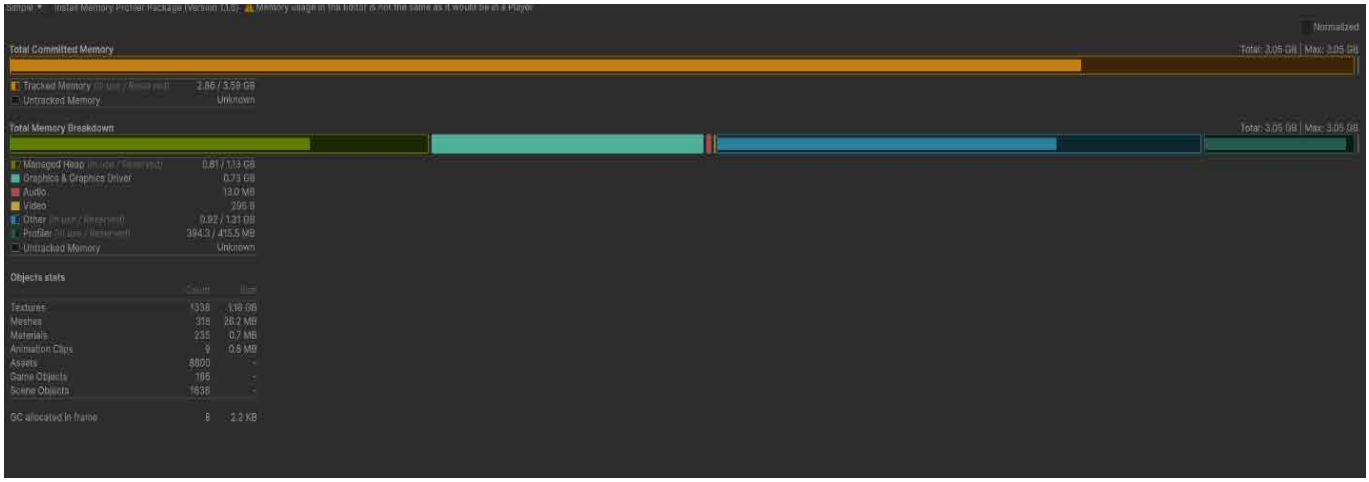


Рис 3.25 Використання пам’яті в Unity Profiler

ВИСНОВКИ

В ході роботи над бакалаврською кваліфікаційною роботою було створено прототип комп’ютерної гри у жанрі 2.5D платформера, реалізований на ігровому рушії Unity. Проект включає в себе весь процес створення ігрової системи — від формулювання завдання до тестування та реалізації.

На етапі проектування було визначено функціональні та нефункціональні потреби, розроблено архітектуру системи з використанням UML-діаграм, а також втілено систему керування сценами, фізичними взаємодіями та взаємодією з гравцем. Застосовано передові підходи до розробки інтерфейсів, анімації та управління ігровими подіями.

Систему протестовано та пристосовано для запуску на ПК, з можливістю подальшої адаптації для інших платформ. Додатково підготовлено інсталяційний пакет та додано інструкції щодо запуску гри.

Отримані результати засвідчують досягнення поставленої задачі – створення високоякісного прототипу гри, що відповідає сучасним стандартам інді-розробки. Проект має потенціал для розвитку: додавання нових рівнів, вдосконалення системи збереження, інтеграції сюжету або багатокористувацького режиму.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Entertainment Software Association. 2015 Essential Facts About the Computer and Video Game Industry [Електронний ресурс]. – Режим доступу: https://www.theesa.com/research/2015-essential-facts-about-the-computer-and-video-game-industry/
2. Unity Technologies. Unity Manual [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Manual/index.html
3. Microsoft. C# Programming Guide [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/
4. JSON.org. Introducing JSON (JavaScript Object Notation) [Електронний ресурс]. – Режим доступу: https://www.json.org/
5. Wikipedia. Game mechanics [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Game_mechanics
6. Unreal Engine. System Requirements [Електронний ресурс]. – Режим доступу: https://docs.unrealengine.com/
7. Unity Technologies. Visual Scripting in Unity [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Manual/VisualScripting.html
8. Blender Foundation. Blender Manual [Електронний ресурс]. – Режим доступу: https://docs.blender.org/manual/
9. Fullerton T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games. 4th ed. – Boca Raton: CRC Press, 2018. – 535 p.
10. Cooper A., Reimann R., Cronin D., Noessel C. About Face: The Essentials of Interaction Design. 4th ed. – Indianapolis: Wiley, 2014. – 720 p.

11. Rollings A., Adams E. Andrew Rollings and Ernest Adams on Game Design. – Indianapolis: New Riders Publishing, 2003. – 638 p.

12. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. – Boca Raton: CRC Press, 2019. – 600 p.

13. Rogers S. Level Up! The Guide to Great Video Game Design. 2nd ed. – Indianapolis: Wiley, 2014. – 552 p.

14. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. – Boston: Addison-Wesley, 2003. – 208 p.

15. Ambler S. W. The Object Primer: Agile Model-Driven Development with UML 2.0. 3rd ed. – Cambridge: Cambridge University Press, 2004. – 472 p.

16. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. – Boston: Addison-Wesley, 2005. – 496 p.

17. Batini C., Ceri S., Navathe S. B. Conceptual Database Design: An Entity-Relationship Approach. – Redwood City: Benjamin-Cummings Publishing, 1992. – 470 p.

18. Date C. J. An Introduction to Database Systems. 8th ed. – Boston: Pearson, 2003. – 1024 p.

19. Teorey T. J., Lightstone S. S., Nadeau T. Database Modeling and Design: Logical Design. 5th ed. – San Francisco: Morgan Kaufmann, 2011. – 608 p.

20. Oracle. MySQL 8.0 Reference Manual [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/

21. Unity Technologies. Unity Asset Store Documentation [Електронний ресурс]. – Режим доступу: https://assetstore.unity.com/

22. Вікіпедія. Логічна модель даних [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Логічна_модель_даних