МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка 2D-платформеру з використанням ігрового рушія Unity**

Виконав: студент 4 курсу, групи 6.121-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
В. С. Чистіков
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., доцент І. А. Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П.О Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Тетяна КРИТСЬКА_
"_01_" ___березня__ 2024_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Чистікову Вадиму Сергійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи __Розробка 2D-платформеру з використанням ігрового рушія Unity
керівник роботи ___Скрипник Ірина Анатоліївна, доцент, к.ф.-м.н._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 26.12.2023 №2215-с

2. Строк подання студентом кваліфікаційної роботи_____07.06.2023_____
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- створення програмного продукту та його опис;
- дослідження поставленої проблеми та розробка висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз предметної області	15.03.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	17.03.23	виконано
3	Аналіз існуючих методів рішення	18.03.23	виконано
4	Дослідження засобів реалізації гри	19.03-25.03.23	виконано
5	Узгодження подальших дій з науковим керівником	30.03.23	виконано
6	Проектування архітектури гри	01.04-15.04.23	виконано
7	Програмна реалізація гри	16.04-15.05.23	виконано
8	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	25.05.23	виконано
9	Реалізація користувацького інтерфейсу гри	26.05-30.05.23	виконано
10	Перевірка працездатності проекту	02.06-04.06.23	виконано
11	Оформлення звіту	05.06-12.06.23	виконано
12	Оформлення презентації. Отримання рецензій від опонентів.	13.06-15.06.23	виконано

Студент _____ Чистіков В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено
Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 67

Рисунків — 13

Джерел — 10

Чистіков В.С. Розробка 2D-платформеру з використанням ігрового рушія Unity: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І. А. Скрипник. Запоріжжя : ЗНУ, 2024. 67 с.

Метою даного проекту є створення 2D-платформеру з використанням ігрового рушія Unity. Гра повинна включати кілька рівнів із різноманітними механіками та викликами для гравця, забезпечуючи захоплюючий ігровий досвід. Особливу увагу буде приділено дизайну рівнів, плавності анімацій та інтуїтивно зрозумілому управлінню.

У процесі розробки була розглянута проблема побудови гри, використовуючи ігровий рушій Unity. У результаті була розроблена гра «Epic Adventure», створена за допомогою ігрового рушія Unity. Дана гра містить в собі 3 рівня. Кожні рівні відрізняються локацією та противниками.

Ключові слова*: інді гра, pixel art, Unity, C#.*

ABSTRACT

Pages — 67

Drawings — 13

Sources — 10

Chystikov V.S. Development of 2D Platformer Using Game-Engine Unity: bachelor's thesis in specialty 121 "Software Engineering" / Science. manager I.A. Skrypnyk. Zaporizhzhia: ZNU, 2023. 67 p.

The objective of this project is to create a 2D platformer using the Unity game engine. The game should include several levels with various mechanics and challenges for the player, providing an engaging gameplay experience. Special attention will be given to level design, smooth animations, and intuitive controls.

During the development process, the problem of building a game using the Unity game engine was addressed. As a result, a game called " Epic Adventure" was developed using the Unity engine. This game contains 3 levels, each distinguished by unique locations and enemies.

Keywords: indie game, pixel art, Unity, C#.

ЗМІСТ

ВСТУП

Актуальність теми

У сучасному світі індустрія відеоігор є однією з найшвидше зростаючих і найприбутковіших сфер розваг. Зокрема, популярність 2D-платформерів, яка зародилася ще в 80-х роках минулого століття, залишається високою і сьогодні. Цей жанр відеоігор продовжує привертати увагу як нових гравців, так і гравців із досвідом, завдяки своїй простоті, доступності та можливостям для творчості. Розробка 2D-платформеру з використанням ігрового рушія Unity є актуальною темою з кількох причин.

По-перше, ігровий рушій Unity є одним з найпопулярніших інструментів для розробки відеоігор. Він пропонує широкий спектр можливостей для створення як 2D, так і 3D ігор. Unity має зручний інтерфейс, багатий набір інструментів для роботи з графікою, фізикою та анімацією, що робить його доступним як для професіоналів, так і для початківців у сфері розробки ігор. Використання Unity дозволяє швидко створювати прототипи, тестувати їх та вдосконалювати, що є важливим аспектом у процесі розробки відеоігор.

По-друге, 2D-платформери мають широкий спектр застосувань та жанрових варіацій. Вони можуть бути як простими аркадами для мобільних пристроїв, так і складними ігровими проєктами з глибоким сюжетом та розвиненими ігровими механіками. Така гнучкість дозволяє розробникам втілювати найрізноманітніші ідеї та підходи, експериментувати з дизайном рівнів, стилями анімації та музичним супроводом.

По-третє, розробка 2D-платформеру з використанням Unity є відмінним способом для новачків у сфері ігрової розробки набути необхідних навичок та компетенцій. Unity підтримує мову програмування C#, яка є однією з найбільш популярних і затребуваних мов у сфері програмування. Вивчення C# у контексті розробки відеоігор дозволяє студентам та початківцям освоїти основи програмування, логіки та алгоритмізації. Крім того, розробка власної

гри сприяє розвитку творчих навичок, критичного мислення та вміння працювати в команді.

Четвертим важливим аспектом є доступність платформи Unity для розробників з усього світу. Unity пропонує безкоштовну версію свого рушія для студентів, інді-розробників та малих студій, що дозволяє почати створення власних ігрових проектів без значних фінансових вкладень. Крім того, Unity має велике ком'юніті розробників, де можна знайти багато корисних ресурсів, таких як відеоуроки, форуми та документація. Це значно полегшує процес навчання та вирішення проблем, що виникають під час розробки.

П'ятою причиною актуальності теми є розвиток інді-ігор. Сьогодні інді-ігри займають важливе місце в ігровій індустрії, пропонуючи гравцям унікальні та нестандартні ігрові досвіди. 2D-платформери є одним з найпопулярніших жанрів серед інді-розробників, оскільки вони не вимагають великих ресурсів для створення, але при цьому дозволяють втілити цікаві ідеї та інноваційні ігрові механіки. Unity, зі своєю потужною екосистемою та підтримкою інді-розробників, є ідеальною платформою для створення таких ігор.

Підсумовуючи, можна сказати, що розробка 2D-платформеру з використанням ігрового рушія Unity є надзвичайно актуальною темою. Вона об'єднує технічні та творчі аспекти, дозволяючи розробникам створювати захоплюючі ігрові проекти. Unity надає всі необхідні інструменти для цього, роблячи процес розробки доступним та ефективним. Завдяки своїй гнучкості та широким можливостям, Unity дозволяє створювати ігри, які можуть конкурувати на ринку та приносити задоволення гравцям по всьому світу.

Мета дослідження

Метою даного дослідження є розробка та аналіз процесу створення 2D-платформеру із застосуванням ігрового рушія Unity. Дослідження спрямоване на вивчення методів та інструментів, які забезпечують ефективний процес

розробки ігрових проектів, а також на виявлення та вирішення проблем, що виникають під час розробки. Особлива увага буде приділена таким аспектам, як дизайн рівнів, оптимізація продуктивності, розробка ігрової механіки, анімація та інтеграція звукових ефектів. Крім того, дослідження матиме на меті визначення найкращих практик для створення інтуїтивно зрозумілого управління та забезпечення захоплюючого ігрового досвіду. Результати дослідження можуть бути корисними як для початківців, так і для досвідчених розробників ігор, що працюють з Unity.

Завдання дослідження

1. Аналіз літератури та існуючих рішень:
   - Дослідити наукові та технічні джерела, що описують процес розробки 2D-платформерів.
   - Вивчити існуючі ігрові проекти, створені за допомогою Unity, з метою визначення їх переваг та недоліків.
2. Вибір та обґрунтування інструментів:
   - Визначити необхідні інструменти та технології для розробки 2D-платформеру.
   - Обґрунтувати вибір ігрового рушія Unity для реалізації проекту.
3. Розробка концепції гри:
   - Створити детальний концепт гри, включаючи сюжет, основні ігрові механіки та візуальний стиль.
   - Розробити дизайн гри, який міститиме всі необхідні специфікації.
4. Проектування та розробка ігрових рівнів:
   - Створити прототипи кількох ігрових рівнів.

- Забезпечити різноманітність локацій та ворогів на кожному рівні.

5. Розробка ігрової механіки та управління:
   - Реалізувати основні ігрові механіки, такі як рух персонажа, стрибки, атаки тощо.
   - Розробити інтуїтивно зрозуміле управління для гравців.
6. Анімація та візуальні ефекти:
   - Створити та інтегрувати плавні анімації персонажів та об'єктів.
   - Розробити та застосувати візуальні ефекти для покращення ігрового досвіду.
7. Інтеграція звукових ефектів та музики:
   - Підібрати та додати звукові ефекти та музичні композиції, що відповідають стилю гри.
   - Забезпечити синхронізацію звукових ефектів з діями гравця.
8. Оптимізація та тестування гри:
   - Провести оптимізацію гри для забезпечення стабільної продуктивності на різних пристроях.
   - Здійснити тестування гри на різних етапах розробки для виявлення та виправлення помилок.

Об’єкт дослідження

Об'єктом дослідження є процес розробки гри на основі ігрового рушія Unity.

Предмет дослідження

Предметом дослідження є вивчення основних принципів роботи з Unity, аналіз сучасних тенденцій у розробці інді-ігор, для врахування актуальних вимог до розроблюваної гри.

Методи дослідження

Проведення детального огляду літературних джерел, наукових публікацій, статей, підручників та інших джерел, що стосуються технологій розробки ігор з використання ігрового рушія Unity; вивчення та аналіз популярних ігор, які написані за допомогою Unity, з метою виявлення їх особливостей, переваг та недоліків, а також накопичення досвіду розробки ігор; апробація результатів.

Практичне значення одержаних результатів

Практичне значення одержаних результатів полягає в тому, що вони можуть бути використані для вдосконалення процесу розробки 2D-платформерів із застосуванням ігрового рушія Unity. Зокрема, результати дослідження можуть мати наступні прикладні аспекти:

1. Покращення методик розробки:
    - Одержані результати допоможуть розробникам оптимізувати процес створення ігрових рівнів, інтеграції анімацій та звукових ефектів, а також налаштування фізики та колізій. Це дозволить зменшити час на розробку та покращити якість кінцевого продукту.
2. Використання у навчанні:
    - Результати дослідження можуть бути використані як навчальні матеріали для курсів з розробки відеоігор.

Розробники-початківці зможуть отримати цінні знання та навички, необхідні для створення власних ігрових проектів.

3. Оптимізація продуктивності:
   - Знання про оптимізацію ігрових проектів на основі Unity дозволить розробникам створювати ігри, які працюють стабільно та без лагів на різних платформах, включаючи мобільні пристрої. Це особливо важливо для досягнення широкого охоплення аудиторії.
4. Покращення геймплейного досвіду:
   - Вивчення та впровадження найкращих практик у сфері дизайну рівнів, інтерактивності та управління дозволить створювати ігри, які забезпечують захоплюючий ігровий досвід. Це сприятиме підвищенню задоволеності гравців та їхньої лояльності до розробника.
5. Зменшення витрат:
   - Одержані результати можуть допомогти зменшити витрати на розробку завдяки оптимізації процесів та впровадженню ефективних методик. Це дозволить малим студіям та інді-розробникам використовувати ресурси більш ефективно.

Підводячи підсумок, практичне значення одержаних результатів полягає в тому, що вони сприятимуть розвитку якості та ефективності розробки 2D-платформерів, підвищуючи таким чином конкурентоспроможність розробників на ринку відеоігор. Вони стануть корисними як для початківців, так і для досвідчених розробників, надаючи їм необхідні інструменти та знання для створення захоплюючих та якісних ігрових проектів.

Глосарій

2D-платформер: Це жанр відеоігор, де гравець керує персонажем, який рухається по 2D-площині, пересуваючись вліво або вправо, стрибаючи та взаємодіючи з об'єктами на екрані.

Unity: Це кросплатформений ігровий рушій та інтегроване середовище розробки (IDE), розроблене компанією Unity Technologies. Використовується для розробки ігор для різних платформ, включаючи комп'ютери, консолі, мобільні пристрої та веб.

Рівень (Level): Це частина гри, яка представляє собою певний ігровий етап або локацію. У платформерах кожен рівень може містити певні завдання, ворогів та перешкоди.

Баг (Bug): Помилка або недолік у програмному коді гри, який призводить до некоректної роботи гри. Баги можуть бути різними за ступенем серйозності – від незначних візуальних дефектів до серйозних проблем, які можуть зіпсувати ігровий процес або зробити гру непрохідною.

Урон (Damage): Показник шкоди, яку отримує персонаж або об'єкт у грі від атак, падінь, пасток або інших небезпечних елементів. Урон може бути фізичним, магічним, стихійним тощо, в залежності від типу гри та її механік.

1 АНАЛІЗ ПРОБЛЕМИ

1.1 Огляд літературних джерел

Ігрова індустрія продовжує стрімко розвиватися, і однією з ключових складових цього процесу є використання ігрових рушіїв, таких як Unity. Unity надає потужну платформу для створення різноманітних ігрових проєктів, включаючи 2D-платформери. Розробка таких ігор на Unity супроводжується різними проблемами та викликами, які можуть бути успішно вирішені завдяки наявним ресурсам.

Проблеми та Виклики

Реалізація фізики руху персонажа: Однією з основних проблем є забезпечення реалістичної та плавної фізики руху персонажа. Це включає управління швидкістю, прискоренням, стрибками та іншими аспектами руху.

- Обробка зіткнень: Важливо правильно обробляти зіткнення персонажа з іншими об'єктами на рівні, що включає врахування фізики та забезпечення коректної реакції на зіткнення.
- Створення анімацій: Для досягнення високої якості гри необхідно забезпечити плавні та реалістичні анімації персонажів, враховуючи всі можливі стани та переходи між ними.
- Реалізація системи зброї: Багато 2D-платформерів включають елементи бойових дій, що вимагає створення функціональної та гнучкої системи зброї.
- Збір предметів та створення меню: Необхідно забезпечити можливість збирання предметів, які впливають на ігровий процес, а також створити зручне та інтуїтивно зрозуміле меню.

Шляхи Рішення

Використання навчальних матеріалів: Існують книги, такі як "Unity in Action" [6] та "Unity Game Development Cookbook" [7], що пропонують детальні інструкції та приклади коду для вирішення вищезгаданих проблем.

Онлайн-ресурси: "Unity Learn" [8] та інші онлайн-ресурси пропонують курси, проєкти та інструкції, які допомагають розробникам розвивати свої навички у створенні 2D-ігор на Unity.

Практичні приклади та інструкції: Використання детальних прикладів коду та покрокових інструкцій дозволяє розробникам зрозуміти складні аспекти розробки та впровадити їх у свої проєкти.

Загалом, наявні ресурси значно полегшують процес розробки 2D-платформерів на Unity. Книги та онлайн-курси надають детальні інструкції та практичні поради, які допомагають розробникам успішно вирішувати проблеми, що виникають під час створення ігор. Завдяки цим ресурсам, розробники можуть ефективно реалізувати фізику руху персонажа, обробку зіткнень, анімації, системи зброї, збір предметів та створення меню, що дозволяє створювати захопливі та якісні 2D-платформери.

1.2 Аналіз існуючих аналогів

1.2.1 Rayman Legends

Rayman Legends – це відома ігра в жанрі платформера [5], розроблена компанією Ubisoft Montpellier. Гра відзначається своїм яскравим арт-стилем, захоплюючим геймплеєм та великим різноманіттям рівнів із цікавими механіками та викликами.

Одним з ключових особливостей Rayman Legends є його чудернацький арт-стиль та деталізована анімація персонажів та оточуючого світу. Гра наповнена яскравими кольорами, веселими персонажами та дивовижними локаціями, що створює магічну атмосферу для гравців будь-якого віку.

У Rayman Legends гравці керують головним героєм Рейманом та його друзями, які вирушають у захоплюючий пригодницький шлях, щоб врятувати

світ від злого злочинця. Гра пропонує багато різноманітних рівнів з різними механіками та викликами, включаючи бойові зони, головоломки та швидкі або захоплюючі перегони.

Особливим елементом гри є її музичні рівні, які відрізняються від інших ігор у жанрі. Гравці мають можливість взаємодіяти з музичними елементами рівня, щоб створювати ритмічні мелодії та відчувати себе частиною музичного світу гри.

Rayman Legends також славиться своєю мультиплатформенністю, що дозволяє гравцям насолоджуватися грою на різних пристроях, включаючи консолі, персональні комп'ютери та портативні пристрої. Це робить гру доступною для широкого кола гравців та дозволяє їм насолоджуватися світом Реймана незалежно від їхнього пристрою вибору.

Узагальнюючи, Rayman Legends - це захоплюючий та яскравий платформер, який вразить вас своїм чудовим арт-стилем, захоплюючим геймплеєм та різноманітністю викликів. Вона стала однією з найуспішніших ігор у серії Rayman та отримала велику кількість похвальних відгуків від гравців та критиків.

Рисунок 1 – *Гра Rayman Legends*

1.2.2 Super Mario

Super Mario – це культова серія ігор в жанрі платформера [4], розроблена японською компанією Nintendo. Гра відзначається своїм неперевершеним геймплеєм, веселими персонажами та іконічними рівнями, що стали невід'ємною частиною історії відеоігор.

Одним з головних елементів гри є її яскравий та кольоровий арт-стиль, що вражає своєю простотою та водночас деталізацією. Гравці керують головним героєм, Маріо, який вирушає на рятування принцеси Піч, зустрічаючи пригоди на своєму шляху.

У грі Super Mario гравці мають можливість пройти через різноманітні рівні, що варіюються від легких до дуже складних, зустрічаючи ворогів, збираючи монети та виконуючи різноманітні завдання. Головними елементами геймплею є стрибки, збір бонусів та взаємодія з оточуючим світом.

Super Mario також відомий своїми культовими рівнями, такими як "1-1" з оригінальної гри, що стали символами в історії відеоігор. Ці рівні відрізняються своєю креативністю та веселощами, що робить гру захоплюючою для гравців будь-якого віку

Однією з особливостей Super Mario є його мультиплатформенність, що дозволяє гравцям насолоджуватися грою на різних пристроях Nintendo, включаючи консолі та портативні пристрої. Це робить гру доступною для широкого кола гравців та дозволяє їм насолоджуватися пригодами Маріо незалежно від їхнього пристрою вибору.

Узагальнюючи, Super Mario – це класичний платформер, який вразить вас своїм яскравим арт-стилем, захоплюючим геймплеєм та незабутніми персонажами. Він став невід'ємною частиною культури відеоігор та продовжує радувати гравців по всьому світу своїми захоплюючими

пригодами.

Рисунок 2 – *Гра Super Mario*

1.3 Постановка завдання

Для успішної розробки 2D-платформера з використанням ігрового рушія Unity, необхідно визначити та сформулювати завдання проекту. Основні пункти постановки завдання включають такі аспекти:

1. Опис гри:
   - Описати загальну концепцію гри, її механіку та цільову аудиторію.
2. Функціональні вимоги:
   - Визначити основні функції та можливості гравця.
   - Описати рівні гри, їх структуру, локації та особливості.
   - Встановити правила гри, включаючи умови перемоги та поразки.
3. Технічні вимоги:

- Визначити технічні характеристики гри, такі як роздільна здатність, кадрова частота тощо.
- Сформулювати вимоги до платформ, на яких буде розгортатися гра.
- Встановити необхідний функціонал для роботи з анімаціями, фізикою, звуком та іншими аспектами гри.

4. Строки:
   - Визначити очікувані терміни реалізації проекту.
5. Результати проекту:
   - Описати очікувані результати вигляду та функціоналу готової гри.
   - Визначити критерії успішності та вимоги до якості продукту.

Постановка завдання є ключовим етапом перед початком розробки, оскільки вона визначає основні вимоги та напрямки роботи над проектом, що дозволяє уникнути непорозумінь та забезпечити успішну реалізацію задуманої ідеї.

1.4 Висновки до розділу 1

1. Було оглянуто головні проблеми створення ігрових рушіїв: складність розробки, вимоги до продуктивності, недостатня підтримка, нестабільність ринку.
2. Проаналізовані найрозповсюджені ігри у жанрі 2Б-платформер: Rayman Legends, Super Mario.
3. Визначена постановка завдання дослідження.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ТЕСТУВАННЯ

2.1 Огляд існуючих ігрових рушіїв

2.1.1 Unity

Unity – це інтегроване середовище розробки (IDE) [2], яке спеціалізується на створенні ігор, а також додатків і віртуальної реальності. Воно підтримує різні платформи, включаючи iOS, Android, Windows, macOS, Linux, PlayStation, Xbox, Nintendo Switch та багато інших. Unity надає розробникам зручний інтерфейс для створення графічних об'єктів, додавання фізики, програмування логіки гри та реалізації інших функціональних можливостей.

Архітектура Unity складається з кількох основних компонентів:

1. Unity Editor (Редактор Unity): Це основне середовище розробки, де розробники створюють і редагують свої проекти. Редактор Unity надає візуальний інтерфейс для розміщення об'єктів у світі гри, програмування їх поведінки, налаштування параметрів та багато іншого.

2. Game Engine (Ігровий двигун): Unity має потужний ігровий двигун, який забезпечує обробку графіки, фізики, аудіо та інших аспектів гри. Він оптимізований для роботи на різних платформах і забезпечує високу продуктивність та відмінну якість графіки.

3. Scripting Environment (Середовище програмування): Unity підтримує різні мови програмування, такі як C#, JavaScript (UnityScript) та Boo. Розробники можуть використовувати ці мови для написання скриптів, які керують поведінкою об'єктів у грі.

4. Asset Store (Магазин ресурсів): Unity має великий і різноманітний магазин ресурсів, де розробники можуть придбати або знайти безкоштовні готові ресурси для своїх проектів, такі як моделі, текстури, анімації та інше.

5. Unity Cloud Services (Хмарні сервіси Unity): Unity пропонує різноманітні хмарні сервіси, такі як аналітика, зберігання, монетизація та інші, які допомагають розробникам у керуванні, аналізі та монетизації їх проектів.

6. Unity Player (Програвач Unity): Це програмне забезпечення, яке дозволяє запускати ігри, створені в Unity, на різних платформах. Unity Player може бути використаний для тестування гри під час розробки або для запуску гри на різних пристроях.

Створення гри в Unity зазвичай включає наступні кроки:

1. Створення проекту: Розробники створюють новий проект в Unity Editor та налаштовують його параметри, такі як розміри екрану, налаштування фізики тощо.

2. Створення об'єктів та сцен: Розробники розміщують різні об'єкти (такі як персонажі, об'єкти оточення, ефекти тощо) на сцені гри за допомогою Unity Editor.

3. Програмування логіки гри: За допомогою мов програмування, таких як C# або JavaScript, розробники програмують логіку поведінки об'єктів у грі, включаючи реакцію на взаємодію з гравцем, обробку подій тощо.

4. Тестування і оптимізація: Після завершення розробки гри розробники проводять тестування для виявлення помилок та оптимізують гру для досягнення максимальної продуктивності на різних платформах.

5. Публікація: Після успішного тестування і оптимізації гри, розробники можуть опублікувати свій проект на різних платформах. Unity надає засоби для експорту гри на платформи, такі як iOS, Android, PC, консолі та інші.

Публікація гри може включати в себе кроки, такі як:

1. Підготовка проекту до експорту: Розробники перевіряють, що всі ресурси та налаштування гри готові для експорту. Вони також можуть налаштовувати параметри експорту для конкретної платформи.

2. Експорт проекту: Unity надає можливість експортувати гру на різні платформи за допомогою вбудованих інструментів. Розробники можуть

вибрати цільову платформу та налаштувати параметри експорту, такі як налаштування графіки, аудіо, управління тощо.

3. Тестування на цільовій платформі: Перед публікацією гри розробники зазвичай проводять тестування на цільовій платформі, щоб переконатися, що гра працює коректно та оптимально на цій платформі.

4. Публікація в магазинах або на інших платформах: Після успішного тестування розробники можуть опублікувати свою гру на цільовій платформі, шляхом розміщення її в магазинах додатків або інших платформах розповсюдження.

5. Підтримка та оновлення: Після публікації гри розробники можуть продовжувати підтримку свого проекту, виправляти помилки, випускати новий контент та оновлення для своєї гри, щоб забезпечити задоволення користувачів та зберегти їхній інтерес до гри.

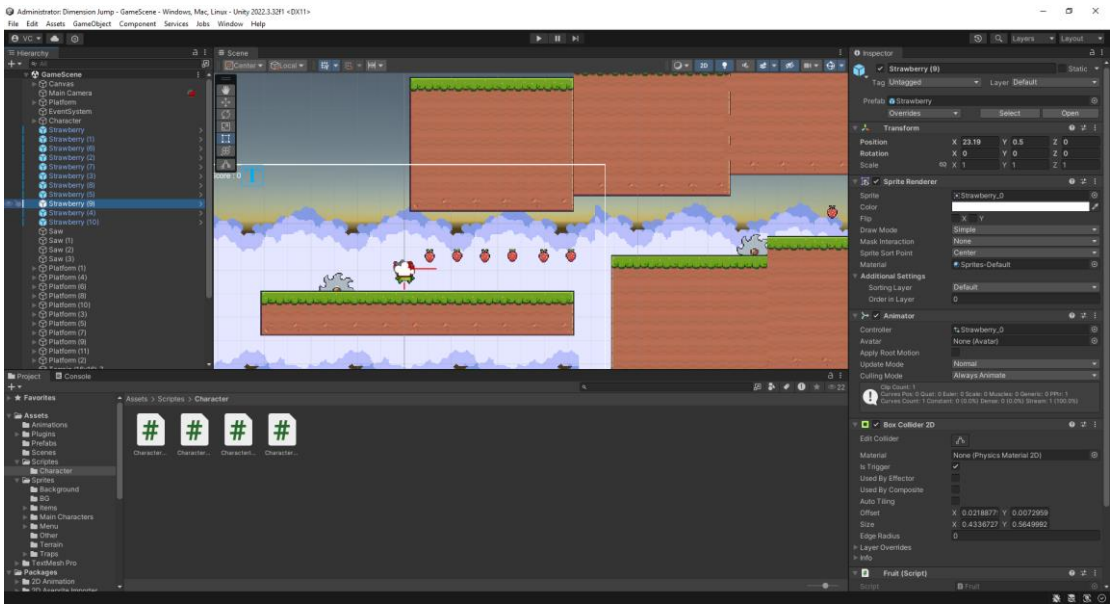

Рисунок 3 –*1 Unity*

2.1.2 Unreal Engine

Unreal Engine – це потужне інтегроване середовище розробки (IDE) [1], яке спеціалізується на створенні високоякісних ігор, віртуальної реальності (VR), а також інтерактивних додатків. Unreal Engine підтримує різні платформи, включаючи Windows, macOS, iOS, Android, Linux, PlayStation,

Xbox, Nintendo Switch та багато інших. Unreal Engine надає розробникам передовий інструментарій для створення графічних об'єктів, додавання фізики, програмування логіки гри та реалізації інших функціональних можливостей.

Архітектура Unreal Engine складається з кількох основних компонентів:

1. Unreal Editor (Редактор Unreal): Це основне середовище розробки, де розробники створюють і редагують свої проекти. Редактор Unreal надає візуальний інтерфейс для розміщення об'єктів у світі гри, програмування їх поведінки, налаштування параметрів та багато іншого.

2. Game Engine (Ігровий двигун): Unreal Engine має потужний ігровий двигун, який забезпечує обробку графіки, фізики, аудіо та інших аспектів гри. Він оптимізований для роботи на різних платформах і забезпечує високу продуктивність та відмінну якість графіки.

3. Blueprints (Візуальне програмування): Однією з відмінних рис Unreal Engine є підтримка візуального скриптування за допомогою Blueprints. Це дозволяє розробникам створювати логіку гри без необхідності писати код, використовуючи графічний інтерфейс для з'єднання вузлів логіки.

4. C++ Scripting (Програмування на C++): Для більш складних і продуктивних рішень Unreal Engine підтримує програмування на мові C++. Розробники можуть використовувати C++ для написання скриптів, які керують поведінкою об'єктів у грі, оптимізуючи продуктивність та додаючи унікальні можливості.

5. Unreal Marketplace (Маркетплейс Unreal): Unreal Engine має великий і різноманітний маркетплейс, де розробники можуть придбати або знайти безкоштовні готові ресурси для своїх проектів, такі як моделі, текстури, анімації та інше.

6. Unreal Cloud Services (Хмарні сервіси Unreal): Unreal Engine пропонує різноманітні хмарні сервіси, такі як аналітика, зберігання, монетизація та інші, які допомагають розробникам у керуванні, аналізі та монетизації їх проектів.

7. Unreal Engine Runtime (Рантайм Unreal Engine): Це програмне забезпечення, яке дозволяє запускати ігри, створені в Unreal Engine, на різних платформах. Unreal Engine Runtime може бути використаний для тестування гри під час розробки або для запуску гри на різних пристроях.

Створення гри в Unreal Engine зазвичай включає наступні кроки:

1. Створення проекту: Розробники створюють новий проект в Unreal Editor та налаштовують його параметри, такі як розміри екрану, налаштування фізики тощо.

2. Створення об'єктів та сцен: Розробники розміщують різні об'єкти (такі як персонажі, об'єкти оточення, ефекти тощо) на сцені гри за допомогою Unreal Editor.

3. Програмування логіки гри: За допомогою C++ або Blueprints, розробники програмують логіку поведінки об'єктів у грі, включаючи реакцію на взаємодію з гравцем, обробку подій тощо.

4. Тестування і оптимізація: Після завершення розробки гри розробники проводять тестування для виявлення помилок та оптимізують гру для досягнення максимальної продуктивності на різних платформах.

5. Публікація: Після успішного тестування і оптимізації гри, розробники можуть опублікувати свій проект на різних платформах. Unreal Engine надає засоби для експорту гри на платформи, такі як iOS, Android, PC, консолі та інші.Публікація гри може включати в себе кроки, такі як:

1. Підготовка проекту до експорту: Розробники перевіряють, що всі ресурси та налаштування гри готові для експорту. Вони також можуть налаштовувати параметри експорту для конкретної платформи.

2. Експорт проекту: Unreal Engine надає можливість експортувати гру на різні платформи за допомогою вбудованих інструментів. Розробники можуть вибрати цільову платформу та налаштувати параметри експорту, такі як налаштування графіки, аудіо, управління тощо.

3. Тестування на цільовій платформі: Перед публікацією гри розробники зазвичай проводять тестування на цільовій платформі, щоб переконатися, що гра працює коректно та оптимально на цій платформі.

4. Публікація в магазинах або на інших платформах: Після успішного тестування розробники можуть опублікувати свою гру на цільовій платформі, шляхом розміщення її в магазинах додатків або інших платформах розповсюдження.

5. Підтримка та оновлення: Після публікації гри розробники можуть продовжувати підтримку свого проекту, виправляти помилки, випускати новий контент та оновлення для своєї гри, щоб забезпечити задоволення користувачів та зберегти їхній інтерес до гри.

Рисунок 42 – *Unreal Engine*

2.1.3 Godot

Godot Engine – це відкрите інтегроване середовище розробки (IDE)[3] і потужний багатоплатформовий ігровий двигун, призначений для створення 2D і 3D ігор, а також інтерактивних додатків. Godot Engine підтримує різні платформи, включаючи Windows, macOS, Linux, iOS, Android, HTML5, а також консолі. Godot Engine надає розробникам широкий набір інструментів

для створення графічних об'єктів, додавання фізики, програмування логіки гри та реалізації інших функціональних можливостей.

Архітектура Godot Engine складається з кількох основних компонентів:

1. Godot Editor (Редактор Godot): Це основне середовище розробки, де розробники створюють і редагують свої проекти. Редактор Godot надає візуальний інтерфейс для розміщення об'єктів у світі гри, програмування їх поведінки, налаштування параметрів та багато іншого.

2. Scene System (Система сцен): Однією з ключових особливостей Godot Engine є її система сцен, де кожна сцена складається з дерева вузлів. Кожен вузол має свою власну функціональність, таку як спрайт, фізика, камера тощо. Це дозволяє розробникам легко компонувати складні об'єкти з простіших.

3. Scripting (Програмування): Godot підтримує декілька мов програмування, включаючи GDScript (вбудована мова, схожа на Python), VisualScript (візуальне програмування), C# і C++. Розробники можуть використовувати ці мови для написання скриптів, які керують поведінкою об'єктів у грі.

4. Animation System (Система анімації): Godot має потужну систему анімації, яка дозволяє створювати і редагувати анімації для будь-яких властивостей вузлів, таких як положення, обертання, масштабування та багато іншого. Це дозволяє створювати складні анімації без необхідності використовувати зовнішні інструменти.

5. Asset Library (Бібліотека ресурсів): Godot Engine має вбудовану бібліотеку ресурсів, де розробники можуть знайти готові ресурси для своїх проектів, такі як моделі, текстури, звукові ефекти та інше. Бібліотека також підтримує плагіни, які розширюють функціональність Godot.

6. Godot Cloud Services (Хмарні сервіси Godot): Godot пропонує різноманітні хмарні сервіси, такі як аналітика, зберігання, монетизація та інші, які допомагають розробникам у керуванні, аналізі та монетизації їх проектів.

7. Godot Export System (Система експорту Godot): Godot Engine має вбудовану систему експорту, яка дозволяє розробникам легко експортувати

свої ігри на різні платформи, такі як Windows, macOS, Linux, iOS, Android, HTML5, а також консолі.

Створення гри в Godot Engine зазвичай включає наступні кроки:

1. Створення проекту: Розробники створюють новий проект в Godot Editor та налаштовують його параметри, такі як розміри екрану, налаштування фізики тощо.

2. Створення об'єктів та сцен: Розробники розміщують різні об'єкти (такі як персонажі, об'єкти оточення, ефекти тощо) на сцені гри за допомогою системи вузлів у Godot Editor.

3. Програмування логіки гри: За допомогою GDScript, C#, C++ або VisualScript, розробники програмують логіку поведінки об'єктів у грі, включаючи реакцію на взаємодію з гравцем, обробку подій тощо.

4. Тестування і оптимізація: Після завершення розробки гри розробники проводять тестування для виявлення помилок та оптимізують гру для досягнення максимальної продуктивності на різних платформах.

5. Публікація: Після успішного тестування і оптимізації гри, розробники можуть опублікувати свій проект на різних платформах. Godot Engine надає засоби для експорту гри на платформи, такі як iOS, Android, PC, веб та інші.

Публікація гри може включати в себе кроки, такі як:

1. Підготовка проекту до експорту: Розробники перевіряють, що всі ресурси та налаштування гри готові для експорту. Вони також можуть налаштовувати параметри експорту для конкретної платформи.

2. Експорт проекту: Godot Engine надає можливість експортувати гру на різні платформи за допомогою вбудованих інструментів. Розробники можуть вибрати цільову платформу та налаштувати параметри експорту, такі як налаштування графіки, аудіо, управління тощо.

3. Тестування на цільовій платформі: Перед публікацією гри розробники зазвичай проводять тестування на цільовій платформі, щоб переконатися, що гра працює коректно та оптимально на цій платформі.

4. Публікація в магазинах або на інших платформах: Після успішного тестування розробники можуть опублікувати свою гру на цільовій платформі, шляхом розміщення її в магазинах додатків або інших платформах розповсюдження.

5. Підтримка та оновлення: Після публікації гри розробники можуть продовжувати підтримку свого проекту, виправляти помилки, випускати новий контент та оновлення для своєї гри, щоб забезпечити задоволення користувачів та зберегти їхній інтерес до гри.

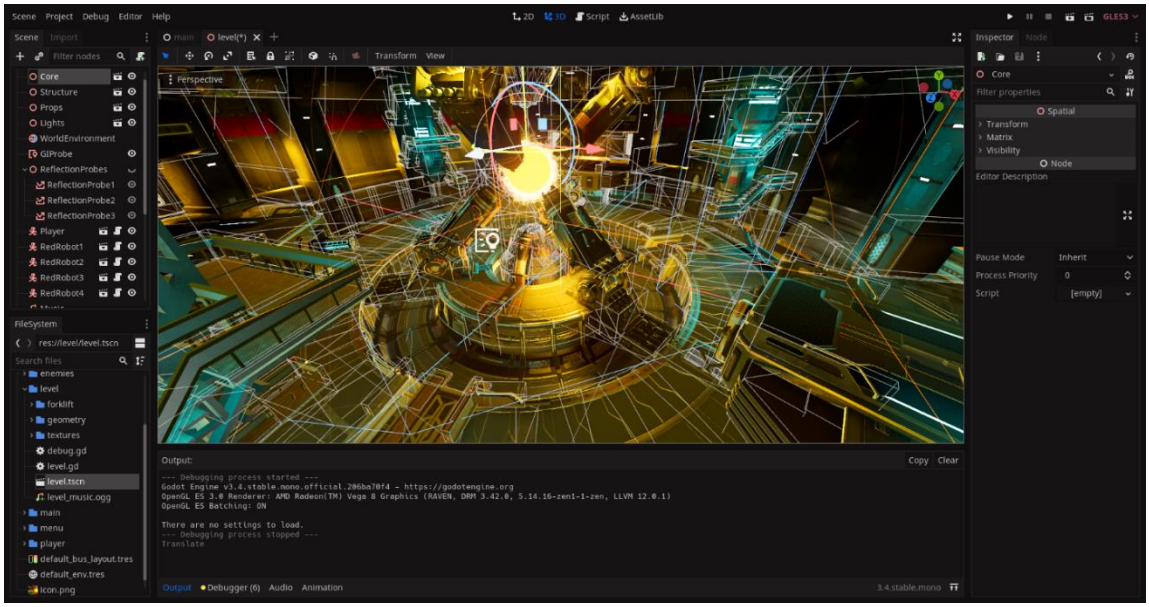

Рисунок 5 –*3 Godot*

2.2 Вибір ігрового рушія

2.2.1 Unity

Переваги:

- Кросплатформеність: Unity підтримує безліч платформ, включаючи мобільні пристрої, ПК, консолі та веб-браузери.
- Зручний інтерфейс: Unity Editor пропонує інтуїтивно зрозумілий і зручний інтерфейс для створення і редагування ігор.

- Велика спільнота: Широка база користувачів та активна спільнота розробників, що сприяє швидкому знаходженню відповідей на питання та рішень проблем.
- Asset Store: Unity має багатий магазин ресурсів, що дозволяє швидко знаходити та використовувати готові моделі, текстури, анімації тощо.
- Підтримка C#: Використання мови програмування C#, яка є зрозумілою та популярною серед розробників.

Недоліки:

- Поточна продуктивність: Незважаючи на постійні покращення, продуктивність Unity може бути нижчою у порівнянні з деякими іншими двигунами.
- Ліцензування: Хоча базова версія безкоштовна, для розширених функцій та підтримки потрібні платні ліцензії.

2.2.2 Unreal Engine

Переваги:

- Висока якість графіки: Unreal Engine відомий своїми потужними можливостями для створення фотореалістичної графіки.
- Blueprints: Візуальна система програмування, яка дозволяє створювати логіку гри без написання коду.
- Потужний ігровий двигун: Забезпечує високу продуктивність і якість графіки навіть для великих проектів.
- Безкоштовний доступ до вихідного коду: Дозволяє глибокі модифікації та налаштування двигуна під конкретні потреби проекту.

Недоліки:

- Високий поріг входження: Через свою складність, Unreal Engine може бути важким для новачків.

- Ресурсоємність: Потребує потужного обладнання для розробки та запуску проектів.

2.2.3 Godot Engine

Переваги:

- Відкритий код: Повністю безкоштовний і відкритий двигун, що дозволяє гнучко налаштовувати проект.
- Легка вага: Godot має невеликі системні вимоги та швидкий запуск.
- Інтуїтивний інтерфейс: Зручний редактор сцени та інструменти для створення 2D та 3D ігор.
- GDScript: Вбудована мова програмування, схожа на Python, що робить її легкою у вивченні.

Недоліки:

- Обмежена документація: Документація може бути менш повною у порівнянні з Unity чи Unreal Engine.
- Менша спільнота: Хоча Godot швидко зростає, його спільнота поки що не така велика, як у Unity чи Unreal.

2.2.4 Вибір Unity

Для проекту я обрав Unity з кількох причин:

- Кросплатформеність: Підтримка багатьох платформ дозволяє забезпечити доступність гри для широкої аудиторії.
- Зручний інтерфейс: Інтуїтивно зрозумілий редактор спрощує процес розробки та знижує поріг входження.
- Велика спільнота та ресурси: Широка спільнота розробників та великий вибір готових ресурсів в Asset Store дозволяють швидко вирішувати проблеми та економити час на створення контенту.

- Підтримка C#: Використання C# забезпечує високу продуктивність та зручність програмування.
- Документація та навчальні матеріали: Наявність великої кількості навчальних матеріалів та документації спрощує навчання та освоєння Unity.

Враховуючи всі ці фактори, Unity виявився найбільш відповідним двигуном для нашого проекту.

2.3 Аналіз вимог до апаратного та програмного забезпечення

Сучасні ігри зазвичай є доволі вимогливими до продуктивності так як вони надають величезний функціонал, включаючи хорошу графіку. У моему випадку я намагався знизити цей поріг [10], щоб більше користувачів могли грати саме в мою гру.

Нижче наведено загальні вимоги до апаратного забезпечення, які можна врахувати при початку гри(характеристики саме такі, так як на такому апаратному забезпечені були проведені заміри продуктивності):

Рекомендовані характеристики апаратного забезпечення:

1. Процесор: Intel Core i7-9750H 2.6Ghz.
2. Графічний процесор: Nvidia GTX 1050 3GB.
3. RAM: 16GB 3200Mhz.
4. Жорсткий диск або SSD: 1GB.
5. Звукова карта: Realtek HD Audio.

2.4 Висновки до розділу 2

1. Розглянуті існуючі ігрові рушії, та обрано Unity.
2. Розглянуті загальні вимоги до апаратного та програмного забезпечення для гри.

3. Вказані рекомендовані параметри, що є достатніми для забезпечення високої продуктивності та графічної якості.

3 РОЗРОБКА ГРИ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY

3.1 Опис предметної області

Ігрова індустрія є однією з найбільш динамічних та інноваційних галузей сучасної цифрової економіки. Вона охоплює широкий спектр технологій, зокрема, розробку ігрових рушіїв, створення тривимірних моделей, розробку алгоритмів штучного інтелекту та багато іншого. В даному розділі буде розглянуто основні аспекти предметної області гри "Epic Adventure", включаючи її концепцію, основні елементи геймплею та технології, що використовуються.

" Epic Adventure" є платформером, де гравці подорожують між різними вимірами для боїв з ворогами і досягнення кінцевої мети гри.

Основні елементи геймплею:

1. Платформні елементи:

Гравець повинен стрибати через платформи, уникати пасток і ворогів, використовуючи свої навички та рефлекси.

Деякі платформи та об'єкти можуть існувати тільки в певних вимірах, що додає складності до геймплею.

Вороги та перешкоди:

У грі присутні різноманітні вороги та перешкоди, які ускладнюють проходження рівнів.

Вороги мають різні поведінкові моделі, що змушує гравця розробляти стратегії для їх подолання.

Технології, що використовуються

1. Ігровий рушій Unity:

Гра " Epic Adventure" розроблена на базі ігрового рушія Unity, який забезпечує потужні можливості для створення тривимірних та двовимірних ігор.

Unity підтримує кросплатформенну розробку, що дозволяє випускати гру на різних пристроях та операційних системах.

2. Графічний рушій:

Використання Unity дозволяє реалізувати високоякісну графіку з оптимізацією для різних пристроїв.

Графічний рушій підтримує роботу з шейдерами, освітленням та іншими візуальними ефектами, що підвищують занурення у гру.

Фізичний рушій:

Вбудований фізичний рушій Unity забезпечує реалістичну симуляцію фізики, що є критично важливим для платформерів.

Фізичний рушій використовується для моделювання руху персонажів, об'єктів та взаємодії між ними.

Система скриптів:

Unity використовує мову програмування C# для написання скриптів, що керують поведінкою об'єктів у грі.

Система скриптів дозволяє створювати складні ігрові механіки та інтерактивність.

3.2 Функціональні вимоги системи

3.2.1 Загальні вимоги

Функціональні вимоги до гри " Epic Adventure" включають різноманітні аспекти, які забезпечують її функціональність, зручність використання та ефективність. Нижче наведено ключові функціональні вимоги, що стосуються як ігрового рушія.

1. Функціональні вимоги до ігрового рушія " Epic Adventure"

A. Рендеринг та графіка

- Підтримка 2D: Гра повинна відображати 2D елементи.

- Система освітлення: Забезпечення освітлення для створення реалістичної атмосфери.

B. Фізика та анімація

- Реалістична фізика об'єктів: Впровадження фізичного рушія для обробки зіткнень, гравітації та інших фізичних взаємодій.
- Анімація персонажів та об'єктів: Підтримка анімації для всіх об'єктів у грі з використанням ключових кадрів та скелетної анімації.

C. Звукові ефекти та музика

- Відтворення звукових ефектів: Включення звукових ефектів для дій, подій та взаємодій у грі.
- Фонове музичне супроводження: Підтримка відтворення фонової музики з можливістю зміни треків під час гри.

D. Ігрова логіка та сценарії

- Скриптовий рушій: Підтримка скриптів для реалізації ігрової логіки, взаємодій та подій.
- Ігрові об'єкти та взаємодія: Визначення поведінки та взаємодій ігрових об'єктів з використанням скриптів.
- Система управління рівнями: Можливість створення та завантаження різних рівнів гри.

E. Користувацький інтерфейс (UI)

- Інтерактивне меню: Підтримка головного меню, меню паузи.
- Ігрові індикатори: Відображення індикаторів здоров'я.
- Система повідомлень: Відображення повідомлень про події у грі.

F. Оптимізація та продуктивність

- Ефективне використання ресурсів: Оптимізація використання пам'яті та обробки для забезпечення плавної роботи на різних пристроях.

3.3 Архітектура системи

Архітектура гри " Epic Adventure" [9] побудована з урахуванням потреб ігрових розробників, забезпечуючи високий рівень продуктивності та гнучкість для майбутнього розвитку. Нижче наведено основні компоненти архітектури системи та їх взаємодію.

1. Основні компоненти архітектури

a. Ігровий рушій Unity

- Використовується для управління всіма аспектами гри, включаючи рендеринг, фізику, звук, анімацію та управління ресурсами.
- Платформа Unity забезпечує кросплатформенну розробку, дозволяючи запускати гру на різних пристроях.

b. Графічний модуль

- Відповідає за відображення графічних об'єктів, використовуючи можливості Unity.
- Використовує шейдери для створення спеціальних ефектів і освітлення.

c. Фізичний модуль

- Використовує вбудований фізичний рушій Unity для симуляції реалістичних фізичних взаємодій між об'єктами.
- Забезпечує моделювання гравітації, зіткнень, руху об'єктів та інших фізичних явищ.

d. Аудіо модуль

- Відповідає за відтворення звукових ефектів та музики.

- Підтримує 3D-звук та інші технології звукової обробки для створення реалістичної звукової атмосфери.

e. Скриптовий модуль

- Реалізує логіку гри за допомогою мови програмування C#.
- Відповідає за керування поведінкою об'єктів, реакцію на події та взаємодію з користувачем.

g. Система управління ресурсами

- Відповідає за завантаження, зберігання та управління ресурсами гри, такими як текстури, моделі, анімації та звукові файли.
- Забезпечує оптимізацію використання пам'яті та швидке завантаження ресурсів.

h. Інтерфейс користувача (UI)

- Використовує Unity UI для створення інтуїтивно зрозумілого та привабливого інтерфейсу для користувачів.
- Включає меню, кнопки, інформаційні панелі та інші елементи UI.

2. Архітектурна схема

a. Клієнтська частина (гра на пристрої користувача)

- Unity Engine: Серце гри, яке обробляє всі аспекти геймплею та графіки.
- Graphics Module: Відповідає за візуалізацію та рендеринг.
- Physics Module: Обробляє фізику об'єктів і взаємодій.
- Audio Module: Управляє звуковими ефектами та музикою.
- Scripting Module: Виконує скрипти, що визначають поведінку об'єктів та ігрову логіку.
- Resource Management System: Управляє ресурсами гри.
- User Interface (UI): Забезпечує інтерфейс для взаємодії з гравцем.

b. Взаємодія компонентів

- Ігровий рушій Unity взаємодіє з усіма модулями, забезпечуючи інтеграцію графіки, фізики, звуку та логіки гри.
- Графічний модуль та фізичний модуль працюють разом для створення реалістичних візуальних і фізичних ефектів.
- Аудіо модуль підключається до Unity для відтворення звукових ефектів відповідно до подій у грі.
- Скриптовий модуль управляє логікою гри, реагуючи на дії гравця та взаємодію об'єктів.
- Система управління ресурсами забезпечує ефективне завантаження та зберігання ресурсів для оптимізації продуктивності.
- Інтерфейс користувача взаємодіє з гравцем, надаючи зворотний зв'язок та керування грою.

3. Вимоги до продуктивності та оптимізація

Архітектура системи розроблена з урахуванням вимог до продуктивності, що дозволяє забезпечити плавний геймплей та швидкий відгук на дії гравця. Використання ефективних алгоритмів та оптимізація ресурсів дозволяє знизити навантаження на систему та забезпечити комфортну гру навіть на менш потужних пристроях.

Архітектура гри "Epic Adventure" забезпечує високий рівень продуктивності та гнучкість для розробки нових функцій і розширень. Використання Unity як основного ігрового рушія дозволяє створювати високоякісні ігри з мінімальними затратами часу та ресурсів. Завдяки добре спланованій структурі та ефективній взаємодії між компонентами, гра " Epic Adventure" може задовольнити потреби як розробників, так і гравців, забезпечуючи захоплюючий та безперебійний геймплей.

3.4 Функціонал гри “ Epic Adventure”

Оскільки гра має надавати якнайбільше функціоналу користувачеві, в "Epic Adventure" було реалізовано значну кількість різноманітних функцій, щоб полегшити гру користувачам. Основна мета полягала в тому, щоб спростити створення базових механік і надати гравцям можливість зосередитися на унікальних функціях гри. Хоча реалізовано багато різноманітних функцій, цей розділ надасть лише поверхневий огляд основних можливостей гри. Це допоможе скласти загальне уявлення про її роботу та алгоритми, які були використані під час розробки.

Таким чином, "Epic Adventure" пропонує гравцям насичений ігровий досвід, наповнений цікавими пригодами та викликами.

На **Помилка! Джерело посилання не знайдено.** 5 зображено діаграму класів персонажу.

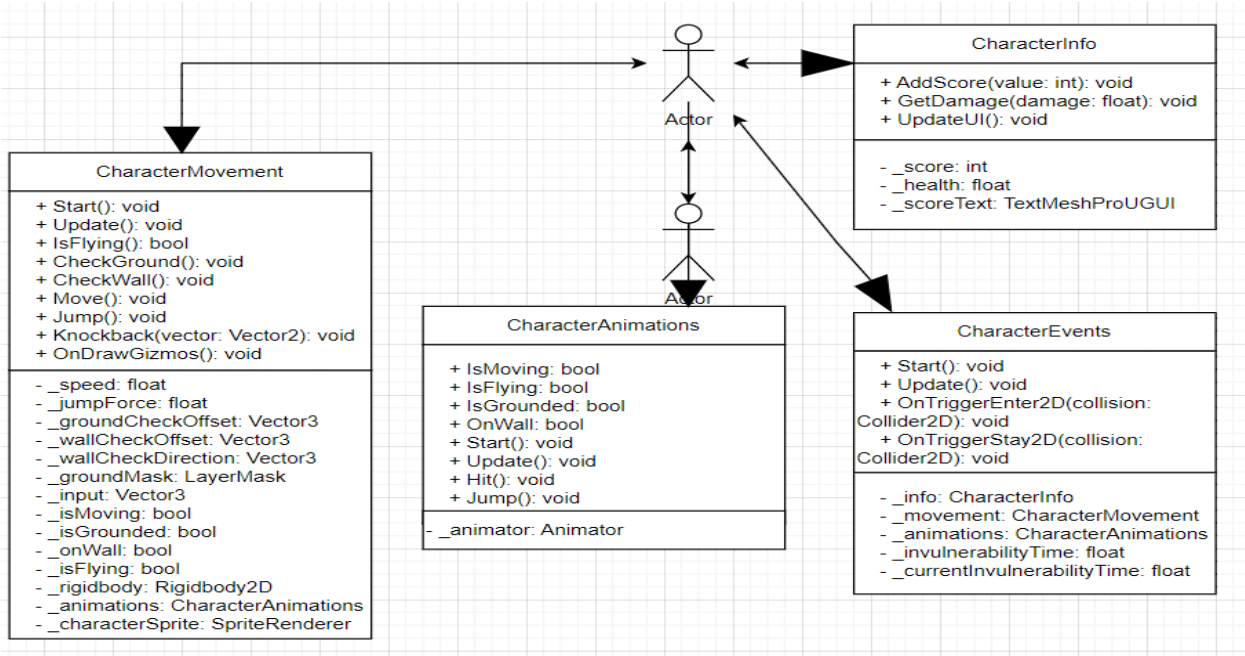

Рисунок 5 – *Діаграма класів персонажу*

Пересування персонажа

В лістингу 1 представлено реалізацію класу, який відповідає за пересування персонажу.

Лістинг *1 Пересування персонажа клас “ CharacterMovement”*

```
using UnityEngine;

public class CharacterMovement : MonoBehaviour
{
    [SerializeField] private float _speed;
    [SerializeField] private float _jumpForce;
    [SerializeField] private Vector3 _groundCheckOffset;
    [SerializeField] private Vector3 _wallCheckOffset;
    [SerializeField] private Vector3 _wallCheckDirection;

    [SerializeField] private LayerMask groundMask;

    private Vector3 _input;
    private bool _isMoving;
    private bool _isGrounded;
    [SerializeField] private bool _onWall;
    private bool _isFlying;

    private Rigidbody2D _rigidbody;
    private CharacterAnimations _animations;
    [SerializeField] private SpriteRenderer _characterSprite;

    private void Start()
    {
        _rigidbody = GetComponent<Rigidbody2D>();
        _animations = GetComponent<CharacterAnimations>();
    }
```

```
private void Update()
{
    Move();
    CheckGround();
    CheckWall();
    if (Input.GetKeyDown(KeyCode.Space))
    {
        Jump();
    }

    _animations.IsMoving = _isMoving;
    _animations.IsGrounded = _isGrounded;
    _animations.OnWall = _onWall;
    _animations.IsFlying = IsFlying();
}

private bool IsFlying()
{
        if (_rigidbody.velocity.y < 0)
        {
            return true;
        }
        else
        {
            return false;
        }
}

private void CheckGround()
{
    float rayLength = 0.3f;
```

```
            Vector3 rayStartPosition = transform.position +
_groundCheckOffset;
            RaycastHit2D                hit                =
Physics2D.Raycast(rayStartPosition,    rayStartPosition    +
Vector3.down, rayLength, groundMask);

            if (hit.collider != null)
            {
                _isGrounded                                =
hit.collider.CompareTag("Ground") ? true : false;
            }
            else
            {
                _isGrounded = false;
            }
        }

        private void CheckWall()
        {
            float rayLength = 0.05f;
            Vector3 rayStartPosition = transform.position +
_wallCheckOffset * _wallCheckDirection.x;
            RaycastHit2D              wallHit              =
Physics2D.Raycast(rayStartPosition,    rayStartPosition    +
_wallCheckDirection, rayLength, groundMask);

            if (wallHit.collider != null)
            {
                if (wallHit.collider.CompareTag("Ground"))
                {
                    if (!_onWall)
                    {
```

```
                _rigidbody.velocity = Vector2.zero;
            }
            _onWall = true;
            if (_rigidbody.velocity.y < 0)
            {
                _rigidbody.gravityScale = 0.1f;
                _rigidbody.mass = 0.5f;
            }
            else
            {
                _rigidbody.gravityScale = 1;
                _rigidbody.mass = 1f;
            }

        }
        else
        {
            _onWall = false;
            _rigidbody.gravityScale = 1;
            _rigidbody.mass = 1f;
        }
    }
    else
    {
        _onWall = false;
        _rigidbody.gravityScale = 1f;
        _rigidbody.mass = 1f;
    }
}

private void Move()
{
```

```
            _input                    =                    new
Vector2(Input.GetAxis("Horizontal"), 0);
            if (!_onWall)
            {
                transform.position  +=  _input  *  _speed  *
Time.deltaTime;
            }
            else
            {
                if (_wallCheckDirection.x > 0)
                {
                    if(_input.x < 0)
                    {
                        transform.position   +=   _input   *
_speed * Time.deltaTime;
                    }
                }
                else
                {
                    if(_input.x > 0)
                    {
                        transform.position   +=   _input   *
_speed * Time.deltaTime;
                    }
                }
            }
            _isMoving = _input.x != 0 ? true : false;

            if (_isMoving && !_onWall)
            {
                _characterSprite.flipX  =  _input.x  >  0  ?
false : true;
```

```
                _wallCheckDirection = _input.x > 0 ?
transform.right : -transform.right;
            }
        }

        private void Jump()
        {

            if (_isGrounded)
            {
                _rigidbody.AddForce(transform.up          *
_jumpForce, ForceMode2D.Impulse);
                _animations.Jump();
            }
            else
            {
                if (_onWall)
                {
                    _rigidbody.velocity = Vector2.zero;
                    _rigidbody.AddForce(transform.up      *
_jumpForce / 2 - _wallCheckDirection * _jumpForce / 7,
ForceMode2D.Impulse);
                    _characterSprite.flipX                =
_wallCheckDirection.x > 0 ? true : false;
                    _input = _wallCheckDirection * -1;
                    _wallCheckDirection *= -1;
                    _animations.Jump();
                }
            }
        }

        public void Knockback(Vector2 vector)
```

```
        {
            _rigidbody.velocity = Vector2.zero;
            float knockbackForce = 3f;
            _rigidbody.AddForce(vector   *   knockbackForce,
ForceMode2D.Impulse);
        }

        private void OnDrawGizmos()
        {
            Gizmos.color = Color.red;
            Vector3 rayStartPosition = transform.position +
_groundCheckOffset;
            Gizmos.DrawLine(rayStartPosition,
rayStartPosition + Vector3.down * 0.3f);

            Gizmos.color = Color.red;
            Vector3 ray2StartPosition = transform.position +
_wallCheckOffset * _wallCheckDirection.x;
            Gizmos.DrawLine(ray2StartPosition,
ray2StartPosition + _wallCheckDirection * 1f);
        }
    }
```

Клас `CharacterMovement` відповідає за керування рухами персонажа в 2D просторі, включаючи рух, стрибки, перевірку на зіткнення з землею і стінами. Він також забезпечує взаємодію з анімаціями, відповідно до дій персонажа. Ось детальний опис основних методів та функцій цього класу:

Основні поля:

– `_speed`: швидкість руху персонажа.

– `_jumpForce`: сила стрибка персонажа.

– _groundCheckOffset, _wallCheckOffset, _wallCheckDirection: вектори для визначення положення перевірки зіткнень із землею та стінами.

– groundMask: маска шару для визначення землі.

– _input: вектор для зберігання напрямку руху.

– _isMoving: булеве значення для визначення, чи рухається персонаж.

– _isGrounded: булеве значення для визначення, чи знаходиться персонаж на землі.

– _onWall: булеве значення для визначення, чи знаходиться персонаж на стіні.

– _isFlying: булеве значення для визначення, чи знаходиться персонаж у повітрі.

– _rigidbody: компонент фізичного тіла персонажа.

– _animations: посилання на компонент анімацій персонажа.

– _characterSprite: спрайт персонажа для керування його відображенням.

Основні методи:

- Start() - Ініціалізація компонентів:

– GetComponent<Rigidbody2D>() отримує компонент фізичного тіла.

– GetComponent<CharacterAnimations>() отримує компонент анімацій.

- Update() - Метод, що викликається кожен кадр, обробляє рух, перевірку на зіткнення, стрибки і оновлює анімації:

– Move(): керування рухом персонажа.

– CheckGround(): перевірка, чи знаходиться персонаж на землі.

– CheckWall(): перевірка, чи знаходиться персонаж на стіні.

- Обробка стрибка при натисканні клавіші пробілу.

- Оновлення станів анімацій відповідно до дій персонажа.

- `IsFlying()` - Метод, що визначає, чи знаходиться персонаж у повітрі:

Перевірка вертикальної швидкості тіла персонажа.

- `CheckGround()` - Метод, що перевіряє наявність землі під персонажем:

Використання променевого перетину для визначення контакту з землею.

- `CheckWall()` - Метод, що перевіряє наявність стіни поруч із персонажем:

- Використання променевого перетину для визначення контакту зі стіною.

- Налаштування фізичних параметрів персонажа залежно від контакту зі стіною.

- `Move()` - Метод, що керує рухом персонажа:

- Отримання вхідних даних від користувача.

- Переміщення персонажа.

- Налаштування напрямку спрайту персонажа залежно від напрямку руху.

- Оновлення стану руху персонажа.

- `Jump()` - Метод, що відповідає за стрибок персонажа:

Додавання сили стрибка, якщо персонаж знаходиться на землі або на стіні.

- `Knockback(Vector2 vector)` - Метод, що реалізує відштовхування персонажа при отриманні удару:

Додавання сили відштовхування персонажа.

- `OnDrawGizmos()` - Метод, що візуалізує перевірку зіткнень в редакторі Unity:

Малювання ліній для перевірки зіткнень із землею та стінами.

Цей клас забезпечує основну функціональність для керування рухом персонажа в 2D грі, надаючи гнучкі можливості для налаштування руху, стрибків та перевірок зіткнень, а також інтеграції з анімаційними системами.

Зміна анімації персонажу

В лістингу 2 представлено реалізацію класу, який відповідає за зміну анімації персонажу.

Лістинг *2 Зміна анімації персонажу клас "CharacterAnimations"*

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

public class CharacterAnimations : MonoBehaviour
{
    private Animator _animator;

    public bool IsMoving { private get; set; }
    public bool IsFlying { private get; set; }
    public bool IsGrounded { private get; set; }
    public bool OnWall { private get; set; }

    private void Start()
    {
        _animator = GetComponentInChildren<Animator>();
    }

    private void Update()
    {
        _animator.SetBool("IsMoving", IsMoving);
```

```
            _animator.SetBool("IsFlying", IsFlying);
            _animator.SetBool("IsGrounded", IsGrounded);
            _animator.SetBool("OnWall", OnWall);
        }

        public void Hit()
        {
            _animator.SetTrigger("Hit");
        }

        public void Jump()
        {
            if(_animator.GetBool("IsFlying") == false)
            {
                _animator.SetTrigger("Jump");
            }
        }
    }
```

Клас `CharacterAnimations` відповідає за керування анімаціями персонажа в грі. Він отримує стан персонажа (рух, стрибок, зіткнення зі стіною тощо) від інших компонентів і передає ці стани до аніматора, щоб правильно відображати відповідні анімації.

Основні поля та властивості

`private Animator _animator`: Приватне поле, яке містить компонент Animator, відповідає за керування анімаціями.

`public bool IsMoving { private get; set; }`: Публічна властивість для визначення, чи рухається персонаж.

`public bool IsFlying { private get; set; }`: Публічна властивість для визначення, чи знаходиться персонаж у повітрі.

`public bool IsGrounded { private get; set; }`: Публічна властивість для визначення, чи знаходиться персонаж на землі.

`public bool OnWall { private get; set; }`: Публічна властивість для визначення, чи знаходиться персонаж на стіні.

Основні методи

- `Start()` - Метод, який викликається під час ініціалізації компонента:

Отримання компонента `Animator` з дочірнього об'єкта персонажа.

- `Update()` - Метод, який викликається кожен кадр:

Оновлює значення булевих параметрів аніматора відповідно до стану персонажа.

- `Hit()` - Метод для активації анімації удару:

Встановлює тригер Hit в аніматорі, що викликає відповідну анімацію.

- `Jump()` - Метод для активації анімації стрибка:

Перевіряє, чи персонаж не знаходиться в стані польоту, і встановлює тригер `Jump` в аніматорі, що викликає відповідну анімацію.

Використання класу

Клас `CharacterAnimations` отримує інформацію про стан персонажа від інших компонентів (наприклад, `CharacterMovement`) і відповідно оновлює анімації. Це забезпечує плавний перехід між різними станами персонажа, такими як рух, стрибок, зіткнення зі стіною тощо.

Обробка подій з персонажем

У лістингу 3 представлено клас, який відповідає за обробку подій персонажем.

Лістинг 3 *Обробка подій з персонажем клас " `CharacterEvents` "*

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

```
using UnityEngine;

public class CharacterEvents : MonoBehaviour
{
    private CharacterInfo _info;
    private CharacterMovement _movement;
    private CharacterAnimations _animations;

    [SerializeField] private float _invulnerabilityTime;
    private float _currentInvulnerabilityTime;

    private void Start()
    {
        _info = FindObjectOfType<CharacterInfo>();
        _movement = GetComponent<CharacterMovement>();
        _animations                                    =
GetComponent<CharacterAnimations>();
    }

    private void Update()
    {
        if(_currentInvulnerabilityTime > 0)
        {
            _currentInvulnerabilityTime                -=
Time.deltaTime;
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if  (collision.TryGetComponent<Fruit>(out  Fruit
fruit))
```

```
            {
                fruit.Destroy();
                _info.AddScore(fruit.Points);
            }
        }

        private void OnTriggerStay2D(Collider2D collision)
        {
            if   (collision.TryGetComponent<Trap>(out   Trap
trap))
            {
                if (_currentInvulnerabilityTime <= 0)
                {
                    Vector2 difference = transform.position
- trap.transform.position;

_movement.Knockback(difference.normalized);
                    _animations.Hit();
                    _currentInvulnerabilityTime           =
_invulnerabilityTime;
                }
            }
        }
    }
```

Клас `CharacterEvents` відповідає за обробку подій, що відбуваються з персонажем під час гри, таких як зіткнення з предметами та пастками. Він також реалізує механіку невразливості персонажа після отримання шкоди.

Основні поля та властивості

`private CharacterInfo _info`: Приватне поле для зберігання інформації про персонажа, наприклад, його очок.

`private CharacterMovement _movement`: Приватне поле для керування рухом персонажа.

`private CharacterAnimations _animations`: Приватне поле для керування анімаціями персонажа.

`[SerializeField] private float _invulnerabilityTime`: Час, протягом якого персонаж буде невразливим після отримання шкоди.

`private float _currentInvulnerabilityTime`: Поточний час невразливості персонажа.

Основні методи

- `Start()`

Метод, який викликається під час ініціалізації компонента:

Ініціалізує поля `_info`, `_movement`, та `_animations`, знаходячи відповідні компоненти.

```
private void Start()
{
    _info = FindObjectOfType<CharacterInfo>();
    _movement = GetComponent<CharacterMovement>();
    _animations = GetComponent<CharacterAnimations>();
}
```

- `Update()`

Метод, який викликається кожен кадр:

Зменшує час невразливості персонажа, якщо він більше нуля.

```
private void Update()
```

```
{
    if(_currentInvulnerabilityTime > 0)
    {
        _currentInvulnerabilityTime -= Time.deltaTime;
    }
}
```

- `OnTriggerEnter2D(Collider2D collision)` - Метод, який викликається при зіткненні персонажа з іншими об'єктами, що мають тригерні коллайдери:

Якщо персонаж зіштовхується з об'єктом типу `Fruit`, він знищує цей об'єкт і додає відповідні очки до загального рахунку персонажа.

- `OnTriggerStay2D(Collider2D collision)` - Метод, який викликається, коли персонаж перебуває в зоні тригерного коллайдера іншого об'єкта:

Якщо персонаж зіштовхується з об'єктом типу `Trap`, він отримує шкоду та відштовхується, якщо час невразливості минув.

Використання класу

Клас `CharacterEvents` інтегрує події гри, такі як збирання предметів та уникнення пасток, з механікою персонажа. Це дозволяє реалізувати різні сценарії взаємодії персонажа з об'єктами гри та додавати відповідні ефекти, такі як анімації удару та невразливість.

Збереження даних персонажа

У лістингу 4 представлено клас, який відповідає за збереження даних персонажу.

Лістинг *4 Збереження даних персонажа клас " `CharacterInfo`"*

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

```
using UnityEngine;
using TMPro;

public class CharacterInfo : MonoBehaviour
{
    private int _score;
    private float _health;

    [SerializeField] private TextMeshProUGUI _scoreText;

    public void AddScore(int value)
    {
        _score += value;
        UpdateUI();
    }

    public void GetDamage(float damage)
    {
        _health -= damage;
    }

    private void UpdateUI()
    {
        _scoreText.text = "Score: " + _score.ToString();
    }
}
```

Клас `CharacterInfo` відповідає за зберігання та керування основними характеристиками персонажа, такими як очки та здоров'я. Він також реалізує оновлення інтерфейсу користувача для відображення поточних очок.

Основні поля та властивості :

- `private int _score`: Приватне поле для зберігання кількості очок персонажа.
- `private float _health`: Приватне поле для зберігання поточного рівня здоров'я персонажа.
- `[SerializeField] private TextMeshProUGUI _scoreText`: Поле для зберігання посилання на текстовий елемент інтерфейсу, що відображає очки персонажа.

Основні методи :

- `AddScore(int value)` - Метод, який додає вказану кількість очок до загального рахунку персонажа:

– Збільшує значення `_score` на вказане значення `value`.

– Викликає метод `UpdateUI()` для оновлення інтерфейсу користувача.

- `GetDamage(float damage)` - Метод, який зменшує рівень здоров'я персонажа на вказане значення:

- Зменшує значення `_health` на вказане значення `damage`.

- `UpdateUI()` - Приватний метод, який оновлює текстовий елемент інтерфейсу користувача для відображення поточних очок:

– Оновлює текстове поле `_scoreText` значенням `_score`.

Клас `CharacterInfo` дозволяє керувати основними характеристиками персонажа, такими як очки та здоров'я, а також забезпечує оновлення інтерфейсу користувача. Це допомагає інтегрувати механіку очок та шкоди в гру та відображати їх для гравця.

Цей клас дозволяє легко керувати основними характеристиками персонажа та забезпечує їх коректне відображення в інтерфейсі користувача, що робить гру більш інтерактивною та інформативною для гравця.

3.5 Реалізація і тестування

3.5.1 Реалізація

Фізичні характеристики поточної версії системи:

Об'єм коду:

1. C# — 9087 кілобайт, ~400 рядків.

Кількість модулів: 7.

Кількість і об'єм, в кілобайтах, програмних компонент:

1. Додаток “TWE” — 1387062,99 кілобайт, 5 компонентів.

У таблиці 1 надано швидкодію гри.

Таблиця *1* - *Таблиця швидкодії гри “Epic Adventure ”*

Стан додатку	CPU, %	RAM, MB	GPU, %	FPS
У редакторі при зупиненій грі.	~ 10.7%	~ 95 MB	~ 42%	~1000 FPS
У редакторі при запущеній грі.	~ 11%	~ 105.4 MB	~ 44%	~900 FPS
У білді гри	~ 12%	~ 82 MB	~ 55%	~1600 FPS

3.6.2 Тестування

У грі "Adventures of Pixel" ви повинні керувати персонажем на ім'я Піксель та долати різноманітні перепони, зібрати полуницю і перестрибувати перешкоди. Щоб завершити гру, потрібно пройти усі рівні, доходячи до фінішу кожного рівня. Ця гра добре відображає взаємодію фізики, як персонаж рухається по платформах, як здійснює стрибки та зіштовхується з об’єктами.

На рисунку 6 зображено головний екран гри "Adventures of Pixel".

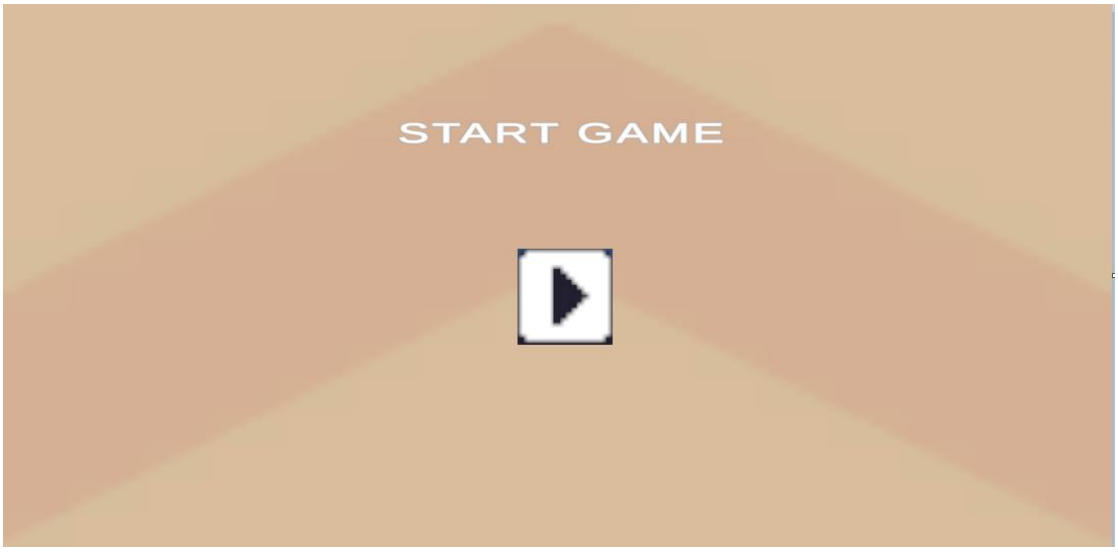

Рисунок 6 – *Головний екран*

На рисунку 7 зображено почато гри "Adventures of Pixel".

Рисунок 7 – *Початок гри*

На рисунку 8 зображено анімацію стрибка.

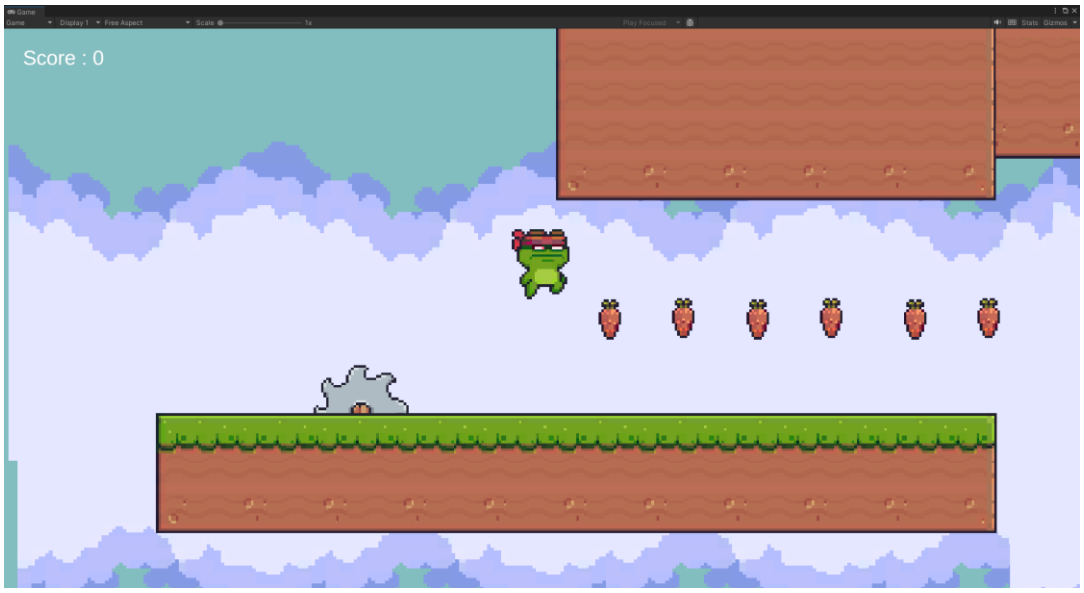

Рисунок 8 – *Анімація стрибка*

На рисунку 9 зображено взаємодію головного персонажу зі стіною

Рисунок 9 – *Взаємодію головного персонажу зі стіною*

На рисунку 10 зображено отримання головним персонажем шкоди від перешкоди

Рисунок 10 – *Отримання головним персонажем шкоди від перешкоди*

На рисунку 11 зображено збір головним персонажем полуниці та збільшення кількості очок

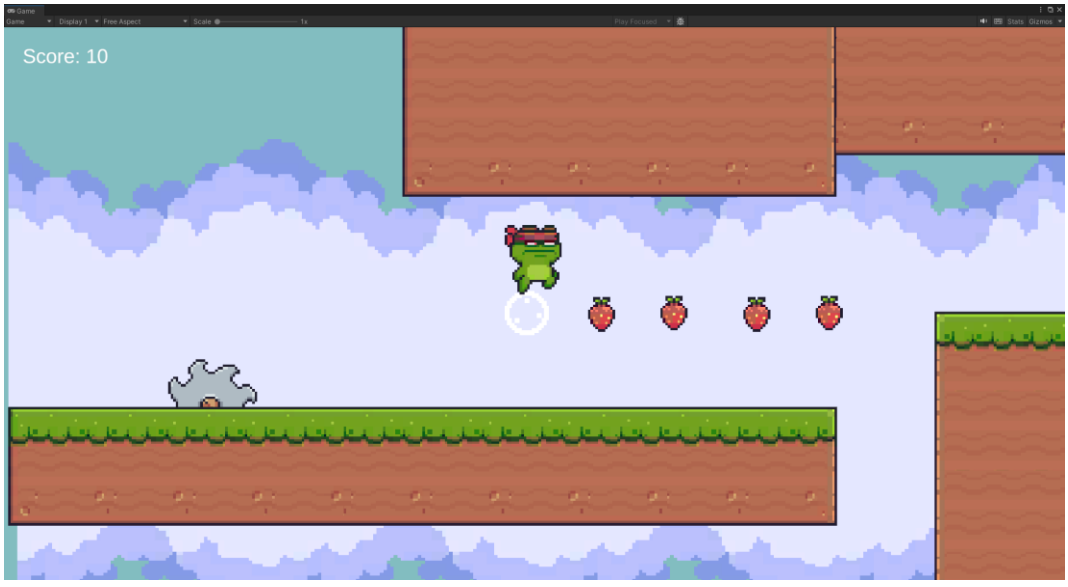

Рисунок 11 – *Збір головним персонажем полуниці та збільшення кількості очок*

Коли гравець доходить до кінцевої точки рівня, у правій стороні екрана з'являється кнопка, яка дає змогу перейти на наступний рівень, на рисунку 12 зображено кнопку для переходу на наступний рівень

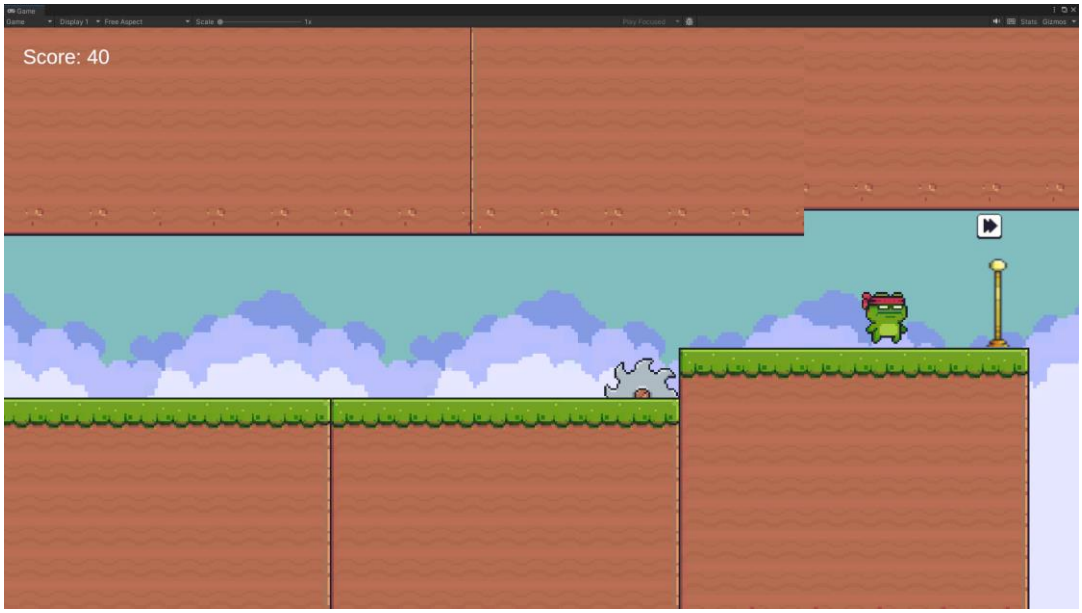

Рисунок 12 – *Кнопка для переходу на наступний рівень*

Коли гравець натискає кнопку для переходу на наступний рівень, з'являється ефект затемнення, який додає атмосфери. На рисунку 13 зображено ефект затемнення, перед появлення гравця на рівні.

Рисунок 13 – *Ефект затемнення перед появою гравця на рівні*

3.7 Висновки до розділу 3

У цьому розділі було детально розглянуто процес розробки гри "Epic Adventure" з використанням ігрового рушія Unity. Основні аспекти, розглянуті в розділі, включають опис предметної області, функціональні вимоги системи, архітектуру системи та основні функціональні можливості гри.

Опис предметної області

Гра "Epic Adventure" є платформером, де гравці подорожують між різними вимірами для боїв з ворогами і досягнення кінцевої мети. Геймплей включає платформні елементи, унікальні для різних вимірів, та різноманітних ворогів з різними поведінковими моделями. Unity, як основний ігровий рушій, дозволяє реалізувати високоякісну графіку, реалістичну фізику та інтерфейс користувача, що забезпечує захоплюючий ігровий досвід.

Функціональні вимоги системи

Функціональні вимоги гри охоплюють різноманітні аспекти, які забезпечують її функціональність та ефективність. Серед ключових вимог – підтримка 2D графіки, реалістична фізика, відтворення звукових ефектів та музики, інтерактивний користувацький інтерфейс, а також оптимізація ресурсів для забезпечення плавної роботи на різних пристроях. Використання мови програмування C# для написання скриптів дозволяє створювати складні ігрові механіки та інтерактивність.

Архітектура системи

Архітектура гри "Epic Adventure" включає основні компоненти, такі як ігровий рушій Unity, графічний модуль, фізичний модуль, аудіо модуль, скриптовий модуль, система управління ресурсами та інтерфейс користувача. Взаємодія цих компонентів забезпечує інтеграцію графіки, фізики, звуку та логіки гри, що дозволяє створити реалістичні візуальні та фізичні ефекти. Ефективне управління ресурсами забезпечує оптимізацію продуктивності та комфортну гру навіть на менш потужних пристроях

Функціонал гри

Гра "Epic Adventure" надає користувачам широкий функціонал для забезпечення насиченого ігрового досвіду. Основні функціональні можливості включають:

- Пересування персонажа: Клас `CharacterMovement` відповідає за керування рухом персонажа в 2D просторі, включаючи рух, стрибки, перевірку на зіткнення з землею і стінами.
- Зміна анімації персонажа: Клас `CharacterAnimations` керує анімаціями персонажа, передаючи стани персонажа до аніматора для правильного відображення відповідних анімацій.

Таким чином, розробка гри "Epic Adventure" з використанням Unity дозволила реалізувати інноваційні рішення для створення інтерактивного та захоплюючого ігрового досвіду. Використання Unity як основного ігрового рушія забезпечило високоякісну графіку, реалістичну фізику та інтерактивність, що робить гру привабливою для гравців різного віку.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи можна зробити наступні висновки:

1. Доведено ефективність Unity для розробки 2D-платформерів.

2. Проведено порівняльний аналіз існуючих ігрових рушіїв на ринку.

3. Використання C# забезпечило створення ігрової логіки, високу продуктивність і стабільність гри.

4. Виявлено гнучкість Unity як засобу розробки та можливість його розширюваності додатковими інструментами.

В цілому, розробка 2D-платформеру з використанням Unity продемонструвала високу ефективність цього ігрового рушія та його спроможність забезпечувати розробників усім необхідним для створення якісних та захоплюючих ігор. Отриманий досвід є цінним і відкриває нові можливості для майбутніх проектів у сфері ігрової індустрії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unreal Engine. URL: https://docs.unrealengine.com/4.27/en-US/. (дата звернення: 03.04.2023).
2. Unity. URL: https://docs.unity.com. (дата звернення: 03.04.2023).
3. Godot. URL: https://docs.godotengine.org/en/stable/. (дата звернення: 03.04.2023).
4. Super Mario. URL: https://super-mario.fandom.com/ru/wiki/Super_Mario_Bros.. (дата звернення: 05.04.2023)
5. Rayman Legends. URL: https://ua.gecid.com/games/rayman_legends/. (дата звернення: 05.04.2023).
6. Unity in Action by Joe Hocking, July 7, 2015. 352 с.
7. Unity Game Development Cookbook by Tim Nugent, April 16, 2019. 405 с.
8. Unity Learn URL: https://learn.unity.com/ (дата звернення: 03.05.2023)
9. Unity 2D Game Development URL: https://docs.unity3d.com/Manual/Unity2D.html (дата звернення: 13.04.2023).
10. Оптимізація графіки на Unity. Як використовувати LOD, Draw Calls, Texture Mipmaps та інші техніки рушія. URL: https://gamedev.dou.ua/blogs/unity-engine-optimization-labyrinths/ (дата звернення: 20.04.2023).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Чистіков Вадим Сергійович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-203@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Розробка 2D-платформеру з використанням ігрового рушія Unity»** відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 17.06.2024 _____ Чистіков Вадим Сергійович
(студент)

Дата 18.06.2024 _____ Скрипник Ірина Анатоліївна
(науковий керівник)