Міністерство освіти і науки України
Університет митної справи та фінансів
Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розробка ігрового додатку за допомогою інтегрованого середовища Unity»

Виконав: здобувач вищої освіти групи ІПЗ19-1
спеціальності 121 «Інженерія програмного забезпечення» першого освітнього рівня забезпечення
Андрасович Максим Вадимович
Керівник: *д.е.н. доцент Небаба Н.О.*

Дніпро – 2023

АНОТАЦІЯ

Андрасович М.В. Розробка ігрового додатку за допомогою інтегрованого середовища Unity.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення» першого освітнього рівня забезпечення». – Університет митної справи та фінансів, Дніпро, 2023.

Дана кваліфікаційна робота присвячена розробці ігрового додатку з використанням інтегрованого середовища Unity. У роботі проведено дослідження предметної області, включаючи аналіз ігрових платформ та рушіїв. Було визначено основні цілі та вимоги до ігрового додатку і розроблено програмне рішення.

Основний акцент у роботі зроблено на використанні інтегрованого середовища Unity, яке є потужним інструментом для розробки якісних ігрових додатків. Проведено детальний аналіз можливостей, функціональності та переваг цього рушія. Вивчено принципи роботи зі сценами, об'єктами, компонентами та скриптами в Unity, а також виконано порівняльний аналіз з альтернативними рушіями.

У процесі розробки ігрового додатку були використані сучасні методики та практики програмування, включаючи використання мови програмування C# для реалізації ігрової логіки та компонентів. Проведено тестування розробленого додатку з метою виявлення та усунення помилок та недоліків.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновку, списку використаних джерел. Робота виконана на 58 сторінках, містить 40 рисунків. Список бібліографічних посилань включає 40 найменувань.

Ключові слова: Unity, source, godot, платформер, рушій.

ABSTRACT

Andrasovych M.V. Game application development using the integrated Unity environment.

Qualification work for the degree of "Bachelor" in the specialty 121 "Software Engineering" of the first level of education. Dnipro: University of Customs and Finance. 2023.

This qualifying work is devoted to the development of a game application using the integrated Unity environment. The research of the subject area, including the analysis of game platforms and engines, was carried out in the work. The main goals and requirements for the game application were determined and a software solution was developed.

The main emphasis in the work is on the use of the integrated Unity environment, which is a powerful tool for the development of high-quality game applications. A detailed analysis of the capabilities, functionality and advantages of this engine has been carried out. The principles of working with scenes, objects, components and scripts in Unity were studied, as well as a comparative analysis with alternative engines was performed.

In the process of developing the game application, modern programming techniques and practices were used, including the use of the C# programming language for the implementation of game logic and components. The developed application was tested in order to identify and eliminate errors and shortcomings.

The qualification work consists of an introduction, three sections, a conclusion, and a list of used sources. The work is completed on 58 pages, contains 40 drawings. The list of bibliographic references includes 40 titles.

Keywords: Unity, source, godot, platformer, engine.

ЗМІСТ

ВСТУП

Однією з найбільш широко розповсюджених індустрій сучасності є галузь розробки комп'ютерних ігор. Швидкі темпи її розвитку та зростання популярності серед користувачів створюють необхідність у постійному удосконаленні інструментів, що використовуються для створення ігрових додатків. У цьому контексті інтегроване середовище Unity виокремляється як потужний інструмент, який надає розробникам можливість створювати вражаючі та захоплюючі ігрові додатки для різних платформ. Робота з Unity дозволяє зосередитися на творчому процесі розробки ігор, максимально використовуючи його готові ресурси та функціонал. Об'ємна функціональність та гнучкість Unity дозволяють реалізувати різноманітні ідеї та створювати ігрові додатки з унікальним геймплеєм та візуальним оформленням.

Метою даної дипломної роботи є розробка функціонального 2D платформера з використанням інтегрованого середовища Unity та дослідження його можливостей та переваг. Головним завданням роботи є створення ігрового додатку, який демонструє різні аспекти роботи з Unity, включаючи захоплюючий геймплей та привабливу графіку. У цьому платформері гравець матиме можливість взаємодіяти з різноманітними предметами та збирати їх у свій інвентар, а також зустрічатиме два різновиди ворогів, що створить додаткові виклики та динаміку у геймплеї.

Актуальність даної роботи обумовлена зростанням попиту на якісні та захоплюючі ігрові додатки у сучасному світі. Споживачі мають все більші вимоги до ігрового досвіду, тому розробка високоякісних ігор стає надзвичайно важливою для успіху на ринку. Використання інтегрованого середовища Unity дозволяє розробникам швидко та ефективно створювати ігри для різних платформ, таких як комп'ютери, мобільні пристрої та консолі. Це забезпечує гнучкість у розробці та максимальне охоплення аудиторії. Розв'язання поставленого завдання має важливе значення для розвитку галузі

комп'ютерних ігор та покращення користувацького досвіду, задовольняючи зростаючі потреби гравців у захоплюючих та якісних ігрових додатках. Таким чином, розробка ігрового додатку з використанням інтегрованого середовища Unity є актуальним завданням, яке відкриває безліч можливостей для створення захоплюючих, високоякісних ігор та сприяє розвитку галузі комп'ютерних ігор в цілому.

На сьогоднішній день існують різні підходи до розробки ігрових додатків, включаючи використання інших інтегрованих середовищ та програмних платформ. Однак, серед цих підходів Unity займає провідну позицію у галузі розробки ігор. Unity є надійним інструментом завдяки своїм зручним інструментам, високій продуктивності та можливостям розробки для різних платформ. Попередні дослідження показують, що Unity має широкий функціонал, включаючи потужність у створенні графіки, фізики та штучного інтелекту, та надає розробникам гнучкість у створенні різноманітних ігрових додатків. Багата екосистема Unity, включаючи магазин активів та готових рішень, також допомагає їм прискорити процес створення гри та полегшити спільну роботу в команді. Таким чином, обрання Unity для розробки 2D платформера є обґрунтованим та перспективним рішенням.

Реалізація даного проекту не тільки сприятиме розвитку навичок роботи з інтегрованим середовищем Unity, але й відкриє можливості для дослідження та вдосконалення ігрових механік. Здійснення даного проекту дозволить набути практичного досвіду в розробці захоплюючих ігрових додатків та вирішенні технічних викликів, що виникають під час розробки подібних ігор. Також, успішна реалізація ігрового додатку може сприяти привабливості проекту для користувачів та сприяти популяризації розробника на ринку ігрової індустрії.

Загалом, дана дипломна робота присвячена розробці ігрового додатку за допомогою інтегрованого середовища Unity. Вона має на меті дослідити можливості та переваги цього середовища, а також створити функціональний ігровий додаток, що демонструє різні аспекти роботи з Unity. Вдале

виконання цієї роботи може мати значний вплив на розвиток галузі комп'ютерних ігор та надати користувачам нові захоплюючі ігрові враження. Подальше вдосконалення розробленого додатку може сприяти його комерційному використанню та привертати увагу інвесторів в ігровій індустрії.

РОЗДІЛ 1
АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз ринку комп'ютерних ігор

У сучасному світі комп'ютерні ігри є однією з найпопулярніших форм розваги, що залучає мільйони гравців по всьому світу. Розвиток технологій і швидкий прогрес у галузі програмного забезпечення призвели до зростання якості графіки, реалістичності, та геймплею в комп'ютерних іграх. Це створило потужну та конкурентну галузь, яка продовжує швидко розвиватися та привертати увагу широкої аудиторії.

З моменту з'явлення перших комп'ютерних ігор у 1950-х роках, галузь ігрової індустрії зазнала значних змін та зростання. Сьогодні ігрова індустрія є однією з найбільш дохідних та впливових галузей в сфері розваг.

Аналізуючи ринок комп'ютерних ігор, важливо враховувати різноманітні його аспекти:

а) Тенденції та розвиток ринку комп'ютерних ігор

Виявлення актуальних тенденцій у галузі розвитку ігрової індустрії є ключовим етапом аналізу ринку комп'ютерних ігор. Завдяки швидкому прогресу технологій і постійному розвитку галузі, з'являються нові технологічні можливості, які впливають на способи створення та використання комп'ютерних ігор. Ось деякі актуальні тенденції, які варто врахувати:

1. Використання віртуальної та доповненої реальності: Віртуальна реальність (VR) та доповнена реальність (AR) стають все більш популярними в галузі комп'ютерних ігор. VR дозволяє гравцям повністю зануритися в віртуальний світ, створюючи іммерсивний геймплей. AR, з свого боку, дозволяє додавати віртуальні об'єкти до реального оточення, розширюючи можливості взаємодії з грою.

2. Штучний інтелект (AI): Використання штучного інтелекту в комп'ютерних іграх стає все поширенішим. AI використовується для створення розумних ігрових персонажів, опонентів з адаптивним ігровим стилем та складними системами поведінки. Також, штучний інтелект може використовуватися для автоматичного генерування контенту та управління грою залежно від дій гравця.

3. Мультиплеєр та онлайн-ігри: Зростання швидкості Інтернету та популярність онлайн-громадських платформ дозволяють гравцям з усього світу з'єднуватися та взаємодіяти в ігрових світах. Мультиплеєрні ігри стають все популярнішими, забезпечуючи можливість спільної гри з друзями або випадковими гравцями з усього світу.

4. Ігри у хмарі: За допомогою технологій обчислення у хмарі, гравці можуть стрімити ігри на свої пристрої без необхідності масштабного обладнання. Це дозволяє грати високоякісні ігри на різних пристроях, таких як смартфони, планшети або навіть телевізори без потреби в потужних обчислювальних системах.

Врахування цих актуальних тенденцій у галузі розвитку ігрової індустрії є важливим для успішного розробки ігрового додатку за допомогою інтегрованого середовища Unity. Розуміння новітніх технологій та споживчих преференцій дозволить мені використовувати актуальні рішення та створити конкурентоспроможний продукт на ринку комп'ютерних ігор.

б) Огляд сучасних платформ та пристроїв для гри

Для успішного аналізу ринку комп'ютерних ігор, необхідно розглянути різні платформи та пристрої, на яких грають користувачі. Ось огляд деяких сучасних платформ та пристроїв для гри:

Персональні комп'ютери є однією з найпопулярніших платформ для гри, забезпечуючи широкий спектр можливостей і гнучкість для гравців. Ось детальніша інформація про платформу ПК:

1. Апаратне забезпечення: ПК мають значну перевагу в апаратному забезпеченні, оскільки їх можна легко оновлювати та модернізувати. Гравці

можуть встановлювати потужні графічні карти, процесори, оперативну пам'ять та сховища для отримання максимальної продуктивності та графічної якості. Це дозволяє грати в вимогливі ігри з високою деталізацією та роздільною здатністю.

2. Операційна система: ПК підтримують різні операційні системи, такі як Windows, macOS і Linux, що дозволяє гравцям вибирати платформу, яка найкраще відповідає їхнім потребам. Кожна з цих операційних систем має велику кількість доступних ігор та ігрових сервісів.

3. Ігрові сервіси та платформи: ПК мають широкий вибір ігрових сервісів та цифрових дистриб'юторів, таких як Steam, Epic Games Store, GOG та інші. Ці платформи дозволяють гравцям купувати, завантажувати та грати в ігри, зберігати геймплейні досягнення, спілкуватися з іншими гравцями та отримувати оновлення та знижки на ігри.

4. Мультимедійні можливості: ПК також надають гравцям доступ до широкого спектру мультимедійних можливостей. Вони можуть відтворювати відео, музику та потокове відео, дозволяючи гравцям не тільки грати, але й насолоджуватися іншими розвагами на своєму ПК.

5. Моддінг: ПК є популярною платформою для моддінгу, що означає зміну або розширення геймплею, візуальних ефектів, мап та багато іншого. Гравці можуть використовувати модифікації, створені іншими гравцями або самостійно створювати власні моди, щоб налаштувати гру під свої потреби та унікальний досвід.

Загалом, ПК є важливою платформою для геймерів, завдяки своїй гнучкості, можливостям модернізації та широкому вибору ігор та сервісів. Він надає гравцям можливість насолоджуватися високоякісним геймплеєм, налаштувати гру під свої потреби та використовувати різноманітні мультимедійні функції.

Ігрові консолі, такі як PlayStation, Xbox і Nintendo Switch, є популярними пристроями для гри, які спеціально розроблені для відтворення високоякісних ігрових вражень.

1. Апаратне забезпечення: Ігрові консолі мають свою власну апаратну платформу, що дозволяє оптимізувати графіку та продуктивність ігор. Вони мають спеціалізований графічний процесор, процесор центрального оброблення, оперативну пам'ять та інші компоненти, які забезпечують швидку та ефективну роботу консолі.

2. Ексклюзивні ігри: Ігрові консолі часто мають свої ексклюзивні ігри, які доступні тільки на даній платформі. Це можуть бути популярні франшизи, ексклюзивні розробки від великих студій або незалежних ігор, які створені спеціально для даної консолі. Ексклюзивні ігри є однією з причин, чому гравці вибирають конкретну ігрову консоль.

Загалом, ігрові консолі є важливою платформою для геймерів, забезпечуючи високоякісні ігрові враження та ексклюзивні ігри.

Мобільні пристрої, такі як смартфони та планшети, стали все більш популярними платформами для гри. Ось детальніша інформація про мобільні пристрої:

1. Портативність: Однією з найбільших переваг мобільних пристроїв є їх портативність. Вони легкі, компактні та можуть бути завжди з собою. Це дозволяє гравцям насолоджуватися іграми в будь-який час та в будь-якому місці, незалежно від наявності комп'ютера або консолі.

2. Сенсорний екран: Мобільні пристрої мають сенсорні екрани, що дозволяють гравцям взаємодіяти з іграми шляхом торкання та свайпів. Це створює інтуїтивний та зручний спосіб керування, особливо для ігор, які розроблені спеціально для мобільних пристроїв.

3. Широкий вибір ігор: Мобільні пристрої мають доступ до великого розмаїття ігор у магазинах додатків, таких як App Store та Google Play. Гравці можуть знайти ігри різних жанрів, від простих пазлів до складних рольових ігор. Це дозволяє знайти ігри, які відповідають індивідуальним вподобанням.

4. Модель бізнесу: Багато мобільних ігор пропонують безкоштовні моделі з внутрішніми покупками (in-app purchases), де гравці можуть безкоштовно завантажити гру, але мати можливість придбати додатковий контент або функціонал у межах гри. Інші ігри пропонують платну модель, де гравці сплачують одноразову плату за повну версію гри. Ці моделі дозволяють розробникам отримувати дохід та гравцям вибирати, як вони бажають спілкуватися з ігровим контентом.

Мобільні пристрої стали доступними та популярними ігровими платформами, які надають зручність, різноманіття ігор та можливість грати у будь-якому місці та часі. Вони привертають як ветеранів ігор, так і новачків, завдяки своїм унікальним можливостям та широкому вибору геймплею.

Віртуальна реальність (VR) та доповнена реальність (AR): VR та AR є відносно новими платформами для ігор, які надають іммерсивний ігровий досвід. VR пристрої, такі як Oculus Rift, HTC Vive та PlayStation VR, дозволяють гравцям погрузитися у віртуальний світ та контролювати гру за допомогою рухів. AR пристрої, такі як HoloLens та мобільні пристрої з AR можливостями, додають ігрові елементи до реального світу. Ці технології надають нові можливості для розробки захоплюючих ігрових вражень.

Врахування цих різних платформ та пристроїв під час розробки ігрового додатку дозволить забезпечити максимальну доступність та задоволення для різних категорій гравців. Також важливо врахувати можливі обмеження апаратних характеристик та функціональних можливостей кожної платформи для досягнення оптимальної продуктивності та задоволення користувачів.

в) Оцінка обсягу ринку комп'ютерних ігор у світі та в окремих регіонах

Оцінка обсягу ринку комп'ютерних ігор є важливим етапом аналізу предметної області, оскільки допомагає зрозуміти масштаби та потенціал цього сектора. Ось огляд оцінок обсягу ринку комп'ютерних ігор у світі та в окремих регіонах:

1. Глобальний ринок комп'ютерних ігор: За останні декілька років глобальний ринок комп'ютерних ігор зазнав значного зростання. За даними дослідницької компанії, такої як Newzoo, загальний обсяг світового ринку комп'ютерних ігор у 2023 році оцінювався близько $ 184.4 млрд. Прогнозується, що цей обсяг буде зростати, зокрема за рахунок зростання мобільних ігор та розширення ринку віртуальної реальності.

2. Регіональні ринки: Ринок комп'ютерних ігор також має регіональні особливості. Одним з найбільших ринків є Північна Америка, де Сполучені Штати та Канада мають значний вплив. Європейський ринок також великий і розгалужений, з такими країнами, як Велика Британія, Німеччина та Франція, як ключовими гравцями. Азійські ринки, зокрема Китай, Японія та Південна Корея, також відомі своїм великим розміром та активністю.

3. Мобільні ігри: Мобільні ігри стали однією з найбільш швидкозростаючих категорій на ринку комп'ютерних ігор. За даними дослідників, у 2023 році глобальний обсяг ринку мобільних ігор склав понад $ 92.2 млрд. Великий попит на мобільні ігри спостерігається у країнах Азії, зокрема в Китаї та Японії, а також у США та Західній Європі.

Оцінка обсягу ринку комп'ютерних ігор у світі та в окремих регіонах допомагає розуміти розмір ринку, його потенціал для росту та можливості для розробки та розміщення ігрових продуктів. Врахування цих даних дозволяє зорієнтуватися на цільові аудиторії, розробити маркетингові стратегії та визначити конкурентні переваги свого ігрового додатку.

г) Монетизація та моделі бізнесу

Монетизація ігрових додатків є важливим аспектом розробки ігор. Розглянемо деякі з найпоширеніших моделей бізнесу та способи отримання прибутку в ігровій індустрії.

Платна. Ця модель передбачає продаж гри за певну ціну. Гравці сплачують одноразову суму, щоб отримати доступ до повної версії гри. Цей

підхід часто використовується для великих ігрових проектів, які пропонують високу якість геймплею та унікальний контент.

Безкоштовна. У цій моделі гра доступна для завантаження та гри безкоштовно. Однак, розробники отримують прибуток з внутрішніх покупок або додаткових преміум-елементів в грі. Це можуть бути косметичні предмети, розблоковані рівні, платні послуги або вимкнення реклами, яка відображається в грі.

Підписка. Ця модель полягає в тому, що гравець платить певну суму за певний період (наприклад, щомісячну або щорічну підписку) для отримання доступу до вибраного набору ігор або послуг. Цей підхід стає все популярнішим, оскільки він забезпечує стабільний потік прибутку для розробників.

Рекламна модель. У цій моделі гра безкоштовна для гравців, але містить рекламу. Рекламні банери, відеоролики або рекламні пропозиції можуть відображатися під час геймплею, між рівнями або як нагороди за досягнення в грі. Розробники отримують прибуток від рекламних показів або кліків на рекламні блоки.

Додатковий контент і доповнення. Цей підхід полягає в продажу додаткового контенту або доповнень до гри. Розробники можуть пропонувати нові рівні, персонажів, костюми, інструменти, мапи або інші елементи, які розширюють геймплей.

Комбінація цих моделей може бути також використана для максимізації прибутку та задоволення потреб різних груп гравців.

1.2. Огляд ігрових категорій та режимів

В наші дні кожен гравець має можливість знайти ігрову платформу, яка відповідає його фінансовим можливостям і особистому смаку. У цьому розділі буде надано багато класифікацій ігор, щоб краще розібратися у їх характеристиках та параметрах.

а) Категорії

Ігрові категорії - це розділення ігор на різні групи в залежності від їхніх основних характеристик, механік геймплею та жанрових особливостей. Категорії допомагають краще розуміти та класифікувати різноманітність ігор, що існують у світі геймінгу. Кожна категорія має свої унікальні особливості, які привертають різні типи гравців та створюють різні ігрові враження. Далі буде розглянуто деякі з найпоширеніших ігрових категорій та їхні ключові аспекти.

Екшн. Ігри жанру екшн зосереджені на швидкому та захоплюючому геймплеї, де гравець контролює головного героя та здійснює різні дії, такі як бійка, стрілянина, пересування тощо. Це можуть бути шутери від першої або третьої особи, файтинги, платформери та інші ігри, де динаміка та активність важливі.

Пригодницькі. Пригодницькі ігри надають гравцям можливість підкорювати вигадані світи, досліджувати та розв'язувати головоломки. Це можуть бути графічні пригоди, рольові ігри з елементами відкритого світу, та інші формати, які покликані розширити історію та геймплей.

Стратегічні. Ігри жанру стратегічних ігор передбачають планування, управління ресурсами та вирішення складних завдань. Гравцям потрібно розвивати стратегію, керувати військами або ресурсами, будувати та управляти базами чи цілими цивілізаціями. Стратегічні ігри можуть бути в реальному часі (RTS), пошаговими, глобальними чи фокусуватися на конкретних аспектах, таких як воєнні стратегії, економічні симулятори тощо.

Рольові (RPG). Ігри жанру RPG дозволяють гравцям відігравати роль вигаданого персонажа, просуватися по сюжету та вирішувати завдання. Гравець може розвивати свого персонажа, отримувати нові вміння, знаряддя, брати участь у боях та взаємодіяти з ігровим світом. RPG-ігри можуть бути в стилі фентезі, науково-фантастичні, постапокаліптичні та інші.

Головоломки. Головоломкові ігри зосереджені на вирішенні логічних задач та головоломок. Гравці повинні застосовувати креативність та

аналітичні навички для досягнення поставленої мети. Це можуть бути різноманітні головоломки, кросворди, гра-головоломки, гра на знаходження прихованих об'єктів та багато іншого.

Спорт. Жанр ігор який відображає віртуальне відтворення різних видів спорту, де гравець може відчути екшн та дух змагань. У спортивних іграх можуть відтворюватися різні види спорту, такі як футбол, баскетбол, хокей, теніс, гольф, бокс та багато інших. Гравці можуть займатися індивідуальними змаганнями або брати участь у командних турнірах, де головна мета - перемога та досягнення високих результатів.

Симулятори. Цей жанр відображає віртуальні середовища, що імітують реальні об'єкти, ситуації або діяльності. Вони надають гравцям можливість відчути себе у ролі справжнього оператора, пілота, фермера, підприємця та багатьох інших професій або ситуацій. Симулятори часто зосереджені на реалістичному відтворенні фізики, управління та взаємодії з оточенням. Вони можуть пропонувати докладну модель поведінки об'єктів, реалістичні графічні ефекти та звукове супроводження, що додають глибини та автентичності грі.

Карткові. Жанр "Карткові ігри" включає в себе ігри, що граються з використанням спеціальних карткових наборів. Цей жанр відомий своєю стратегічністю, тактичними вирішеннями та елементами удачі. У карткових іграх гравці використовують набори карт, які можуть мати різні значення, характеристики та спеціальні властивості. Кожна карта може представляти різних персонажів, заклинання, ресурси або інші геймплейні елементи.

Платформер. Це жанр комп'ютерних ігор, в якому основним елементом геймплею є переміщення головного персонажа по різних платформах та перешкодам. Головне завдання гравця полягає в успішному подоланні рівнів, які часто включають стрибки, біг, лазіння та розв'язування головоломок. У платформерах зазвичай існує горизонтальна або вертикальна прокрутка рівня, де гравець взаємодіє з платформами, сходинками, рухомими об'єктами та іншими перешкодами. Важливими елементами жанру є точність та вміння

контролювати персонажа, оскільки неправильні рухи можуть призвести до падіння або зіткнення з ворогами, що призводить до втрати життя або зменшення прогресу. Платформери можуть мати різні стилі та настрої, включаючи пригодницькі, фантастичні, науково-фантастичні та комедійні. Вони також можуть включати різноманітні геймплейні елементи, такі як збирання предметів, боротьба з ворогами, вирішення головоломок та досягнення певних цілей. Платформери є популярним жанром серед гравців будь-якого віку. Вони надають можливість насолоджуватися швидким та веселим геймплеєм, вимагають від гравця швидкісного реагування, координації рухів та стратегічного мислення.

б) Режими

У сучасних іграх існують різні режими гри, які надають гравцям різноманітні способи взаємодії з віртуальним світом. Найпоширенішими режимами є:

1. Одиночний режим: Цей режим дозволяє гравцеві грати самостійно без участі інших гравців. Гравець може насолоджуватися сюжетом гри, виконувати завдання, розв'язувати головоломки або досліджувати віртуальний світ власним темпом.

2. Мультиплеєрний режим: Цей режим дозволяє гравцям грати разом з іншими гравцями через Інтернет або локальну мережу. Вони можуть змагатися один з одним у командних або індивідуальних битвах, співпрацювати для досягнення спільних цілей або взаємодіяти у віртуальному світі.

3. Кооперативний режим: Цей режим дозволяє гравцям об'єднатися в команди і спільно виконувати завдання або протистояти ворогам. Гравці можуть обмінюватися ресурсами, доповнювати один одного в навичках або використовувати стратегію для досягнення успіху.

1.3. Формування задачі кваліфікаційної роботи

Перш за все, необхідно розглянути ігрові рушії і їх можливості, переглянути їхній інтерфейс та приклади ігор розроблених з використанням цих рушіїв.

Основною метою даної дипломної роботи повинно бути створення 2D платформера - гри, яка надає гравцеві цікавий та захоплюючий геймплей, має велику базу гравців, а також забезпечує візуальну привабливість та високу якість виконання.

1.4. Висновки до першого розділу

В першому розділі дипломної роботи розкриваються важливі аспекти, пов'язані з розробкою ігрового додатку. У цьому розділі було проведено аналіз ринку комп'ютерних ігор, оглянуто сучасні платформи та пристрої для гри, розглянуто різні ігрові категорії, а також розглянуто питання монетизації та моделей бізнесу.

Аналізуючи ринок комп'ютерних ігор, було виявлено, що ігрова індустрія є динамічною та постійно розвивається. Актуальні тенденції, такі як використання віртуальної та доповненої реальності, штучного інтелекту, мультиплеєра та ігор у хмарі, впливають на напрямок розвитку ігрової індустрії та створюють нові можливості для ігрових розробників.

Огляд сучасних платформ та пристроїв для гри показав, що персональні комп'ютери, ігрові консолі та мобільні пристрої є популярними платформами серед гравців. Кожна з цих платформ має свої особливості та переваги, які варто враховувати при розробці ігрового додатку.

Детальний огляд ігрових категорій дозволив розібратися в різноманітності ігрових жанрів та стилів. Від спортивних ігор, що передають атмосферу спортивних змагань, до симуляторів, що дозволяють користувачам відчути себе в ролі різних професій або ситуацій.

Монетизація та моделі бізнесу є важливими аспектами розробки ігрових додатків. Розуміння різних моделей бізнесу, таких як продаж гри, фрі-ту-плей, мікротранзакції та реклама, допомагає розробникам ефективно планувати стратегію монетизації своїх ігор та забезпечувати стійкий дохід.

У підсумку, розділ надав обґрунтовану базу для подальшої розробки ігрового додатку. Аналіз ринку, платформ, жанрів і моделей бізнесу дозволив зрозуміти потреби гравців та визначити цілісний підхід до розробки гри. Завдання дослідження було поставлено з урахуванням цих факторів, а відповідність їх виконання буде визначати успіх всього проекту.

РОЗДІЛ 2
МЕТОДИ ВИРІШЕННЯ ЗАДАЧ ПОСТАВЛЕНИХ ВИКОНАННЯМ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1. Загальна інформація про ігровий рушій

Ігрові рушії (Game engines) є ключовими інструментами у розробці комп'ютерних ігор. Вони надають потужність та гнучкість для створення, візуалізації та функціонування ігрових проектів. У цьому розділі буде розглянуто загальні відомості про ігрові рушії, їх функції та можливості, а також проведений аналіз різних рушіїв, щоб визначити найбільш підходящий для розробки.

Ігрові рушії є комплексними інтегрованими середовищами, які надають розробникам зручний набір інструментів для створення ігрових проектів. Вони включають в себе набір різноманітних функцій і компонентів, таких як рендеринг графіки, фізика, звук, штучний інтелект, анімація, управління користувачем та інші. Ігрові рушії дозволяють розробникам зосередитися на творчості та геймплеї, не витрачаючи багато часу на низькорівневу оптимізацію та розробку базових функцій.

На ринку існує безліч різних ігрових рушіїв, кожен з яких має свої переваги та обмеження. Деякі з найпопулярніших ігрових рушіїв включають Unity, Unreal Engine, CryEngine, Godot, GameMaker і багато інших. Кожен рушій має свою власну специфіку, екосистему та підхід до розробки ігор.

Аналіз ігрових рушіїв полягає в дослідженні їх можливостей, продуктивності, підтримки платформ, доступності інструментів та ресурсів для розробки, а також вартості ліцензування. Залежно від потреб і вимог проекту, буде вирішено, який рушій найбільше відповідає вимогам та допоможе досягти мети - створення ігрового додатку.

2.2. Сучасні ігрові рушії, їх плюси та мінуси

В даний час на ринку існує велика кількість різних ігрових рушіїв. У пункті, 2.2, буде розглянуто найбільш відомі та популярні глобальні рушії, що використовуються в галузі розробки ігор.

а) Unity

Unity - це інноваційний ігровий рушій, створений американською компанією Unity Technologies. Випущений в 2005 році, цей рушій продовжує активно розвиватися. Він відомий не лише як простий ігровий рушій, але і як кросплатформене багатофункціональне середовище розробки ігор.

Unity надає розробникам можливість створювати різноманітні ігри та програми, які можуть бути запущені на близько 25 різних ігрових платформах, зокрема комп'ютерах, ігрових консолях та мобільних телефонах. Цей рушій виступає в ролі повноцінного середовища для розробки ігор, оскільки поєднує різноманітні програмні засоби, необхідні для розробки програмного забезпечення - відладчик, дебагер і багато інших.

Особливістю редактора Unity є простий та зрозумілий інтерфейс для користувача, який підтримує технологію Drag & Drop. Це дозволяє розробникам зручно маніпулювати об'єктами та ресурсами, перетягуючи їх у відповідні області редактора. Такий підхід спрощує процес розробки та забезпечує більшу продуктивність.

Unity займає лідируючу позицію в галузі розробки ігор завдяки своїй широкій функціональності, кросплатформеності та зручному інтерфейсу. Він є популярним вибором для розробників ігор будь-якого рівня досвіду.

Рисунок 2.1 – Інтерфейс редактора Unity

У рушії Unity для написання скриптів розробники можуть використовувати мову програмування C#. Значна перевага полягає в тому, що ця мова є потужною та широко використовується в індустрії програмування. Починаючи з версії 2017.1, Unity перестав підтримку мови програмування UnityScript. Крім того, до версії 5 рушій підтримував мову програмування Boo.

Unity є популярним серед розробників завдяки своїй гнучкості і можливості створювати різноманітні додатки, які відповідають потребам різних гравців, платформ і жанрів. Він отримав широке визнання як серед незалежних розробників, так і серед професійних компаній. Багато успішних ігор і додатків було створено з використанням Unity, що свідчить про його популярність та ефективність в галузі розробки ігор.

Плюси Unity:

1. Кросплатформеність: Unity дозволяє розробляти ігри для різних платформ, таких як ПК, консолі, мобільні пристрої та віртуальна реальність.

Це забезпечує широку аудиторію та можливість досягти більшого кола гравців.

2. Широкий набір інструментів: Unity надає велику кількість готових ресурсів, бібліотек, компонентів та плагінів, що допомагають розробникам прискорити процес розробки ігор.

3. Спільнота розробників: Unity має велику активну спільноту розробників, яка надає підтримку, допомогу та обмін знаннями. Це дозволяє швидко розв'язувати проблеми та знаходити рішення.

Мінуси Unity:

1. Вартість: Використання деяких продуктів та функцій Unity може бути платним. Ліцензування Unity може бути дорогим для незалежних розробників або стартапів з обмеженими фінансовими ресурсами.

2. Вимоги до продуктивності: Розробка деяких складних ігрових проектів у Unity може вимагати великих обчислювальних потужностей та оптимізації для досягнення плавності та високої продуктивності.

3. Навчання та крутий початок: Unity має певну крутість криву навчання, особливо для новачків у розробці ігор. Відомості про використання інструментів та програмування в Unity можуть бути потрібні для ефективного використання цього рушія.

Не зважаючи на ці мінуси, Unity залишається популярним інструментом для розробки ігор завдяки своїй широкій функціональності, підтримці різних платформ та активній спільноті розробників.

Unity пропонує кілька варіантів ліцензування, які відповідають потребам різних типів розробників і компаній. Основні типи ліцензій в Unity включають:

1. Personal (Особиста ліцензія): Це безкоштовна ліцензія, яка дозволяє незалежним розробникам і початківцям використовувати Unity для

особистих, некомерційних проектів. З цією ліцензією можна створювати ігри та додатки для різних платформ. Однак, якщо доходи розробника перевищують 100 тис. доларів в рік, розробник гри мусить придбати професійну версію.

2. Plus (Ліцензія "Плюс"): Ця ліцензія призначена для незалежних розробників з обмеженим бюджетом. Вона включає розширені можливості, такі як підтримка певних платформ, покращена аналітика та підтримка користувачів.

3. Pro (Професійна ліцензія): Це комерційна ліцензія, яка надає повний доступ до всіх функцій та можливостей Unity. Вона розрахована на професійних розробників та компанії, які створюють ігри та додатки для комерційної експлуатації. Ліцензія Pro надає додаткові інструменти, підтримку платформ та інші переваги.

Приклади ігор розроблених на Unity.

Рисунок 2.2 – Приклад гри на Unity. Cuphead

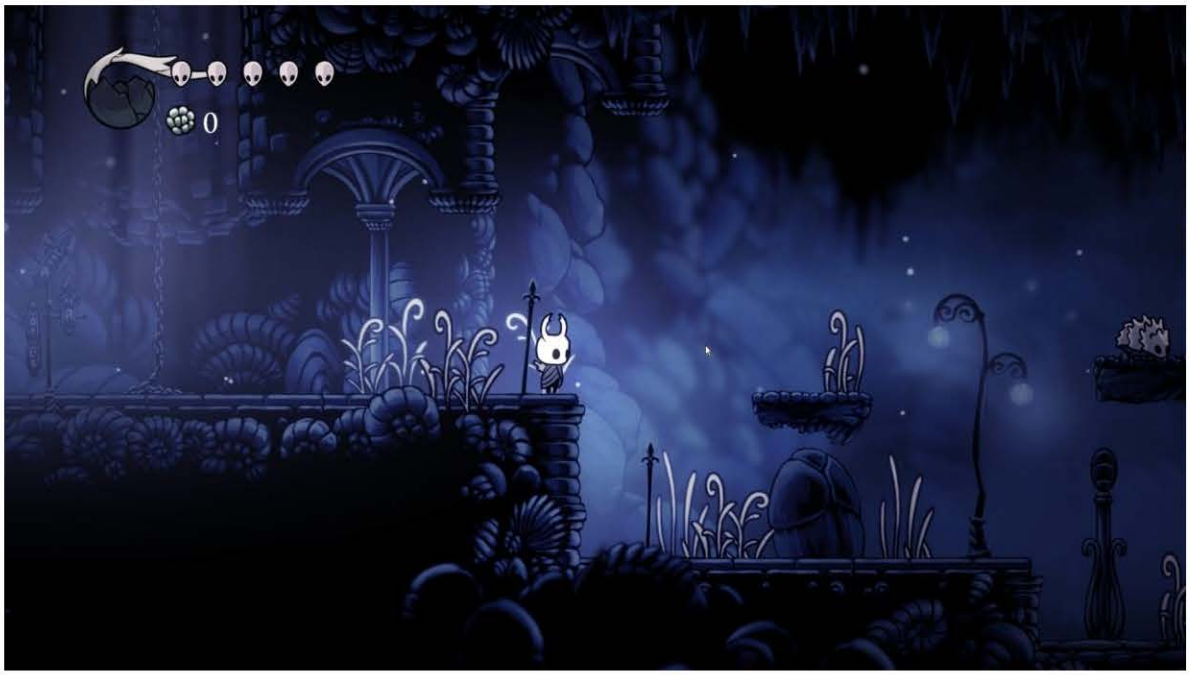

Рисунок 2.3 – Приклад гри на Unity. Hollow Knight

Рисунок 2.4 – Приклад гри на Unity. Rust

б) CryEngine V

CryEngine V - це потужний ігровий рушій, розроблений німецькою компанією Crytek. Цей рушій був випущений в 2016 році і продовжує активно розвиватись до сьогоднішнього дня.

Для програмування в CryEngine V використовується мова програмування C++. Ця мова є широко використовуваною в індустрії розробки ігор і надає можливість створювати потужні та ефективні ігрові програми.

CryEngine V відомий своїми високоякісними графічними можливостями, потужними інструментами розробки та широкими можливостями розробки ігор для різних платформ, включаючи ПК, консолі та віртуальну реальність. Він використовується як незалежними розробниками, так і професійними компаніями для створення захоплюючих ігрових проектів.

Крім того, у CryEngine V є також інша цікава технологія під назвою Flowgraph. Flowgraph - це візуальний інструмент для програмування, що дозволяє розробникам створювати складні ігрові логіки та взаємодії без необхідності писати код. За допомогою Flowgraph, розробники можуть визначати поведінку об'єктів, управляти подіями та взаємодіями між різними елементами гри, що спрощує розробку та відлагодження.

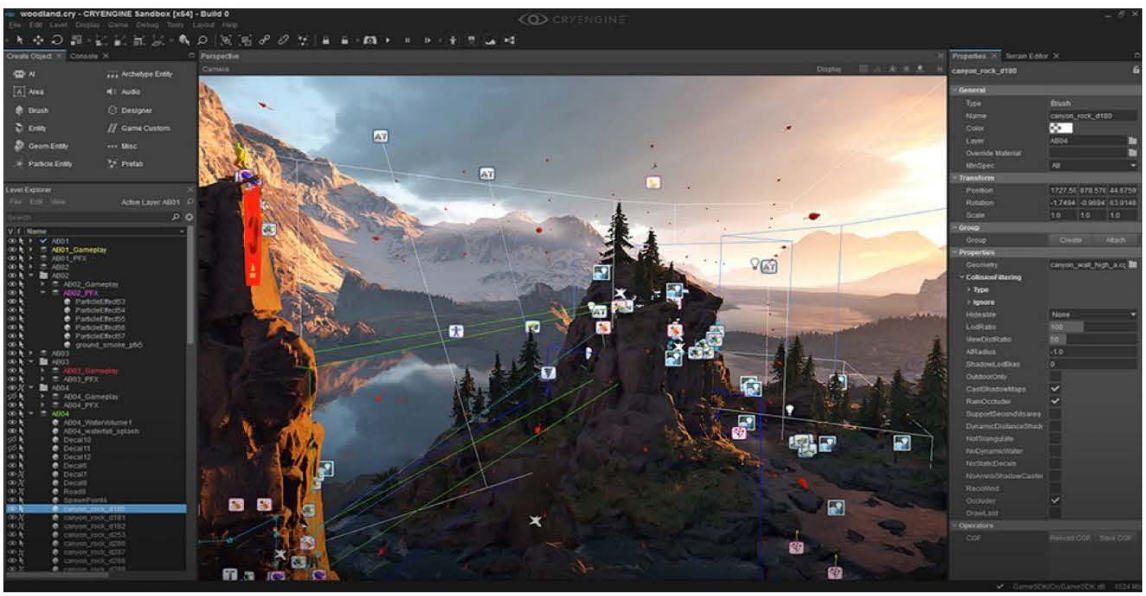

Рисунок 2.5 – Інтерфейс редактора CryEngine V

CryEngine V пропонує кілька варіантів ліцензування, які відповідають потребам різних типів розробників і компаній.

1. Ліцензія для спільноти розробників (CryEngine Community Edition): Це безкоштовна ліцензія, яка надається розробникам з метою підтримки спільноти та стимулювання творчості. Розробники можуть використовувати CryEngine V для створення своїх проектів безкоштовно, але з обмеженими можливостями доступу до певних функцій та інструментів.

2. Ліцензія для комерційних проектів (CryEngine Royalty-Based Model): Цей тип ліцензування передбачає сплату роялті (відсотка від прибутку) після досягнення певного рівня комерційного успіху проекту. Зазвичай, угода про ліцензування укладається безпосередньо з командою Crytek, де визначаються умови роялті та інші деталі.

3. Ліцензія для великих партнерів (CryEngine Source Code Access): Цей тип ліцензії надається великим компаніям та партнерам, які отримують повний доступ до вихідного коду CryEngine V. Це дозволяє здійснювати більш глибокі модифікації та налаштування рушія, що відповідає конкретним потребам проекту.

Плюси CryEngine V:

1. Графічна потужність: CryEngine V володіє вражаючими графічними можливостями, здатними створювати деталізовані світи, реалістичне освітлення та спеціальні ефекти.

2. Фізика та анімація: Рушій має вбудовану підтримку реалістичної фізики та високоякісної анімації, що дозволяє розробникам створювати живі та взаємодіючі ігрові світи.

3. Розширюваність: CryEngine V надає можливості розширення та налаштування рушія за допомогою модульної архітектури, що дозволяє розробникам використовувати власні інструменти та функціонал.

4. Підтримка віртуальної реальності: CryEngine V має вбудовану підтримку віртуальної реальності, що дозволяє розробникам створювати ігри та додатки для VR-платформ.

Мінуси CryEngine V:

5. Складність в освоєнні: Рушій має круту криву навчання, і його освоєння може вимагати часу та зусиль, особливо для новачків.

6. Обмежена документація: Порівняно з деякими іншими рушіями, CryEngine V може мати обмежену кількість документації та ресурсів, що можуть ускладнити процес розробки.

7. Обмежена підтримка платформ: У порівнянні з іншими рушіями, CryEngine V може мати обмежену підтримку платформ, що може обмежити доступність гри на різних пристроях.

8. Вартість: Для комерційних проектів, ліцензування та використання CryEngine V може вимагати фінансових витрат, особливо при використанні додаткових платних сервісів або інструментів.

Приклади ігор розроблених на CryEngine V.

Рисунок 2.6 – Приклад гри на CryEngine V. Ryse: Son Of Rome

Рисунок 2.7 – Приклад гри на CryEngine V. Prey

Рисунок 2.8 – Приклад гри на CryEngine V. Star Citizen

в) Unreal Engine 4

Unreal Engine 4 (UE4) - ігровий рушій, що був створений компанією Epic Games. Число "4" в назві вказує на поточну версію цього рушія, яка була випущена у 2014 році. Перша версія рушія, відома як Unreal Engine 1, була випущена аж у 1998 році. Подібно до рушія Unity, UE4 дозволяє вам розробляти ігри або додатки для різних ігрових платформ.

UE4 використовує мову програмування C++ як основну мову для створення ігрової логіки та функціональності. C++ є потужною мовою програмування, що дозволяє розробникам мати повний контроль над рушієм і створювати високоякісні ігрові додатки. Використання C++ дозволяє розробникам оптимізувати продуктивність ігри та забезпечувати високу швидкодію.

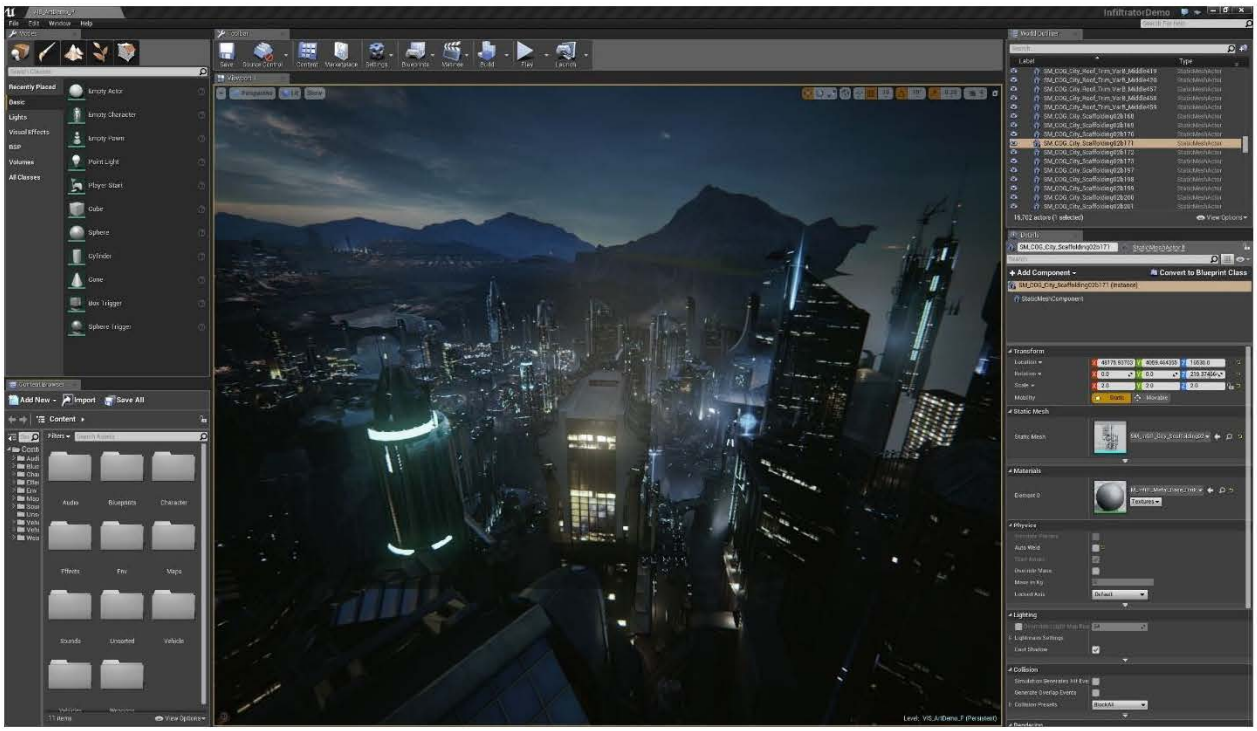

Рисунок 2.9 – Інтерфейс редактора Unreal Engine 4

Дуже корисною технологією у Unreal Engine 4 є Blueprints. Blueprints є візуальною системою скриптування, яка дозволяє розробникам створювати логіку та функціональність своїх ігор без необхідності програмування.

Замість писання коду вручну, розробник може використовувати графічний інтерфейс та з'єднувати блоки разом для створення складних логічних схем.

Однією з переваг Blueprints є їх візуальний підхід, який робить розробку ігор доступною для широкого спектру розробників, незалежно від рівня їхніх навичок програмування. Blueprints дозволяють визначати рух персонажів, взаємодію з об'єктами, обробляти події, створювати штучний інтелект та багато іншого, все це візуально та інтуїтивно.

Проте, важливо зазначити, що при складних проектах або необхідності оптимізації продуктивності, може виникнути потреба в додатковому програмуванні на мові C++. У таких випадках розробники можуть поєднувати використання Blueprints зі спеціалізованим програмуванням, щоб досягти більшої гнучкості та ефективності.

Загалом, Blueprints є потужним інструментом для розробки ігор у Unreal Engine 4, який спрощує процес розробки та дозволяє розробникам втілювати свої ідеї без глибокого володіння програмування.

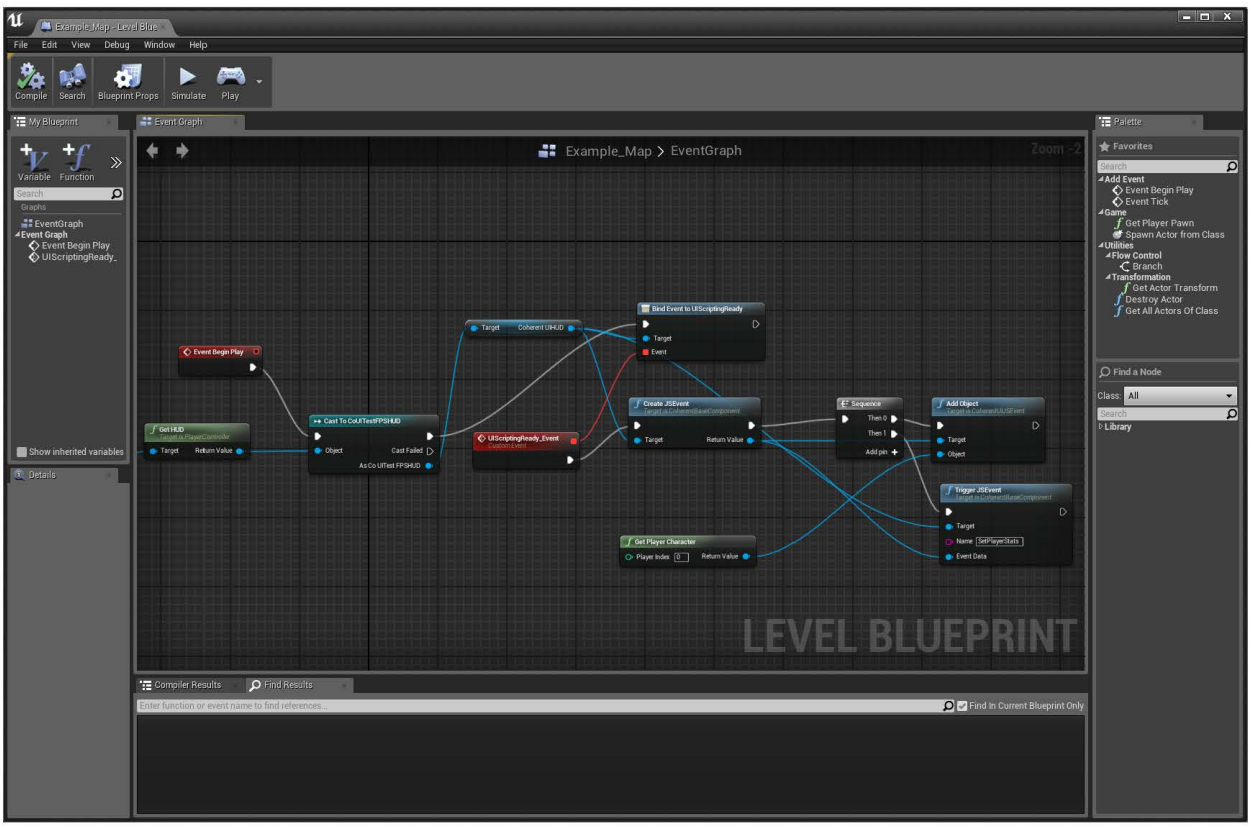

Рисунок 2.10 – Редактор Blueprint

Плюси Unreal Engine 4:

1. Візуальна якість: UE4 надає потужні інструменти для створення вражаючої візуальної якості. Рушій підтримує фотореалістичну графіку, високоякісне освітлення, реалістичну фізику та ефекти.

2. Розширена функціональність: UE4 має велику кількість готових модулів та інструментів, що дозволяють розробникам швидко створювати різноманітний контент. Він має потужну систему частинок, систему фізики, інструменти для роботи зі штучним інтелектом та багато іншого.

3. Кросплатформеність: UE4 підтримує різні ігрові платформи, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність. Це дозволяє розробникам створювати ігри, які можна запускати на різних пристроях без необхідності переписування коду.

4. Спільнота розробників: Unreal Engine має велику активну спільноту розробників, де можна знайти підтримку, навчальні матеріали, готові рішення та додаткові ресурси. Це сприяє швидкому розвитку і спільній роботі над проектами.

Мінуси Unreal Engine 4:

1. Високий поріг входу: Розробникам, які не мають попереднього досвіду у роботі з рушіями, може знадобитися додатковий час та зусилля, щоб оволодіти всіма можливостями та інструментами.

2. Великі вимоги до апаратного забезпечення: UE4 вимагає потужних комп'ютерів для ефективної роботи. Розробка та тестування ігрових проектів у великій мірі залежить від продуктивності обладнання, що може бути фінансово вимогливим.

3. Розмір файлів: Проекти, створені у Unreal Engine 4, можуть мати великий обсяг файлів через високу якість графіки та розширену функціональність. Це може призвести до збільшення часу завантаження та виконання проектів.

4. Ліцензійні обмеження: Unreal Engine має власну ліцензійну угоду, яка може мати певні обмеження та вимоги для комерційного

використання. Розробники повинні ознайомитися з умовами ліцензії та враховувати їх при розробці та розповсюдженні своїх проектів.

У Unreal Engine 4 (UE4) використовується ліцензія, відома як Unreal Engine End User License Agreement. Ця ліцензія регулює умови використання рушія та правила розповсюдження створених на його основі проектів.

Unreal Engine 4 можна використовувати безкоштовно для некомерційних проектів, таких як навчання, особисті проекти або експерименти. Розробники не платять ліцензійний внесок, але відсоток від прибутку може стягуватися, якщо досягнута певна сума прибутку від комерційних проектів.

Приклади ігор розроблених на Unreal Engine 4.

Рисунок 2.11 – Приклад гри на Unreal Engine 4. Final Fantasy 7 Remake

Рисунок 2.12 – Приклад гри на Unreal Engine 4. Star Wars Jedi: Fallen Order

Рисунок 2.13 – Приклад гри на Unreal Engine 4. Sea Of Thieves

г) Godot

Godot - це відкритий ігровий рушій, що був розроблений спільнотою розробників та підтримується некомерційною організацією Godot Engine.

Рушій був випущений вперше у 2014 році, і з тих пір активно розвивається та отримує оновлення.

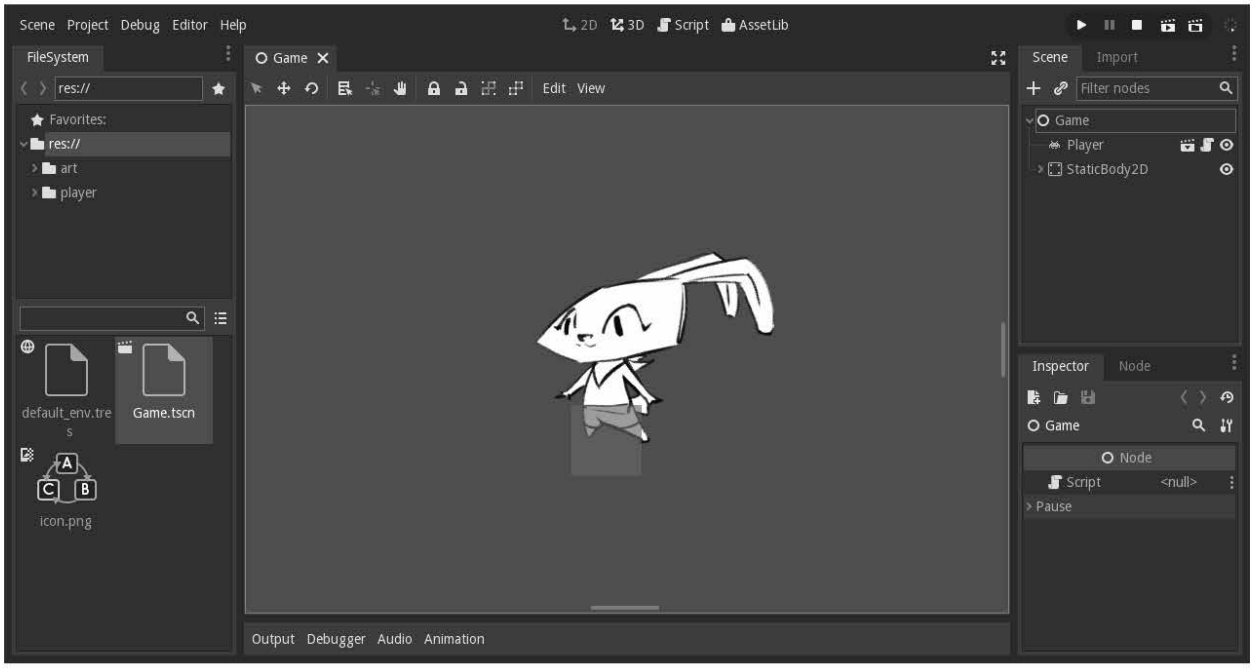

Рисунок 2.14 – Інтерфейс редактора Godot

Godot підтримує кілька мов програмування для розробки ігрової логіки. Основною мовою є GDScript, що є спеціально розробленою скриптовою мовою для Godot. Крім того, рушій підтримує мови, такі як C#, C++ і VisualScript, що дає розробникам різні варіанти для програмування їх проектів.

Також Godot має потужний візуальний редактор, який дозволяє розробникам створювати і налаштовувати сцени, об'єкти, анімацію та інше без необхідності написання коду. Це спрощує процес розробки і дозволяє швидше прототипування ігрових ідей.

Плюси Godot рушія:

1. Відкритий код: Godot є відкритим проектом з відкритим вихідним кодом, що дозволяє розробникам переглядати, змінювати та вдосконалювати сам рушій. Це сприяє співпраці спільноти розробників, покращенню функціональності рушія та розширенню можливостей розробки.

2. Кросплатформовість: Godot підтримує різні платформи, включаючи Windows, macOS, Linux, Android, iOS, HTML5 та інші. Це

дозволяє вам створювати ігри, які можуть працювати на різних пристроях та операційних системах.

3. Легкість використання: Godot має дружній та інтуїтивно зрозумілий інтерфейс, що робить його доступним для розробників з різним рівнем досвіду. Рушій пропонує широкий спектр інструментів і можливостей, що спрощує процес створення ігор.

Мінуси Godot рушія:

1. Обмежені ресурси та документація: У порівнянні з відомішими рушіями, такими як Unity або Unreal Engine, Godot має меншу кількість ресурсів, документацію та підтримку. Це може ускладнити процес навчання та вирішення проблем, особливо для новачків.

2. Менша спільнота: Спільнота розробників Godot є меншою порівняно з більш відомими рушіями. Це може призвести до складнощів у пошуку відповідей на питання або спеціалістів, які б займалися розробкою або підтримкою проекту.

3. Відсутність готових рішень та активних ринкових майданчиків: Оскільки Godot є менш популярним рушієм, може бути складніше знайти готові рішення, плагіни чи активні ринкові майданчики для розробки або продажу ігор.

4. Обмежені можливості 3D-рендерингу: Хоча Godot підтримує 3D-графіку, його можливості 3D-рендерингу можуть бути менш розширеними порівняно з деякими іншими рушіями. Це може бути обмеженням для розробки складних ігор зі складними візуальними ефектами.

Godot розповсюджується безкоштовно, а його ліцензія (MIT License) дозволяє використовувати, модифікувати та поширювати рушій як у комерційних, так і у некомерційних проектах. Це означає, що можна використовувати Godot безкоштовно для створення ігор або додатків, навіть змінювати його за потреби та поширювати створені проекти.

Приклади ігор розроблених на Godot.

Рисунок 2.15 – Приклад гри на Godot. Quest Of Graal

Рисунок 2.16 – Приклад гри на Godot. Monster Outbreak

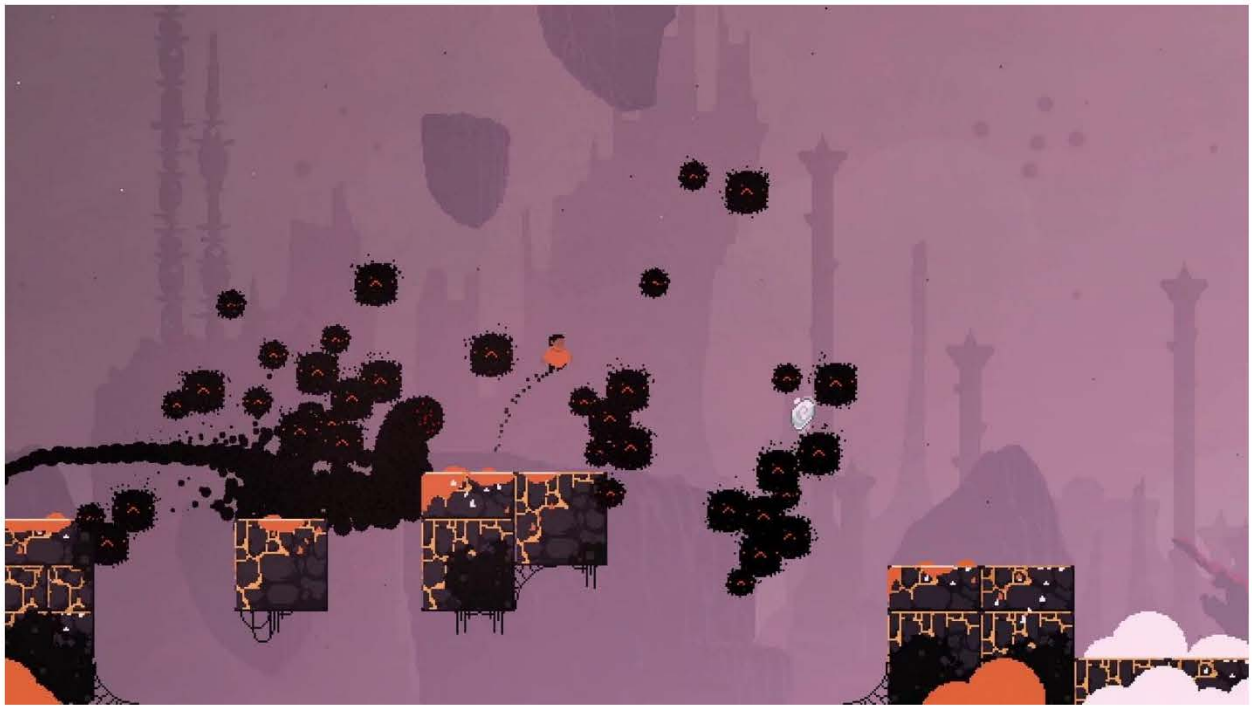

Рисунок 2.17 – Приклад гри на Godot. RUN: The world in-between

д) Source

Source є ігровим рушієм, розробленим компанією Valve Corporation. Він був вперше представлений у 2004 році та став основою для таких відомих ігор, як Half-Life 2, Counter-Strike: Global Offensive, Portal і Team Fortress 2.

Рушій Source підтримує кілька мов програмування, включаючи C++, Lua та Squirrel. Це дає розробникам можливість створювати складні ігрові системи та сценарії.

Особливості рушія Source включають потужну фізичну симуляцію, високоякісну рендерингову систему, розширені можливості штучного інтелекту, підтримку мережевої гри та інструменти для створення ігрових рівнів і модифікацій. Рушій також має вбудовану підтримку для розробки віртуальної реальності.

Загалом, рушій Source відомий своєю гнучкістю, можливостями налаштування та широким спектром використання, що дозволяє розробникам

створювати ігри з різноманітним геймплеєм та вражаючою візуальною якістю.

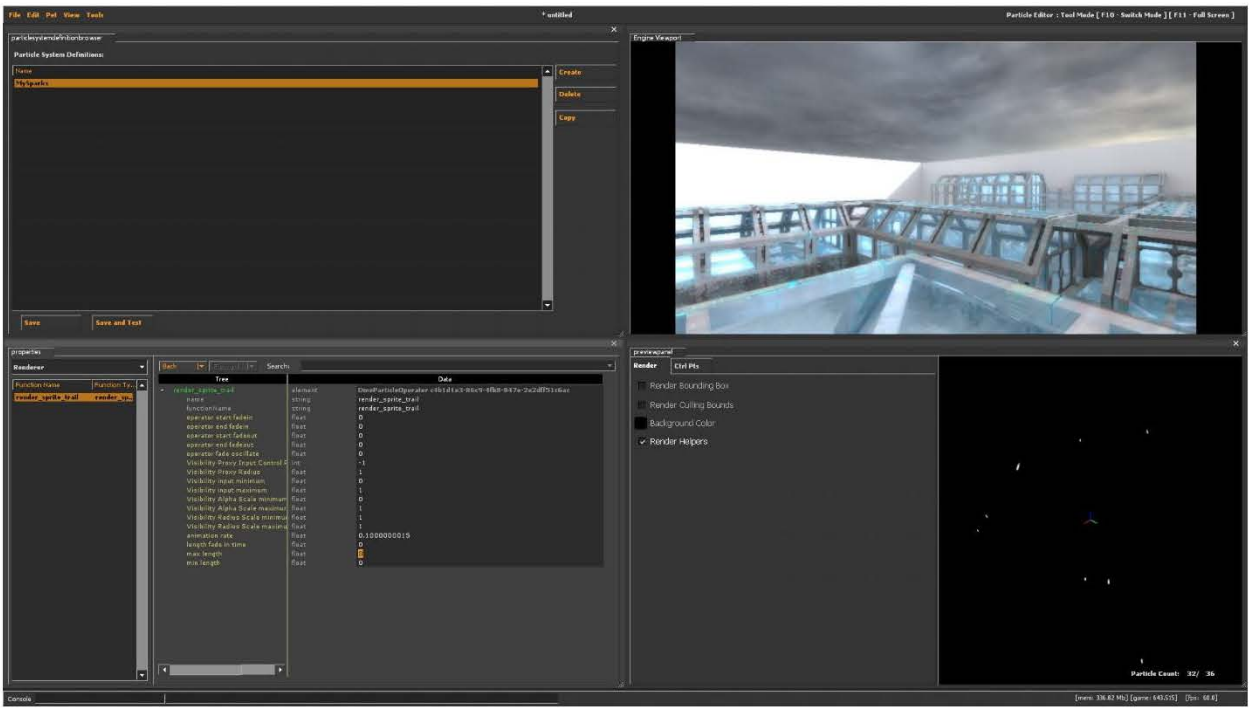

Рисунок 2.18 – Інтерфейс редактора

Source Engine має свої переваги та недоліки, які можуть впливати на вибір рушія для розробки ігор. Ось кілька плюсів та мінусів Source Engine:

Плюси:

1. Потужний двигун: Source Engine відомий своєю високою продуктивністю. Він може обробляти великі рівні з високоякісною графікою та фізикою, забезпечуючи швидкий та плавний геймплей.

2. Багата функціональність: Source Engine має великий набір функцій та інструментів, що дозволяють розробникам створювати різноманітні та складні геймплейні механіки, фізичні ефекти, штучний інтелект та інше.

3. Розширення та модифікації: Source Engine підтримує модифікації, що дозволяє розробникам створювати та випускати власні моди, розширюючи можливості гри та привносячи свої ідеї до спільноти.

Мінуси:

1. Старий код та інструменти: Source Engine розроблений багато років тому, і його код та інструменти можуть виглядати застарілими порівняно з сучасними рушіями. Це може ускладнити розробку та використання новітніх технологій.

2. Висока важкість навчання: Розробка на Source Engine вимагає від розробників високого рівня володіння додатками, такими як Hammer Editor та іншими інструментами. Вивчення цих інструментів може бути доволі складним для новачків.

3. Обмежена платформна підтримка: Source Engine переважно спрямований на ПК та консолі, і не має широкої підтримки мобільних платформ або веб-ігор.

Source Engine має декілька різних ліцензій, включаючи Source SDK Base License та Source SDK Base Multiplayer License.

Source SDK Base License надає розробникам доступ до Source Engine SDK для створення власних ігор або модифікацій. Ця ліцензія дозволяє використовувати рушій для некомерційних проектів без обмежень.

Source SDK Base Multiplayer License, зазвичай використовується для створення мультиплеєрних ігор, може вимагати додаткової ліцензійної угоди з Valve Corporation для комерційного використання або розповсюдження на платформах, таких як Steam.

Умови ліцензування та вартість використання Source Engine можуть різнитися в залежності від конкретних угод та домовленостей, які розробники укладають з Valve Corporation. Точна інформація про плату та ліцензійні умови повинна бути вказана в угоді між розробником і Valve Corporation.

Приклади ігор розроблених на Source.

Рисунок 2.19 – Приклад гри на Source. Half-Life 2

Рисунок 2.20 – Приклад гри на Source. Portal 2

Рисунок 2.21 – Приклад гри на Source. Counter-Strike: Global Offensive

2.3. Вибір рушія

При розгляді вибору рушія для створення ігрового додатку було обрано Unity. Unity - це потужний ігровий рушій, який надає розробникам широкі можливості та інструменти для створення ігор та додатків для різних платформ.

Основні причини, чому був обраний Unity, включають його кросплатформеність. Unity дозволяє розробляти ігри та додатки для комп'ютерів, мобільних пристроїв, ігрових консолей та віртуальної реальності, що дозволяє охопити широку аудиторію користувачів.

Також, Unity має активну спільноту розробників, яка надає доступ до безлічі ресурсів, документації та плагінів. Це допомагає розробникам знайти відповіді на свої питання, отримати підтримку та знайти рішення для своїх проектів.

Unity також використовує мову програмування C#, яка є потужною та широко використовуваною мовою. Використання C# дозволить мати повний

контроль над логікою гри та функціональністю, а також забезпечіть високу швидкодію та ефективність.

Крім того, Unity є легким у використанні, має інтуїтивний інтерфейс та підтримує технологію Drag & Drop, що спрощує процес створення та розміщення об'єктів у віртуальному просторі.

Узагалі, вибір Unity був обґрунтованим для мене через його кросплатформеність, активну спільноту, використання мови програмування C#, легкість використання та доступність різноманітних інструментів та ресурсів. Ці фактори роблять Unity привабливим ігровим рушієм.

2.4. Висновки до другого розділу

В другому розділі було надано загальний огляд найпопулярніших ігрових рушіїв, таких як Unity, CryEngine V, Unreal Engine 4 і Godot. Проаналізовано їх основні характеристики, можливості та особливості.

За результатами аналізу, було вирішено обрати Unity для створення мого ігрового додатку. Адже він відповідає всім потребам проекту. Цей рушій надає потужні інструменти та ресурси для розробки ігрового додатку.

Загально кажучи, розділ 2 допоміг краще розуміти різні ігрові рушії та їх можливості. Вибір Unity відповідає потребам і допоможе досягти успіху в розробці ігрового проекту.

РОЗДІЛ 3
ВИРІШЕННЯ ПОСТАВЛЕННОЇ В КВАЛІФІКАЦІНІЙ РОБОТІ ЗАДАЧІ

3.1. Розробка ігрового додатку

Для реалізації поставленої в кваліфікаційній роботі задачі, було обрано Unity як основний інструмент розробки гри. Unity є потужним і популярним ігровим рушієм, що надає широкий набір інструментів та можливостей для створення ігрових додатків.

Одним із ключових факторів у виборі Unity є підтримка мови програмування C#. Яка є потужною та простою у використанні мовою, вона дозволить мені ефективно керувати логікою гри та взаємодіяти з компонентами Unity.

Для програмування на C# використовується інтегроване середовище розробки Microsoft Visual Studio. Він надає потужні інструменти для розробки, відладки та управління проектом. Його інтуїтивний інтерфейс та широкий спектр функціональності допомагають забезпечити ефективну розробку гри.

Unity та C# разом з Microsoft Visual Studio утворюють потужний стек інструментів, які допомагають мені реалізувати весь потенціал ідеї гри. З їх допомогою можливо обробляти взаємодію з користувачем, створювати анімації, візуальні ефекти та багато іншого.

Такий інтегрований підхід спрощує процес розробки та дозволяє мені зосередитися на створенні захоплюючої гри, використовуючи розширені можливості Unity та гнучкість мови програмування C#.

3.2. Результати розробки

При старті додатку, користувач спочатку бачить екран запуску, що називається "Launch" екран. Цей екран вказує на використання рушія Unity для розробки гри, і протягом цього часу додаток завантажує ресурси та виконує код, який повинен виконатися під час запуску гри. Головною метою показу цього екрану є відображення логотипу розробника. Цей екран триває лише декілька секунд і не є інтерактивним.

Рисунок 3.1 – «Launch» екран

Після запуску додатку, гравець потрапляє до головного меню. Головне меню є окремою сценою в Unity, яка відображається на екрані. У цьому меню гравець має можливість обрати рівень, з якого він хоче почати гру, або вийти з гри.

Рисунок 3.2 – Головне меню

Після вибору першого рівня, гра починає завантажувати сцену з цим рівнем. Гравець опиняється в захоплюючому світі гри, де може контролювати головного персонажа.

Рисунок 3.3 – Перший рівень

На рисунку 3.4 зображено ворога який патрулює певну територію і який атакує гравця якщо він потрапить у його поле зору.

Рисунок 3.4 – Ворог

На щастя, у головного героя гри є можливість атакувати ворогів. Якщо гравець клікне лівою кнопкою миші, персонаж виконає атаку.

Рисунок 3.5 – Атака

У лівому верхньому куті екрану розміщені три червоних серця, які виступають індикаторами здоров'я персонажа. Ці серця відображають

кількість доступного здоров'я гравця. Якщо гравець не уникає атаку ворога, то він втратить одне серце зі свого запасу здоров'я.

Рисунок 3.6 – Здоров’я персонажа

Для подолання перешкод у грі будуть доступні дві важливі механіки - подвійний стрибок і "coyote jump". Подвійний стрибок дозволяє гравцеві зробити другий стрибок у повітрі після першого. Це дає можливість перетнути широкі прірви або досягти вищих платформ. "Coyote jump" використовується для надання гравцю короткого періоду часу, після відокремлення від землі, під час якого він може зробити стрибок. Ці механіки значно полегшують подолання перешкод і додають більше гнучкості та контролю гравцю під час гри.

Рисунок 3.7 – Подвійний стрибок

На малюнку 3.8 зображено зілля, яке гравець може підібрати.

Рисунок 3.8 – Зілля

Після того, як гравець підбирає зілля, воно автоматично потрапляє до його інвентаря. Звідки його можна використати щоб відновити 1 очко здоров'я.

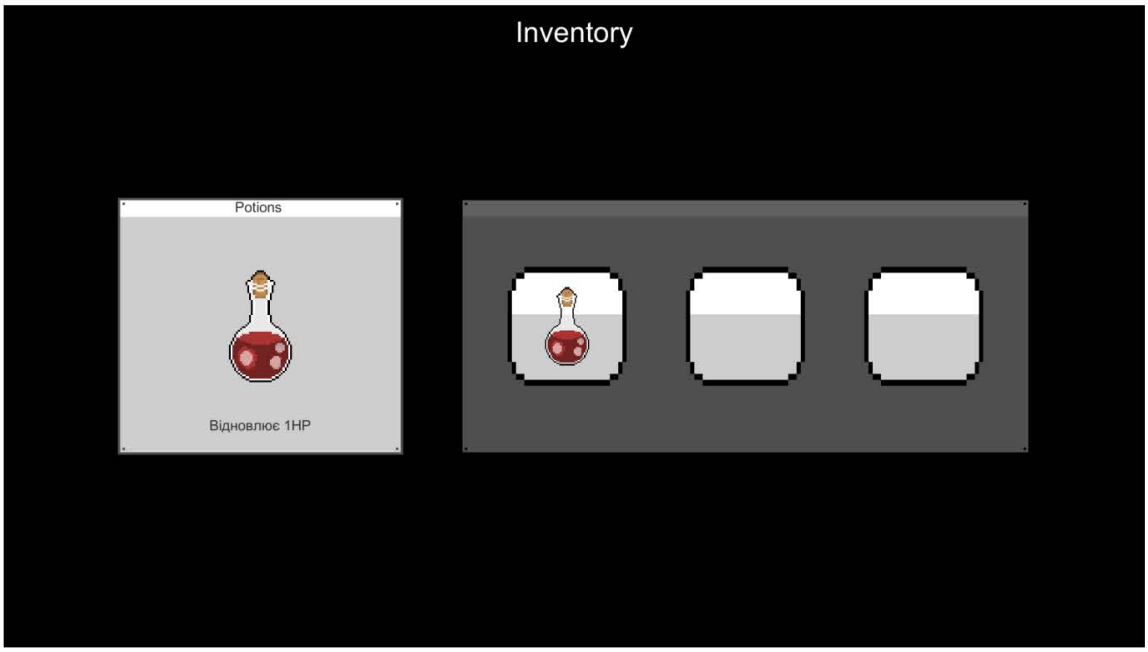

Рисунок 3.9 – Інвентар

У грі присутні пастки, які становлять небезпеку для гравця. Якщо гравець буде необережним та зачепить пастку, його персонажу буде віднято 1 очко здоров'я.

Рисунок 3.10 – Пастки

На малюнку 3.11 зображено важіль, який є інтерактивним об'єктом в грі. Гравець може взаємодіяти з цим важелем, натискаючи на нього. Взаємодія з важелем може мати певний ефект, наприклад, відкриття дверей,

що дозволить гравцеві перейти на наступний рівень. Це один з елементів геймплею, який забезпечує взаємодію та прогрес гравця через різні рівні гри.

Рисунок 3.11 – Важіль

Рисунок 3.12 – Взаємодія з предметом

В другому рівні гравця одразу зустрічає новий тип пастки - рухома пилка. Ця пастка знаходиться в русі і переміщається по заданому шляху. Якщо персонаж зіткнеться з пилкою, у нього буде віднято 1 очко здоров'я. Це

створює додатковий елемент виклику та випробування для гравця, оскільки він повинен уникати рухомих пилок, щоб зберегти своє здоров'я. Гравець повинен бути обережним та швидко реагувати, щоб уникати зіткнень з цими пастками і продовжити проходження гри.

Рисунок 3.13 – Рухома пилка

Також у другому рівні гравець може зустріти новий тип ворога з дальнім типом атаки. Цей ворог має здатність атакувати гравця з віддалі, випускаючи снаряди в його напрямку. Це додає додатковий рівень складності до гри, оскільки гравець повинен бути обережним і вміло уникати атак.

Рисунок 3.14 – Ворог з дальнім типом атаки

Якщо персонаж втрачає все своє здоров'я, то він помирає, і перед гравцем з'являється повідомлення про кінець гри з трьома варіантами дій:

1. Рестарт: Цей варіант дозволяє гравцеві перезапустити рівень. При виборі цього варіанту гравець повертається до початку поточного рівня з повним здоров'ям, а весь прогрес та зібрані предмети на рівні скидаються.
2. Головне меню: Вибір цього варіанту приводить гравця до головного меню гри.
3. Вихід: Цей варіант дозволяє гравцеві вийти з гри. Гра закривається, і гравець повертається до робочого столу.

Ці варіанти дій дозволяють гравцеві вибрати, як він хоче продовжувати гру після поразки персонажа. Він може спробувати ще раз, повернутися до головного меню або припинити гру.

Рисунок 3.15 – Кінець

Якщо натиснути на кнопку Esc під час гри, відкриється меню паузи, в якому можна продовжити гру, вийти до головного меню або завершити гру.

Рисунок 3.16 – Пауза

В грі присутні персонажі, з якими гравець може взаємодіяти, щоб провести діалог. Вони можуть надати корисну інформацію або дати пораду.

Рисунок 3.17 – Діалог

Фіналом гри є проходження третього рівня, під час якого гравець знаходить скарб. При взаємодії з цим скарбом, гравець отримує повідомлення про успішне завершення гри. Це відзначається як досягнення головної мети гри та досягнення успіху у подоланні усіх викликів та завдань, які виникали на шляху гравця.

Рисунок 3.18 – Скарб

Рисунок 3.19 – Фінал гри

3.3.Висновки до третього розділу

Мені вдалося створити захопливий платформер з цікавими механіками стрибків та атаки, ворогами різних типів та різноманітними пастками. Декількох рівнів достатньо, щоб випробувати навички гравця та представити різні виклики. Інвентар та діалоги додають глибину геймплею та роблять гру більш насиченою та цікавою для гравця.

В розділі 3 був обраний жанр гри, а саме платформер, і було уточнено, які інструменти та мова програмування були використані для створення гри. В даному випадку рушій Unity виступив основним інструментом для розробки гри, а мова програмування C# була використана для програмування різних ігрових компонентів та логіки.

Результатом розробки додатку стала гра, яка виконує задачі, поставлені в попередніх розділах. В розділі були пояснені основні можливості та особливості геймплею, що дозволяє гравцям легко орієнтуватись та використовувати додаток без проблем.

Завдяки використанню рушія Unity розробка гри стала швидкою та ефективною. Багатофункціональний редактор Unity дозволяє зручно налаштовувати та створювати ігрові об'єкти, а мова програмування C# забезпечує гнучкість та можливості для реалізації різноманітних ігрових механік.

Усі ці фактори сприяли успішному створенню гри і надали їй потенціал стати цікавим та захоплюючим додатком для гравців. Пояснення та опис можливостей гри сприятимуть легкому та комфортному використанню додатку користувачами.

ВИСНОВКИ

В даній дипломній роботі було проведено розробку ігрового додатку з використанням інтегрованого середовища Unity. Результати дослідження підтвердили, що Unity є потужним інструментом для створення якісних ігрових додатків.

Аналіз предметної області, зокрема ринку та ігрових платформ, є надзвичайно важливим етапом розробки ігрового додатку. Цей аналіз допомагає отримати глибоке розуміння потреб та вимог цільової аудиторії, а також визначити можливості та виклики, пов'язані з обраною платформою.

Один з ключових аспектів аналізу - це дослідження ринку ігрової індустрії. Вивчення поточних тенденцій, популярних жанрів, успішних ігрових продуктів та сегментів аудиторії дозволяє виявити можливості для успіху свого додатку. Аналіз ринку також допомагає виявити конкурентні переваги та ніші, які можна використовувати для позиціонування свого додатку.

Дослідження різних ігрових платформ, таких як персональні комп'ютери, консолі та мобільні пристрої, дозволяє зрозуміти їх особливості, технічні обмеження та цільову аудиторію. Кожна платформа має свої особливості, які слід враховувати під час розробки ігрового додатку. Наприклад, розмір екрану та способи вводу можуть вплинути на геймплей та інтерфейс, а технічні характеристики платформи можуть вимагати оптимізації для досягнення плавної роботи додатку.

Аналіз предметної області також включає вивчення особливостей цільової аудиторії. Розуміння їхніх вподобань, очікувань та поведінки гравців допомагає створити ігровий додаток, що задовольняє їхні потреби. Наприклад, деякі аудиторії можуть більше цінувати мультиплеєрні режими, тоді як інші - сюжетно-орієнтовані одиночні ігри. Аналіз допомагає визначити, які елементи гри можуть бути привабливими для аудиторії та які функціональні можливості варто реалізувати.

Враховуючи всі ці фактори, аналіз предметної області робить можливим розробку ігрового додатку, який відповідає потребам та очікуванням цільової аудиторії, максимально використовує можливості обраної платформи і має конкурентні переваги на ринку.

Під час огляду існуючих розв'язків для поставленої задачі, було проведено аналіз різних ігрових рушіїв з метою вибору найбільш підходящого методу вирішення. Аналіз рушіїв включав огляд їх функціональних можливостей, технічних обмежень, доступності документації та підтримки.

Починаючи з огляду рушіїв, було проаналізовано Unity - один з найпопулярніших ігрових рушіїв у галузі розробки. Unity відомий своєю гнучкістю, широким спектром функціональних можливостей та підтримкою різних платформ, включаючи ПК, консолі та мобільні пристрої. Його зручний інтерфейс та наявність великої спільноти розробників роблять Unity привабливим вибором для розробки ігрових додатків.

Також було проаналізовано Unreal Engine - потужний ігровий рушій, відомий своєю графікою та фізичною моделлю. Unreal Engine надає широкі можливості для створення реалістичних ігрових світів і вражаючих візуальних ефектів. Його потужність та підтримка віртуальної реальності роблять Unreal Engine привабливим вибором для проектів, що потребують високої якості графіки та іммерсивного досвіду геймплею.

У процесі аналізу рушіїв були враховані інші альтернативи, такі як CryEngine, Godot та Source, які мають свої унікальні особливості та переваги. Вибір методу вирішення задачі базувався на специфікаціях проекту, та вимогах до функціональності ігрового додатку.

В результаті проведеного аналізу різних ігрових рушіїв, було прийнято рішення вибрати Unity як основний рушій для розробки ігрового додатку, адже він задовільняє всі вимоги.

Під час розробки було вивчено основні принципи роботи з Unity, зокрема роботу зі сценами, об'єктами, компонентами та скриптами.

Після завершення розробки ігрового додатку було проведено його тестування та оптимізацію. Було виявлено та виправлено деякі помилки, а також забезпечено плавну роботу додатку.

Отриманий ігровий додаток відповідає вимогам, поставленим у першому та другому розділі, та демонструє гарні візуальні та геймплейні характеристики.

Загалом, робота підтверджує, що розробка ігрових додатків з використанням інтегрованого середовища Unity є ефективним та результативним процесом. Unity надає широкі можливості для створення якісних ігор з відмінною візуальною та геймплейною складовими. Результати даної роботи можуть бути використані в подальшому розвитку індустрії геймдеву та сприяти створенню нових захоплюючих ігор для різних платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кравчук І. Використання інтегрованого середовища Unity для розробки ігрових додатків // Вісник Національного університету "Львівська

політехніка". Серія: Інформаційні системи та мережі. - 2010. - Вип. 679. - С. 177-183.

2. Hutnik M., Mardesić S., Vukoja J. Overview and comparison of game engines Unity, Unreal Engine and CryENGINE // MIPRO, 2013 Proceedings of the 36th International Convention. - IEEE, 2013. - С. 1425-1430.

3. Романюк В., Пашковський І. Розробка ігор з використанням Unity: навчальний посібник. - Київ: Інститут комп'ютерних наук ім. В. М. Глушкова НАН України, 2014.

4. Unity Technologies. Unity User Manual. URL: https://docs.unity3d.com/Manual/index.html

5. Mauthe A., Timmerer C. Networked Applications: A Guide to the New Computing Infrastructure. - Morgan Kaufmann, 2016.

6. Голуб І. Розробка ігор з використанням Unity: практикум. - Київ: Видавництво "Просвіта", 2017.

7. Unity Technologies. Unity Scripting API Documentation. URL: https://docs.unity3d.com/ScriptReference/index.html

8. Huang R. Unity Virtual Reality Projects: Learn Virtual Reality by Building Immersive Applications and Games with Unity. - Packt Publishing Ltd, 2018.

9. Данильченко В. Створення мобільних ігор з використанням Unity. - Київ: Видавничий дім "Символ", 2018.

10. Unity Technologies. Unity Asset Store. URL: https://assetstore.unity.com/

11. Harrison H. Mastering Unity 2D Game Development: Building Exceptional 2D Games with Unity. - Packt Publishing Ltd, 2018.

12. Федоров М. Створення ігор на Unity для Android: практичний посібник. - Київ: Видавничий дім "Символ", 2019.

13. Unity Technologies. Unity Learn. URL: https://learn.unity.com/

14. Zambelli A. Unity 2019 Cookbook: Over 100 recipes to spice up your Unity skills. - Packt Publishing Ltd, 2019.

15. Карякін О., Підгурський Р. Ігрові проекти з Unity: методологія, алгоритми, реалізація. - Київ: Видавничий дім "Символ", 2020.

16. Harrison H. Mastering Unity 2020: Build advanced games and applications with Unity and the latest tools in the Unity ecosystem. - Packt Publishing Ltd, 2020.

17. Unity Technologies. Unity Forum. URL: https://forum.unity.com/

18. Ferreira H., Markovic M. Unity 2020 By Example: A Project-Based Guide to Learning Unity Game Development. - Packt Publishing Ltd, 2021.

19. Огар Ю. Розробка ігор на Unity. - Київ: Видавничий дім "Символ", 2021.

20. Unity Technologies. Unity Blog. URL: https://blogs.unity3d.com/

21. Герасименко О. Unity для дітей: курс програмування на C# зі створенням ігор. - Київ: Видавництво "Просвіта", 2019.

22. Unity Technologies. Unity Documentation. URL: https://docs.unity3d.com/

23. Курило А. Основи розробки ігор з використанням Unity. - Київ: Видавничий дім "Символ", 2020.

24. Unity Technologies. Unity Asset Store: 3D models, animation, particles, and other digital assets. URL: https://assetstore.unity.com/

25. Уиттон Б. Unity 2020 Shaders and Effects Cookbook: Transform your game into a visually stunning masterpiece with over 90 recipes. - Packt Publishing Ltd, 2020.

26. Гевко І. Створення мобільних ігор з використанням Unity: посібник для початківців. - Київ: Видавничий дім "Символ", 2021.

27. Unity Technologies. Unity Asset Store: Tools, services, and infrastructure for your project. URL: https://assetstore.unity.com/

28. Лепіш Ю. Unity Game Development Cookbook: Over 100 recipes exploring the new and exciting features of Unity 2021. - Packt Publishing Ltd, 2021.

29. Купрій В., Прудіус В. Графічна програма Unity: навчальний посібник. - Київ: Видавничий дім "Символ", 2022.

30. Unity Technologies. Unity Asset Store: Audio, music, and sound effects for your game. URL: https://assetstore.unity.com/

31. Волков О., Євтушенко С. Unity для вчителів: навчальний посібник з програмування ігор. - Київ: Видавничий дім "Символ", 2022.

32. Unity Technologies. Unity Certification. URL: https://certification.unity.com/

33. Ребрина О. Розробка ігор з використанням Unity 3D: навчальний посібник. - Київ: Видавничий дім "Символ", 2022

34. Unity Technologies. Unity Analytics: Insights for your game. URL: https://unity.com/solutions/analytics

35. Петренко І., Костюк О. Розробка ігор на движку Unity 3D: навчальний посібник. - Київ: Видавництво "Просвіта", 2023.

36. Unity Technologies. Unity Machine Learning: Enhance your games and simulations with AI. URL: https://unity.com/solutions/machine-learning

37. Ліберман М. Графічна програма Unity: навчальний посібник. - Київ: Видавничий дім "Символ", 2023.

38. Unity Technologies. Unity Services: Monetize, engage, and discover games. URL: https://unity.com/solutions/services

39. Старченко Д. Розробка 3D-ігор з використанням Unity: навчальний посібник. - Київ: Видавництво "Просвіта", 2023

40. Unity Technologies. Unity Collaborate: Unity in the cloud. URL: https://unity.com/solutions/collaborate

ДОДАТОК А

ПРОГРАМНИЙ КОД

Файл Camera.cs

```
public class Camera : MonoBehaviour
```

```
{
    public Transform target;
    public Vector3 offset;
    [Range(1, 10)]
    public float smoothFactor;
    public Vector3 minValues, maxValue;
    private void FixedUpdate()
    {
        Follow();
    }
    void Follow()
    {
        Vector3 targetPosition = target.position + offset;
        Vector3 boundPosition = new Vector3(
            Mathf.Clamp(targetPosition.x, minValues.x, maxValue.x),
            Mathf.Clamp(targetPosition.y, minValues.y, maxValue.y),
            Mathf.Clamp(targetPosition.z, minValues.z, maxValue.z));
        Vector3 smoothPosition = Vector3.Lerp(transform.position, boundPosition,
smoothFactor * Time.fixedDeltaTime);
        transform.position = smoothPosition;
    }
```

```
}
```

Файл Dialogue.cs

```
public class Dialogue : MonoBehaviour
{
    public GameObject window;
    public TMP_Text dialogueText;
    public List<string> dialogues;
    public float writingSpeed;
    private int index;
    private int charIndex;
    private bool started;
    private bool waitForNext;
    private void Awake()
    {
        ToggleWindow(false);
    }
    private void ToggleWindow(bool show)
    {
        window.SetActive(show);
    }
    public void StartDialogue()
    {
        if (started)
            return;
        started = true;
        ToggleWindow(true);
        GetDialogue(0);
```

```
}
private void GetDialogue(int i)
{
    index = i;
    charIndex = 0;
    dialogueText.text = string.Empty;
    StartCoroutine(Writing());
}
public void EndDialogue()
{
    started = false;
    waitForNext = false;
    StopAllCoroutines();
    ToggleWindow(false);
}
IEnumerator Writing()
{
    yield return new WaitForSeconds(writingSpeed);
    string currentDialogue = dialogues[index];
    dialogueText.text += currentDialogue[charIndex];
    charIndex++;
    if (charIndex < currentDialogue.Length)
    {
        yield return new WaitForSeconds(writingSpeed);
        StartCoroutine(Writing());
    }
    else
    {
```

```
            waitForNext = true;
        }
    }
    private void Update()
    {
        if (!started)
            return;
        if (waitForNext && Input.GetKeyDown(KeyCode.E))
        {
            waitForNext = false;
            index++;
            if (index < dialogues.Count)
            {
                GetDialogue(index);
            }
            else
            {
                EndDialogue();
            }
        }
    }
}
```

Файл DialogueTrigger.cs

```
public class DialogueTrigger : MonoBehaviour
{
```

```
    public Dialogue dialogueScript;
    private bool playerDetected;
    public void OnTriggerEnter2D(Collider2D collision)
    {
        playerDetected = true;
    }
    public void OnTriggerExit2D(Collider2D collision)
    {
        playerDetected = false;
        dialogueScript.EndDialogue();
    }
      private void Update()
    {
      if(playerDetected && Input.GetKeyDown(KeyCode.E))
      {
        dialogueScript.StartDialogue();
      }
    }
}
```

Файл InteractionSystem.cs

```
public class InteractionSystem : MonoBehaviour
```

```
{
    public Transform detectionPoint;
    private const float detectionRadius = 0.2f;
    public LayerMask detectionLayer;
    public GameObject detectedObject;
    public GameObject examineWindow;
    public GameObject grabbedObject;
    public float grabbedObjectYValue;
    public Transform grabPoint;
    public Image examineImage;
    public Text examineText;
    public bool isExamining;
    void Update()
    {
        if (DetectObject())
        {
            if (InteractInput())
            {
                detectedObject.GetComponent<Item>().Interact();
            }
        }
```

```
    }
    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.green;
        Gizmos.DrawSphere(detectionPoint.position, detectionRadius);
    }
    bool InteractInput()
    {
        return Input.GetKeyDown(KeyCode.E);
    }
    bool DetectObject()
    {
        Collider2D obj = Physics2D.OverlapCircle(detectionPoint.position,
detectionRadius, detectionLayer);
        if (obj == null)
        {
            detectedObject = null;
            return false;
        }
        else
        {
```

```
            detectedObject = obj.gameObject;
            return true;
        }
    }
    public void ExamineItem(Item item)
    {
        if (isExamining)
        {
            examineWindow.SetActive(false);
            isExamining = false;
            Time.timeScale = 1;
        }
        else
        {
            examineImage.sprite = item.GetComponent<SpriteRenderer>().sprite;
            examineText.text = item.descriptionText;
            examineWindow.SetActive(true);
            isExamining = true;
            Time.timeScale = 0;
        }
    }
```

```
}
```

Файл Item.cs

```
public class Item : MonoBehaviour
{
    public enum InteractionType { NONE, PickUp, Examine}
    public enum ItemType { Staic, Consumables }
    public InteractionType interactType;
    public ItemType type;
    public string descriptionText;
    public UnityEvent customEvent;
    public UnityEvent consumeEvent;
    private void Reset()
    {
        GetComponent<Collider2D>().isTrigger = true;
        gameObject.layer = 8;
    }
    public void Interact()
    {
        switch (interactType)
        {
            case InteractionType.PickUp:
```

```
            FindObjectOfType<InventorySystem>().PickUp(gameObject);
            gameObject.SetActive(false);
            break;
        case InteractionType.Examine:
            FindObjectOfType<InteractionSystem>().ExamineItem(this);
            break;
        default:
            Debug.Log("NULL ITEM");
            break;
    }
    customEvent.Invoke();
  }
}
```

Файл MainMenu.cs

```
public class MainMenu : MonoBehaviour
{
  [SerializeField] private GameObject menuScreen;

  public void Level1()
  {
    SceneManager.LoadScene(1);
```

```
        Time.timeScale = 1;
    }
    public void Level2()
    {
        SceneManager.LoadScene(2);
        Time.timeScale = 1;
    }
    public void Level3()
    {
        SceneManager.LoadScene(3);
        Time.timeScale = 1;
    }
    public void Quit()
    {
        Application.Quit();
    }
}
```

Файл melleAttack.cs

```
public class melleAttack : MonoBehaviour
```

```
{
    [SerializeField] private float attackCooldown;
    [SerializeField] private float range;
    [SerializeField] private int damage;
    [SerializeField] private float colliderDistance;
    [SerializeField] private BoxCollider2D boxCollider;
    [SerializeField] private LayerMask enemyLayer;
    private Animator anim;
    private PlayerMovement playerMovement;
    private float cooldownTimer = Mathf.Infinity;
    private Health playerHealth;
    private void Awake()
    {
        anim = GetComponent<Animator>();
        playerMovement = GetComponent<PlayerMovement>();
    }
    private void Update()
    {
        if (Input.GetMouseButton(0) && cooldownTimer > attackCooldown &&
Time.timeScale > 0)
            Attack();
```

```
        cooldownTimer += Time.deltaTime;
    }
    private void Attack()
    {
        anim.SetTrigger("Attack");
        cooldownTimer = 0;
    }
    private bool InSight()
    {
        RaycastHit2D hit =
            Physics2D.BoxCast(boxCollider.bounds.center + transform.right * range *
transform.localScale.x * colliderDistance,
            new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y,
boxCollider.bounds.size.z),
            0, Vector2.left, 0, enemyLayer);
        if (hit.collider != null)
            playerHealth = hit.transform.GetComponent<Health>();
        return hit.collider != null;
    }
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
```

```
        Gizmos.DrawWireCube(boxCollider.bounds.center + transform.right * range
* transform.localScale.x * colliderDistance,
            new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y,
boxCollider.bounds.size.z));
    }
    private void DamageEnemy()
    {
        if (InSight())
            playerHealth.TakeDamage(damage);
    }
}
```

Файл NextLevel.cs

```
public class NextLevel : MonoBehaviour
{
    public int sceneBuildIndex;
    private void OnTriggerEnter2D(Collider2D others)
    {
        print("Trigger Entered");
        if (others.tag == "Player")
        {
            print("Switching Scene to " + sceneBuildIndex);
            SceneManager.LoadScene(sceneBuildIndex, LoadSceneMode.Single);
```

```
        }
    }
}
```

Файл PlayerMovement.cs

```
public class PlayerMovement : MonoBehaviour
{
    [SerializeField] private float speed;
    [SerializeField] private float jumpPower;
    [SerializeField] private float coyoteTime;
    private float coyoteCounter;
  public DialogueTrigger dialogueTrigger;
  private int jumpCounter;
    [SerializeField] private LayerMask groundLayer;
    private Rigidbody2D body;
    private Animator anim;
    private BoxCollider2D boxCollider;
    private float horizontalInput;

    private void Awake()
    {
        body = GetComponent<Rigidbody2D>();
```

```
        anim = GetComponent<Animator>();
        boxCollider = GetComponent<BoxCollider2D>();
    }
    private void Update()
    {
    if (CanMoveOrInteract() == false)
        return;
    horizontalInput = Input.GetAxis("Horizontal");
    if (horizontalInput > 0.01f)
            transform.localScale = new Vector3((float)2.5, 4, 1);
    else if (horizontalInput < -0.01f)
            transform.localScale = new Vector3((float)-2.5, 4, 1);
        anim.SetBool("Run", horizontalInput != 0);
        anim.SetBool("grounded", isGrounded());
    if (Input.GetKeyDown(KeyCode.Space))
            Jump();
        if (Input.GetKeyUp(KeyCode.Space) && body.velocity.y > 0)
            body.velocity = new Vector2(body.velocity.x, body.velocity.y / 2);
        else
        {
            body.velocity = new Vector2(horizontalInput * speed, body.velocity.y);
```

```
            if (isGrounded())
            {
                coyoteCounter = coyoteTime;
            }
            else
                coyoteCounter -= Time.deltaTime;
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("NPC"))
        {
            DialogueTrigger dialogueTrigger =
collision.GetComponentInChildren<DialogueTrigger>();
            if (dialogueTrigger != null)
            {
                dialogueTrigger.OnTriggerEnter2D(collision);
            }
        }
    }
    bool CanMoveOrInteract()
```

```
{
    bool can = true;
    if (FindObjectOfType<InteractionSystem>().isExamining)
        can = false;
    if (FindObjectOfType<InventorySystem>().isOpen)
        can = false;
    return can;
}
private void Jump()
    {
        if (coyoteCounter <= 0 ) return;
        else
        {
            if (isGrounded())
                body.velocity = new Vector2(body.velocity.x, jumpPower);
            else
            {
                if (coyoteCounter > 0)
                    body.velocity = new Vector2(body.velocity.x, jumpPower);
            }
            coyoteCounter = 0;
```

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

}
```

Файл UIManager.cs

```
public class UIManager : MonoBehaviour
{
  [SerializeField] private GameObject gameOverScreen;
  [SerializeField] private GameObject pauseScreen;
  private void Awake()
  {
```

```
    gameOverScreen.SetActive(false);
}
private void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        PauseGame(!pauseScreen.activeInHierarchy);
    }
}
public void PauseGame(bool status)
{
    pauseScreen.SetActive(status);
    if (status)
        Time.timeScale = 0;
    else
        Time.timeScale = 1;
}

public void GameOver()
{
    gameOverScreen.SetActive(true);
```

```
    }
    public void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        Time.timeScale = 1;
    }
    public void MainMenu()
    {
        SceneManager.LoadScene(0);
    }
    public void Quit()
    {
        Application.Quit();
    }
}
```

Файл EnemyDamage.cs

```
public class EnemyDamage : MonoBehaviour
{
    [SerializeField] protected float damage;
    protected void OnTriggerEnter2D(Collider2D collision)
    {
```

```
        if (collision.tag == "Player")
            collision.GetComponent<Health>()?.TakeDamage(damage);
    }
}
```

Файл EnemyFireballHolder.cs

```
public class EnemyFireballHolder : MonoBehaviour
{
    [SerializeField] private Transform enemy;
    private void Update()
    {
        transform.localScale = enemy.localScale;
    }
}
```

Файл EnemyPatrol.cs

```
public class EnemyPatrol : MonoBehaviour
{
    [SerializeField] private Transform leftEdge;
    [SerializeField] private Transform rightEdge;
    [SerializeField] private Transform enemy;
    [SerializeField] private float speed;
    private Vector3 initScale;
```

```
private bool movingLeft;
[SerializeField] private float idleDuration;
private float idleTimer;
[SerializeField] private Animator anim;
private void Awake()
{
    initScale = enemy.localScale;
}
private void OnDisable()
{
    anim.SetBool("moving", false);
}
private void Update()
{
    if (movingLeft)
    {
        if (enemy.position.x >= leftEdge.position.x)
            MoveInDirection(-1);
        else
            DirectionChange();
    }
```

```
    else
    {
        if (enemy.position.x <= rightEdge.position.x)
            MoveInDirection(1);
        else
            DirectionChange();
    }
}
private void DirectionChange()
{
    anim.SetBool("moving", false);
    idleTimer += Time.deltaTime;

    if (idleTimer > idleDuration)
        movingLeft = !movingLeft;
}
private void MoveInDirection(int _direction)
{
    idleTimer = 0;
    anim.SetBool("moving", true);
    enemy.localScale = new Vector3(Mathf.Abs(initScale.x) * _direction,
```

```
            initScale.y, initScale.z);
        enemy.position = new Vector3(enemy.position.x + Time.deltaTime *
_direction * speed,
            enemy.position.y, enemy.position.z);
    }
}
```

Файл MeleeEnemy.cs

```
public class MeleeEnemy : MonoBehaviour
{
    [SerializeField] private float attackCooldown;
    [SerializeField] private float range;
    [SerializeField] private int damage;
    [SerializeField] private float colliderDistance;
    [SerializeField] private BoxCollider2D boxCollider;
    [SerializeField] private LayerMask playerLayer;
    private float cooldownTimer = Mathf.Infinity;
    private Animator anim;
    private Health playerHealth;
    private EnemyPatrol enemyPatrol;
    private void Awake()
    {
```

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

private void Update()
{
    cooldownTimer += Time.deltaTime;
    if (PlayerInSight())
    {
        if (cooldownTimer >= attackCooldown)
        {
            cooldownTimer = 0;
            anim.SetTrigger("meleeAttack");
        }
    }
    if (enemyPatrol != null)
        enemyPatrol.enabled = !PlayerInSight();
}
private bool PlayerInSight()
{
    RaycastHit2D hit =
```

```
        Physics2D.BoxCast(boxCollider.bounds.center + transform.right * range *
transform.localScale.x * colliderDistance,
        new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y,
boxCollider.bounds.size.z),
        0, Vector2.left, 0, playerLayer);
    if (hit.collider != null)
        playerHealth = hit.transform.GetComponent<Health>();
    return hit.collider != null;
  }
  private void OnDrawGizmos()
  {
    Gizmos.color = Color.red;
    Gizmos.DrawWireCube(boxCollider.bounds.center + transform.right * range
* transform.localScale.x * colliderDistance,
        new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y,
boxCollider.bounds.size.z));
  }
  private void DamagePlayer()
  {
    if (PlayerInSight())
        playerHealth.TakeDamage(damage);
  }
```

```
}
```

Файл Projectile.cs

```
public class Projectile: EnemyDamage
{
    [SerializeField] private float speed;
    [SerializeField] private float resetTime;
    private float lifetime;
    private Animator anim;
    private BoxCollider2D coll;
    private bool hit;
    private void Awake()
    {
        anim = GetComponent<Animator>();
        coll = GetComponent<BoxCollider2D>();
    }
    public void ActivateProjectile()
    {
        hit = false;
        lifetime = 0;
        gameObject.SetActive(true);
        coll.enabled = true;
```

```
}
private void Update()
{
    if (hit) return;
    float movementSpeed = speed * Time.deltaTime;
    transform.Translate(movementSpeed, 0, 0);
    lifetime += Time.deltaTime;
    if (lifetime > resetTime)
        gameObject.SetActive(false);
}
private void OnTriggerEnter2D(Collider2D collision)
{
    hit = true;
    base.OnTriggerEnter2D(collision);
    coll.enabled = false;
    if (anim != null)
        anim.SetTrigger("explode");
    else
        gameObject.SetActive(false);
}
private void Deactivate()
```

```
    {
        gameObject.SetActive(false);
    }
}
```

Файл RangedEnemy.cs

```
public class RangedEnemy : MonoBehaviour
{
    [SerializeField] private float attackCooldown;
    [SerializeField] private float range;
    [SerializeField] private Transform firepoint;
    [SerializeField] private GameObject[] fireballs;
    [SerializeField] private float colliderDistance;
    [SerializeField] private BoxCollider2D boxCollider;
    [SerializeField] private LayerMask playerLayer;
    private float cooldownTimer = Mathf.Infinity;
    private Animator anim;
    private EnemyPatrol enemyPatrol;
    private void Awake()
    {
        anim = GetComponent<Animator>();
        enemyPatrol = GetComponentInParent<EnemyPatrol>();
```

```
}
private void Update()
{
    cooldownTimer += Time.deltaTime;
    if (PlayerInSight())
    {
        if (cooldownTimer >= attackCooldown)
        {
            cooldownTimer = 0;
            anim.SetTrigger("rangedAttack");
        }
    }
    if (enemyPatrol != null)
        enemyPatrol.enabled = !PlayerInSight();
}
private void RangedAttack()
{
    cooldownTimer = 0;
    fireballs[FindFireball()].transform.position = firepoint.position;
    fireballs[FindFireball()].GetComponent<Projectile>().ActivateProjectile();
}
```

```
    private int FindFireball()
    {
        for (int i = 0; i < fireballs.Length; i++)
        {
            if (!fireballs[i].activeInHierarchy)
                return i;
        }
        return 0;
    }
    private bool PlayerInSight()
    {
        RaycastHit2D hit =
            Physics2D.BoxCast(boxCollider.bounds.center + transform.right * range *
transform.localScale.x * colliderDistance,
            new Vector3(boxCollider.bounds.size.x * range, boxCollider.bounds.size.y,
boxCollider.bounds.size.z),
            0, Vector2.left, 0, playerLayer);
        return hit.collider != null;
    }
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
```

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

Файл Health.cs

```
public class Health : MonoBehaviour
{
    [SerializeField] private float startingHealth;
    public float currentHealth { get; private set; }
    private Animator anim;
    private bool dead;
    private SpriteRenderer spriteRend;
    private UIManager uiManager;
    [SerializeField] private Behaviour[] components;
    private bool invulnerable;
    [SerializeField] private AudioClip deathSound;
    [SerializeField] private AudioClip hurtSound;
    private void Awake()
    {
        currentHealth = startingHealth;
```

```
    anim = GetComponent<Animator>();
    spriteRend = GetComponent<SpriteRenderer>();
    uiManager = FindObjectOfType<UIManager>();
}
public void TakeDamage(float _damage)
{
    if (invulnerable) return;
    currentHealth = Mathf.Clamp(currentHealth - _damage, 0, startingHealth);
    if (currentHealth > 0)
    {
        anim.SetTrigger("hurt");
    }
    else
    {
        if (!dead)
        {
            anim.SetTrigger("die");
            if (GetComponent<PlayerMovement>() != null)
            {
                GetComponent<PlayerMovement>().enabled = false;
                uiManager.GameOver();
```

```
                return;
            }
                if (GetComponentInParent<EnemyPatrol>() != null)
                GetComponentInParent<EnemyPatrol>().enabled = false;
            if (GetComponent<MeleeEnemy>() != null)
                GetComponent<MeleeEnemy>().enabled = false;
            if (GetComponent<RangedEnemy>() != null)
                GetComponent<RangedEnemy>().enabled = false;
            dead = true;
        }
    }
}
public void AddHealth()
{
    currentHealth = Mathf.Clamp(currentHealth + 1, 0, startingHealth);
}
private void Deactivate()
{
    gameObject.SetActive(false);
}
}
```

Файл HP_Bar.cs

```
public class HP_Bar : MonoBehaviour
{
    [SerializeField] private Health playerHealth;
    [SerializeField] private Image totalhealthBar;
    [SerializeField] private Image currenthealthBar;
    private void Start()
    {
        totalhealthBar.fillAmount = playerHealth.currentHealth / 10;
    }
    private void Update()
    {
        currenthealthBar.fillAmount = playerHealth.currentHealth / 10;
    }
}
```

EnemySideway.cs

```
public class EnemySideway : MonoBehaviour
{
    [SerializeField] private float movementDistance;
    [SerializeField] private float speed;
    [SerializeField] private float damage;
```

```
    private bool movingLeft;
    private float leftEdge;
    private float rightEdge;
    private void Awake()
    {
        leftEdge = transform.position.x - movementDistance;
        rightEdge = transform.position.x + movementDistance;
    }
    private void Update()
    {
        if (movingLeft)
        {
            if (transform.position.x > leftEdge)
            {
                transform.position = new Vector3(transform.position.x - speed *
Time.deltaTime, transform.position.y, transform.position.z);
            }
            else
                movingLeft = false;
        }
        else
```

```
        {
            if (transform.position.x < rightEdge)
            {
                transform.position = new Vector3(transform.position.x + speed *
Time.deltaTime, transform.position.y, transform.position.z);
            }
            else
                movingLeft = true;
        }
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.tag == "Player")
        {
            collision.GetComponent<Health>().TakeDamage(damage);
        }
    }
}
```