Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Баляна Андрія Давідовича*
(ПІБ)

академічної групи *121-19-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка кросплатформного ігрового застосунку на основі технологічного стеку Unity/C#*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи				
розділів:				
спеціальний	*доц. Ширін А.Л.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*ст.викл. Мартиненко А.А.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-19-1* (група) *Баляна Андрія Давідовича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка кросплатформного ігрового застосунку на основі технологічного стеку Unity/C#*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц. Ширін А.Л.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Балян А.Д.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 60 с., 33 рис., 3 дод., 23 джерел.

Об’єкт розробки: мультиплатформний ігровий додаток.

Мета кваліфікаційної роботи: розробка мультиплатформного ігрового застосунку на основі технологічного стеку Unity/C#.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми та її розробка, наводиться опис алгоритму і структура функціонування додатку, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота та завантаження програми.

В економічному розділі визначено трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню додатку та розраховується час на його створення.

Практичне значення полягає у розробці ігрового додатку, що можна запускати на одній з декількох платформ та дозволяє користувачам використовувати його для сниження стрессу та розвидку посидючості.

Актуальність програмного продукту визначається значним ростом популярності комп’ютерних та мобільних ігор та загальним стабільним та потужним зростанням ігрової індустрії.

Список ключових слів: ГРА, МУЛЬТИПЛАТФОРМНИЙ ДОДАТОК, ІГРОВИЙ ДОДАТОК, UNITY, ІГРОВИЙ ДВИГУН, КОРИСТУВАЧ, ПЛАТФОРМЕР.

ABSTRACT

Explanatory note: 51 p., 33 figs., 3 appx., 23 sources.

Object of development: multiplatform game application.

Purpose of the qualification work: Development of a cross-platform game application based on the Unity/C# technology stack.

In the introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiated the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development is carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work and running of the program is described.

In the economic section it is determined the complexity of the developed software product, calculated the cost of work to create an application and calculated the time to create it.

Of practical importance is the development of a game application that can be run on one of several platforms and allows users to use it for reducing stress and developing of perseverance.

The relevance of the software product is determined by the significant growth in the popularity of computer and mobile games and the overall stable and strong growth of the gaming industry.

Keywords: GAME, MULTIPLATFORM APPLICATION, GAME APPLICATION, UNITY, GAME ENGINE, USER, PLATFORMER.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UNITY – багатоплатформний інструмент для розробки відеоігор і додатків;

C# – мова програмування;

JSON – текстовий формат обміну даними;

EXE – розширення виконуваного файлу, що застосовується в системах Microsoft Windows;

ОС – операційна система;

ПК – персональний комп’ютер;

GPU – графічний процесор;

CPU – центральний процесор.

Платформер – жанр ігор, в якій гравець багато стрибає.

Паралакс ефект – ефект, при якому об’єкти рухаються з різною швидкістю залежно від відстані до камери гравця.

Геймдев – ігрова розробка.

Катсцена – катсцена визначається як візуальне представлення гри, де відбувається розміщення та організація об'єктів, графічних елементів та логіки гри.

ЗМІСТ

ВСТУП

Ігрова індустрія є однією з найшвидше розвиваючихся та впливових галузей сучасного світу. Вона об'єднує технології, креативність та розваги, створюючи неповторний світ віртуальних ігор, які привертають увагу мільйонів людей по всьому світу.

Ігрова індустрія має багато різноманітних аспектів, які включають в себе розробку, дизайн, програмування, мистецтво, аудіо та маркетинг. Ці різні елементи співпрацюють, щоб створити захоплюючий та емоційно збагачений геймплей для гравців.

Метою данної кваліфікаційної роботи є розробка мультиплатформного ігрового застосунку на основі технологічного стеку Unity/C#.

Поглиблене вивчення індустрії відеоігор та реалізація ігрового застосунку «Rising of Slime: Reslimed» буде допомагати знизити рівень стресу та розвивати усидливість, що є важливим та актуальним у сьогоденні.

Розробленний застосунок повинен дозволяти виконувати наступні функції для гравця:

- Двигати головного героя.
- Знаходити та підбирати предмети.
- Купувати предмети.
- Відкрити меню паузи де можна:
    - Відновити ігрову сесію.
    - Відкрити настройки:
        - Змінити режим повноекранного екрану.
        - Змінити гучність музики.
        - Змінити гучність звуків.
    - Вийти в головне меню.
    - Вийти з гри.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Розробка відеоігор - це складний та творчий процес, який вимагає злагодженої роботи команди фахівців. Розробники ігор відповідають за створення історій, персонажів, рівнів та геймплею, забезпечуючи гравцям незабутні пригоди. Вони використовують різні інструменти та технології, такі як двигуни гри, 3D моделювання, анімацію, текстурування та спеціальні ефекти, щоб створити візуально привабливі та реалістичні світи.

Дизайн гри є ще одним важливим аспектом ігрової індустрії. Геймдизайнери розробляють концепції ігор, задають ігрові механіки, виклики та баланс гри. Вони враховують естетику гри, психологію гравців та взаємодію з ігровим світом, створюючи неповторний ігровий досвід.

Програмування також грає важливу роль у розробці відеоігор. Програмісти використовують мови програмування, такі як C++, C#, Python та інші, щоб створювати логіку гри, реалізовувати інтерактивність та оптимізувати продуктивність ігри. Вони вирішують складні технічні проблеми та створюють інструменти для полегшення розробки.

Мистецтво також є невід'ємною частиною ігрової індустрії. Художники та дизайнери створюють візуальні елементи гри, такі як персонажі, рівні, об'єкти та інтерфейс користувача. Вони використовують різні техніки малювання, моделювання, текстурування та анімації, щоб створити чарівний та привабливий вигляд гри.

Аудіо відіграє також важливу роль у створенні незабутній ігровий досвід. Звукові дизайнери створюють звукові ефекти, музику та діалоги, щоб підкреслити атмосферу та емоції гри. Вони використовують акустичні ефекти, записи голосу та композиції, щоб занурити гравців у світ гри.

Крім того, маркетинг ігрової індустрії грає важливу роль у просуванні та комерціалізації відеоігор. Маркетологи вивчають ринок, проводять дослідження аудиторії та розробляють стратегії маркетингу, щоб залучити та зберегти гравців.

Ігрова індустрія не тільки надає розвагу та розвиває креативність, але й має потенціал для використання в освіті, медицині, симуляціях та інших сферах. Вона поєднує технології, мистецтво та інновації, щоб створювати неперевершені світи та незабутній досвід для гравців у всьому світі.

Створення відеоігор – одна із найбільших сегментів індустрії розваг. Масштаби ігрової промисловості можна порівняти, наприклад, з кіноіндустрією. А за швидкістю зростання за останні п'ять років індустрія відеоігор суттєво її випереджала.

І це зрозуміло, тому що головна перевага відеоігор в тому, що вони дають можливість напряму впливати на історію та буквально стати головним героєм. Ніякий фільм не може зробити те, що може гра, але люба гра може зробити те, що може фільм.

За ступенем впливу на споживачів та залучення їх до інтерактивного оточення, пропонованого відеоіграми, цей сегмент вже давно виділяється серед інших видів розваг.

Геймдев чи розробку ігор неможливо розглядати відокремлено від індустрії комп'ютерних ігор загалом. Безпосередньо створення ігор – це лише частина комплексної «екосистеми», що забезпечує повний життєвий цикл виробництва, розповсюдження та споживання таких складних продуктів як комп'ютерні ігри.

У структурі сучасної ігрової промисловості можна назвати такі рівні: платформи, ігрові движки, розробка відеоігор, видання, популяризація і споживання.

Є багато платформ на яких можна грати:

- Персональні комп’ютери (Windows, Linux та MacOS).
- Мобільні пристрої (Android, Windows, iOS).

- Аркадні автомати.
- Ігрові консолі (PlayStation, Xbox, Nintendo Switch, SteamDeck і тд.).
- Web-платформи (Браузери).

Ігрові движки важлива частина ігрової індустрії. Саме вони дозволяють навіть не дуже досвідченим розробникам створювати ігри та привносити свої ідеї у світ.

Є багато ігрових движків, від просунутого рівня (Unity, Unreal Engine) до дуже легких (Ren'Py, GameMaker), так щось середнє (Godot Engine).

У легких движків є одна проблема, ігри які створюються на них, багато в чому схожі. Для моєї гри потрібен просунутий движок, для того, щоб створити гру, не похожу на інші.

Дуже популярний движок Unity на якому я і буду створювати свій ігровий додаток. Це не самий легкий в використанні движок, але настільки популярний із-за своєї універсальності, що на нього є дуже багато навчальних матеріалів.Велика кількість компаній та незалежних команд займаються створенням комп'ютерних ігор. У розробці бере участь спеціалісти різних професій: програмісти, гейм-дизайнери, художники, QA спеціалісти та ін.

До розробки великих комерційних ігрових продуктів залучаються великі професійні команди 100+ фахівців. І коштуватимуть подібні проекти у розробці можуть десятки мільйонів доларів.

Однак успішні ігрові проекти можуть втілюватися і невеликими командами ентузіастів. Цьому сприяє присутність на ринку великої кількості відкритих і поширених платформ, якісні та практично безкоштовні двигуни, майданчики із залучення «народних» інвестицій (краудфандинг) та доступні канали розповсюдження.

Гравці - це основне джерело прибутку для ігрових продуктів. Але в сучасному світі найбільш активні гравці стали суттєвою рушійною силою у популяризації ігор та частково у розширенні контенту.

У зв'язку з широким поширенням кіберспорту найдосвідченіші гравці отримують можливість перекладати свої захоплення іграми на професійні рейки.

У світі проводиться чимало чемпіонатів, а ставки настільки високі, що результативні гравці можуть зробити на цьому хорошу кар'єру кіберспортсмена. Зараз кіберспорт - це ціла індустрія зі своєю інфраструктурою, фінансуванням та знаменитостями.

Найбільш віддані фанати ігор не тільки проводять за іграми багато років свого життя, а й найчастіше створюють різноманітний контент навколо улюбленої гри. Це можуть бути цілі сайти, малюнки, косплеї, журнали, відеопередачі, ігрові доповнення або навіть повноцінні ігри.

Більшість людей, які стикалися з комп'ютерними іграми, мають про них негативну або позитивну думку. Байдужих, загалом, небагато. І ось вже багато років не втихають суперечки між противниками та прихильниками комп'ютерних ігор, причому аргументи наводяться найрізноманітніші. Часом дивуєшся, які логічні ланцюжки вибудовують. І переважно обидві сторони наголошують на моменти, пов'язані з психікою, соціумом та особистісним розвитком.

Зрештою психологи все частіше почали говорити, що час вже звернути увагу не лише на негативні, а й на позитивні сторони захоплення відеоіграми, які зазвичай ігноруються, адже дослідники спочатку ставлять собі за мету пошук негативу. Адже ігри в останні роки сильно змінилися, стали набагато складнішими, реалістичнішими, більш соціально-орієнтованими. Повинно це вплинути. І результати деяких досліджень, проведених останні п'ять років, свідчать, що з іграми не так погано. Психологи відзначають, що сучасні відеоігри можуть сприяти розвитку соціальних та когнітивних навичок, дають новий емоційний досвід і навіть іноді сприяють загальному оздоровленню психіки.

1.2. Призначення розробки та галузь застосування

Ігровий застосунок “Rising of Slime: Reslimed” призначенний для розслаблення, зниження стресу та розвитку посидючості.

Цей застосунок пропонується використовувати у період часу, коли гравець має багато стрессу, йому скучно або є проблеми с концетрацією уваги.

Розроблений додаток дозволяє:

- Двигати головного героя.
- Знаходити та підбирати предмети.
- Купувати предмети.
- Відкрити меню паузи де можна:
  - Відновити ігрову сесію.
  - Відкрити настройки:
    - Змінити режим повноекранного екрану.
    - Змінити гучність музики.
    - Змінити гучність звуків.
  - Вийти в головне меню.
  - Вийти з гри.

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;

– завдання на кваліфікаційну роботу на тему “Розробка кросплатформного ігрового застосунку на основі технологічного стеку Unity/C#”

1.4. Постановка завдання

Завданням даної роботи є розробка мультиплатформного ігрового застосунку в жанрі “платформер” на основі технологічного стеку Unity/C#. Для прикладу будуть використані платформи Windows, Linux та MacOS.

Додаток повинен реалізувати наступні функції:

- Двигати головного героя.
- Знаходити та підбирати предмети.
- Купувати предмети.
- Відкрити меню паузи де можна:
  - Відновити ігрову сесію.
  - Відкрити настройки:
    - Змінити режим повноекранного екрану.
    - Змінити гучність музики.
    - Змінити гучність звуків.
  - Вийти в головне меню.
  - Вийти з гри.

Для досягнення поставленої мети необхідно:

- вивчити предметну галузь розв'язуваної задачі;

- створити алгоритм для реалізації поставленого завдання.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцева версія гри повинна дотримуватися ряду вимог, щоб забезпечити якість, функціональність та задоволення гравців. Основні вимоги до кінцевої версії гри включають наступне:

- Функціональність: Гра повинна працювати без помітних помилок, зависань або збоїв. Всі основні функції, такі як управління персонажем, інтерфейс користувача, механіки геймплею, місії або рівні повинні бути функціональними та забезпечувати задуманий геймплей.
- Графіка та звук: Візуальні та аудіоелементи гри повинні бути високої якості. Графічний дизайн, текстури, моделі персонажів та об'єктів, анімації та звукові ефекти повинні бути привабливими й передавати атмосферу гри.
- Геймплей: Гра повинна мати збалансований, захоплюючий та викликаючий інтерес геймплей. Вона повинна бути чіткою, веселою та викликати потребу гравця продовжувати грати.
- Керування: Керування грою повинно бути зручним і інтуїтивно зрозумілим для гравців. Використання контролерів, клавіатури або дотикового екрана, якщо це застосовується, повинно бути ефективним і надавати точний контроль над персонажем або об'єктами в грі.
- Стабільність та оптимізація: Гра повинна працювати стабільно на різних пристроях та платформах. Потрібно піклуватися про оптимізацію продуктивності, щоб гра запускалася плавно та мала стабільну кадрову частоту.

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

- Зрозумілий та зручний користувацький інтерфейс.
- Збереження прогресу та налаштувань користувача між сесіями.
- Можливість керування за допомогою клавіатури та мишки.
- Додаток матиме три екрани: головне меню, налаштування, ігровий єкран.

1.5.2. Вимоги до інформаційної безпеки

Ігри можуть використовувати персональні дані гравців для різних цілей, які можуть включати:

- Реєстрація та аутентифікація: Багато ігор вимагають, щоб гравці створили обліковий запис та залогінилися, використовуючи свої персональні дані, такі як ім'я, електронна пошта та пароль. Це дозволяє зберігати прогрес гравця, здійснювати оплату або надавати персоналізовані послуги.
- Збереження гри: Ігри можуть зберігати персональні дані гравців, такі як рівні прогресу, досягнення, статистика та налаштування, на серверах або локально на пристрої користувача. Це дозволяє гравцям продовжувати гру з того місця, де вони припинили, навіть якщо вони змінили пристрій або переінсталювали гру.
- Мультиплеєр та соціальні функції: Деякі ігри можуть використовувати персональні дані для побудови мережевих зв'язків між гравцями, дозволяючи їм грати разом, викликати друзів на поєдинки або обмінюватися повідомленнями. Також можуть бути використані соціальні медіа-інтеграції, щоб дозволити гравцям ділитися своїми досягненнями та скріншотами гри в соціальних мережах.
- Аналітика та персоналізація: Розробники ігор можуть аналізувати персональні дані гравців, щоб отримати інсайти щодо їх поведінки, вподобань та звичок. Це дозволяє вдосконалювати геймплей, рекомендувати контент, пропонувати персоналізовані пропозиції.

Для уникнення некоректної роботи програмного додатку та забезпечення контролю та цілісності даних ця гра ніяк не збирає інформацію про користувача та про його персональний комп’ютер. Тому нічого для забезпечення інформаційної безпеки робити не треба.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для роботи із застосунком «Rising of Slime: Reslimed» треба встановити його дистрибутив на операційній системі (ОС) користувача.

Відповідні характеристики необхідні для роботи додатку наведено нижче:

Для ОС Windows та Linux:

− ЦП з архітектурою x-86 або x-64;

− відеоадаптер з підтримкою DX10, DX11, DX12;

− 100мб+ вільного місця на диску;

− оперативна пам’ять 2 Гб.

Для MacOS:

− ЦП з архітектурою x-64;

− відеоадаптер з підтримкою Metal;

− 100мб+ вільного місця на диску;

− оперативна пам’ять 2 Гб.

1.5.4. Вимоги до інформаційної та програмної сумісності

Ігровий мультиплатформний застосунок «Rising of Slime: Reslimed» має бути розроблений на мові C# з використанням ігрового двигуна Unity, що забезпечить сумісність із майже усіма сучасними операційними системами, такими як Windows, MacOS та Linux.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом виконання даної кваліфікаційної роботи є розроблений ігровий мультиплатформний додаток «Rising of Slime: Reslimed» із використанням технологічного стеку Unity/C#.

В ході роботи програми повинні підтримуватися правила гри, а також зрозумілий для гравців ігровий процес. По завершенню роботи програми повинні бути збережені усі налаштування користувача.

Основним призначенням додатку є використання у мотиваційних цілях. Гра сприяє підвищенню рівня мотивації та розвитку посидливості.

Призначення розробленого додатка:

- Знизити стресс гравця.
- Розвити посидючість.

Розроблений додаток реалізує наступні функції:

- Двигати головного героя.
- Знаходити та підбирати предмети.
- Купувати предмети.
- Відкрити меню паузи де можна:
  - Відновити ігрову сесію.
  - Відкрити настройки:
    - Змінити режим повноекранного екрану.
    - Змінити гучність музики.
    - Змінити гучність звуків.
  - Вийти в головне меню.
  - Вийти з гри.

Для досягнення поставленої задачі розроблений програмний додаток підтримує виконання таких операцій:

- Зміна повноєкранного режиму.
- Зміна гучності музики.
- Зміна гучності звуків.
- Збереження ігрової сессії.
- Управління головним героєм.
- Підбирання предметів.
- Купівля предметів.
- Використання предметів.

2.2. Опис застосованих математичних методів

В даному ігровому додатку не використовуються складні математичні методи, а лише прості арифметичні операції, здебільшого з бібліотеки Mathf.

При розробці додатку були використанні такі математичні методи: Pow (возведення у ступінь), Clamp (обмежує значення між максимальним та мінімальним значеннями), Lerp (лінійна інтерполяція).

Оскільки для гри в жанрі «платформер» не потрібне застосування математичних методів, при розробці системи відображення та управління контенту програмного додатку математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проєктування

Unity відповідає за багато аспектів розробки гри, включаючи:

1. Графічний двигун.

Unity надає розробникам потужний графічний двигун, який дозволяє створювати реалістичну 2D та 3D графіку для ігор. Він підтримує рендеринг освітлення, тіней, текстур, анімацій та спеціальних ефектів.

2. Редагування об'єктів та сцен.

Unity має вбудовані редактори, які дозволяють розробникам створювати та редагувати об'єкти, сцени та рівні гри. Це включає переміщення, масштабування,

обертання об'єктів, налаштування їх властивостей та розміщення їх у просторі гри.

3. Керування ресурсами.

Unity дозволяє керувати ресурсами гри, такими як моделі персонажів, текстури, звукові файли та інші асети. Він надає зручний інтерфейс для імпорту, організації та використання цих ресурсів в грі.

4. Скриптінг та програмування.

Unity використовує мову програмування C#, що дозволяє розробникам створювати логіку гри, управляти поведінкою об'єктів, обробляти введення гравця та багато іншого. Це включає можливості створення скриптів для контролю руху, колізій, штучного інтелекту, анімацій та інших геймплейних елементів.

5. Фізика та колізії.

Unity має вбудовану систему фізики, яка дозволяє моделювати реалістичну фізичну взаємодію між об'єктами в грі. Вона включає симуляцію гравітації, руху, столкнувся

Основні функції та відповідальності C# при розробці гри на Unity включають:

1. Логіка гри.

C# використовується для написання коду, який керує логікою гри. Розробники можуть використовувати C# для створення скриптів, які визначають поведінку персонажів, управляють штучним інтелектом, обробляють взаємодію гравця з об'єктами та реалізують різні геймплейні механіки.

2. Управління ресурсами.

C# дозволяє розробникам керувати ресурсами гри, такими як моделі, текстури, звукові файли та інші асети. Вони можуть використовувати C# для завантаження, збереження, організації та використання цих ресурсів в грі.

3. Інтерфейс користувача.

C# використовується для розробки коду, який керує інтерфейсом користувача гри. Розробники можуть створювати скрипти, які відповідають за

відображення графічних елементів, управління кнопками, меню, інтерактивними формами та іншими елементами інтерфейсу гри.

4. Взаємодія з іншими системами.

C# дозволяє взаємодіяти з різними системами та сервісами в межах гри. Розробники можуть використовувати C# для реалізації мережевого взаємодії, роботи з базами даних, зовнішніми API, системами штучного інтелекту, фізикою, анімацією та багатьма іншими аспектами розробки гри.

5. Розширення та модифікація.

За рахунок повної інтеграції з Unity мова C# надає певні можливості розширення та модифікації функціональності двигуна гри.

2.4. Опис використаних технологій та мов програмування

Unity - це популярний і потужний ігровий двигун та розробницька платформа, що використовується для створення різноманітних ігрових додатків, від ігор до симуляцій та віртуальної реальності. Він був створений компанією Unity Technologies і має широкий спектр можливостей, які дозволяють розробникам реалізувати свої творчі ідеї.

Одним з головних переваг Unity є його кросплатформеність. Він підтримує багато платформ, включаючи Windows, macOS, Linux, iOS, Android, Xbox, PlayStation та багато інших. Це дозволяє розробникам створювати ігри, які працюють на різних пристроях і операційних системах, збільшуючи свою аудиторію та досягаючи широкого кола гравців.

Unity надає розробникам доступ до потужних інструментів та ресурсів, що полегшують процес розробки. Його візуальний редактор дозволяє швидко створювати та редагувати ігрові об'єкти, налаштовувати їх поведінку та взаємодію. Він також має широкий набір компонентів та скриптів, які допомагають реалізувати різноманітні ігрові механіки та функціональність.

Unity підтримує різні мови програмування, включаючи C#, що є основною рекомендованою мовою для розробки в Unity. Це міцно інтегрована мова з

потужними можливостями, яка дозволяє розробникам контролювати логіку гри, створювати взаємодію з об'єктами, реалізовувати штучний інтелект та багато іншого.

Unity також підтримує велику спільноту розробників, де можна знайти велику кількість документації, плагінів, готових рішень та підручників. Це допомагає розробникам вирішувати проблеми, вивчати нові можливості та спілкуватися з колегами.

За допомогою Unity можна створювати ігри різних жанрів, від 2D платформерів до реалістичних 3D шутерів. Він підтримує різні фізичні двигуни, системи частинок, освітлення та ефекти, що дозволяють створювати вражаючі візуальні ефекти та реалістичні світи.

Крім того, Unity має інтегровану систему аналітики, що дозволяє розробникам збирати дані про гравців, аналізувати їх поведінку та вдосконалювати геймплей та монетизацію своїх ігор.

Загалом, Unity є потужним та універсальним інструментом для розробки ігор, який надає розробникам велику свободу творчості та допомагає їм реалізувати свої ідеї у віртуальному світі.

C# (вимовляється як "Сі шарп") - це сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона є однією з основних мов програмування для розробки програмного забезпечення на платформі .NET.

C# має багато переваг і широке застосування у різних сферах розробки програмного забезпечення. Ось кілька ключових особливостей C#:

1. Об'єктно-орієнтованість.

C# підтримує принципи об'єктно-орієнтованого програмування, такі як спадкування, поліморфізм, інкапсуляція та абстракція. Це дозволяє розробникам створювати модульний, гнучкий та легкозмінний код.

2. Мультиплатформеність.

Мова C# може бути використана для розробки програмного забезпечення на різних платформах, включаючи Windows, macOS і Linux. Вона інтегрована з платформою .NET, що забезпечує кросплатформенність і переносимість коду.

3. Потужна бібліотека класів.

C# має велику бібліотеку класів, включаючи базові класи, колекції, ввід-вивід, роботу з мережею, роботу з базами даних та багато іншого. Це спрощує розробку програм і дозволяє ефективно використовувати готові компоненти.

4. Підтримка асинхронного програмування.

C# має вбудовану підтримку асинхронного програмування, що дозволяє ефективно використовувати багатопотоковість та забезпечує покращену продуктивність програм.

5. Легкість використання.

C# має чистий і зрозумілий синтаксис, який наближений до мови людей. Він надає розробникам широкий набір зручних функцій, таких як автоматична підказка коду, збирач сміття і перевірка типів в реальному часі.

6. Широке застосування.

C# використовується для розробки різноманітних програмних продуктів, включаючи веб-додатки, настільні програми, мобільні додатки, ігри та послуги хмарних обчислень.

Загалом, C# є потужною мовою програмування з великим набором функцій і широким спектром застосувань. Вона дозволяє розробникам ефективно створювати складні програми, забезпечуючи швидкість, безпеку і гнучкість розробки.

JSON (JavaScript Object Notation) - це легкий, текстовий формат обміну даними, що використовується для передачі структурованих інформаційних об'єктів між різними програмами. Він є популярним форматом для зберігання та передачі даних веб-додатками і служить альтернативою XML формату.

Основні риси JSON:

1. Простота читання та написання.

JSON представляє дані у зрозумілому для людини форматі, який легко читати і редагувати. Він використовує розряджені тексти з легким синтаксисом, що спрощує роботу з даними.

2. Структуровані об'єкти.

JSON дозволяє створювати структуровані об'єкти, які складаються з пар ключ-значення. Це дозволяє організувати дані у логічні групи та забезпечує зручний доступ до них.

3. Підтримка різних типів даних.

JSON підтримує різні типи даних, включаючи рядки, числа, булеві значення, масиви, об'єкти та значення null. Це дозволяє представляти різнорідні дані і зберігати структуру даних.

4. Незалежність від платформи.

JSON є незалежним від платформи форматом, що означає, що дані, представлені у форматі JSON, можуть бути оброблені різними мовами програмування та різними платформами.

5. Розширюваність.

JSON дозволяє використовувати власні типи даних та розширювати формат за допомогою власних атрибутів та розширень. Це дозволяє пристосовувати формат під конкретні потреби додатка.

JSON легко обробляється за допомогою багатьох мов програмування, оскільки багато мов мають бібліотеки або вбудовані функції для роботи з JSON. Він широко використовується для передачі даних між клієнтом та сервером, зберігання конфігураційних файлів, обміну даними між різними додатками та багато іншого.

2.5. Опис структури програми та алгоритмів її функціонування

Для демонстрації варіантів сценаріїв використання користувачем програмного додатку та послідовностей ігрового процесу нижче наведені ULM діаграми варіантів використання ігрового процессу (рис.2.1-2.2):

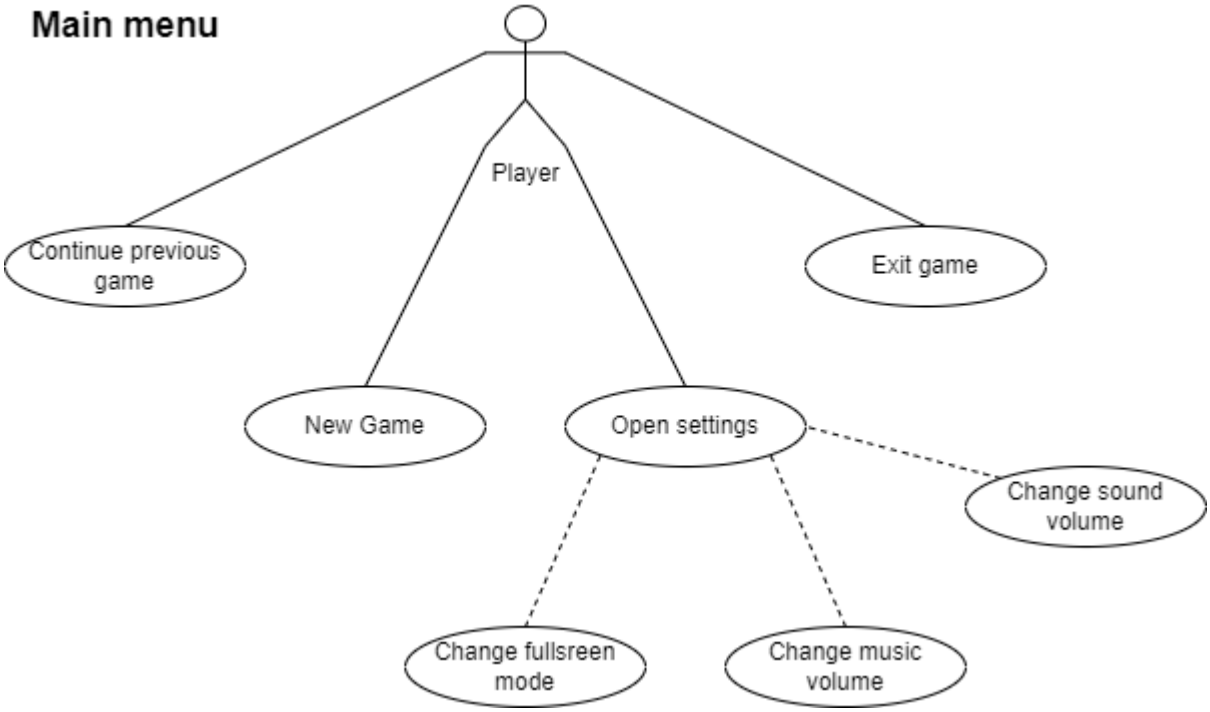

Рис. 2.1. Діаграма варіантів використання для головного меню

Гравець з головного меню може почати нову гру, продовжити минулу ігрову сесію, відкрити налаштування, включити або виключити повноекранний режим, вийти з гри.

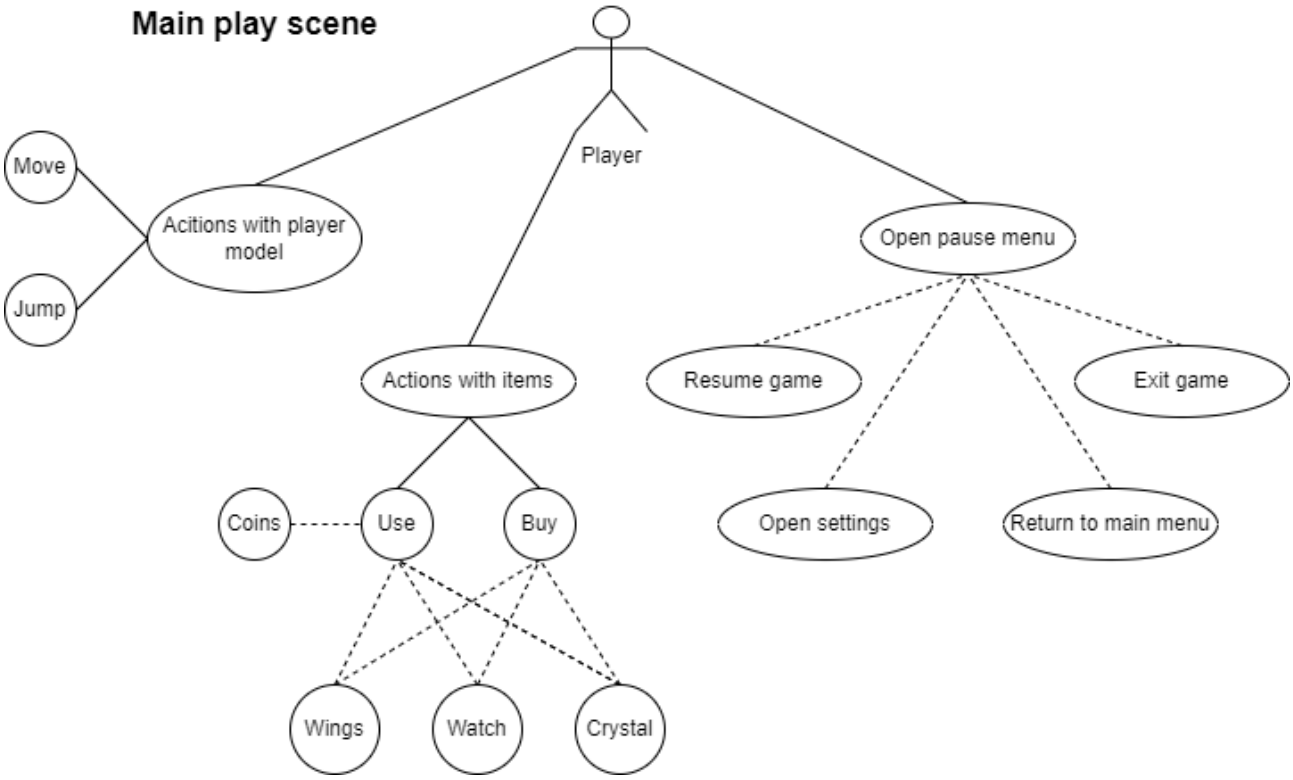

Рис. 2.2. Діаграма варіантів використання для основного ігрового екрану

На головній ігровій сцені гравець може:

- Двигати головного героя.
- Знаходити та підбирати предмети.
- Купувати предмети.
- Відкрити меню паузи де можна:
  - Відновити ігрову сесію.
  - Відкрити настройки:
    - Змінити режим повноекранного екрану.
    - Змінити гучність музики.
    - Змінити гучність звуків.
  - Вийти в головне меню.
  - Вийти з гри.

При розробці ігрового додатку мною були прийняті такі патерни проектування для створення функціоналу програми:

Сінглтон - це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, і надає до нього глобальну точку доступу. Всі реалізації сінглтона зводяться до того, щоб приховати конструктор за замовчуванням і створити публічний статичний метод, який і контролюватиме життєвий цикл об'єкта-сінглтона. Якщо у вас є доступ до класу сінглтона, значить, буде доступ і до цього статичного методу. З якої точки коду ви б його не викликали, він завжди віддаватиме той самий об'єкт.

Unity використовує свою систему івентів, який має строгу послідовність визову своїх івентів (рис. 2.3.)

Система івентів (Event System) в Unity є потужним механізмом для обміну та спілкування між різними об'єктами у сцені гри. Вона базується на паттерні проектування "Спостерігач" (Observer) і дозволяє реалізувати реакцію на події, які відбуваються в грі.

Основні компоненти системи івентів в Unity включають:

1. Event.

Це подія або сигнал, який може бути викликаний або сприйнятий іншими об'єктами. Події можуть мати різні типи, наприклад, натискання кнопки, зіткнення об'єктів, зміна стану тощо.

2. Event Listener.

Це компонент, який слухає певні події і виконує певні дії, коли такі події стають активними. Він підписується на певні події і реагує на них, виконуючи код або викликаючи функції.

3. Event Trigger.

Це компонент, який викликає певні події в реакцію на визначені умови або дії. Він може спрацювати при натисканні кнопки, зіткненні об'єктів, коли об'єкт досягає певної позиції тощо.

4. Event Handler.

Це код або функція, яка виконується при спрацюванні певної події. Це місце, де ви можете виконати певні дії або змінити стан гри, відповідно до події.

Система івентів дозволяє розділити функціональність та зменшити залежність між об'єктами у грі. Замість прямого виклику методів об'єктів, вони можуть відправляти або сприймати івенти, що спрощує розвиток, збільшує його гнучкість та робить код більш модульним.

Система івентів в Unity є важливим інструментом при розробці гри, дозволяючи ефективно керувати взаємодією об'єктів та реалізувати складну поведінку без прямого зв'язку між ними.

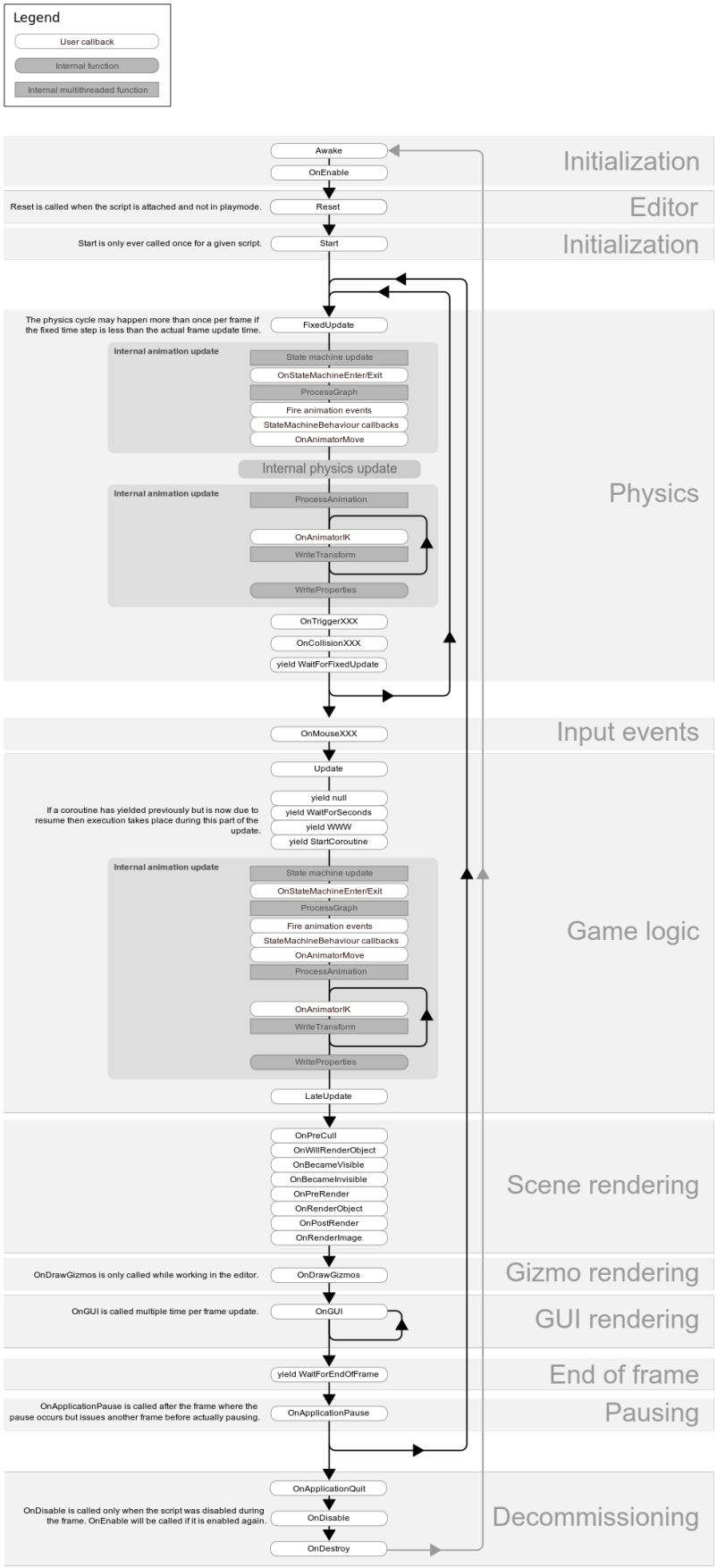

Рис. 2.3. Порядок викликів вбудованих методів івентів

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні данні гра отримує від клавіатури та миші гравця та з файлу збережених даних минулої ігрової сесії.

Вихідні данні, у свою чергу, це файли збереження ігрового процессу. В них збережено:

- Позиція гравця.
- Злякані кроти.
- Підібрані монети.
- Наявні предмети.

2.7. Опис розробленої системи

2.7.1. Використані технічні засоби

Розроблений додаток можна встановити на ноутбуці, ПК, смартфоні та планшеті різноманітних видів та конфігурацій під управлінням різних операційних систем але треба дотримуватись вимог до складу та параметрів технічних засобів які описано у п.1.5.3.

Розробка та тестування додатку проводилось на персональному компьютері з наступними характеристиками:

− операційна система: Windows 10 Home;

− ЦП: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz;

− відеокарта: NVIDIA GeForce MX150;

− аудіокарта: Realtek High Defenition Audio;

− накопичувачи: SATA SSD 256 GB x2;

− оперативна пам’ять: 12 Гб.

А також засоби вводу-виведення: монітор з роздільною здатністю 1920*1080, клавіатура, комп’ютерна миша.

2.7.2. Використані програмні засоби

Visual Studio Code (VS Code) використовується як інтегроване середовище розробки (IDE) для розробки гри на Unity.

VS Code - це безкоштовний, легкий у використанні та розширюваний текстовий редактор, розроблений компанією Microsoft. Він став популярним серед розробників завдяки своїй потужності та гнучкості. VS Code надає розширену підтримку для багатьох мов програмування та технологій, включаючи HTML, CSS, JavaScript, Python, C#, Java та багато інших. Редактор має широкий набір функцій, таких як підказка коду, вбудована консоль, налаштовувані команди, система керування версіями та інтегрована підтримка розширень, що дозволяє розробникам налаштовувати та розширювати редактор під свої потреби. Завдяки своїм можливостям, VS Code став одним з найпопулярніших інструментів для розробки програмного забезпечення, надаючи зручне та продуктивне середовище для написання коду, роботи з проектами та спілкування зі спільнотою розробників.

Основні функції та відповідальності VS Code при розробці гри на Unity включають:

1. Редактор коду.

VS Code надає потужний редактор коду, який підтримує мову програмування C#, що використовується для розробки гри на Unity. Він має функціональність автодоповнення, підсвічування синтаксису, перехід до визначення, рефакторинг та інші інструменти, що полегшують написання та редагування коду.

2. Інтеграція з Unity.

VS Code може бути налаштований для інтеграції з Unity, що дозволяє здійснювати розробку гри безпосередньо з IDE. Розробники можуть виконувати запуск гри, керувати сценами та ресурсами, відлажувати код та отримувати повідомлення про помилки прямо в VS Code.

3. Розширення та плагіни.

VS Code підтримує широкий спектр розширень та плагінів, які полегшують розробку гри на Unity. Ці розширення можуть додати додаткові функціональність, таку як підтримка специфічних форматів файлів, інструменти для роботи з асетами, підсвічування Unity API та багато іншого.

4. Версійний контроль.

VS Code добре інтегрується з системами версійного контролю, такими як Git, що дозволяє розробникам легко відстежувати зміни в коді, співпрацювати з іншими розробниками та керувати різними версіями гри.

5. Налаштування та розширені можливості.

VS Code має широкі можливості налаштування, що дозволяє розробникам налаштувати редактор під свої потреби. Він також підтримує встановлення розширень для додаткової функціональності, такої як сніппети коду, дебагери, лінтери та багато іншого, що полегшує процес розробки гри на Unity.

Отже, VS Code відповідає за забезпечення потужного редактора коду, інтеграцію з Unity, розширення та плагіни, версійний контроль та надає розробникам розширені налаштування та можливості для зручної розробки гри на Unity.

Unity відповідає за багато аспектів розробки гри, включаючи:

1. Графічний двигун.

Unity надає розробникам потужний графічний двигун, який дозволяє створювати реалістичну 2D та 3D графіку для ігор. Він підтримує рендеринг освітлення, тіней, текстур, анімацій та спеціальних ефектів.

2. Редагування об'єктів та сцен.

Unity має вбудовані редактори, які дозволяють розробникам створювати та редагувати об'єкти, сцени та рівні гри. Це включає переміщення, масштабування, обертання об'єктів, налаштування їх властивостей та розміщення їх у просторі гри.

3. Керування ресурсами.

Unity дозволяє керувати ресурсами гри, такими як моделі персонажів, текстури, звукові файли та інші асети. Він надає зручний інтерфейс для імпорту, організації та використання цих ресурсів в грі.

4. Скриптінг та програмування.

Unity використовує мову програмування C#, що дозволяє розробникам створювати логіку гри, управляти поведінкою об'єктів, обробляти введення гравця та багато іншого. Це включає можливості створення скриптів для контролю руху, колізій, штучного інтелекту, анімацій та інших геймплейних елементів.

5. Фізика та колізії.

Unity має вбудовану систему фізики, яка дозволяє моделювати реалістичну фізичну взаємодію між об'єктами в грі. Вона включає симуляцію гравітації та руху персонажу гри.

Adobe Photoshop - це найпопулярніший графічний редактор і програмне забезпечення для обробки та редагування фотографій, графічного дизайну та створення мистецтва. Завдяки своїй потужній набору інструментів, Photoshop надає безліч можливостей для створення творчих та професійних робіт. Він дозволяє ретушувати фотографії, видаляти недоліки шкіри, налаштовувати кольорову гаму, виправляти експозицію та додавати спеціальні ефекти. Крім того, Photoshop має потужні інструменти для створення графічного дизайну, таких як створення логотипів, банерів, постерів та ілюстрацій. Він підтримує роботу з шарами, масками, фільтрами, текстом та багатьма іншими функціями, що дає безмежні можливості для творчого виразу. Adobe Photoshop є незамінним інструментом для фотографів, дизайнерів, художників та всіх, хто бажає створювати і редагувати графічні зображення з професійною якістю і безкомпромісною творчістю.

Adobe Photoshop відіграє важливу роль при розробці гри на Unity, виконуючи наступні функції:

1. Графічний дизайн.

Adobe Photoshop є потужним інструментом для створення та редагування графічних елементів, таких як фони, спрайти, текстури, іконки, логотипи та інші візуальні компоненти гри. Він надає розробникам можливість створювати привабливі та естетично збалансовані графічні ресурси, які використовуються в грі.

2. Робота з текстом.

Photoshop дозволяє редагувати та стилізувати текстові елементи, які використовуються в грі. Розробники можуть створювати заголовки, надписи, кнопки та інші текстові елементи з використанням різноманітних шрифтів, ефектів та форматування.

3. Оптимізація та редагування ресурсів.

Photoshop надає можливість оптимізувати графічні ресурси для використання в грі, такі як стиснення текстур, видалення зайвого шуму, зміна розмірів зображень та інше. Він також дозволяє редагувати та змінювати ресурси для досягнення бажаного вигляду та ефектів.

4. Анімація.

Adobe Photoshop підтримує можливість створювати прості анімації, такі як анімація спрайтів чи переходи між станами. Це може бути корисно для створення рухомих частин, ефектів переходу або анімованих інтерфейсів в грі.

5. Робота з шарами та масками.

Photoshop надає зручні інструменти для роботи з шарами та масками, що дозволяє розробникам більш гнучко керувати графічними елементами гри.

GitHub - це веб-платформа для розміщення та спільної розробки програмного забезпечення з використанням системи керування версіями Git. Вона надає розробникам можливість створювати репозиторії, завантажувати свій код, ведення відстеження змін, співпрацювати з командою розробників та керувати процесом розробки. GitHub дозволяє легко співпрацювати над проектами, надавати коментарі, створювати гілки для розробки нових функцій

та об'єднання змін в одині головний код (мерджі). Крім того, GitHub надає інструменти для управління завданнями (issues), автоматизації процесу розробки (actions) та відстеження питань (pull requests). Ця платформа також дозволяє спільноті розробників взаємодіяти, знайти відкриті проекти, внести свій внесок до великих відкритих проектів та навчитися від інших розробників. Завдяки своїй популярності, GitHub став одним з найважливіших інструментів для спільної розробки програмного забезпечення, забезпечуючи простоту, зручність та надійність у керуванні проектами і спільною роботою над кодом.

GitHub відіграє важливу роль при розробці гри на Unity, виконуючи наступні функції:

1. Керування версіями.

GitHub забезпечує систему контролю версій, що дозволяє розробникам відстежувати та зберігати всі зміни в коді та ресурсах гри. Він забезпечує можливість створювати гілки розробки, злиття (merge) змін, відкат до попередніх версій та співпрацювати з іншими розробниками.

2. Спільна робота.

GitHub створює зручні умови для спільної роботи над проектом. Він дозволяє кільком розробникам працювати над одним і тим же кодом одночасно, вносити зміни та коментувати код. Це полегшує комунікацію та співпрацю між розробниками гри.

3. Віддалений доступ.

GitHub надає можливість зберігати проект в хмарному сховищі, що дозволяє розробникам отримувати віддалений доступ до коду та ресурсів гри. Це зручно для роботи з декількома комп'ютерами або співпраці з розробниками, які знаходяться в різних місцях.

4. Контроль помилок.

GitHub надає можливість створювати та відстежувати проблеми (issues) та помилки в проекті. Розробники можуть створювати звіти про помилки, пропонувати виправлення та відстежувати їх статус. Це допомагає впорядкувати процес виправлення помилок та покращує якість гри.

5. Документація та спільнота.

GitHub надає можливість створювати та публікувати документацію до проекту. Розробники можуть додавати пояснення, приклади коду та іншу інформацію, що полегшуэ розуміння проекту для інших розробників або спільноти. Крім того, GitHub має активну спільноту розробників, де можна обговорювати питання, ділитися досвідом та знаходити відповіді на питання.

Fruity Loops Studio - FL Studio (раніше відомий як FruityLoops) - це потужна програма для створення музики та аудіо-продакшну. Вона надає широкий набір інструментів та можливостей для композиції, запису, аранжування, зведення та мастерингу музичних треків. FL Studio має інтуїтивно зрозумілий інтерфейс, що дозволяє як початківцям, так і досвідченим музикантам легко працювати з програмою. Вона включає в себе велику бібліотеку звуків, ефектів та інструментів, таких як синтезатори, семплери, секвенсори, ефектори та багато іншого. FL Studio підтримує різні стилі музики, від електронної до хіп-хопу, від поп-музики до року, і дозволяє створювати професійні звукозаписи з високою якістю звуку. Завдяки своїй гнучкості та розширюваності, FL Studio стала популярною серед музикантів і продюсерів усього світу, надаючи їм засоби для творчого виразу та реалізації своїх музичних ідей.

FL Studio є професійним програмним забезпеченням для створення музики та аудіо-записів. При розробці гри на Unity, FL Studio відіграє наступні ролі та виконує функції:

1. Створення та редагування музики.

FL Studio надає потужні інструменти для створення музики, компонування мелодій, створення ритмів та аранжування звукових треків. Розробники можуть використовувати FL Studio для створення унікальної та настроюваної музичної підкладки, яка відповідає настрою та атмосфері гри.

2. Запис та редагування звукових ефектів.

FL Studio дозволяє розробникам записувати та редагувати звукові ефекти, такі як звукові елементи, голоси персонажів, звуки оточуючого середовища та

інші аудіо-елементи, які використовуються в грі. Він надає широкі можливості для обробки звукового матеріалу, включаючи зміну тембру, еквалайзер, ефекти віддаленості та багато іншого.

3. Імпорт та експорт аудіо-файлів.

FL Studio підтримує різноманітні формати аудіо-файлів, що дозволяє легко імпортувати та експортувати звукові треки, звукові ефекти та інші аудіо-ресурси в форматі, який відповідає вимогам гри. Це спрощує інтеграцію музики та звукових ефектів у процес розробки гри.

4. Міксування та мастеринг.

FL Studio забезпечує можливість міксування та мастерингу звукових треків та ефектів. Розробники можуть налаштовувати баланс гучності, панорамування, ефекти простору та інші аспекти звукового доріжки, щоб створити професійний та високоякісний звук для гри.

5. Робота з MIDI.

FL Studio підтримує MIDI-інтерфейс, що дозволяє розробникам використовувати MIDI-клавіатури, контролери та інші пристрої для створення музики та керування звуковими ефектами в грі. Це дозволяє більш точно контролювати звукові параметри та створювати виразні музичні елементи.

Загалом, FL Studio є важливим інструментом для розробки аудіо-контенту у грі на Unity, дозволяючи розробникам створювати унікальну та високоякісну музику та звукові ефекти, які підсилюють враження від гри.

2.7.3. Виклик та завантаження програми

Для комп’ютерів на ОС Windows, дистрібутив завантажується через інтернет та розпаковується через програму архіватор (рис.2.5). Після цього гру можна запустити через .exe файл.

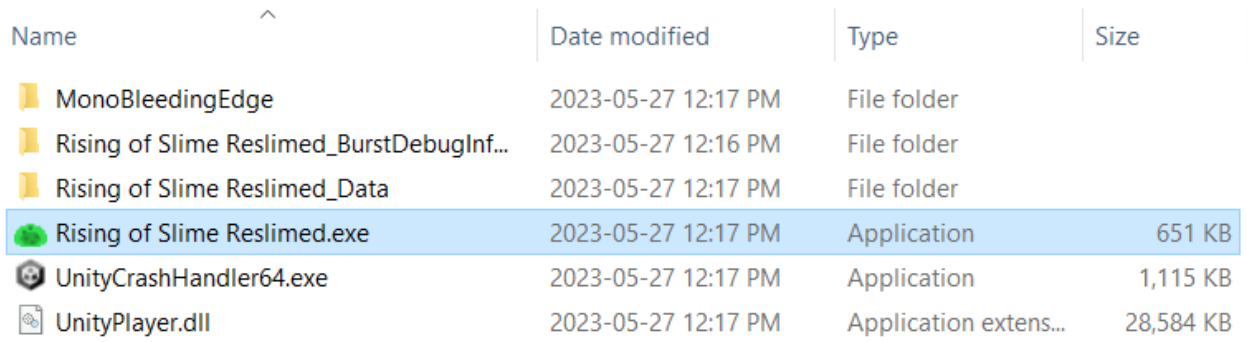

Рис. 2.5. Приклад папки з додатком після розпакування архіву для Windows

На рис.2.6. наведено файли для запуску розробленого застосунку на ОС MacOS.

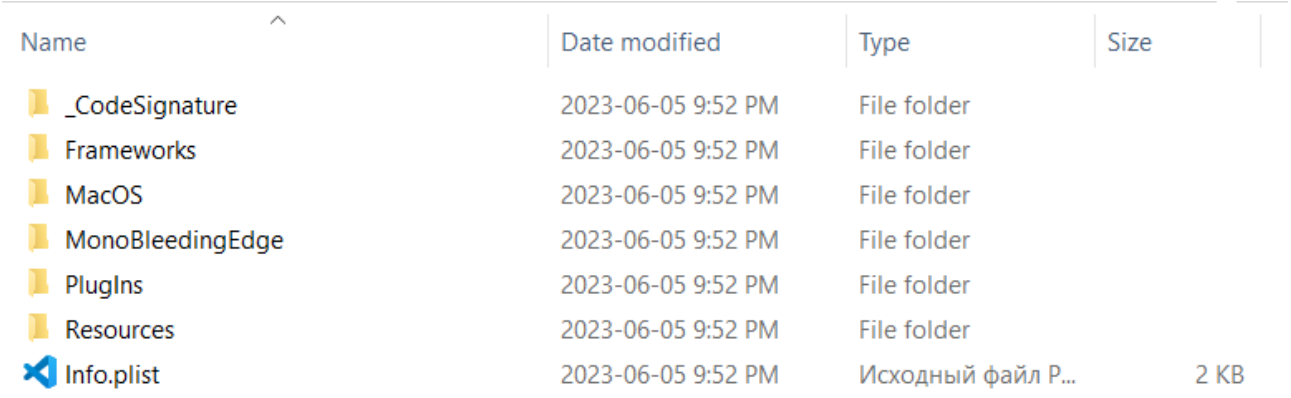

Рис. 2.6 Приклад папки з додатком після розпакування архіву для MacOS

На рис.2.7. наведено файли для запуску розробленого застосунку на ОС Linux.

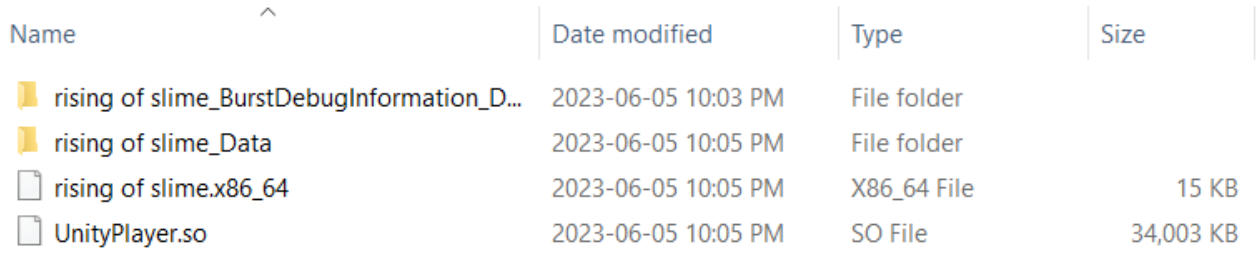

Рис. 2.7 Приклад папки з додатком після розпакування архіву для Linux.

2.7.4. Опис інтерфейсу користувача

Після запуску гри, гравець бачить головне меню (рис.2.8). Тут він може продовжити минулу ігрову сесію, розпочати нову, перейти в меню налаштувань, та вийти з гри.

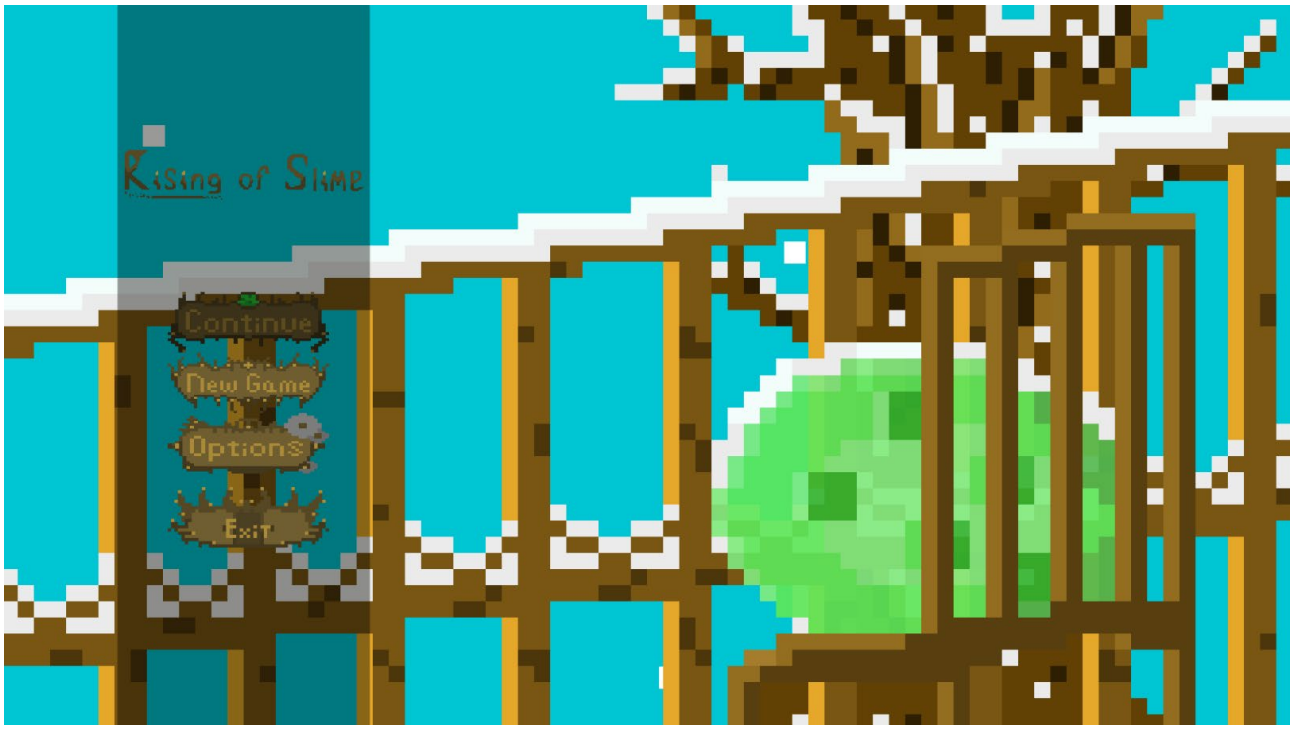

Рис. 2.8. Головне меню

В меню налаштувань (рис.2.9) можна включити, або виключити повноекранний режим, змінити гучність звуків, та музики.

Рис. 2.9. Меню налаштувань

Після вибору нової гри, гравець спочатку бачить невелику катсцену (рис.2.10), яка розказує початок невеликого сюжету гри, щоб гравець розумів хто він, що повинен робити, та навіщо.

Рис. 2.10 Катсцена

Після катсцени, гравець получає контроль над персонажем та може взаємодіяти с оточуванням. Гравець спочатку бачить першу локацію гри (рис.2.11).

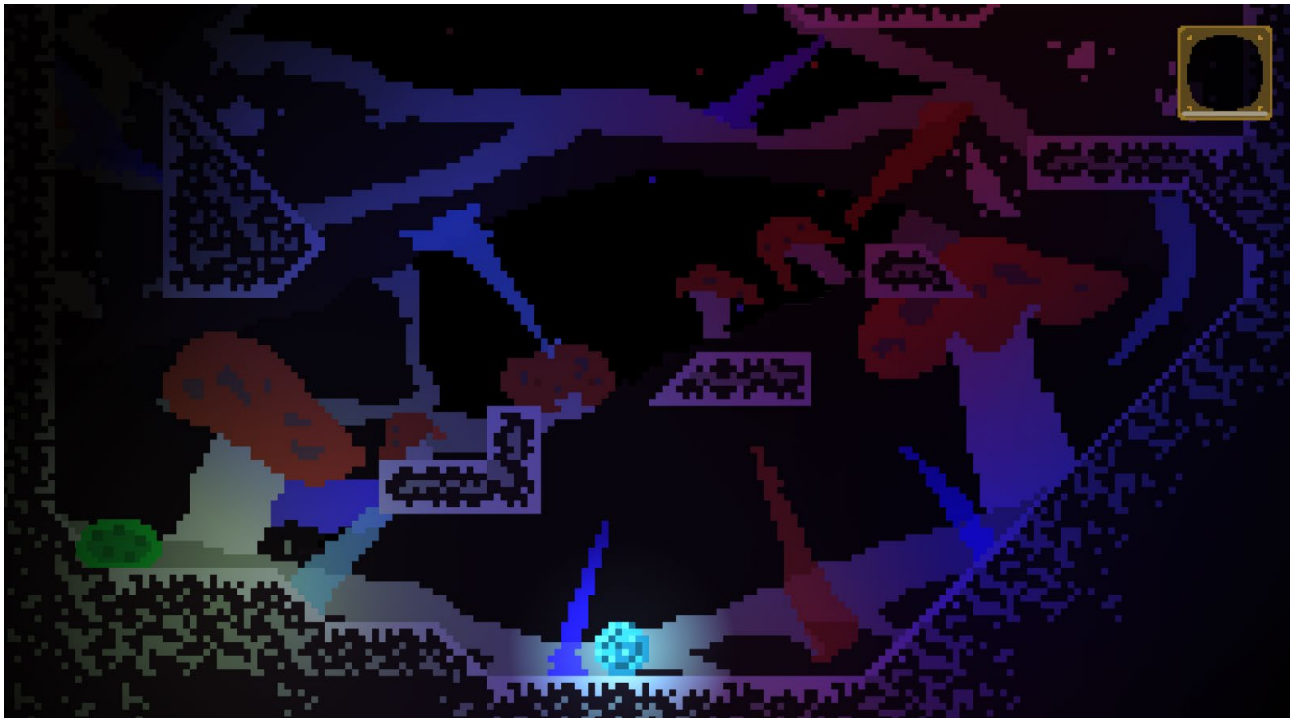

Рис. 2.11 Початок гри

Гравець може підбирати предмети. (рис.2.12)

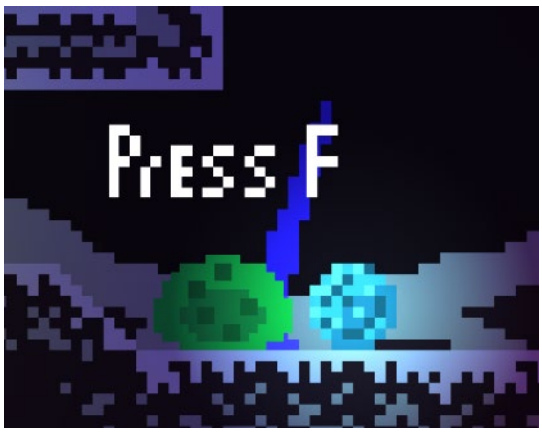

Рис. 2.12 Підбирання сфери

В новій грі, гравцю показуються підказки (рис.2.13), які пояснюють як грати в гру.

Рис. 2.13 Підказка

В грі 5 локацій. В кожній локації добавляється по одній новій механіці. Також нову механіку у вигляді предмету можна знайти або купити.

Усі механіки гри наведено нижче:

- Крюки (рис.2.14).

Зачепившись за крюки, гравець може пригнути, але триматися за крюк можна лише деякий час.

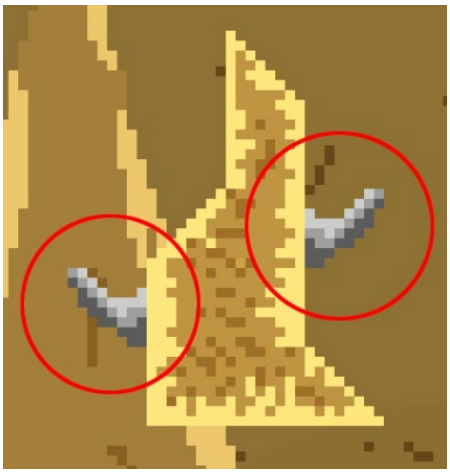

Рис. 2.14 Крюки

- Сфера (рис.2.15).

  Підібрав сферу та активувавши її, гравець може підсвітити своє оточення.

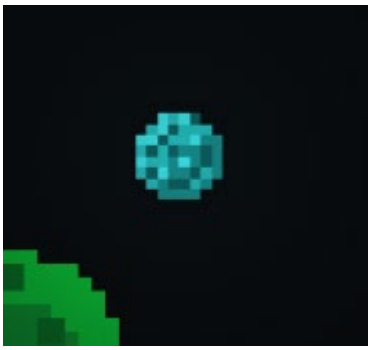

Рис. 2.15 Сфера

- Платформи, що ламаються (рис.2.16).

  Вставши на таку платформу, через невеликий час вона зламається. Але через декілька секунд знову відбудується.

Рис. 2.16 Платформи, що ламаються

- Крюки, що ламаються (рис.2.17).

  Працює як звичайний крюк, але ламається після стрибка, вони відбудовується через декілька секунд.

Рис. 2.17 Крюки, що ламаються

- Платформи, що двигаються (рис.2.18).

  Коли гравець становиться на таку платформу, вона починає рухатись по своєму маршруту.

Рис. 2.18 Платформи, що двигаються

- Крила (рис.2.19).

  Активувавши крила (рис.2.20), гравець може підлетіти вище.

Рис. 2.19 Крила

Рис 2.20 Активовані крила

• Часи (рис.2.21).

Активувавши часи (рис.2.22), час сповільнюється на деякий час. Усе, окрім гравця сповільнюється.

Рис. 2.21 Часи

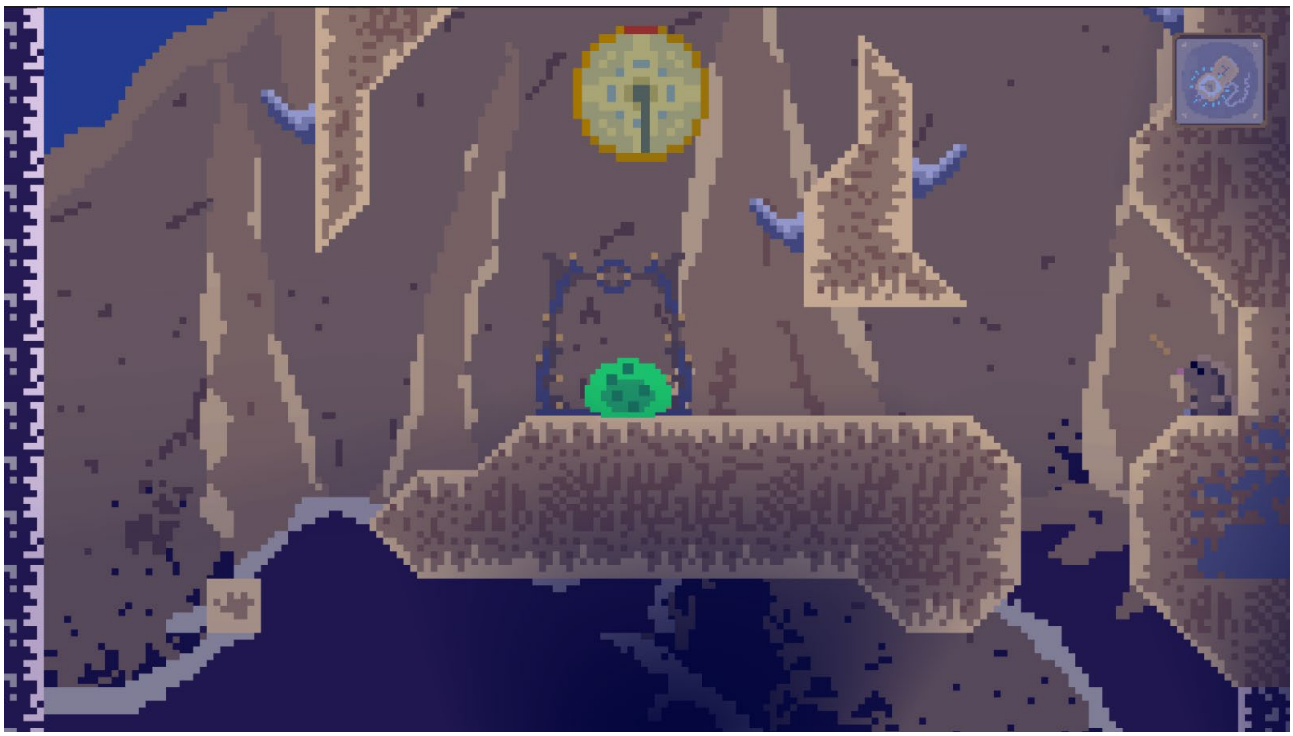

Рис. 2.22 Активовані часи

- Портали з кристалами (рис.2.23).

На початку кожної глави, крім початкової, стоїть портал (рис.2.24).

Купивши кристали, їх можна використати для активації порталів (рис.2.25).

Портали дають телепортуватися між ними.

Рис. 2.23 Кристал

Рис 2.24 Не активований портал

Рис. 2.25 Активований портал

- Покупка за монети (рис.2.27).

Монети можна знаходити в ігровому світі (рис.2.26).

Рис. 2.26 Монета

Рис 2.27 Меню покупки за монети

- Кроти (рис.2.28).

Якщо підійти до крота, то він злякається, втече та залишить монету.

Рис. 2.28 Крот

Перша локація «Яма» (рис.2.29). З неї гравець починає нову ігрову сессію. В ній спочатку гравець бачить катсцену, після неї підсказки, які с проходженням гри, розказують більше про те, як грати в гру.

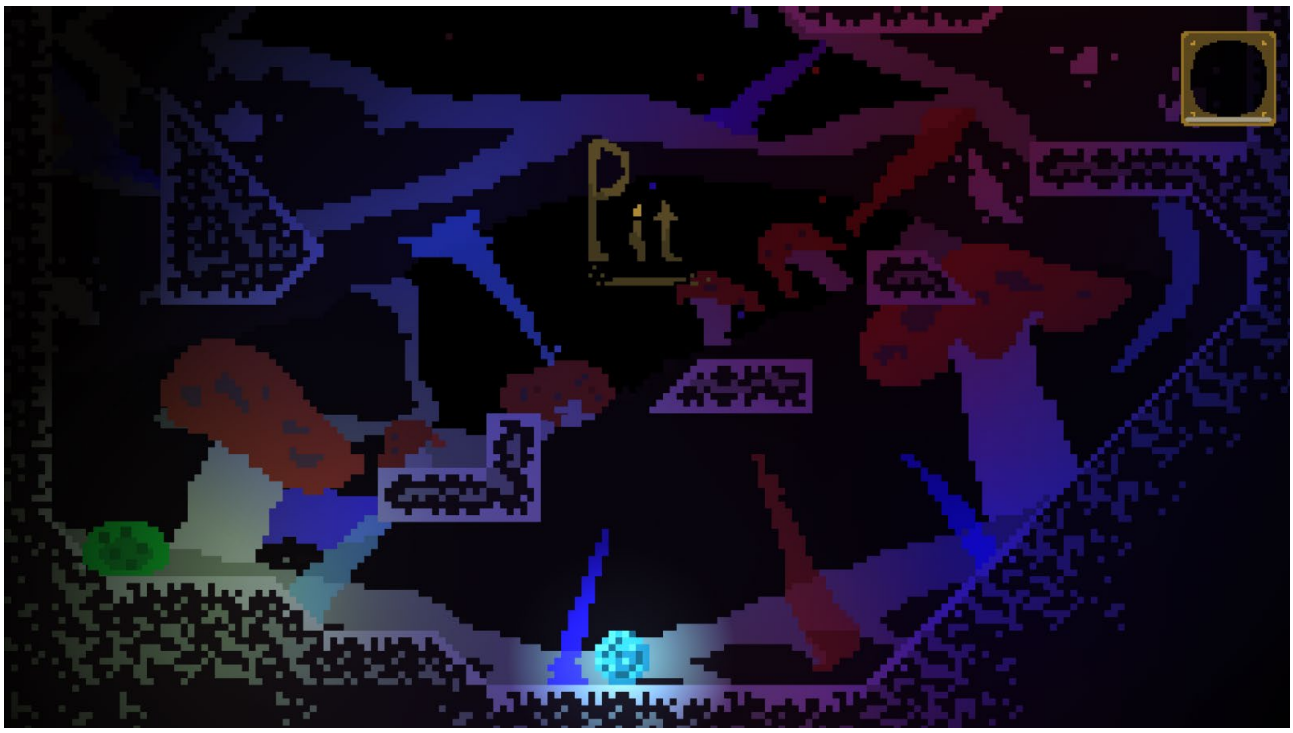

Рис. 2.29 Локація «Яма»

В другій локації «Пустиня» (рис.2.30) добавляються крюки. Це доповнення дає змогу спроектувати уровні більш цікаво та складно. Тепер ціна помилки для гравця буде вище. Якщо не встигнути відреагувати на момент зацепу головного героя за крюк, то гравець впаде доволі низько майже завжди.

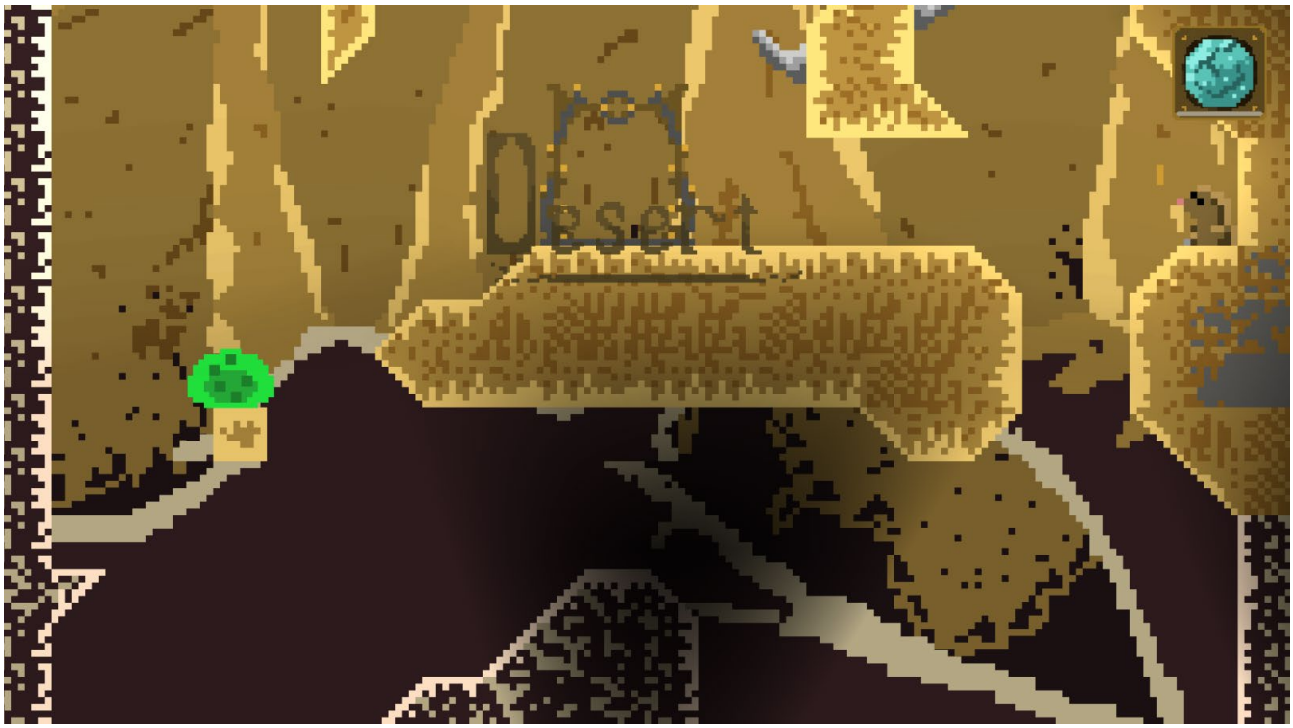

Рис. 2.30 Локація «Пустиня»

В третій локації «Ліс» (рис.2.31) добавляються крюки та платформи, що ламаються, покупка предметів. Якщо гравець не ігнорував більш складні участки рівнів, тоді в нього буде достатньо монет, щоб купити собі декілька предметів, які будут облегшувати йому подальше проходження гри.

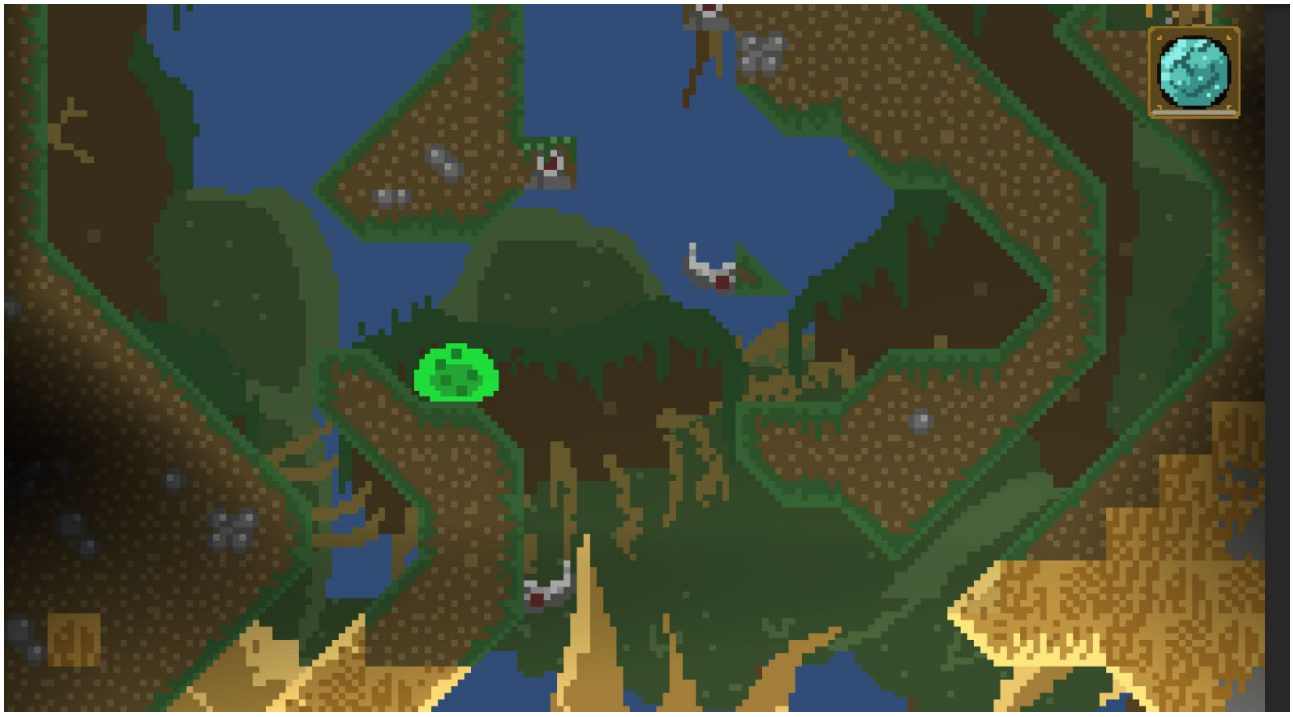

Рис. 2.31 Локація «Ліс»

В четвертій локації «Покинута вежа» (рис.2.32) добавляються платформи, що рухаються. Гравець не знає заздалегіть, куди вони будут двигатися, тому, якщо гравець буде спішити, то є велики шанс зробити помилку. Також цей рівень темний і це створює більш складні ситуації, бо гравець не може бачити далеко від себе.

І остання, пята локація «Зимовий пік» (рис.2.33) добавляється вітер. Вітер впливає на гравця коли він стрибає, змінюючи направлення його падіння. В цій локації немає крюків та платформ, що ламаються, бо з вітром це створило би рівень, який буде дуже складно пройти.

Рис. 2.32 Локація «Покинута вежа»

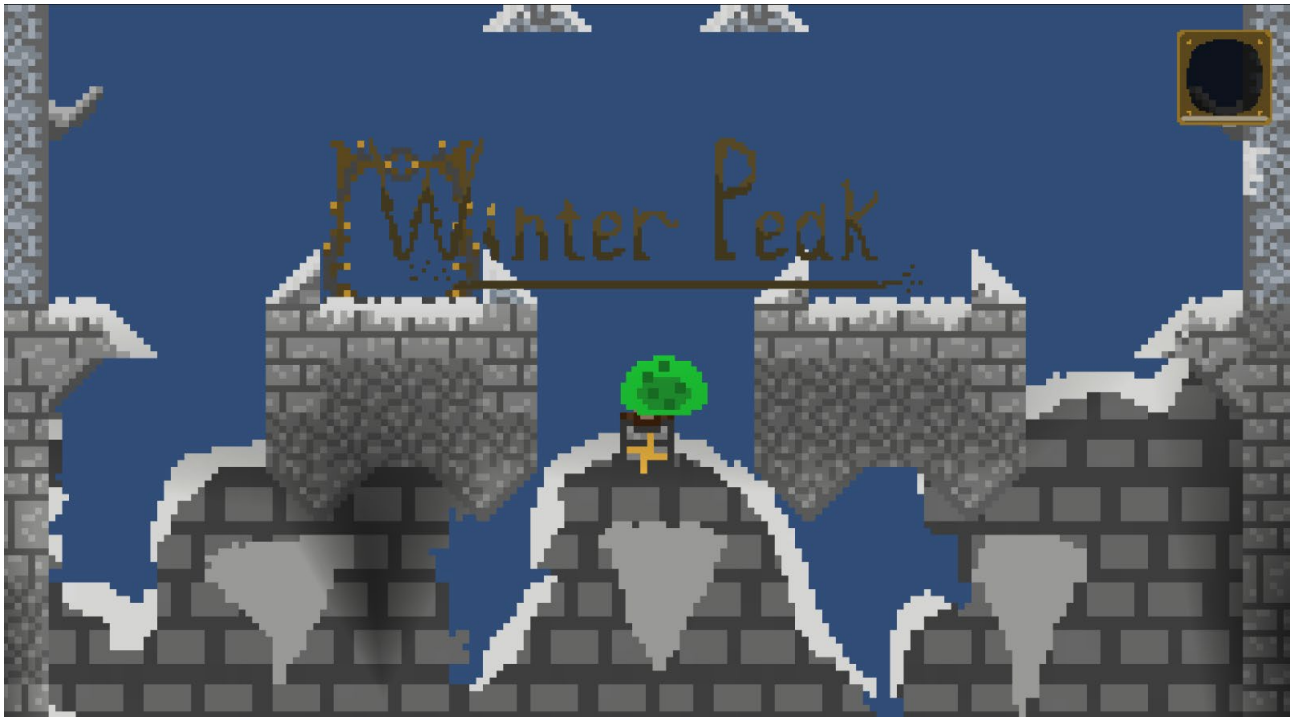

Рис. 2.33 Локація «Зимовий пік»

РОЗДІЛ 3
ЕКОНОМІЧНІЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 3110;
2. коефіцієнт складності програми – 1,4;
3. коефіцієнт корекції програми в ході її розробки – 0,2;
4. годинна заробітна плата програміста – 160 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,3;
7. вартість машино-години ЕОМ – 0,132 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{ , людино-годин,} \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, що розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \qquad (3.2)$$

де q – передбачуване число операторів (3110);

C – коефіцієнт складності програми (1,4);

p – коефіцієнт кореляції програми в ході її розробки (0,2).

$$Q = 3110 \cdot 1{,}4 \cdot (1 + 0{,}2) = 5224.8;$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$tu = \left(\frac{Q*B}{(75...85) \cdot \text{К}}\right), \text{людино-годин} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2);

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (1,3);

$$t_u = 5224.8 \cdot 1.3 / (85 \cdot 1{,}3) = 110.5, \text{людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot \text{К}}; \qquad (3.4)$$

$$t_a = 5224.8 / (20 \cdot 1{,}3) = 200.9, \text{людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot \text{К}}; \qquad (3.5)$$

$$t_n = 5224.8 / (25 \cdot 1.3) = 160.7, \text{людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

− за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5) \cdot \text{К}}; \qquad (3.6)$$

$$t_{\text{отл}} = 5224.8 /(5 \cdot 1{,}3) = 803.8, \text{ людино-годин},$$

− за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{\text{к}} = 1{,}2 \cdot t_{\text{отл}}; \qquad (3.7)$$

$$t_{\text{отл}}^{\text{к}} = 1.2 \cdot 803.8 = 964.5, \text{ людино-годин},$$

Витрати праці на підготовку документації:

$$t_{\partial} = t_{\partial p} + t_{\partial o}; \qquad (3.8)$$

де $t_{\partial p}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_{\partial} = \frac{Q}{(15...20) \cdot \text{К}}; \qquad (3.9)$$

$t_{\partial} = 5224.8 / (20 \cdot 1.3) = 200.9$, людино-годин,

де $t_{\partial o}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{\partial o} = 0{,}75 * t\partial_{\text{р}}; \qquad (3.10)$$

$t_{\partial o} = 0{,}75 \cdot 200.9 = 150.6$, людино-годин.

$$t_{\partial} = 200.9 + 150.6 = 351.5, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 110.5 + 200.9 + 160.7 + 964.5 + 351.5 = 1838.1, \text{ людино-годин.}$$

У результаті ми розрахували, що в загальній складності необхідно 2359.7 людино-годин для розробки даного програмного забезпечення.

3.2. Рахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми ЗЗП і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{грн,} \qquad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t^* C_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 160 грн/год, то отримаємо:

$$З_{ЗП} = 1838.1 \cdot 0.132 = 294096, \text{ грн.}$$

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} * C_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 803.8 \cdot 0.132 = 106.1 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 294096 + 106.1 = 294202.1 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k * F_p} \text{ мес.} \qquad (3.14)$$

де B_k – число виконавців;

$F_р$ – місячний фонд робочого часу (при 49 годинному робочому тижні Fр=196 годин).

Витрати на створення програмного продукту:

$$T = 1838.1 \ / \ (1 \cdot 196) = 9.4 \text{ міс.}$$

Висновки. Розроблений ігровий додаток має вартість 294202.1 грн. Ймовірний очікуваний час розробки – 9.4 місяці при 49-годинному робочому тижні і 196-годинному робочому місяці. Цей термін пов’язаний із значною кількістю операторів і включає в себе час для дослідження та розробку алгоритму розв’язання задачі, розробку дизайну і створення документації. На розробку гри буде витрачено 1838.1 людино-годин.

ВИСНОВКИ

У цій роботі було розглянуто процес розробки 2D гри «Rising of Slime: Reslimed» в жанрі «Платформер» за допомогою двигуна Unity. Детально проаналізовано кожен етап розробки, від створення графічного контенту до програмування логіки гри.

Використання Unity надало зручність та потужність у створенні графічних сцен, управління фізикою, анімацією та звуком. Процес розробки був покроково описаний, розкриваючи важливі аспекти програмування гри, такі як обробка введення користувача, рух об'єктів, детекція колізій та робота зі звуком. Було проаналізовано ефективні методи оптимізації та управління ресурсами для підвищення продуктивності гри.

В результаті було успішно створено функціональну та привабливу 2D гру платформер, яка демонструє основні елементи геймплею та може бути подальше розширена та вдосконалена. Дипломна робота підтверджує важливість і потенціал використання Unity для розробки якісних ігрових проектів та висвітлює ключові аспекти розробки 2D ігор платформерів.

Розроблене програмне забезпечення дозволяє:

- Зміна повноєкранного режиму.
- Зміна гучності музики.
- Зміна гучності звуків.
- Збереження ігрової сессії.
- Управління головним героєм.
- Підбирання предметів.
- Купівля предметів.
- Використання предметів.

Програма реалізована на мові програмування C# з використанням ігрового двигуна Unity, середи розробки VScode, редактору зображень Adobe Photoshop та редактор музики FL Studio.

В ході виконання даної кваліфікаційної роботи ще було визначено трудомісткість розробленої системи, проведено розрахунок вартості роботи з створення програми, використовуючи середню зарплату розробника. Він складає 294202.1 грн. Також було розраховано час для створення програмного додатку - 1838.1 людино-годин, тобто приблизно 9.4 місяців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity documentation. URL: https://docs.unity.com (дата звернення: 01.02.2023)
2. C# documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 03.02.2023)
3. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web (дата звернення: 01.02.2023)
4. Gaming well: links between videogames and flourishing mental health. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3978245/ (дата звернення: 7.02.2023)
5. StackOverflow. URL: https://stackoverflow.com (дата звернення: 10.02.2023)
6. GitHub Documentation. URL: https://docs.github.com/ru (дата звернення: 15.02.2023)
7. Photoshop Documentation. URL: https://helpx.adobe.com/photoshop/user-guide.html (дата звернення: 20.02.2023)
8. Luenendonk M. "The Gaming Industry – An Introduction". URL: https://www.cleverism.com/gaming-industry-introduction/ (дата звернення: 12.02.2023)
9. Parallax Scrolling. URL: https://www.wix.com/blog/2019/08/what-is-parallax-scrolling-explained-with-examples/?utm_source=google&utm_medium=cpc&utm_campaign=13708482660^124757113592&experiment_id=^^530755701287^^_DSA&gclid=CjwKCAjw36GjBhAkEiwAKwIWyT_ThfROK9O5dx-7qOX_ArV93am3JPtAUR6J7H0xibk_WuxwcpaHsxoChQ8QAvD_BwE (дата звернення: 18.04.2023)
10. What is a Platform Game? URL: https://www.lifewire.com/what-is-a-platform-game-812371 (дата звернення: 22.04.2023)

11. Types of Game Designers. URL: https://bbrathwaite.wordpress.com/2007/11/20/types-of-game-designers/ (дата звернення: 11.02.2023)
12. Game design basics: How to start creating video games. URL: https://www.cgspectrum.com/blog/game-design-basics-how-to-start-building-video-games (дата звернення: 04.02.2023)
13. Playing Video Games Offers Learning Across Life Span, Say Studies. URL: https://www.sciencedaily.com/releases/2008/08/080817223442.htm (дата звернення: 14.02.2023)
14. Game design tips for new developers. URL: https://unity.com/how-to/beginner/10-game-design-tips-new-developers (дата звернення: 07.03.2023)
15. 3D Vs. 2D: The Eternal Battle to Develop Video Games. URL: https://starloopstudios.com/3d-vs-2d-the-eternal-battle-to-develop-video-games/ (дата звернення: 08.03.2023)
16. 8 Free Game Development Software Tools to Make Your Own Games. URL: https://www.makeuseof.com/tag/five-free-game-development-tools-make-your-own-games/
17. The Reality of Game Development. URL: https://www.gamedeveloper.com/business/the-reality-of-game-development (дата звернення: 01.03.2023)
18. The Association Between Video Gaming and Psychological Functioning. URL: https://www.frontiersin.org/articles/10.3389/fpsyg.2019.01731/full (дата звернення: 10.02.2023)
19.. Axon S. Unity at 10: For better—or worse—game development has never been easier. URL:https://arstechnica.com/gaming/2016/09/unity-at-10-forbetter-or-worse-game-development-has-never-been-easier/ (дата звернення: 05.03.2023)

20. The Psychology of Video Games: Current Research on the Impact of Playing Games. URL: https://joanganzcooneycenter.org/2021/05/17/the-psychology-of-video-games/ (дата звернення: 17.03.2023)
21. Game developer: середня зарплата в Україні. URL: https://www.work.ua/salary-game+developer/ (дата звернення: 15.05.2023)
22. Computer electricity usage. URL: https://news.energysage.com/how-many-watts-does-a-computer-use/#:~:text=On%20average%2C%20laptops%20use%20about,hours%20of%20electricity%20per%20year. (дата звернення: 11.05.2023)
23. Ukraine electricity cost. URL: https://fakty.com.ua/ua/ukraine/suspilstvo/20230601-zonni-taryfy-na-elektroenergiyu-osoblyvosti-ta-pidklyuchennya/#:~:text=Нагадуємо%2C%20Кабмін%20затвердив%20підвищення%20з,якщо%20понад%20250%20кВт-год. (дата звернення: 11.05.2023)

ДОДАДОК А

КОД ПРОГРАМИ

Лістинг Coin.cs:

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

public class Coin : MonoBehaviour
{
 ItemID item;
 void Start()
 {
  item = gameObject.GetComponent<ItemID>();

  for (int i = 0; i < SaveSystem.saveData.pickedCoins.Count; i++)
  {
   if (SaveSystem.saveData.pickedCoins[i] == item.individualItemID)
   {
    gameObject.SetActive(false);
    break;
   }
  }
 }

 void Update()
 {

 }
}
```

Лістинг Crystal.cs:

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

public class Crystal : MonoBehaviour
{
 public CooldownManager cooldownManager;
 int itemID = 3;
 IEnumerator cooldownCoroutine;
 bool cooldownCoroutineIsRunning = false;
 // bool portalTouched = false;
 Collider2D touchedPortalCollider;
 Animator touchedPortalAnim;
 void Start()
 {
  cooldownCoroutine = cooldownManager.ItemCooldown(itemID);
 }
 void Update()
 {
  if (Input.GetKeyDown(KeyCode.R) && InventoryManager.activatedItemID == itemID &&
InventoryManager.currentItems[itemID].timer <= 0)
  {
   touchedPortalCollider = Physics2D.OverlapCircle(PlayerScript.rb.gameObject.transform.position, 0.2f,
LayerMask.GetMask("Portal"));
   if (touchedPortalCollider)
   {
```

```
        touchedPortalAnim = touchedPortalCollider.gameObject.GetComponent<Animator>();
        if (!touchedPortalAnim.GetBool("PortalActivated"))
        {
          touchedPortalAnim.SetBool("PortalActivated", true);
          InventoryManager.UseItem(itemID, 1);
        }
      }
    }

    cooldownManager.CooldownControl(itemID, ref cooldownCoroutineIsRunning, ref cooldownCoroutine);
  }

}
```

Лістинг ItemID.cs:

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

public class ItemID : MonoBehaviour
{
  public int itemID, individualItemID, itemAmount;
}
```

Лістинг Sphere.cs:

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

public class Sphere : MonoBehaviour
{
  public GameObject SphereObj;
  public CooldownManager cooldownManager;
  int itemID = 4;
  IEnumerator cooldownCoroutine;
  bool cooldownCoroutineIsRunning = false;
  void Start()
  {
    cooldownCoroutine = cooldownManager.ItemCooldown(itemID);
  }
  void Update()
  {
    if (Input.GetKeyDown(KeyCode.R) && InventoryManager.activatedItemID == itemID &&
InventoryManager.currentItems[itemID].timer <= 0)
    {
      if (SphereObj.activeSelf)
      {
        SphereObj.SetActive(false);
      }
      else
      {
        SphereObj.SetActive(true);
      }

      InventoryManager.UseItem(itemID, 0);
    }

    cooldownManager.CooldownControl(itemID, ref cooldownCoroutineIsRunning, ref cooldownCoroutine);
  }
}
```

Лістинг Watch.cs:

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

public class Watch : MonoBehaviour
{
 public CooldownManager cooldownManager;
 public GameObject WatchUI, WatchBackgroundUI;
 int itemID = 2;
 IEnumerator cooldownCoroutine;
 bool cooldownCoroutineIsRunning = false;
 void Start()
 {
  cooldownCoroutine = cooldownManager.ItemCooldown(itemID);
 }
 void Update()
 {
  if (Input.GetKeyDown(KeyCode.R) && InventoryManager.activatedItemID == itemID &&
InventoryManager.currentItems[itemID].timer <= 0)
  {
   StartCoroutine(WatchWorking());
   InventoryManager.UseItem(itemID, 0);
  }

  cooldownManager.CooldownControl(itemID, ref cooldownCoroutineIsRunning, ref cooldownCoroutine);
 }

 IEnumerator WatchWorking()
 {
  Time.timeScale = 0.5f;
  PlayerScript.walkSpeed *= 2;
  PlayerScript.rb.drag = 10;
  PlayerScript.watchIsWorking = true;
  WatchBackgroundUI.SetActive(true);
  WatchUI.SetActive(true);
  yield return new WaitForSeconds(3f);
  Time.timeScale = 1f;
  PlayerScript.walkSpeed = PlayerScript.initialWalkSpeed;
  PlayerScript.rb.drag = 0;
  PlayerScript.watchIsWorking = false;
  WatchBackgroundUI.SetActive(false);
  WatchUI.SetActive(false);
 }
}
```

Лістинг Wings.cs:

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

public class Wings : MonoBehaviour
{
 public CooldownManager cooldownManager;
 public GameObject WingsObj;
 int itemID = 1;
 IEnumerator cooldownCoroutine;
 bool cooldownCoroutineIsRunning = false;
 void Start()
 {
  cooldownCoroutine = cooldownManager.ItemCooldown(itemID);
 }
```

```
  void Update()
  {
    if (Input.GetKeyDown(KeyCode.R) && InventoryManager.activatedItemID == itemID &&
InventoryManager.currentItems[itemID].timer <= 0)
    {
     PlayerScript.rb.velocity = new Vector2(PlayerScript.rb.velocity.x, 2f);
     InventoryManager.UseItem(itemID, 0);
     StartCoroutine(WingsWorking());
    }

    cooldownManager.CooldownControl(itemID, ref cooldownCoroutineIsRunning, ref cooldownCoroutine);
  }

  IEnumerator WingsWorking()
  {
    WingsObj.SetActive(true);
    yield return new WaitForSeconds(1f);
    WingsObj.SetActive(false);
  }
}
```

Лістинг CooldownManager:

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

public class CooldownManager : MonoBehaviour
{
  GameObject CooldownSliderObj;
  Slider CooldownSlider;

  void Start()
  {
    CooldownSliderObj = gameObject;
    CooldownSlider = CooldownSliderObj.GetComponent<Slider>();
    CooldownSlider.value = 0;
  }
  void Update()
  {

  }

  public void FillSlider()
  {
    CooldownSlider.value = 1;
  }

  public void EmptySlider()
  {
    CooldownSlider.value = 0;
  }

  public void CooldownControl(int itemID, ref bool cooldownCoroutineIsRunning, ref IEnumerator cooldownCoroutine)
  {
    if (InventoryManager.activatedItemID == itemID)
    {
      if (!cooldownCoroutineIsRunning)
      {
        StartCoroutine(cooldownCoroutine);
        cooldownCoroutineIsRunning = true;
      }
```

```
    }

    if (InventoryManager.currentItems[itemID].timer <= 0 || InventoryManager.activatedItemID != itemID)
    {
      if (cooldownCoroutineIsRunning)
      {
        StopCoroutine(cooldownCoroutine);
        cooldownCoroutineIsRunning = false;
      }
    }
  }

  public IEnumerator ItemCooldown(int itemID)
  {
    float sliderStep = CooldownSlider.maxValue / InventoryManager.currentItems[itemID].cooldown;
    float sliderCurrentLoad;

    while (true)
    {
      sliderCurrentLoad = CooldownSlider.maxValue - sliderStep * (InventoryManager.currentItems[itemID].cooldown -
InventoryManager.currentItems[itemID].timer);
      CooldownSlider.value = sliderCurrentLoad;
      yield return new WaitForSeconds(1f);
    }
  }
}
```

Лістинг GuideManager:

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

public class GuideManager : MonoBehaviour
{
  public GameObject GuideText;
  TextMeshProUGUI guideTextComponent;
  Animator anim;
  List<IEnumerator> guidesQueue = new List<IEnumerator>();
  bool textIsWorking = false;

  void Start()
  {
    guideTextComponent = GuideText.GetComponent<TextMeshProUGUI>();
    anim = GuideText.GetComponent<Animator>();


    // guideTextComponent.gameObject.transform.parent = PlayerScript.Player.transform;
  }

  void Update()
  {
    if (guidesQueue.Count > 0 && !textIsWorking)
    {
      StartCoroutine(guidesQueue[0]);
    }
    // print("guidesQueue.Count: " + guidesQueue.Count);
    // guideTextComponent.gameObject.transform.position = PlayerScript.Player.transform.position + new Vector3(0, 1,
0);
    // for (int i = 0; i < guidesQueue.Count; i++)
    // {
    //   print(guidesQueue[i]);
```

```
    // }
  }

  IEnumerator WriteText(string text, float time)
  {
    guidesQueue.RemoveAt(0);
    textIsWorking = true;

    guideTextComponent.SetText(text);

    anim.SetBool("GuideON", true);
    anim.SetBool("GuideOFF", false);

    yield return new WaitForSeconds(time);

    anim.SetBool("GuideON", false);
    anim.SetBool("GuideOFF", true);

    yield return new WaitForSeconds(3f);

    textIsWorking = false;
  }

  public void AddGuideInQueue(string text, float time)
  {
    guidesQueue.Add(WriteText(text, time));
  }
}
```

Лістинг InventoryManager.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using TMPro;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class Item
{
  public int id;
  public string name;
  public string description;
  public int attachedContainer = -1;
  public Sprite inventoryView;
  public bool canBeOnlyOne;
  public int count;
  public float cooldown, timer = 0;
  public bool infinite;
  public int price;

  public Item(int id, string name, string description, Sprite inventoryView, bool canBeOnlyOne, int count, float cooldown,
bool infinite, int price)
  {
    this.id = id;
    this.name = name;
    this.description = description;
    this.inventoryView = inventoryView;
    this.canBeOnlyOne = canBeOnlyOne;
    this.cooldown = cooldown;
    this.infinite = infinite;
```

```
    this.price = price;

    if (canBeOnlyOne)
    {
      if (count == 0) this.count = 0;
      else if (count == 1) this.count = 1;
    }
    else this.count = count;
  }

  public void AddItem(int amount)
  {
    if (this.canBeOnlyOne && amount == 1) this.count = 1;
    else this.count += amount;

    for (int i = 0; i < SaveSystem.saveData.Items.Length; i++)
    {
      if (SaveSystem.saveData.Items[i].id == this.id)
      {
        SaveSystem.saveData.Items[i].count = this.count;
      }
    }
  }

  public void DeleteItem(int amount)
  {
    if (this.count >= amount) this.count -= amount;
    else this.count = 0;

    if (this.count == 0)
    {
      this.attachedContainer = -1;
      InventoryManager.activatedItemID = -1;
      InventoryManager.ActiveInventoryItemImage.overrideSprite = null;
      InventoryManager.ActiveInventoryItemImage.color = new Color(0, 0, 0, 0);

      SaveSystem.saveData.activatedItem = -1;
    }

    for (int i = 0; i < SaveSystem.saveData.Items.Length; i++)
    {
      if (SaveSystem.saveData.Items[i].id == this.id)
      {
        SaveSystem.saveData.Items[i].count = this.count;
      }
    }
  }

  public void ResetTimer()
  {
    timer = cooldown;
  }
}

public class InventoryManager : MonoBehaviour
{
  public GameObject Inventory, InventoryContainers, ItemName, ItemDescription, ActiveInventory;
  List<GameObject> Containers = new List<GameObject>();
  List<int> takenContainersIDs = new List<int>();
  public static Item[] currentItems = new Item[5];
  public Sprite coin, wings, clock, crystal, sphere;
  public static Image ActiveInventoryItemImage;
  int choosenItemID = 0;
```

```
 static public int activatedItemID;
 public static bool InventoryIsON = false;

 void Start()
 {
  List<Transform> InventoryContainersTransform = new List<Transform>();
  foreach (Transform containerTransform in InventoryContainers.transform)
   InventoryContainersTransform.Add(containerTransform);

  for (int i = 0; i < InventoryContainersTransform.Count; i++)
   Containers.Add(InventoryContainersTransform[i].gameObject);

  ActiveInventoryItemImage = ActiveInventory.transform.GetChild(0).GetComponent<Image>();

  currentItems[0] = new Item(
   0,
   "Coin",
   "Shiny coins",
   coin,
   false,
   0,
   5,
   false,
   0
  );
  currentItems[1] = new Item(
   1,
   "Wings",
   "A little damaged, pitch black wings. I don't know which creature they belonged, but now he doesn\'t need them. Wearing wings rise to the sky. Since older times, people want to fly, it\'s quite funny, that this wish come true only if you cease to be a human. It\'s a pity they won\'t stand it for long.",
   wings,
   true,
   0,
   60,
   true,
   6
  );
  currentItems[2] = new Item(
   2,
   "Watch",
   "Pretty old-looking watch. Looks like they will be broken soon. A legend says that a long time ago clocks were used not only for measurement of time, but also for manage it. But in our time, they have lost this power. Perhaps I can at least slightly reveal their strength. Though it just the legend.",
   clock,
   true,
   0,
   60,
   true,
   6
  );
  currentItems[3] = new Item(
   3,
   "Crystal",
   "Translucent crystal. Such are located deep underground, for centuries filling with an energy. Usually they hardly glow. But this one shines clearly turquoise color. Such crystals are used to activate ancient mechanisms. Such an energy-filled crystal can power a large mechanism. How long have it been underground?",
   crystal,
   false,
   0,
   1,
   false,
   1
```

```
    );
    currentItems[4] = new Item(
      4,
      "Sphere",
      "Almost transparent sphere",
      sphere,
      true,
      0,
      1,
      true,
      0
    );

    for (int i = 0; i < currentItems.Length; i++)
    {
      for (int j = 0; j < SaveSystem.saveData.Items.Length; j++)
      {
        if (currentItems[i].id == SaveSystem.saveData.Items[j].id)
        {
          currentItems[i].count = SaveSystem.saveData.Items[j].count;
        }
      }
    }

    InsertItemsToContainers();
    DrawItemsInContainers();

    if (SaveSystem.saveData.activatedItem != -1)
    {
      for (int i = 0; i < currentItems.Length; i++)
      {
        if (SaveSystem.saveData.activatedItem == currentItems[i].id)
        {
          ActiveInventoryItemImage.overrideSprite = currentItems[i].inventoryView;
          ActiveInventoryItemImage.color = Color.white;
          activatedItemID = SaveSystem.saveData.activatedItem;
          break;
        }
      }
    }
    else
    {
      activatedItemID = -1;
    }
  }

  void Update()
  {
    if (Input.GetKeyDown(KeyCode.Tab) && !PortalManager.PortalMenuIsON && !Pause_Menu.PauseMenuIsON &&
!TradeManager.TradeMenuIsON)
    {
      if (Inventory.activeSelf)
      {
        DeactivateInventory();
      }
      else
      {
        ActivateInventory();
      }
    }
    else if (Input.GetKeyDown(KeyCode.Escape) && !PortalManager.PortalMenuIsON &&
!Pause_Menu.PauseMenuIsON)
    {
```

```
    DeactivateInventory();
  }

  if (Inventory.activeSelf)
  {
    InsertItemsToContainers();
    DrawItemsInContainers();
  }
  // Контроль откатов предметов.
  for (int i = 0; i < currentItems.Length; i++)
  {
    if (currentItems[i].cooldown != 0)
    {
      currentItems[i].timer -= Time.deltaTime;
    }
  }

  // for (int i = 0; i < currentItems.Length; i++)
  // {
  //   print("Item name: " + currentItems[i].name + "\nAttached container: " + currentItems[i].attachedContainer);
  // }
}

void ActivateInventory()
{
  Inventory.SetActive(true);
  Time.timeScale = 0f;
  Containers[0].GetComponent<Button>().Select();
  InventoryIsON = true;
  ChangeSelectedContainer(0);
}

public void DeactivateInventory()
{
  Inventory.SetActive(false);
  Time.timeScale = 1f;
  InventoryIsON = false;
}

bool ContainerIsFree(int containerID)
{
  bool containerIsFree = true;

  for (int i = 0; i < currentItems.Length; i++)
  {
    if (containerID == currentItems[i].attachedContainer)
    {
      containerIsFree = false;
    }
  }

  return containerIsFree;
}

void InsertItemsToContainers()
{
  for (int i = 0; i < Containers.Count; i++)
  {
    for (int j = 0; j < currentItems.Length; j++)
    {
      if (currentItems[j].count > 0 && currentItems[j].attachedContainer == -1 && ContainerIsFree(i))
      {
        currentItems[j].attachedContainer = i;
```

```
        break;
       }
      }
     }
    }

    void DrawItemsInContainers()
    {
     for (int i = 0; i < Containers.Count; i++)
     {
      for (int j = 0; j < currentItems.Length; j++)
      {
       if (i == currentItems[j].attachedContainer)
       {
        Containers[i].GetComponent<Image>().overrideSprite = currentItems[j].inventoryView;

        if (!currentItems[j].canBeOnlyOne)
        {

Containers[i].transform.GetChild(0).GetComponent<TextMeshProUGUI>().SetText(Convert.ToString(currentItems[j].
count));
        }

        break;
       }
       else
       {
        Containers[i].GetComponent<Image>().overrideSprite = null;
        Containers[i].transform.GetChild(0).GetComponent<TextMeshProUGUI>().SetText("");
       }
      }
     }
    }

    public void ChangeSelectedContainer(int containerID)
    {
     if (ContainerIsFree(containerID))
     {
      choosenItemID = -1;
      ItemName.GetComponent<TextMeshProUGUI>().SetText("");
      ItemDescription.GetComponent<TextMeshProUGUI>().SetText("");
     }
     else
     {
      for (int i = 0; i < currentItems.Length; i++)
      {
       if (currentItems[i].attachedContainer == containerID)
       {
        choosenItemID = currentItems[i].id;
        ItemName.GetComponent<TextMeshProUGUI>().SetText(currentItems[i].name);
        ItemDescription.GetComponent<TextMeshProUGUI>().SetText(currentItems[i].description);
        break;
       }
      }
     }
    }

    public static void AddItem(int itemID, int amount)
    {
     for (int i = 0; i < currentItems.Length; i++)
     {
      if (itemID == currentItems[i].id)
      {
```

```
        currentItems[i].AddItem(amount);
      }
    }
  }

  public static void RemoveItem(int itemID, int amount)
  {
    for (int i = 0; i < currentItems.Length; i++)
    {
      if (itemID == currentItems[i].id)
      {
        currentItems[i].DeleteItem(amount);
      }
    }
  }

  public void ActivateItem()
  {
    for (int i = 0; i < currentItems.Length; i++)
    {
      if (choosenItemID == currentItems[i].id)
      {
        ActiveInventoryItemImage.overrideSprite = currentItems[i].inventoryView;
        ActiveInventoryItemImage.color = Color.white;
        activatedItemID = choosenItemID;
        SaveSystem.saveData.activatedItem = activatedItemID;
        break;
      }
    }
  }

  public static void UseItem(int itemID, int amount)
  {
    for (int i = 0; i < currentItems.Length; i++)
    {
      if (itemID == currentItems[i].id)
      {
        currentItems[i].ResetTimer();

        if (!currentItems[i].infinite)
        {
          currentItems[i].DeleteItem(amount);
        }
      }
    }
  }
}
```

Лістинг ItemPickUp.cs:

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

public class ItemPickUp : MonoBehaviour
{
  public GameObject PressF;
  Animator pressFAnim;

  void Start()
  {
    pressFAnim = PressF.GetComponent<Animator>();
  }
```

```
  void PickUp(Collider2D col)
  {
   if (col.transform.gameObject.layer == LayerMask.NameToLayer("Item"))
   {
    pressFAnim.SetBool("Press_F_Show", true);
    pressFAnim.SetBool("Press_F_Hide", false);

    if (Input.GetKey(KeyCode.F))
    {
     ItemID item = col.transform.gameObject.GetComponent<ItemID>();
     InventoryManager.AddItem(item.itemID, item.itemAmount);
     Destroy(col.transform.gameObject);

     if (item.itemID == 0)
     {
      SaveSystem.saveData.pickedCoins.Add(item.individualItemID);
     }
    }
   }
  }

  void OnTriggerEnter2D(Collider2D col)
  {
   PickUp(col);
  }

  void OnTriggerStay2D(Collider2D col)
  {
   PickUp(col);
  }

  void OnTriggerExit2D(Collider2D col)
  {
   if (col.transform.gameObject.layer == LayerMask.NameToLayer("Item"))
   {
    pressFAnim.SetBool("Press_F_Hide", true);
    pressFAnim.SetBool("Press_F_Show", false);
   }
  }
}
```

Лістинг Main_Menu.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using UnityEngine.Audio;
using System.IO;

public class Main_Menu : MonoBehaviour
{
  public GameObject MainMenu, achievements, authors;
  public Button ContinueButton, NewGameButton;
  public AudioSource menuMusic;
  public AudioMixerGroup mixer;

  void Start()
  {
   SaveSystem.LoadOptions();
   Screen.fullScreen = SaveSystem.optionsData.isFullScreen;
```

```
    mixer.audioMixer.SetFloat("MusicVolume", SaveSystem.optionsData.musicVolume);
    mixer.audioMixer.SetFloat("SoundsVolume", SaveSystem.optionsData.soundsVolume);
    mixer.audioMixer.SetFloat("OtherSoundsVolume", SaveSystem.optionsData.soundsVolume);

    if (File.Exists(SaveSystem.savePath))
    {
      ContinueButton.gameObject.SetActive(true);
      ContinueButton.Select();
    }
    else
    {
      NewGameButton.Select();
    }

    menuMusic.Play();
  }

  void Update()
  {
    if (Input.GetKeyDown(KeyCode.Escape))
    {
      MainMenu.SetActive(true);
      authors.SetActive(false);
    }
  }

  void OnDestroy()
  {
    CustomSceneManager.previousSceneName = gameObject.scene.name;
  }

  public void Continue()
  {
    SaveSystem.LoadData();
    SceneManager.LoadScene("Game");
  }
  //Удаление всех сохранённых данных, кроме данных настроек
  public void New_Game()
  {
    SaveSystem.ResetSaveData();
    SaveSystem.SaveData();
    SceneManager.LoadScene("Game");
  }
  //Включение меню настроек
  public void Options()
  {
    SceneManager.LoadScene("Options");
  }
  //Включение меню разработчиков
  public void Authors()
  {
    MainMenu.SetActive(false);
    authors.SetActive(true);
  }
  //Выход из игры
  public void Exit()
  {
    Application.Quit();
  }
}
```

Лістинг Options_Menu.cs:

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

public class Options_Menu : MonoBehaviour
{
 public Toggle options_fullscreen;
 public Slider options_musicVolume, options_soundsVolume;
 public AudioMixerGroup mixer;
 // Start is called before the first frame update
 void Start()
 {
  SaveSystem.LoadOptions();
  options_fullscreen.isOn = SaveSystem.optionsData.isFullScreen;
  options_musicVolume.value = SaveSystem.optionsData.musicVolume;
  options_soundsVolume.value = SaveSystem.optionsData.soundsVolume;
 }

 // Update is called once per frame
 void Update()
 {
  if (Input.GetKeyDown(KeyCode.Escape))
  {
   switch (CustomSceneManager.previousSceneName)
   {
    case "Game":
     SceneManager.LoadScene("Game");
     break;
    case "Main_Menu":
     SceneManager.LoadScene("Main_Menu");
     break;
   }
  }
 }

 void OnDestroy()
 {
  CustomSceneManager.previousSceneName = gameObject.scene.name;
  SaveSystem.SaveOptions();
 }

 //Включение или выключение полноэкранного режима
 public void FullScreenON()
 {
  if (options_fullscreen.isOn)
  {
   Screen.fullScreen = true;
  }
  else
  {
   Screen.fullScreen = false;
  }

  SaveSystem.optionsData.isFullScreen = Screen.fullScreen;
 }
 //Изменение гучности музыки
 public void ChangeMusicVolume()
 {
  mixer.audioMixer.SetFloat("MusicVolume", options_musicVolume.value);
  SaveSystem.optionsData.musicVolume = (int)options_musicVolume.value;
```

```
  }
  //Изменение гучности звуков
  public void ChangeSoundsVolume()
  {
    mixer.audioMixer.SetFloat("SoundsVolume", options_soundsVolume.value);
    mixer.audioMixer.SetFloat("OtherSoundsVolume", options_soundsVolume.value);
    SaveSystem.optionsData.soundsVolume = (int)options_soundsVolume.value;
  }
}
```

Лістинг Pause_Menu.cs:

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

public class Pause_Menu : MonoBehaviour
{
  // bool GameIsPaused = false;
  public GameObject PauseMenuUI;
  public static bool PauseMenuIsON = false;

  void Update()
  {
    //При нажатии escape
    if (Input.GetKeyDown(KeyCode.Escape) && !PortalManager.PortalMenuIsON && !InventoryManager.InventoryIsON && !TradeManager.TradeMenuIsON)
    {
      //Если игра на паузе
      if (PauseMenuIsON)
      {
        //Убрать паузу
        Resume();
      }
      else
      {
        //Поставить на паузу
        Pause();
      }
    }
  }

  void OnDestroy()
  {
    Resume();
  }
  //Восстановление игры
  public void Resume()
  {
    PauseMenuUI.SetActive(false);
    Time.timeScale = 1f;
    PauseMenuIsON = false;
  }
  //Пауза игры
  public void Pause()
  {
    PauseMenuUI.SetActive(true);
    Time.timeScale = 0f;
    PauseMenuIsON = true;
    PauseMenuUI.transform.GetChild(0).GetComponent<Button>().Select();
```

```
  }

  public void Options()
  {
    // Time.timeScale = 1f;
    SceneManager.LoadScene("Options");
  }
  //Сохранение нужных данных и переход на главное меню
  public void MainMenu()
  {
    // Time.timeScale = 1f;
    SceneManager.LoadScene("Main_Menu");
  }
  //Сохранение нужных данных и выход из игры
  public void Exit()
  {
    Application.Quit();
  }
}
```

Лістинг PortalManager.cs:

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

public class PortalManager : MonoBehaviour
{
  public GameObject PortalMenuContainer;
  public static int currentWorkingPortalID;
  public static bool PortalMenuIsON = false;
  Button[] buttonsObj;
  void Start()
  {
    buttonsObj = PortalMenuContainer.GetComponentsInChildren<Button>(true);

    for (int i = 0; i < buttonsObj.Length / 2; i++)
    {
      Button tmp = buttonsObj[i];
      buttonsObj[i] = buttonsObj[buttonsObj.Length - i - 1];
      buttonsObj[buttonsObj.Length - i - 1] = tmp;
    }
  }

  void Update()
  {
    if (PortalMenuContainer.activeSelf)
    {
      PortalMenuIsON = true;
      if (Input.GetKeyDown(KeyCode.Escape) && !Pause_Menu.PauseMenuIsON && !InventoryManager.InventoryIsON)
        DeactivatePortalUI();

      for (int i = 0; i < SaveSystem.saveData.passedPortals.Count; i++)
      {
        for (int j = 0; j < buttonsObj.Length; j++)
        {
          if (SaveSystem.saveData.passedPortals[i] == j)
          {
            buttonsObj[j].gameObject.SetActive(true);
          }
```

```
      }
     }
    }
    else
     PortalMenuIsON = false;
  }

  public void DeactivatePortalUI()
  {
   PortalMenuContainer.SetActive(false);
   Time.timeScale = 1f;
  }
}
```

Лістинг TeleportTo.cs:

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

public class TeleportTo : MonoBehaviour
{
  public GameObject PortalMenuContainer;//, PortalsContainer;

  public void TeleportPlayerToPortal(GameObject portalToTeleport)
  {
   PlayerScript.rb.gameObject.transform.position = new Vector3(
     portalToTeleport.gameObject.transform.position.x,
     portalToTeleport.gameObject.transform.position.y,
     PlayerScript.Player.transform.position.z);

   GameObject[] PortalsObjArray = GameObject.FindGameObjectsWithTag("Portal");

   PortalsObjArray[PortalManager.currentWorkingPortalID].GetComponent<Animator>().SetBool("PortalActivated",
false);
   //Deleting activated portal in save data after portal has been used

SaveSystem.saveData.activatedPortals.RemoveAt(SaveSystem.saveData.activatedPortals.IndexOf(PortalManager.curre
ntWorkingPortalID));

   PortalMenuContainer.SetActive(false);
   Time.timeScale = 1f;

   // for (int i = 0; i < PortalsContainer.transform.childCount; i++)
   // {
   //   PortalsContainer.transform.GetChild(i).GetComponent<Animator>().SetBool("PortalActivated", false);
   // }
  }
}
```

Лістинг Trade.cs:

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

public class Trade : MonoBehaviour
{
   public GameObject WingsText, ClockText, CrystalText, BuyWings, BuyClock, BuyCrystal;
   public GameObject NotEnoughtMoney_Wings, NotEnoughtMoney_Clock, NotEnoughtMoney_Crystal;
   //Включение описания и кнопки для покупки при нажатии на крылья в меню покупок
   public void ClickWings()
   {
```

```
    WingsText.SetActive(true);
    ClockText.SetActive(false);
    CrystalText.SetActive(false);

    BuyWings.SetActive(true);
    BuyClock.SetActive(false);
    BuyCrystal.SetActive(false);

    NotEnoughtMoney_Crystal.SetActive(false);
    NotEnoughtMoney_Clock.SetActive(false);
}
//Включение описания и кнопки для покупки при нажатии на часы в меню покупок
public void ClickClock()
{
    WingsText.SetActive(false);
    ClockText.SetActive(true);
    CrystalText.SetActive(false);

    BuyWings.SetActive(false);
    BuyClock.SetActive(true);
    BuyCrystal.SetActive(false);

    NotEnoughtMoney_Wings.SetActive(false);
    NotEnoughtMoney_Crystal.SetActive(false);
}
//Включение описания и кнопки для покупки при нажатии на кристалл в меню покупок
public void ClickCrystal()
{
    WingsText.SetActive(false);
    ClockText.SetActive(false);
    CrystalText.SetActive(true);

    BuyWings.SetActive(false);
    BuyClock.SetActive(false);
    BuyCrystal.SetActive(true);

    NotEnoughtMoney_Wings.SetActive(false);
    NotEnoughtMoney_Clock.SetActive(false);
}
//Разрешение или отказ на покупку крыльев
public void buyWings()
{
    if (PlayerPrefs.GetInt("Counter") >= 2 && PlayerPrefs.GetInt("BoughtWings", 0) < 3)
    {
        PlayerPrefs.SetInt("fromTradeWings", 1);
    }
    else
    {
        NotEnoughtMoney_Wings.SetActive(true);
    }
}
//Разрешение или отказ на покупку часов
public void buyClock()
{
    if (PlayerPrefs.GetInt("Counter") >= 3 && PlayerPrefs.GetInt("BoughtClock", 0) < 3)
    {
        PlayerPrefs.SetInt("fromTradeClock", 1);
    }
    else
    {
        NotEnoughtMoney_Clock.SetActive(true);
    }
}
```

```
    //Разрешение или отказ на покупку кристаллов
    public void buyCrystal()
    {
        if (PlayerPrefs.GetInt("Counter") >= 3 && PlayerPrefs.GetInt("BoughtCrystal", 0) < 3)
        {
            PlayerPrefs.SetInt("fromTradeCrystal", 1);
        }
        else
        {
            NotEnoughtMoney_Crystal.SetActive(true);
        }
    }
}
```

Лістинг TradeBuyItem.cs:

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

public class TradeBuyItem : MonoBehaviour
{
  public TradeManager tradeManager;

  void Start()
  {
    gameObject.GetComponent<Button>().onClick.AddListener(() => BuyItem());
  }

  void BuyItem()
  {
    int itemPrice = InventoryManager.currentItems[tradeManager.currentChosenItemID].price;

    if (InventoryManager.currentItems[0].count >= itemPrice)
    {
      InventoryManager.RemoveItem(0, itemPrice);
      InventoryManager.AddItem(tradeManager.currentChosenItemID, 1);
    }
  }
}
```

Лістинг TradeItemContainer.cs:

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

public class TradeItemContainer : MonoBehaviour
{
  public int itemID;
  public TradeManager tradeManager;

  void Start()
  {
    gameObject.GetComponent<Button>().onClick.AddListener(() => ChangeChosenItemID());
    gameObject.transform.GetChild(0).GetComponent<Image>().overrideSprite                                    =
InventoryManager.currentItems[itemID].inventoryView;
    gameObject.transform.GetChild(1).GetComponent<TextMeshProUGUI>().SetText(""                              +
InventoryManager.currentItems[itemID].price);
  }
```

```
  // Update is called once per frame
  void Update()
  {

  }

  public void ChangeChosenItemID()
  {
    tradeManager.currentChosenItemID = itemID;
  }
}
```

Лістинг TradeManager.cs:

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

public class TradeManager : MonoBehaviour
{
  public GameObject TradeMenu;
  public TextMeshProUGUI itemNameText, itemDescriptionText;
  public Animator pressFAnim;
  bool inTradeZone = false;
  public static bool TradeMenuIsON = false;
  public int currentChosenItemID = 1;

  void Start()
  {
    // anim = gameObject.GetComponent<Animator>();
  }

  void ActivateTradeMenu()
  {
    TradeMenu.SetActive(true);

    Button[] tradeMenuButtons = TradeMenu.GetComponentsInChildren<Button>(true);
    // tradeMenuButtons[0].onClick.Invoke();
    tradeMenuButtons[0].Select();

    Time.timeScale = 0;
    TradeMenuIsON = true;
  }

  public void DisactivateTradeMenu()
  {
    TradeMenuIsON = false;
    TradeMenu.SetActive(false);
    Time.timeScale = 1;
  }

  void InTradeZone()
  {
    if (Input.GetKeyDown(KeyCode.F) && !Pause_Menu.PauseMenuIsON && !InventoryManager.InventoryIsON &&
!PortalManager.PortalMenuIsON)
    {
      ActivateTradeMenu();
    }

    if (Input.GetKeyDown(KeyCode.Escape) && TradeMenuIsON ||
```

```
  Input.GetKeyDown(KeyCode.Tab) && TradeMenuIsON)
  {
   DisactivateTradeMenu();
  }
 }

 void Update()
 {
  if (inTradeZone)
  {
   InTradeZone();
  }

  if (TradeMenuIsON)
  {
   itemNameText.SetText(InventoryManager.currentItems[currentChosenItemID].name);
   itemDescriptionText.SetText(InventoryManager.currentItems[currentChosenItemID].description);
  }
 }

 void ActivatePressF(Collider2D col)
 {
  if (inTradeZone)
  {
   pressFAnim.SetBool("Press_F_Show", true);
   pressFAnim.SetBool("Press_F_Hide", false);
  }
 }

 void OnTriggerEnter2D(Collider2D col)
 {
  if (col.tag == "Player")
  {
   inTradeZone = true;
   ActivatePressF(col);
  }
 }

 void OnTriggerStay2D(Collider2D col)
 {
  if (col.tag == "Player")
  {
   inTradeZone = true;
   ActivatePressF(col);
  }
 }

 void OnTriggerExit2D(Collider2D col)
 {
  if (col.transform.gameObject.layer == LayerMask.NameToLayer("Player"))
  {
   inTradeZone = false;
   pressFAnim.SetBool("Press_F_Hide", true);
   pressFAnim.SetBool("Press_F_Show", false);
  }
 }
}
```

Лістинг Ambient.cs:

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

```
public class Ambient : MonoBehaviour
{
   //Переменная хранящая в себе компонент с эмбиентом
   private AudioSource Source;

   void Start()
   {
      Source = GetComponent<AudioSource>();
   }
   //При входе в территорию
   void OnTriggerEnter2D(Collider2D other)
   {
      if (other.tag == "Player")
      {
         Source.Play();
      }
   }
   //При выходе из территории
   void OnTriggerExit2D(Collider2D other)
   {
      Source.Stop();
   }
}
```

Лістинг AmbientManager.cs:

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

public class AmbientManager : MonoBehaviour
{
 public AudioMixer audioMixer;
 AudioSource audioComponent;
 bool ambientIsWorking = false;
 IEnumerator audioCoroutine;

 void Start()
 {
  audioComponent = gameObject.GetComponent<AudioSource>();
 }

 IEnumerator PlayAmbient(AudioClip ambient)
 {
  ambientIsWorking = true;
  audioComponent.clip = ambient;
  audioComponent.Play();

  string exposedParam = "OtherSoundsVolume";
  float duration = 4f;
  float targetVolume = SaveSystem.optionsData.soundsVolume;
  float currentTime = 0;
  float currentVol;

  audioMixer.GetFloat(exposedParam, out currentVol);

  while (currentTime < duration)
  {
   currentTime += Time.deltaTime;
   audioMixer.SetFloat(exposedParam, Mathf.Lerp(currentVol, targetVolume, currentTime / duration));
   yield return null;
```

```
    }

    yield return new WaitForSeconds(ambient.length);

    audioComponent.Stop();
    ambientIsWorking = false;
  }

  IEnumerator FadeAmbient()
  {
    string exposedParam = "OtherSoundsVolume";
    float duration = 4f;
    float targetVolume = -80;
    float currentTime = 0;
    float currentVol;

    audioMixer.GetFloat(exposedParam, out currentVol);

    while (currentTime < duration)
    {
      currentTime += Time.deltaTime;
      audioMixer.SetFloat(exposedParam, Mathf.Lerp(currentVol, targetVolume, currentTime / duration));
      yield return null;
    }

    audioComponent.Stop();
    ambientIsWorking = false;
  }

  public void AddAmbient(AudioClip ambient)
  {
    if (!ambientIsWorking)
    {
      audioCoroutine = PlayAmbient(ambient);
      StartCoroutine(audioCoroutine);
    }
  }

  public void StopAmbient()
  {
    if (ambientIsWorking)
    {
      StopCoroutine(audioCoroutine);
      StartCoroutine(FadeAmbient());
    }
  }
}
```

Лістинг AmbientTrigger.cs:

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

public class AmbientTrigger : MonoBehaviour
{
  AmbientManager ambientManager;
  public AudioClip ambient;
```

Решта коду буде на диску.

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

" Розробка кросплатформного ігрового застосунку на основі технологічного стеку Unity/C#"

студента групи 121-19-1 Баляна Андрія Давідовича

Керівник економічного розділу доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Балян.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Балян.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Балян.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Балян.ppt	Презентація кваліфікаційної роботи