ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему:

«Розробка гри жанру платформер на мові програмування C# за допомогою ігрового двигуна Unity»

Виконав: студент 4 курсу, групи ПД-44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Рудий А.С.
(прізвище та ініціали)
Керівник Дібрівний О.А.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти – «Бакалавр»
Напрям підготовки – 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
Негоденко О.В
“ ____ ” _____ 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Рудому Андрію Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка гри жанру платформер на мові програмування C# за допомогою ігрового двигуна Unity

Керівник роботи Дібрівний О.А., доктор філософії, доцент _____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “18” лютого 2022 року № .

2. Строк подання студентом роботи 03.06.2022
3. Вхідні дані до роботи:
   3.1 Технічна документація
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1 Розробка комп’ютерної гри у жанрі Платформер, використовуючи мову програмування C# за допомогою ігрового рушія Unity та середу розробки Visual Studio для роботи з даною мовою
   4.2 Опис та класифікація предметної області, визначення функціоналу та алгоритму проекту
   4.3 Аналіз існуючих ігор у цьому жанрі
   4.4 Дослідження рушіїв
   4.5 Визначення функціоналу та алгоритму проекту
5. Перелік графічного матеріалу:
   5.1 Струтурна схема алгоритму

5.2 Діаграма класів
5.3 Use-case діаграма
5.4 Висновки

6. Дата видачі завдання 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04 - 14.04	
2	Вивчення та аналіз задачі	15.04 - 17.04	
3	Розробка структури додатку	18.04 - 21.04	
4	Розробка дизайну та графічних елементів	22.04 - 25.04	
5	Програмна реалізація системи	26.04 – 05.05	
6	Налагодження програми	05.05 – 07.05	
7	Вступ, висновки, реферат	07.05 – 10.05	
8	Розробка обов‘язкових демонстраційних матеріалів	11.05 – 15.05	
9	Попередній захист роботи	16.05 – 01.06	
10	Здача роботи	03.06	

Студент _____ Рудий А.С
(підпис) (прізвище та ініціали)
Керівник роботи _____ Дібрівний О.А
(підпис) (прізвище та ініціали)

Реферат

Текстова частина бакалаврської роботи 83 с., 34 рис., 1 табл., 26 джерел.

ГРА UNITY, 2D РОЗРОБКА, ЖАНР ПЛАТФОРМЕР, РУШІЇ, РЕАЛІЗАЦІЯ ГРИ ЗА ДОПОМОГОЮ МОВИ C#, VISUAL STUDIO.

Об'єкт дослідження – реалізація гри у жанрі Платформер.

Предмет дослідження – гра платформер за допомогою рушія Unity на мові програмування C#.

Мета роботи – аналіз засобів розробки ігор, щоб надалі розробити гру у жанрі Платформер. Створення гри з основними механіками жанру познайомити потенційних гравців з таким цікавим жанром. Отримання цінного досвіду в різних областях, та оволодіти навичками роботи з рушієм, щоб надалі набагато швидше працювати з іншими проектами.

З поставленою метою в проекті вирішили наступні питання:

- Аналіз засобів розробки ігор;
- Аналіз аудиторії;
- Взявши до уваги всі аналізи, розробили гру у жанрі Платформер.

Дана гра дасть змогу гравцеві відчути жанр Платформер, без прикладання великих зусиль розібратись із механіками жанру та вникнути у сюжет.

В роботі виконано аналіз існуючих ігор у жанрі Платформер

Проаналізовано можливість середовища розробки Visual Studio

Галузь використання – завдяки аналізу, розроблена гра охопить широку аудиторію, які хочуть познайомитись із жанром.

ЗМІСТ

ВСТУП

Зовсім недавно індустрія комп'ютерних ігор стала однією з провідних у світі позицій. Це має просте пояснення – стрімке зростання технологій з кінця 20 століття, а також поява інтернету. Завдяки цій причині комп'ютерна гра стала більш доступною та популярною, ніж інші види розваг. В даний час розробка нових комплектуючих для комп'ютерів тісно пов'язана з ігровою індустрією, оскільки якість та вимоги зростають, що дає можливість розвивати комп'ютерні технології.

Також варто враховувати, що комп'ютерні ігри є не лише однією з видів розваг сучасних людей. Наприклад, у багатьох сучасних інтернет-ресурсах використовуються комп'ютерні програми для підготовки, створюються системи для синхронізації, які використовуються для підготовки фахівців різних професій та напрямків. На жаль, у країнах Західної Європи розвиток ігрового ринку не досягає міжнародного рівня розвитку.

Перший загальний розвиток комп'ютерної культури до нас прийшов пізно, і для розвитку її не було належних спеціалістів та навичок. Таким чином, тема розробки гри дуже актуальна в країнах СНД, вимагає розвитку та появи нових фахівців та компаній-розробників, щоб вони змогли конкурувати із закордонним розробником та вийти на світову ринкову нішу.

ДОСЛІДЖЕННЯ КЛАСИФІКАЦІЙ ТА АНАЛІЗ ІСНУЮЧИХ РОЗРОБОК КОМП’ЮТЕРНИХ ІГОР

1.1 Аналіз та загальна характеристика комп’ютерних ігор

Комп'ютерна гра є програмою, яка служить для організації процесу взаємодії з партнерами в грі або сама виступає партнером. Під час гри спеціальні програми створюють імітації прямого зв'язку у віртуальному просторі між ігровим персонажем та гравцем (або групою гравців) за певними алгоритмами. Спостерігаючи за ігровою ситуацією на моніторі, гравець на неї впливає за допомогою клавіш, «миші» та джойстиків, тощо.

Комп'ютерна гра часто створюється на основі інших сторонніх джерел, наприклад, книг та фільмів, але останнім часом з'являються приклади зворотного зв'язку, коли в основі відомих ігор або ігрових серій починають появлятись додаткові матеріали для розширення всесвіту ігор.

Крім того, спеціальні ігри можна використовувати як навчальний матеріал або дозволяють використати гравця у наукових дослідженнях. Деякі популярні ігри проводять змагання різного масштабу від регіонального до світового, що мають окреме ім'я «кіберспорт».

Тому деякі європейські навчальні заклади почали застосовувати відомі ігри для навчання. Наприклад симулятори тренування солдатів для армії.

Взявши це до уваги, можна легко сказати, що на сьогоднішній день, ігри досить популярні та являються актуальними у різних сферах життя

Що до класифікації, то її немає досихпір. В основному на поділ жанрів впливають дії, які відбуваються в грі.

Виділемо основні жанри ігор.

Таблиця 1 - Жанри ігор

• Action-RPG • Roguelike	• MMOFPS • Survival	• RTS • MOBA
• Open RPG	• Open Action	• Global Strategy
• RPG • MUD MMORPG	• Action • Slasher • Battle Racing	• Strategy • Sim Strategy
• Puzzle • Quest • Browse RPG • Adventure	• Platformer • Stealth-Action Fighting • Racing	• Economical • Tower Defense • Wargame • Cardgame
• Education • Test • Contact • Hero • Toure	• Arcade • Horror • Shooter • Sport • Simulator	• Logic • Tactic • MicroControl • Building • Life Sim
1.1 Навчання 1.2 Загадки 1.3 Спілкування 1.4 Роль 1.5 Вивчення	2.1 Збирання 2.2 Ухилення 2.3 Нищення 2.4 Змагання 2.5 Водіння	3.1 Турбота 3.2 Створення 3.3 Контроль 3.4 Тактика 3.5 Планування

Всі ігри розділені на одиночні та мультиплеєр. Мультиплеєри в свою чергу поділяються на:

- Мультиплеєр для одного комп'ютера;
- Мультіплеєрні офлайн ігри;
- Масова онлайн гра.

Саме через цей класифікатор встановлено режими, які присутні в ігри. Одиночна гра – це вид гри, у якому бере участь одна людина. Зазвичай гравець захищається штучним інтелектом, і його завдання – пройти до кінця ігрового процесу, накопичити ресурси або прокачувати навички. Зазвичай завдання комбінуються.

Інший вид ігор – мультиплеєрна гра. Цей режим гри підходить для гри більш ніж однієї людини. У деяких іграх комбінується одиночний та мультиплеєрний режими.

Ігри також поділяються на текстову(де основні дії відбуваються за допомогою тексту), у 2D (двовимірна графіка) та 3D (тривимірна графіка)

Для першої розробки, враховуючи актуальність жанрів, було обрано гру Платформер. Ігри даного жанру мають зрозумілу і просту механіку, що являється великим плюсом для розробника-початківця, так як на 3D потрібна команда та більше часу.

1.2 Характеристика комп’ютерних ігор та їх аналіз

Платформер – жанр гри, де головний персонаж проходить рівень пригаючи по платформах, йому в цьому заважають вороги та пастки. У продовж гри герой колекціонує предмети, необхідні для завершення рівнів. Традиційно персонаж у перших двовимірних платформерах міг рухатись лише зліва направо. Проте у сучасній грі головний персонаж може рухатися у будь-якому напрямку.

Під час проходження гри, підняті предмети можуть давати бонуси, наприклад додаткове здоров’я, додатковий захист персонажа, зброя, тощо.

Що до ворогів, вони можуть мати будь-який облік, як видуманих створінь, так і реальних. Вони наділені штучним інтелектом, завдяки чому вони намагаються максимально наблизитися до персонажа і нашкодити йому, або не мають інтелекту зовсім. Їх пересування зазвичай задається по певній траєкторії. Якщо близько наблизитись до ворога, то він почне атакувати. Влучивши в головного персонажа, буде зменшуватись здоров’я після чого гравець може померти. Інколи вороги

можуть, видавати звуки, тікати або змінювати зовнішність. Нові ігри цього жанру зазвичай мають механіку бою зброєю ближнього та дальнього бою.

Рівні нерідко мають приховані предмети та проходи в стінах, які допоможуть гравцеві полегшити проходження гри.

Більшість ігор платформерів являють в собі фентезі-елементи та мальовану графіку. Головними персонажами як і вороги, можуть мати будь-який облік, наприклад: тварини, динозаври, ельфи.

Жанр платформер почав набирати обертів на початку 1980-х років. Тоді з'явилися легендарні платформери Pitfall і Super Mario Bros. Завдяки ним, стрімко почав розвиватись цей жанр.

Гра Pitfall – одна з перших ігор, яка показала гравцям горизонтальне проходження гри. (Рис.1.1)

Рис. (1.1) Гра Pitfall

Super Mario Bros – на той час не було багато пам’яті на комп’ютерах. У цій грі розробники показали, що це не є проблемою і створили гру, яка вміщувала в собі великі різноманітні рівні із різною складністю. (Рис. 1.2)

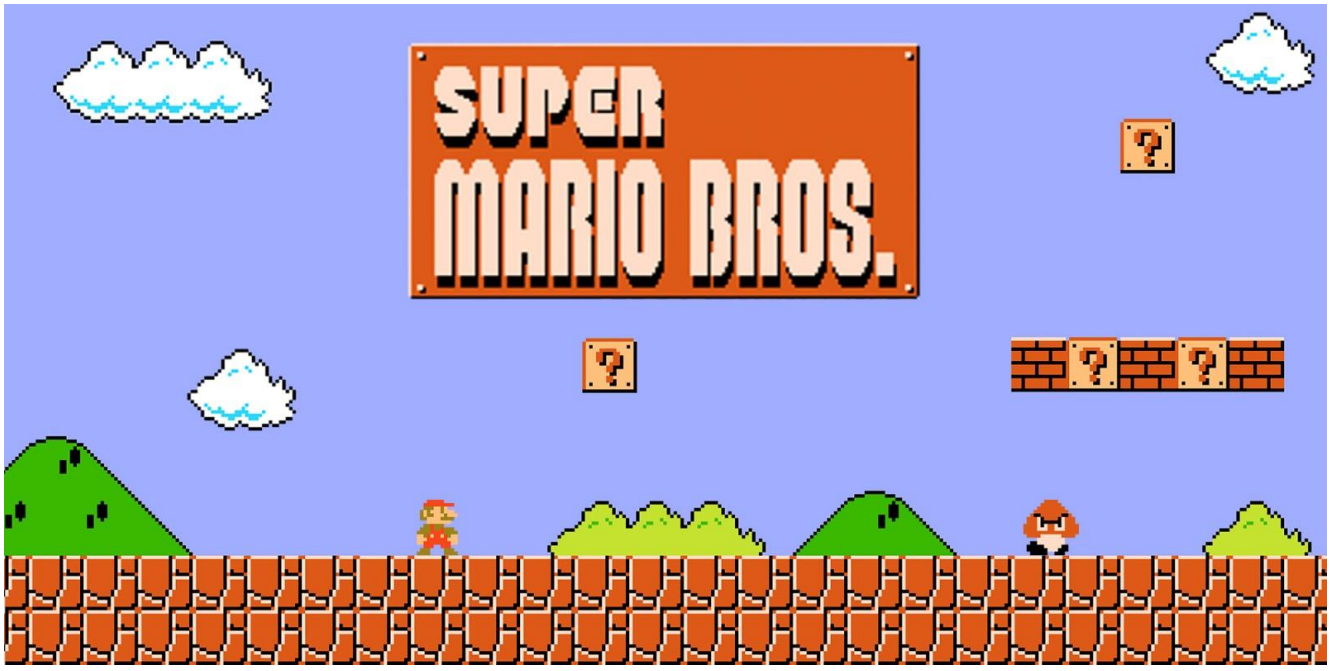


Рис. 1.2 Гра Super Mario Bros.

У зв'язку з тим, що технології почали розвиватися з кінця 1980-х, ми аналізуємо саме сьогоднішні приклади платформерів, наприклад:

- Cuphead;
- Starbound;
- Dead Cells;
- Mark of the Ninja.

1.2.1 Cuphead

Відеогра в жанрі платформер, створена та опублікована канадською компанією StudioMDR Entertainment у 2017 р. для PC та XBox One. Гра була розроблена на ігровому рушії Unity. Гра має стилістику мультфільмів Disney 1930-х років.

Головний персонаж - Cuphead, хлопчик у якого чаша замість голови. Він бореться з ворогами за допомогою пострілів із пальця. Програвши суперечку з дияволом, гравцю потрібно перемогти босів, щоб повернути борг та отримати свою душу.

Вибір рівнів здійснюється через зовнішню карту світу, виконаній у стилі гри жанру ActionRPG . Сама карта - це розгалужена послідовність рівнів та має свої таємні проходи та магазин, в якому персонаж може отримати додаткове життя та різні вміння для того, щоб боротися із ворогом.

Крім звичайного пострілу з пальця, Cuphead також має навички парування різних видів атак та об’єктів долонями. Після успішного парування накопичується спеціальна шкала для того, щоб використовувати інші спеціальні навички. Загалом система бойових дій у грі складається з численних битв з босами. Отримавши перемогу, гравець має можливість переглянути статистику успіхів гравця, щоб при бажанні можна було підвищити свої результати.

Можна відмітити чудову механіку, це кооперативний режим. Що є рідкістю у цьому жанрі. Там ви будете грати за персонажа Mugman та допомагати головному персонажу. (Рис. 1. 3)

Рис 1.3 Гра Cuphead.

1.2.2 Starbound

Гра платформер змішена із жанром пісочниці, створена та випущена в 2016 році студією Xbox Games для портативних комп'ютерів, PlayStation, Xbox One. Гра написана повністю програмною мовою C++ на власному ігровому рушії. (Рис. 1. 4)

Ця гра зацікавила багатьох гравців своїм сюжетом. Спочатку ви появляєтесь на планеті, яка генерується у процедурному режимі. Там ви вільні робити, все що вам довподоби, наприклад: викопувати цінні ресурси, бігати у пошуках скарбів, будувати будинок, тощо. Вам дається вибір рас, за яких ви можете грати. Вони мають чуттєві відмінності.

Сюжет розпочнеться з появи на кораблі, де обираєте расу. Це також вплине на сюжет.

Ви будете подорожувати на своєму кораблі по всесвіту, проходячи багато різноманітних завдань.

Також цікавим елементом гри являється випадкова генерація. Вона охопить зовнішній вигляд планет, матеріалів, відмінність ресурсів, тощо.

Рис 1.4 Гра Starbound.

1.2.3 Dead Cells

Комп'ютерна гра змішаного жанру rogue та платформера розроблені французькою компанією «Motion Twin» для Windows, Apple та Linux, а також для консолей Nintendo Switch, PlayStation 4 та Xbox One. (Рис. 1. 5)

У Dead Cells гравець керує істотою, яка намагається вийти із лабіринту. Так як і в минулій грі, рівні генеруються процедурним режимом. Також це стосується і ворогів, різних бонусів, озброєнння, тощо. Гра в жанрі rougelike, де персонаж Dead Cells у разі смерті повинен спочатку розпочати гру. Але після смерті втрачається не все. Під час проходження ви прокачуєте навички, які залишаться після вашої смерті.

Під час гри потрібно знаходити нове озброєння, та вивчати локацію. Також можна знаходити бонуси, які будуть покращувати зброю.

Після смерті, рівень буде згенеровано по новому, що додає складності для проходження. Адже на цих рівнях є потайні проходи, які відкриті лише деякий час. Якщо гравець не встигає, він більше не зможе попасти і йому прийдеться прикласти більше зусиль, щоб пройти рівень.

Незважаючи на складність гри, вона має досить високу оцінку. Цьому сприяє як гра виглядає візуально та механікам гри.

Рис 1.5 Гра Dead Cells.

1.2.4 Mark of the Ninja

Гра яка вміщує в собі не лише жанр платформер, а й елементи інших жанрів, такі як екшн та стелс. Гра була розроблена для PC та Xbox One Канадською компанією Klei Entertainment. (Рис 1. 6)

У даній грі досить цікавий сюжет. Ми будемо грати за безіменного ніндзю. Його надзвичайна сила надходить із його татуювання, проте із-за неї він поступово сходить з глузду. Його основна задача врятувати членів свого клану.

Головний герой буде проходити через безліч пасток, та буде намагатись уникати зустрічі з ворогом, щоб без шуму пройти далі. Але якщо дійде до бою, то на озброєнні буде меч та дротики для того, щоб відвернути увагу ворога, або знешкодити світло.

Таким чином гра надає два варіанти проходження, де гравець уникає всіх противників, або знешкоджує всіх на своєму шляху.

Під час проходження гравець буде знаходити нове озброєння та накопичувати бали. За них можливо буде вдосконалювати озброєння, броню та відкривати нові види прийомів.

Ця гра - чудовий стелс-екшн серед двовимірних ігор такого типу. Жанр, якого не так багато останніми роками виробляється. Вміщає в собі приємні графічні аспекти, що створює чудову атмосферу впродовж всієї історії.

Рис 1.6 Гра Mark of the Ninja

Висновок до розділу 1

Після аналізу, можна дійти до висновку, що завдяки стрімкому розвитку жанру Платформер, гравців стало важко вражати, тому розробники почали добавляти механіки із різних жанрів. Це дало розробникам можливість отримувати більший прибуток за рахунок більшого охоплення аудиторії.

Підвівши підсумки, стало зрозуміло, що на теперішній час, щоб гра була актуально, цікавою та успішною, потрібно використовувати механіки з інших жанрів.

РОЗДІЛ 2

РОЗРОБКА ПРОЕКТУ

2.1 Аналіз та вибір рушія для розробки проекту

Ігровий рушій – програма для розробки комп'ютерних ігор та програм. Для аналізу обрали кілька рушіїв, таких як:

- Cry Engine 5
- Unreal Engine 4
- Unity 2019.

2.1.1 Cry Engine 5

Рушій, у якому можна створити неймовірно реалістичну графіку. Перша гра, яка вийшла на екрани Cry Engine, це Far Cry 2004-го року. (Рис. 2.1)

Рис 2.1 Гра Far Cry

Cry Engine 5 має підтримку таких платформ як: Windows, PlayStation 4, Xbox One та Oculus Rift. Не зважаючи на потужність рушія, для роботи з ним потрібні не високі характеристики:

- PC із операційною системою Windows 7, 8, 10 64x;
- Відеокарта NVIDIA GeForce 450, AMD Radeon HD 5750 (або вище з підтримкою DirectX 11)
- Процесор Intel Dual-core (або інший від 2GHz);
- Оперативна пам’ять 4 GB;

Що до інтерфейсу, тут не виникає питань. Він простий та зрозумілий. (Рис. 2.2)

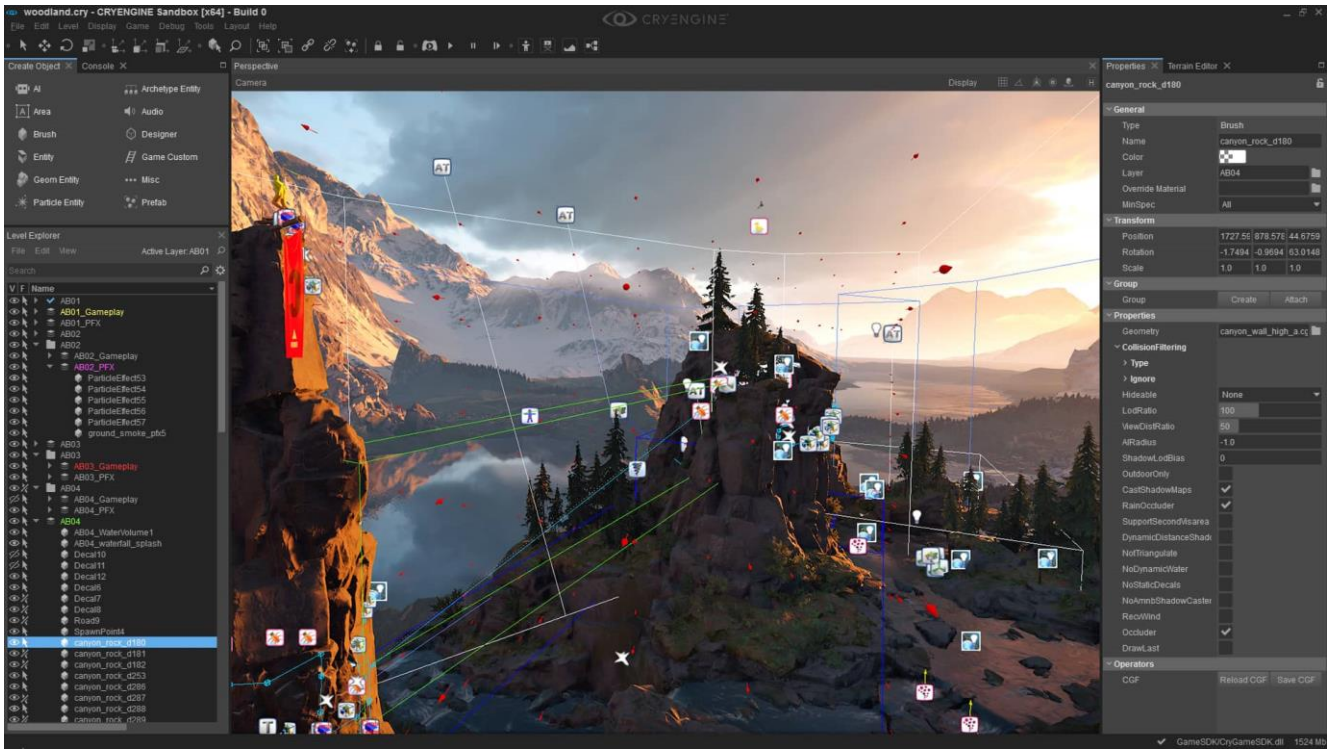


Рис 2.2 Інтерфейс Cry Engine 5

Cry Engine має великий функціонал для роботи з графікою. Він має підтримку DirectX 12.

Для роботи з матеріалами допоможе редактор SandBox.

Рушій має вбудований редактор Cry Engine SandBox для керування матеріалами. Завдяки ньому, відкривається великий інструментарій для побудови рівнів та ігрових світів.

Cry Engine має чудову систему FlowGraph. За допомогою неї можна створити та контролювати ігрову логіку без скриптів.

Designer Tool потрібен для змінення моделей.

У рушії є редактор для створення відео під назвою TrackView.

Physical Rendering, система для роботи зі світлом, яка дає змогу створити реалістичні джерела. Цей рушій має інструменти, здатні ефективно згладжувати та зменшувати пікселізацію зображення.

Рушій з легкістю справиться із кістковою анімацією. Також на допомогу можна застосувати систему, яка дозволяє створити реалістичну поведінку.

У Cry Engine можливо провести аналіз продуктивності під час гри.

Також чудовий інструмент у рушії, це Statoscope (Рис. 2.3). За допомогою нього, можна детально розглянути витрати ресурсів PC на графіку, що надалі дуже спростить оптимізацію проекту.

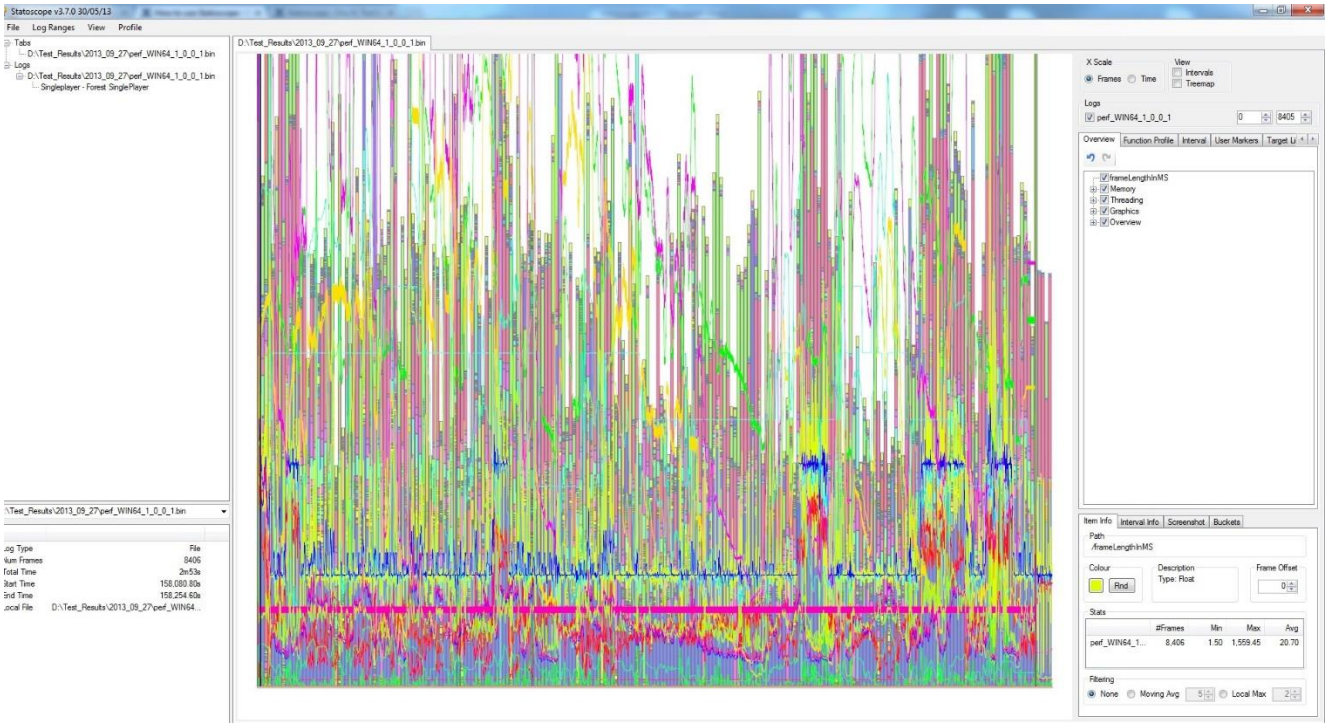


Рис 2.3 Інструмент Statoscope.

Рушій Cry Engine має офіційний сайт з власним магазином. Там знаходиться навчальна документація та потрібні матеріали для розробки.

Cry Engine має вільну ліцензію і його можна використовувати будь-кому. Проте, якщо проект створений у цьому рушії приносить дохід більш ніж п’ять тисяч доларів у рік, то 5% від прибутку потрібно перерахувати компанії Crytek.

Маючи такий функціонал, з легкістю можна сказати, що даний рушій відмінно підійде для великих проектів у 3D.

2.1.2 Unreal Engine 4

Розробники Unreal Engine є компанія Epic Games. Їх розвиток розпочався із гри шутера Unreal у 1998 р. (Рис. 2.4). На початку Unreal Engine призначиний був для шутерів від першої особи, однак в наступних версіях використовувався для розробки ігор різних жанрів таких як:

- Стелс
- Fighting
- Мультиплеєрні рольові ігри

Рис 2.4 Гра Unreal

Unreal Engine підтримує багато існуючих платформ та операційних систем таких як: Windows, Linux; MacOS; Xbox One; Xbox 360; Sony PlayStation 2, Sony PlayStation 3, Sony PlayStation 4, Android; iOS; Nintendo Wii, Nintendo Wii U Switch, PSP, Dreamweaver. Щоб без проблем користуватись рушієм Unreal Engine, комп'ютер повинен мати слідуючі мінімальні системні вимоги:

- Операційна система Windows 7, 8, 10 64x, MacOS 10.13, Linux Ubuntu 15.04;
- Відеокарта NVIDIA GeForce 470, GTX AMD Radeon 6870 HD та вище
- Процесор Intel Quad-core чи AMD 2.5Hz та вище;
- Оперативну пам’ять 8 GB;

Рушій досить зручний, має легкий для розуміння інтерфейс. (Рис 2.5)

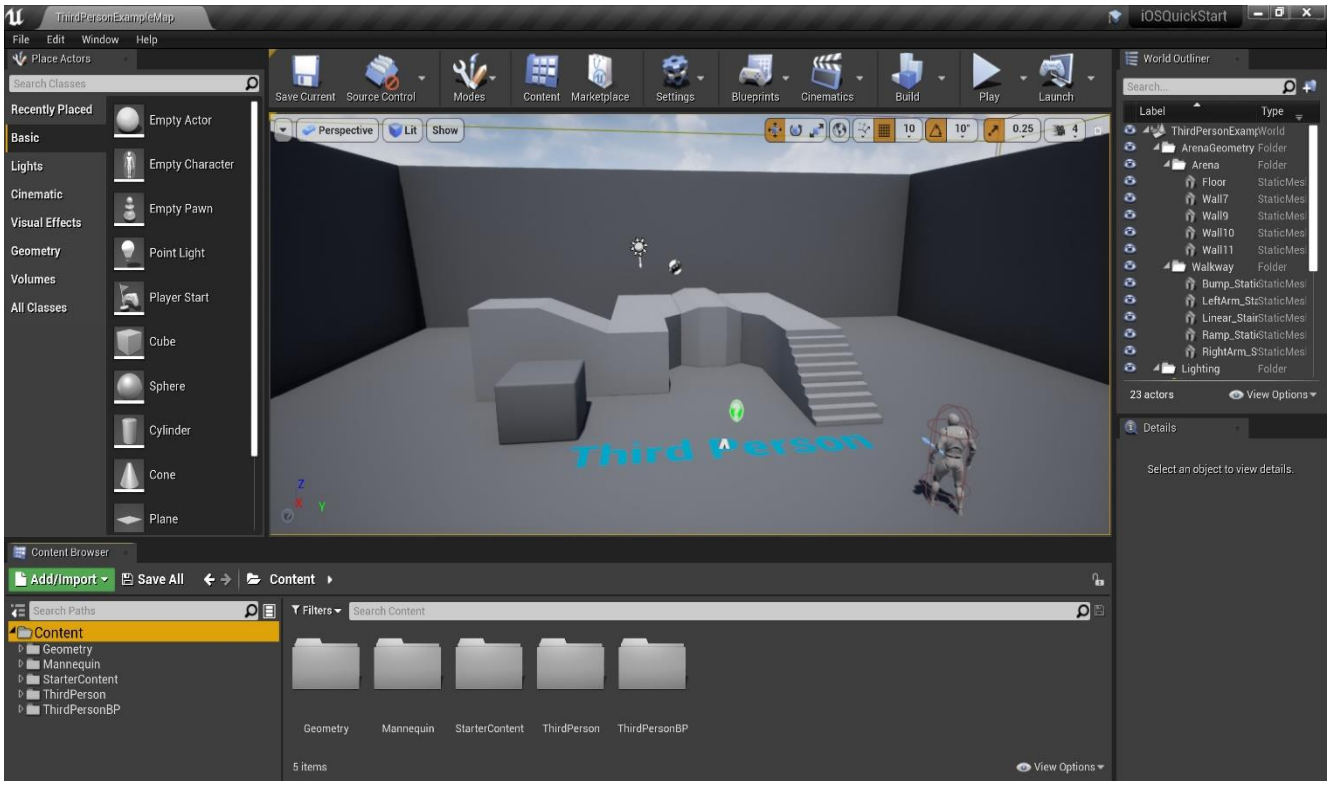


Рис 2.5 Інтерфейс Unreal Engine 4.

Так як і в минулому рушії, в Unreal Engine можливо реалізувати логіку без коду за допомогою Blueprint.

Даний рушій має достатню кількість налаштувань для створення чудової графіки.

Що до відео, як і минулий рушій, Unreal Engine має інструмент Sequencer. За допомогою нього з легкістю можна настроїти камеру, світло та об’єкти.

В Unreal Engine можливо працювати з віртуальною реальністю, так як з ними працюють популярні розробники цих засобів. Тому не дивно, що даний рушій чудово підходить для даної розробки.

Unreal Engine не виключення стосовно документації та навчальних матеріалів. В його магазині достатньо інформації як для новачків, так і для розробників з досвідом.

Unreal Engine також має вільну ліцензію з 2015 року. Проте, якщо проект, розроблений в Unreal Engine, приносить дохід понад 3 тисячі доларів за квартал, то розробнику слід перерахувати 5 відсотків від прибутку компанії Epic Games.

Підсумувавши підсумки, наглядно видно, що даний рушій як і Cry Engine підходить в основному для 3D розробки, але дає більший вибір платформ.

2.1.3 Unity 2019

Unity – рушій, розроблений компанією Unity Technologies. Unity має підтримку більшості платформ, такі як: Windows, Linux, MacOS, FireOS, IOS, Android, Android TV, Xbox One, PlayStation Vita, PlayStation 4, PlayStation VR, Oculus Rift, Nintendo Switch, , Gear VR, Nintendo 3DS, Steam VRSmart TV, TvOS.

Щоб займатись розробкою на Unity потрібен комп’ютер з такими мінімальними вимогами:

- Операційна система Windows 7, 8, 10, 64x; Mac OS X 10.12+; Ubuntu 16.04, 18.04, CentOS 7;
- Відеокарта з підтримкою DirectX 10;
- Процесор з підтримкою SSE2.

В Unity дуже зрозумілий та легкий інтерфейс, що дає розробникам початківцям швидко розібратись. (Рис 2.6)

Рис 2.6 Інтерфейс Unity.

Unity є різнобічною програмою розробки. Вона підходить як для програмістів, так і для художників, завдяки різноманітним інструментам таких як:

- Progressive Lightmapper – інструмент для освітлення, в основі якого лежить трасування шляху з прогресивним оновленням;
- Autodesk Maya – програма для створення гарного контенту у 3D. До нього можна віднести і моделі, анімації, візуальні ефекти, ігри та моделювання;
- Timeline – чудовий інструмент для створення катсцен(анімаційні сцени). Можна використовувати готові ігрові моделі, що допоможе досить сильно заощадити час та фінанси;
- Cinemachine – набір інструментів для створення динамічних камер, без додавання коду. Cinemachine дасть змогу налаштовувати, розробляти, задавати поведінку камери в реальному часі.

У Unity дуже легко працювати як з 3D так і 2D проектами завдяки вбудованим інструментам, наприклад для роботи з 2D об’єктами Box2D.

Що до оптимізації, то Unity справляється з цим чудово. Розроблювані проекти працюють швидко та якісно.

Рушій має підтримку таких мову програмування, як C# та UniScript.

Рушій Unity не поступається місцем з іншими, які аналізували раніше. Сайт Unity надає велику кількість документації різного характеру. Якщо розробник новачок, він з легкістю за короткий термін розбереться із потрібною йому задачею.

Варто мати на увазі, що є магазин Asset Store. У ньому можливо знайти майже все для розробки, навіть скрипти.

У цього рушія є 3 варіанти придбання:

- Personal – безкоштовна, якщо прибуток не переважає в 100 тисяч доларів;
- Plus – 35 доларів за місяць, якщо прибуток не переважає за 200 тисяч доларів;
- Pro – 125 доларів за місяць, тут прибуток допустимий без обмежень.

Оскільки основними критеріями для вибору середовища для розробки є вільна ліцензія, документи, простий інтерфейс, низькі вимоги для комп’ютера та інструменти для розробки 2D-проектів. Тому в нашому випадку зупинемось на ігровому рішії Unity.

Писати наші скрипти будемо у такій програмі, як Microsoft Visual Studio. (Рис 2.7)

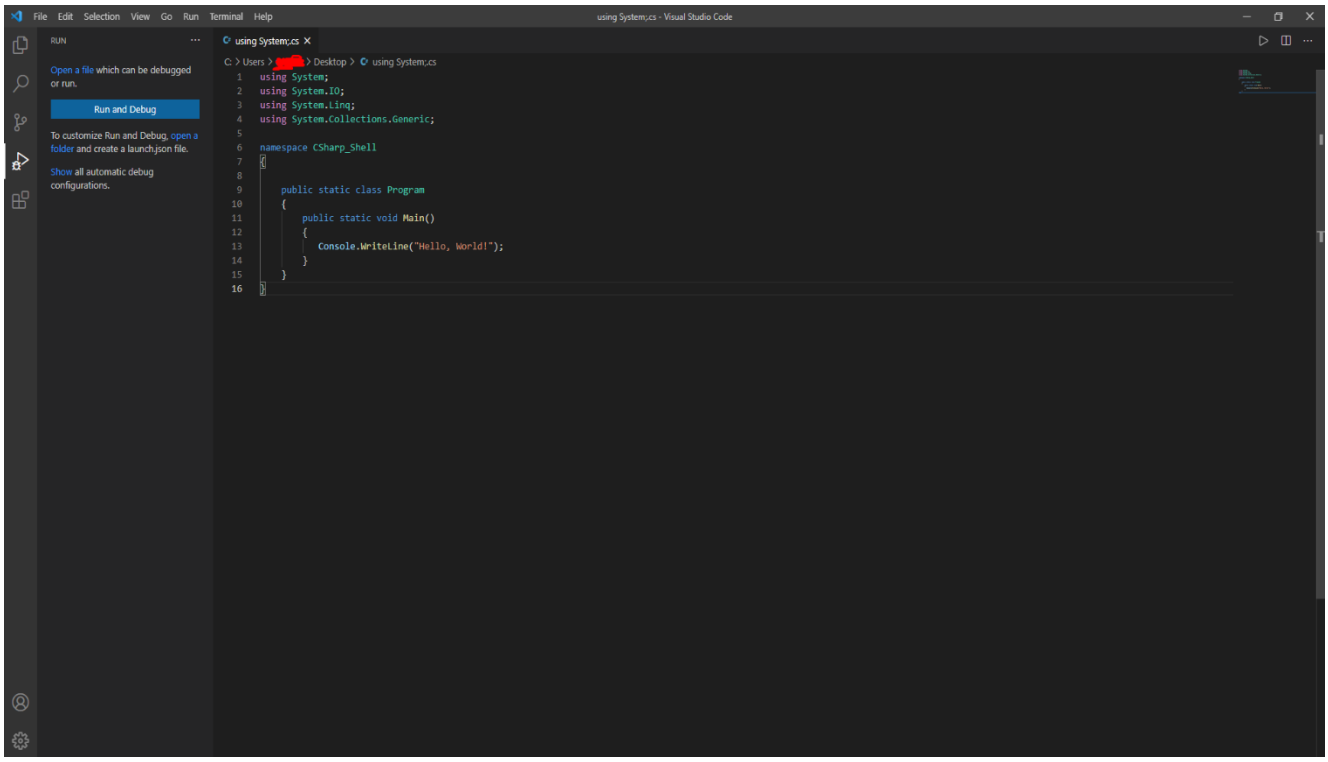


Рис. 2.7 Інтерфейс програми Visual Studio.

Розробники цього середовища для розробки, являється компанія Microsoft. До його складу входить редактор вихідного коду та можливість рефакторінгу. В нашому випадко, це найкращий вибір середовища для розробки.

2.2 Алгоритм реалізації проекту

Алгоритми розробки мало чим відрізняються від розробки іншого проекту. Відзначимо 3 етапи:

- Проектування;
- Розробка;
- Реліз та підтримка.

На етапі проектування визначаємо мету гри, застосунки за допомогою яких будемо розробляти гру.

Під час визначення мети, потрібно виділити ідею, жанр та сетинг гри. Ідея це те, що заманює гравца пограти у гру. Це також стосується жанру. Наприклад,

основною ідеєю РПГ, це дозволити гравцю відігравати роль, яку він сам обере. Що до шутерів, вони надають можливість поринути у реальні чи видумані бої. Тому, визначивши основні ідеї ігрового жанру, скоріше за все жанр визначиться майже відразу.

Після жанру та ідеї гри, наступним кроком буде вибір стилю гри. Сеттинг, це місце, де відбувається основний ігровий процес. Від нього залежить місце, час, умови дій. Зазделегіть вибраний сеттинг значно полегшить створення сценарію, орієнтуючись на цільову аудиторію.

Засоби розробки відносяться до програмного коду та ігрового рушія. Швидкість розробки залежить від вибору засобів для неї. Що до програмного коду, він залежить від обраної платформи у якій буде створюватись гра. Наприклад, якщо гра створюється для персонального комп'ютера, то скоріш за все це будуть мови програмування C# або C++, але якщо гра для браузерів, то тут скоріш за все вибір впаде на такі мови як Java або Flash.

Далі починається етап розробки проекту.

Великий та довгий етап, який вміщує в собі багато різних кроків, без яких проект не буде працездатним.

По-перше, потрібно визначити сюжет та механіку гри. Комусь може показатись, що сюжет не важливий, проте він грає не останню роль. Завдяки сюжету гравцю стає цікаво грати. Сюжет поділяють на 2 види:

- Літературний сценарій – опис персонажів гри та подій, які відбуваються в ігрі
- Режисерський сценарій – інформація про події на рівні гри, які проходять саме на цьому рівні

Ігрова механіка базується на цілях гри, де визначає правила та об'єкти, за якими персонаж взаємодіє. Як правило, розробка механік гри іде разом із написання ігрового сюжету.

Також на цьому етапі розпочинається рання обробка графічних елементів та дизайну ігри. Операючись на сюжет та попередньо розроблений дизайн,

створюються перші концепти, по яких надалі будуть розроблятись зовнішній вид гри та персонажі.

Далі йде, безпосередньо, розробка гри.

Маючи концепт, починається створення персонажів, об’єктів гри та паралельно розробка рівнів гри. Під час розробки, створюється простий схематичний рівень, де буде показато предмети, з якими буде взаємодіяти персонаж.

У такий спосіб створюється перший рівень. Це пуста локація з мінімальною кількістю об’єктів потрібних для проходження. Цей рівень використовується для тестування, а саме чи можливо його пройти. Пройшовши успішне тестування додаємо об‘єкти на сцену.

Після цих етапів, у нас повинно бути декілька рівнів, настає час прототипу гри (альфа-версія). Ця версія потрібна розробнику, щоб він міг провести тестування основних механік гри та перевірити, наскільки вони відповідають вимогам розробника. В альфа версії в основному розробник залишає об'єкти без текстур.

Після успішного тестування альфа-версії, йде опрацювання механік та об’єктів гри.

Далі йде етап розробки повноцінного рівня та механізму гри та додавання додаткових елементів, наприклад: кат сцени, відеоролики, діалоги, тощо. Перші сюжетні моменти у грі, наприклад, відео, сюжетна діалогія та катання. Паралельно виправляються помилки в коді, виявлені на тесті.

Слідуючий етап, це перехід з альфа в бета-версію. Бета-версія потрібна для перевірки гри на несправності. У цій версії допустимі елементи гри, які не докінця завершені, але вони не повинні впливати на ігровий процес. Якщо проект має мультиплеєр, то для знаходження різних недоліків заохочують звичайниї гравців. Це значно скорочує час тестування і дає змогу забрати частину роботи з команди розробників.

Після бета-тесту, гру допрацьовують, виправляють помилки та виходить в реліз.

Далі залишається лише підтримувати гру. Основна задача підтримки, це випуск патчів, де виправляють помилки у готовій грі. Також для заохочування гравців продовжувати грати, для неї додають додатковий новий контент під назвою DLC. Туди можуть входити нові локаціх, предмети, можливості для персонажа та інше.

Проаналізувавши етапи розробки, наглядно видно, що розробка гри не сильно відрізняється від розробки інших проектів.

2.3 Аналіз потенційної аудиторії гри

Аналіз аудиторії, це невід'ємна частина будь-якого проекту. Розуміння аудиторії та що їй потрібно, дає розробнику чітке бачення для розвитку.

Провівши аналіз, було обрано аудиторію від 15 до 40 років. Ця категорія вибрана з аналізу середнього віку людей, які активно грають.

Цільова аудиторія гри, це люди які мають бажання ознайомитись із даним жанром. Якщо це ознайомлення, щоб не злякати потенційних гравців, складність гри потрібно буде створити простішою. Це потрібно для того, щоб гравець міг відчути жанр, зрозуміти механіки гри та поринути в сюжет.

Обравши таку тактику, проект буде мати широку аудиторію, що додасть успіху проекту.

2.4 Актуальність проекту

Сьогодні ігрова індустрія розвивається дуже швико. Це додає стимулу розробникам створювати нові цікаві проекти. По цій же причині, появляються нові студії розробки ігор. Зрозуміло, що це приносить великий дохід. Через деякий час почало активно розвиватись таке русло, як інді-розробка. Це призвело до багатьох цікавих проектів, які створили незалежні розробники.

З великою впевненістю можна сказати, що даний проект буде мати великий інтерес серед цільової аудиторії завдяки обраному жанру гри. Ігри жанру

платформер мають низький рівень входження та доступні для великого кола гравців завдяки своїй простоті та ігровому процесу. На даний момент, сегмент ігор перенасичений великими проектами, які вимагають потужних пристроїв для гри та глибокого занурення у механіки гри. Також проходження гри займає багато часу, в свою чергу гра з простими механіками, графікою, зрозумілим процесом може більше зацікавити людей, які не мають потужних пристроїв та багато вільного часу.

2.5 Мета проекту

Головна мета проекту – це аналіз засобів розробки ігор, щоб надалі розробити гру у жанрі Платформер. Створення гри з основними механіками жанру. Познайомити потенційних гравців з таким цікавим жанром. Отримання цінного досвіду в різних областях, та оволодіти навичками роботи з рушієм, щоб надалі набагато швидше працювати з іншими проектами.

2.6 Функціонал проекту

Даний проект – це комп’ютерна гра жанру платформер у 2D-стилі, мета якої надати гравцю можливість розважитись, відпочити та гарно провести вільний час.

Для цього, гра повинна відповідати наступним вимогам:

- Повинна мати простий та зрозумілий ігровий процес та механіки;
- Мати просте управління;
- Мати можливість бігати, стрибати, стріляти по ворогам для ліквідації.
- Появлення ворогів у відведених для цього місцях, та при зустрічі з гравцем вони повинні атакувати його;
- Повинні бути предмети, які можна зібрати;
- Після проходження рівня або коли гравець програє, має з’являтись меню, в якому можна перейти до головного меню або пройти рівень заново.

Ця функціональна частина є основною, тому повинна бути розроблена у першу чергу.

2.7 Моделювання об’єктів проектування

2.7.1 Діаграма прецедентів

Uml Use Case diagram (діаграма прецедентів) – діаграма, яка графічно показує взаємодію користувача з програмою. Вона показує різні типи використання, які має система.

Дана діаграма створюється за допомогою об'єктів таких як: Actor, Use case, Package.

Actor - представляє роль користувача, який взаємодіє з системою, яку ви моделюєте. Користувачем може бути людина, організація, машина або інша зовнішня система. (Рис. 2.8)

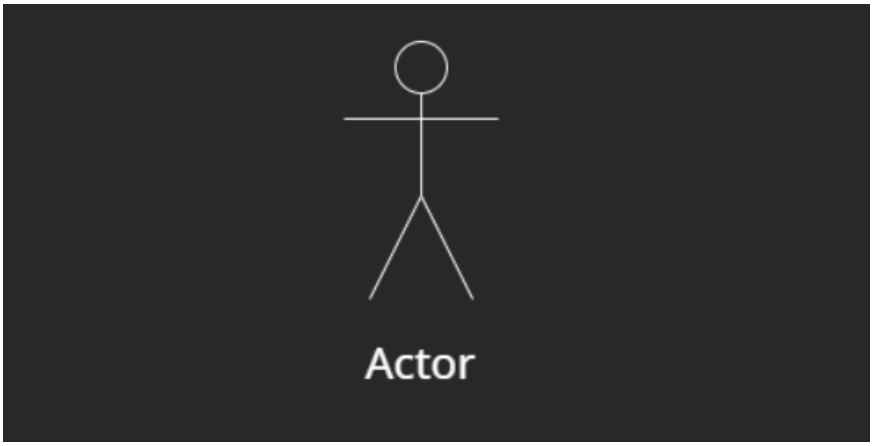


Рис. 2.8 Об'єкт Actor

Use case - цей тип діаграми UML має бути оглядом високого рівня взаємовідносин між акторами та системами, тому вона може бути чудовим інструментом для пояснення вашої системи нетехнічній аудиторії. (Рис. 2.9)

Рис. 2.9 Об'єкт use case

Package - використовуються, щоб показати залежності між різними пакетами в системі. Пакет, зображений у вигляді папки файлів, організовує елементи моделі, такі як варіанти використання або класи, у групи. (Рис. 2.10)

Рис. 2.10 Об'єкт Package

Приклад діаграми прецедентів (Рис. 2.11)

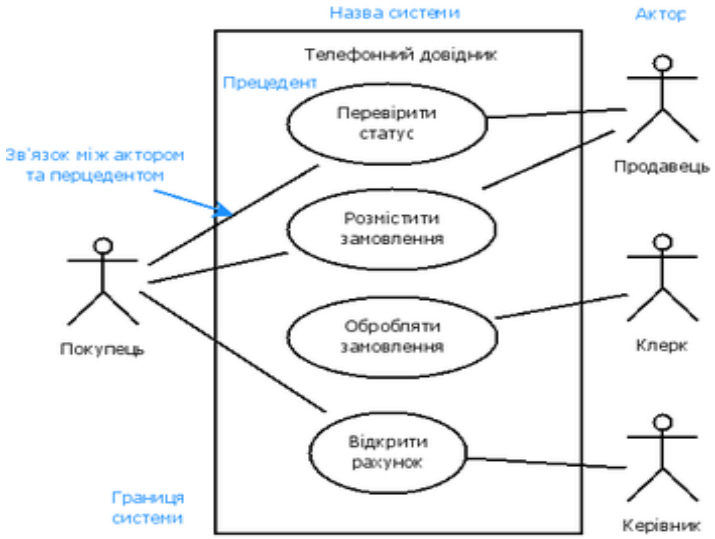


Рис. 2.11 Приклад діаграми прецедентів

2.7.2 Діаграма прецедентів для гри

Щоб розуміти логіку гри, створимо власну діаграму. На (Рис. 2.12) наглядно видно, що саме буде робити гравець.

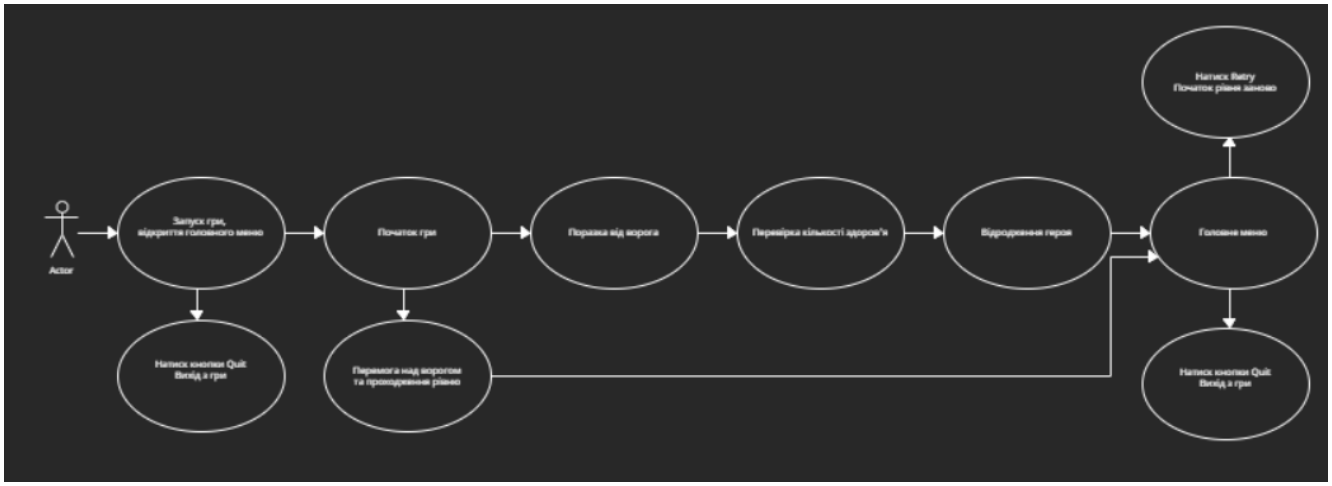


Рис. 2.12 Діаграма прецедентів

2.7.3 Діаграма діяльності

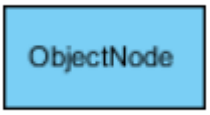
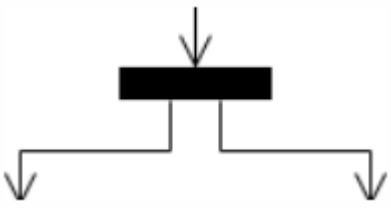
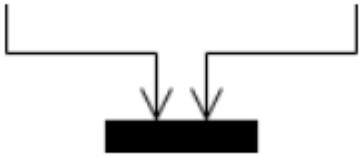
Діаграма діяльності є ще однією важливою діаграмою поведінки в діаграмі UML для опису динамічних аспектів системи. Діаграма діяльності, по суті, є

розширеною версією блок-схеми, яка моделює перехід від однієї діяльності до іншої.

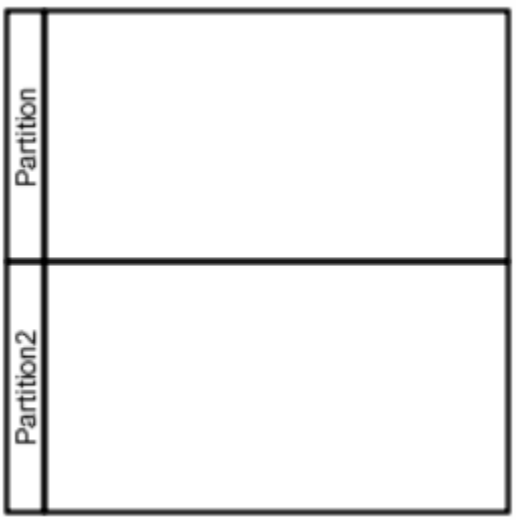
Таблиця 2 – Опис об’єктів діаграми діяльності

Activity Використовується для представлення набору дій	
Action Завдання, яке потрібно виконати	
Control Flow Показує послідовність виконання	
Object Flow Показує перехід об’єкта від однієї діяльності (або дії) до іншої діяльності (або дії).	
Initial Node Показує початок набору дій або діяльності	
Activity Final Node Зупиняє всі потоки керування та потоки об’єктів у дії	

Продовження таблиці 2 – Опис об’єктів діаграми діяльності

Object Node Представляє об’єкт, який пов’язаний з набором потоків об’єктів	
Decision Node Представляє умову тестування, щоб переконатися, що контрольний потік або потік об’єкта йде тільки по одному шляху	
Merge Node Об’єднує різні шляхи прийняття рішень, створені за допомогою вузла прийняття рішень.	
Fork Node Розділяє поведінку на набір паралельних або одночасних потоків дій (або дій)	
Join Node Об’єднує набір паралельних або одночасних потоків діяльності (або дій).	

Продовження Таблиця 2 – Опис об’єктів діаграми діяльності

Swimlane and Partition Спосіб групування дій, які виконує один і той самий суб'єкт, на діаграмі діяльності або групування дій в одному потокі	

Приклад діаграми діяльності. (Рис. 2.13)

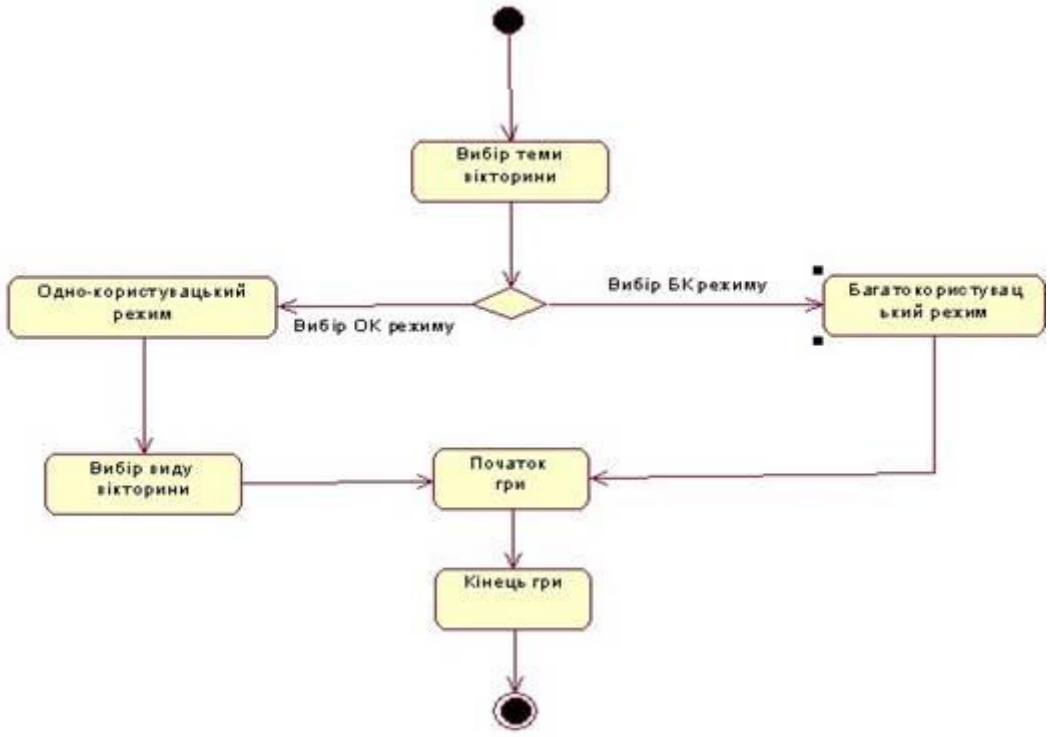


Рис. 2.13 – Приклад діаграми діяльності

2.7.4 Діаграма діяльності для гри

Створюємо власну діаграму діяльності для розуміння логіки гри. (Рис. 2.14)

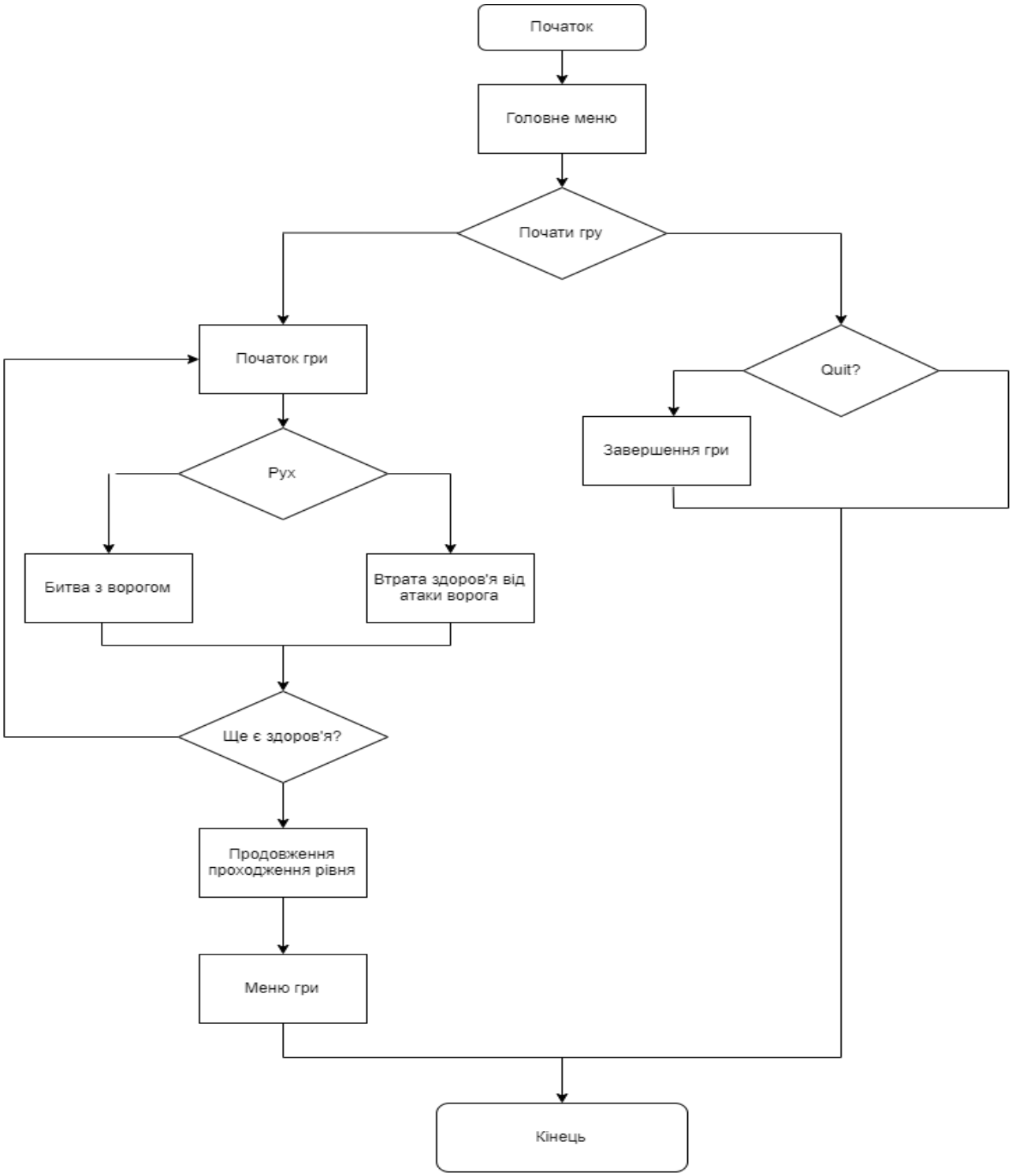


Рис. 2.14 – Діаграма діяльності гри

Висновок до розділу 2

Провівши детальний аналіз рушіїв для розробки комп’ютерних ігор, їх відмінності, призначення, доступ до навчальних матеріалів, можливість розробляти на безкоштовній основі стало зрозуміло, що для нашої 2D гри відмінно підійде рушій Unity. Обравши його, провели аналіз середовища Microsoft Visual Studio, яке інтегрується з Unity та у якому проходить все написання програмного коду проекта.

Слідуючим етапом було визначення алгоритму розробки. Описали етапи створення будь-якої гри, від сценарію до релізу та подальшої підтримки проекту. Також обрали цільову аудиторію, яка допомогла нам визначитись, як саме будемо розробляти гру. Обрали мету проекту та розібрались, чому дана гра буде актуальною.

Також визначили основний функціонал гри, який повинен бути у завершеній версії.

Вкінці другого розділу ми розглянули діаграму прецедентів та діаграму діяльності, розібрались із позначеннями. Також створили дані діаграми для нашого проекту.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Графічне оформлення

Основою будь-якої гри являється її візуальна складова. Основні поняття у геймдизайні, це спрайт та тайл. Детальніше про них:

Спрайт (з англ. Sprite) – 2D графічні об’єкти для персонажів, пострілів та інших різних елементів гри.

Тайл (з англ. Tile) – Компонент Tilemap — це система для створення 2D-рівнів. Це фрагмент текстури, яка повторюється для створення загальної картини.

Для того щоб реалізувати графічну складову нашої гри, оберемо готові 2D спрайти в магазині Unity Asset Store. У ньому є спрайти різних жанрів та типів ігор, що з легкістю дає нам змогу знайти те, що підходить нашій грі. Працювати із спрайтами будемо за допомогою внутрішнього інструменту Sprite Editor (Рис. 3.1)

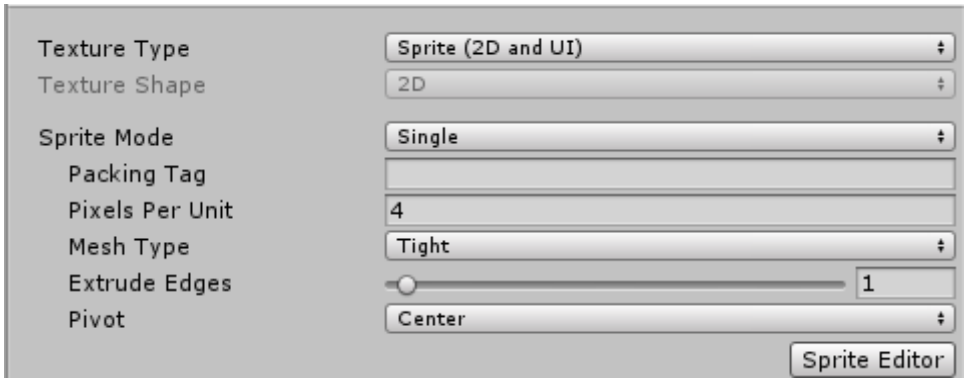


Рис 3.1 Вікно функції Sprite Editor.

Spride Editor розділяє спрайт на декілька частин для створення анімаці.

3.1.1 Спрайти ігрового героя

Головний персонаж нашої гри, буде дракон. У нього є свої спрайти для анімації, а саме: простій, біг, присідання, стрибок, атака ближнього та дальнього бою, сильна атака. (Рис. 3.2)

Рис 3.2 Головний персонаж.

3.1.2 Спрайти інтерфейсу

Далі обрав спрайти фону та візуальних елементів гри. Фонове зображення сцени буде у вигляді неба з хмарами. (Рис. 3.3)

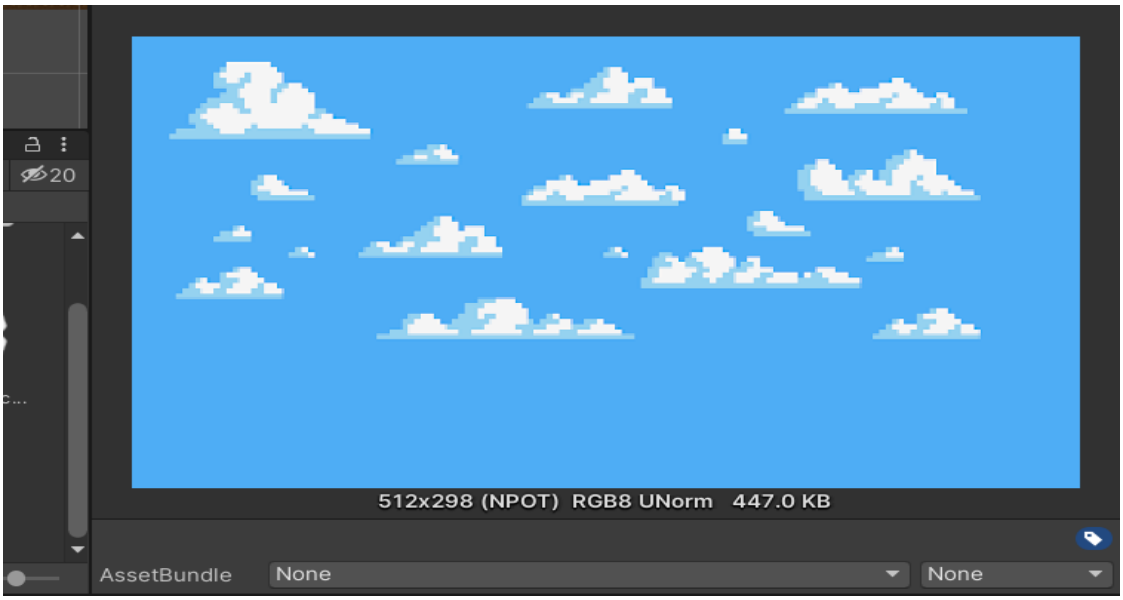


Рис 3.3 Фонове зображення.

На рахунок візуальних елементів на сцені, було обрано пак спрайтів, для потрібної атмосфери давнени та селища. (Рис. 3.4)

Рис 3.4 Спрайти візуальних елементів.

Так як наша гра у жанрі платформер, то для ігрового процесу потрібні платформи, де персонаж зможе переміщатись.

Було прийнято рішення обрати спрайти із паку, що описували вище. Він чудово підходить під стиль та атмосферу гри. (Рис. 3.5)

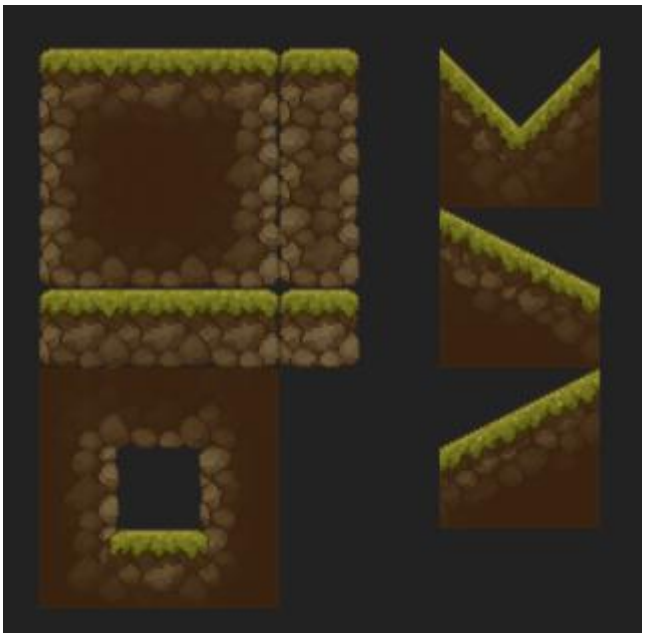


Рис 3.5 Спрайти платформ.

3.1.3 Спрайт ворогів

Також обрали спрайти ворогів, які будуть атакувати нашого персонажа. Один із перших ворогів на нашому шляху, буде жаба. (Рис. 3.6)

Рис 3.6 Спрайт ворога - жаба.

Інший вид ворога, це лучник, який буде атакувати на відстані. Рис (3.7)

Рис 3.7 Спрайт ворога - лучник.

3.1.4 Спрайт пасток

Окрім ворогів, на сцені будуть також пастки, які будуть перешкоджати проходити рівень гравцеві. У цьому нам допоможе пак спрайтів. (Рис. 3.8)

Рис 3.8 Спрайт пасток.

3.1.5 Спрайт зілля

Останнім, що хотілось би виділити це одну з механік гри, а саме зілля. Для них також підібрані спрайти (Рис 3.9) На даний момент їх 2 типи

- Heal – зілля, піднявши яке, поповнить здоров’я головного персонажа
- Mana – зілля, піднявши яке, поповнить запас сил для дальньої атаки
-

Рис 3.9 Спрайт зілля.

3.2 Проектування гри

Так як ми розібрались із візуальною складовою гри, переходимо до самого проектування елементів.

Перш ніж розпочати, ми використаємо функцію Tags & Layers, яка призначає для кожного елементу свій шар. Це дасть нам змогу регулювати відображення елементів поверх інших, щоб уникнути візуальних проблем.

3.2.1 Фізичні властивості об’єктів

Розпочнемо із ігрових об’єктів та їх фізичних властивостей. В першу чергу візьмемо платформу, по якій буде бігати персонаж, та додамо на неї колайдер. Завдяки ньому, програма буде рахувати платформу за фізичний об’єкт і гравець не буде безкінечно падати.

Беремо платформу, на неї додаємо із функціоналу Unity, Box Collider 2D. Саме він надасть фізичну форму. Далі такі ж самі дії робимо і на персонажі, але з деякими відмінностями. Додаємо йому Capsule Collider 2D, вн працює так само, але має форму овалу, а не квадрату. Також потрібен головний компонент для персонажу, це Rigidbody 2D. Це дасть рушію змогу працювати із об’єктом, як з фізичним. Також завдяки коду, можна добавити різні властивості. Це і буде початком програмної частини.

3.2.2 Рух об’єктів

Далі потрібно створити скрипти руху персонажу. Для початку задамо змінні:

```
private Rigidbody2D _playerRB;
```

Змінна _playerRB отримує значення Rigidbody2D. Це як основа об’єкту для виконання фізичних дій.

За допомогою змінних _speed та _crouchSpeedReduce регулюються розробником швидкість бігу та швидкість ходьби присівши.

```
[Header("Horizontal movement")]
[SerializeField] private float _speed;
[Range(0, 1)]
[SerializeField] private float _crouchSpeedReduce;
```

На початку, наш персонаж може стрибати безкінечно угору, тому слід це виправити. Для цього створемо пустий об’єкт _grounded.

```
_grounded = Physics2D.OverlapCircle(_groundCheck.position, _radius, _whatIsGround);
if (_playerAnimator.GetBool("Hurt") && _grounded && Time.time - _lastHurtTime > 0.5f)
EndHurt();
```

CheckGround перевіряє, чи є якийсь інший колайдер поряд у заданому радіусі біля об’єкту _groundCheck.

Після цього потрібно зробити, щоб персонаж міг рухатись. Для цього пишемо метод, де змінну speed множимо на move та speedModificator, щоб регулювати швидкість об’єкту.

```
_playerRB.velocity = new Vector2(_speed * move * speedModificator, _playerRB.velocity.y);
```

При цьому об’єкт буде дивитись лише в одну сторону, тому створюємо дану функцію створивши раніше змінну _faceRight:

```
if (move > 0 && !_faceRight)
        {
            Flip();
        }
        else if (move < 0 && _faceRight)
        {
            Flip();
        }
```

Якщо метод move має значення більше 0 (персонаж рухається вправо), то метод transformRotate (відповідає за поворот об’єкту) надає йому значення по осі y 0, отже персонаж буде дивитись праворуч. Так само, якщо метод move буде менше 0, методом transformRotate об’єкт матиме значення 180 по осі y і персонаж буде дивитись ліворуч.

Після руху персонажа перейдемо до стрибків.

```
if (jump)
        {
            if (_grounded)
            {
                _playerRB.velocity = new Vector2(_playerRB.velocity.x, _jumpForce);
                _canDoubleJump = true;
            }
```

```
        else if (_canDoubleJump)
        {
            _playerRB.velocity = new Vector2(_playerRB.velocity.x, _jumpForce);
            _canDoubleJump = false;
        }
    }
```

Робимо перевірку, чи натиснена клавіша стрибку. Також перевіряємо через змінну _grounded, чи знаходиться об’єкт на поверхні, якщо ні то вона обмежить стрибки.

3.2.3 Анімації для героя

Для того, щоб створити анімацію, використовуємо в Unity вбудований інструмент Animation та Animator. (Рис 3.10)

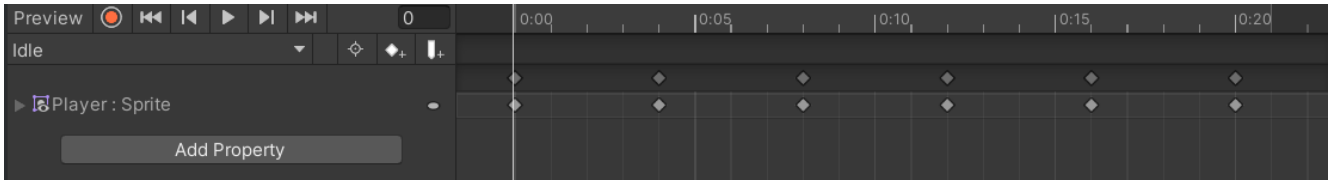


Рис 3.10 Вікно Animation.

Далі потрібно встановити зв’язки створених анімацій між собою, щоб отримати перехід від однієї до іншої. (Рис. 3.11)

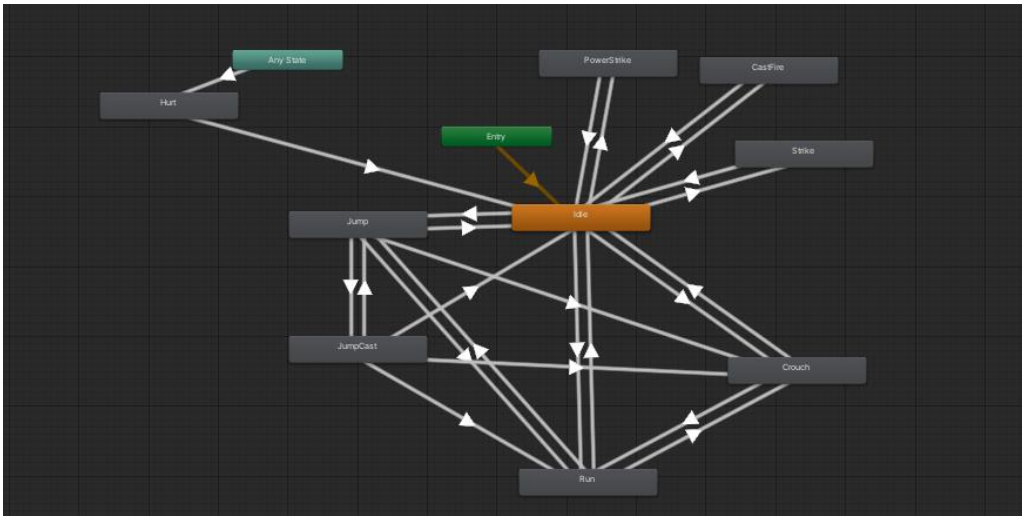


Рис 3.11 Вікно зв’язків анімацій.

3.2.4 Атака героя

Також потрібно звернути увагу на спосіб боротьби. Під стилістику гри підходить в ближньому бої удар рогом, а для дальнього бою вистріл вогнем.

Отже, задаємо спочатку змінні:

```
[Header("Casting")]
    [SerializeField] private GameObject _fireBall;
    [SerializeField] private Transform _firePoint;
    [SerializeField] private float _fireBallSpeed;
    [SerializeField] private int _castCost;
    private bool _isCasting;

    [Header("Strike")]
    [SerializeField] private Transform _strikePoint;
    [SerializeField] private int _damage;
[SerializeField] private float _strikeRange;
[SerializeField] private LayerMask _enemies;
private bool _isStriking;

[Header("PowerStrike")]
[SerializeField] private float _chargeTime;
[SerializeField] private float _powerStrikeSpeed;
[SerializeField] private Collider2D _strikeCollider;
[SerializeField] private int _powerStrikeDamage;
[SerializeField] private int _powerStrikeCost;
[SerializeField] private float _pushForce;
```

Змінна _fireBallSpeed у нас 0, вона налаштовує швидкість пострілу вогнем, змінна _castCost відповідає за кількість потрачених сил на постріл, _damage вказує кількість урону, який буде нанесено ворогу. LayerMask _enemies створює маску, по ній програма буде розуміти, до якого об’єкту застосовувати постріл.

Схожі змінні присутні у ближніх атаках, таких як Strike та PowerStrike.

Далі йде функція атаки. Щоб анімація не повторювалась, ми робипо перевірку, якщо ми в даний момент зробили удар (_isStriking), то визиваємо return, щоб не заходити в метод знову. Також робимо перевірку, щоб при сильній атаці, персонаж не міг бігати.

```
        if (_isStriking || _playerRB.velocity != Vector2.zero)
            return;

        _canMove = false;
        if (holdTime >= _chargeTime)
        {
            if (!_playerController.ChangeMP(-_powerStrikeCost))
                return;
            _playerAnimator.SetBool("PowerStrike", true);

        }
        else
        {
```

```
            _playerAnimator.SetBool("Strike", true);
        }

        _isStriking = true;
```

Також розглянемо постріл вогнем, спершу ми створюємо об’єкт у заданій позиції, із поворотом у потрібно сторону. Далі запускаємо наш об’єкт із потрібною нам швидкістю звернувшись до компоненту Rigidbody2D. Щоб не завантажувати пам’ять гри, нам потрібно через деякий час, щоб зникали постріли, так як вони залишаються на сцені. Ми беремо Destroy, у ньому вказуємо потрібний об’єкт і через скільки він зникне.

```
        GameObject fireBall = Instantiate(_fireBall, _firePoint.position,
Quaternion.identity);
        fireBall.GetComponent<Rigidbody2D>().velocity = transform.right * _fireBallSpeed;
        fireBall.GetComponent<SpriteRenderer>().flipX = !_faceRight;
        Destroy(fireBall, 5f);
```

3.2.5 Смерть та відродження головного персонажа

Перш ніж створювати ворогів, потрібно налаштувати при яких умовах та які наслідки будуть після смерті головного персонажа, а саме здоров’я персонажа та відродження після смерті.

```
[SerializeField] private int _maxHP;
private int _currentHP;
```

Задаємо змінну для здоровья _maxHP, кількість здоров’я буде задаватись в інспекторі Unity, та актуальне здоров’я _currentHP.

```
if (!_canBeDamaged)
            return;

        _currentHP -= damage;
        if (_currentHP <= 0)
        {
            OnDeath();
        }

        switch(type)
        {
            case DamageType.PowerStrike:
                _playerMovement.GetHurt(enemy.position);
                break;
        }
        _hpSlider.value = _currentHP;
```

Далі створюємо умову, якщо змінна _currentHP буде менше або дорівнювати 0 , то програма вирішить, що персонаж загинув.

Після умови для смерті, створюємо відродження. Для цього створюємо стартову позицію.

```
Vector2 _startPosition;
```

Цю позицію вказуємо у методі смерті. Тобто, якщо наш персонаж загинув, він появиться у стартовій позиції.

```
public void OnDeath()
    {
        _serviceManager.Restart();
    }
```

3.2.6 Штучний інтелект ворогів

Створюємо клас EnemyControllerBase, там будуть схожі функції, як і у персонажа, але відповідати вони будуть за ворогів.

```
[Header("HP")]
    [SerializeField] protected int _maxHp;
    [SerializeField] protected Slider _hpSlider;
    protected  int _currentHp;
```

Отримання урону схоже, як у персонажа.

```
if (_currentState == EnemyState.Death)
            return;

        _currentHp -= damage;
        _hpSlider.value = _currentHp <= 0 ? 0:_currentHp ;
        if (_currentHp <= 0)
        {
            _currentHp = 0;
            ChangeState(EnemyState.Death);
        }
```

Далі метод, який перевіряє, чи головний персонаж у зоні, щоб атакувати його. Якщо головний персонаж попадає в зону, то ворог розвертається до нього та атакує, якщо виходить із зони, він перестає атакувати.

```
if(_player == null || _attacking)
        {
            return;
        }

        if (Vector2.Distance(transform.position, _player.transform.position) < _angerRange)
        {
            _isAngry = true;
            TurnToPlayer();
            ChangeState(EnemyState.Shoot);
        }
        else
            _isAngry = false;
```

3.2.7 Графічний інтерфейс для героя та ворога

Слідуючим етапом буде графічний інтерфейс. Для початку створимо відображення здоров’я та запасу сил для головного персонажа та ворогів. (Рис. 3.12)

Рис 3.12 Смуга життя ворога.

Для цього, створюємо Canvas, (полотно, яке буде показувати нам слайдер поверх всієї сцени) у ньому створюємо Slider. Далі створюємо змінні у EnemyControllerBase та призначаємо нашому слайдеру максимальне здоров’я.

```
[SerializeField] protected Slider _hpSlider;


_hpSlider.maxValue = _maxHp;
_hpSlider.value = _maxHp;
```

Далі призначаємо слайдеру актуальне здоров’я, після атак у методі отримання урону.

```
_hpSlider.value = _currentHp <= 0 ? 0:_currentHp ;
```

Схожим методом робимо для головного персонажа, але ще додаємо смугу із запасом сил. (Рис. 3.13)

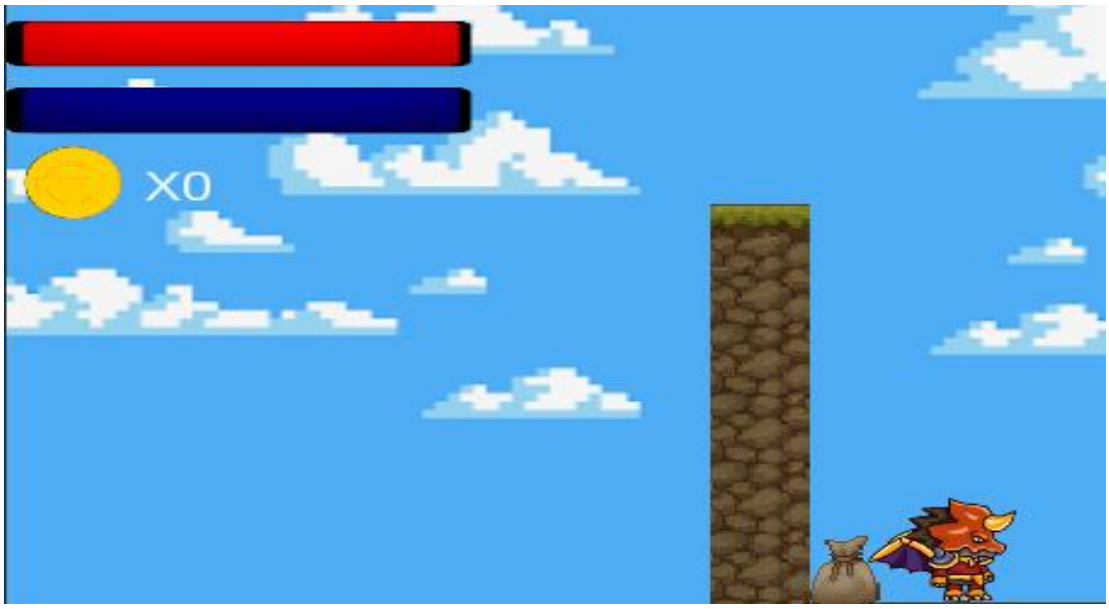


Рис 3.13 Смуга життя та запасу сил персонажу.

3.2.8 Взаємодія між об’єктами

Далі перейдемо до взаємодії ворога та персонажу, а саме нанесення урону один одному. Розпочнемо з персонажу.

Для цього вводимо змінну:

```
[SerializeField] private int _damage;
```

Вона відповідає за кількість нанесення урону.

Далі присвоюємо за допомогою компонента info.GetComponent об’єкту EnemyControllerBase можливість отримувати шкоду від атак.

```
EnemyControllerBase enemy = info.GetComponent<EnemyControllerBase>();
       if (enemy != null)
              enemy.TakeDamage(_damage, DamageType.Projectile);
               Destroy(gameObject);
```

Переходимо до ворога. По аналогії робимо, як з персонажем.

```
if ((DateTime.Now - _lastEncounter).TotalSeconds < _timeDelay/2)
            return;

        _lastEncounter = DateTime.Now;
        _player = info.GetComponent<Player_controller>();
        if (_player != null)
                  _player.TakeTamage(_damage);
```

3.2.9 Меню гри

Для головного меню нам потрібна нова сцена. Створюємо наступні кнопки: (Рис 3.14)

- Play – відповідає за початок гри
- Choose lvl – відповідає за вибір рівня
- Settings – відповідає за налаштування гри
- Reset – відповідає за проходження рівня заново
- Quit – відповідає за вихід з гри

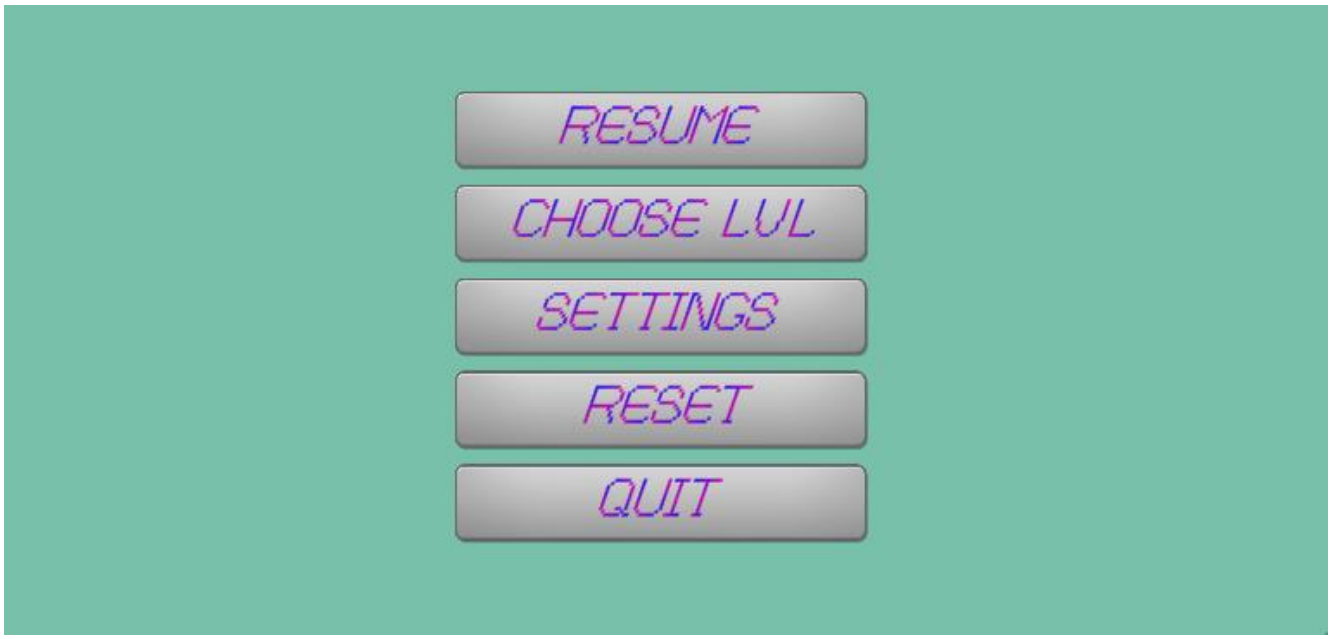


Рис 3.14 Кнопки головного меню.

Створюємо скрипт для головного меню, добавляємо у нього наступні функції:

```
private void Start()
    {
        Time.timeScale = 1;

        if(SceneManager.GetActiveScene().buildIndex!=0)
        {
            PlayerPrefs.SetInt(GamePrefs.LastPlayedLvl.ToString(),
SceneManager.GetActiveScene().buildIndex);
```

```
            PlayerPrefs.SetInt(GamePrefs.LvlPlayed.ToString() +
SceneManager.GetActiveScene().buildIndex, 1);
        }
    }
    public void Restart()
    {
        ChangeLvl(SceneManager.GetActiveScene().buildIndex);
    }

    public void EndLevel()
    {
        ChangeLvl(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void ChangeLvl(int  lvl)
    {
        SceneManager.LoadScene(lvl);
    }

    public void Quit()
    {
        Application.Quit();
        Debug.Log("Quit");
    }

    public void ResetProgres()
    {
        PlayerPrefs.DeleteAll();
    }
```

Start відповідає за початок, він переносить нас на сцену із самою грою, Restart програє рівень з початку, EndLevel відмічає, що гравець пройшов рівень і змінює сцену на іншу з другим рівнем. ChangeLvl відкриває меню вибору рівня, Quit відповідає за вихід з гри

ВИСНОВКИ

У першому розділі ми провели аналіз ігор у жанрі Платформер та оглянули переваги та недоліки, також розглянули жанри ігор.

У другому розділі розібрались, що являє собою жанр Платформер, зробили аналіз рушіїв для розробки, виявили переваги та недоліки та обрали рушій для нашого проекту. Далі визначились з алгоритмом розробки гри, зрозуміли яка наша потенційна аудиторія, описали мету гри, її актуальність та визначились із мінімальним функціоналом. У кінці розібрали та створили діаграми прецедентів та діяльності.

Далі ми дійшли до реалізації гри у третьому розділі. Розібрались стосовно графічного оформлення. Далі описували компоненти гри, які були застосовані у грі. Також описали основні методи та класи гри.

Виконавши усі етапи, в результаті маємо гру у жанрі Платформер, яка є зрозумілою по механікам та простою у інтерфейсі. Надалі дану гру можна розвивати, наповнюючи новими механіками, оновлюючи графіку, тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Комп'ютерна гра [Електронний ресурс] – Режим доступу до ресурсу: https://esu.com.ua/search_articles.php?id=4393

2. Класифікація ігор за жанрами [Електронний ресурс] – Режим доступу до ресурсу: http://zps19.at.ua/publ/mediateksti_vid_ekspertiv/klasifikacija_i_zhanri_komp_juternikh_i_videoigor/15-1-0-226.

3. Гра Mark of the Ninja [Електронний ресурс] – Режим доступу до ресурсу: https://www.klei.com/games/mark-ninja.

4. Гра Cuphead [Електронний ресурс] – Режим доступу до ресурсу: https://www.xbox.com/ru-RU/games/cuphead.

5. Гра Starbound [Електронний ресурс] – Режим доступу до ресурсу: https://playstarbound.com/.

6. Гра Dead Cells [Електронний ресурс] – Режим доступу до ресурсу: https://deadcells.gamepedia.com/Dead_Cells_Wiki.

7. Найкращі ігрові рушії [Електронний ресурс] – Режим доступу до ресурсу: https://www.gamedesigning.org/career/video-game-engines.

8. Unreal Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/.

9. Cry Engine [Електронний ресурс] – Режим доступу до ресурсу: https://www.cryengine.com/.

10. Unity Game Engine [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/ru.

11. Unity Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/index.html.

12. David B. Hands-On Game Development Patterns with Unity / Baron David., 2019. – 116 с.

13. Створення 2D на Unity [Електронний ресурс] – Режим доступу до ресурсу: https://gamedevacademy.org/how-to-build-a-complete-2d-platformer-in-unity/.

14. Пояснення про спрайти [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/ScriptReference/Sprite.html.
15. Пояснення про тайли [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/Tilemap-ScriptableTiles-Tile.html
16. Hocking J. Unity in Action. Multiplatform game development in C# with Unity 5 / Joseph Hocking., 2015. – 352 с.
17. Smith G. Basic Math for Game Development with Unity 3D / G. Smith, K. Sung., 2018. – 279 с.
18. Thorn A. Mastering Unity Scripting / Alan Thorn., 2015. – 380 с.
19. Ferrone H. Mastering Unity Scripting / Harrison Ferrone., 2016. – 379 с.
20. Halpern J. Developing 2D Games with Unity: Independent Game Programming with C# / Jared Halpern., 2018. – 408 с.

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка гри жанру платформер на мові програмування C# за допомогою ігрового двигуна Unity

Виконав студент 4 курсу
групи ПД-44
Рудий Андрій Сергійович
Керівник роботи
Дібрівний Олесь Андрійович

Київ – 2022

АНАЛОГИ

Назва	Переваги	Недоліки
Dead Cells	Графіка	Відсутня можливість позбавитись зброї, яка відкрита за допомогою схем
	Бойова система	Відсутність інноваційних систем
Cuphead	Візуальний стиль мультфільмів Disney	Важкість проходження залежить від завченності гравця карти та ворогів
	Великий різновид контенту	
Mark of the Ninja	Графіка	Звичаний сюжет
	Варіативність проходження	
Starbound	Процедурна генерація карти	Однотипні місії
	Мультиплеєр	Незначні баги

2

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - аналіз засобів розробки ігор, щоб надалі розробити гру у жанрі Платформер. Створення гри з основними механіками жанру познайомити потенційних гравців з таким цікавим жанром. Отримання цінного досвіду в різних областях, та оволодіти навичками роботи з рушієм, щоб надалі набагато швидше працювати з іншими проектами.
- **Об'єкт дослідження –** реалізація гри у жанрі Платформер
- **Предмет дослідження -** гра платформер за допомогою рушія Unity на мові програмування C#

3

ТЕХНІЧНІ ЗАВДАННЯ

- *Зробити аналіз комп'ютерних ігор*
- *Зробити аналіз рушіїв для розробки*
- *Описати алгоритм розробки*
- *Зробити аналіз потенційної аудиторії*
- *Описати актуальність проекту*
- *Описати мету проекту*
- *Описати функціонал проекту*
- *Моделювання об'єктів проектування*

4

ПРОГРАМНІ ТА ТЕХНІЧНІ ЗАСОБИ РЕАЛІЗАЦІЇ

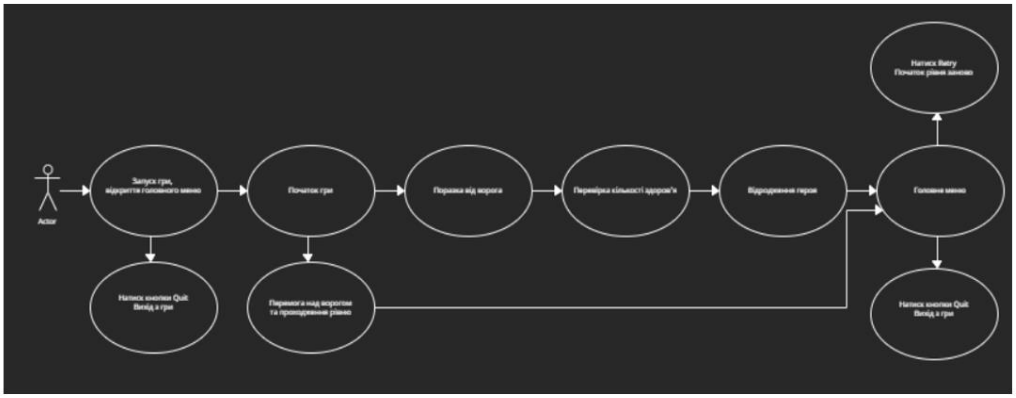
unity

VISUAL STUDIO 2019

C#

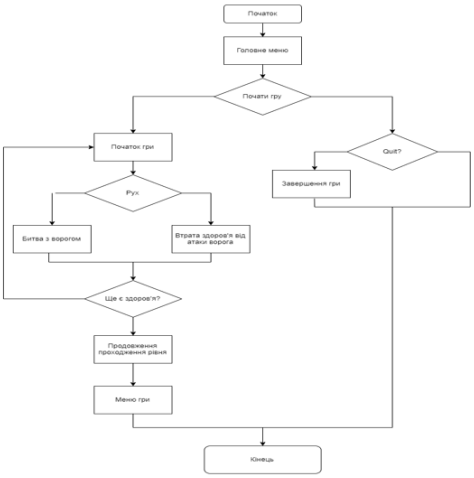
5

ДІАГРАМА ПРЕЦЕДЕНТІВ



6

ДІАГРАМА ДІЯЛЬНОСТІ



7

ВИСНОВКИ

Виконано:

- *Аналіз комп'ютерних ігор*;
- *Аналіз рушіїв*
- *Опис алгоритмів розробки*
- *Аналіз аудиторії*
- *Опис актуальності проекту*
- *Опис мети проекту*
- *Опис мінімального функціоналу проекту*
- *Створено 2 діаграми*

8

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- Рудий А.С., ОСОБЛИВОСТІ РОЗРОБКИ ІГОР НА UNITY, XIV НАУКОВО-ТЕХНІЧНІЙ КОНФЕРЕНЦІЇ «СУЧАСНІ ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ»
- Рудий А.С., ОСНОВНІ ПРИНЦИПИ ГЕЙМДИЗАЙНУ, Науково-технічна конференція «Застосування програмного забезпечення в ІКТ», К.: ДУТ, 2022»

9

ДЯКУЮ ЗА УВАГУ!

12