Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти − бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

Тема роботи:
«Розробка 2D-платформенної гри в жанрі runner на базі програмного середовища Unity»

Виконав студент гр. КН-20 _____ Дьомін М. Д.

Керівник _____ Маринич І. А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *27* » *березня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-20 *Дьоміну Миколі Дмитровичу*

1. Тема кваліфікаційної роботи: *«Розробка 2D-платформенної гри в жанрі runner на базі програмного середовища Unity»*

затверджено наказом по університету № *235с* від *27.03.2024 р.*

2. Термін здачі кваліфікаційної роботи: *05.06.2024 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 68с., додатки, презентація у Microsoft PowerPoint (9 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. Маринич І. А.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	01.04.24
2	*Розділ 1*	03.04.24
3	*Розділ 2*	05.05.24
4	*Висновки*	10.05.24
5	*Оформлення кваліфікаційної роботи*	15.05.24
6	*Підготовка презентації та графічного матеріалу*	25.05.24
7	*Підготовка доповіді до захисту*	05.06.24

6. Дата видачі завдання: *29.01.2024р.*

Керівник _____ /Маринич І. А./

7. Запевнення: *Я, Дьомін Микола Дмитрович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Дьомін М. Д./

АНОТАЦІЯ

Дьомін М.Д. Розробка 2D-платформенної гри в жанрі runner на базі програмного середовища Unity.

Кваліфікаційна робота на здобуття ступеня вищої освіти – бакалавр, за спеціальністю 122 – Комп'ютерні науки. – Криворізький національний університет, Кривий Ріг, 2024.

Робота Складається зі вступу, двох розділів, висновків, переліку використаної літератури з 26 позицій та 0 додатків. Загальний обсяг роботи становить 68 сторінок, з яких основний зміст роботи викладено на 61 сторінках, включає 1 таблицю і 73 рисунки.

Був розглянутий стан ігрової індустрії. Були проаналізовані жанри відеоігор, виявлені причини популярності жанру раннер та проаналізовані відомі проекти у жанрі раннер. Було досліджено основні програмні середовища для розробки відеоігор, такі як Unity, Unreal Engine та Godot Engine. На основі аналізу був вибраний Unity, з огляду на його потужні інструменти для розробки 2D-платформенних відеоігор, ліцензію, широку спільноту розробників та доступність численних ресурсів і навчальних матеріалів. На основі теоретичних досліджень було розроблено та реалізовано 2D-платформенну гру в жанрі runner на базі програмного середовища Unity. Проєкт включає декілька режимів гри, такі як проходження рівнів та безкінечний біг. Були реалізовані основні елементи геймплею, такі як подолання перешкод та ворогів, збирання винагород у вигляді монет та розблокування нових персонажів. Були додані соціальні функції, такі як досягнення та список лідерів. Відеогра пройшла закрите тестування в Google Play Console, з залученням зовнішніх користувачів. На основі отриманих відгуків було проведено оптимізації ігрових механік та усунено виявлені помилки.

Зміст

ВСТУП

Комп'ютерні ігри стали неодмінною частиною сучасної культури, привертаючи увагу мільярдів гравців по всьому світу[1]. Ця індустрія постійно розвивається і на сьогоднішній день є однією з найбільш прибуткових та впливових галузей розваг. Її популярність пояснюється не лише технологічними досягненнями, а й здатністю занурити гравців у неймовірні світи, де вони можуть відчути себе героями власних пригод. Цей вид розваги став не просто відпочинком, а повноцінним культурним явищем, що формує та впливає на смаки та інтереси багатьох людей.

Ігрова індустрія є великою та динамічною системою, що охоплює створення, розробку, публікацію та розповсюдження відеоігор на різних платформах. Однією з визначальних характеристик ігрової індустрії є її різноманітність. Ігри бувають різних жанрів, включаючи екшн, пригоди, рольові ігри, стратегії, спортивні, симулятори та багато інших. Це розмаїття задовольняє широкий спектр аудиторій, від казуальних гравців, які шукають швидких розваг, до ентузіастів, які прагнуть повного занурення у світ ігор. Індустрія також зазнає значного впливу технологічних інновацій. Графіка, звуковий дизайн, штучний інтелект, віртуальна та доповнена реальність - це лише деякі зі сфер, де відбувається постійний прогрес. Кожне нове досягнення відкриває перед розробниками ігор нові можливості для створення більш захопливого та реалістичного досвіду для гравців. Популярність ігрової індустрії стрімко зросла за останні роки завдяки таким факторам, як розвиток кіберспорту, стрімінгових платформ, таких як Twitch і YouTube, а також все більша інтеграція ігор у масову культуру. Ігри перестали бути просто формою розваги, а стали соціальною активністю, змагальним видом спорту та засобом художнього самовираження. Крім того, глобальний масштаб ігрової індустрії проявляється у її впливі на економіку, створення робочих місць та культурний обмін. Студії та видавництва, що займаються розробкою ігор, працюють по всьому світу, співпрацюючи з талановитими людьми різного походження та сприяючи зростанню місцевої економіки.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз ігрової індустрії

Ігрова індустрія, також відома як індустрія відеоігор, охоплює широкий спектр діяльності, пов'язаної зі створенням, розробкою, маркетингом та розповсюдженням відеоігор. Це глобальна багатомільярдна індустрія, яка значно розвинулася з моменту свого заснування, глибоко впливаючи на сучасні розваги, технології та культуру.

Ігри швидко перетворилися з нішевого хобі на один з найбільших ринків індустрії розваг. На даний момент в світі налічується 3,09 мільярдів гравців і ця цифра стрімко зростає[1]. Так ще вісім років назад ця цифра налічувала 2,17 мільярдів. (Таблиця 1.1)

Таблиця 1.1 – Кількість гравців в світі.

Рік	Кількість гравців (мільярдів)	Зростання у порівнянні з попереднім роком	Зростання у порівнянні з попереднім роком (%)
2016	2,17	140 мільйонів	6,9%
2017	2,33	160 мільйонів	7,37%
2018	2,49	160 мільйонів	6,87%
2019	2,64	150 мільйонів	6,02%
2020	2,81	170 мільйонів	6,44%
2021	2,96	150 мільйонів	5,34%
2022	3,09	130 мільйонів	4,39%
2023	3,22	130 мільйонів	4,21%
2024	3,32	100 мільйонів	3,11%

Відеоігри пройшли великий шлях з часів понгу та тетрісу. З появою домашніх консолей, таких як Atari 2600 наприкінці 70-х – на початку 80-х років, відеоігри

почали здобувати загальне визнання. Ці ранні ігри були відносно простими, складалися з базової ігрової механіки та обмежених графічних можливостей. Однак вони заклали фундамент того, що згодом стало багатомільярдною індустрією. З розвитком технологій розвивалися і відеоігри. Поява 3D-графіки в середині 1990-х проклала шлях до більш складних і візуально привабливих ігор. Цей зсув дозволив розробникам створювати реалістичні середовища та персонажів, покращуючи загальний ігровий досвід. Такі ігри, як Super Mario 64 та The Legend of Zelda: Ocarina of Time, стали культовими та продемонстрували потенціал інтерактивного оповідання у відеоіграх[2].

Розвиток технологій призвів до можливості створювати все більш розвинуті ігри, що з візуальної сторони, що і з сторони нових механік та збільшення віртуальних світів. Такий стрімкий розвиток стрімко збільшив кількість гравців по всьому світу, що призвело до збільшення ігрових платформ. Зараз в відеоігри можна грати не лише на PC або консолях, а й на телефонах. Сучасні смартфони дають змогу грати в відеоігри з приголомшливою графікою, так на IPhone 15 Pro можна запустити Resident Evill Village і Death Stranding, що є сучасними іграми для консолей та PC, з сучасною графікою. Окрім цього у 2024 році дохід світового ринку комп'ютерних ігор наблизився до 40 мільярдів доларів, тоді як дохід ринку мобільних ігор перевищив 92 мільярди доларів. Варто зазначити, що сегмент мобільних ігрових консолей посів друге місце з показником $51 млрд. За прогнозами, до 2025 року ігровий ринок досягне 211 мільярдів доларів США, причому мобільні ігри вже зараз приносять понад 60% доходу світового ринку відеоігор. Варто зазначити, що кількість ігор, які щорічно випускаються для ПК і консолей, залишається відносно рівною. Незважаючи на те, що на ринку консолей часто з'являються ексклюзивні ігри (ігри консольної якості для андроїду), дохід від консолей посідає друге місце після мобільних пристроїв, значно перевищуючи дохід від комп'ютерів.

Таким чином, якщо говорити про мобільні, консольні та комп'ютерні ігри з точки зору рентабельності ресурсів розробників, то успіху можна досягти в будь-якій сфері за умови наявності належного досвіду та знань індустрії[3].

Такий швидкий розвиток ігрової індустрії призвів до необхідності класифікувати відеоігри. Один з перших способів класифікації відеоігор навів ігровий дизайнер Кріс Кроуфорд у своїй книзі «The Art of Computer Game Design» 1982 року[4]. Він запропонував розділити ігри за подібністю в дизайні, створивши таксономію комп’ютерних ігор.

Так Кріс Кроуфорд пише про такі основні жанри відеоігор:

– Ігри на навички та дії (skill-and-action games) – це один з найбільших і найпопулярніших жанрів відеоігор, характеризується грою в реальному часі, використання джойстиків або педалей замість клавіатури. Основні навички, що вимагаються від гравця, - це зорово-моторна координація та швидка реакція.

– Бойові ігри (combat games) – ігри в цьому жанрі являють собою пряме, жорстоке протистояння гравця та ворогів, керованих комп’ютером.

– Лабіринтові ігри (maze games) – визначальною характеристикою ігор-лабіринтів є лабіринт шляхів, якими гравець повинен рухатися. Яскравим представником жанру є Pac-Man.

– Спортивні ігри (sports games) – ігри, що дають гравцю можливість грати в класичні ігри, за мотивами баскетболу, футболу та інших.

– Ігри на відбивання (paddle games) – ігри, що засновані на Pong. Центральний елемент гри – перехоплення м’яча за допомогою ракетки.

– Гонки (race games) – ігри, що передбачають перегони з використанням різного транспорту

– Стратегічні ігри (strategy games) – ігри, акцент в яких робиться на роздумах, а не на діях.

– Пригоди (adventures) - У цих іграх шукач пригод повинен рухатися через складний світ, накопичуючи інструменти та здобич, достатні для подолання кожної перешкоди, поки нарешті не досягне скарбу або мети.

– D&D-ігри (D&D games) – ігри подібні до рольової гри Dungeons and Dragons, складною некомп'ютерною грою про дослідження, співпрацю та конфлікти у казковому світі

– Варгейми (wargames) – клас стратегічних ігор, що являють собою військовими іграми. Комп’ютерні варгейми копіють настільні варгейми.

– Азартні ігри (games of chance) – ігри, що копіюють фізичні варіанти азартних ігор. Незважаючи на широку доступність, ці ігри не виявилися не дуже популярними, бо вони не використовують сильні сторони комп'ютера, та втрачають переваги оригінальних, фізичних ігор.

– Навчальні та дитячі ігри (educational and children’s games) – розвиваючі ігри, часто направлені для навчання дітей.

– Міжособистісні ігри (Interpersonal Games) – ігри, що зосереджуються на стосунках між окремими людьми або групами людей.

Але вже в 1986 році Ральф Кохен пише, що чітко визначити жанр відеогри неможливо, бо жанр є відкритою категорією, та кожен може змінити жанр додаванням, перевертанням або зміною складових[5].

В наш час кількість жанрів постійно росте, оскільки ігри постійно змінюються і вдосконалюються, відкриваючи нові можливості для гравців і розширюючи горизонти творчості розробників. Нові технології, такі як віртуальна реальність та штучний інтелект, сприяють створенню більш іммерсивних і складних ігрових світів. Розробники, в пошуках унікальних концепцій, створюють нові жанри або поєднують елементи різних жанрів, щоб задовольнити різноманітні смаки гравців і забезпечити позитивний досвід від гри.

1.2 Аналіз існуючих проектів

Раннер, або безкінечний раннер (endless runner) є жанром, що походить від ігор-платформерів, в яких персонаж біжить нескінченну кількість часу, долаючи різноманітні перешкоди. Головна мета гравця – пробігти якомога далі, та досягти високих балів в забігу.

Жанр походить від відеоігр 1970-х років, частіше всього це були гоночні ігри, де гравець постійно рухається вперед, ухиляючись від перешкод. Першою такою грою вважається Taito's Speed Race, що була створена в 1974 році. З появою сенсорних екранів на смартфонах починається нова ера цього жанру, через покращення управління, що може надати сенсорний екран, в порівнянні з кнопками. В 2009 році виходить Doodle Jump (Рис 1.1), що стає однією з перших раннерів для сенсорних смартфонів.

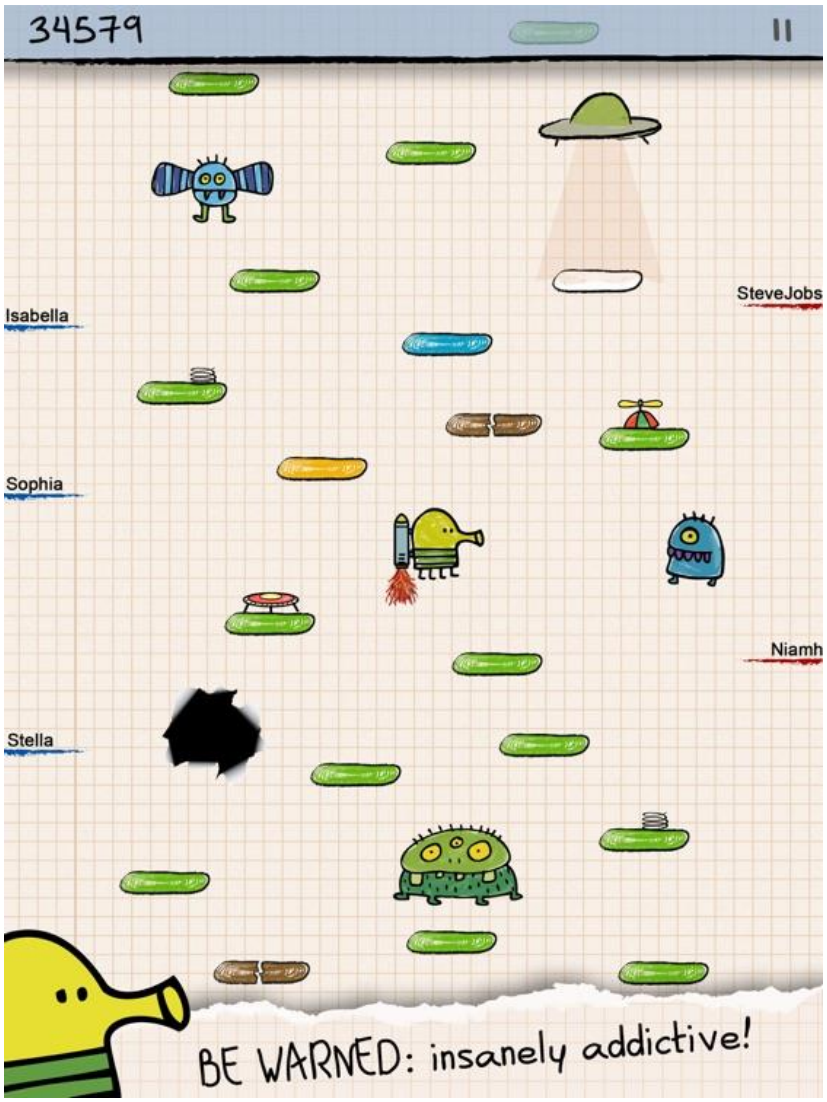

Рисунок 1.1 – Скріншот екрану з гри Doodle Jump.

Після виходу Doodle Jump раннери почали набирати велику популярність. Так можна виділити яскравих представників жанру:

1. Subway Surfers, розроблена SYBO Games.

Раннер в якому герой біжить по міських вулицях і намагається уникати перешкод (Рис 1.2). Мета гри полягає в тому, щоб пробігти якомога більше відстані, уникати зіткнень з перешкодами і збирати різні предмети, такі як монети і бонуси. Subway Surfers стала відомою через свою яскраву графіку, та захоплюючий геймплей, де гравець намається пробігти якомога далі і побити власний рекорд.

З основних функціональних можливостей можна виділити:

– Гравець, свайпами по екрану, керує персонажем.
– Нескінченний режим з рандомною генерацією рівня.
– Генерація бонусів і монет.
– Можливість купувати нових персонажів.

Рисунок 1.2 – Скріншот екрану з гри Subway Surfers.

Гра стала однією з самих популярних мобільних ігор в світі і на даний момент нараховує більше 4 мільярдів завантажень.

2. Jetpack Joyride, розроблена Halfbrick Studios

У грі гравець керує персонажем на ім'я Баррі Стейкфрай, який використовує реактивний ранець для польоту та просування через різноманітні перешкоди. Головна мета гри - пройти якомога більше відстані, збираючи монети і бонуси, уникаючи при цьому різних перешкод і пасток. Баррі може керувати ранцем, піднімаючись вгору, спускаючись вниз або рухаючись вперед, щоб уникати зіткнень (Рис 1.3).

Рисунок 1.3 – Скріншот екрану з гри Jetpack Joyride.

Одним із основних елементів геймплею є використання різних видів ранців. Наприклад, є ранець з великими зубами, який руйнує перешкоди, ранець-катамаран, який збільшує ширину персонажа, або ранець-пороховий заряд, що дає додаткову швидкість. Крім того, гравець може знаходити та використовувати різноманітні бонуси, такі як прискорення, магніт для монет, щити для захисту від перешкод тощо. Ці бонуси допомагають гравцеві подолати більше відстані та

збільшити свій рахунок. Гра має також різноманітні локації та рівні, які додають різноманіття і виклик геймплею. Наприклад, у деяких рівнях можна зустріти босів, яких треба перемогти, або спеціальні завдання, що збільшують інтерес до гри.

З основних функціональних можливостей можна виділити:

– Гравець керує персонажем змінюючи вертикальне положення персонажу.

– Нескінченний режим з рандомною генерацією рівня.

– Генерація динамічних та статичних перешкод.

– Генерація бонусів та монет.

– Можливість покращувати персонажа.

3. Crossy Road, розроблена Hipster Whale

У грі гравець керує персонажем, що намагається перетнути різні дороги, залізниці та водні перешкоди, уникаючи зіткнень з автомобілями, потягами та іншими перешкодами. Головна мета гри - пройти якомога більше відстані, збираючи монети та намагаючись не потрапити під машини або потяги. У грі існує також можливість розвивати вміння персонажа, збираючи певну кількість монет або досягаючи певних досягнень (Рис 1.4).

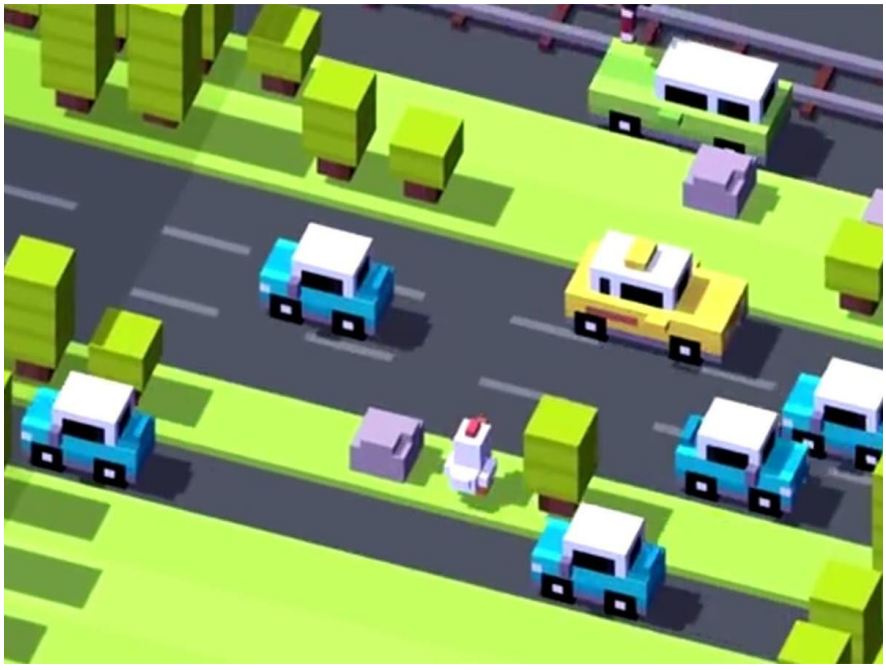

Рисунок 1.4 – Скріншот екрану з гри Crossy Road.

Крім того, Crossy Road відзначається яскравим і кольоровим візуальним стилем, а також різноманітністю персонажів, яких можна використовувати під час гри. Кожен персонаж має свої унікальні характеристики та вигляд, що додає грі різноманітності

З основних функціональних можливостей можна виділити:

- Гравець, свайпами по екрану, керує персонажем.
- Нескінченний режим з рандомною генерацією рівня.
- Генерація бонусів і монет.
- Можливість відкривати нових персонажів.

Гра отримала багато позитивних відгуків, що від гравців, що від критиків. Гра заробила більше 10 мільйонів доларів і мала більше 50 мільйонів завантажень через 3 місяці після релізу.

Ігри в жанрі раннер є популярними вже більше 10 років і приваблюють гравців з багатьох причин. Спрощений геймплей цих ігор дозволяє легкою вчитися і надає доступність для широкого кола аудиторії, включаючи новачків в світі відеоігор. Ця простота поєднується з захоплюючим геймплеєм, де гравець безперервно рухається вперед, намагаючись уникнути перешкод, що стимулює адреналін і дає виклик.

Раннери також вражають своєю різноманітністю. Вони можуть включати різні типи перешкод, локацій та персонажів, що додає цікавості і різноманітності до гри. Кожна нова спроба може призвести до використання різних стратегій та методів, що робить гру більш привабливою для гравця. Окрім того, раннери ідеально підходять для коротких ігрових сесій. Мобільні ігри часто розробляються з урахуванням сучасного темпу життя, де люди мають обмежений час для відпочинку. Раннери дозволяють швидко зануритися в гру та насолоджуватися її веселощами протягом кількох хвилин.

1.3 Аналіз програмних середовищ для розробки відеоігор

Програмне середовище для розробки відеоігор, або ігровий рушій (game engine) – це програмне забезпечення, яке використовується для створення і розробки відеоігор. Інструменти ігрових рушіїв дозволяють пришвидшити процес розробки відеоігор, та допомогти людям з обмеженими навичками в програмуванні складних систем.

Існує багато ігрових рушіїв, що мають спільні риси, але і багато відмінностей. Кожен ігровий рушій виконує своє конкретне завдання і вибирати ігровий рушій необхідно з того, якою відеогрою ви займаєтесь. До цього вибору необхідно підійти серйозно, бо від вибору ігрового рушія може залежати багато чинників в відеогрі: починаючи візуалом і закінчуючи оптимізацією.

На сьогоднішній час найбільш популярними ігровими рушіями є Unity, Unreal Engine та Godot Engine [6]

1.3.1 Unity

Unity – це ігровий рушій, створений Unity Technologies, використовується для створення 2D, 3D ігор, ігор для віртуальної реальності (VR). Unity дозволяє створювати відеоігри, що можуть працювати на різних платформах, таких як PC, консолі, мобільні пристрої та веб браузери. Відеоігри створені на за допомогою Unity підтримують DirectX та OpenGL. Unity має зручний інтерфейс, що складається з різних панелей, вікон та інструментів, що допомагають розробникам працювати з об’єктами, сценами, скриптами та іншими компонентами проєкту (Рис 1.5). Скриптова система Unity написана на Mono – відкритому проєкті, що є імплементацією Microsoft .NET Framework [7]. Для написання скриптів розробники використовують C#, також Unity має підтримку візуального програмування (Рис 1.6).

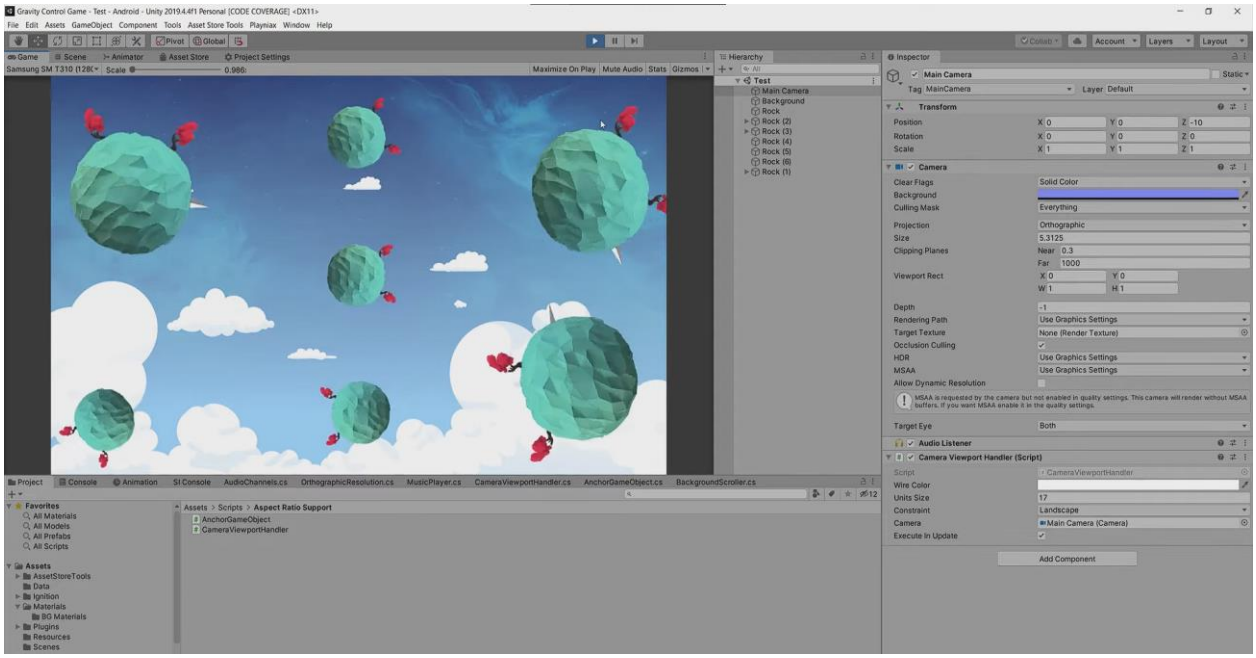

Рисунок 1.5 – Скріншот інтерфейсу редактору Unity.

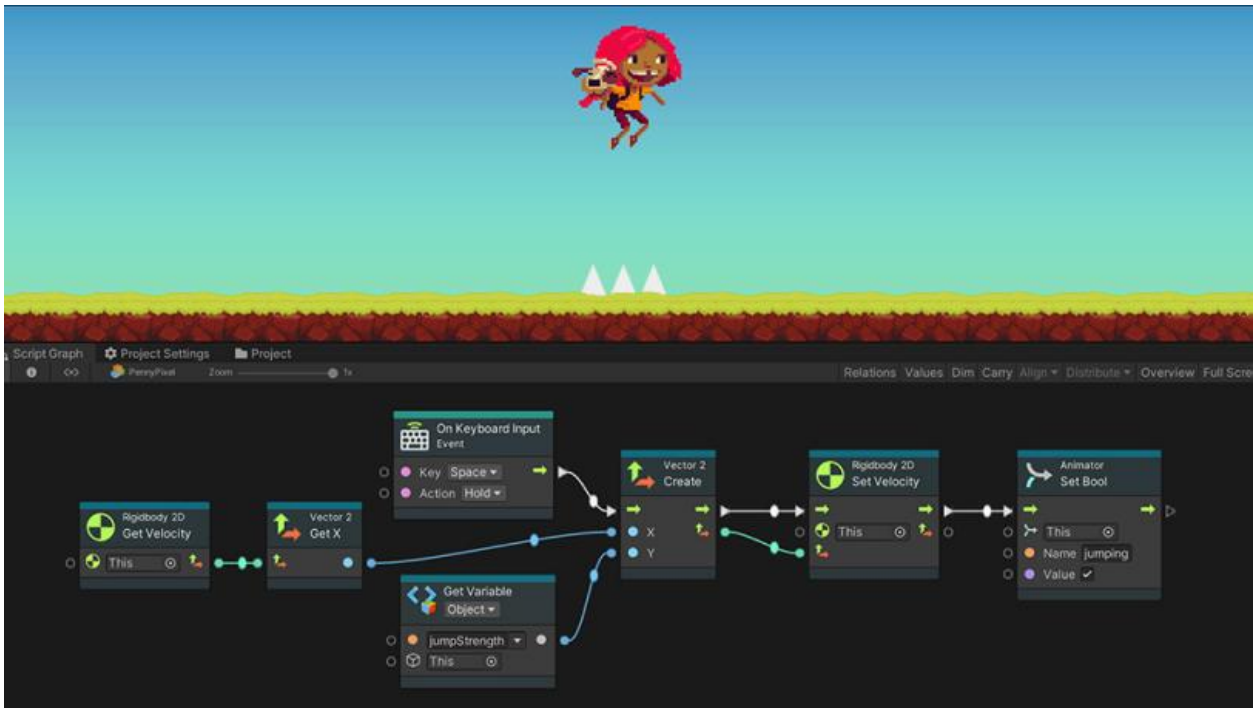

Рисунок 1.6 – Візуальне програмування в Unity

Однією із головних причин популярності Unity серед розробників є підтримкою нових розробників: Unity має велику кількість документації з різних направлень, а також має вправи для навчання [8]. Також допомогу з різних питань, пов’язаних з розробкою відеоігор можна знайти на форумі Unity. Також Unity має

свій магазин з цифровими продуктами для розробників, де можна придбати скрипти, моделі, спрайти, музику, або різні інструменти, що покращують розробку відеоігор.

Unity має безкоштовну ліцензію з повним функціоналом, яку можна використовувати якщо користувач не має річного валового доходу понад $100 тисяч, незалежно від того, чи використовується Unity Personal для комерційних цілей, чи для внутрішнього проєкту або створення прототипів, або проєкт не залучив коштів на суму понад $100 тисяч.

Загалом Unity – це потужний ігровий рушій, який надає розробникам широкі можливості для створення якісних ігор на різних платформах. Завдяки кросплатформеності, візуальному редактору, активній спільноті та розширюваності, Unity займає високу позицію серед ігрових рушіїв на ринку. Його легкість використання, а також можливість створювати як 2D, так і 3D ігри, роблять його відмінним вибором для розробників будь-якого рівня досвіду. Також важливо відзначити, що Unity є ефективним інструментом для створення ігор для віртуальної реальності, що дозволяє розробникам втілити свої ідеї в захоплюючих VR-проектах. Unity продовжує залишатися одним з найпопулярніших і найефективніших ігрових рушіїв у сучасній індустрії відеоігор.

1.3.2 Unreal Engine

Unreal Engine - це потужний та популярний ігровий рушій, розроблений компанією Epic Games. Він відомий своїми високоякісними графічними можливостями, широкими інструментами для розробки ігор та інтеграцією з різними платформами. Використовується для створення різноманітних ігор: 2D, 3D, VR. Та підтримує різні платформи: PC, консолі, мобільні девайси. Проте основним призначенням Unreal Engine є розробка саме 3D ігор, адже має потужні інструменти для створення фотореалістичної графіки, такі як фізично коректний рендеринг (PBR), динамічне глобальне освітлення (Lumen), трасування променів (Ray Tracing) та інші. Ігровий рушій має комплексний та багатофункціональний

інтерфейс, що забезпечує широкий спектр інструментів для створення ігор (Рисунок 1.7)

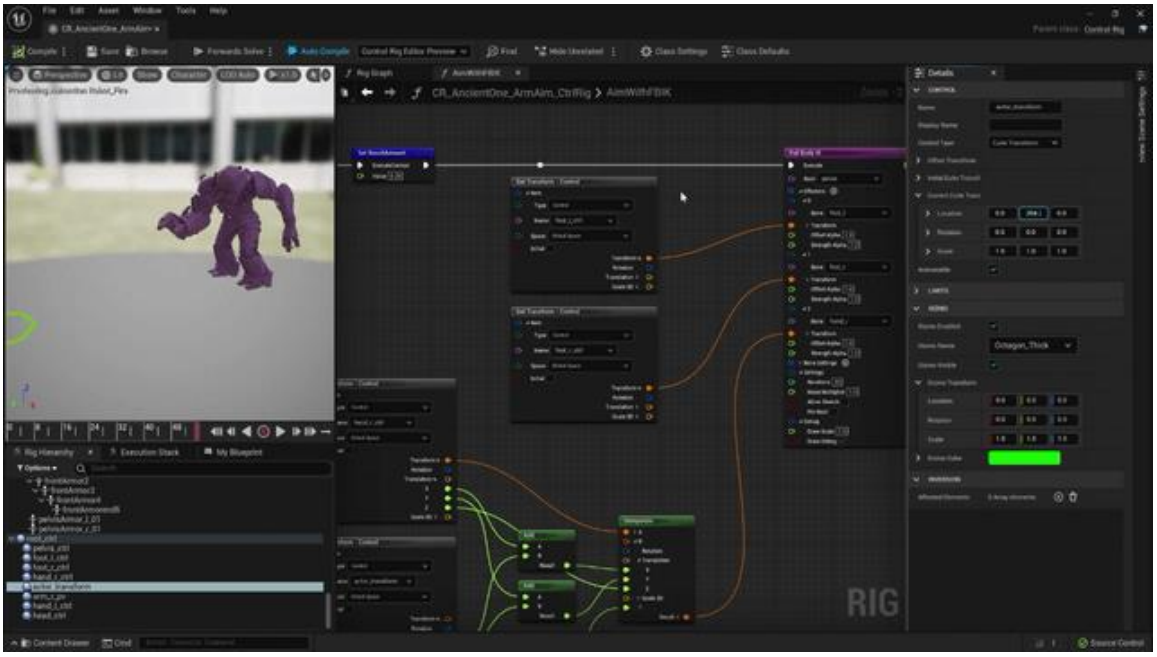

Рисунок 1.7 – Інтерфейс Unreal Engine

Unreal Engine написаний на C++ і для написання скриптів розробники також мають використовувати C++, також є підтримка візуального програмування.

Ігровий рушій розповсюджується за умовно-безкоштовною ліцензією. Розробники можуть безкоштовно використовувати рушій, але Epic Games вимагає 5% роялті коли валовий дохід від цього продукту за все життя перевищить 1 мільйон доларів США, іншими словами, перші 1 мільйон доларів не підлягають оподаткуванню роялті.

Unreal Engine є одним з найпопулярніших ігрових рушіїв у світі з багатьох причин:

– Висока якість графіки. Unreal Engine відомий своїми можливостями рендерингу, завдяки своїх можливостей у рендерингу візуалу, з використанням новітніх технологій, таких як PBR, Ray Tracing та Lumen.

– Візуальне програмування. Unreal Engine підтримує систему Blueprints, що дозволяє створювати ігрову логіку без необхідності писати код. Це спрощує

етап створення прототипу гри, та спрощує процес розробки для дизайнерів та художників, що не мають високих навичок в написані коду.

– Підтримка багатоплатформенності. Unreal Engine підтримує широкий спектр платформ, включаючи Windows, macOS, Linux, iOS, Android, консолі (PlayStation, Xbox, Nintendo Switch), а також VR і AR пристрої. Це дозволяє розробникам створювати ігри для різних аудиторій без необхідності використовувати різні рушії для кожної платформи.

– Підтримка та документація. Unreal Engine має велику кількість документації та навчальної інформації, також магазин з цифровими продуктами для розробників [9].

1.3.3 Godot Engine

Godot Engine – це безкоштовний ігровий рушій з відкритим вихідним кодом, що розробляється співавторством Godot Engine Community. Використовується для створення 2D та 3D ігор. Та підтримує різні платформи: PC, консолі, мобільні девайси. Ігровий рушій відомий своєю гнучкістю, зручним інтерфейсом і активною спільнотою. Саме спільнота Godot Engine і відповідає за розвиток ігрового рушія, адже метою цього ігрового рушія це бути повністю самодостатнім середовищем для розробки ігор під вільною ліцензією MIT. Godot Engine дозволяє створювати ігри з нуля, не використовуючи жодних інших інструментів, крім інструментів, необхідних для створення ігрового контенту, таких як графіка та музика. Архітектура Godot Engine побудована навколо концепції дерева сцен (Рис 1.8). Кожен елемент сцени може бути повноцінної сценою в будь-який час. Тому в процесі розробки легко змінювати архітектуру проєкту, розширювати його елементи і маніпулювати сценами. Для написання скриптів розробники можуть використовувати C# або C++, також є GDScript – об’єктно-орієнтована мова програмування створена спеціальна для Godot.

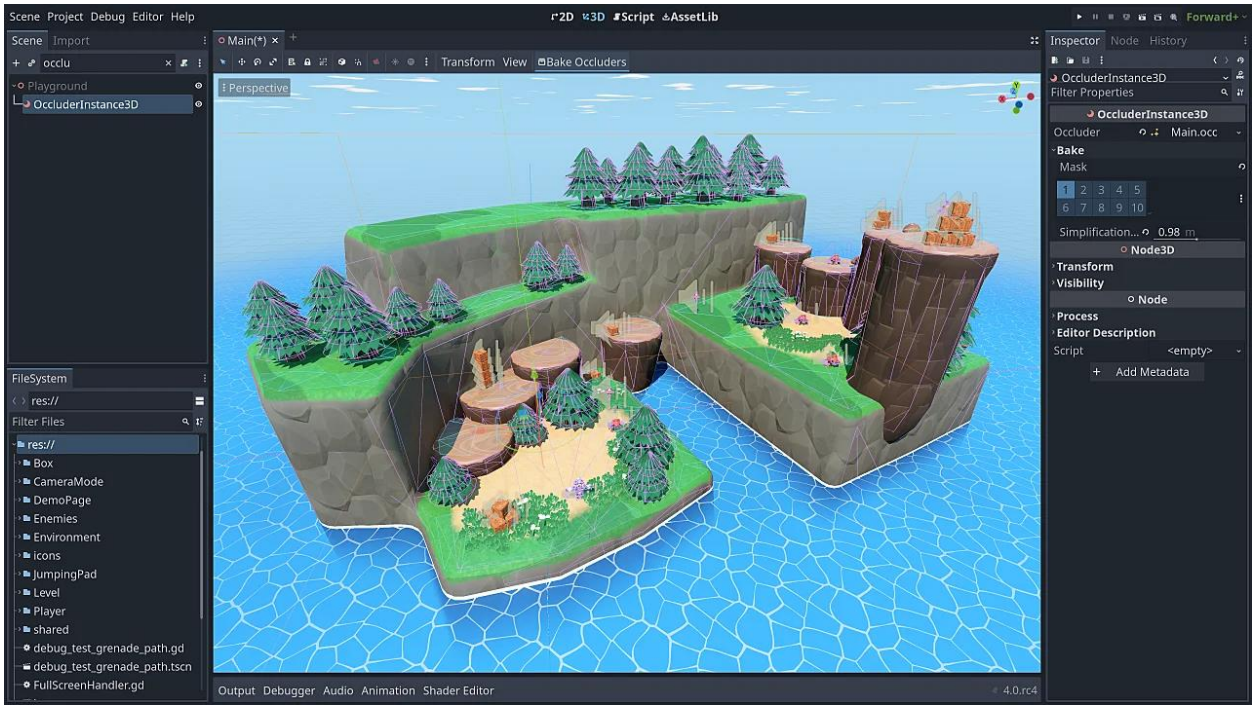

Рисунок 1.8 – Інтерфейс Godot Engine

Графічна система базується на OpenGL ES2.0. Рендеринг включає в себе повноекранні постефекти такі як FXAA, HDR, гама-корекція, динамічні тіні. Для створення шейдерів використовується спрощена шейдерна мова, близька до мови GLSL, також можливо створювати повноцінні шейдери в візуальному редакторі.

Godot Engine став популярним завдяки кільком факторам, що роблять його привабливим вибором для розробників ігор. Серед ключових факторів можна виділити:

– Безкоштовний доступ та відкритий вихідний код. Godot Engine повністю безкоштовний і не має роялті чи ліцензійних платежів, що робить його доступним для всіх розробників. Рушій має ліцензію MIT, що дозволяє вільно використовувати, змінювати та поширювати його, сприяючи розвитку.

– Активна спільнота та підтримка. Активна спільнота та документація з численними приладами і туторіалами полегшує процес навчання.

– Швидке створення прототипів на гнучкість. Візуальне програмування дозволяє створювати ігрову логіку без написання коду, що

полегшує створення прототипів і тестування ідей. Архітектура дозволяє легко розширювати проект.

Підсумовуючи, кожен з ігрових рушіїв має свої переваги та недоліки, і вибір між ними залежить від конкретних потреб. Проте є кілька ключових причин, чому для реалізації нашого проєкту був вибраний саме Unity:

– Вбудовані інструменти. Unity має велику кількість вбудованих інструментів для реалізації 2D-платформенних ігор.

– Інтуїтивно зрозумілий редактор. Unity має зручний і інтуїтивно зрозумілий інтерфейс, що дозволяє швидко працювати в редакторі.

– Спільнота та підтримка. Unity має одну з найбільших спільнот розробників відеоігор, що забезпечує доступ до великої кількості туторіалів, форумів та відеоуроків. Також Unity пропонує детальну офіційну документацію, що допомагає швидко розбиратися з можливостями рушія, та реалізовувати нові механіки.

– Безкоштовна ліцензія. Unity має безкоштовну ліцензію, що підходить для створення інді-ігор.

Підсумовуючи, Unity є відмінним вибором для створення 2D-платформенної гри в жанрі раннер.

1.4 Планування проєкту

Для планування проєкту необхідно визначити цілі проєкту, та результати, що ми хочемо отримати. Основною ціллю та кінцевою метою проєкту є створення відеогри, що може надати гравцю ігровий досвід, який відповідає концепції і особливостям жанру runner. Гра повинна бути цікавою та захоплюючою для гравців, щоб вони залюбки поверталися до неї знову та знову. Гра повинна мати яскраву та привабливу графіку, яка буде привертати увагу користувачів та створювати приємне візуальне враження. Система повинна бути оптимізована для

мобільних пристроїв та забезпечувати плавну гру. Система повинна мати можливості для реалізації реклами, або інших методів монетизації гри.

Виділимо вимоги до проєкту:

Бізнес вимоги:

1. Геймплей та функціональність:
   - Гра повинна мати простий, динамічний геймплей, щоб бути привабливою для широкої аудиторії.
   - Система управління повинна бути оптимізована для сенсорних екранів мобільних пристроїв.
2. Вміст гри:
   - Наявність декількох режимів гри.
   - Різноманітні перешкоди, які гравець повинен уникати або подолати.
   - Система збирання монет, та биття рекордів для стимулювання гравця до активної гри
   - Яскрава, піксельна графіка.
3. Залучення користувачів:
   - Можливість підключення до онлайн-лідербордів для стимулювання конкуренції між гравцями.
   - Регулярне оновлення контенту та додавання нових елементів для збереження інтересу гравців на тривалий час.

Вимоги користувачів:

- Інтуїтивне та легке управління.
- Динамічний та різноманітний геймплей, щоб уникнути монотонності та зберегти інтерес гравців на тривалий час.
- Система досягнень, що мотивує гравців.

Функціональні вимоги:

- Гравець керує персонажем, змінюючи напрямок стрибків.
- Рівні на проходження та нескінченний режим з рандомною генерацією рівня.

- Генерація динаміних та статичних перешкод.
- Генерація бонусів у вигляді монет.
- Можливість купувати різних героїв за монети.

Також необхідно визначити перелік користувачів, для яких створюється відеогра:

- Люди, які зазвичай грають у мобільні ігри, і шукають нові аркадні ігри для проходження.
- Гравці, що цінують можливість грати в гру на мобільних пристроях, у будь-який час і в будь-якому місці.
- Гравці, що грали в інші раннери і хочуть спробувати нові ігри в цьому жанрі.
- Любителі викликів, які шукають гру зі зростаючою складністю та можливістю побити власні рекорди.

Визначаємо ризики при розробці системи:

- Неякісне управління. Складне або неефективне управління героєм може зіпсувати геймплей та зробити гру неприємною для користувачів.
- Монотонність геймплею. Повторюваність та відсутність різноманітності може призвести до втрати інтересу гравців.
- Технічні проблеми. Баги та збої можуть спричинити поганий досвід користувача та негативно вплинити на репутацію гри.
- Недостатній маркетинг. Низька відомість гри серед цільової аудиторії може призвести до низького обсягу завантажень та низьких доходів.

Розробка мобільної гри – це комплексний процес, що вимагає ретельного планування та врахування багатьох аспектів. В даному плані виділили основні типи вимог до проєкту, зокрема бізнес вимоги, вимоги користувачів, функціональні вимоги. Це дозволяє чітко зрозуміти, які саме функції та характеристики повинні бути реалізовані в грі, щоб забезпечити її відповідність очікуванням користувачів. Також визначили цільову аудиторію відеогри, що включає різні категорії користувачів. Це допомагає краще адаптувати контент гри

та її механіку до потреб і вподобань різних груп гравців. Окрему увагу було приділено ідентифікації ризиків, які можуть виникнути під час розробки гри. Виявлення ризиків на ранніх стадіях дозволяє розробити стратегії для їх мінімізації та забезпечити безперебійний процес розробки. Таким чином, планування проєкту враховує всі ключові аспекти розробки мобільної гри, це слугує надійною основою для успішної реалізації проєкту, забезпечуючи чіткий напрямок розробників та гарантує високу якість кінцевого продукту, що відповідатиме очікуванням користувачів і ринковим вимогам.

РОЗДІЛ 2 РЕАЛІЗАЦІЯ ПРОЄКТУ

2.1 Графічні елементи проєкту

Графіка відіграє одну з ключових ролей у створені відеоігор, впливаючи на різні аспекти ігрового досвіду. Саме графіка впливає на перші враження від гри, привертаючи увагу гравців. Візуально привабливі ігри мають вищі шанси на успіх, графіка допомагає створити відповідну атмосферу та підкреслити тематику гри.

Для створення графічної частини 2D-відеогри використовують різні елементи, такі як тайл та спрайт.

Тайл (англійською “tile”) – основний елемент у створені ігрових рівнів в 2D-відеоіграх. Тайл представляє собою невелике, зазвичай квадратне, зображення або шматок текстури, що повторюються і розташовуються у вигляді сітки для створення більших, складніших середовищ. Тайли використовуються для створення ігрових рівнів, компонуючи їх у сітки для формування ігрового середовища, це дозволяє швидко і ефективно будувати великі ігрові світи.

Спрайт (англійською “sprite”) – двовимірне зображення або анімація, що використовується для представлення об’єктів у відеоіграх. Спрайти можуть бути персонажами, об’єктами, фонами, елементами інтерфейсу та іншими компонентами гри. Спрайти можуть бути анімовані шляхом швидкої зміни серії зображень.

2.1.1 Тайли ігрового рівня

Для створення рівнів та побудови оточення були використані тайли з відкритою ліцензією [10]. Ці тайли необхідні для створення рівнів (Рис 2.1).

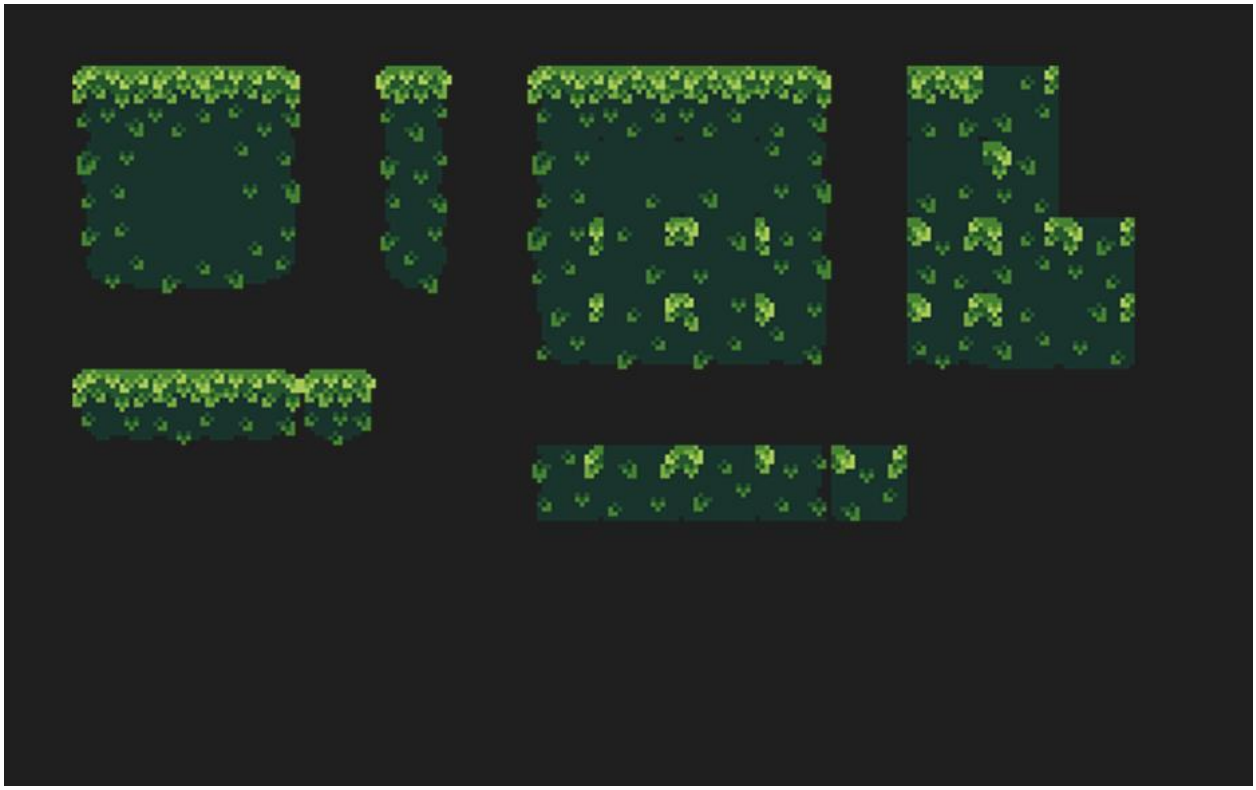

Рисунок 2.1 – Тайли джунглів.

2.1.2 Спрайти фону

Для створення оточення були використаін спрайти з відкритою ліцензією [10]. Ці спрайти необхідні для відображення фону і створення ефекту паралаксу (Рис 2.2)

Рисунок 2.2 – Спрайти фону гри

2.1.3 Спрайти головного героя та ворогів

Головним героєм виступає ніндзя, для створення спрайтів головного героя був використаний aseprite – програма для створення та редагування піксельних зображень. Спрайт має частини тіла, що будуть анімовані за допомогою інструментів ігрового рушія (Рис 2.3)

Рисунок 2.3 – Спрайт головного герою

В грі існує два види ворогів, один з них створений в aseprite, його спрайти представляють собою колекцію кадрів для створення анімації (Рис 2.4, Рис 2.5).

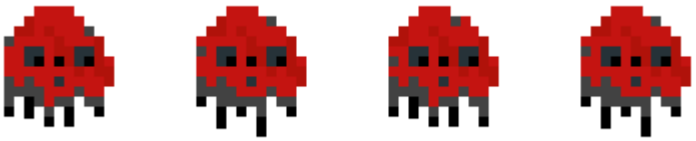

Рисунок 2.4 – Спрайт ворогу в стані спокою

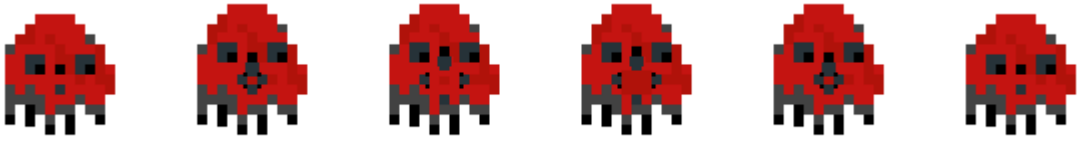

Рисунок 2.5 – Спрайт ворогу в стані атаки

Для другого ворогу були взяті спрайти з відкритою ліцензією [11]. Ці спрайти представляють собою колекцію кадрів для створення анімації (Рис 2.6, Рис 2.7)

Рисунок 2.6 – Спрайт ворогу в стані спокою

Рисунок 2.7 – Спрайт ворогу в стані атаки

2.1.4 Спрайти ігрового оточення

Для створення ігрового оточення в грі були використані різні перешкоди та пастки, для їх створення були використані спрайти з відкритою ліцензією [12]. Деякі з цих спрайтів використовуються для анімації (Рис 2.8, 2.9)

Рисунок 2.8 – Спрайт циркулярної пили

Рисунок 2.9 – Спрайт шипів на колесах

Та монети, що виступають валютою в відеогрі (Рис 2.10)

Рисунок 2.10 – Спрайт монет

2.1.5 Спрайти інтерфейсу

Інтерфейс користувача є важливою складовою будь-якої відеогри, оскільки він забезпечує зв’язок між гравцем та грою. Для інтерфейсу були створені спрайти в aseprite (Рис 2.11), та використані спрайти з відкритою ліцензією [13]. Ці спрайти представляють собою меню та кнопки для інтерфейсу (Рис 2.12)

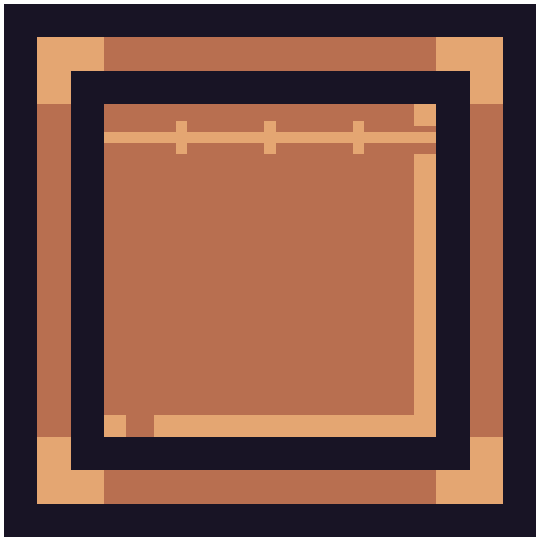

Рисунок 2.11 – Спрайт панелі для меню

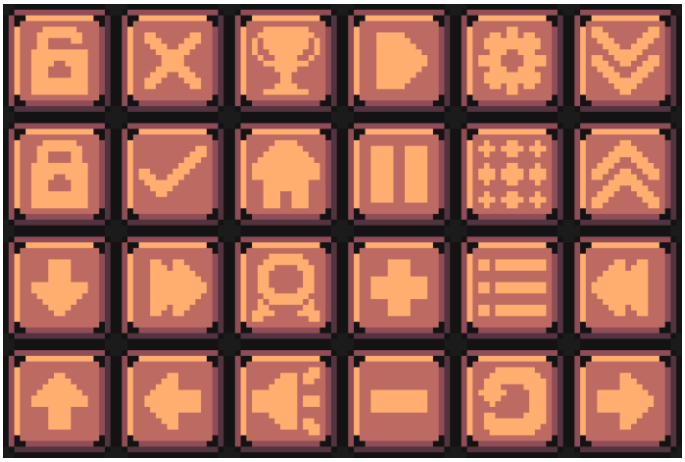

Рисунок 2.12 – Спрайти кнопок для інтерфейсу гри

2.2 Створення анімацій

Ігровий рушій Unity має вбудовані інструменти для створення анімації, такі як Animation та Animator.

Animation – інструмент для створення і редагування анімаційних кліпів, що можуть бути застосовані до різних об’єктів в грі. Дозволяє створювати рухомі дії для персонажів, об’єктів і навіть інтерфейсних елементів.

Animator – інструмент для управління складними анімаційними системами через створення і налаштування анімаційних станів та переходів між ними [14]. Animator надає можливість створювати логічні зв’язки між різними анімаційними кліпами, що робить анімації більш динамічними та інтерактивними.

2.2.1 Анімації головного героя

Для створення анімації головного героя використаємо скелетну анімацію. Скелетна анімація це техніка анімації, що використовується для створення рухомих моделей, персонажів або будь-яких інших істоти, що мають рухомі частини. Скелетна анімація складається з двох основних частин: скелета та мешу. При використанні скелетної анімації зменшується використання пам’яті та ресурсів, оскільки зберігаються лише дані про рухи кісток, а не кожен окремий кадр, як при покадровій анімації. Анімаційні кліпи, створені для одного персонажа, можуть бути легко повторно використані для інших персонажів з подібною скелетною структурою, що значно економить час і ресурси при створенні нових анімацій.

Для створення скелетної анімації в Unity має інструмент Skinning Editor [15]. За допомогою Skinning Editor на спрайті героя створюємо скелет (Рис 2.13)

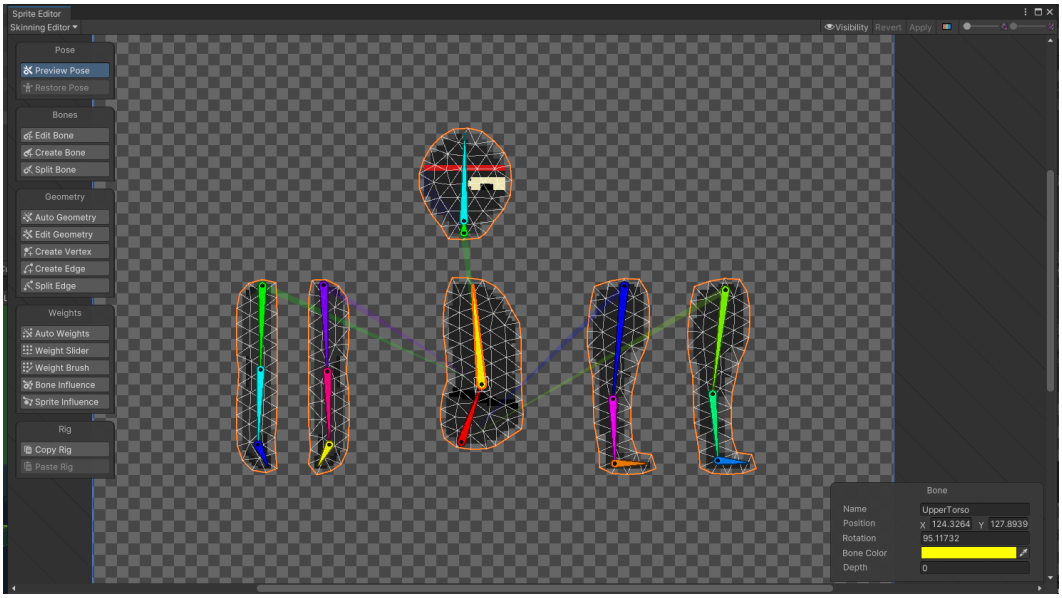

Рисунок 2.13 – Скелет головного герою

Після створення скелету – генеруємо меш, що визначає, які кістки впливають на певні вершини і наскільки сильно (Рис 2.14)

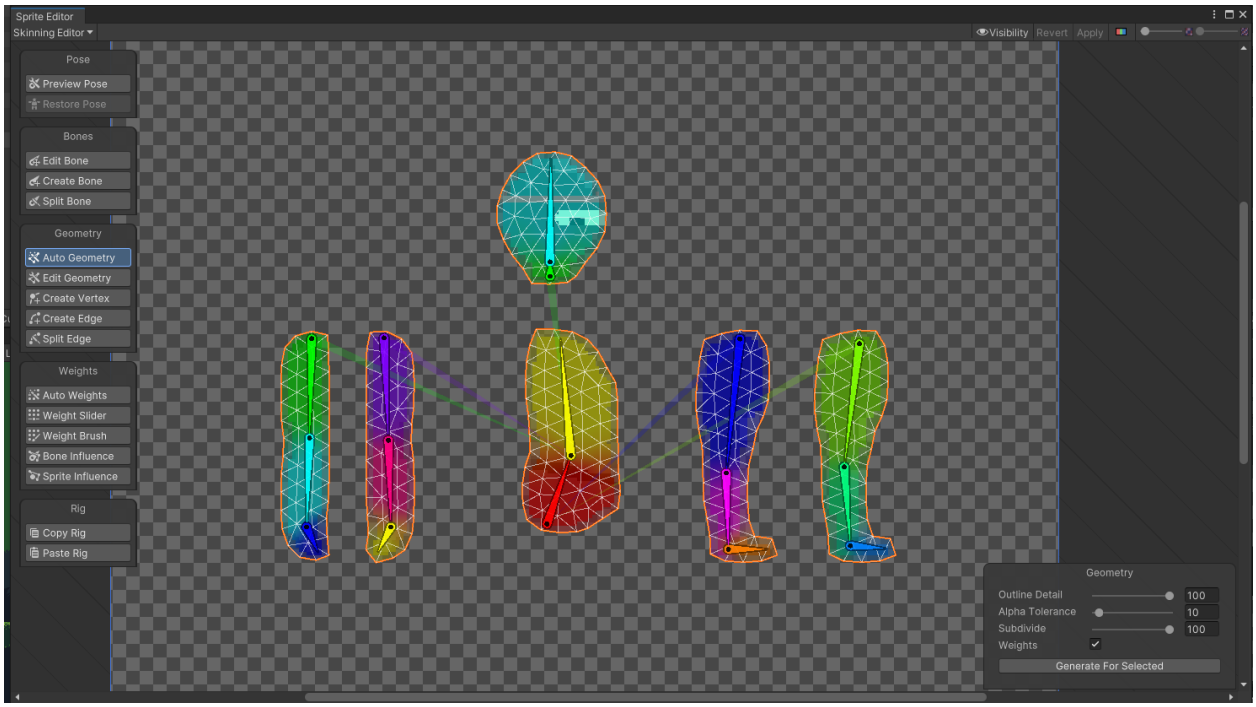

Рисунок 2.14 – Згенерований меш головного герою.

З готовим скелетом і мешем анімуємо героя за допомогою Animation, в аніматорі створюємо дерево станів та параметрів (Рис 2.15)

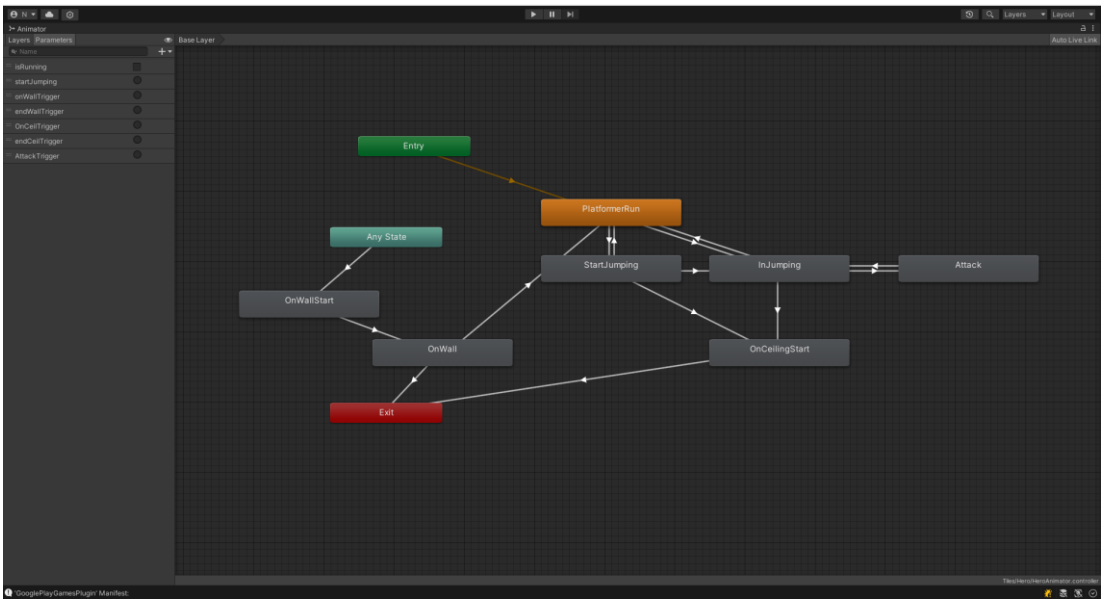

Рисунок 2.15 – Вікно аніматору з деревом станів герою.

Створення анімації з використанням скелетної анімації відбувається шляхом зміни розташування кісток (Рис 2.16), таким чином створюємо анімацію для кожного стану героя.

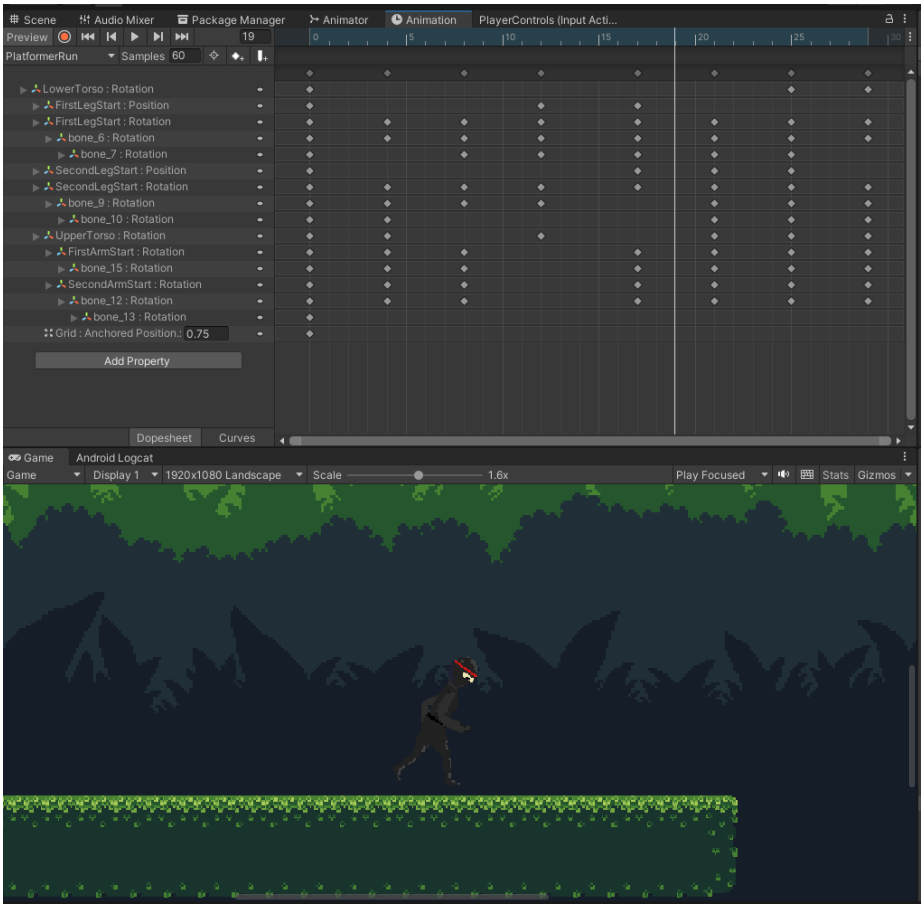

Рисунок 2.16 – Вікно Animation та візуалізація анімації бігу героя.

2.2.2 Анімація ворогів та оточення

Для створення анімації ворогів та оточення використаємо покадрову анімацію. Покадрова анімація – це традиційна техніка анімації, де кожен кадр створюється окремо, щоб зображати послідовні рухи або зміни об’єкта. Коли кадри відтворюються у швидкій послідовності, створюється ілюзія руху.

Для створення покадрової анімації в Unity, використаємо інструмент Animation, анімуємо одного з ворогів шляхом виставлення кадрів на таймлайн (Рис 2.17). Таким чином анімуємо і інші об'єкти, що використовують покадрову анімацію

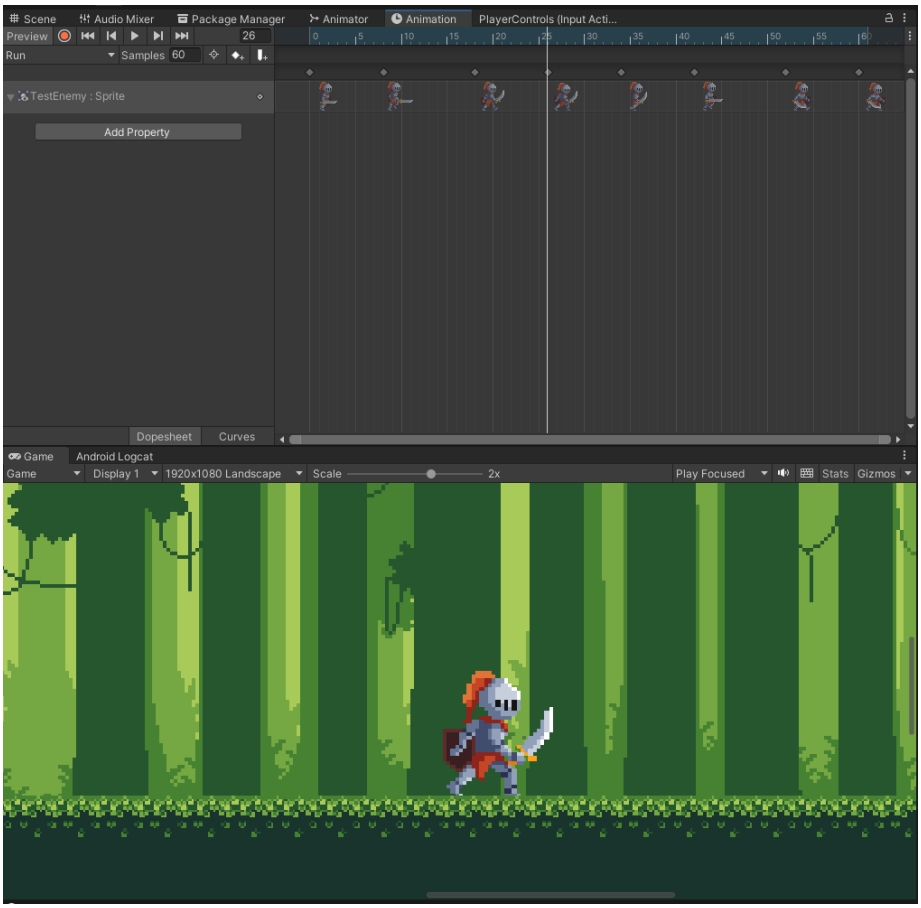

Рисунок 2.17 – Вікно Animation та візуалізація анімації бігу ворогу

2.3 Головний герой

Головний герой представляє собою ніндзю, що має долати перешкоди та ворогів, щоб проходити рівні.

2.3.1 Управління головним героєм

Для управління та взаємодії з грою Unity надає систему введення (Input System), що надає розробникам можливість легко налаштовувати і керувати вводом з різних пристроїв, забезпечуючи більш зручний та ефективний процес розробки [16]. Для цього створюємо Input Action Asset, в якому додаємо Action maps для самого вводу гри (Touch) та для взаємодії з інтерфейсом (UI) . В Touch Action maps створюємо дії (Рис 2.18):

- PrimaryContact. Представляє собою кнопку, що визначає чи торкається користувач екрану
- PrimaryPosition. Представляє собою двовимірний вектор, що має інформацію де саме знаходиться палець користувача в даний момент.
- PrimaryStartPosition. Представляє собою двовимірний вектор, що має інформацію де знаходився палець користувача в момент початку введення.

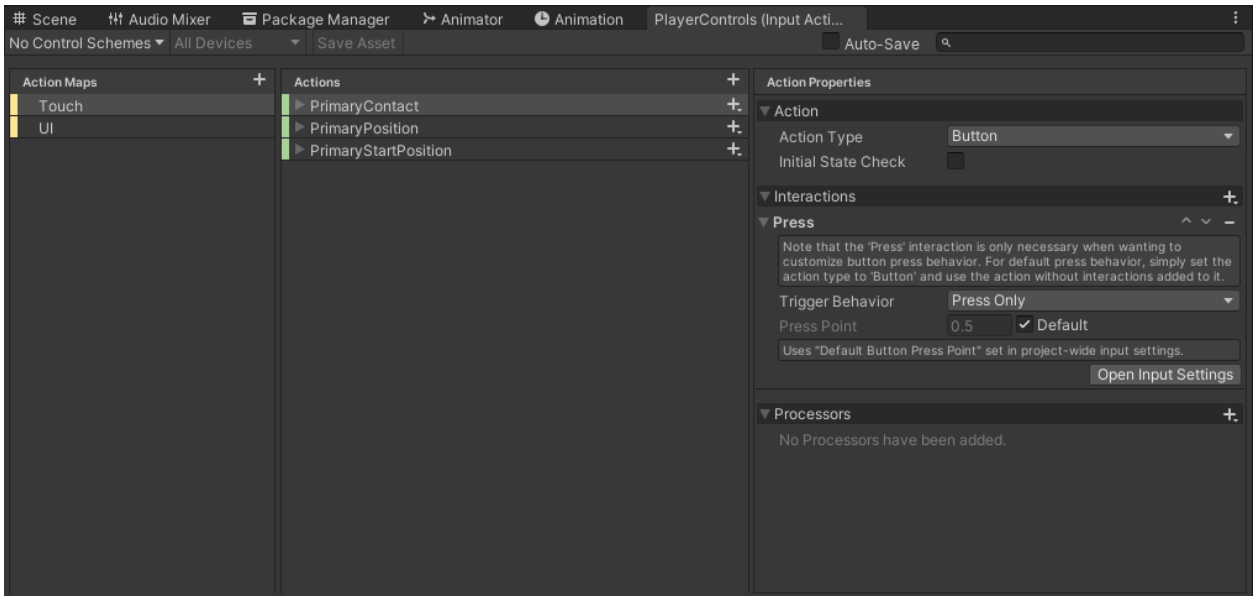

Рисунок 2.18 – Меню Input Action Asset

Тепер для написання логіки вводу, на сцені, створюємо InputManager (Рис 2.19), до якого додаємо Player Input, що буде мати посилання на Input Action Asset, та створюємо скрипт New Input Manager, зв’язок між іншими скриптами буде відбуватися за допомогою Unity Event (Рис 2.20).

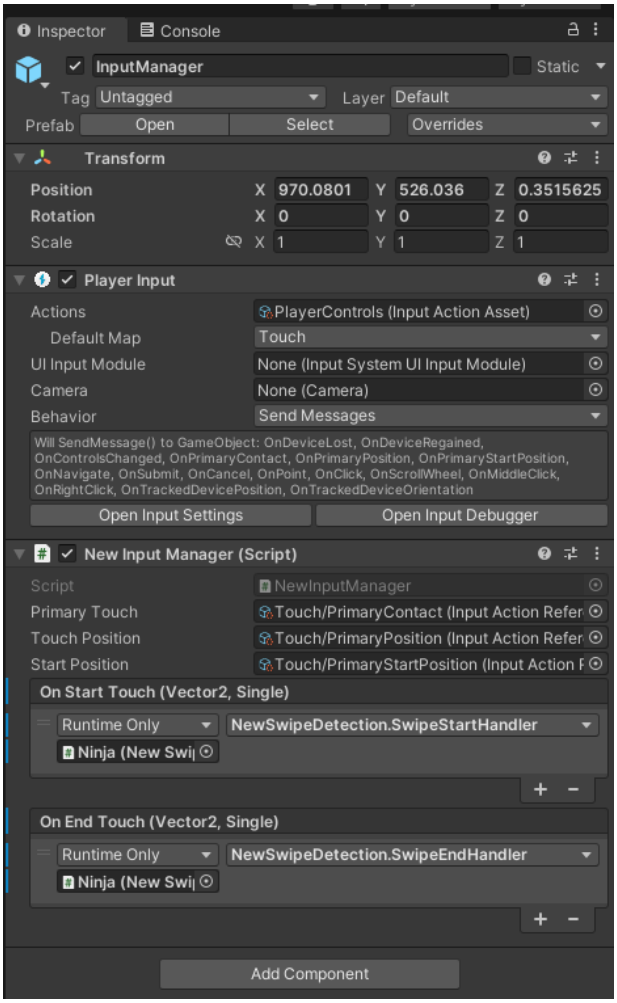

Рисунок 2.19 – Об’єкт InputManager.

```
2 usages  new *
private async void StartTouchPrimary(InputAction.CallbackContext ctx)
{
    await Task.Delay(10);

    OnStartTouch?.Invoke((Vector2)Utils.ScreenUtils.ScreenToWorld(mainCamera,
        (Vector3)_startPosition.ReadValue<Vector2>()), (float)ctx.startTime);
    if (isUseCancellationToken)
    {
        TokenSource?.Cancel();
        isUseCancellationToken = false;
    }
}

2 usages  Roflanpepe *
private void EndTouchPrimary(InputAction.CallbackContext ctx)
{
    OnEndTouch?.Invoke((Vector2)Utils.ScreenUtils.ScreenToWorld(mainCamera,
        (Vector3)_touchPosition.ReadValue<Vector2>()), (float)ctx.startTime);
}
```

Рисунок 2.20 – Виклик Unity Event.

До об’єкта героя додаємо компоненти Rigidbody 2D та Capsule Collider 2D, що необхідні для того, щоб об’єкт вважався фізичним тілом в Unity, та створюємо скрипт New Swipe Detection, що буде приймати інформацію з New Input Manager, та перевіряти чи можливо виконати стрибок. Зв’язок з іншими системами буде виконуватися за допомогою event Action OnSwipe (Рис 2.21).

```
1 usage   Roflanpepe
private void DetectSwipe()
{
    // Debug.Log("######################");
    // Debug.Log($"Vector swipe distance: {Vector3.Distance(startPosition,endPosition)}");

    if (Vector3.Distance((Vector3) a: startPosition, (Vector3) b: endPosition) >= minimumDistance &&
        (endTime - startTime) <= maximumTime)
    {
        // Debug.Log("Jump swipe distance: "+ Vector3.Distance(startPosition, endPosition));
        --currentSwipeCount;

        Debug.DrawLine((Vector3)startPosition, (Vector3)endPosition, Color.red , duration: 5f);

        directionSwipe = (Vector3)(endPosition - startPosition);
        OnSwipe?.Invoke();
        // ResetAllValue();
    }
}
```

Рисунок 2.21 – Виклик Action OnSwipe.

2.3.2 Пересування головного героя

В раннерах головний герой постійно біжить вперед, а гравець може рухати персонажа так, щоб уникати перешкод. Для пересування персонажу використовується стейт-машина (State Machine) та скрипт MovementComponent, який зберігає інформацію про дані, що використовуються для визначення положення героя (Рис 2.22). Стейт-машина – концептуальна модель, яка використовується для представлення системи, що перебуває в одному з декількох можливих станів, в нашому випадку це стани пересування героя: біг, стрибок, висіння на стіні, висіння на стелі, політ, стан атаки. Всі ці стани наслідуються від базового та контролюються скриптом стейт-машини (Рису 2.23).

```
public class MovementComponent : MonoBehaviour
{
    Frequently called  1 usage
    [field: SerializeField] public float Speed { get; private set; }  "400"
    Frequently called  5 usages
    [field: SerializeField] public float GroundCheckRadius { get; private set;}  "0.26"
    Frequently called  5 usages
    [field: SerializeField] public LayerMask GroundLayer { get; private set; }  Serializabl
    Frequently called  5 usages
    [field: SerializeField] public Vector3 GroundCheckPosition { get; private set;}  Se
    Frequently called  3 usages
    [field: SerializeField] public Vector3 CeilingCheckPosition { get; private set; }
    [field: SerializeField] public float WallRayLength { get; private set; }  "0.39"
    Frequently called  2 usages
    [field: SerializeField] public float CeilRayLength { get; private set; }  "0.39"

    Frequently called  5 usages
    [field: SerializeField] public AgentBoxDetection WallDetection { get; private set; }

    private NewSwipeDetection swipeDetection;
    public Rigidbody2D _rigidbody2D;  Changed in 1 asset

    private bool isGrounded = false;
    private bool isOnWall = false;
```

Рисунок 2.22 – Поля класу MovementComponent.

```
16 usages  Roflanpepe  2 exposing APIs
public class PlayerStateMachine
{
    Frequently called  8 usages
    public BasedState CurrentState { get; set; }

    1 usage  Roflanpepe
    public void Initialize(BasedState startingState)
    {
        CurrentState = startingState;
        CurrentState.EnterState();
    }

    Frequently called  15 usages  Roflanpepe
    public void ChangeState(BasedState newState)
    {
        CurrentState.ExitState();
        CurrentState = newState;
        CurrentState.EnterState();
    }
}
```

Рисунок 2.23 – Клас стейт-машини.

2.3.3 Здоров’я та атака героя

Головний герой може зіштовхуватися з перешкодами та ворогами, тому необхідна система здоров’я, що гравець буде втрачати при поразці, та скрипт

атаки героя, щоб була можливість атакувати ворогів. Створимо клас PlayerHealth (Рис 2.24), що буде наслідуватися від абстрактного класу Hittable (Рис 2.25).

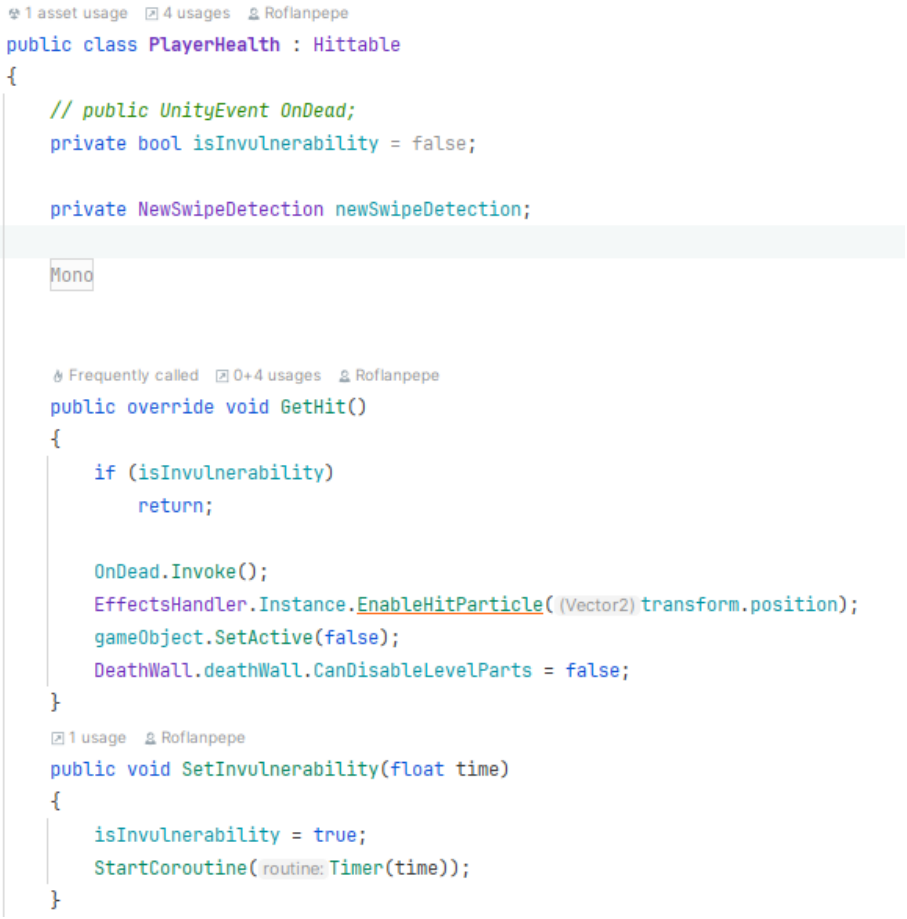

Рисунок 2.24 – клас Player.

Рисунок 2.25 – Абстрактний клас Hittable.

Для можливості атаки створюємо клас PlayerAttack, що буде наслідуватися від абстрактного AttackComponent (Рис 2.26)

```
public abstract class AttackComponent : MonoBehaviour
{
    public Collider2D[] TargetCollider2Ds;  Serializable

    Frequently called  1 usage  1 override  Roflanpepe
    protected virtual void Attack(Collider2D[] targetCollider2Ds)
    {

    }
}
```

Рисунок 2.26 – Абстрактний клас AttackComponent.

На початку атаки знаходимо колайдери в області атаки і якщо вони є – то шукаємо класи, що успадковуються від інтерфейсу IDamageable та викликаємо метод Damage() (Рис 2.27).

```
Frequently called  2 usages  Roflanpepe
private void StartAttack()
{
    TargetCollider2Ds = agentBoxDetection.OverlapBox();

    if (TargetCollider2Ds == null)
        return;
    if(TargetCollider2Ds.Length == 0)
        return;

    Attack(TargetCollider2Ds);
}
Frequently called  1+1 usages  Roflanpepe
protected override void Attack(Collider2D[] targetCollider2Ds)
{
    base.Attack(targetCollider2Ds);

    //AttackEffect.gameObject.SetActive(true);
    effect = attackEffectObjectPool.Get();
    effect.transform.position = transform.position;
    //effect scale
    AgentUtils.SpriteDirection( targetTransform: effect.transform,  gameObjectTransform: transform);

    StartCoroutine( routine: DelayEndEffect());

    foreach (var item :Collider2D in targetCollider2Ds)
    {
        if (item.TryGetComponent(out IDamageable damageable))
            damageable.Damage();
        if (item.TryGetComponent(out EnemyDeath enemyDeath))
            if(enemyDeath.IsRefreshSwipeCount)
                enemyDeath.PlayerRefreshSwipeCount(gameObject);
    }

    TargetCollider2Ds = null;
}
```

Рисунок 2.27 – Метод атаки героя.

2.4 Створення ігрових рівнів

Рівні в відеоіграх є важливими складовими структури великої частини відеоігор. Ігрові рівні визначають структуру та забезпечують логічний прогрес. Кожен рівень, зазвичай стає складнішим, ніж попередній, що допомагає гравцям поступово нарощувати свої навички. Цей поступовий підхід сприяє підтримці інтересу та мотивації гравця.

2.4.1 Побудова макету ігрового рівня

Побудова макету ігровогу рівня – ключовий етап у процесі дизайну рівнів, що включає створення базової стрктури рівня. Unity має вбудовані інструменти для побудови макету рівня, такі як Tilemap та Tilemap Collider. Tilemap дозволяє розробникам зручно та ефективно розміщувати та маніпулювати плитками на сітці. Для цього створюється палітра тайлів (Рис 2.28). Tilemap Collider дозволяє додати тайлам колізію для контакту з фізичними об’єктами рівню.

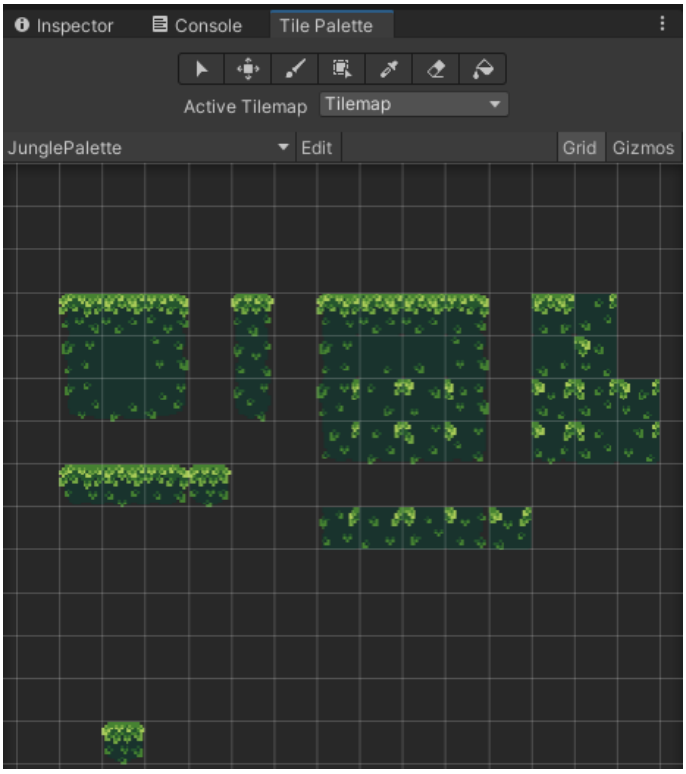

Рисунок 2.28 – Вікно палітри тайлів з тайлами джунглів.

За допомогою Tile Palette створюємо макет першого ігрового рівня (Рис 2.29), як видно з рисунка Tilemap Collider 2D автоматично створює колізію для ігрового рівня, що пришвидшує розробку.

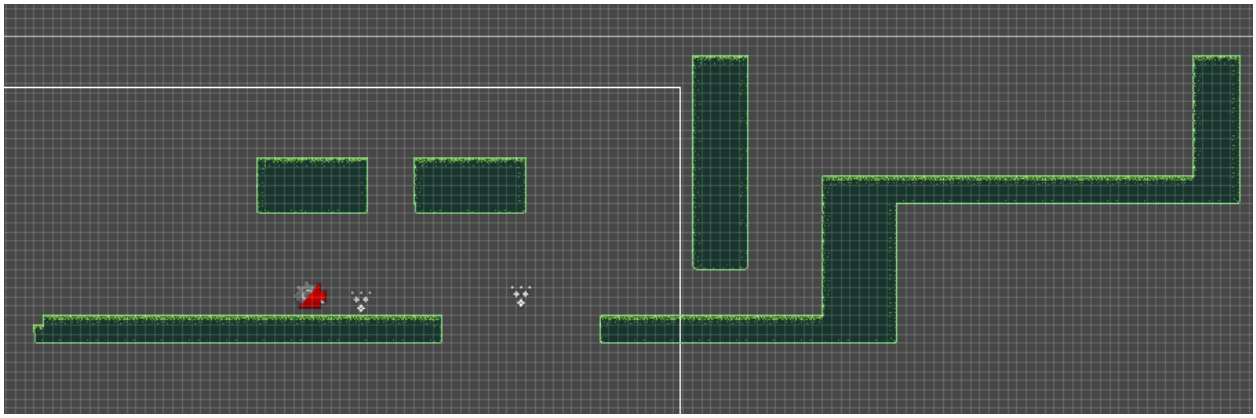

Рисунок 2.29 – Вікно сцени першого рівня.

2.4.2 Створення фону ігрового рівня

Фон ігрового рівня – це графічний елемент, що створює візуальне середовище на задньому плані ігрового рівня. Він відіграє важливу роль у визначені контексту та локації дії.

В нашому випадку фон представляє собою набір спрайтів, що рухаються з різною швидкістю, для створення ефекту паралаксу (Рис 2.30 – 2.31)

```
public class ParallaxBackGround : MonoBehaviour
{
    private float distance, temp, length, startPos;
    private Camera mainCamera;
    [SerializeField] private float parallaxEffect;  Changed in 2 assets
```

Рисунок 2.30 – Поля класу ParallaxBackGround.

```
void FixedUpdate()
{
    temp = (mainCamera.transform.position.x * (1 - parallaxEffect));

    distance = (mainCamera.transform.position.x * parallaxEffect);

    transform.position = new Vector3( x: startPos + distance,
        transform.position.y, transform.position.z);

    if (temp > startPos+ length)
    {
        startPos += length;
    }
    else if (temp<startPos - length)
    {
        startPos -= length;
    }
}
```

Рисунок 2.31 – логіка руху спрайтів для створення ефекту паралаксу.

Такий фон ігрового рівня дозволяє додати глубини зображенню, та покращити сприйняття гри (Рис 2.32)

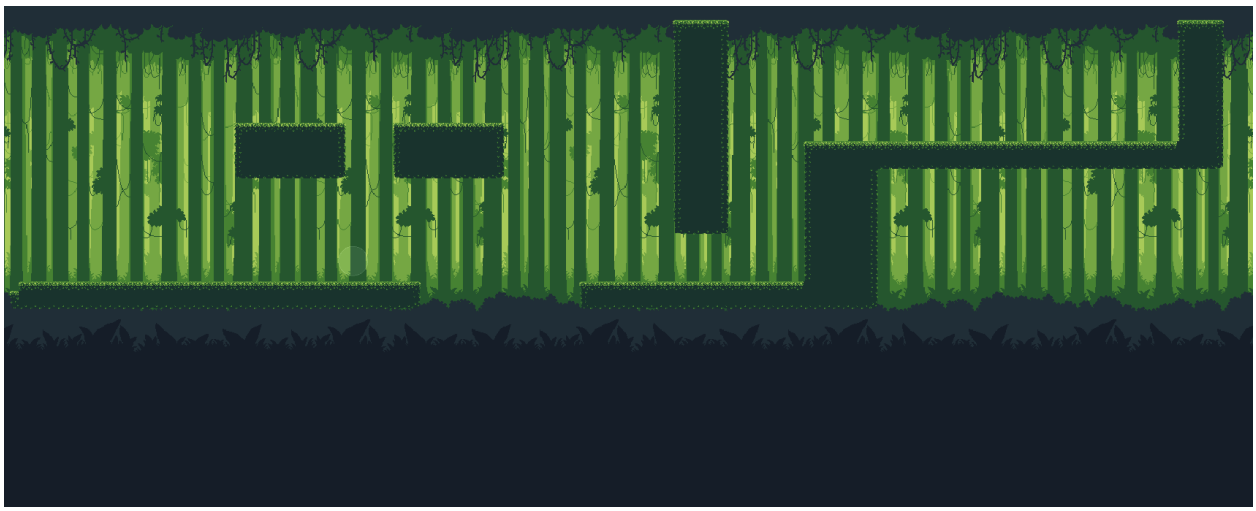

Рисунок .2.32 – Вікно сцени першого рівня з фоном.

2.4.3 Створення ігрового оточення рівня

Ігрове оточення рівня – це сукупність усіх елементів, що створюють простір, в якому гравець взаємодіє протягом ігрового процесу, це включає в себе об’єкти декорацій, перешкоди, пастки та бонуси.

В якості пасток та перешкод виступають статичні та динамічні об’єкти, такі як шипи, що просто розташовані на рівні (Рис 2.33) або стіна пилок, що переслідує головного героя і виступає стимулом до постійного пересування по рівню (Рис 2.34).

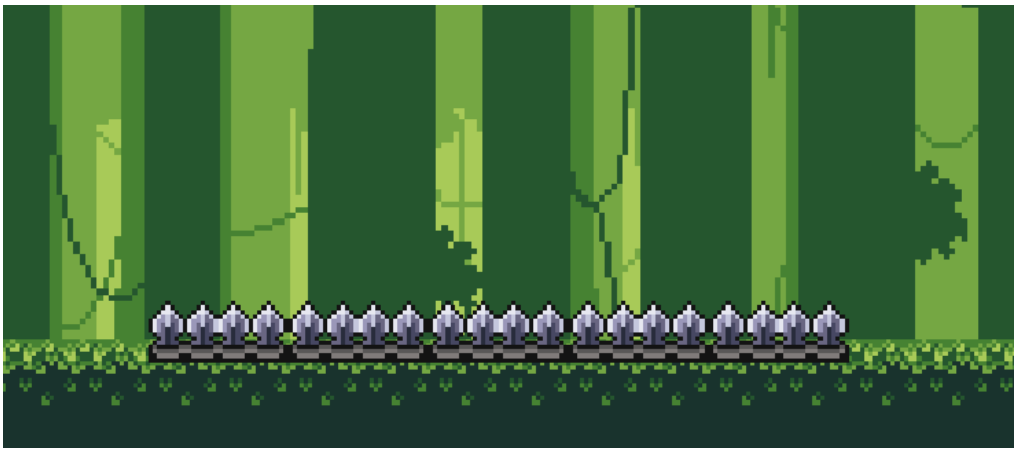

Рисунок 2.33 – Статичні шипи на рівні.

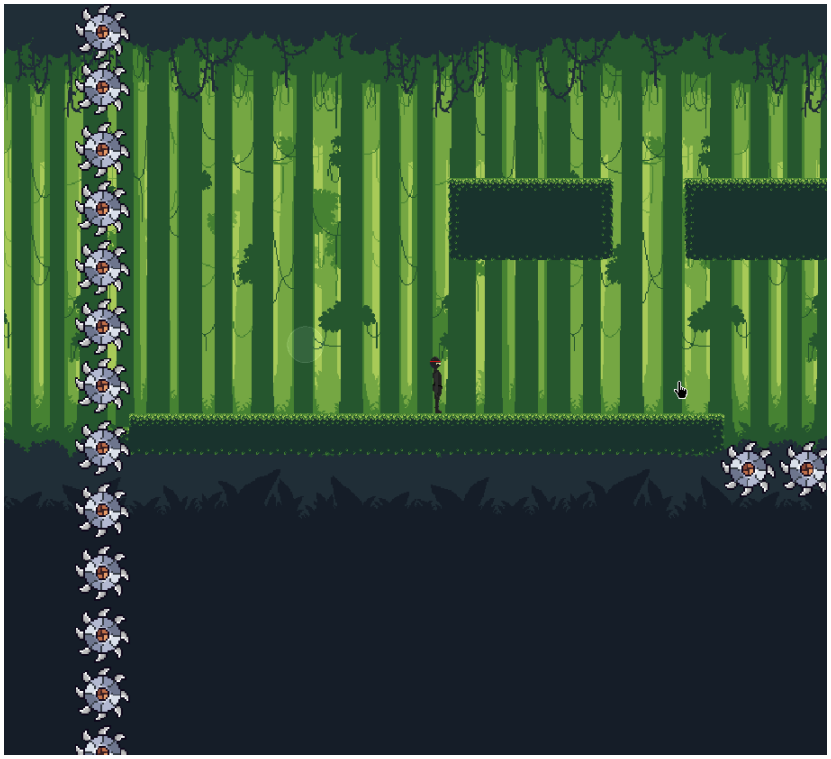

Рисунок 2.34 – стіна пилок.

В якості бонусів в грі виступають монети, що є валютою в грі. Монети діляться за рідкістю, від мідяних до золотих (Рисунок 2.35)

Рисунок 2.35 – Різні види монет.

Головною метою кожного рівня є його проходження – подолати всі перешкоди та добігти до фінішу, яким виступає портал (Рис 2.36)

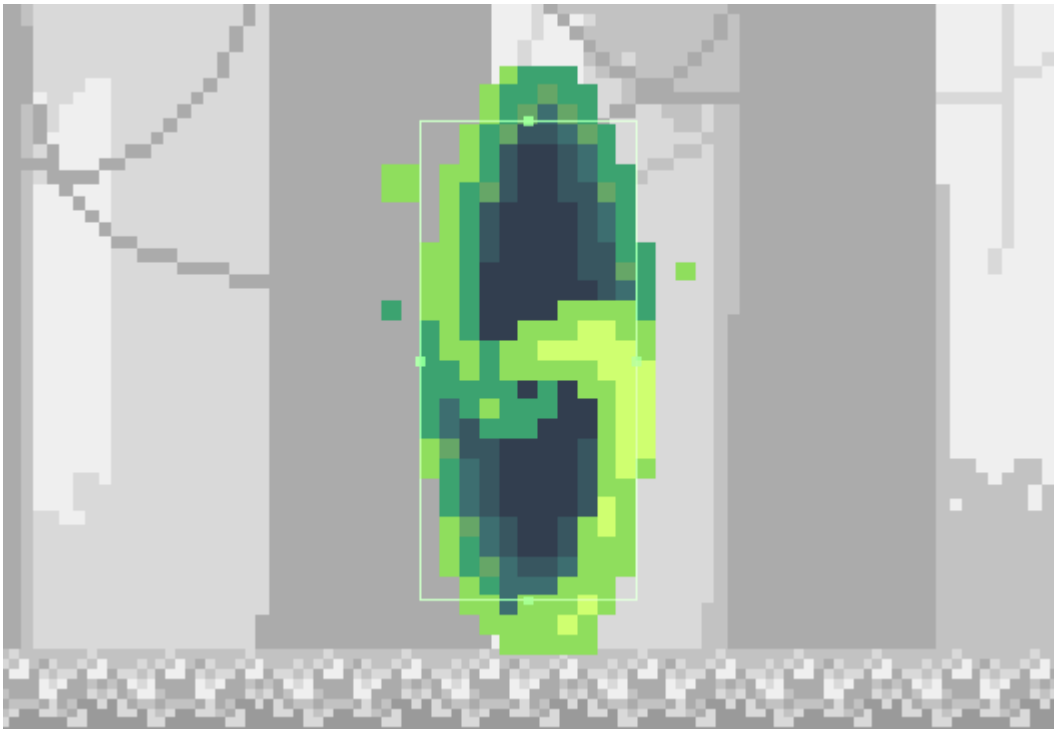

Рисунок 2.36 – Кінець рівню.

2.5 Вороги

Вороги є ще одним типом перешкод і діляться за видами атаки: дистанційна та атака ближнього бою. Всі типи ворогів працюють за допомогою створеного класу EnemyAI (Рис 2.37). Класи EnemyAttack і EnemyAbstractMovement є абстрактними і кожен з ворогів реалізує методи цих класи по-своєму.

```
public class EnemyAI : MonoBehaviour
{
    Frequently called  5 usages
    public Vector2 DirectionVector { get; private set; }
    5 usages
    public EnemyAnimator EnemyAnimator { get; private set; }
    11 usages
    public EnemyAnimationEventHandler EnemyEventHandler { get; private se

    private EnemyAttack.EnemyAttack enemyAttack;
    private EnemyAbstractMovement enemyMovement;

    Event function  Roflanpepe
    private void Awake(){...}

    Event function  Roflanpepe
    private void Start(){...}

    Event function  Roflanpepe
    private void OnDestroy(){...}

    Event function  Roflanpepe
    private void FixedUpdate()
    {
        if (enemyAttack.Attacking)
            return;
        if (!enemyMovement.IsCanMove)
            return;

        enemyMovement.Movement();

        AgentUtils.SpriteDirection(transform, DirectionVector);

    }
```

Рисунок 2.37 – Клас EnemyAI.

2.5.1 Ворог ближнього бою

Ворогом ближнього бою є лицар з мечем, що пересувається по землі (Рис 2.38 – 2.40).

```
Frequently called  1+3 usages  new *
public override void Movement()
{
    base.Movement();

    if(transform.position.x > rightCorner.position.x)
        ChangeDirection(Vector2.left);
    if(transform.position.x<leftCorner.position.x)
        ChangeDirection(Vector2.right);

    _rigidbody.velocity = new Vector2( x: runSpeed * directionVector.x * Time.deltaTime,  y: -2);
}
```

Рисунок 2.38 – Метод пересування лицаря з мечем.

Атакує мечем в ближньому бою

```
2+1 usages  Roflanpepe
protected override void StartAttackPlayer()
{
    base.StartAttackPlayer();
    Attacking = true;

    exclamationPoint.gameObject.SetActive(false);

    tokenSource?.Cancel();
    tokenSource?.Dispose();
    tokenSource = null;

    enemyAi.EnemyAnimator.Anim.SetTrigger(enemyAi.EnemyAnimator.AttackTriggerKey);
}
```

Рисунок 2.39 – Метод атаки лицаря з мечем.

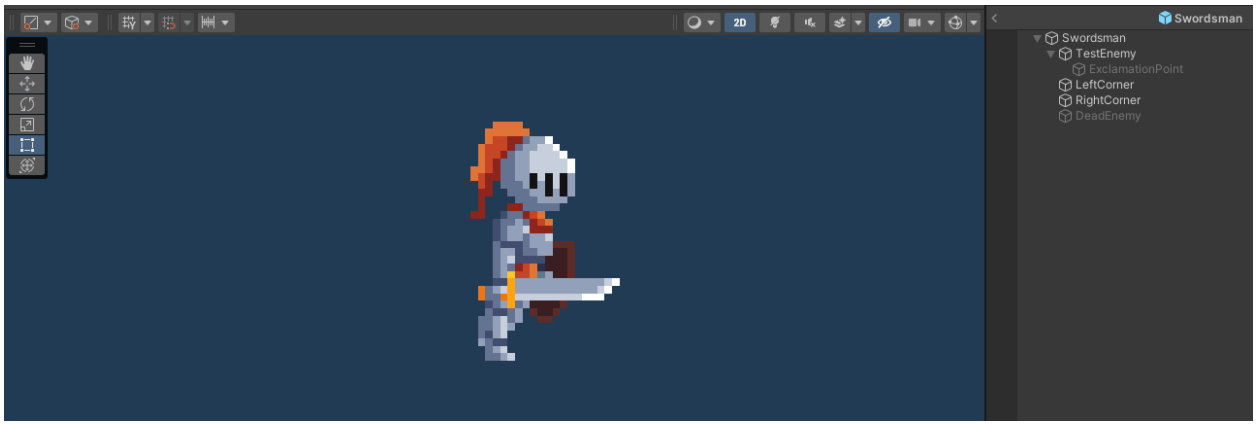

Рисунок 2.40 Об’єкт ворогу з мечем

2.5.2 Ворог з дистанційною атакою

Ворог з дистанційною атакою є червоним колом, що може літати (Рис 2.41 - 43).

```
Frequently called  0+3 usages  Roflanpepe
public override void Movement()
{
    base.Movement();
    if (Vector2.Distance((Vector2) a: transform.position, (Vector2) b: firstPosition.position) < 1f)
        ChangeDirection((Vector2)(secondPosition.position - transform.position).normalized);
    else if (Vector2.Distance((Vector2) a: transform.position, (Vector2) b: secondPosition.position) < 1f)
        ChangeDirection((Vector2)(firstPosition.position - transform.position).normalized);

    _rigidbody2D.velocity = new Vector2(x: flySpeed * directionVector.x * Time.deltaTime,
        y: flySpeed * directionVector.y * Time.deltaTime);
}
```

Рисунок 2.41 – Метод пересування ворогу з дистанційною атакою.

Ворог може атакувати дистанційно, за допомогою снарядів

```
2 usages  Roflanpepe
private void AttackPlayer()
{
    var playerCollider = GetPlayerCollider();
    if (playerCollider == null)
        return;

    ShootProjectile(projectilePool, transform,
        (Vector2) direction: (playerCollider.transform.position - transform.position));
    OnShoot?.Invoke();
}
```

2.42 – метод дистанційної атаки.

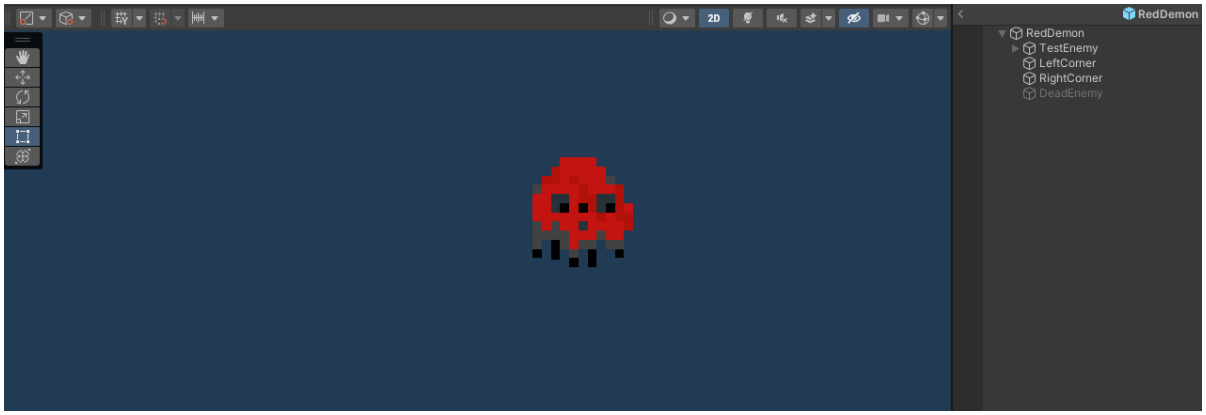

Рисунок 2.43 – Об’єкт ворогу з дистанційною атакою.

2.6 Режим нескінченного бігу

Режим нескінченного бігу є основним режимом в цьому проєкті, основною метою якого є пробігти якомога більше, уникаючи перешкод та пасток. Ігровий рівень, в цьому режимі, рандомно генерується з заздалегіть створених частинок рівня.

2.6.1 Частинки рівня

Кожна частинка рівня складається з батьківського об’єкта з компонентом LevelPart на ньому, LevelPart містить інформацію про ID частинки рівня та позиції де ця частинка рівня закінчується (Рис 2.44).

Рисунок 2.44 – Поля класу LevelPart.

Частинки рівня можуть мати об’єкти з компонентом EntityRandomSpawner (Рис 2.45) на них, що дозволяє рандомно генерувати сутності (Рис 2.46). Кожна сутність має свій шанс спавну, щоб більш цінні бонуси або складні перешкоди – з’являлися не так часто.

```
61 asset usages  Roflanpepe
public class EntityRandomSpawner: MonoBehaviour
{
    [SerializeField] private Spawnable[] spawnables;  Serializable

    Event function  Roflanpepe
    private void OnEnable()
    {
        SpawnRandomObject();
    }

    1 usage  Roflanpepe
    private void SpawnRandomObject()
    {
        float totalWeight = 0f;

        foreach (Spawnable spawnable in spawnables)
        {
            totalWeight += spawnable.spawnWeight;
            spawnable.prefab.SetActive(false);
        }

        int randomValue = Random.Range(0, Convert.ToInt32(totalWeight));
        float cumulativeWeight = 0f;

        foreach (Spawnable spawnable in spawnables)
        {
            cumulativeWeight += spawnable.spawnWeight;

            if (randomValue <= cumulativeWeight)
            {
                // Instantiate(spawnable.prefab, transform.position, Quaternion.identity);
                spawnable.prefab.SetActive(true);
                return;
            }
        }
    }
}
```

Рисунок 2.45 – Клас EntityRandomSpawner.

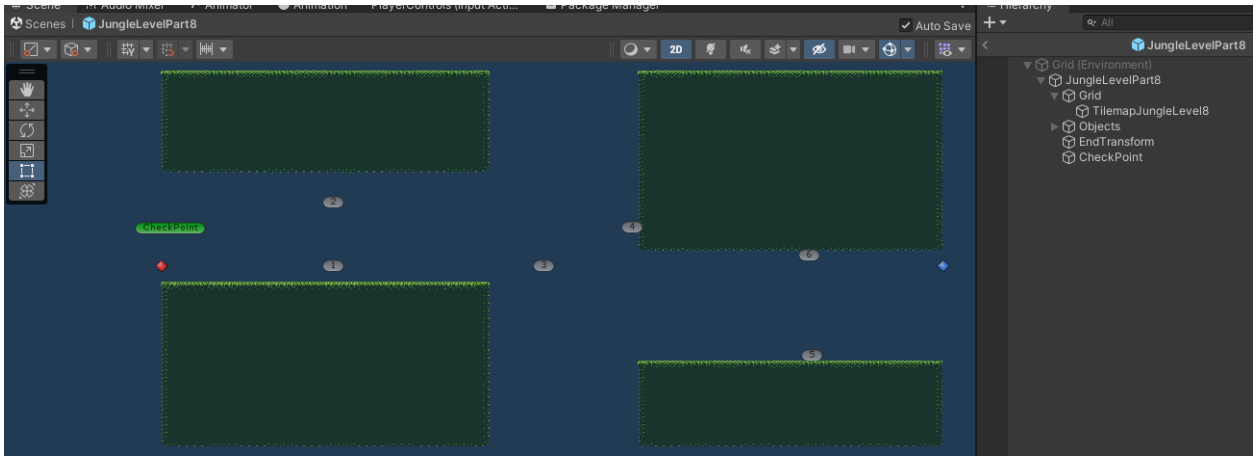

Рисунок 2.46 – Частинка рівня, цифрами показані місця з об’єктами EntityRandomSpawner.

Кожна частинка рівня має компонент LevelPartDisabler, що відключає частинку рівня, якщо вона вже пройдена гравцем (Рис 2.47).

```
1 usage  Roflanpepe
private void TryDisableLevelParts()
{
    if (!DeathWall.deathWall.CanDisableLevelParts)
        return;
    if (transform.position.x + disableDistance < DeathWall.deathWall.transform.position.x)
    {
        if (isHasCheckPoint)
        {
            var LeftCheckPoints :IEnumerable<CheckPoint> = FindAllActiveCheckPoints().Where(
                checkPoint => checkPoint.transform.position.x < player.transform.position.x);

            if(LeftCheckPoints.Count() > 3)
                gameObject.SetActive(false);
        }
        else
        {
            gameObject.SetActive(false);
        }
    }
}
```

Рисунок 2.47 – метод що перевіряє чи треба відключити частинку рівня.

2.6.2 Створення рівня з режимом нескінченного бігу

Для створення рівня з режимом нескінченного бігу створимо LevelGenerator, що буде перевіряти дистанцію між головним героєм і останнього кінця частинки рівня – якщо ця дистанція менше ніж дистанція для спавну буде відбуватися розміщення нової частинки рівня (Рис 2.48). Таким чином рівень буде постійно створюватися далі.

```
Event function  Roflanpepe
private void FixedUpdate()
{
    if(player==null)
        return;

    if (Vector3.Distance( a: player.position,  b: endTransform.position) < distanceSpawnLevelPart)
    {
        // SpawnLevelPart(Random.Range(1, levelPartsCount));

        levelPartSpawner.Spawn(SpawnLevelPart);
    }
}
```

Рисунок 2.48 – Перевірка чи необхідне створення нової частинки рівня.

Частинка рівня, що має з’явитися на рівні може бути конкретною частинкою рівня, або бути вибрана випадково. Для цього використовується інтерфейс ILevelPartSpawner (Рис 2.49).

```
public interface ILevelPartSpawner
{
    Frequently called  1 usage  2 implementations  Roflanpepe
    public void Spawn(LevelPartSpawnAction action){}
}
```

Рисунок 2.49 – Інтерфейс ILevelPartSpawner

Конкретний спавнер використовується для створення границь рівня (Рис 2.50).

```
public delegate void LevelPartSpawnAction(int id);

1 asset usage  Roflanpepe
public class ConcreteSpawner: MonoBehaviour, ILevelPartSpawner
{
    [SerializeField] private int idToSpawn;  Unchanged

    private LevelPartSpawnAction levelPartCallback;

    Frequently called  0+1 usages  Roflanpepe
    public void Spawn(LevelPartSpawnAction action)
    {
        action?.Invoke(idToSpawn);
    }
}
```

Рисунок 2.50 – Клас ConcreteSpawner.

Випадковий спавнер використовується для створення частинок рівня (Рис 2.51).

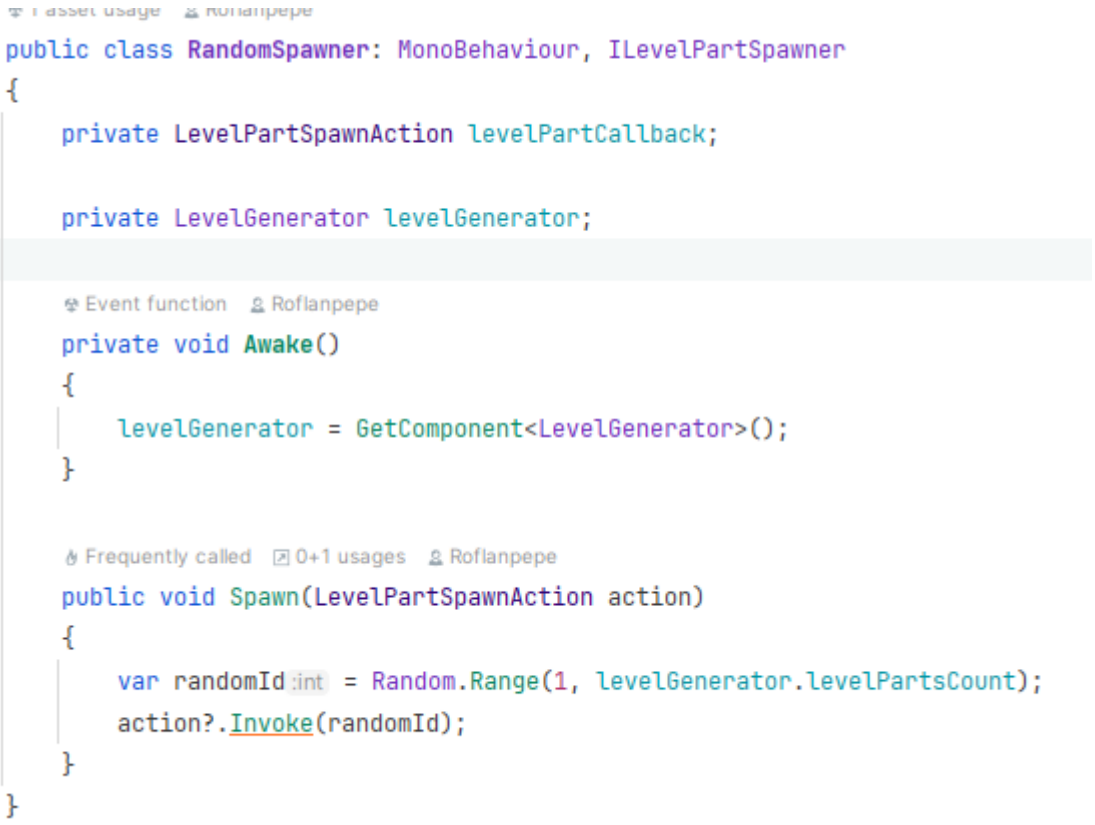

Рисунок 2.51 – клас RandomSpawner.

2.7 Система збереження

Система збереження в відеоіграх є однією з найважливіших складових, що визначають комфорт та задоволення гравців від процесу гри. Вона дозволяє гравцям зберігати прогрес та продовжувати гру з певного моменту, забезпечуючи безперервність ігрового досвіду. Одним з основних завдань системи збереження є можливість продовжити гру з того місця, де гравець зупинився. Це особливо важливо в іграх з довгим часом проходження та складними рівнями. Завдяки системи збереження гравці можуть повертатися до гри у зручний для них час, не турбуючись про втрату прогресу.

Для створення системи збереження створимо клас GameData, що буде зберігати всі данні, що необхідні для системи збереження (Рис 2.52). Та клас

DataPersistenceManager, що відповідає за виклики збереження та завантаження даних.

```
[System.Serializable]
31 usages  Roflanpepe  2 exposing APIs
public class GameData
{
    public int CoinsCount;  Serializable
    // public SpriteLibraryAsset HeroSpriteLibrary;
    public int HeroSpriteLibraryID;  Serializable

    public SerializableDictionary<string, bool> LevelPassed;  Serializable
    public string levelNeedToPass;  Serializable
    public SerializableDictionary<int, bool> PurchasedSkins;  Serializable
    public int PurchasedSkinsCount;  Serializable

    //Settings
    public float MasterVolume;  Serializable
    public float SFXVolume;  Serializable
    public float MusicVolume;  Serializable

    //Achievement
    public bool IsBeginnerRunnerAlreadyComplited;  Serializable

    Frequently called  3 usages  Roflanpepe
    public GameData()
    {
        CoinsCount = 0;
        HeroSpriteLibraryID = 0;
        LevelPassed = new SerializableDictionary<string, bool>();
        levelNeedToPass = "1";
        PurchasedSkins = new SerializableDictionary<int, bool>();
        PurchasedSkinsCount = 1;
        PurchasedSkins.Add(0, true); // add default skin
        //Settings
        MasterVolume = 1f;
        SFXVolume = 1f;
        MusicVolume = 1f;
        //Achievement
        IsBeginnerRunnerAlreadyComplited = false;
    }
}
```

Рисунок 2.52 – клас GameData.

Якщо гравець не аутентифікований в Google Play Games – зберігаємо і завантажуємо дані з пам’яті пристрою. Для збереження інформації клас GameData серіалізуємо в Json-файл (Рис 2.53), а для завантаження даних десеріалізуємо дані з Json-файлу до класу GameData.

```
Frequently called  2 usages  Roflanpepe
public void Save(GameData data)
{
    string fullPath = Path.Combine(dataDirPath, dataFileName);
    try
    {
        //Create directory if it dosnt already exist
        Directory.CreateDirectory(Path.GetDirectoryName(fullPath));
        // serialize the C# game AnimationData object into Json
        string dataToStore = JsonUtility.ToJson(data, prettyPrint: true);

        if (useEncryption)
            dataToStore = EncryptDecrypt(dataToStore);

        // write the serialized AnimationData to the file
        using (FileStream stream = new FileStream(fullPath, FileMode.Create))
        {
            using (StreamWriter writer = new StreamWriter(stream))
            {
                writer.Write(dataToStore);
            }
        }
    }
    catch (Exception e)
    {
        Debug.LogError( message: "Error occured when trying to save data to file: " + fullPath + "\n" + e);
    }
}
```

Рисунок 2.53 – Збереження даних до пам’яті пристрою.

Якщо ж гравець аутентифікований в Google Play Games – зберігаємо та завантажуємо файли з хмари, що надає Google [17] (Рис 2.54).

```
private void OnSavedGameOpened(SavedGameRequestStatus status, ISavedGameMetadata meta) {
    if (status == SavedGameRequestStatus.Success) {
        if (isSaving)
        {
            // CloudSaveGameUI.Instance.LogText.text += "Status successful, attempting to save...";
            //convert to byte array
            byte[] myData = System.Text.ASCIIEncoding.ASCII.GetBytes( s: GetSaveString());

            //metadata
            SavedGameMetadataUpdate updateForMetadata = new SavedGameMetadataUpdate.Builder()
                .WithUpdatedDescription("Appdate game data at: " + DateTime.Now.ToString()).Build();

            //saving
            ((PlayGamesPlatform)Social.Active).SavedGame.CommitUpdate(meta, updateForMetadata, myData, SaveCallback);
        }
        else
        {
            // CloudSaveGameUI.Instance.LogText.text += "Status successful, attempting to load...";
            // //loading
            ((PlayGamesPlatform)Social.Active).SavedGame.ReadBinaryData(meta, LoadGameCallback);
        }
    } else {
        Debug.LogError( message: "Error when try Open SavedGame");
    }
}
```

Рисунок 2.54 – Метод збереження та завантаження даних з хмари.

2.8 Музика та звуки

Музика та звукові ефекти є одними з ключових компонентів відеоігор, що значно впливають на загальний ігровий досвід. Вони здатні створювати емоційний фон, занурювати гравця у віртуальний світ та підсилювати взаємодію з ігровим середовищем. Приємні мелодії та цікаві звукові ефекти роблять гру більш привабливою та стимулюють гравців повертатися до неї знову і знову. Наприклад, весела та енергійна музика може мотивувати гравців на активні дії, тоді як заспокійливі мелодії сприяють тривалому та невимушеному ігровому процесу.

Музика та звуки для цього проєкту були взяті з Unity AssetStore [18], та є безшовними. Для програвання музики створюємо клас MusicManager, що буде містити масив класу SoundTrack (Рис 2.55), що містить назву аудіо кліпу та його шанси на програвання, та повторне програвання.

```
[Serializable]
3 usages  Roflanpepe
public class SoundTrack
{
    public string AudioClipName;  Serializable

    [Space( height: 20f)]
    [Range(0f, 1f)] public float SongChance = 0.5f;  Serializable
    [Range(0f, 1f)] public float SongReplayChance = 0.5f;  Serializable
}
```

Рисунок 2.55 – Клас SoundTrack

Для програвання звуків створюємо SoundFXHandler, що містить посилання на аудіо кліпи та програє їх (Рис 2.56).

```
public class SoundFXHandler :MonoBehaviour
{
    [SerializeField] private AudioClip[] SFXClips;  Serializable


    25+ asset usages  Roflanpepe
    public void Play()
    {
        PlaySoundFX(SFXClips);
    }


    1 usage  Roflanpepe
    private void PlaySoundFX(AudioClip[] audioClips)
    {
        if (audioClips == null || audioClips.Length == 0)
        {
            Debug.LogWarning( message: "Audio clip not assigned/ GameObject:" + gameObject.name);
            return;
        }
        SoundFxManager.instance.PlaySoundFxClip(audioClips,  volume: 1f);
    }
}
```

Рисунок 2.56 – Клас SoundFXHandler.

2.9 Користувацький інтерфейс

Користувацький інтерфейс (UI) – це сукупність графічних елементів інтерфейсу, що дозволяють гравцям взаємодіяти з грою. Від якісного та добре продуманого інтерфейсу залежить не лише зручність використання, але й загальне задоволення від взаємодії з продуктом.

Для створення користувацького інтерфейсу Unity має компонент Canvas, що представляє абстрактний простір, в якому розміщується та візуалізується інтерфейс користувача [19].

2.9.1 Головне меню відеогри

Головне меню в відеоіграх виконує декілька ключових функцій, забезпечуючи зручність та ефективність взаємодії гравця з грою. Воно слугує початковою точкою для доступу до різних частин гри та налаштувань.

В цьому проєкті головне меню виступає сценою з головним персонажем та панеллю вибору інших меню (Рис 2.57). Також в головному меню можна змінити налаштування гри, та продивитися досягнення та табло лідерів.

Рисунок 2.57 – Головне меню гри

Кнопка Levels відкриває сцену з вибором рівнів для проходження (Рис 2.58)

Рисунок 2.58 – Сцена з вибором рівнів для проходження

Кнопка Endless run переносить гравця в режим безкінечного бігу. Кнопка Skins відкриває меню вибору персонажів (Рис 2.59). Гравець може придбати та використовувати цих персонажів.

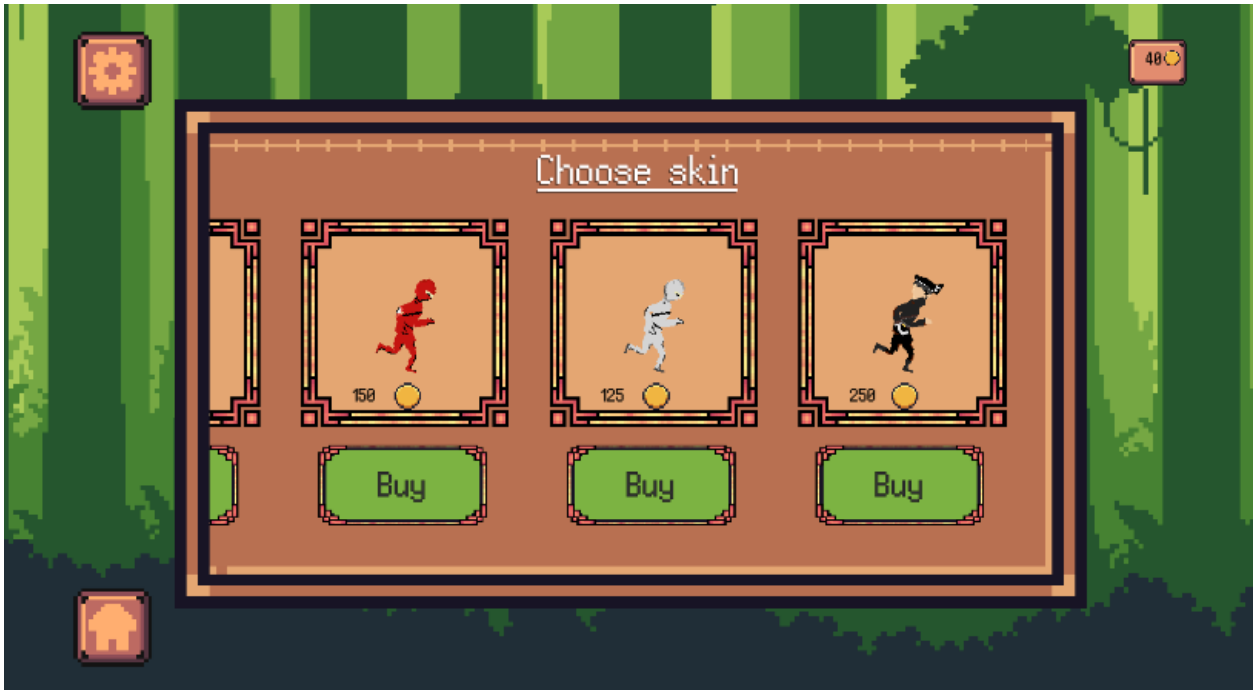

Рисунок 2.59 – Меню вибору персонажів

2.9.2 Меню налаштувань

Меню налаштувань надає можливість кожному користувачеві змінювати налаштування в грі. В данному проєкті меню налаштувань дає змогу змінювати гучність музики та звуків, змінювати розмір камери та змінювати кількість кадрів в секунду (Рис 2.60)

Рисунок 2.60 – Меню налаштувань, що відображається в головному меню гри

2.10 Сервіси Google Play

[20].

Для входу користувача в його обліковий запис Google Play Games створюємо клас Authentication, що при старті відеогри буде намагатися автоматично аутентифікувати гравця (Рис 2.61)

```
private void Start()
{
    if (!Social.localUser.authenticated)
    {
        PlayGamesPlatform.DebugLogEnabled = true;
        PlayGamesPlatform.Activate();
        PlayGamesPlatform.Instance.Authenticate(ProcessAuthentication);
    }

}
```

Рис 2.61 – Автоматична аутентифікаця гравця.

При помилці автоматичної аутентифікації гравець зможе спробувати зайти в свій обліковий засіб вручну (Рис 2.62).

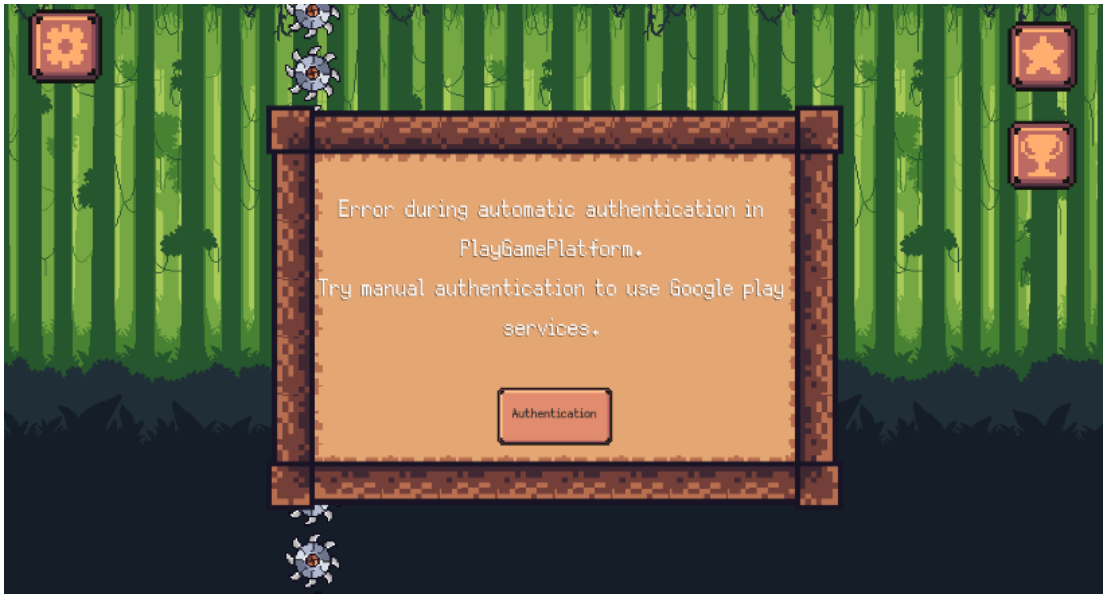

Рисунок 2.62 – Повідомлення про помилку під час автоматичної аутентифікації

Аутентифікація в Google Play Games дозволяє використовувати хмарні збереження прогресу гравця, мати доступ до досягнень та таблиці лідерів.

Досягнення у відеоіграх є важливим елементом, який значно збагачує ігровий досвід та стимулює гравців до активного і тривалого ігрового процесу, досягнення сприяють підвищенню реіграбельності гри, заохочуючи гравців проходити гру повторно, щоб отримати всі досягнення.

Для створення та додавання досягнення необхідно створити досягнення в Google Play Console та використовуючи згенеровані ідентифікатори додаємо їх в гру. Створюємо клас Achievement, його публічні методи будуть викликатися тоді – коли досягнення вважається отриманим (Рис 2.63).

```
public void BeatThemAll()
{
    PlayGamesPlatform.Instance.IncrementAchievement("CgkI5fOH1boJEAIQAw", steps: 1, callback: (bool success) =>
    {

    });
}
```

Рисунок 2.63 – Публічний метод класу Achivement.

Таблиця лідерів – це список гравців, ранжований за досягненнями, очками, рівнями або іншими показниками успіху в грі. Таблиця дозволяє гравцям порівнювати свої результати з результатами інших гравців. Змагання з іншими гравцями підвищує інтерес до гри і сприяє утриманню гравців, оскільки вони намагаються піднятися на вершину рейтингу або втримати свої позиції.

В цьому проєкті ранжування гравців відбувається за очками в режимі безкінечного бігу (Рис 2.64).

```
private void FixedUpdate()
{
    if(!isUpdateScore)
        return;
    if (playerTransform.position.x > currentMaxPosition+1f)
    {
        score++;
        scoreText.text ="Score: " + score.ToString();
        currentMaxPosition = (int)playerTransform.position.x;

        if(!isBeginnerRunnerAlreadyCompleted && score >= 2000)
            Achievement.Instance.BeginnerRunner();
    }
}
```

Рисунок 2.64 – Нарахування очок в режимі безкінечного бігу.

Отже, чим більше гравець пробіжить – тим більше очок він отримує. Отримуючи більше очок гравець збільшує своє місце в списку лідерів, що мотивує до подальшої гри.

2.11 Тестування відеогри

Тестування є важливою частиною процесу розробки, оскільки воно забезпечує збільшення якості кінцевого продукту та допомагає виявити програмні помилки, що можуть призвести до неправильного функціонування гри або її аварійного завершення. Регулярне тестування дозволяє переконатися, що відеогра працює стабільно на різних пристроях, особливо, якщо брати до уваги

кількість різноманітних мобільних девайсів. Також тестування допомагає перевірити ігрові механіки, це включає перевірку балансу складності, фізики руху персонажів, перевірка системи введення.

Для можливості опублікувати гру в Google Play необхідно провести закрите тестування, з залученням зовнішніх користувачів [21]. Закрите тестування має проходити більше двох тижнів, з залученням, як мінімум, 20 користувачів. Користувачі можуть залишати коментарі, та надсилати зворотній зв'язок. Для полегшення вияву проблем у тестерів, був створений клас LogRecordService. При виході з відеогри створюється файл логування з помилками (Рис 2.65)

```
//Called when there is an exception
2 usages  Roflanpepe
void LogCallback(string condition, string stackTrace, LogType type)
{
    //Create new Log
    Logs logInfo = new Logs(condition, stackTrace, type, DateTime.Now.ToString( format: "yyyy-MM-ddTHH:mm:sszzz"));

    //Add it to the List
    logs.logInfoList.Add(logInfo);
}


//Save log  when focus is lost
Event function  Roflanpepe
void OnApplicationFocus(bool hasFocus)
{
    if (!hasFocus)
    {
        //Save
        if (enableSave)
            dataSaver.Save(logs);
    }
}
```

Рисунок 2.65 – методи створення інформації про помилки та виклику створення файлу логування.

Файл зберігається в внутрішній пам'яті пристрою і тестер може відправити його електронною поштою, для полегшення знаходження помилки.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи відбувся аналіз ігрової індустрії. Було виявлено, що на даний момент в світі налічується більше 3 мільярдів гравців. Були проаналізовані жанри відеоігор, виявлені причини популярності жанру раннер та проаналізовані відомі проекти у жанрі раннер, такі як “Doodle Jump”, “ Subway Surfers”, “ Jetpack Joyride”, “Crossy Road”. Виявлено їх основні функціональні можливості, це дозволило визначити найкращі практики, які можуть бути використані при розробці власного проєкту.

Було досліджено основні програмні середовища для розробки відеоігор, такі як Unity, Unreal Engine та Godot Engine. На основі аналізу був вибраний Unity, з огляду на його потужні інструменти для розробки 2D-платформенних відеоігор, ліцензію, широку спільноту розробників та доступність численних ресурсів і навчальних матеріалів. Проведено планування проєкту, виділені вимоги до проєкту, визначений перелік користувачів, для яких створюється відеогра та визначені ризики при розробці проєкту.

На основі теоретичних досліджень було розроблено та реалізовано 2D-платформенну гру в жанрі runner на базі програмного середовища Unity. Проєкт включає декілька режимів гри, такі як проходження рівнів та безкінечний біг. Були реалізовані основні елементи геймплею, такі як подолання перешкод та ворогів, збирання винагород у вигляді монет та розблокування нових персонажів. Були додані соціальні функції, такі як досягнення та список лідерів. Відеогра пройшла закрите тестування в Google Play Console, з залученням зовнішніх користувачів. На основі отриманих відгуків було проведено оптимізації ігрових механік та усунено виявлені помилки.

В результаті виконання дипломної роботи було створено повнофункціональний ігровий продукт, готовий до запуску на ринку. Гра має потенціал для подальшого розвитку, зокрема через додавання нових рівнів, персонажів та нових типів винагород.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Howarth J. How Many Games Are There? (New 2024 Statistics). URL: https://explodingtopics.com/blog/number-of-gamers (дата звернення: 10.05.2024).

2. Anderson N. The Popularity of Video Games: Entertainment and Interactive Storytelling. URL: https://medium.com/@noah.anderson_84365/the-popularity-of-video-games-entertainment-and-interactive-storytelling-0c1671365797#:~:text=One%20of%20the%20key%20reasons,story%20rather%20than%20passive%20observers (дата звернення: 10.05.2024).

3. Holoskova D. Mobile Gaming Vs PC Gaming: Key Differences. URL: https://whimsygames.co/blog/mobile-gaming-vs-pc-gaming-key-differences/#:~:text=Moving%20on%20to%20the%20revenue,ranked%20second%20with%20%2451%20billion (дата звернення: 11.05.2024).

4. Crawford C. The Art of Computer Game Design. Берклі, 1984. 113 с.

5. Clearwater D. What Defines Video Game Genre? Thinking about Genre Study after the Great Divide. *The Journal of the Canadian Game Studies Association*. Vol 5(8). С. 29-49.

6. Game Engine Popularity in 2024. URL: https://gamefromscratch.com/game-engine-popularity-in-2024/ (дата звернення: 12.05.2024).

7. Mono. URL: https://www.mono-project.com/ (дата звернення: 12.05.2024).

8. Unity Documentation. URL: https://docs.unity3d.com/ru/530/Manual/ (дата звернення: 12.05.2024).

9. Unreal Engine Marketplace. URL: https://www.unrealengine.com/marketplace/en-US/store (дата звернення: 13.05.2024).

10. Jungle Asset Pack. URL: https://yan-san.itch.io/platformer-free-asset-1-jungle-pixelart (дата звернення: 13.05.2024).

11. Knight 2D Pixel Art. URL: https://xzany.itch.io/free-knight-2d-pixel-art (дата звернення: 14.05.2024).

12. Free Trap Platformer. URL: https://bdragon1727.itch.io/free-trap-platformer (дата звернення: 14.05.2024).

13. Pixel UI Button Icon Platformer. URL: https://bdragon1727.itch.io/pixel-ui-button-icon-platformer (дата звернення: 14.05.2024).

14. Animator Controller Asset. URL: https://docs.unity3d.com/2021.3/Documentation/Manual/Animator.html (дата звернення: 15.05.2024).

15. Skinning Editor module. URL: https://docs.unity3d.com/Packages/com.unity.2d.animation@2.2/manual/SkinningEditor.html (дата звернення: 15.05.2024).

16. Input Action Asset. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@0.9/manual/ActionAssets.html (дата звернення: 15.05.2024).

17. Play Games Services, Saved Games. URL: https://developers.google.com/games/services/common/concepts/savedgames (дата звернення: 16.05.2024).

18. Seamless Loop and Short Music. URL: https://assetstore.unity.com/packages/audio/music/seamless-loop-and-short-music-107732#description (дата звернення: 16.05.2026).

19. Unity Canvas Component. URL: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/class-Canvas.html (дата звернення: 17.05.2026)

20. Google Play Services. URL: https://developer.android.com/distribute/play-services (дата звернення: 17.05.2026).

21. Test your app or game. URL: https://playacademy.exceedlms.com/student/path/4915%22?utm_source=console&utm_medium=banner&utm_campaign=mktpages&use_locale=true%22

22. Моркун Н. В., Маринич І. А. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 151 “Автоматизація

та комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2019. 50 с.

23. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

24. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

25. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).

26. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).