МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ПРИХОДСЬКИЙ ГЕОРГІЙ СЕРГІЙОВИЧ**

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

# РОЗРОБКА КОМП'ЮТЕРНОЇ ГРИ В ЖАНРІ ПЛАТФОРМЕР

Спеціальність 122 Комп'ютерні науки

**Кваліфікаційна (бакалаврська) робота**

Керівник:
Надія ПОТАПОВА, доцент кафедри
інформаційних технологій,
к. е. н., доцент

_____

Оцінка: _____ / _____ / _____
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця - 2026

# АНОТАЦІЯ

**Приходський Г. С. Розробка комп'ютерної гри в жанрі платформер.** Спеціальність 122 "Комп'ютерні науки", освітня програма "Комп'ютерні науки". Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано процес розробки комп'ютерних ігор у жанрі платформер, визначено основні принципи їх побудови та функціонування. Висвітлено основні аспекти створення ігрових елементів. За допомогою ігрового рушія Godot та його технологій було розроблено повноцінний ігровий продукт підвищеної складності у жанрі 2D-платформер. Створена гра об'єднує в собі всі необхідні для сучасного ігрового проєкту системи, компоненти та механіки взаємодії.

**Ключові слова:** комп'ютерна гра, 2D-платформер, технології Godot Engine, ігрові механіки, ігрова логіка, ігрові системи, ігрові персонажі.

87 ст., 76 рис., 60 джерел.

# ABSTRACT

**Prykhodskyi H. S. Development of a Computer Game in the Platformer Genre.** Specialty 122 "Computer Science", educational program "Computer Science". Vasyl Stus Donetsk National University, Vinnitsya 2026.

In the qualification (bachelor's) thesis, the process of computer game development in the platformer genre is researched and analyzed, and the main principles of their construction and functioning are determined. Key aspects of creating game elements are discussed. With the help of the Godot game engine and its technologies, a full-fledged game product of increased difficulty in the 2D platformer genre was developed. The created game combines all the systems, components and interaction mechanics necessary for a modern game project.

**Keywords:** computer game, 2D platformer, Godot Engine technologies, game mechanics, game logic, game systems, game characters.

# ЗМІСТ

# ВСТУП

Сьогодні ігрова індустрія є однією з найдинамічніших сфер цифрових технологій, що стрімко розвивається та впливає на культуру, освіту та розваги. Комп'ютерні ігри не лише стали популярним видом розваги, а й слугують засобом для розвитку творчих навичок, логічного мислення та програмування. У цьому контексті розробка власної гри є цікавим і складним завданням, що вимагає ретельного планування, знання інструментів розробки та розуміння особливостей жанру.

Одним із найбільш популярних і водночас доступних жанрів у сфері ігрової розробки є платформер – жанр комп'ютерних ігор, у якому основою ігрового процесу є переміщення персонажа між платформами, подолання перешкод та взаємодія з ігровим середовищем. Завдяки простоті базових механік, широким можливостям для реалізації різноманітних ігрових систем та відносно невеликим вимогам до ресурсів платформери залишаються актуальними як серед незалежних розробників, так і серед великих ігрових студій.

Створення сучасної комп'ютерної гри потребує технічних і творчих рішень, які охоплюють усі етапи розробки – від створення концепту до повної фіналізації проєкту та його виходу на ігровий ринок. Одним із сучасних інструментів для створення 2D-ігор є ігровий рушій Godot Engine, який надає зручне середовище розробки, підтримку двовимірної графіки, систему сцен і вузлів, а також вбудовану мову програмування GDScript для реалізації будь-яких аспектів ігрової логіки.

*Метою бакалаврської роботи* є розробка комп'ютерної гри у жанрі 2D-платформер із використанням ігрового рушія Godot Engine, яка являтиме собою повноцінний готовий ігровий продукт та міститиме всі основні компоненти сучасної відеогри.

*Об'єктом дослідження* є процес розробки комп'ютерної гри у жанрі 2D-платформер із використанням сучасних інструментів ігрової розробки.

*Предметом дослідження* є методи, алгоритмічні підходи та програмні засоби, що застосовуються під час створення 2D-ігор.

*Для досягнення поставленої мети було сформульовано такі завдання:*

1. дослідити теоретичні аспекти розробки відеоігор, визначити ключові особливості жанру 2D-платформер, платформ для цифрової дистрибуції відеоігор, способи їх поширення, а також принципи проєктування рівнів, ігрової прогресії та балансу складності;

2. дослідити технології Godot Engine та обґрунтувати його використання як середовища розробки для 2D-ігор;

3. створити головного персонажа і реалізувати систему керування ним, фізичну взаємодію та базові механіки руху;

4. створити ігрове середовище та систему рівнів;

5. реалізувати ігрові механіки, інтерактивні об’єкти та систему взаємодії між ними;

6. розробити користувацький інтерфейс, аудіосупровід, сюжетну складову, а також виконати тестування та експорт готової гри.

*Структура роботи складається* зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі досліджуються теоретичні аспекти розробки комп’ютерних ігор жанру платформер, проаналізовано існуючі ігри та сформовано концепцію власного проєкту. У другому розділі наведено аналітичне обґрунтування вибору методів та інструментів розробки відеоігор, принципи проєктування рівнів та ігрової прогресії, а також особливості використання Godot Engine для створення 2D-ігор. У третьому розділі наведено програмну реалізацію готового ігрового проєкту, що включає розробку персонажів, побудову ігрового середовища, реалізацію ігрових механік, інтерактивних об’єктів, користувацького інтерфейсу, загальної атмосфери та інших важливих елементів сучасної відеогри.

*Практичне значення* роботи полягає у створенні комп’ютерної гри у жанрі 2D-платформер, яка може використовуватись як приклад реалізації основних механік жанру та демонстрація можливостей рушія Godot Engine під час

розробки 2D-ігор. Результати роботи можуть бути корисними при навчанні розробників у сфері геймдеву та всіх, хто цікавиться створенням комп'ютерних ігор.

*Апробація результатів дослідження відбулась на* VII Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології» ( м. Вінниця, 15 травня 2026 р.).

# РОЗДІЛ 1
# ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР ЖАНРУ ПЛАТФОРМЕР

## 1.1 Теоретичні основи сучасної ігрової розробки та цифрової дистрибуції

Розробка сучасних відеоігор є складним процесом, який поєднує в собі велику кількість різноманітних аспектів. Залежно від жанру та технічної реалізації ігрові проєкти можуть суттєво відрізнятися за структурою, способом взаємодії з користувачем та особливостями побудови ігрового світу.

Щоб краще розуміти сутність відеоігор та принципи їх побудови, необхідно розглянути основні терміни та поняття, які використовуються у сфері ігрової розробки:

- Відеогра – це цифровий продукт, основна функція якого полягає у створенні віртуального середовища для інтерактивної взаємодії з гравцем. Вона передбачає використання графіки, звуку, сценаріїв, правил і механік, що забезпечують участь гравця у подіях в ігровому світі.
- Ігрова механіка – це набір правил, за якими відбувається ігровий процес. Це може бути стрільба, вирішення головоломок, взаємодія з інтерактивними об'єктами, розвиток персонажа або проходження рівнів.
- Геймплей – досвід взаємодії гравця з грою, який формується в результаті поєднання графіки, звуку, механік, управління та сценарію.
- Геймдизайн – процес проєктування структури гри, ігрових механік, рівнів, системи прогресії та взаємодії з ігровим середовищем.
- Ігровий рушій – програмне забезпечення, що забезпечує базову функціональність для розробки гри: візуалізацію, програмування, анімації, звукове супроводження, систему колізій та обробку вводу користувача.
- Чекпоїнт – контрольна точка, з якої гравець може продовжити гру після поразки або перезапуску рівня.

- Квест – завдання, яке гравець повинен виконати для просування сюжетом або отримання винагороди.
- Інвентар – система зберігання предметів, які гравець може збирати, використовувати або змінювати в процесі проходження гри.
- Саундтрек – музичне оформлення гри, яке створює атмосферу та емоційний настрій, підтримує динаміку геймплею.
- Хітбокс – невидима область довкола об’єкта чи персонажа, яка використовується для визначення взаємодії між елементами гри, зокрема попадання ударів чи пострілів, збору предметів, активації бонусів та інших інтерактивних механік.
- NPC – персонаж, який керується ігровою системою, а не гравцем.
- HUD – елементи користувацького інтерфейсу, які відображають важливу ігрову інформацію, зокрема кількість життів, очок, ресурсів або карту рівня.
- Pixel Art – художній стиль двовимірної графіки, у якому зображення формується за допомогою окремих пікселів.
- Side Scroller – тип 2D-ігор із боковим видом камери, у яких персонаж переважно рухається по горизонталі.
- Asset – окремий ресурс гри, зокрема текстури, моделі, звуки, анімації та інші елементи проєкту.
- FPS – кількість кадрів, які відображаються за одну секунду, що впливає на плавність ігрового процесу [1; 2].

Відеоігри класифікують за різними критеріями, серед яких основними є тип взаємодії з гравцем, жанр, платформа, технічна реалізація та спосіб поширення. Класифікація цих критеріїв дозволяє краще аналізувати особливості різних ігрових проєктів та визначити їх цільову аудиторію.

За жанром відеоігри поділяються на:

- Екшени (Action) – ігри, орієнтовані на динамічний процес, швидку реакцію та активну взаємодію гравця з навколишнім середовищем.
- Платформери – жанр, у якому основою геймплею є переміщення

персонажа між платформами, подолання перешкод, використання стрибків та взаємодія з елементами рівня.

- Шутери – ігри, побудовані на використанні зброї та бойових взаємодіях між персонажами.
- Пригодницькі ігри – зосереджені на сюжеті, дослідженні світу, взаємодії з персонажами та виконанні різноманітних завдань.
- Рольові ігри (RPG) – дозволяють розвивати персонажа, покращувати його характеристикии та впливати на розвиток подій у грі.
- Стратегії – вимагають від гравця тактичного мислення, управління ресурсами та планування дій.
- Симулятори – імітують реальні або вигадані процеси, професії чи системи.
- Гоночні ігри – орієнтовані на керування транспортними засобами та участь у змаганнях.
- Спортивні ігри – відтворюють різні види спорту та спортивні змагання.
- Файтинги – базуються на бойових поєдинках між персонажами з використанням різних прийомів та комбінацій.
- Головоломки – спрямовані на розвиток логічного мислення та вирішення інтелектуальних завдань.
- Survival Horror – поєднують елементи виживання, обмежені ресурси та атмосферу напруження й страху.
- MMO-ігри – багатокористувацькі проєкти, у яких велика кількість гравців взаємодіє в єдиному ігровому світі [3; 4].

Залежно від платформи запуску відеоігри поділяються на:

- Комп’ютерні ігри – запускаються на персональних комп’ютерах і мають широкі можливості налаштування.
- Консольні ігри – працюють на спеціалізованих ігрових приставках.
- Мобільні ігри – призначені для смартфонів і планшетів.
- Браузерні ігри – запускаються безпосередньо у веббраузері.

- Хмарні ігри або сервіси – працюють на віддалених серверах та передають користувачу відеопотік. Хмарні ігри дозволяють користувачам з обмеженими можливостями або комп'ютерами грати в сучасні масштабні ігрові проєкти, де ігровий процес залежить від швидкості та стабільності інтернет-з'єднання.
- VR/AR ігри – використовують технології віртуальної та доповненої реальності [23].

За технічною реалізацією ігри поділяються на:

- 2D-ігри – використовують двовимірну графіку. У жанрі платформерів такі ігри зазвичай орієнтовані на рух по горизонталі або вертикалі.
- 3D-ігри – застосовують тривимірну графіку. У жанрі платформерів такі ігри дозволяють вільно переміщуватись у просторі.
- Гібридні ігри – поєднують елементи 2D та 3D-графіки або різні жанрові механіки.

Цифрові платформи поширення ігор також мають важливу роль в індустрії. Подібні сервіси дозволяють розробникам публікувати власні проєкти, а користувачам – купувати, завантажувати та оновлювати ігри через інтернет. Розподіл ринку цифрової дистрибуції відеоігор наведено на рисунку 1.1.

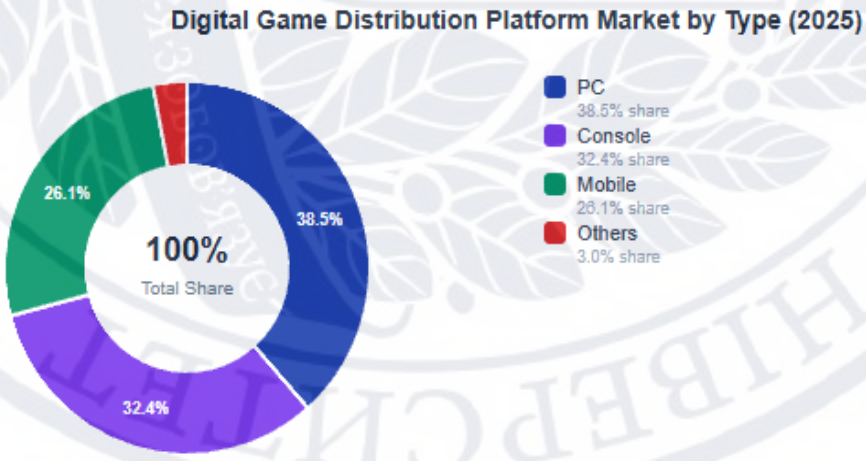


Рисунок 1.1 – Розподіл ринку цифрової дистрибуції відеоігор за типом платформ запуску у 2025 році [5]

Особливе місце серед цифрових платформ займає Steam (рис. 1.2), який є одним із найбільших сервісів поширення комп’ютерних ігор у світі та займає домінуючу позицію у своєму сегменті. Steam надає розробникам усі необхідні інструменти для інтеграції різноманітних онлайн-функцій у власний ігровий продукт та його подальший розвиток. Станом на 2025 рік Steam налічував поняд 132 мільйони активних користувачів щомісяця, а кількість одночасно активних користувачів перевищувала 37 мільйонів осіб [5]. Основними конкурентами Steam у сегменті цифрової дистрибуції комп’ютерних ігор є Epic Games Store та GOG, які також активно розвивають власні платформи поширення ігор.


Рисунок. 1.2 – Головна сторінка цифрової платформи Steam із каталогом популярних та нових відеоігор

Завдяки розвитку цифрових платформ навіть невеликі та незалежні команди розробників отримали можливість поширювати власні ігрові проєкти серед великої аудиторії користувачів без необхідності фізичного видання гри.

Незважаючи на активний розвиток тривимірної графіки, розробники 2D-ігор, зокрема жанру платформерів, продовжують вдосконалювати власні проєкти та створювати нові креативні ігри з унікальними особливостями, що дозволяє цьому напряму залишатися актуальним протягом багатьох років (рис. 1.3).

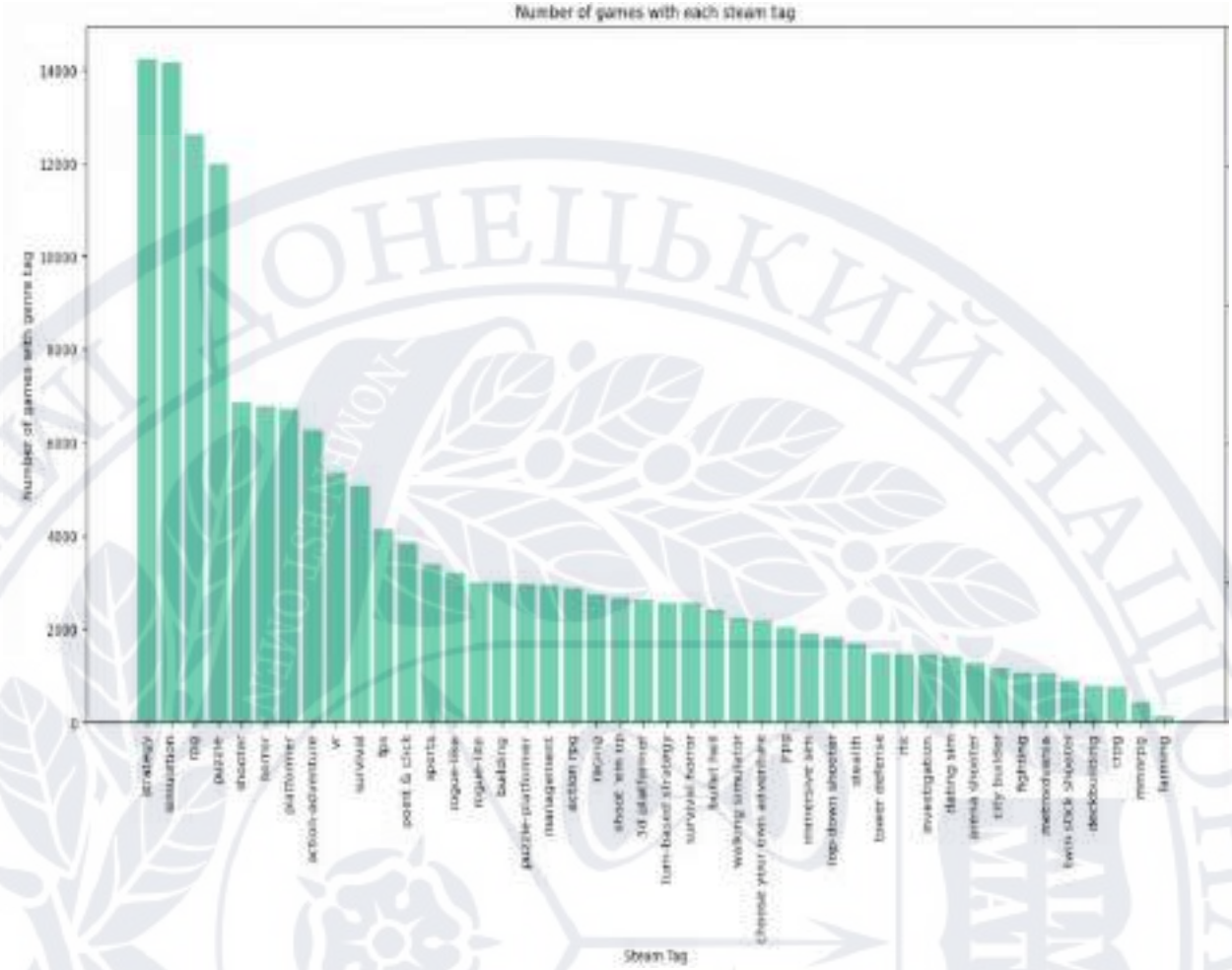


Рисунок 1.3 – Статистика кількості відеоігор за жанрами на цифровій платформі Steam [6; 7]

На основі статистики цифрової платформи Steam можна зробити висновок, що платформери й надалі залишаються одним із достатньо популярних напрямків серед сучасних відеоігор. Попри домінування жанрів Strategy, Simulation та RPG, ігри цього жанру продовжують займати помітне місце серед інших категорій на платформі Steam.

На наведеному графіку (рис. 1.4) представлено співвідношення між кількістю випущених ігор певного жанру та відсотком проєктів, які отримали понад 500 користувацьких відгуків на платформі Steam. Такий показник дозволяє оцінити поширеність жанру й рівень зацікавленості користувачів.

Згідно з наведеними статистичними даними, на платформі Steam представлено понад 6500 ігор жанру платформерів, проте лише близько 12%

відсотків таких проєктів отримали більше ніж 500 користувацьких відгуків. Таке співвідношення свідчить про високий рівень конкуренції серед ігор цього жанру та значну залежність популярності проєкту від якості реалізації, візуального стилю та унікальності.

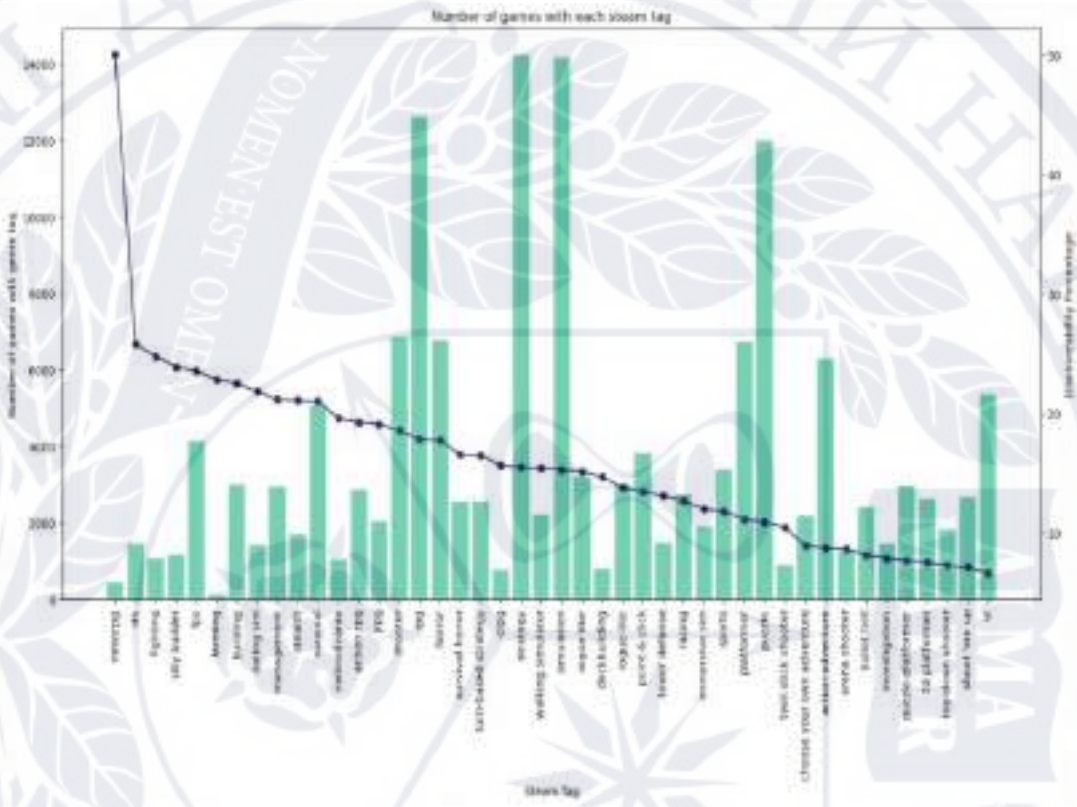

Рисунок 1.4 – Співвідношення кількості ігор та рівня популярності жанрів на цифровій платформі Steam [6; 7]

Отже, сучасна розробка відеоігор охоплює не лише створення програмної та візуальної складової проєкту, але й використання цифрових платформ поширення, системи взаємодії з користувачами та інструментів популяризації ігрового продукту. Успішність сучасного ігрового проєкту значною мірою залежить не лише від якості реалізації гри, а й від можливостей її поширення, підтримки спільноти користувачів та загальної доступності для гравців.

Ігрова індустрія є однією з найбільших і перспективних сфер цифрових технологій. Вона об’єднує мільйони гравців по всьому світу та охоплює різноманітні жанри, піджанри, платформи й бізнес-моделі. З розвитком сучасних

технологій і появою потужного програмного забезпечення для розробки ігор процес створення ігрових продуктів став доступним не лише для великих компаній, а й для незалежних розробників та ентузіастів [8].

За останнє десятиліття можна спостерігати значний розвиток інді-ігор, що зумовлено популяризацією платформ для цифрової дистрибуції, таких як Steam та Epic Games Store. Розвиток технологій розробки, таких як ігрові рушії, також вплинув на розвиток ігрової індустрії. Завдяки таким рушіям, як Unity, Unreal Engine та Godot, створення якісних ігор стало можливим навіть без великих фінансових ресурсів чи команд. Особливою популярністю користуються 2D-ігри, оскільки вони мають простішу технічну реалізацію, менші вимоги до апаратного забезпечення та широкі можливості для творчого вираження [9].

Один з найстаріших і найпопулярніших жанрів відеоігор – платформери, який зародився ще в 1980-х роках. Ключові особливості цього жанру включають переміщення персонажа між локаціями, подолання перешкод різного типу, виконання стрибків та використання різних механік для взаємодії з ігровим середовищем. Класичні представники жанру, такі як Super Mario Bros., Sonic the Hedgehog, Donkey Kong, заклали основи платформерів, які розвиваються і вдосконалюються навіть сьогодні [10].

Сучасні 2D-платформери досі займають важливе місце в індустрії відеоігор завдяки простоті освоєння, динамічному геймплею, можливості реалізації цікавих механік та доступності для різноманітних пристроїв. Ігри, такі як Celeste, Hollow Knight, Ori the Blind Forest, демонструють, що сьогодні навіть у 2D-форматі можна створити унікальний і вражаючий ігровий досвід із глибокою історією, захопливим ігровим процесом та стильним візуальним оформленням [11].

Вибір жанру 2D-платформер для бакалаврської роботи обумовлений кількома важливими факторами:

- Популярність жанру. Платформери залишаються затребуваними серед гравців завдяки динамічному ігровому процесу, доступному управлінню та різноманітності унікальних механік.

- Доступність розробки. Програмне забезпечення, наприклад Godot, має всі необхідні вбудовані інструменти для створення ігор цього жанру, що робить процес розробки зручним та швидким.
- Гнучкість у реалізації. У межах жанру можна експериментувати з різноманітними ігровими механіками, складністю рівнів і стилістикою гри.
- Відносно невеликі ресурси. 2D-ігри мають менші вимоги до графічних і апаратних ресурсів, що дозволяє їх запускати на широкому спектрі пристроїв [10; 11].

Таким чином, створення комп'ютерної гри у жанрі 2D-платформера є доцільним та обґрунтованим вибором для реалізації бакалаврського проєкту. Це дозволяє поглибити знання в розробці ігор, ознайомитися з основами програмування в ігровому середовищі, освоїти роботу з графікою та фізикою персонажів, а також дослідити ключові принципи ігрового дизайну.

### 1.2 Аналіз існуючих 2D-платформерів та їх особливостей

Жанр платформерів має дуже багату історію розвитку та здійснив значний вплив на формування сучасної ігрової індустрії. Від перших простих аркадних проєктів до сучасних інтерактивних і масштабних ігрових продуктів цей жанр постійно еволюціонував, адаптуючись до нових технологічних можливостей і вимог користувачів. Сучасні 2D-платформери поєднують у собі як класичні ігрові елементи або механіки, притаманні раннім іграм, так і інноваційні, що дозволяє створювати унікальні та захопливі ігрові проєкти [12].

Сучасні 2D-платформери також поєднують елементи різних піджанрів, що дозволяє створювати різноманітний та унікальний ігровий процес. Серед найбільш популярних піджанрів можна виділити metroidvania, run-and-gun, roguelike та puzzle-platformer. Metroidvania поєднує дослідження великого ігрового світу з поступовим відкриттям нових локацій і можливостей персонажа, roguelike-ігри базуються на випадковій генерації та високій складності проходження, run-n-gun-проєкти орієнтовані на динамічний рух і бойову систему, а puzzle-platformer змушує гравця вирішувати головоломки різного типу. Таке

поєднання жанрових особливостей активно використовується у проєктах різного масштабу [13].

Окрім жанрових особливостей та ігрових механік, вижливу роль у розробці відеоігор відіграють художній стиль, дизайн персонажів, візуальна атмосфера та організація ігрового середовища, оскільки саме ці елементи впливають на перше враження, загальне сприйняття гри користувачем і його занурення в ігровий світ. У сучасному геймдизайні значна увага також приділяється цілісності стилістичного оформлення, проектуванню інтерфейсу та продуманій організації ігрового простору. Всі ці аспекти дозволяють зробити ігровий процес більш зрозумілим, атмосферним і цікавим для гравця. [14].

Розглянемо характеристики основних представників найбільш популярних ігор жанру платформер:

1) Super Mario Bros.

Рисунок 1.5 – Ігровий процес Super Mario Bros [15]

Super Mario Bros – культовий 2D-платформер від Nintendo, випущений у 1985 році. Гра стала основою для всього жанру платформерів, пропонуючи простий, але водночас захопливий геймплей.

Основні особливості – прості та зрозумілі механіки керування, поступове ознайомлення гравця з перешкодами та велика кількість прихованих елементів, які стимулюють дослідження рівнів. Гра також сформувала класичний підхід до побудови платформерів, у межах якого складність зростає поступово, а нові механіки вводяться через сам процес проходження [12].

2) Sonic the Hedgehog


Рисунок 1.6 – Ігровий процес Sonic the Hedgehog [16]

Sonic the Hedgehog – легендарний 2D-платформер від Sega, випущений у 1991 році. Гра стала головним конкурентом Super Mario Bros. і відзначалася високою швидкістю ігрового процесу [12].

Надзвичайно швидкий темп проходження, орієнтація на безперервний рух та велика кількість альтернативних маршрутів на рівнях робить цю гру унікальною навіть сьогодні [12]. Значна увага приділяється інерції руху персонажа, взаємодії з елементами середовища та створенню відчуття швидкості під час проходження.

3) Celeste

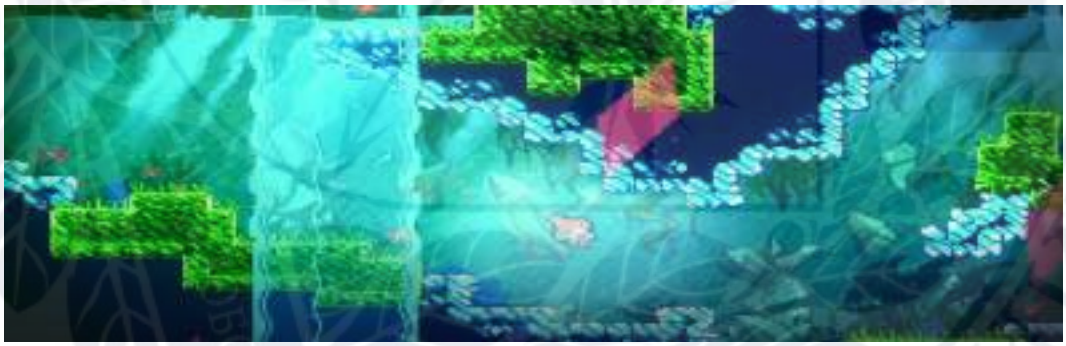

Рисунок 1.7 – Ігровий процес Celeste [17]

Celeste – інді-платформер, випущений у 2018 році студією Maddy Makes Games [17].

Основними особливостями гри є високоточне керування персонажем, складні випробування завдяки платформам та механіка повітряного ривка, навколо якої побудовано значну частину рівнів. Важливу роль відіграє емоційна

сюжетна складова, яка поєднується з атмосферною музикою та піксельним художнім стилем, створюючи цілісний ігровий досвід [17].

4) Hollow Knight


Рисунок 1.8 – Ігровий процес Hollow Knight [11]

Hollow Knight – атмосферний 2D-платформер у жанрі метроїдванія, випущений студією Team Cherry у 2017 році. [18]

Гра особлива тим, що має великий нелінійний світ, систему дослідження локацій та поступове відкриття нових можливостей персонажа, які дозволяють отримувати доступ до раніше недоступних зон. Гра також вирізняється унікальним стилем візуального оформлення, виконаним у ручній рисовці, похмурою, але водночас атмосферною стилістикою світу та якісним музичним супроводом, який підсилює відчуття загадковості королівства Hallownest [18].

5) Ori and the Blind Forest

Ori and the Blind Forest – емоційно насичений 2D-платформер, випущений у 2015 році студією Moon Studios [19].

Основними особливостями є плавна система пересування, поєднання платформінгу з елементами дослідження та емоційна подача сюжету. Значну

увагу приділено візуальному оформленню, динамічній анімації та оркестровому музичному супроводу, які формують виразну атмосферу ігрового світу [19].

Рисунок 1.9– Ігровий процес Ori and the Blind Forest [19]

6) Cuphead

Рисунок 1.10– Ігровий процес Cuphead [20]

Cuphead – 2D-платформер із елементами run-and-gun, розроблений студією StudioMDHR та випущений у 2017 році [20].

Гра прославилась складними боями з босами, які потребують швидкої реакції, точного ухилення та запам’ятовування атак супротивників [20]. Cuphead

вирізняється ручною анімацією, стилізованою під американські мультфільми 1930-х років, а музичний супровід, виконаний в жанрах джазу, свінгу, регтайму та раннього біг-бенду, додатково підсилює атмосферу класичної анімації та формує впізнаваний стиль проєкту. Атмосферу гри було доповнено звуковими ефектами, які також створювались у стилі старих мультфільмів і використовували характерні “мультяшні” звуки, притаманні анімації того періоду [21]. Крім того, у грі реалізовано кооперативний режим для двох гравців, який дозволяє проходити рівні спільно та покращує взаємодію між користувачами під час гри [20].

7) Dead Cells

Рисунок 1.11 – Ігровий процес Dead Cells [22]

Dead Cells – динамічний 2D-платформер у жанрі roguelike-метроїдванія, розроблений студією Motion Twin і випущений у 2018 році [22].

Особливостями гри є швидкий бойовий процес, процедурна генерація рівнів та велика кількість комбінацій зброї і здібностей [22]. Гра стимулює постійне експерементування зі стилями проходження та поєднує елементи roguelike із системою поступового розвитку персонажа.

Крім великих комерційних проєктів, значний вплив на розвиток жанру платформерів також мають так звані Game Jam – короткотривалі змагання серед

розробників, у межах яких необхідно створити прототип гри за обмежений проміжок часу. Подібний формат дозволяє експерементувати з новими механіками, художніми стилями та ідеями ігрового процесу. Деякі успішні інді-проєкти, що були розглянуті раніше, також мають зв'язок із Game Jam. Наприклад, ранній прототип Hollow Knight був створений під час конкурсу Ludum Dare 27, а перша версія Celeste розроблялась у межах чотириденного Game Jam у 2016 році [23]. Це демонструє, що навіть невеликі екперементальні проєкти можуть у майбутньому перетворюватись на повноцінні та успішні відеоігри.

Отже, проведений аналіз 2D-платформерів дозволяє зробити висновок, що ключовими особливостями жанру є поєднання динамічних механік руху, взаємодії з ігровим середовищем, поступового ускладнення рівнів, художнього стилю та атмосферного оформлення. Крім того, важливу роль у платформерах відіграють нелінійність проходження, система прогресії та баланс між складністю й доступністю ігрового процесу для різних категорій гравців.

### 1.3 Концептуальний підхід до розробки гри жанру платформер

Під час створення відеоігор важливу роль відіграє не лише технічна і графічна реалізація проєкту, але й формування цілісної концепції гри, яка визначає взаємодію користувача з ігровим середовищем. Успішний ігровий проєкт повинен поєднувати різноманітну кількість елементів, що сформують ігровий досвід користувача та підтримають його зацікавленість протягом проходження гри [24; 25].

Важливою складовою розробки є також розуміння психології гравця та аналіз його поведінки під час взаємодії з грою. У сучасному геймдеві поширеним є підхід, відповідно до якого розробник повинен створювати гру, у яку він сам хотів би грати. Подібний принцип дозволяє краще відчувати баланс між складністю, динамікою та задоволенням від проходження. Крім того, значну увагу приділяють аналізу відгуків користувачів, тестуванню механік та врахуванню сучасних тенденцій розвитку ігрової індустрії [24; 25].

Під час формування концепції власного проєкту було обрано підхід, орієнтований на створення простого але водночас динамічного 2D-платформера CoinQuest: Hardcore з підвищеним рівнем складності, у якому основний акцент планується зробити на точності керування персонажем, взаємодії з ігровим середовищем та поступовому ускладненні ігрового процесу. Концепція гри передбачає створення зрозумілого для користувача ігрового досвіду з використанням піксельної стилістики, різноманітних механік, систем та елементів сюжетної складової. Водночас концепція враховує загальну атмосферу гри, яка має формуватись за рахунок візуального оформлення, анімацій, користувацького інтерфейсу та аудіосупроводу.

Концептуальний підхід до розробки гри має враховувати наступні складові:

- реалізація системи керування персонажем, яка включає базовий рух та стрибки;
- забезпечення коректної фізичної взаємодії персонажа з ігровим середовищем, включаючи обробку зіткнень та вплив гравітації;
- розробка системи платформ, включаючи статичні та рухомі платформи, які формують основу ігрового середовища;
- розробка системи ігрових об’єктів, зокрема монет, бонусів та інших елементів, що впливають на ігровий процес;
- реалізація бонусних механік, які тимчасово змінюють характеристики персонажа (збільшення швидкості, покращення стрибка, можливість взаємодії зі стінами);
- створення небезпечних зон ігрового середовища (шипи, лава, вода та інші), які призводять до втрати життя персонажа при взаємодії з ними;
- розробка системи ворогів із різними типами поведінки, включаючи ворогів, що здійснюють дистанційні атаки;
- реалізація системи снарядів для деяких ворогів, включаючи стрільбу, рух куль та взаємодію з персонажем гравця;
- створення системи бос-битв, що включає ворогів із унікальною поведінкою, переслідуванням гравця та атакувальними механіками;

- реалізація механіки взаємодії з дверима та переходами між частинами рівнів;
- створення елементів сюжетної складової гри;
- реалізація системи анімацій персонажів та ігрових об’єктів;
- реалізація системи рівнів та переходів між ними, включаючи логіку відкриття нових рівнів;
- розробка системи збереження прогресу гри, яка дозволяє зберігати досягнутий рівень проходження;
- створення користувацького інтерфейсу, включаючи головне меню, меню паузи та елементи відображення ігрової інформації (кількість зібраних монет, смертей);
- реалізація звукового супровіду гри, зокрема фонову музику та звукові ефекти;
- реалізація візуальних ефектів, зокрема плавні переходи під час смерті персонажа та перезапуску рівня;
- реалізація системи навчальних підказок та текстових повідомлень, що пояснюють правила гри, ігрові механіки та сюжетні події.

# РОЗДІЛ 2
# МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ ГРИ ЖАНРУ ПЛАТФОРМЕР

## 2.1 Принципи та інструменти проєктування рівнів та ігрової прогресії

Проєктування рівнів є дуже серйозним і одним з найголовніших етапів розробки будь-якої комп'ютерної гри, оскільки саме через структуру рівня формується взаємодія і знайомство гравця з ігровим середовищем. Глибоке проєктування рівнів допоможе визначити темп проходження, рівень складності та загальний ігровий досвід. На відміну від загального геймдизайну, який визначає правила та основні ігрові механіки, левел-дизайн орієнтується на створення конкретного ігрового досвіду в межах окремих рівнів, локацій та ігрових ситуацій[26].

У відеоіграх рівень виконує не лише функцію ігрового простору, але й виступає засобом навчання гравця, розвитку сюжету, створення атмосфери та підтримання інтересу до проходження. Розробник має поступово знайомити гравця з грою саме через структуру рівня[27].

Базовим прикладом побудови рівнів є ускладнення ігрового процесу. Наприклад, на початкових етапах гри користувач лише ознайомлюється з базовими механіками руху без ризиків зробити помилку. Після цього гра поступово може комбінувати вже відомі елементи гри з новими перешкодами, збільшуючи складність і динаміку проходження.

Індустрія розробки ігор рідко обмежується одноразовим створенням механік або рівнів. Більшість сучасних проєктів використовує ітеративний підхід до розробки, відповідно до якого окремі елементи гри багаторазово тестуються, аналізуються та вдосконалюються. Відповідний процес дозволяє поступово покращувати баланс складності, дизайн, якість ігрових механік та загальний користувацький досвід [28]. Крім того, ітеративний підхід дозволяє своєчасно виявляти технічні помилки, недоліки дизайну та проблеми взаємодії користувача з грою ще на ранніх етапах розробки [29].


Рисунок 2.1 – Ітеративний цикл проєктування, тестування та вдосконалення ігрового процесу [28]

На рисунку показано, що процес створення гри складається з декількох взаємопов'язаних етапів: планування, проєктування, програмування, тестування, оцінювання результату та випуск готового продукту. Після кожного етапу тестування розробники можуть повертатись до попередніх стадій для внесення змін і вдосконалення окремих елементів гри, що забезпечує більш гнучкий процес розробки та дозволяє поступово досягати необхідного рівня якості, стабільності та балансу ігрового процесу [29; 18].

Одним з найважливіших принципів побудови рівнів є правильне формування кривої складності (difficulty curve), яка визначає співвідношення між навичками гравця та рівнем складності рівня під час гри. Основною метою такої системи є підтримання так званого "стану потоку" (flow state), за якого складність гри залишається високою для підтримки інтересу гравця, але водночас не такою складною, щоб викликати фрустрацію або втрату мотивації гравця під час нескінчених невдалих спроб проходження гри або рівня[27].

У випадку, коли рівень складності значно перевищує можливості гравця, виникає зона фрустрації, у якій користувач починає часто помилятися та втрачати інтерес до ігрового процесу та гри в цілому. Водночас надто проста гра формує “зону нудьги”, оскільки відсутність виклику знижує рівень залучення гравця[27]. Саме тому сучасний левел-дизайн орієнтується на поступове та контрольоване зростання складності, яке дозволяє користувачу постійно розвивати свої навички та відчувати прогрес під час проходження[26].

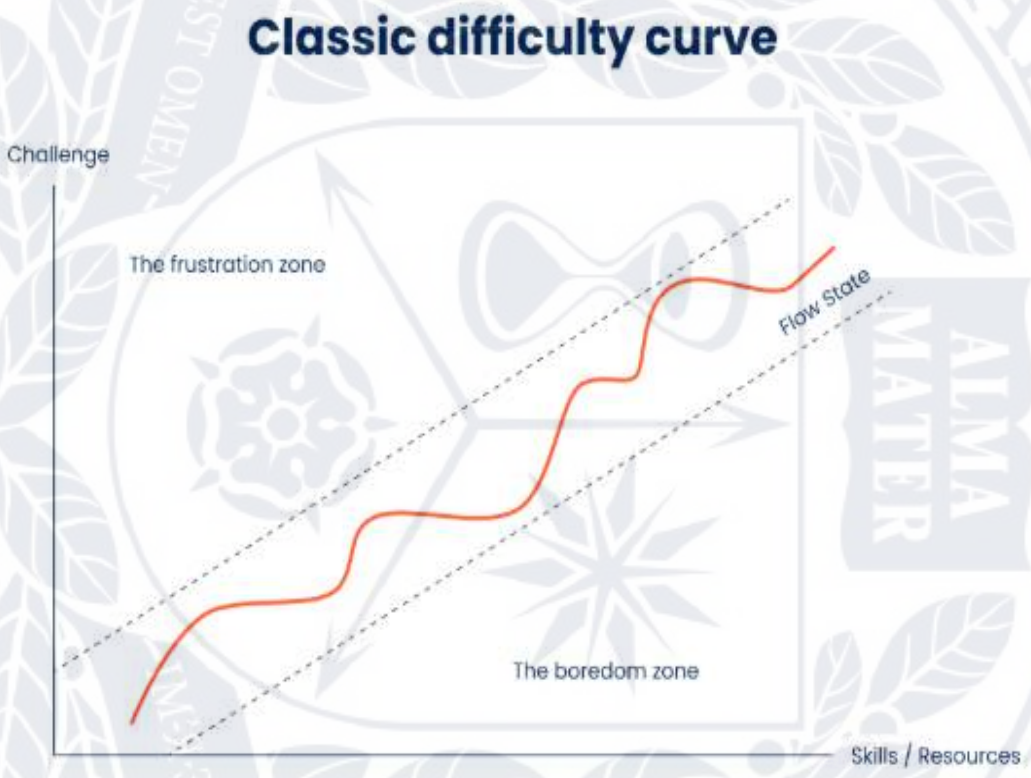


Рисунок 2.2 – Класична крива складності та співвідношення між навичками гравця і рівнем виклику [27]

Для підтримки балансу складності розробники ігор можуть змінювати такі аспекти як: кількість ресурсів, складність перешкод, швидкість ігрового процесу, кількість одночасних завдань. Крім того, важливу роль відіграє система винагород і допоміжних елементів. Дотримання балансу між цими аспектами дозволяє компенсувати різкі стрибки складності та підтримувати комфортний темп і цікаве проходження гри.

Таблиця 2.1 – Основні елементи проєктування ігрового процесу та їх призначення

| Елемент | Призначення |
|---|---|
| Крива складності | Підтримання балансу між рівнем виклику та навичками гравця |
| Система прогресії | Створення відчуття розвитку персонажа |
| Система винагород | Мотивація гравця до подальшого проходження |
| Дерево розвитку | Формування індивідуального стилю гри |
| Сюжетна прогресія | Підвищення емоційного залучення гравця |
| Візуальний стиль | Формування атмосфери та навігації |
| Звукове оформлення | Підсилення емоційного сприйняття гри |
| Тестування | Виявлення проблем балансу та взаємодії. |

Важливим елементом підтримання інтересу гравця є система прогресії і винагород. Ця система створює відчуття розвитку персонажа, досягнення цілей та постопового освоєння нових можливостей. У відеоіграх прогресія використовується як механізм поступового розвитку ігрового процесу та як інструмент мотивації гравця до подальшого проходження гри [27;29].

Одним із найпопулярніших підходів є рівнева система прогресії (level-based progression). В межах цього підходу розвиток гравця, персонажа або сюжетної лінії поділяється на фіксовані етапи. Наприклад, перемоги над ворогами, проходження рівнів або додаткових квестів. Подібний принцип використовується у грі Lies of P, де персонаж отримує досвід за перемогу над

противниками та босами, що дозволяє покращувати характеристики героя та відкривати нові бойові можливості. Така система створює відчуття розвитку та дозволяє гравцю поступово розвиватись в ігровому середовищі [30].

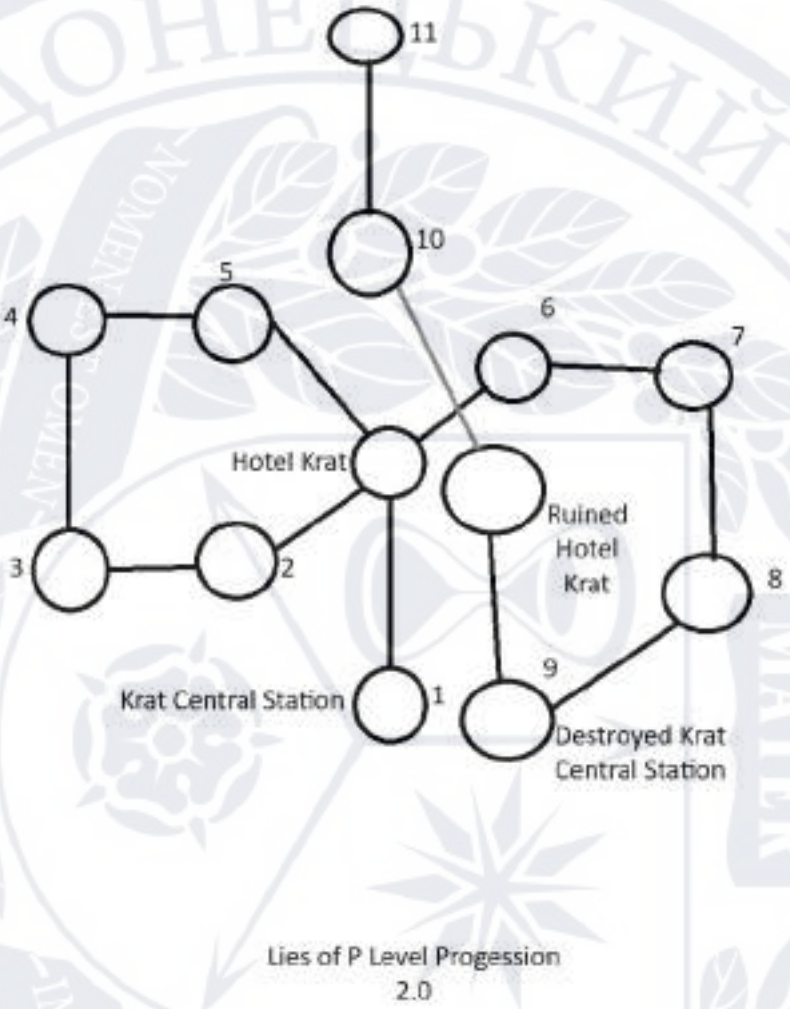


Рисунок 2.3– Графова модель послідовності проходження та взаємозв’язку ігрових рівнів у грі Lies of P [32]

На схемі представлено структуру переходів між основними локаціями гри та можливі маршрути просування гравця, що дозволяє формувати логічну систему прогресії, постопово відкривати нові ігрові локації та контролювати темп проходження гри.

Крім класичної системи рівнів, у сучасних іграх активно використовується система навичок (skill tree). Система навичок дозволяє гравцю самостійно обрати напрям розвитку персонажа. Даний підхід забезпечує більшу варіативність

ігрового процесу та підвищує рівень залучення гравця [30]. Наприклад, у онлайн-грі ARC Raiders реалізовано систему розвитку персонажа, що включає декілька основних напрямків навичок, серед яких Conditioning, Mobility та Survival. Кожен із напрямів впливає на окремі характеристики персонажа та стиль гри. Подібний підхід змушує гравця думати наперед та враховувати переваги і недоліки конкретних навичок, які гра пропонує обрати.


Рисунок 2.4 – Дерево розвитку навичок у грі ARC Raiders

На рисунку представлено приклад системи розвитку навичок певним гравцем у грі ARC Raiders. Напрям Mobility орієнтований на швидкість пересування, витривалість та загальну мобільність персонажа. Survival відповідає за виживання, ефективність пошуку ресурсів, побудову предметів та можливість переносити більше спорядження. Гілка Conditioning спрямована на покращення бойової ефективності персонажа, зокрема керування витривалістю під час бою, використання спорядження та взаємодію зі зброєю. Така система робить ігровий процес більш цікавим і непередбачуваним, оскільки дозволяє тисячам гравців формувати максимально оригінальне дерево розвитку. Наприклад, один гравець може орієнтуватись на мобільність і швидке пересування картою, інший – на виживання та ефективне накопичення ресурсів, а хтось – на агресивний стиль гри з акцентом на бойові здібності.

Окреме значення у сучасному геймдизайні має сюжетна прогресія, за якої розвиток персонажа та відкриття нових можливостей безпосередньо пов’язані з подіями гри та рішеннями користувача. Сюжетна прогресія дозволяє зробити проходження більш емоційним, варіативним та індивідуальним для кожного гравця [30; 31]. Найвідомішим прикладом такої системи є гра Detroit: Become Human, у якій кожне рішення користувача може впливати на розвиток сюжету, взаємодію між персонажами та фінал історії. Особливістю гри є використання блок-схем (flowcharts), відповідних до окремих глав гри, що демонструють різні варіанти проходження рівня та наслідки виборів гравця. Загалом гра містить 30 глав та понад 80 різних можливих кінцівок, що формує високу варіативність проходження та стимулює користувачів до повторного проходження гри[34; 35].

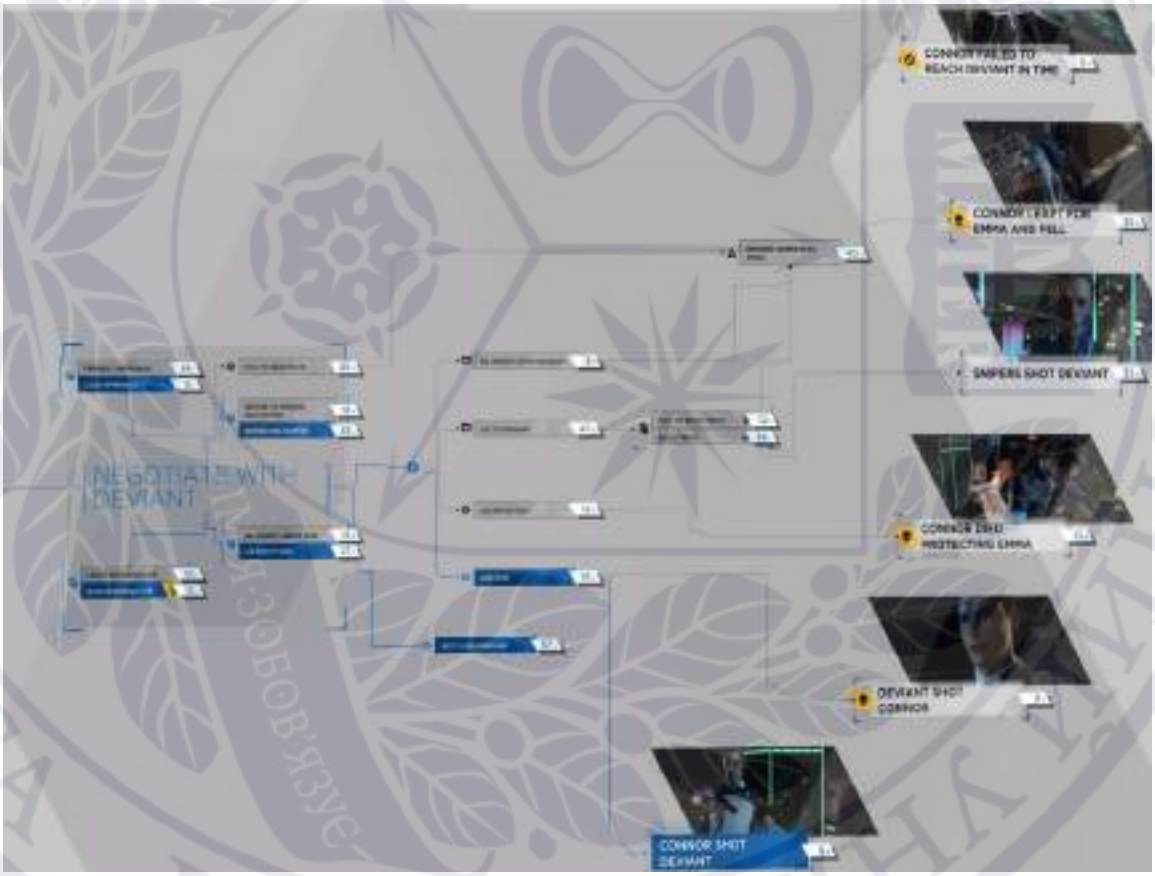


Рисунок 2.5 – Приклад фрагменту блок-схеми першого рівня у грі Detroit: Become Human [34]

На фрагменті блок-схеми представлено один із можливих варіантів розвитку подій та кінцівки глави у грі. Схема демонструє, як окремі рішення та

дії користувача впливають на подальший розвиток сюжету, взаємодії між персонажами та фінальний результат проходження епізоду. На схемі також відображається статистика виборів користувачів, що дозволяє оцінити популярність окремих рішень серед гравців та їх вплив на розвиток сюжетної лінії [34].

Не менш важливу роль у проєктуванні рівнів відіграють візуальний стиль та звуковий супровід. Саме ці елементи значною мірою формують емоційне сприйняття ігрового світу, впливають на атмосферу проходження та доповнюють основну ідею проєкту. У геймдизайні візуальна складова використовується як для покращення зовнішнього вигляду гри, так і як інструмент навігації. Акцентування уваги та створення певного емоційного стану дозволяє гравцю зрозуміти, на якому етапі проходження він знаходиться, чи оточує його небезпека, наскільки складною є поточна ігрова ситуація та як необхідно взаємодіяти з навколишнім середовищем [36; 18]. Отже, правильне поєднання графіки, кольорової палітри, освітлення та звукового супроводу дозволяє зробити рівні більш впізнаваними та атмосферними.

У різних відеоіграх використовуються різні художні стилі залежно від жанру, атмосфери та загальної концепції проєкту. Наприклад, гра Cuphead, яка була розглянута у попередніх розділах і є 2D-платформером, виконана у стилі класичної американської анімації 1930-х років, а музичний супровід у стилі тих часів додатково підсилює атмосферу гри та формує її унікальний стиль. Водночас у грі Limbo, яка також є 2D-платформером, використовується мінімалістичний художній підхід із чорно-сірою кольоровою гамою, обмеженим освітленням та майже повною відсутністю музичного супроводу у традиційному його розумінні. Основний акцент у грі зроблено саме на атмосферному звуковому оформленні: звуках кроків, шумі навколишнього середовища, взаємодії об’єктів та раптових аудіоефектах, які створюють напруження та підсилюють відчуття небезпеки під час проходження. Наведений приклад демонструє, що для створення сильної атмосфери у грі не завжди необхідне постійне використання музики, оскільки

якісне звукове оформлення саме по собі може стати важливою складовою ігрового досвіду [37; 38; 39].

Рисунок 2.6 – Приклад мінімалістичного художнього стилю та атмосферного оформлення у грі Limbo [39]

На рисунку представлено приклад використання мінімалістичного художнього стилю у грі Limbo. У сцені поєднується темна кольорова палітра, силуетне зображення персонажів, обмежене освітлення та ефект туману, що створює похмуру й напружену атмосферу ігрового світу. Особливу роль у формуванні емоційного сприйняття відіграє невідомість та приховування значної частини оточення у тіні, у поєднанні із звуковим оформленням, відбувається ефект постійної небезпеки та невизначенності під час проходження.

Незалежно від початкової концепції гри, жанру чи використаних механік, процес розробки сучасних відеоігор вимагає постійного тестування та вдосконалення ігрового проєкту. Розробники на різних етапах проєкту продовжують аналізувати окремі елементи гри та постійно їх змінювати під час тестування. У деяких випадках подібні зміни дозволяють частково або повністю переосмислити початкову ідею гри та покращити загальний користувацький досвід, а в інших – навпаки допомагають поступово довести основну концепцію проєкту до фінального стану без втрати її ідеї або ключових особливостей. Тому,

сучасний геймдев значною мірою базується на ітеративному підході до розробки, який передбачає постійне тестування, аналіз поведінки гравців та поетапне вдосконалення окремих компонентів ігрового процесу [40].

У сучасній ігровій індустрії використовується велика кількість різних типів тестування, кожен з яких спрямований на оцінювання окремих аспектів гри:

- функціональне тестування – перевірка коректності роботи механік, взаємодії об’єктів та основних ігрових систем;
- тестування продуктивності – оцінювання стабільності гри, використання ресурсів системи та моніторинг рівня FPS;
- usability-тестування – аналіз зручності керування, користувацького інтерфейсу та загального комфорту взаємодії з грою;
- beta-тестування – залучення користувачів для отримання відгуків та виявлення недоліків перед фінальним випуском гри;
- regression-тестування – перевірка коректності роботи наявних функцій після внесення змін до коду.
- stress/load-тестування – оцінювання роботи гри при високому навантаженні або нестандартних умовах використання [40; 41].

Подібний комплексний підхід дозволяє своєчасно виявляти технічні помилки, оцінювати реакцію користувачів на окремі механіки та формувати більш якісний, збалансований, захопливий і повноцінний досвід для гравця [40; 41].

Підсумовуючи, можна зробити висновок, що сучасне проєктування ігрових рівнів є складним багатокомпонентним процесом, а використання ітерактивного підходу до розробки дозволяє своєчасно виявляти недоліки, вдосконалювати окремі елементи гри та підвищувати загальну якість ігрового проєкту. Поєднання грамотної і продуманої проєкції та ітеративного підходу сприяє створенню більш збалансованого, атмосферного та цілісного ігрового досвіду для гравця.

## 2.2 Godot Engine як середовище розробки 2D-ігор

Godot Engine – це сучасний безкоштовний ігровий рушій, який використовується для створення 2D- та 3D-ігор різної складності. За останні роки Godot став популярним інструментом серед незалежних розробників завдяки простоті використання, модульній структурі та широким можливостям для створення інтерактивних проєктів. Однією з головних переваг рушія є повноцінна підтримка 2D-розробки, яка реалізована окремо від 3D-системи, що дозволяє забезпечити високу продуктивність та точне керування двовимірною графікою [43].

У порівнянні з іншими популярними рушіями, такими як Unity або Unreal Engine, Godot має значно простіший інтерфейс та менші системні вимоги, що робить його зручним для навчання, створення прототипів та розробки інді-ігор. Крім того, рушій поширюється за відкритою ліцензією MIT, є повністю безкоштовним та дозволяє розробникам вільно користуватися, змінювати й поширювати модифіковані версії рушія, зокрема й у комерційних проєктах. Відкритий вихідний код Godot Engine надає можливість адаптувати функціональність рушія відповідно до потреб конкретного ігрового проєкту, що є важливою перевагою для незалежних розробників та невеликих команд. [2; 52].

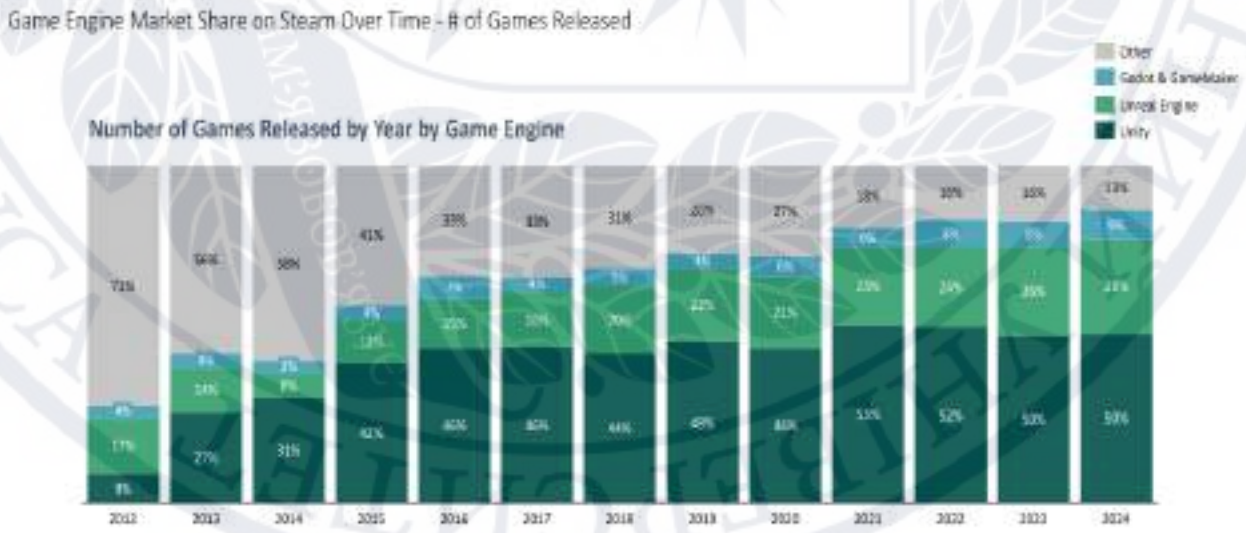


Рисунок 2.7 – Динаміка використання ігрових рушіїв серед проєктів платформи Steam [42]

На наведеному графіку представлено зміну популярності ігрових рушіїв серед відеоігор, опублікованих у сервісі Steam у період з 2012 по 2024 рік. Можна побачити, що частка ігор, створених на власних рушіях, поступово зменшується, тоді як популярність готових рішень для розробки ігор постійно зростає.

Найбільшу частку ринку протягом останніх років займає Unity, який використовується більш ніж у половині ігор, опублікованих у Steam. Водночас Unreal Engine, Godot та GameMaker також демонструють стабільне зростання популярності. [42].

Аналіз тенденції свідчить про активний розвиток доступних ігрових рушіїв та збільшення кількості незалежних розробників, які використовують готові інструменти для створення власних ігрових проєктів.

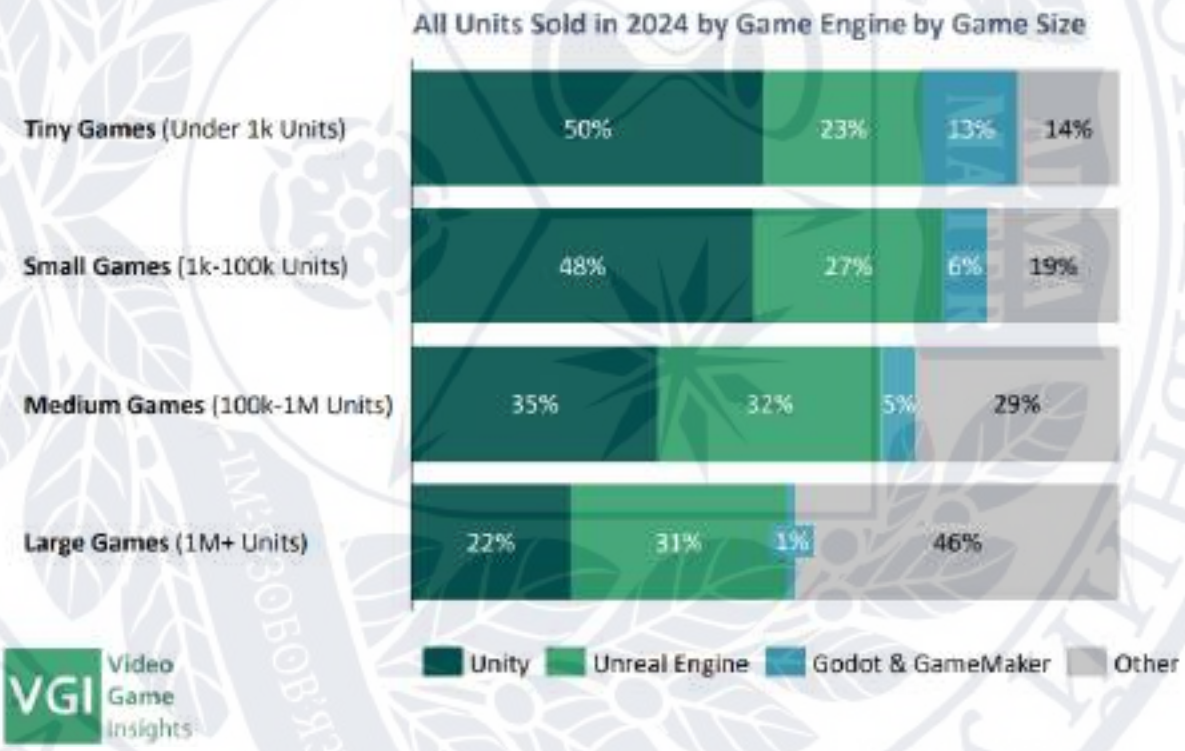


Рисунок 2.8 – Співвідношення популярності ігрових рушіїв залежно від масштабу ігрових проєктів [42]

На наведеному графіку показано залежність між масштабом ігрових проєктів та вибором ігрового рушія. Можна помітити, що невеликі та незалежні ігри найчастіше створюються з використанням Unity, Godot та GameMaker, тоді

як великі комерційні проєкти частіше використовують Unreal Engine або власні рушії.

Регулярний розвиток Godot Engine та розширення його функціональних можливостей сприяють збільшенню кількості розробників, які використовують рушій для створення власних проєктів. Доступність рушія також відкриває можливості для початківців у сфері розробки, що відображає загальне зростання популярності цифрової продукції та сучасних засобів її створення.

Основою роботи Godot є система сцен та вузлів. Кожен елемент гри у середовищі Godot представлений окремим вузлом (Node), який виконує певну функцію: відповідає за графіку, фізику, анімацію, інтерфейс або програмну логіку. Сукупність вузлів утворює сцену, яка може використовуватись як окремий компонент гри [43].

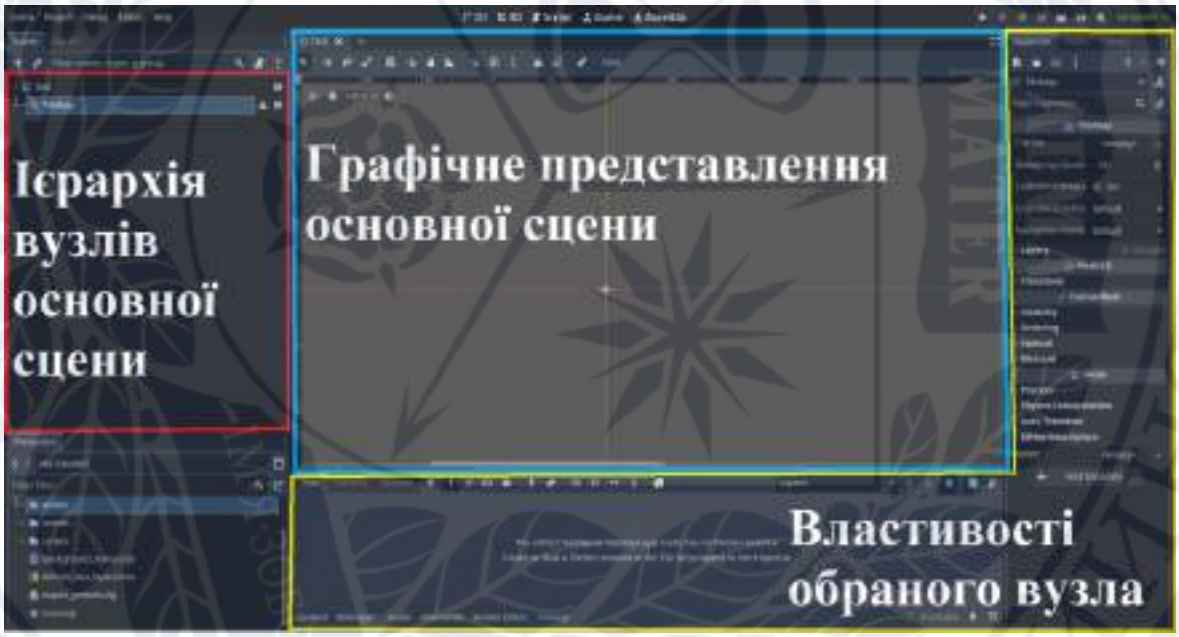


Рисунок 2.9 – Інтерфейс середовища розробки Godot Engine

Сцени у Godot організовані у вигляді деревоподібної структури, де кожен вузол може містити дочірні вузли, що дозволяє створювати складні ігрові об’єкти, поєднуючи декілька компонентів у межах однієї сцени. Наприклад, персонаж гри може складатися з вузлів CharacterBody2D, CollisionShape2D, AnimatedSprite2D, Camera2D та багатьох інших допоміжних елементів в залежності від проєкту. У документації Godot для створення статичного

персонажа використовується структура сцени, що складається з CharacterBody2D, Sprite2D та CollisionShape2D [43; 44].


Рисунок 2.10 – Приклад структури дерева та її графічного відображення у Godot Engine [43]

Важливою складовою створення персонажа є система колізій, яка визначає взаємодію об’єкта з іншими елементами рівня. Для цього у Godot використовуються вузли CollisionShape2D та різні типи фізичних форм, наприклад RectangleShape2D або CircleShape2D. У документації Godot як приклад використовується CircleShape2D, однак у платформерах часто також застосовується прямокутна форма колізії на певних об’єктах, яка забезпечує більш точну взаємодію персонажа з платформами та поверхнями [43; 45].

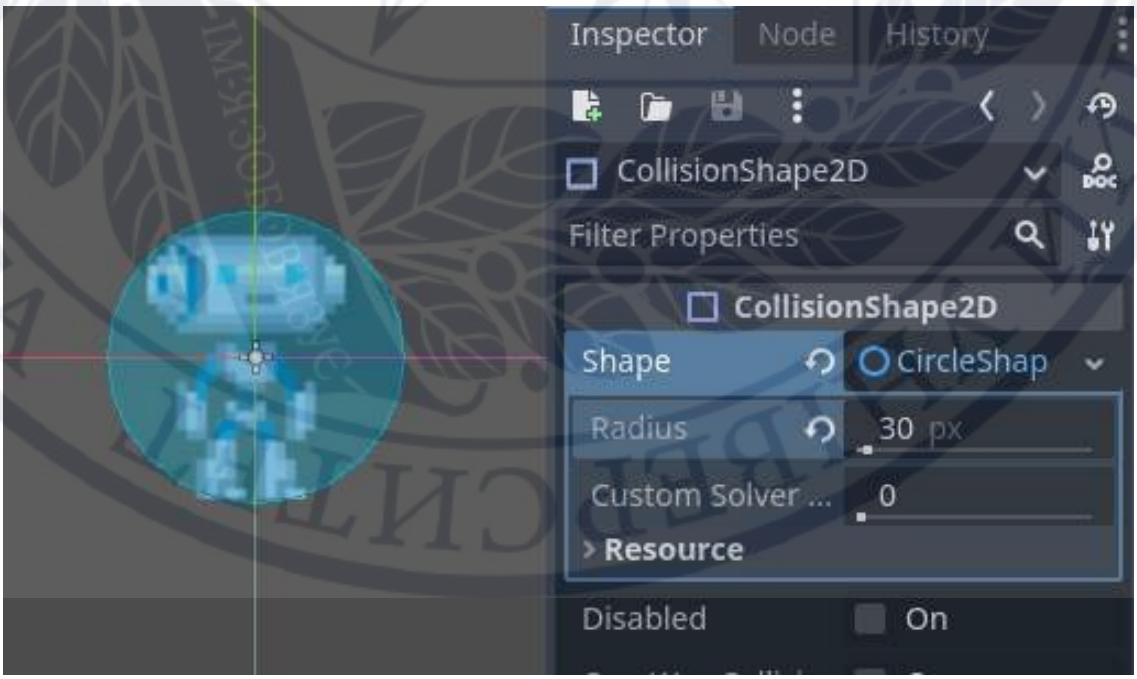


Рисунок 2.11 – Приклад налаштування CollisionShape2D у Godot Engine [43]

Для програмування ігрової логіки в Godot використовується мова GDScript. Вона схожа на Python, але модифікована й оптимізована рушієм для швидкої та зручної взаємодії між сценами, вузлами, групами, сигналами та додатковими функціями. Кожен вузол може містити окремий скрипт, який визначає його поведінку та взаємодію з іншими компонентами сцени [43].

У документації Godot для реалізації руху персонажа використовується функція _physics_process(), яка викликається синхронно з фізичним рушієм та забезпечує більш стабільну й передбачувану роботу фізики у порівнянні зі звичайним процесом _process(). Подібний підхід використовується для реалізації руху, гравітації, стрибків та інших механік платформерів [43].

```
extends CharacterBody2D

const GRAVITY = 200.0
const WALK_SPEED = 200

func _physics_process(delta):
    velocity.y += delta * GRAVITY

    if Input.is_action_pressed("ui_left"):
        velocity.x = -WALK_SPEED
    elif Input.is_action_pressed("ui_right"):
        velocity.x =  WALK_SPEED
    else:
        velocity.x = 0

    # "move_and_slide" already takes delta time into account.
    move_and_slide()
```

Рисунок 2.12 – Приклад реалізації базового руху персонажа за допомогою GDScript [43]

У наведеному фрагменті коду реалізовано базовий рух персонажа, обробку гравітації та взаємодію з клавішами керування. Для переміщення персонажа використовується функція move_and_slide(), яка автоматично враховує фізику руху та зіткнення з поверхнями. Обробка натискання клавіш виконується за

допомогою системи налаштувань InputMap, що дозволяє гнучко налаштовувати керування персонажем у середовищі Godot Engine [43; 46].

Для побудови ігрових рівнів у 2D-проєктах в Godot широко використовується система TileMap та TileSet. TileMap являє собою сітку плиток, яка дозволяє швидко створювати великі ігрові локації шляхом розміщення графічних ресурсів (тайлів) на рівні. Використання TileMap робить процес розробки швидким, спрощує редагування рівнів та дозволяє повторно використовувати графічні елементи у різних сценах [43; 47].

Окрім графічної побудови рівнів, TileMap підтримує використання колізій, використання декількох шарів (за допомогою вузла TileMapLayer), автоматичного поєднання тайлів та інших інструментів, які дозволяють створювати більш складне та деталізоване ігрове середовище [43; 47].

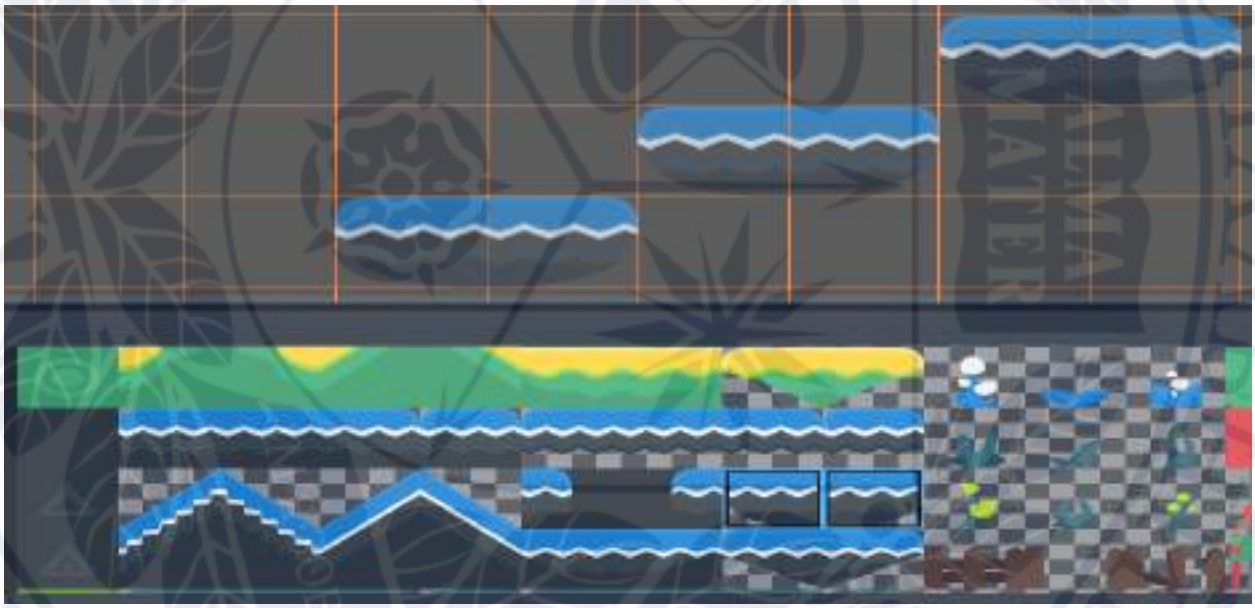

Рисунок 2.13 – Приклад використання TileMap для побудови рівня у Godot Engine [43]

Для досягнення додаткових візуальних ефектів у Godot використовуються шейдери – спеціальні програмні інструменти, які дозволяють змінювати спосіб відображення текстур, освітлення та інших графічних елементів. За допомогою шейдерів можна реалізовувати ефекти тіней, світіння, води, туману, зміни кольорів та інші візуальні особливості [43; 48].

Godot підтримує власну Shader Language, що дозволяє створювати та налаштовувати шейдери безпосередньо у середовищі розробки. Використання шейдерів дає можливість покращити візуальну складову гри та зробити ігрове середовище атмосфернішим і динамічнішим [43; 48].

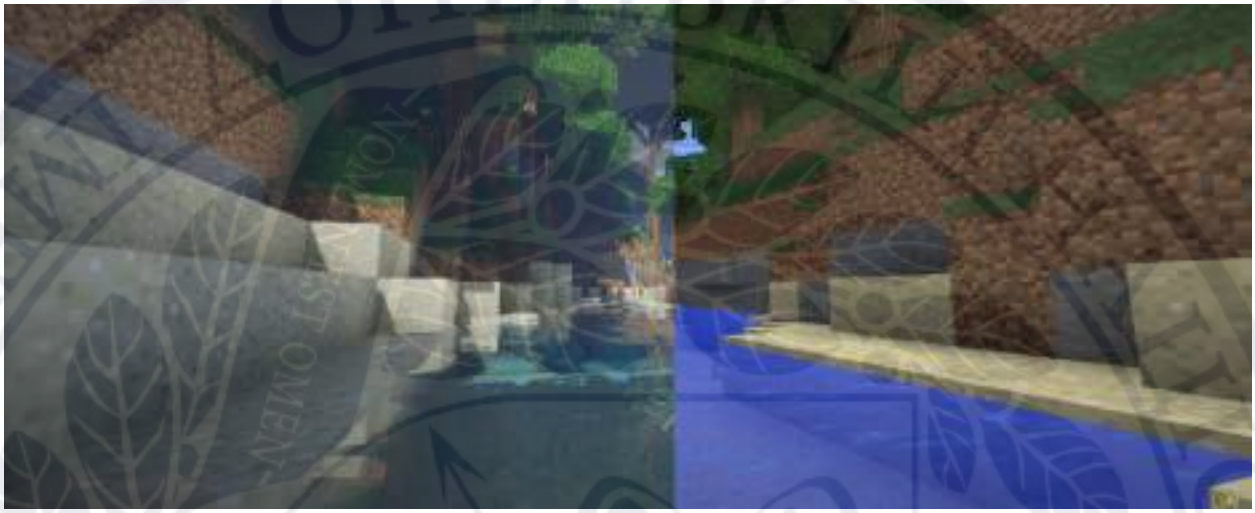

Рисунок 2.14 – Приклад з шейдерами проти без шейдерів на основі гри Minecraft

Для створення анімацій у Godot Engine використовується вузол AnimatedSprite2D, який дозволяє відображати послідовність кадрів та створювати різні стани персонажа. Анімації використовуються для відтворення руху, стрибків, зміни напрямку персонажа та інших дій під час ігрового процесу [43].


Рисунок 2.15 – Приклад налаштування анімацій персонажа у Godot Engine [43]

Для роботи з AnimatedSprite2D використовується ресурс SpriteFrames, який містить набір окремих анімацій та кадрів [43]. У наведеному прикладі створено стани анімацій walk та up, що відповідають руху персонажа та зміні напрямку його переміщення. Кожна анімація складається з окремих спрайтів, які послідовно відображаються під час гри.

Оскільки Godot використовує ієрархію сцен і вузлів, GDScript дозволяє легко взаємодіяти з іншими об’єктами в межах цієї структури. За допомогою програмного коду можна створювати зв’язки між вузлами, запускати анімації, змінювати властивості об’єктів або керувати їхньою поведінкою у реальному часі [43; 46]. Для роботи з анімаціями у скриптах Godot використовується підключення анімаційних вузлів до програмного коду за допомогою спеціальних змінних, що дозволяє отримувати доступ до AnimatedSprite2D, змінювати поточну анімацію, напрямок спрайта та керувати його відображенням залежно від стану персонажа під час гри.

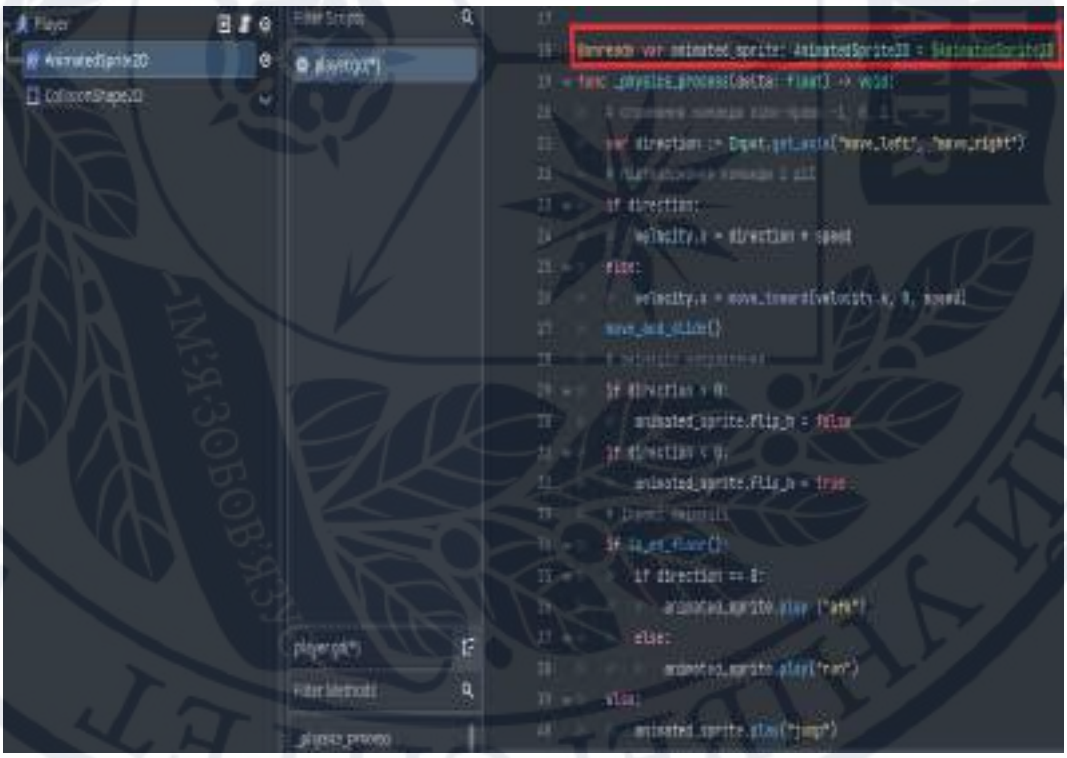

Рисунок 2.16 – Приклад створення зв’язку між вузлом персонажа та вузлом його анімації у скрипті player.gd за допомогою посилання @onready

Додатково у Godot використовується система сигналів, яка дозволяє

організовувати взаємодію між різними вузлами сцени [43]. Сигнали застосовуються для обробки натискання кнопок, взаємодії персонажа з об’єктами, переходів між сценами та інших ігрових подій. Подібний підхід дозволяє створювати більш гнучку та зручну структуру проєкту.

Окрему роль у сучасних відеоіграх відіграє користувацький інтерфейс, який забезпечує взаємодію гравця з ігровим середовищем та надає доступ до основних елементів керування грою. Для створення інтерфейсу у Godot використовуються спеціальні вузли, серед яких Control, Button, Label, TextureRect та інші UI-компоненти [43; 49].

Godot також підтримує систему контейнерів (наприклад, VBoxContainer), яка дозволяє автоматично адаптувати розташування елементів інтерфейсу залежно від розміру вікна гри або роздільної здатності екрана. Цей підхід спрощує створення меню, текстових повідомлень, кнопок та інших елементів взаємодії з користувачем.

Наприклад, сцена головного меню гри може складатися з кореневого вузла Control та містити кнопки Button, текстові елементи Label, фонові зображення та інші компоненти інтерфейсу. У найпростішому випадку головне меню може містити кнопки Play та Exit для запуску гри та виходу з програми після їх програмування за допомогою GDScript.

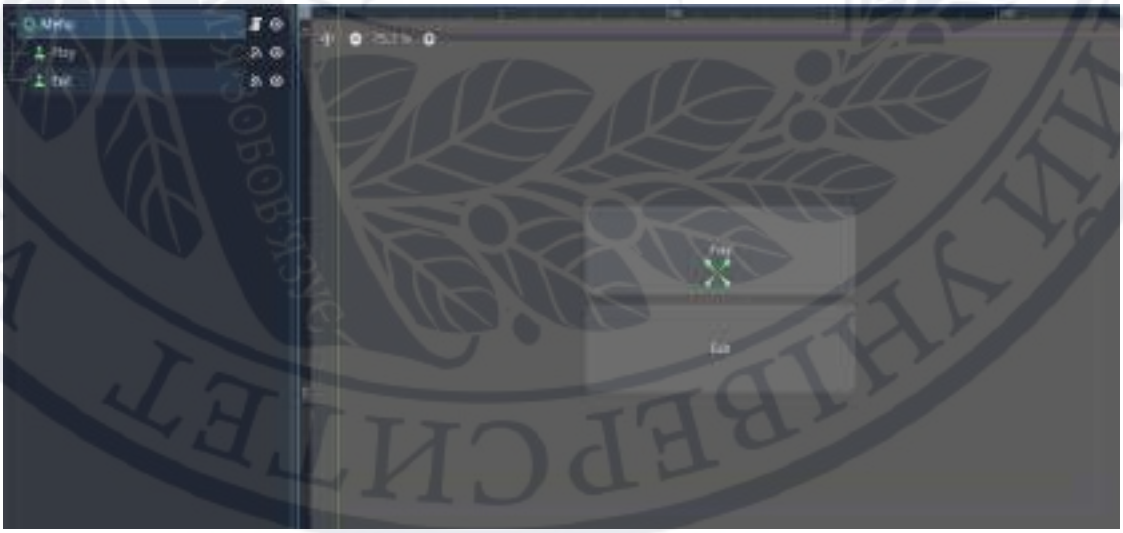

Рисунок 2.17 – Приклад сцени базового головного меню у середовищі розробки Godot Engine

Рисунок 2.18 – Приклад використання сигналів для кнопок головного меню та їх програмування

Наступною важливою складовою сучасних відеоігор є аудіосупровід, який використовується для створення атмосфери, передачі емоцій та покращення взаємодії гравця з ігровим середовищем. У Godot для роботи зі звуком у 2D-середовищі використовується вузол AudioStreamPlayer2D, який дозволяє відтворювати фонову музику, звукові ефекти та інші аудіоелементи [43; 49].

Godot підтримує роботу з різними аудіоформатами, зокрема WAV, MP3 та OGG, а також дозволяє налаштувати гучність, циклічне відтворення та інші параметри звуку безпосередньо у середовищі розробки [43].


Рисунок 2.19 – Приклад використання аудіо у середовищі розробки Godot Engine [43]

Таким чином, Godot Engine містить повноцінний набір інструментів для створення 2D-ігор, включаючи систему сцен і різноманітних вузлів із унікальними і широкими можливостями налаштування, зручні засоби програмування та підтримку основних компонентів сучасної ігрової розробки. Для кращого розуміння особливостей рушія доцільно розглянути його основні переваги та недоліки під час розробки 2D-інді-ігор.

Таблиця 2.2 – Переваги та недоліки Godot Engine розробки 2D-ігор [2; 49]:

| Переваги | Недоліки |
|---|---|
| Безкоштовний рушій з відкритим кодом | Менша кількість навчальних матеріалів у порівнянні з Unity |
| Низькі системні вимоги | Менша популярність серед великих студій |
| Простий та зрозумілий інтерфейс | Менша кількість вбудованих налаштувань та ресурсів |
| Повноцінна підтримка 2D-розробки | Обмежена кількість великих комерційних проєктів |
| Вбудована мова GDScript для швидкої розробки | Деякі інструменти ще активно розвиваються |
| Зручна система сцен і вузлів | Менша кількість спеціалістів на ринку праці |
| Наявність різноманітних і зручних інструментів для створення 2D-ігор | |
| Можливість експорту на різні платформи | |

Отже, Godot Engine надає все необхідне для реалізації власної ідеї та дозволяє ефективно організовувати процес створення гри. Завдяки поєднанню функціональності, доступності та зручності роботи даний рушій було обрано мною для розробки свого першого ігрового проєкту у жанрі 2D-платформер.

### 2.3. Ресурси візуалізації проєкту гри

Перед початком розробки 2D-гри важливо підготувати всі необхідні візуальні ресурси: текстури, тайли, спрайти персонажів, анімації, фони тощо. Це забезпечить цілісне візуальне оформлення проєкту та полегшить подальшу роботу з об’єктами в ігровому середовищі. Для пошуку графічних матеріалів можна скористатись спеціалізованими сайтами, які пропонують безкоштовні або платні пакети текстур, згруповані за різними жанрами. Вибір якісного і стилістично узгодженого пакета дозволяє з самого початку задати потрібну атмосферу гри.

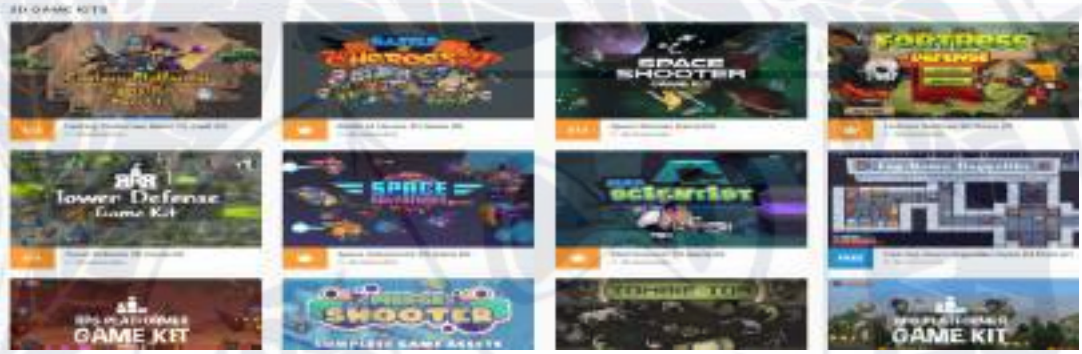


Рисунок 2.20 – Приклад сайту, який містить необхідні ресурси для розробки ігор [50]

Після ретельного пошуку на декількох сайтах були знайдені безкоштовні графічні матеріали, які використовуватимуться у грі.

Рисунок 2.21 – Графічні ресурси, які я використовуватиму у своїй грі [50;51]

Аналогічно до графічних матеріалів, можна знайти й шрифти, звукові ефекти, музику та інші аудіоресурси, які будуть імпортовані під час розробки у гру.

Тепер, після того як були знайдені всі необхідні ресурси для створення гри, потрібно налаштувати проєкт у середовищі Godot. Це включає в себе створення всіх необхідних папок та додавання матеріалів, які були завантажені раніше та які планується використовувати в грі.

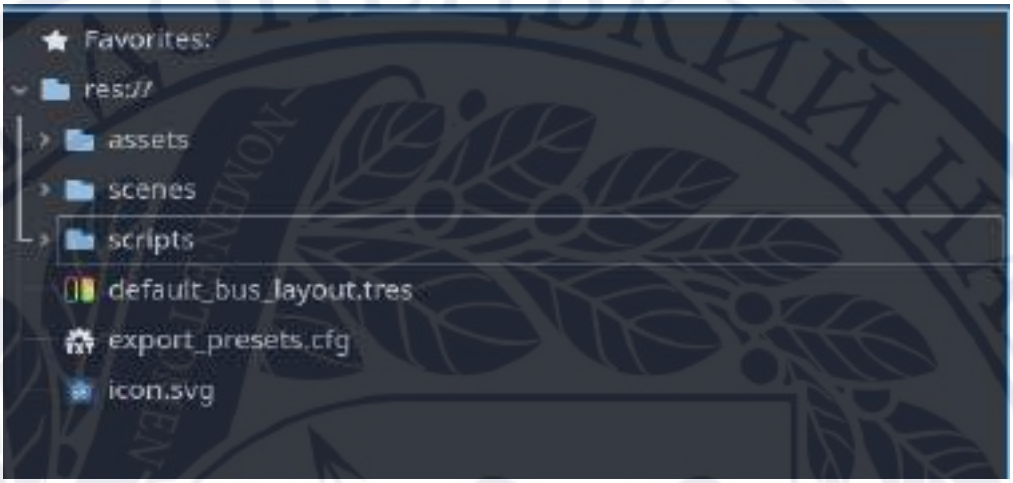


Рисунок 2.22 – Створення трьох основних папок, необхідних для коректного функціонування та організації структури гри

Кожна з папок використовується для зберігання окремих типів ресурсів гри:

- Assets – ця папка слугує для зберігання всіх графічних матеріалів, аудіофайлів та інших медіа-ресурсів. Усі елементи, що будуть використовуватись для візуалізації та звукового оформлення гри, повинні зберігатись у цій папці для зручності доступу та підтримки порядку.
- Scripts – у цій папці зберігатимуться скрипти, які відповідають за логіку гри. Всі скрипти, що описують механіку персонажів, взаємодії між об’єктами, а також інші аспекти ігрового процесу, будуть організовані у цьому каталозі.
- Scenes – у цій папці будуть зберігатись самі сцени гри, що містять різні рівні, персонажів, небезпечні зони, інтерфейси та інші важливі аспекти. Кожна сцена є важливою складовою гри, оскільки може представляти як окрему одиницю ігрового середовища або один елемент, так і повністю реалізований готовий рівень [54].

Організація цих папок допоможе підтримувати структуру проєкту, зробить

процес розробки зручнішим і дозволить легко знаходити необхідні файли під час подальшої розробки гри [54].

Після підготовки всіх необхідних ресурсів, можна розпочати розробку головного персонажа гри. Для створення персонажа в середовищі Godot нам знадобляться кілька ключових елементів:

- CharacterBody2D – основний вузол для персонажа, який відповідає за фізичну взаємодію з навколишнім середовищем. Цей вузол надасть можливість реалізувати рух персонажа, обробку зіткнень та взаємодію з механіками гри.
- AnimatedSprite2D – вузол, який дозволяє додавати анімацію до персонажа. З його допомогою ми можемо змінювати спрайти персонажа в залежності від його стану (наприклад, рух, стрибок, смерть тощо)
- CollisionShape2D – необхідний для налаштування фізичних зіткнень. За допомогою цього вузла ми визначаємо форму, з якою персонаж буде взаємодіяти з іншими об’єктами в грі (наприклад, з платформами, ворогами, стінами) [44].

Правильне налаштування цих елементів є ключовим для того, щоб персонаж міг нормально взаємодіяти з навколишнім середовищем, мати анімацію відповідно до дій і реагувати на фізичні впливи. Після налаштування цих вузлів ми можемо перейти до створення скриптів для управління рухом персонажа, а також додавання анімацій для різних станів гри.

# РОЗДІЛ 3
# ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ ГРИ ЖАНРУ ПЛАТФОРМЕР

## 3.1 Підготовка проєкту та розробка головного персонажа

Першочергово потрібно створити нову сцену, де основним вузлом буде CharacterBody2D. До нього необхідно додати та налаштувати вузол AnimatedSprite2D для реалізації анімацій персонажа, додавши відповідні анімаційні спрайти та налаштувавши швидкість, порядок відтворення кадрів і параметр зациклення. Також необхідно додати вузол CollisionShape2D для визначення області зіткнення та забезпеченння коректної фізичної взаємодії з майбутнім середовищем гри. У цій сцені буде зібрано всі елементи персонажа та забезпечено коректну роботу його складових, адже саме вона стане основою для створення керованого ігрового героя.

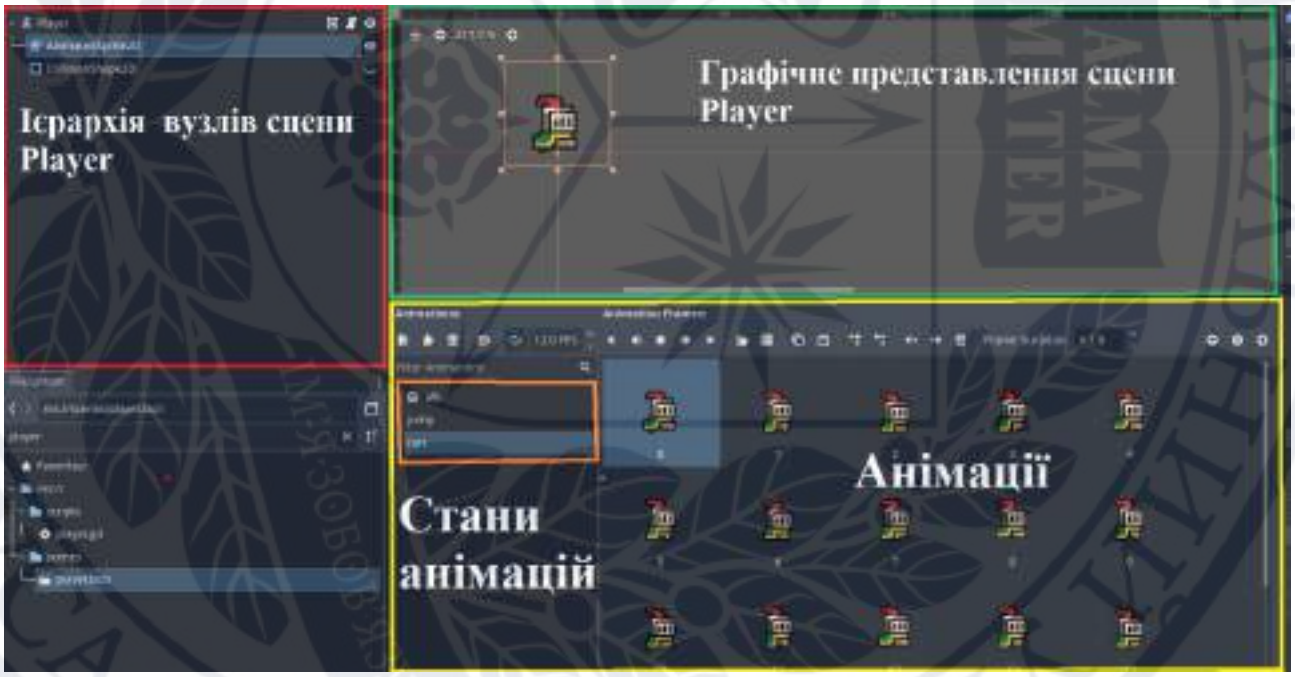


Рисунок 3.1 – Готова сцена Player, яка формує фінальний вигляд головного персонажа

Можна побачити, що для анімації персонажа було використано кілька станів: спокійний стан, стрибок і біг.

Головний персонаж майже готовий, залишилось лише зберегти

його сцену та додати її до головної сцени Game, яка є звичайним вузлом Node2D (рис. 3.2). Саме ця сцена буде використовуватись як основа для побудови рівня та миститиме всі об’єкти, механіки й інші елменти, що будуть розроблені у майбутньому для конкретного рівня гри.

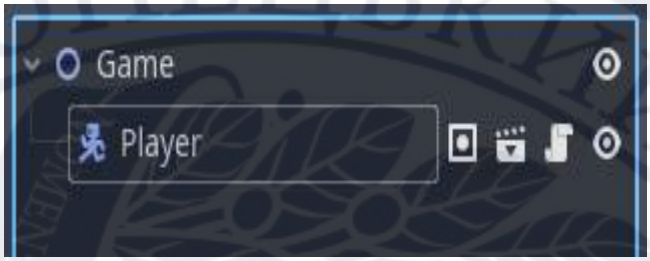


Рисунок 3.2 – Додавання сцени Player до головної сцени рівня Game

Також потрібно додати вузол Camera2D (рис. 3.3), щоб камера слідувала за рухами персонажа, забезпечуючи коректне відображення ігрового процесу.

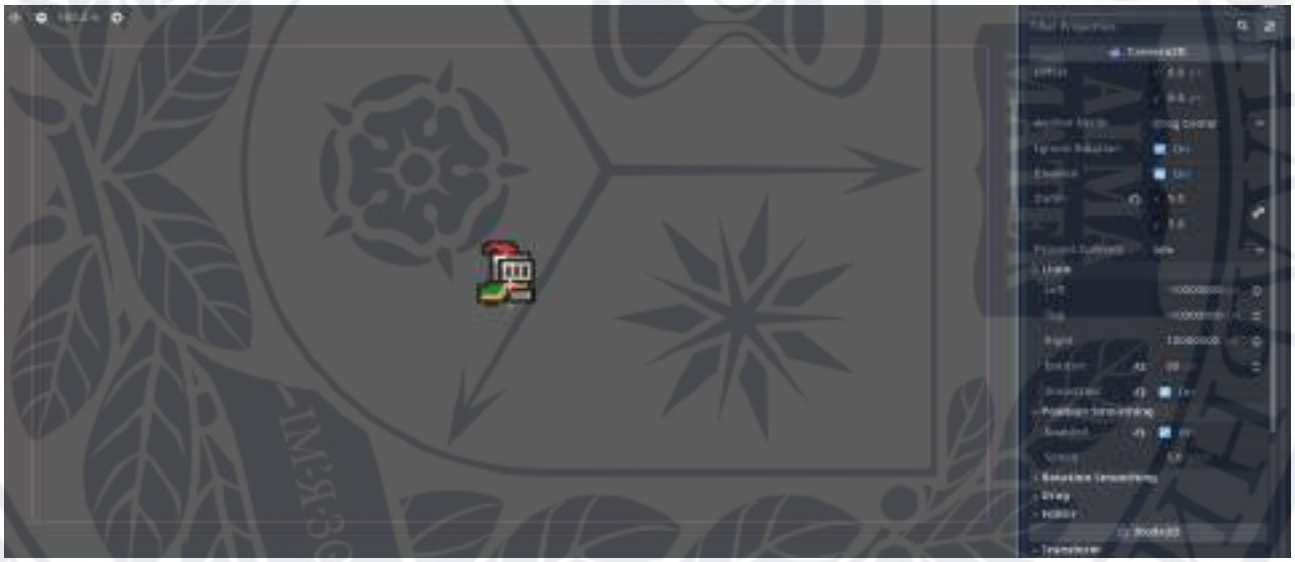

Рисунок 3.3 – Додавання вузла Camera2D до сцени Player, де фіолетовим кольором позначена область відображення ігрового процесу

Тепер, коли головний персонаж готовий, потрібно реалізувати його механіки (наприклад, швидкість руху), систему управління, а також налаштувати відтворення відповідних анімацій для різних станів. Для розробки даної функціональності було створено сприпт player.gd для сцени Player та зареєстровано клавіші управління в додаткових налаштуваннях проєкту. Встановлення швидкості та висоти стрибка персонажа, визначивши їх за

допомогою констант, щоб забезпечити бажану механіку руху наведено на рисунку 3.4.


Рисунок 3.4 – Встановлення швидкості та висоти стрибка персонажа, визначивши їх за допомогою констант, щоб забезпечити бажану механіку руху

Налаштування системи управління персонажем в InputMap наведено на рисунку 3.5.

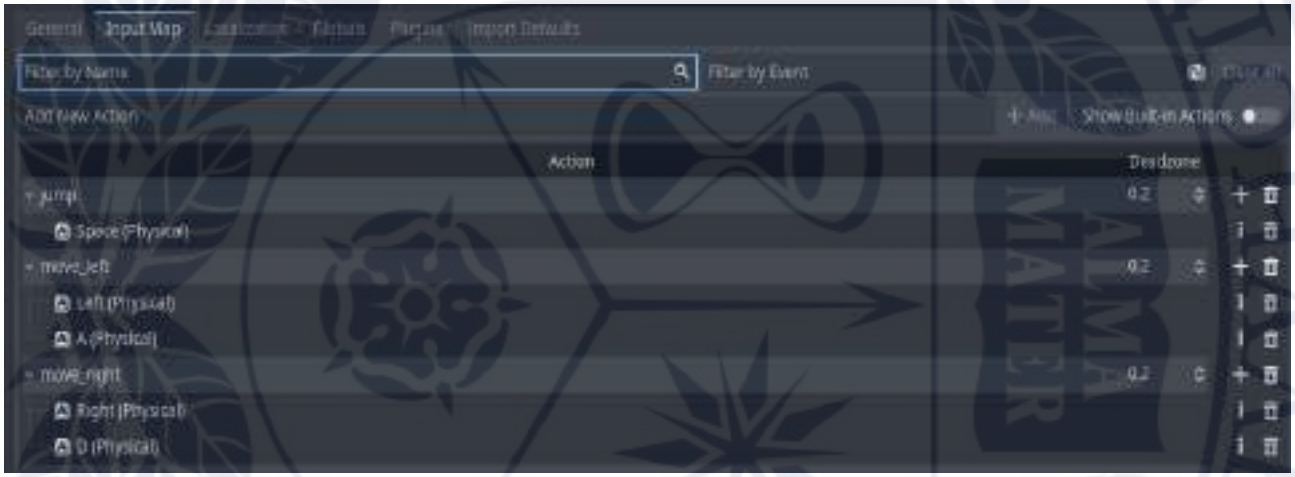


Рисунок 3.5 – Налаштовуємо систему управління персонажем в InputMap

Після визначення основних констант та налаштувань системи керування їх можна використати у скрипті для реалізації механік руху (додаток Б, рис. Б.1) та анімацій (додаток Б, рис. Б.2).

### 3.2 Побудова ігрового середовища та рівнів

Для побудови ігрового світу потрібно додати новий вузол TileMap та імпортувати в нього раніше завантажені візуальні ресурси.

Після імпорту, потрібно обрати певні тайли та встановити їм фізичні властивості, щоб персонаж міг по ним рухатись, взаємодіяти з ними та правильно відчувати зіткнення, забезпечуючи коректну фізику в грі (рис. 3.6).

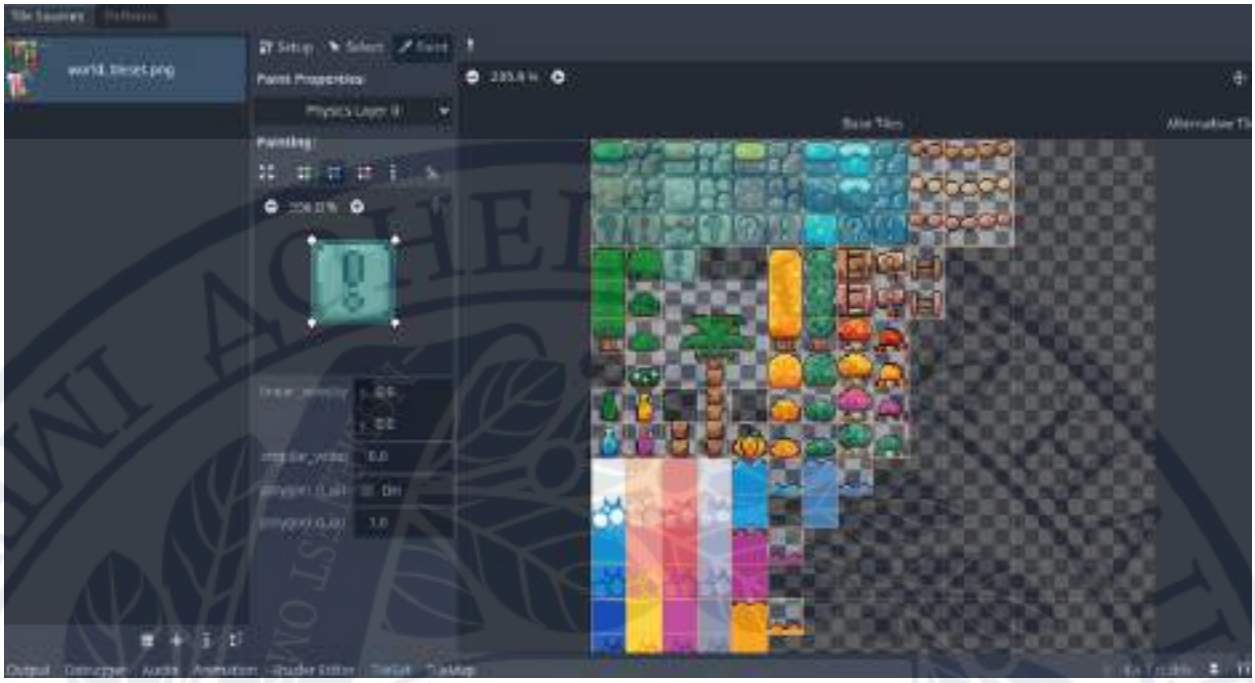

Рисунок 3.6 – Встановлення фізичних властивостей для конкретних тайлів

Налаштування фізичних властивостей для тайлів мосту, вручну окресливши зону зіткнення, наведено на рисунку 3.7.

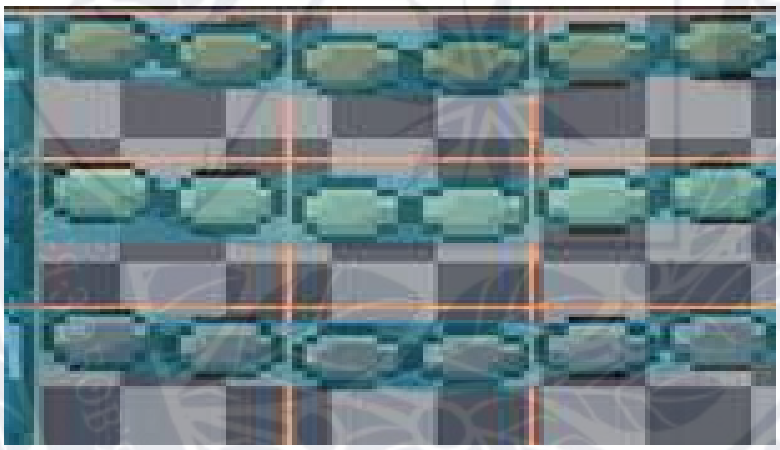

Рисунок 3.7 – Для тайлів мосту налаштовуємо фізичні властивості, вручну окресливши зону зіткнення

Це необхідно для того, щоб персонаж “не літав” над поверхнею, а коректно взаємодіяв із нею. Однією з основних частин геймплею моєї гри є платформи, тому одразу після тестування тайлів я перейду до їх розробки. У проєкті буде кілька типів платформ: ті, що рухаються горизонтально, ті, що рухаються

вертикально, а також статичні платформи.

Для створення ститичної платформи необхідно створити нову сцену, додати вузли AnimatableBody2D, Sprite2D та CollisionShape2D (рис. 3.8).


Рисунок 3.8 – Додаємо текстуру платформи до Sprite2D та визначаємо форму CollisionShape2D для фізичних налаштувань

Після розробки сцени статичної платформи, її можна здублювати та використати як фундамент для рухомих платформ, додатково додавши вузол AnimationPlayer. Встановлення швидкості руху та активація опції циклічності з використанням даних про розташування платформи наведено на рисунку 3.9.

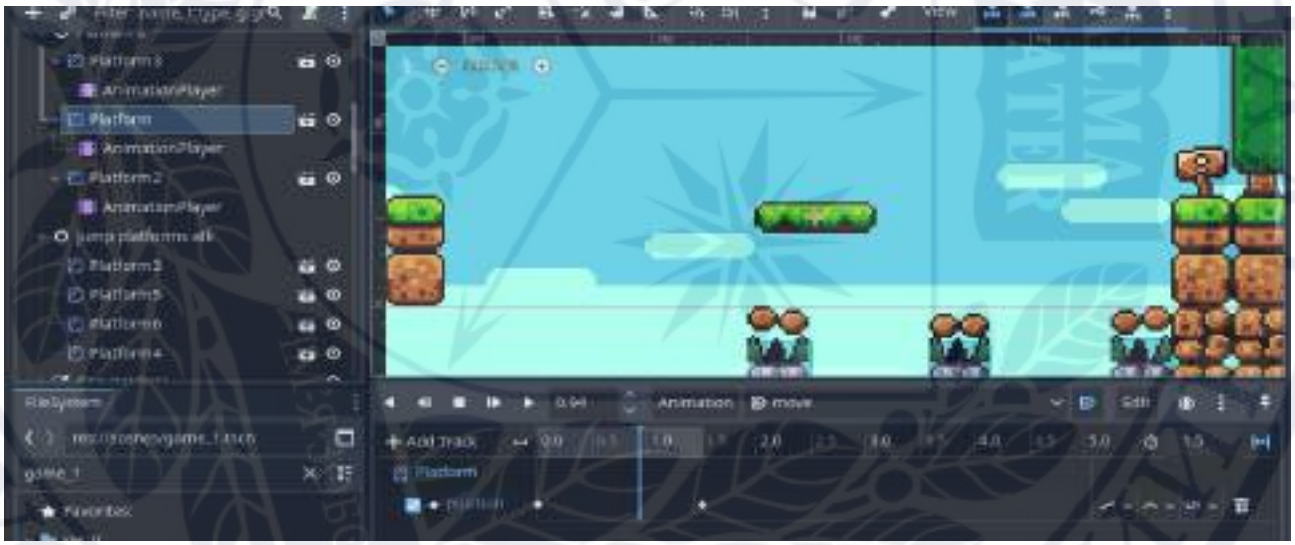

Рисунок 3.9 – Використовуємо дані про розташування платформи, вказуємо швидкість руху та активуємо опцію циклічності

Щоб зробити гру більш привабливою, було додано задній фон. Для цього в інтернеті було знайдено тайли  неба та використано їх як тло, розділивши ігровий простір за допомогою двох окремих вузлів TileMapLayer: один для основного геймплею (mainlayer), а інший – для заднього плану (background), у якому буде вимкнено колізії (рис. 3.10).

Рисунок 3.10 – Результат побудови заднього фону і платформ

Додавання фонового шару дозволило покращити візуальне сприйняття ігрового середовища та зробити рівень більш атмосферним.

Наступним етапом є додавання небезпечних зон, які створюватимуть загрозу для персонажа під час проходження рівнів. У моєму проєкті такими зонами є область падіння за межі карти, наприклад у лаву чи воду, а також шипи, розміщені безпосередньо на рівнях.

Була створена сцена небезпечної зони Killzone з кореневим вузлом Area2D. До кореневого вузла був доданий вузол CollisionShape2D, щоб визначити область, у яку при потраплянні персонаж загине, що дозволяє реалізувати логіку перезапуску рівня.

Рисунок 3.11 – Додавання нескінченної небезпечної зони до рівня, яка охоплює всю нижню частину


Рисунок 3.12 – Скрипт взаємодії небезпечної зони з персонажем

Потрапляння персонажа в область небезпечної зони викликає функцію die(), у межах якої реалізовується логіка загибелі персонажа та подальшого перезапуску рівня.

Після реалізації базової небезпечної зони було додано додатковий тип перешкод – шипи, які розміщуються безпосередньо на рівні та створюють локальні зони підвищеної небезпеки для гравця.

На відміну від глобальної зони падіння, шипи мають обмежену область дії та використовуються для ускладнення проходження окремих ділянок рівня. Аналогічно до фонового шару (Background), для реалізації шипів було створено окремий рівень TileMap під назвою Spikes, у який були додані відповідні тайли, знайдені на графічних вебресурсах. Логіка взаємодії шипів з гравцем аналогічна до раніше створеної небезпечної зони, але з іншою областю колізій: при потраплянні персонажа в межі цієї області відбувається його миттєва поразка з подальшим перезапуском рівня. Це забезпечує єдиний підхід до обробки небезпечних елементів середовища та спрощує реалізацію ігрової логіки.

Використання шипів дозволить підвищити складність рівнів, змушуючи гравця більш точно контролювати рух персонажа та уникати небезпечних ділянок (рис. 3.13).

Рисунок 3.13 – Реалізація додаткових небезпечних зон у вигляді шипів різних типів

Для реалізації механіки загибелі та відродження персонажа було створено окрему систему анімації та перезапуску рівня. Після взаємодії з небезпечною зоною запускається анімація загибелі та ефект затемнення екрана, після чого відбувається автоматичне перезавантаження поточного рівня.

Для покращення візуального сприйняття ігрового процесу ефект затемнення реалізовано за допомогою вузлів CanvasLayer та ColorRect, які забезпечують плавну зміну рівня прозорості чорного шару через Tween-анімацію, що створюється безпосередньо в програмному коді і використовується для поступової зміни певних властивостей вузлів, об’єктів або інтерфейсу протягом заданого проміжку часу [43] (рис. 3.14).

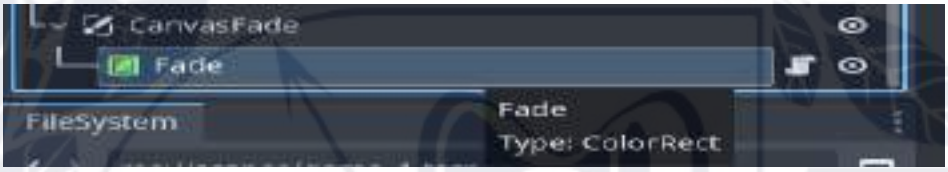


Рисунок 3.14 – Використання вузлів CanvasLayer та ColorRect для реалізації ефекту затемнення екрану

Під час загибелі персонажа додатково зупиняється його поточна анімація, тимчасово зменшується швидкість ігрового процесу та вимикається область зіткнення персонажа. Така реалізація дозволить зробити момент поразки більш плавним і зрозумілим для гравця, а також приховати можливі візуальні артефакти, які можуть виникати під час повторного завантаження сцени.

Вузол CanvasLayer використовується для відображення ефекту затемнення поверх усіх елементів сцени незалежно від положення камери. Усередині нього розміщується вузол ColorRect, який охоплює всю область екрана та забезпечує плавне затемнення шляхом зміни рівня прозорості під час анімації загибелі персонажа.

Скрипт fade.gd, що відповідає за плавне зникнення затемнення після завантаження сцени наведено на рисунку 3.15. Даний скрипт програмно реалізує плавне зникнення затемнення після завантаження сцени. Після запуску рівня

прозорість вузла ColorRect поступово змінюється від повністю непрозорого стану до прозорого, створюючи ефект поступової появи зображення на екрані.


Рисунок 3.15 – Скрипт fade.gd, що відповідає за плавне зникнення затемнення після завантаження сцени

Фрагмент програмного коду, що відповідає за запуск анімації загибелі головного персонажа, початковий ефект затемнення екрана та автоматичний перезапуск поточного рівня, наведено в додатку Б, рис. Б.3. Приклад спрацювання анімації загибелі персонажа після взаємодії з шипами показано на рисунку 3.16.

Рисунок 3.16 – Приклад спрацювання анімації загибелі персонажа після взаємодії з шипами

Отже, у межах даного підрозділу було реалізовано основні елементи ігрового середовища, включаючи структуру рівнів, платформи, небезпечні зони та систему загибелі й відродження персонажа. Реалізовані механіки формують базову основу ігрового процесу та забезпечують коректну взаємодію персонажа з навколишнім середовищем.

### 3.3 Ігрові механіки та інтерактивні об’єкти

У проєкті вже створено головного персонажа та базовий ігровий світ, який можна досліджувати. Однак наразі в грі бракує динаміки та взаємодії, тому необхідно додати інтерактивні об’єкти, бонусні механіки та елементи, які покращать ігровий процес. Для реалізації інтерактивних об’єктів як основний вузол використано Area2D, який працює як зона-тригер. На відміну від твердих перешкод, він не блокує рух головного персонажа, а лише фіксує момент, коли гравець перетинає його межі, та запускає відповідну ігрову логіку.


Рисунок 3.17 – Сцена coin

Сцена розміщення монет по ігровому середовищу приведена на рисунку 3.18.

Рисунок 3.18 – Розміщення монет по ігровому середовищу

Для покращення ігрового процесу та підвищення зацікавленості гравця буде додано ще один тип об’єктів, які надаватимуть спеціальні бонуси на певний період часу. Передбачено три види таких об’єктів: перший надає можливість подвійного стрибка (рис. 3.19), другий – підвищує швидкість пересування персонажа, а третій – дозволяє відштовхуватись від стін. Використання цих

бонусів розширює ігрові можливості та забезпечує проходження рівнів різної складності й типу.


Рисунок 3.19 – Приклад сцени DoubleJump, за допомогою якої реалізовано подвійний стрибок

Аналогічно до сцени DoubleJump були створені сцени Speed (додаток В, рис. В.1) та WallJump (додаток В, рис. В.2) для реалізації бонусів збільшення швидкості та відштовхування від стін.

Програмну реалізацію бонусних механік подвійного стрибка, відштовхування від стін та додаткової системи наведено у додатку Б, рис. Б.4.

Додатково було додано механіку coyote time, яка дозволяє персонажу виконати стрибок протягом короткого проміжку часу після сходження з твердої поверхні.

На відміну від механік подвійного стрибка та відштовхування від стін, для реалізації яких використовується система додаткових перевірок у логіці стрибків персонажа, механіка прискорення реалізована значно простіше – шляхом збільшення значення змінної швидкості руху персонажа. Функції активації бонусних механік зображено у додатку Б, рис. Б.5.

Після реалізації базових інтерактивних об’єктів та бонусних механік наступним етапом є створення системи переходу між рівнями. Для цього у проєкті буде реалізовано спеціальний портал, який виконуватиме роль точки переходу до наступного рівня. При взаємодії персонажа з порталом відбуватиметься автоматична зміна поточного рівня, що забезпечує безперервність ігрового процесу та навігацію між різними локаціями.

Для побудови порталу буде створено окрему сцену, аналогічно до монет і бонусних об’єктів, що вже були реалізовані раніше, але з функціоналом переходу гравця до наступного рівня.

Для коректної роботи механіки переходу, головного персонажа необхідно додати до окремої групи, що дозволить обмежити взаємодію порталу лише з керованим персонажем. Сцена порталу NextLevel та додавання порталу і головного персонажа до групи “Player” представлена на рисунку 3.20.

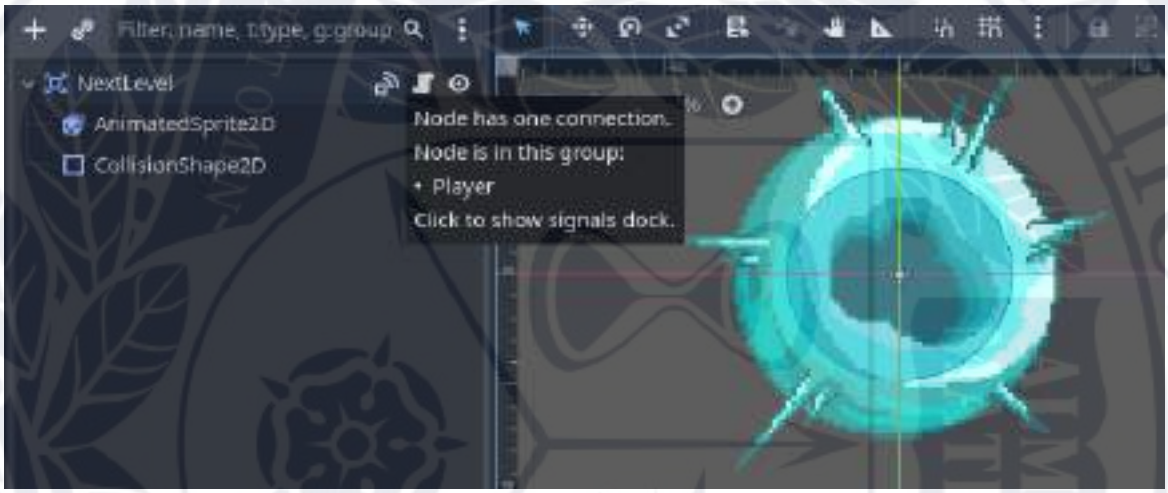


Рисунок 3.20 – Сцена порталу NextLevel та додавання порталу і головного персонажа до групи “Player”

Для сцени порталу потрібно створити скрипт next_level.gd, який відповідатиме за перевірку умов активації та подальший перехід до наступного рівня. У моєму проєкті перехід буде можливий лише після збору всіх 20 монет, розміщених на рівні. Якщо необхідну кількість монет не зібрано, портал залишатиметься неактивним.

Такий підхід буде стимулювати гравця до повного дослідження рівня, взаємодіяти з інтерактивними об’єктами та виконувати поставлені ігрові завдання перед переходом до наступної локації.

У скрипті next_level.gd (додаток Б, рис. Б.6) перевіряється, чи належить об’єкт до групи Player та чи виконано умову збору необхідної кількості монет.

Після цього визначається номер поточного рівня, формується шлях до наступної сцени та виконується перехід між рівнями.

Крім того, у скрипті реалізовано базову систему збереження прогресу, яка дозволяє зберігати інформацію про відкритий рівень перед переходом до наступної сцени.

Для створення наступних рівнів використовувалось дублювання попередньо розроблених та налаштованих сцен, що містили необхідні компоненти, програмну логіку та механіки.

Перевірка кількості зібраних монет під час проходження рівня здійснюється за допомогою скрипта game_manager.gd, який відповідає за підрахунок монет та оновлення ігрових параметрів у реальному часі (додаток Б, рис. Б.7). У майбутньому цей скрипт буде використано під час розробки користувацького інтерфейсу, що відображатиме кількість зібраних монет.

Для розширення інтерактивності ігрового процесу у проєкті також реалізовано систему динамічних блоків, які відіграють роль дверей та дозволяють автоматично відкривати або закривати окремі проходи після взаємодії персонажа зі спеціальною областю активації.

Для реалізації даної механіки використовується вузол Area2D, який містить дочірній вузол CollisionShape2D для визначення області активації, а також вузол Timer із затримкою 0.6 с., що забезпечує плавне спрацювання механізму.

У скрипті open_blocks.gd (додаток Б, рис. Б.8) після потрапляння персонажа в область активації запускається таймер, після завершення якого відповідний шар TileMapLayer приховується та вимикається його колізія. Це дозволяє автоматично відкрити прохід до наступної частини рівня.

Аналогічним чином реалізовано механізм закриття проходу, який використовується для тимчасового блокування окремих ділянок рівня.

У скрипті CloseBlocks.gd (додаток Б, рис. Б.9) після активації таймера відповідний шар TileMapLayer стає видимим, а його колізія знову вмикається, що призводить до блокування проходу (рис. 3.21).

Рисунок 3.21 – Приклад використання механізму блокування проходу та допоміжної зони AntiBugKillZone

У даному прикладі реалізовано взаємодію одразу між кількома механізмами. Після виходу гравця із зони активації одна частина проходу автоматично закривається за допомогою CloseBlocks, обмежуючи можливість повернення назад. Одночасно запускається таймер механізму OpenBlocks, після завершення якого відкривається новий прохід до наступної частини рівня. Додатково активується зона AntiBugKillZone, яка запобігає виходу персонажа за межі передбаченої ігрової області та забезпечує коректне проходження рівня.

Такий підхід дозволяє створювати послідовні інтерактивні сценарії проходження, керувати переміщенням гравця та формувати ізольовані ігрові зони. При цьому розроблені механізми можуть використовуватись як у взаємозв’язку між собою, так і окремо залежно від структури рівня та необхідної ігрової логіки.

Після реалізації інтерактивних об’єктів, бонусних механік, порталів і динамічних проходів наступним етапом стало додавання ворогів. Вони створюють додаткові перешкоди для гравця, підвищують складність проходження рівнів та роблять ігровий процес більш динамічним. Аналогічно до головного персонажа, усі противники у проєкті мають власні анімації руху та поведінку, що забезпечує унікальність ворогів та цілісність візуального стилю гри. У проєкті реалізовано декілька типів противників: слайм, звичайні бджоли, бос-слайм, бос-бджола та ворог-стрілець.

Слайм є базовим наземним ворогом, який переміщується вліво та вправо в межах визначеної ділянки рівня. Сцена цього ворога наведена у додатку В, на рис. В.3.

У скрипті slime.gd (додаток Б, рис. Б.10) реалізовано рух ворога по горизонталі та перевірку зіткнення зі стінами за допомогою RayCast2D. Якщо ворог наближається до перешкоди, напрямок його руху змінюється, а спрайт віддзеркалюється відповідно до нового напрямку.

Рисунок 3.22 – Реалізація ворога Slime у грі

Окрім наземних ворогів, у грі реалізовано літаючих противників у вигляді бджіл (рис. 3.23). Вони рухаються в повітрі за заданою траєкторією, що дозволяє створювати додаткові перешкоди на різних ділянках рівня. Сцена звичайної бджоли зображена у додатку В, рис. В. 4.

Програмний код, що реалізує горизонтальний рух бджоли між двома точками, наведено у додатку Б, рис. Б. 11. Після досягнення встановленої відстані напрямок руху змінюється на протилежний. Аналогічно були реалізовані інші варіанти бджіл, які можуть рухатись за іншими траєкторіями, зокрема вертикально.

Для ускладнення проходження було реалізовано боса-бджолу. На відміну від звичаної бджоли, цей ворог має значно більший розмір, підвищену небезпеку та активується після наближення гравця на визначену відстань. Після активації бос починає переслідувати головного персонажа, створюючи постійний тиск на гравця та змушуючи його швидко реагувати під час проходження рівня. Сцена бджоли-боса наведена у додатку В, на рис. В. 5, а фрагмент програмної реалізації поведінки боса – у додатку Б, на рис. Б. 12.

Рисунок 3.23 – Реалізація звичайних літаючих ворогів у грі

У скрипті перевіряється відстань між босом і персонажем. Якщо гравець наближається на відстань активації, ворог переходить в активний стан і починає рухатись в напрямку персонажа. Особливістю цього ворога є можливість переслідувати гравця крізь елементи ігрового середовища, оскільки він не взаємодіє з перешкодами. Це робить ворога більш небезпечним та змушує гравця постійно рухатись.

Рисунок 3.24 – Реалізація переслідування гравця босом-бджолою у грі

На відміну від боса-бджоли, який переслідує гравця у відкритому просторі, наступним складним противником у проєкті став бос-слайм. Даний ворог є ускладненим варіантом наземного противника, який поєднує механіку переслідування персонажа зі стрибками та активною взаємодією з ігровим середовищем. Бос-слайм використовується у спеціальному замкнутому рівні з

механікою дверей та ізоляції ігрової зони (рис. 3.25). Після входу персонажу в кімнату проходи автоматично закриваються, а сам бос активується і починає рухатись у бік гравця та періодично виконує стрибки. Для проходження даної частини рівня необхідно протриматись певний час і виконати поставлені ігрові умови, після чого двері відкриваються та дозволять продовжити проходження. Сцена бос-слайма наведена у додатку В, рис. В.6. Фрагмент коду boss_slime.gd для активації, стрибків та переслідування гравця наведено у додатку Б, рис. Б.13.

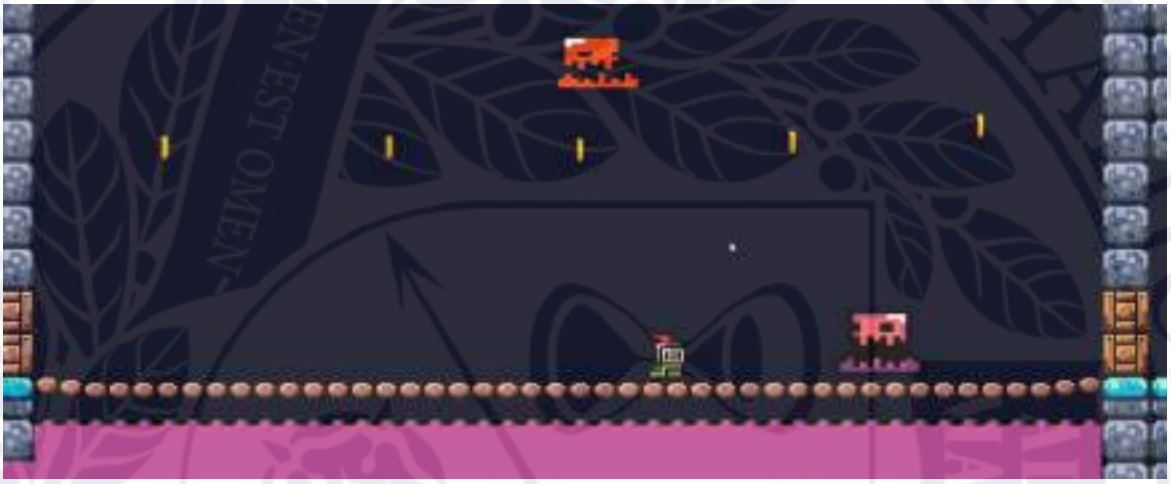

Рисунок 3.25 – Реалізація фрагменту рівня з босом-слаймом

Останнім типом противника у проєкті є ворог-стрілець, який відрізняється від попередніх ворогів наявністю дистанційної атаки. Його основна особливість полягає у створенні куль, які рухаються в напрямку персонажа та змушують гравця використовувати конкретні елементи середовища для захисту.

На відміну від інших ворогів у грі, ворог-стрілець має унікальну і складнішу структуру, оскільки його реалізація розділена на дві окремі сцени: безпосередньо самого стрільця (додаток В, рис. В.7) та об’єкта кулі (додаток В, рис. В.8). Сцена ворога побудована за чіткою ієрархією: основний вузол gun_boss (Node2D) містить персонажа, анімаційний спрайт, окремий спрайт зброї, точку появи кулі gun_point (Marker2D) та таймер стрільби. Така модульна структура дозволяє незалежно керувати зовнішнім виглядом ворога, напрямком його зброї та моментом створення снаряда.

У скрипті gun_boss.gd (додаток Б, рис. Б.14) реалізовано механіку активації

ворога після наближення персонажа. Після входу персонажа в зону виявлення запускається таймер стрільби, а напрямок спрайтів ворога та його зброї автоматично розвертається в бік гравця. Щоразу при спрацюванні таймера код скрипта звертається до сцени Bullet та генерує кулю у точці gun_point, після чого запущена куля починає рух у напрямку головного персонажа.

Автономна сцена кулі складається з кореневого вузла Killzone, а також дочірніх вузлів – візуального спрайту та області зіткнення. Завдяки такій структурі при потраплянні персонажа в зону колізії снаряда він помирає.

У скрипті bullet.gd (додаток Б, рис. Б.15) реалізовано рух кулі у напрямку персонажа, перевірку зіткнення зі спеціальним шаром anti_bullet, реалізованим за допомогою вузла TileMapLayer, а також взаємодію з гравцем. При контакті зі спеціальними блоками куля автоматично видаляється, а при зіткненні з персонажем викликається функція die(). Використання шару anti_bullet дозволяє створювати укриття та урізноманітнювати проходження рівнів, в яких присутні дистанційні атаки.

Рисунок 3.26 – Реалізація дистанційної атаки ворога-стрільця та захисного шару anti_bullet у грі

Реалізація ворога-стрільця та системи дистанційних атак додатково дозволила збільшити ігровий досвід, додати нові способи взаємодії з ігровим середовищем і створити механіку укриттів.

Отже, у межах даного підрозділу було реалізовано інтерактивні об’єкти, бонусні механіки, систему переходів між рівнями, динамічні елементи середовища та різні типи ворогів, що дозволило сформувати майже повноцінний ігровий процес та забезпечити різноманітність проходження рівнів.

### 3.4 Користувацький інтерфейс, аудіосупровід та додаткові системи.

Після реалізації основних ігрових механік, інтерактивних об’єктів та системи ворогів завершальним етапом розробки стало додавання користувацького інтерфейсу, звукових ефектів, музики та додаткових систем, які покращать взаємодію гравця з грою і зроблять проєкт повноцінним. Дані елементи відповідають за відображення ігрової інформації, візуальні та звукові ефекти, збереження прогресу, а також формують загальну атмосферу гри та завершеність проєкту.

Після того як було підібрано відповідні звукові аудіофайли, які гармонійно підходять до атмосфери моєї гри, я імпортую їх до проєкту у створену для цього папку sounds. Після імпорту я перейду до налаштування звуку безпосередньо в ігровому середовищі, використовуючи вузол AudioStreamPlayer2D, який дозволяє керувати різноманітними параметрами звуку.


Рисунок 3.27 – Додавання фонової музики до гри

Аналогічно, додамо звукові ефекти, які будуть відтворюватися, коли персонаж збирає монету або бонус. Після цього перейдемо до налаштувань

гучності звукових ефектів і музики, щоб забезпечити комфортний і збалансований звук у грі.

Оскільки музичний супровід у грі має фоновий і циклічний характер, додаткові налаштування для його відтворення не потрібні. Натомість для монет та інших інтерактивних об’єктів необхідно реалізувати відповідний сценарій, який забезпечуватиме відтворення звуку під час їх підбору. Після взаємодії з гравцем об’єкт має зникати та більше не бути доступним для повторного використання. Зазначена логіка реалізується у декількох скриптах за допомогою додаткового вузла AnimationPlayer (рис. 3.28). Його необхідно додати до сцен тих об’єктів, які повинні відтворювати звуковий ефект та зникати після контакту з головним персонажем.

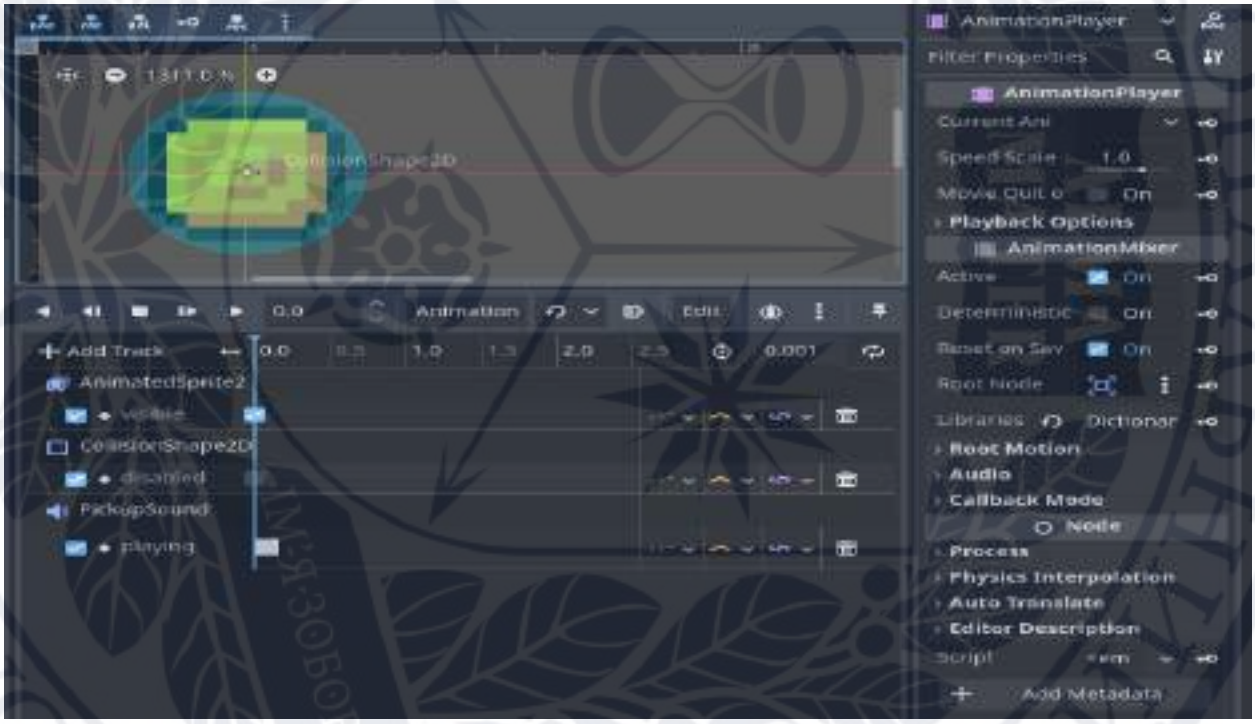


Рисунок 3.28 – Приклад використання AnimationPlayer для створення сценарію pickup

Для відображення кількості зібраних монет у грі використовується система підрахунку, реалізована за допомогою скрипта game_manager.gd, який було розглянуто у попередньому розділі. Для виведення інформації на екран до основної сцени рівня додається вузол CanvasLayer, а всередині нього – вузол

Label з назвою ScoreLabel. Даний елемент використовується для відображення кількості зібраних монет під час проходження рівня. Для кращого візуального оформлення інтерфейсу було використано спеціальний знайдений в інтернеті шрифт, підібраний відповідно до стилістики гри. Аналогічний шрифт також буде використовуватись під час розробки інших текстових компонентів гри.


Рисунок 3.29 – Скрипт coin.gd

На рисунку наведено скрипт coin.gd, у якому реалізовано взаємодію монети з персонажем, запуск анімації pickup та оновлення лічильника монет за допомогою game_manager.gd. Під час відтворення анімації також виконується програвання звукового ефекту та подальше видалення об’єкта монети зі сцени.


Рисунок 3.30 – Результат розробки лічильника у лівій верхній частині екрана під час ігрового процесу

Аналогічний підхід використовується і для бонусних об’єктів. Після взаємодії з персонажем бонус активує відповідну механіку та відтворює анімацію підбору, однак на відміну від монет не використовує GameManager та додаткові вузли для відображення, оскільки бонуси не беруть участі у системі прогресу

рівня та не потребують окремого підрахунку. Натомість скрипт бонуса безпосередньо звертається до player.gd, активуючи відповідну функцію персонажа, яка відповідає за надання нової здібності. Фрагмент скрипта одного з бонусних об'єктів на прикладі подвійного стрибка наведено в додатку Б, рис. Б.16.

Окрім базового внутрішньоігрового інтерфейсу, важливою складовою проєкту є головне меню, яке забезпечує взаємодію користувача з основними функціями гри. У моєму проєкті головне меню містить кнопки запуску гри, вибору рівнів та виходу з програми, реалізованих за допомогою вузла Button а також фонове відео, створене у програмі для монтажу на основі фрагментів проходження гри. Вузол VideoStreamPlayer дозволив зробити меню більш динамічним та візуально привабливим, створюючи ефект короткого трейлера гри ще до початку проходження. Сцену головного меню наведено в додатку В, рис. В.9, а скрипт menu.gd, що відповідає за реалізацію його функціоналу – у додатку Б, рис. Б.17.

У скрипті menu.gd реалізовано завантаження збереженого прогресу за допомогою SaveManager, запуск останнього відкритого рівня, перехід до сцени вибору рівнів та завершення роботи гри. Використання окремих функцій для кожної кнопки дозволяє забезпечити зручну взаємодію користувача з головним меню.

Оскільки головне меню використовує систему збереження прогресу, у проєкті було реалізовано окремий скрипт SaveManager.gd (додаток Б, рис. Б.18), який відповідає за збереження та завантаження основних ігрових параметрів.

У даному скрипті реалізовано збереження максимально відкритого під час проходження гри рівня та кількості смертей гравця у файл формату .save. Під час запуску гри SaveManager автоматично перевіряє наявність файлу на збереження та завантажує необхідні дані. Крім того, скрипт містить функції оновлення прогресу після проходження рівнів і підрахунку кількості смертей персонажа. Завдяки цьому інформація про прогрес гравця зберігається навіть після закриття та повторного запуску гри.


Рисунок 3.31 – Результат розробки головного меню гри


Рисунок 3.32 – Результат реалізації лічильника смертей у лівій нижній частині екрану під час ігрового процесу

Окрім запуску гри та завершення роботи програми, у головному меню також реалізовано кнопку Levels, яка дозволяє перейти до окремої сцени вибору рівнів. Дана система дає можливість повторно проходити вже відкриті рівні, а також обмежує доступ до наступних рівнів, якщо попередні ще не були пройдені. Сцену вибору рівнів наведено у додатку В, рис. В. 10.

Для реалізації системи вибору рівнів було створено окремий скрипт level_select.gd (додаток Б, рис. Б.19), який відповідає за доступ до всіх рівнів гри.

У скрипті підключаються кнопки вибору рівнів, після чого за допомогою SaveManager перевіряється максимально відкритий рівень. Якщо певний рівень

ще не пройдено, відповідна кнопка автоматично стає недоступною для натискання. Після вибору доступного рівня виконується перехід до відповідної ігрової сцени.


Рисунок 3.33 – Результат реалізації меню вибору рівнів у грі

Окрім головного меню та вибору рівнів, у проєкті також реалізовано сцену PauseMenu (додаток В, рис. В.11), яка дозволяє швидко отримати доступ до основних функцій гри без виходу з поточного рівня. На відміну від головного меню, даний інтерфейс накладається поверх ігрової сцени в реальному часі за допомогою CanvasLayer, залишаючи ігровий процес видимим на задньому плані.

Для реалізації сцени паузи було створено окремий скрипт pause_menu.gd (додаток Б, рис. Б.20), який відповідає за її функціонал.

Розроблений скрипт реалізував відкриття та закриття меню після натискання клавіші Esc, яка була попередньо прив’язана до дії pause аналогічно до системи керування персонажем. Також у скрипті реалізовано обробку кнопок Resume, MainMenu та Exit. Використання CanvasLayer дозволяє відображати

меню поверх гри без необхідності переходу між сценами.


Рисунок 3.34 – Вигляд меню паузи під час гри

Для спрощення взаємодії гравця з ігровим середовищем у проєкті також реалізовано систему текстових підказок та інформаційних повідомлень за допомогою вузлів Label. Дані елементи допомагають ознайомитись з основними механіками гри, пояснюють призначення бонусів та спрощують орієнтування під час проходження рівнів.

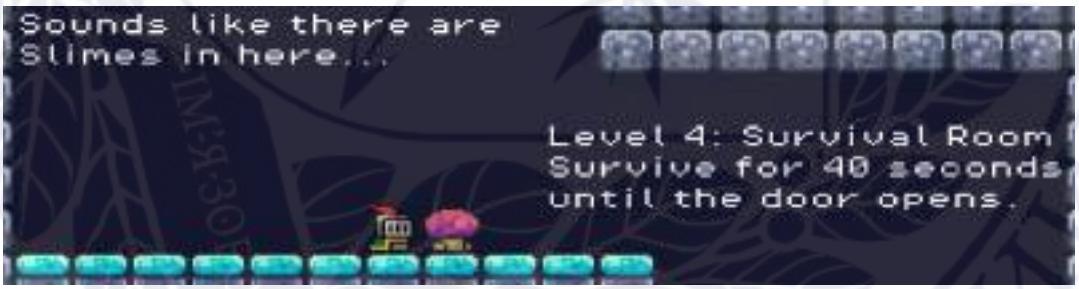


Рисунок 3.35 – Приклад назви рівня та інформаційних повідомлень під час проходження гри

Окрім інформаційних повідомлень і підказок, у проєкті також реалізовано просту сюжетну складову, яка супроводжує гравця під час проходження рівнів.

Основою сюжету є переслідування кота, який постійно переміщується між різними локаціями та направляє гравця до нових частин ігрового світу. Під час взаємодії з котом на екрані також відображаються короткі текстові повідомлення,

які допомагають передати сюжетний контекст та підтримують атмосферу гри. У різних частинах гри використовується декілька персонажів кота з різною поведінкою: одні персонажі рухаються вліво або вправо після наближення гравця, інші зникають після досягнення певної точки або взаємодії з порталом. Сцена кота наведена у додатках В, рис. В.12.

У проєкті використовується поєднання декількох типів тригерних зон і персонажів кота, які разом запускають різного типу сценарії поведінки. Один із прикладів реалізації тригерної зони наведено в додатку Б, рис. Б.21, а пов’язаний із нею скрипт cat2.gd, що відповідає за рух і зникнення кота – у додатку Б, рис. Б.22.

У наведеній реалізації передбачено запуск руху кота вліво після взаємодії із зоною-тригером, зміну орієнтації спрайта відповідно до напрямку переміщення та поступове зникнення персонажа після досянення заданої точки.

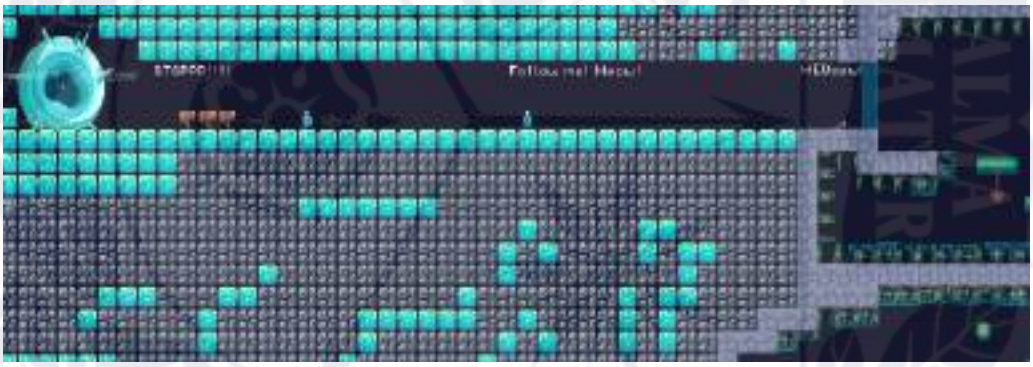

Рисунок 3.36 – Розміщення кота, зони активації та сюжетних реплік у середовищі розробки Godot

Реалізація переслідування та зникнення кота під час проходження рівня показана на рисунку 3.37.

Таким чином, використання сюжетних реплік, текстових підказок та персонажа кота дозволило поєднати навігацію гравця з елементами сюжетного супроводу, що зробило проходження рівнів більш послідовним, зрозумілим, та атмосферним.

Після проходження основної частини гри реалізовано фінальний ігровий рівень, у якому гравець повинен використати всі отримані навички та бонусні

механіки для завершення проходження. Даний рівень об’єднує раніше вивчені механіки руху, взаємодії з бонусами та проходження небезпечних ділянок. Графічне представлення рівня наведено у додатку А, рис. А.8.


Рисунок 3.37 – Реалізація переслідування та зникнення кота під час проходження рівня

Після завершення фінального рівня у проєкті передбачено декілька сюжетних сцен без небезпечних елементів. Їхньою основною метою є логічне завершення сюжетної складової гри та створення плавного переходу до титрів.


Рисунок 3.38 – Приклад першої сюжетної сцени з появою батька кота

У завершальній сцені поступово з’являються брати та сестри кота. Для реалізації даного епізоду було модифіковано поведінку персонажів: після проходження певної частини шляху коти тимчасово зупиняються та повертаються у бік гравця, а після входу персонажа в зону активації знову продовжують рух у напрямку порталу. Така логіка поведінки дозволила створити більш живу та послідовну сцену, показуючи, що коти чекають на гравця, а не тікають.


Рисунок 3.39 – Реалізація фінальної сюжетної сцени з очікуванням гравця біля порталу

Після завершення фінальної сюжетної сцени у грі реалізовано титри, які використовуються як заключний елемент проходження та завершують сюжетну складову проєкту. У титрах відображається діалог між головним персонажем та його котом, який пояснює основну ідею подорожі гравця та підводить сюжет до завершення.

Сцена титрів (додаток В, рис. В.13) побудована на основі вузлів CanvasLayer, Control, RichTextLabel, ColorRect та Label. RichTextLabel використовується для відображення основного тексту титрів, ColorRect – для реалізації ефектів затемнення екрана, а Label – для виведення повідомлення про можливість повернення до головного меню після завершення титрів (рис. 3 40).

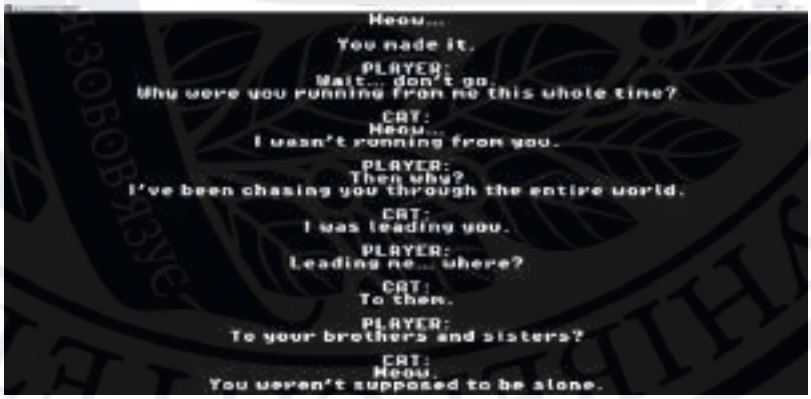


Рисунок 3.40 – Результат відображення титрів після завершення гри

Для реалізації даної сцени було створено скрипт ending.gd (додаток Б, рис. Б.23), який забезпечує автоматичний рух тексту титрів угору, початковий ефект

появи сцени за допомогою затемнення та відображення повідомлення “Натисніть будь-яку клавішу для продовження” після завершення титрів. Такий підхід дозволить гравцеві плавно перейти до головного меню.


Рисунок 3.41– Відображення повідомлення Press Any Key після завершення титрів

Таким чином, фінальна сцена та система титрів забезпечили логічне завершення сюжетної складової гри, об’єднали основні події проходження та створили завершальний фінал проєкту.

Після завершення розробки, всі основні системи, компоненти і механіки гри були повністю інтегровані в єдиний проєкт. Завершальним етапом стало налаштування експорту гри для операційної системи Windows за допомогою інструментів рушія Godot.

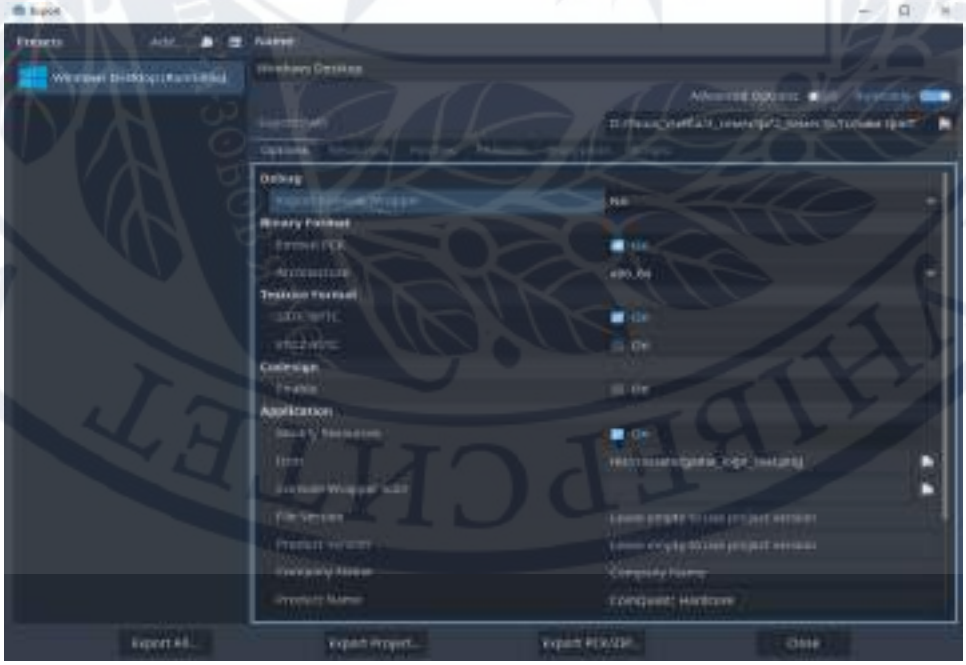

Рисунок 3.42 – Налаштування експорту ігрового проєкту у середовищі Godot

Під час налаштування експорту було обрано платформу Windows Desktop, архітектуру x86_64, а також активовано Binary Format: Embed PCK, що дозволяє вбудувати ресурси гри безпосередньо у виконуваний файл .exe. Для коректного відображення текстур було використано формат S3TC/BPTC, що дозволило суттєво зменшити споживання відеопам'яті та оптимізувати програму завдяки апаратному стисненню графіки без помітної втрати її якості.

Крім того, для гри було створено власну іконку, яка використовується для виконуваного файлу .exe, відображається у верхній частинії вікна гри та на панелі відкритих програм операційної системи Windows. Під час експорту проєкту грі також було надано назву CoinQuest: Hardcore, яка використовується у назві виконуваного файлу та відображається у вікні запущеної гри (рис. 3.42). Для коректного застосування іконки під час експорту було використано параметр Modify Resources, а також додатковий інструмент rcedit, який інтегрується з Godot Engine та дозволяє змінювати ресурси виконуваного файлу Windows.


Рисунок 3.43 – Результат експорту гри у форматі .exe

Після завершення налаштування параметрів проєкт було успішно експортовано у готовий виконувальний файл, який можна запускати без необхідності встановлення середовища розробки Godot. Результати тестування підтвердили коректну роботу ігрових механік, реалізованих систем, користувацького інтерфейсу та інших компонентів проєкту.

# ВИСНОВКИ

У межах цієї роботи було проаналізовано і досліджено не лише особливості готових ігрових проєктів, а й ключові аспекти їх створення. Результатом дослідження є повноцінний ігровий продукт у жанрі 2D-платформер, який об’єднує всі необхідні системи, компоненти та характерні механіки цього жанру. За результатами проведеного дослідження можна зробити наступні висновки:

1. Аналіз актуальності розробки та існуючих успішних аналогів дозволив визначити ключові особливості жанру 2D-платформерів, якими є поєднання динамічних механік руху, подолання перешкод, взаємодія з ігровим середовищем та інтерактивними об’єктами, різноманітність піджанрів, а також стиль й атмосферне оформлення. Такий аналіз дозволяє чітко сформувати ідею гри ще до початку роботи в середовищі розробки. Його результати лягли в основу проєкту CoinQuest: Hardcore – 2D-платформера з акцентом на точність керування, взаємодію з оточенням, піксельну стилістику та підвищену складність ігрового процесуДослідження теоретичних основ розробки відеоігор, їх класифікації, жанрів та платформ поширення дозволило зрозуміти базову термінологію й чітко визначити критерії проєктування обраного типу ігрового проєкту. Це забезпечило вільне орієнтування у принципах ігрової розробки та архітектурі ігрових рушіїв. За основним типом платформ запуску відеоігри поділяються на персональні комп’ютери, консолі та мобільні пристрої. Статистичні дані за 2025 рік свідчать про домінування ПК-платформ, які займають найбільшу частку ринку – 38,5%, тоді як консолі становлять 32,4%, а мобільний сегмент – 26,1%. Такий розподіл підтверджує найвищий рівень популярності та перспективності саме комп’ютерних ігор у сучасній індустрії.

Цифрові сервіси поширення відеоігор відкрили незалежним розробникам доступ до широкої аудиторії без необхідності видання фізичного продукту. Особливе місце серед них посідає Steam, який є найпопулярнішою платформою для комп’ютерних ігор у світі та налічує понад 132 мільйони активних користувачів щомісяця. Певна статистика цього сервісу підтверджує стійку

популярність жанру 2D-платформерів із загальною кількістю понад 6500 ігор, проте лише 12% із них здобувають значну популярність через високий рівень конкуренції на ринку.

2. Вибір ігрового рушія Godot Engine обґрунтований тим, що він є безкоштовним, зручним, ефективним та повністю готовим середовищем для 2D-розробки з відкритим кодом і низькими системними вимогами. Аналіз його технологій, зокрема системи сцен та вузлів, мови програмування GDScript, спеціалізованих інструментів, властивостей та унікальних для кожного вузла налаштувань довів, що рушій надає інді-розробникам повноцінний набір засобів для швидкого створення якісних ігрових проєктів, через що його популярність стабільно зростає. Перед початком безпосередньої розробки гри потрібно підготувати проєкт, що включає пошук і підбір необхідних графічних та аудіоресурсів, а також організацію структури папок у середовищі Godot.

3. У процесі розробки створено головного персонажа та його графічне відображення, а також реалізовано систему керування, фізичну взаємодію з навколишнім середовищем, базові механіки руху та зміну станів анімації, що забезпечило формування цілісного ігрового об’єкта.

4. Проєктування ігрових рівнів та систем прогресії є фундаментальним етапом розробки, оскільки саме через структуру локацій та первинних механік розвитку відбувається знайомство користувача з грою і формування його подальшого ігрового досвіду. Основними системами прогресії виступають рівнева система, дерево навичок, сюжетна прогресія та механізм поступового ускладнення ігрового процесу.

5. Під час проєктування ігрових рівнів потрібно враховувати криву складності, яка балансує між навичками гравця та викликами середовища для підтримання зацікавленості. Ітеративний підхід під час розробки ігор також є дуже важливим процесом, оскільки циклічне проєктування, програмування та тестування дозволяє своєчасно усувати помилки, а також забезпечувати поступове підвищення стабільності гри та досягнення високої якості фінального продукту.  Побудована систему рівнів включає платформи різних типів і

небезпечні зони. Також реалізовано механіку смерті й відродження персонажа, яку поєднано із спеціальним візуальним ефектом затемнення екрана для плавного перезапуску рівня. Створено комплексну систему ігрових механік та інтерактивних об’єктів, що включає збірні елементи, необхідні для проходження рівнів, а також спеціальні бонуси для тимчасового покращення можливостей персонажа. Реалізовано логіку переходу на наступний рівень, а також динамічні елементи середовища для автоматичного відкриття й блокування проходів. Система ворогів різного типу, разом із механіками укриттів та закритого простору, забезпечила високу динаміку, різноманітність ігрового процесу та поступове підвищення складності.

6. На завершальному етапі розробки було створено користувацький інтерфейс, що включає головне меню з вибором рівнів, систему паузи та лічильники ігрової статистики. Було додано аудіосупровід різного типу, текстові підказки, систему збереження прогресу та сюжетну складову з фінальними сюжетними рівнями й титрами. Успішне тестування, оптимізація ігрового процесу та експорт проєкту забезпечили створення цілісного, стабільного й готового ігрового продукту.

Отже, виконана робота підтвердила ефективність використання Godot Engine для створення сучасних 2D-ігор та продемонструвала практичне застосування теоретичних принципів ігрової розробки, проєктуванню рівнів і програмуванню різноманітних механік та логіки гри. Отримані результати можуть бути використані як основа для подальшого вдосконалення проєкту, створення нових ігрових систем та розробки складніших ігрових застосунків засобами Godot Engine.

# СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Game development glossary – terminology and slang. URL: https://medium.com/@brownale/game-development-terminology-and-slang-f28fd38ebb91 (дата звернення: 12.03.2026).

2. Bradfield C., Godot Engine Game Development Projects. Packt Publishing Ltd, 2018. 298 p.

3. Усі жанри комп'ютерних ігор. URL: https://uaplay.com.ua/usi-zhanry-pc-ihor/ (дата звернення: 12.03.2026).

4. The Ultimate Guide To Video Game Genres. URL: https://gameopedia.com/blogs/the-ultimate-guide-to-video-game-genres (дата звернення: 12.03.2026).

5. Digital Game Distribution Platform Market. URL: https://dataintelo.com/report/global-digital-game-distribution-platform-market (дата звернення: 13.03.2026)

6. Steam Games Dataset: Information of more than 122,000 games published on Steam. URL: https://www.kaggle.com/datasets/fronkongames/steam-games-dataset (дата звернення: 15.03.2026).

7. What type of Indie Games Actually Sell Best? (And Why). URL: https://www.youtube.com/watch?v=FA4haoZg_oc (дата звернення: 15.03.2026).

8. Що таке інді-ігри, або як розробити гру на одному бажанні. URL: https://skvot.io/uk/blog/shcho-take-indi-ihry (дата звернення: 20.02.2026).

9. Choosing a Game Engine: Unity vs. Unreal Engine vs. Godot (When to use What). URL: https://redappletechnologies.medium.com/choosing-a-game-engine-unity-vs-unreal-vs-godot-when-to-use-what-db17c9a95906 (дата звернення: 26.02.2026).

10. Cario E. Press Start: The History of Video Gaming. Ilex Press, 2025. 288 p.

11. Що таке ігри-платформери. URL: https://blog.acer.com/ua/discussion/3083/scho-take-igri-platformeri (дата звернення: 21.02.2026).

12. Peter D. McDonald. Run and Jump: The Meaning of the 2D Platformer. ВД: MIT Press, 2024, 184 p.

13. The Ultimate Guide to 2D Game Genres. URL: https://gamemaker.io/en/blog/2d-game-genres (дата звернення: 23.02.2025).

14. Клюцевська А. В., Дубрівна А. П. Особливості сучасного графічного проектування комп'ютерних ігор. Інноватика в освіті, науці та бізнесі: виклики та можливості : матеріали II Всеукраїнської конференції здобувачів вищої освіти і молодих учених, м. Київ, 18 листопада 2021 року. – Т. 1. – Київ : КНУТД, 2021. – С. 272-276. URI: https://er.knutd.edu.ua/handle/123456789/19529

15. Сайт YouTube. Super Mario Bros. (1985) Full Walkthrough NES Gameplay [Nostalgia]. URL: https://www.youtube.com/watch?v=rLl9XBg7wSs (дата звернення: 23.02.2026).

16. Сайт Youtube. Sonic The Hedgehog – Complete Walkthough. URL: https://www.youtube.com/watch?v=gjDanVVS5EI (дата звернення: 23.02.2026).

17. Сайт цифрової дистрибуції Steam. Інформаційна сторінка гри Celeste. URL: https://store.steampowered.com/app/504230/Celeste/ (дата звернення: 24.02.2026).

18. Сайт цифрової дистрибуції Steam. Інформаційна сторінка гри Hollow Knight. URL: https://store.steampowered.com/app/367520/Hollow_Knight/ (дата звернення: 24.02.2026).

19. Сайт цифрової дистрибуції Steam. Інформаційна сторінка гри Ori and the Blind Forest. URL: https://store.steampowered.com/app/261570/Ori_and_the_Blind_Forest/ (дата звернення: 24.02.2026).

20. Сайт цифрової дистрибуції Steam. Інформаційна сторінка гри Cuphead. URL: https://store.steampowered.com/app/268910/Cuphead/ (дата звернення: 25.02.2026).

21. Behind the delightfully vintage sound of 'Cuphead' – with Samuel Justice. URL: https://www.asoundeffect.com/cuphead-sound/ (дата звернення: 25.02.2026)

22. Сайт цифрової дистрибуції Steam. Інформаційна сторінка гри Dead Cells. URL: https://store.steampowered.com/app/588650/Dead_Cells/ (дата звернення: 25.02.2026).

23. What Are The Best Game Jam Games Ever Made? URL: https://gamemaker.io/en/blog/best-game-jam-games (дата звернення: 26.02.2026).

24. Bradfield C. Godot 4 Game Development Projects: Build five cross-platform 2D and 3D games using one of the most powerful open-source game engines. 2nd ed. Packt Publishing, 2023. 264 p.

25. Schell J. The Art of Game Design: A Book of Lenses. 3rd ed. CRC Press, 2019. 652 p.

26. Лугова Т. А., Блажко О. А. Проектування комп'ютерних ігор для навчання. Навчальний підручник. — Одеса: ФОП «Побута». 2018. С. 120-128. URI: https://library.megu.edu.ua:9443/jspui/handle/123456789/4220

27. Difficulty curves – it's not that hard. URL: https://supersonic.com/learn/blog/difficulty-curves/ (дата звернення: 16.03.2026).

28. What Is Iterative Game Design? URL: https://gamedesigning.org/learn/iterative/ (дата звернення: 18.03.2026).

29. Fulletron T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games. 5th ed. CRC Press, 2024. 586 p.

30. Level Up! The Art of Designing Game Progression and Player Rewards. URL: https://dev.to/gamepill/level-up-the-art-of-designing-game-progression-and-player-rewards-2603 (дата звернення: 20.03.2026).

31. Green A. M. Storytelling in Video Games: The Art of the Digital Narrative (Studies in Gaming). McFarland, 2017. 236p.

32. Сайт Reddit. Lies of P Progression Paths URL: https://www.reddit.com/r/LiesOfP/comments/1egf4jb/id_be_extremely_interested_to_see_how_lops/ (дата звернення: 23.03.2026).

33. Боговаров І. О. Розробка дизайну середовища та ігрових компонентів для всесвіту комп'ютерної гри M_Carnage. URI: https://er.knutd.edu.ua/handle/123456789/32353

34. How Games Tell Tales, Part 3: Detroit: Become Human, Freedom of Choice, and Intended Play. URL: https://interactivepasts.com/how-games-tell-tales-part-3-detroit-become-human-freedom-of-choice-and-intended-play/(дата звернення: 24.03.2026).

35. Сайт Reddit. All 100% FlowCharts with Global Stats Included (along with some interesting player statistics). URL: https://www.reddit.com/r/DetroitBecomeHuman/comments/99b65j/all_100_flowcharts_with_global_stats_included/ (дата звернення: 24.03.2026).

36. Video Game Level: A Designer's Introduction. URL: https://gamedesignskills.com/game-design/video-game-level/ (дата звернення: 24.03.2026).

37. Game Sound Design: The Art of Video Game Sound. URL: https://digitalresidency.com/game-sound-design-the-art-of-video-game-sound/ (дата звернення: 25.03.2026).

38. Different types of audio elements in video games. URL: https://www.quanticlab.com/different-types-of-audio-and-sound-design-in-video-games/ (дата звернення: 25.03.2026).

39. Сайт цифрової дистрибуції Steam. Інформаційна сторінка гри Limbo: https://store.steampowered.com/app/48000/LIMBO/ (дата звернення: 25.03.2026).

40. Howell C. Modern Game Testing: A Pragmatic Guide to Test Planning and Strategy. Modern Game Testing Company, 2022. 392p.

41. What Are the Main Types of Game Testing? Your Question Answered. URL: https://game-ace.com/blog/types-of-game-testing/ (дата звернення: 27.03.2026).

42. The Big Game Engine Report of 2025. End of the Era of In-House Engines. URL: https://app.sensortower.com/vgi/assets/reports/The_Big_Game_Engines_Report_of_2025.pdf (дата звернення: 30.03.2026)

43. Документація Godot Engine українською мовою. URL: https://docs.godotengine.org/uk/4.x/getting_started/introduction/introduction_to_godot.html (дата звернення: 30.03.2026)

44. What are Nodes and Scenes in Godot? A cooking perspective. URL: https://ljvmiranda921.github.io/notebook/2021/04/19/godot-nodes-and-scenes/ (дата звернення: 30.03.2026)

45. Oliveira. Games with Godot 4: The Ultimate Beginner's Guide to 2D Game Development: From Zero to Published. Kindle Edition, 2026. 89 p.

46. Vanhove S. Learning GDScript by Developing a Game with Godot 4: A Fun Introducion to Programming in GDScript 2.0 and Game Development Using The Godot Engine. Packt Publishing, 2024. 378p.

47. Сайт YouTube. How to use the new TileMap in Godot 4. URL: https://www.youtube.com/watch?v=tQSL2scuqeU (дата звернення: 06.04.2026).

48. Сайт YouTube. The ultimate introduction to shaders in Godot. URL: https://www.youtube.com/watch?v=J9qx8MxuJYM (дата звернення: 06.04.2026).

49. Hernand C. Godot GDScript Mastery for 2D and 3D Games: Build Stunning Games in Godot 4 with Step-By-Step Code and Real Projects. Independently Published, 2025. 259p.

50. Сайт CraftPix. URL: https://craftpix.net/ (дата звернення: 03.02.2026).

51. Сайт itch.io. URL: https://itch.io/ (дата звернення: 03.02.2026).

52. Godot Engine License. URL: https://godotengine.org/license/ (дата звернення: 30.03.2026)

53. The Book of Nodes: 2D. URL: https://christinec-dev.medium.com/the-book-of-nodes-2d-16f13691ac5f (дата звернення: 20.04.2026).

54. How to structure your Godot project (so you don't get confused). URL: https://new.pythonforengineers.com/blog/how-to-structure-your-godot-project-so-you-dont-get-confused/ (дата звернення: 04.02.2026).

55. Кушнер Д.: Володарі DOOM. Як двоє хлопців створили ігрову індустрію та виховали ціле покоління геймерів. Mal'opus, 2025. 352 с.

56. Створення вашої першої гри в Godot. URL: https://uk.sharpcoderblog.com/blog/creating-your-first-game-in-godot

57. Introduction to GDScript. URL: https://www.gdquest.com/tutorial/godot/gdscript/intro/

58. What is the best game engine for you? URL: https://game-wisdom.com/general/pros-cons-unity-best-game-engine

59. How to start making a game (a guide to planning your first project). URL: https://gamedevbeginner.com/how-to-start-making-a-game-a-guide-to-planning-your-first-project/

60. Як розробляють ігри? URL: https://lemon.school/blog/yak-rozroblyayut-igry

# ДОДАТКИ

# ДОДАТОК А

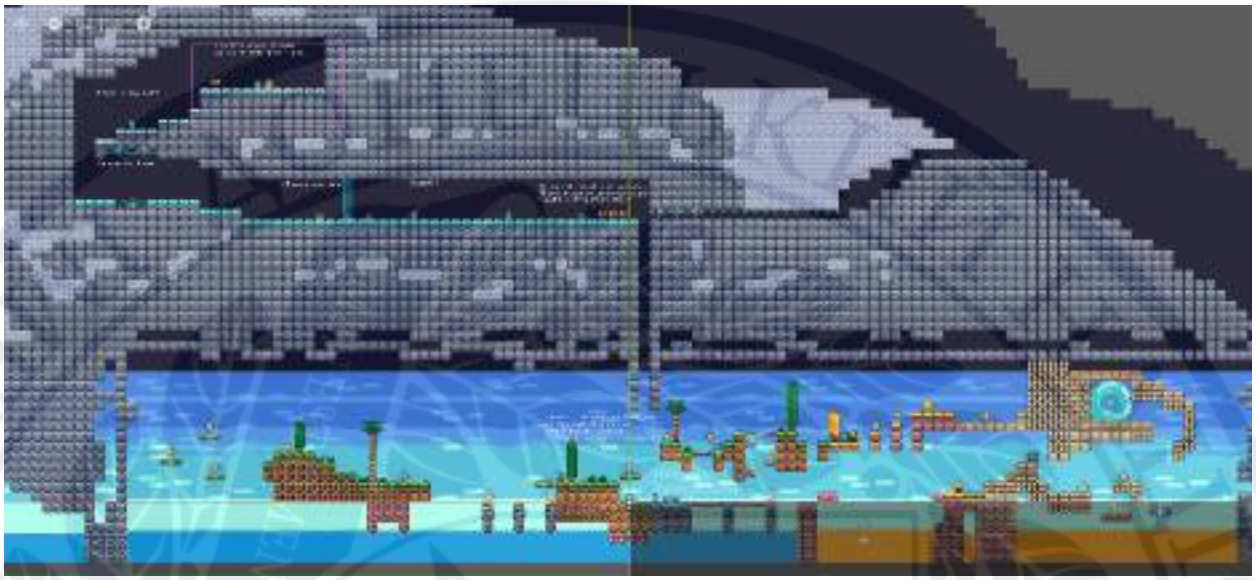

Рисунок А.1 Перший рівень гри “Where It All Begins” (“Там, де все починається”)

Рисунок А.2 Другий рівень гри “Parkour” (“Паркур”)

Рисунок А.3 Третій рівень гри “Slime Castle” (“Замок слизнів”)

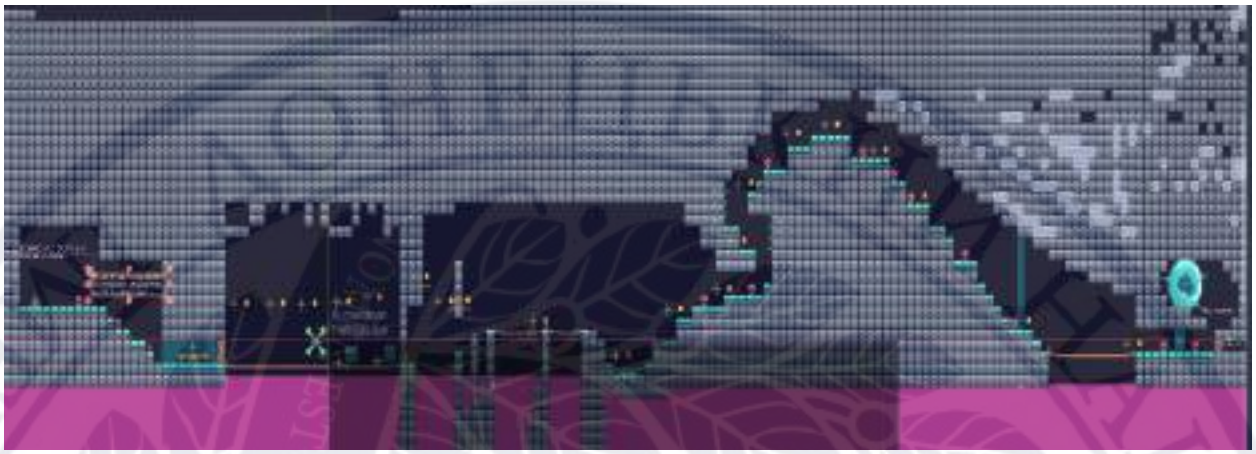

Рисунок А.4 Четвертий рівень гри “Survival Room” (“Кімната виживання”)

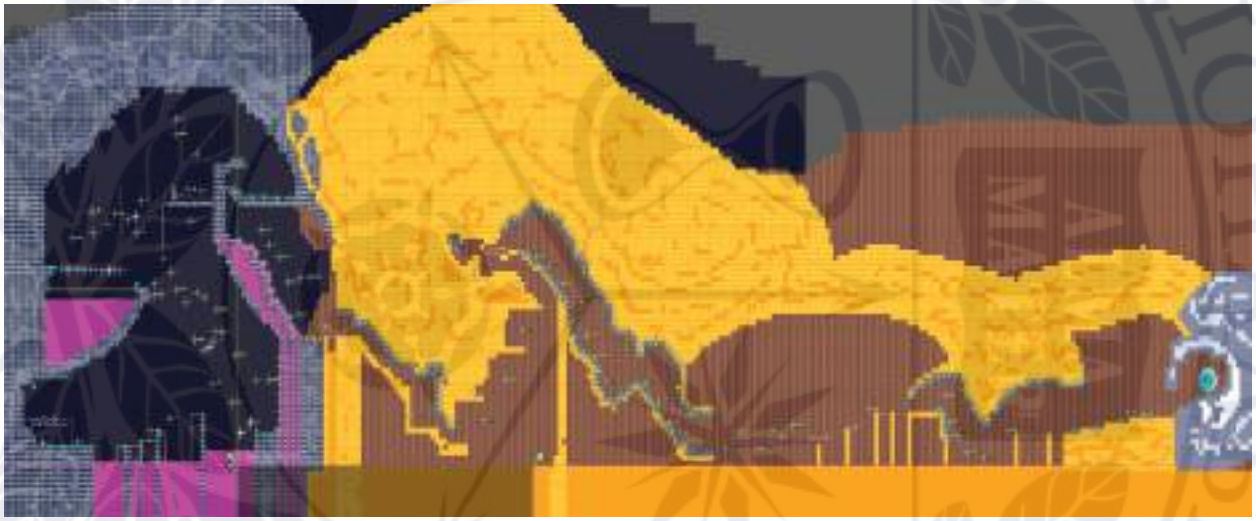

Рисунок А.5 П’ятий рівень гри “The Hive” (“Вулик”)

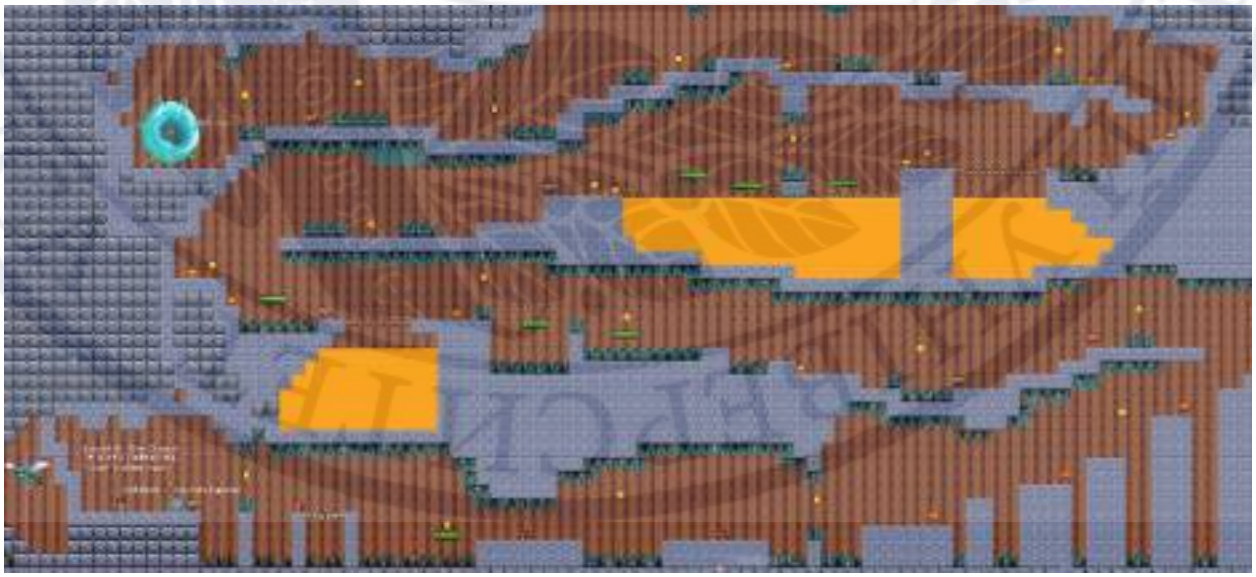

Рисунок А.6 Шостий рівень гри “The Chase” (“Переслідування”)

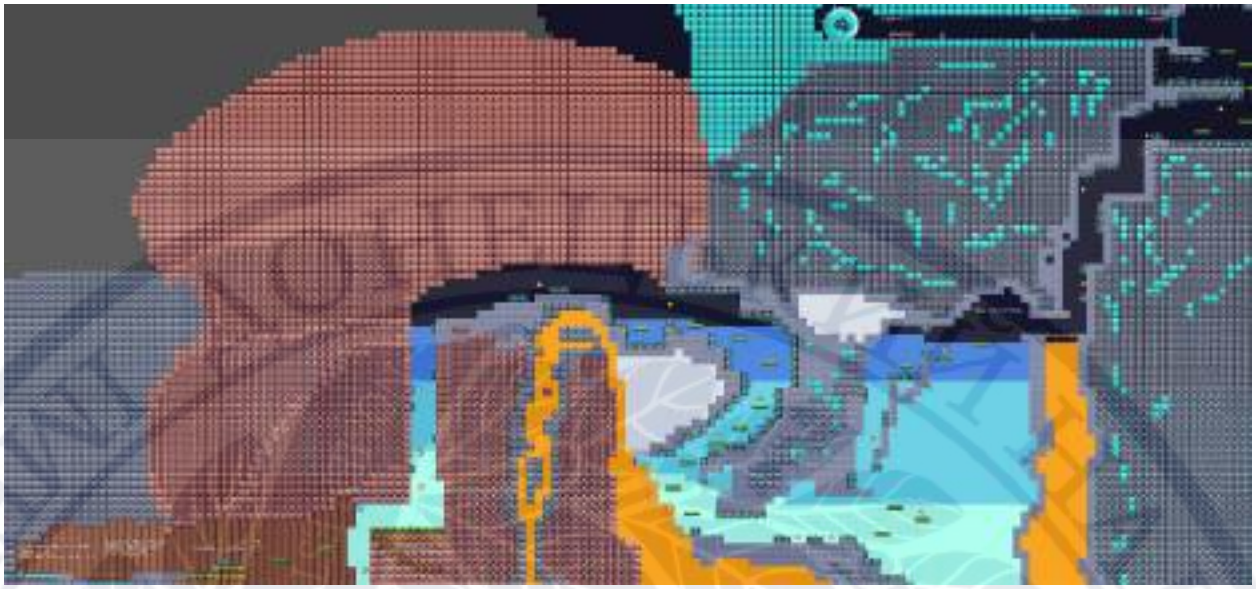

Рисунок А.7 Сьомий рівень гри “Wall Climb” (“Сходження по стінах”)

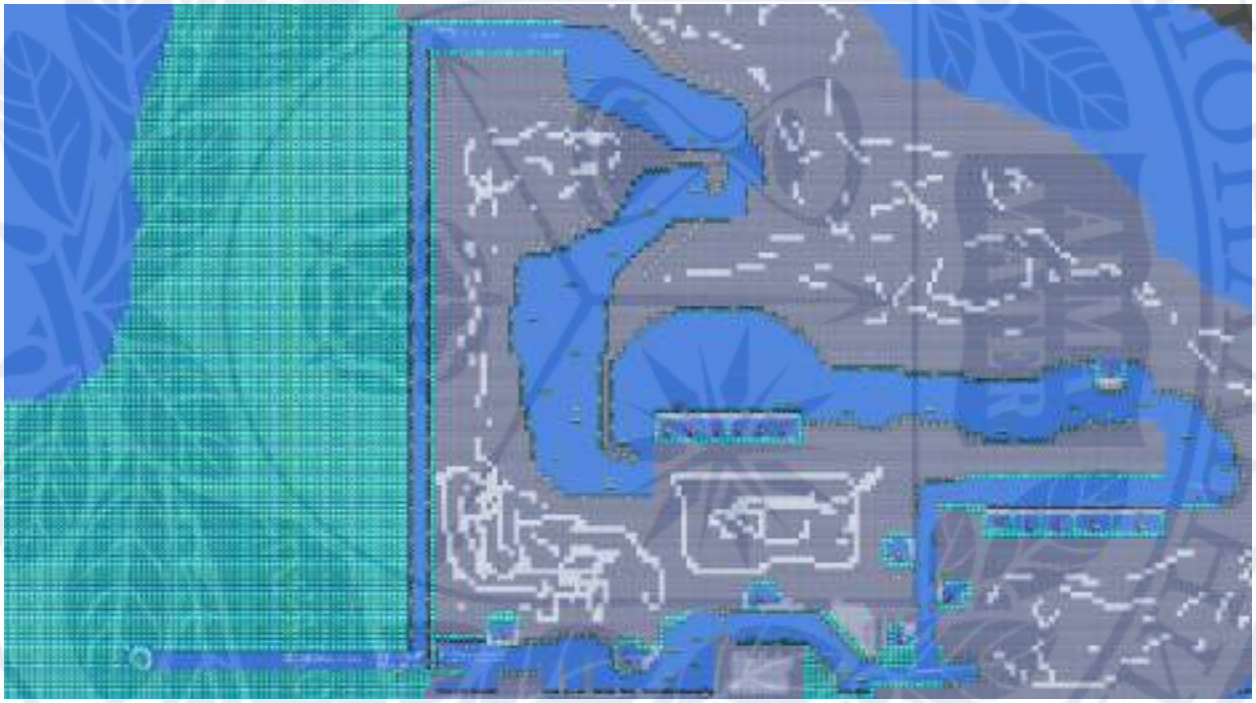

Рисунок А.8 Восьмий рівень гри “The Final” (“Фінал”)

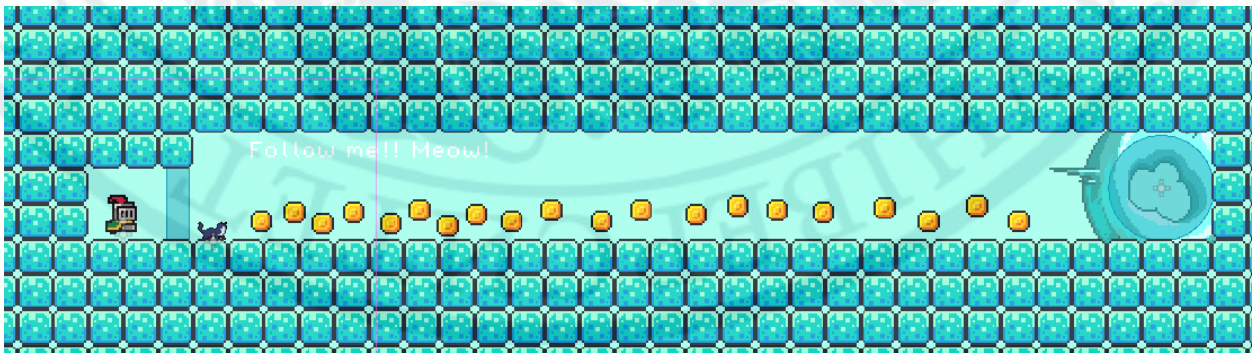


Рисунок А.9 Перший передфінальний сюжетна рівень: переслідування кота головним героєм

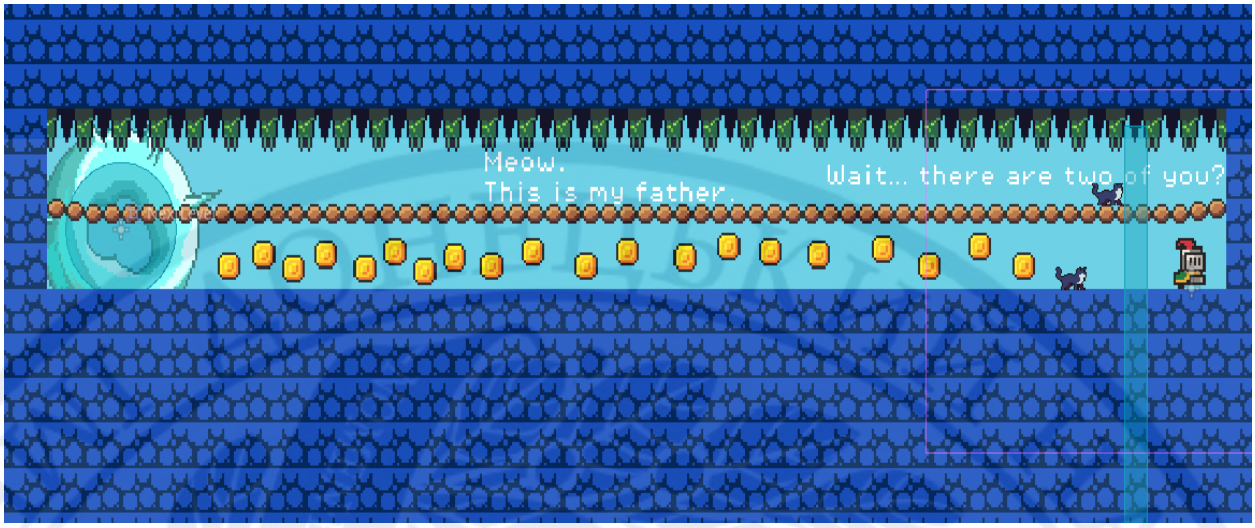


Рисунок А.10 Другий передфінальний рівень: зустріч із батьком кота та спільний перехід через портал

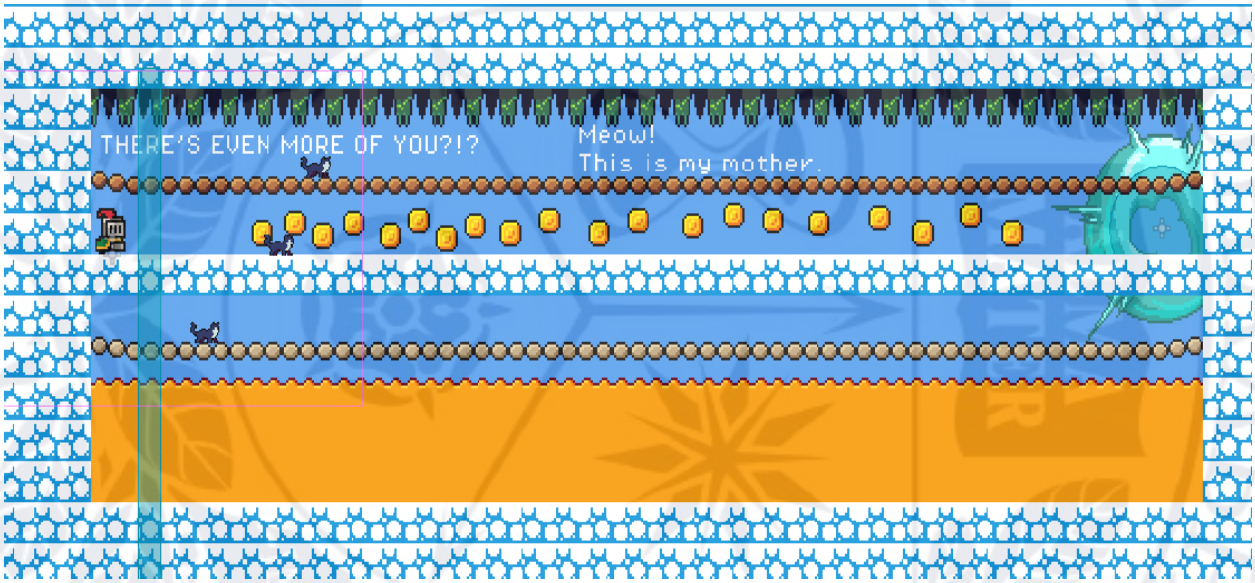


Рисунок А.11 Третій передфінальний рівень: зустріч із матір'ю кота та подальший супровід героя

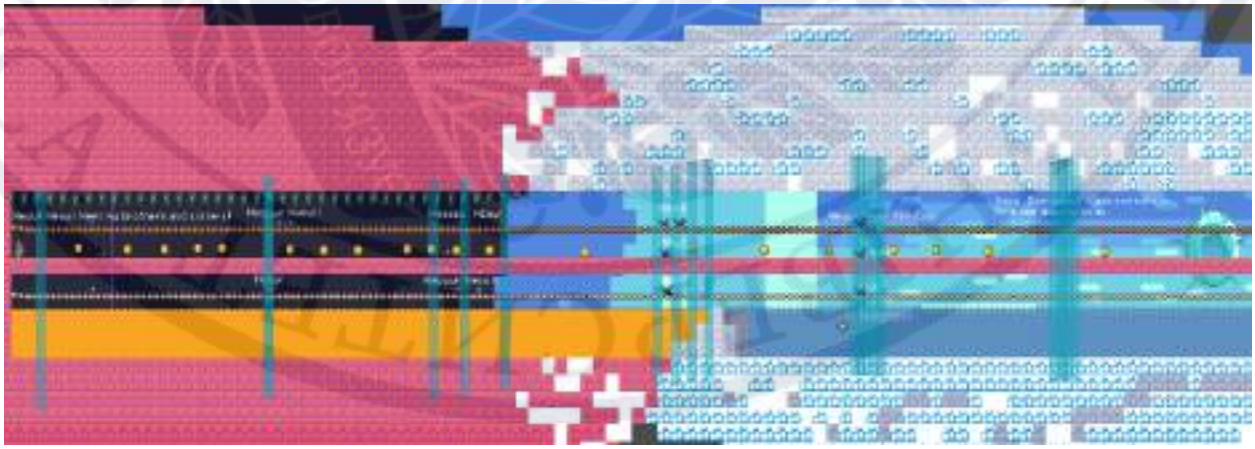

Рисунок А.12 Останній фінальний рівень: зустріч із братами та сестрами кота та спільний перехід всіх персонажів через портал

**ДОДАТОК Б**

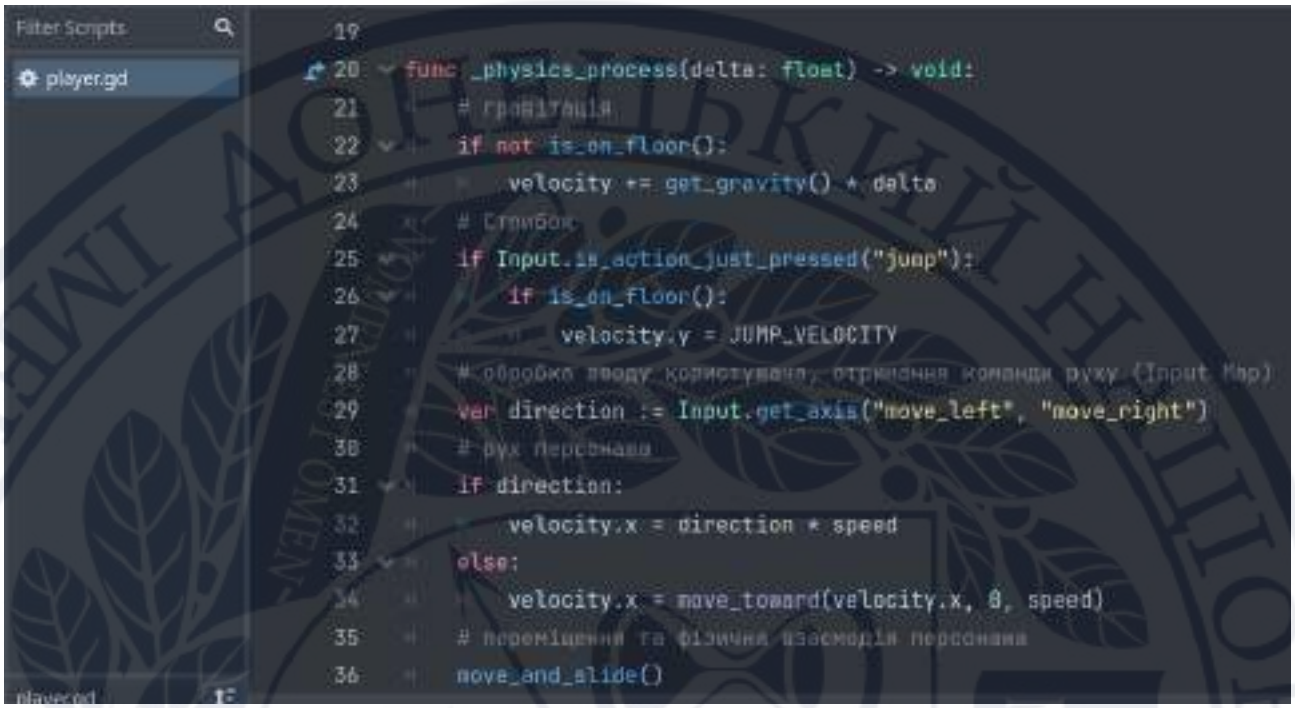


Рисунок Б.1 – Скрипт player.gd. Фрагмент програмування руху головного персонажа

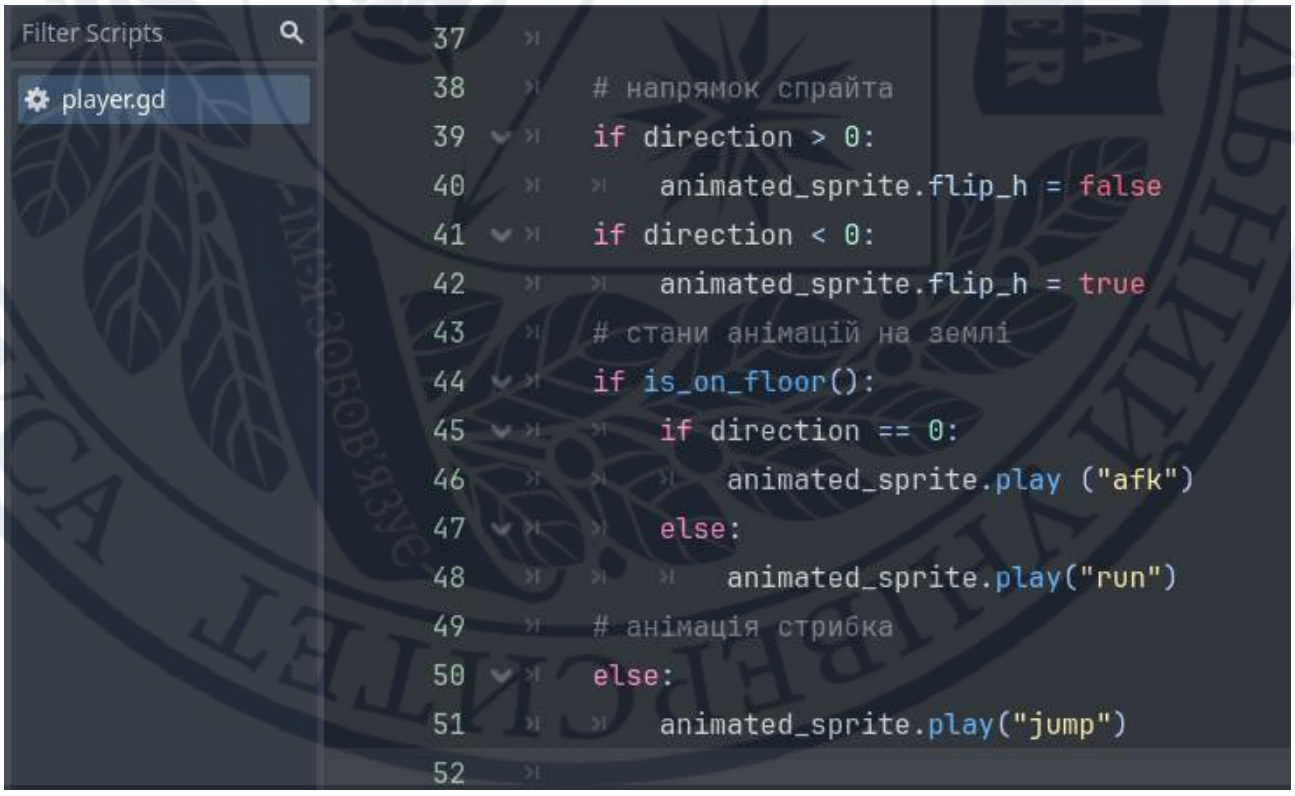


Рисунок Б.2 – Скрипт player.gd. Програмування анімацій головного персонажа

```gdscript
func die():
    if is_dead: #Захист від повторної смерті
        return
    is_dead = true

    Engine.time_scale = 0.5 #Уповільнення гри та зупинка анімації персонажа
    animated_sprite.stop()

    if has_node("CollisionShape2D"): #Видалення колізій, щоб персонаж більше не взаємодіяв з об'єктами
        get_node("CollisionShape2D").queue_free()
    #Сам затемнення
    var tween = create_tween()
    tween.tween_property(fade, "color:a", 1.0, 0.4)
    await tween.finished

    RenderingServer.set_default_clear_color(Color.BLACK) #Чорний фон під час перезавантаження сцени
    Engine.time_scale = 1 #Перезапуск рівня
    get_tree().reload_current_scene()
```

Рисунок Б.3 – Фрагмент коду Player.gd, який відповідає за запуск анімації загибелі персонажа, затемнення екрана та перезапуск поточного рівня

```gdscript
    # jump
    if Input.is_action_just_pressed("jump"):
        if is_on_floor():
            velocity.y = JUMP_VELOCITY
            coyote_timer = 0.0
            used_double_jump = false

        elif coyote_timer > 0:
            velocity.y = JUMP_VELOCITY
            coyote_timer = 0.0
            used_double_jump = true

        elif can_wall_jump and is_on_wall() and not wall_jump_cooldown:
            velocity.y = JUMP_VELOCITY

            var wall_normal = get_wall_normal()
            velocity.x = wall_normal.x * 500

            wall_jump_cooldown = true
            await get_tree().create_timer(0.2).timeout
            wall_jump_cooldown = false

        elif can_double_jump and not used_double_jump:
            velocity.y = JUMP_VELOCITY
            used_double_jump = true
```

Рисунок Б.4 – Фрагмент коду з реалізацією бонусних механік подвійного стрибка, відштовху від стін та coyote time у скрипті player.gd

```gdscript
func enable_double_jump():
    can_double_jump = true
    await get_tree().create_timer(9999999.0).timeout
    can_double_jump = false

func enable_speed_boost():
    speed = 250.0
    await get_tree().create_timer(9999999.0).timeout
    speed = BASE_SPEED

func enable_wall_jump():
    can_wall_jump = true
    await get_tree().create_timer(9999999.0).timeout
    can_wall_jump = false
```

Рисунок Б.5 – Функції активації бонусних механік персонажа у скрипті player.gd

```gdscript
extends Area2D
const FILE_BEGIN = "res://scenes/game_"
@onready var game_manager: Node = %SaveManager

func _on_body_entered(body: Node2D) -> void:
    if body.is_in_group("Player") and game_manager.score >= 20:
        var current_scene_file = get_tree().current_scene.scene_file_path
        var current_level_number = current_scene_file.get_file().get_basename().replace("game_", "").to_int()
        var next_level_number = current_level_number + 1
        var next_level_path = FILE_BEGIN + str(next_level_number) + ".tscn"
        call_deferred("load_next_scene", next_level_path, next_level_number)

func load_next_scene(path: String, level_number: int) -> void:
    SaveManager.save_progress(level_number)
    get_tree().change_scene_to_file(path)
```

Рисунок Б.6 – Скрипт next_level.gd для переходу до наступного рівня після збору 20 монет

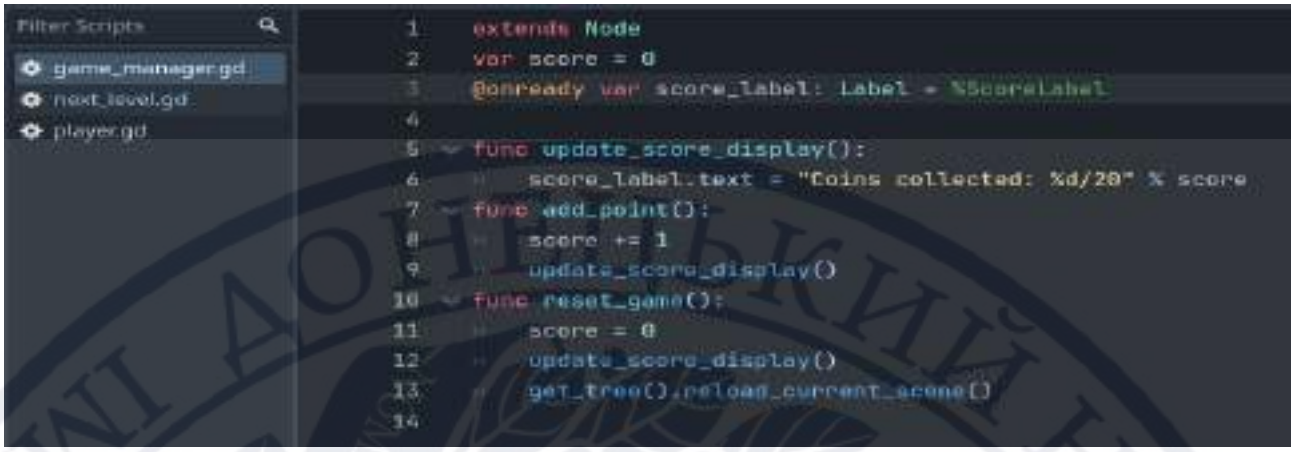


Рисунок Б.7 – Скрипт game_manager.gd


Рисунок Б.8 – Скрипт open_blocks.gd для відкриття проходу


Рисунок Б.9 – Скрипт CloseBlocks.gd для блокування проходу.

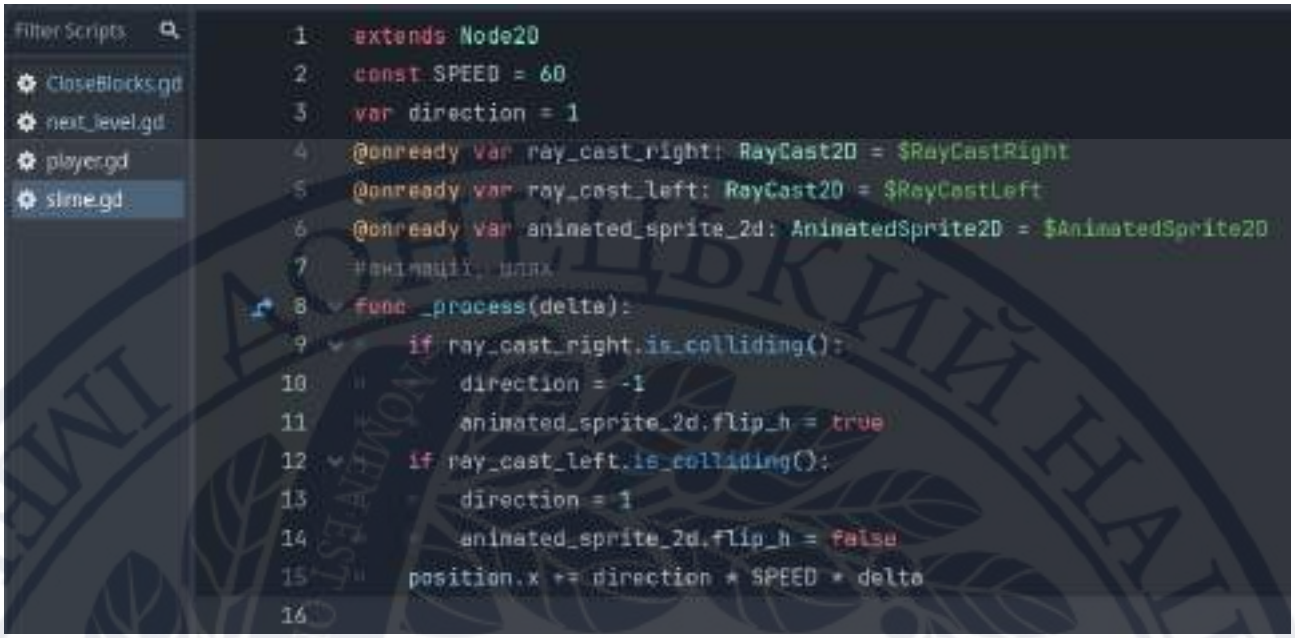


Рисунок Б.10 – Скрипт slime.gd для реалізації руху ворога та зміни напрямку після зіткнення з перешкодами


Рисунок Б.11 – Код Bee_Left_Right.gd для реалізації горизонтального руху бджоли між двома точками після запуску сцени.


Рисунок Б.12 – Фрагмент коду boss_bee.gd для активації та переслідування гравця


Рисунок Б.13 – Фрагмент коду boss_slime.gd для активації боса, стрибків та переслідування гравця

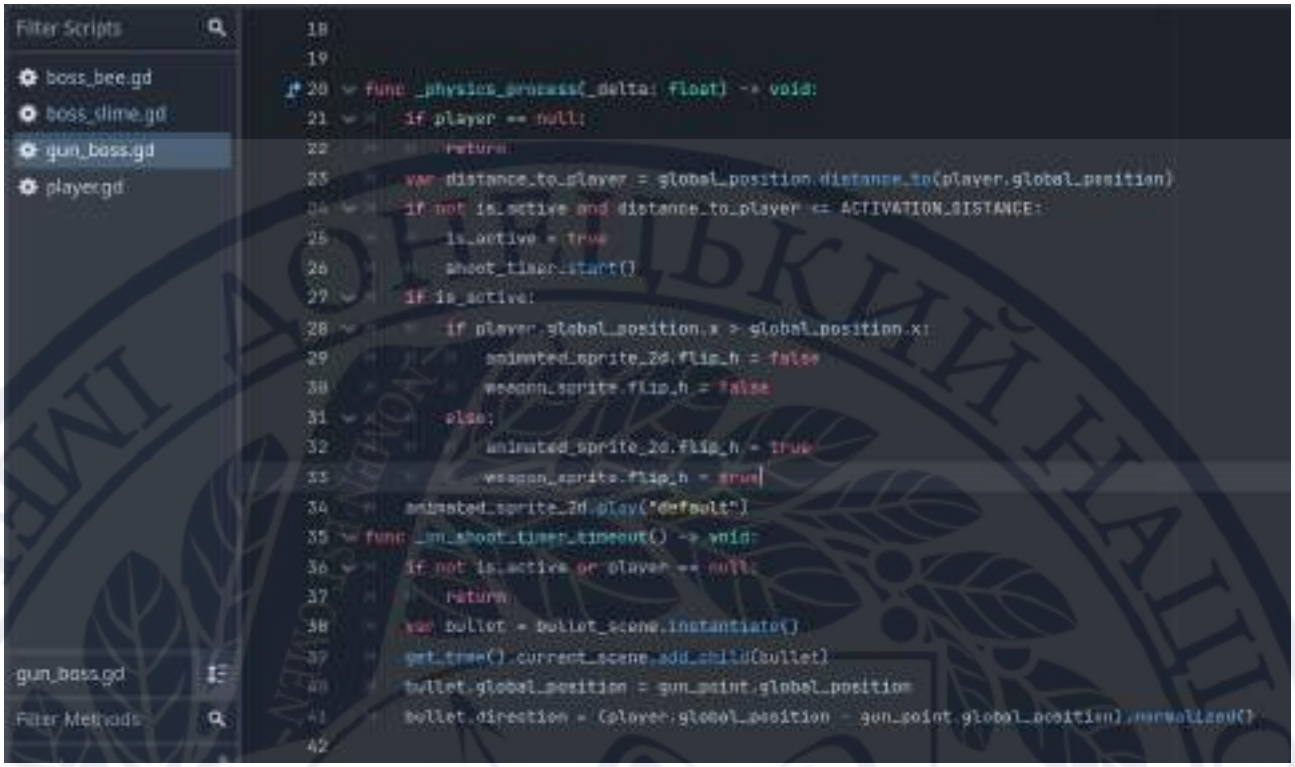


Рисунок Б.14 – Фрагмент коду gun_boss.gd для активації ворога-стрільця та створення кулі


Рисунок Б.15 – Скрипт bullet.gd для реалізації руху кулі та взаємодії з ігровим середовищем.


Рисунок Б.16 – Скрипт double_jump.gd для активації бонусу подвійного стрибка

Рисунок Б.17 – Скрипт menu.gd для реалізації функціоналу головного меню

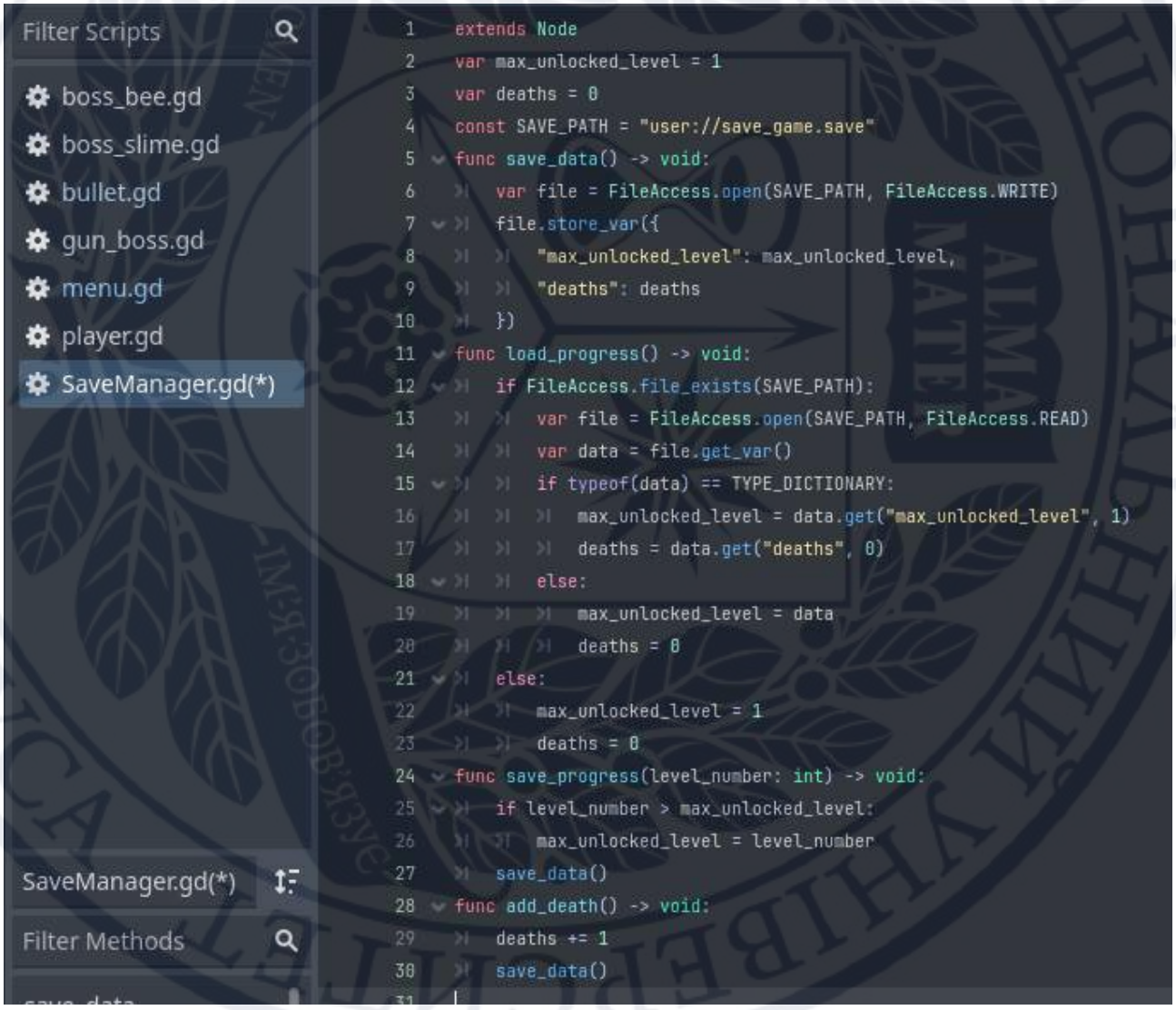


Рисунок Б.18 – Скрипт SaveManager.gd для реалізації системи збереження прогресу.

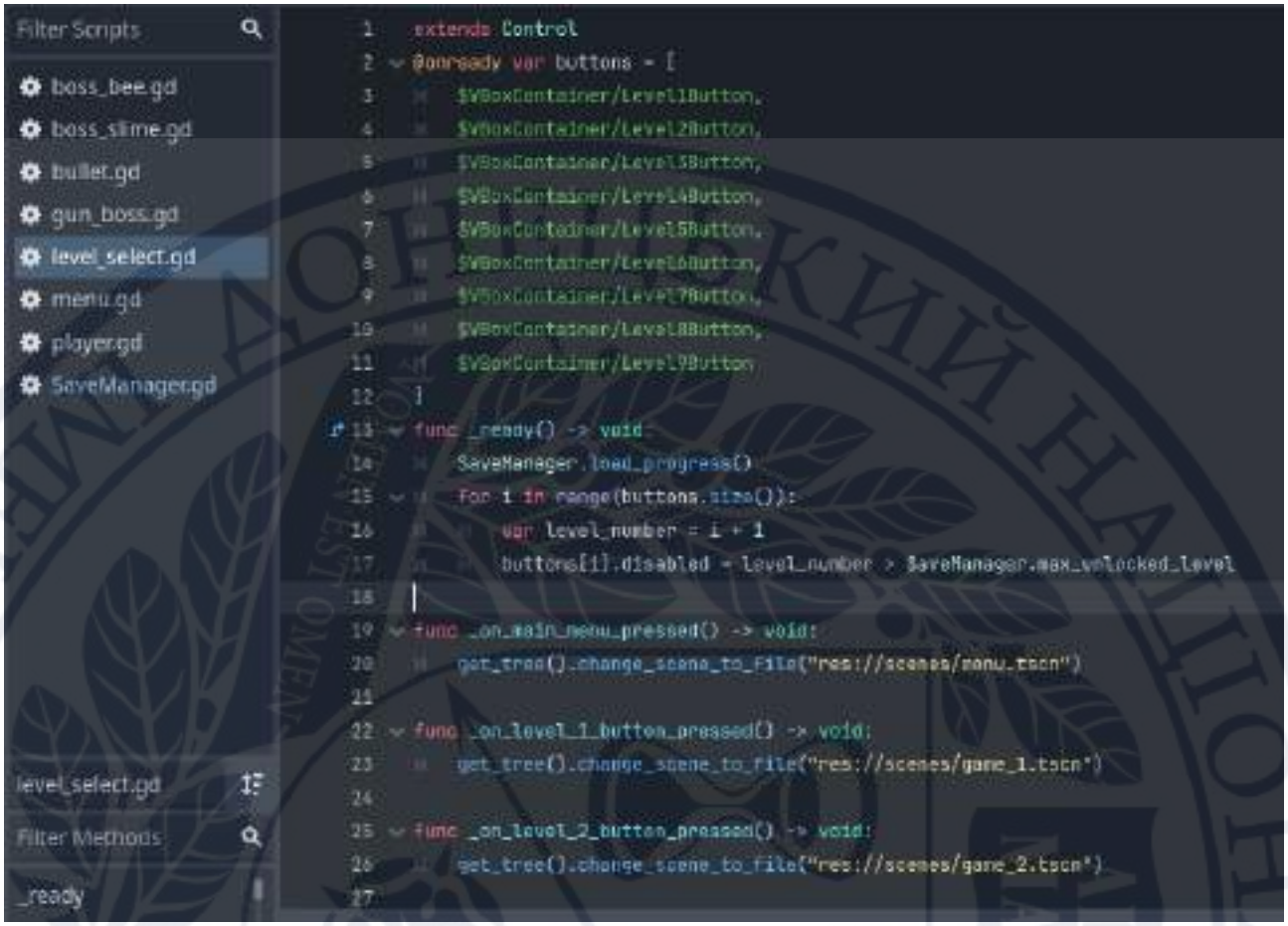


Рисунок Б.19 – Фрагмент коду level_select.gd для реалізації системи вибору рівнів


Рисунок Б.20 – Скрипт pause_menu.gd для реалізації меню паузи.

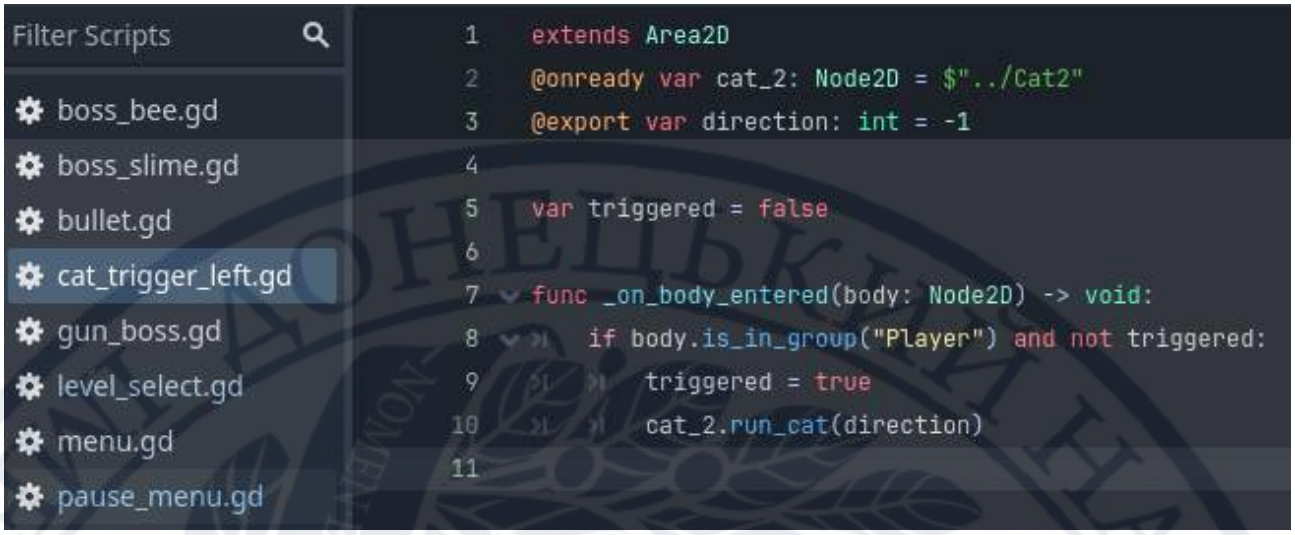


Рисунок Б.21 – Скрипт cat_trigger_left.gd для активації руху кота вліво


Рисунок Б.22 – Фрагмент коду cat2.gd для руху та зникнення кота

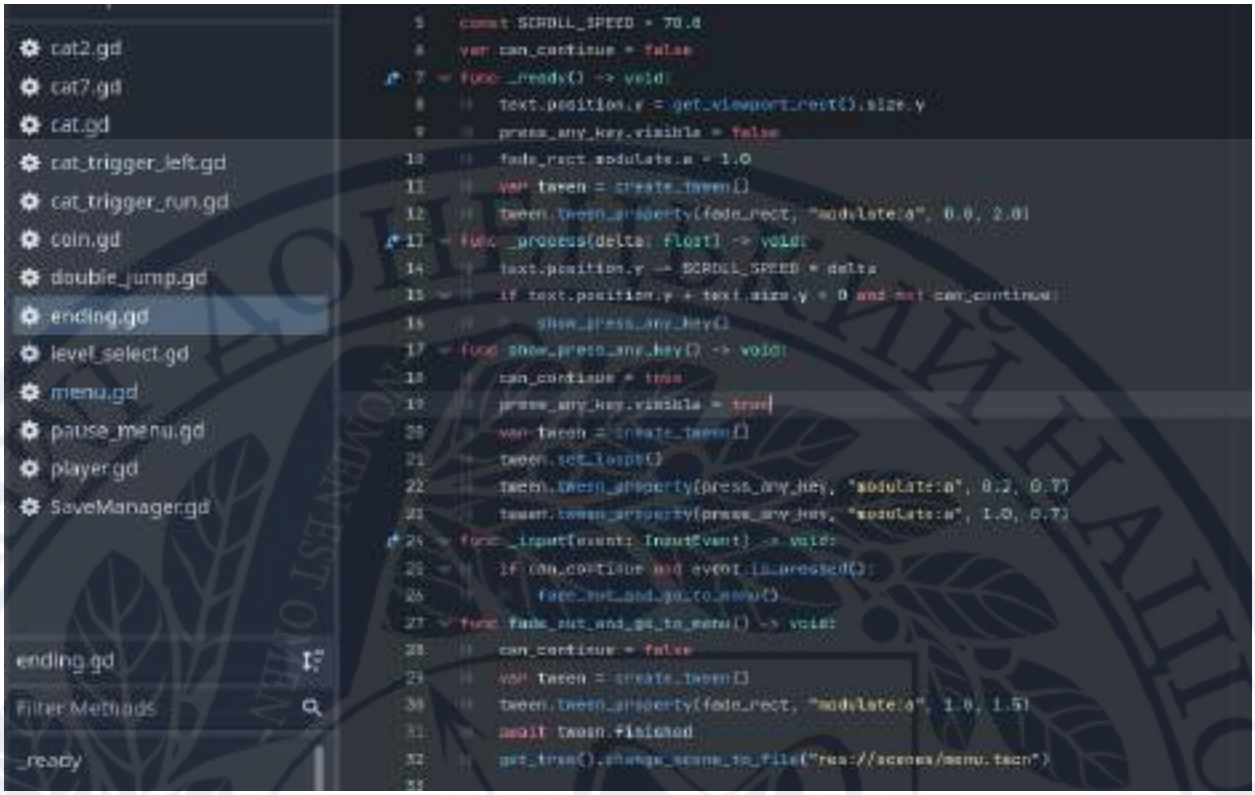


Рисунок Б.23 – Скрипт ending.gd для реалізації титрів

**ДОДАТОК В**


Рисунок В.1 – сцена Speed

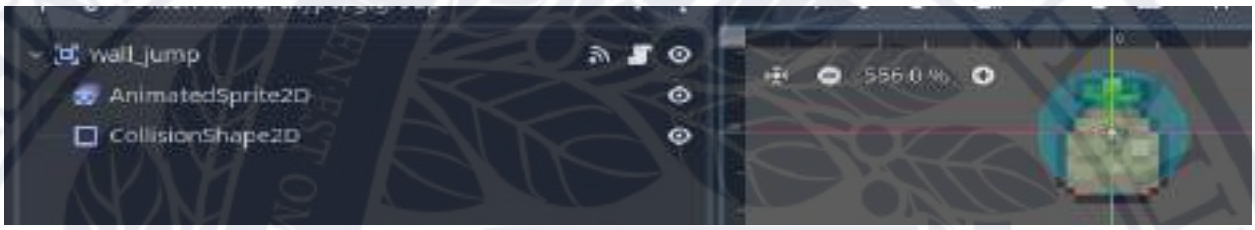


Рисунок В.2 – Сцена WallJump


Рисунок В.3 – Сцена Slime


Рисунок В.4 – Сцена Bee

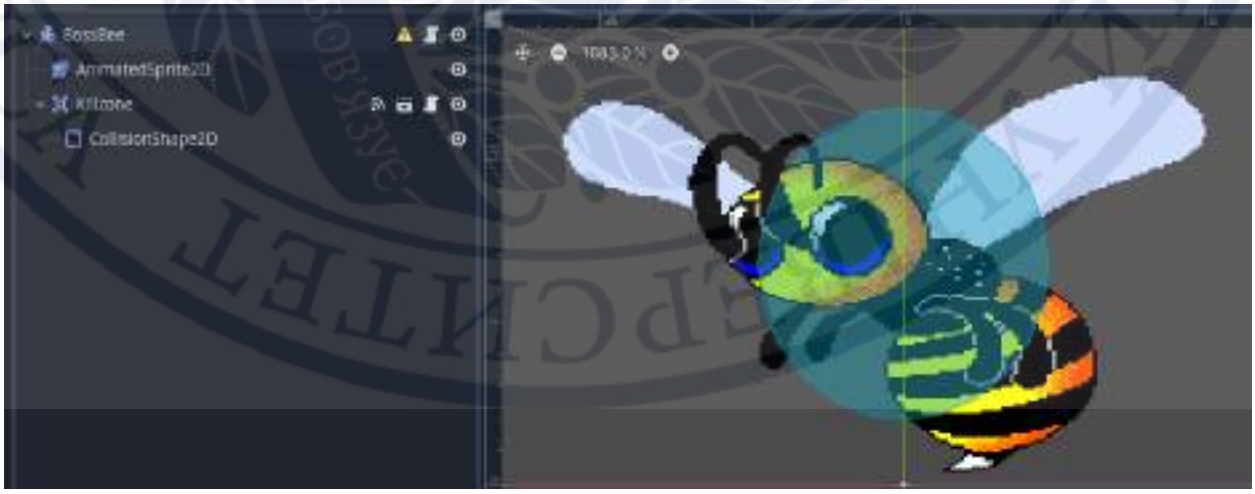


Рисунок В.5 – Сцена BossBee


Рисунок В.6 – Сцена BossSlime

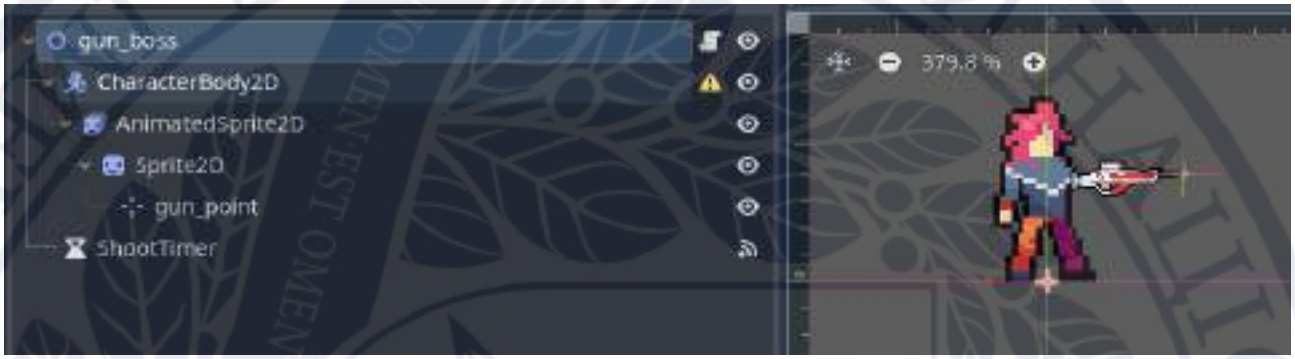


Рисунок В.7 – Сцена GunBoss

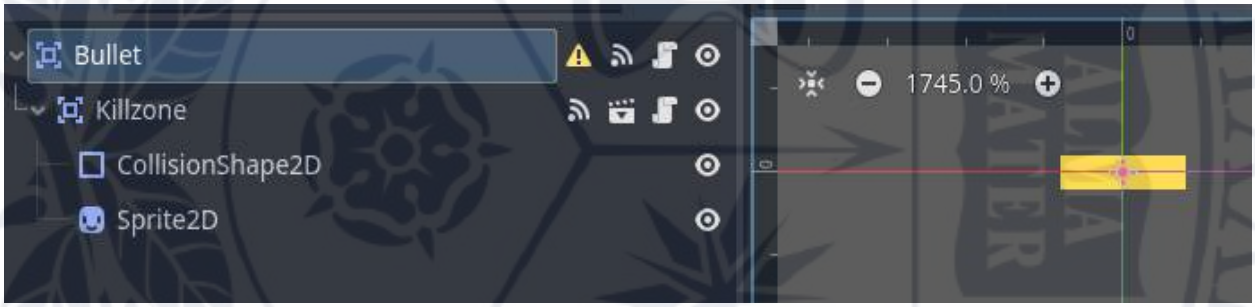


Рисунок В.8 – Сцена Bullet


Рисунок В.9 – Сцена головного меню гри


Рисунок В.10 – Сцена вибору рівнів LevelSelect

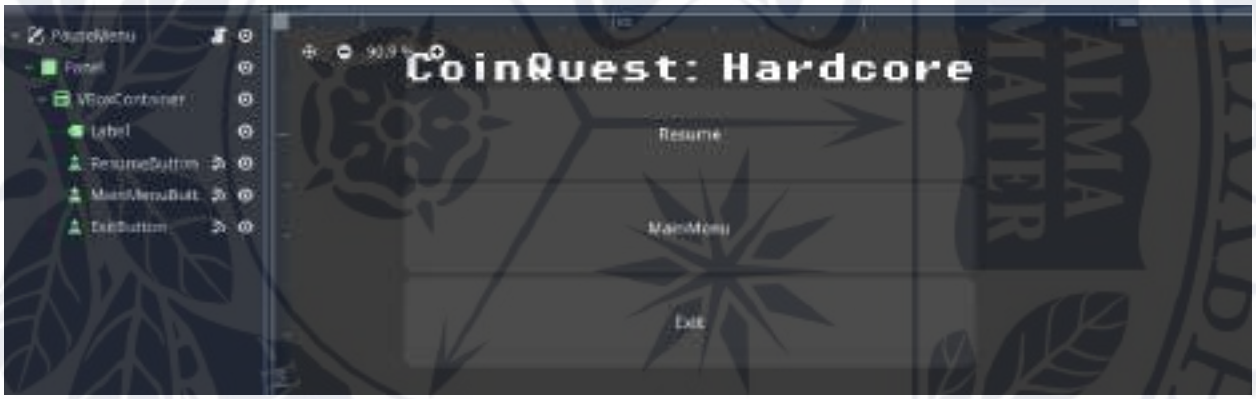


Рисунок В.11 – Сцена PauseMenu


Рисунок В.12 – Сцена Cat

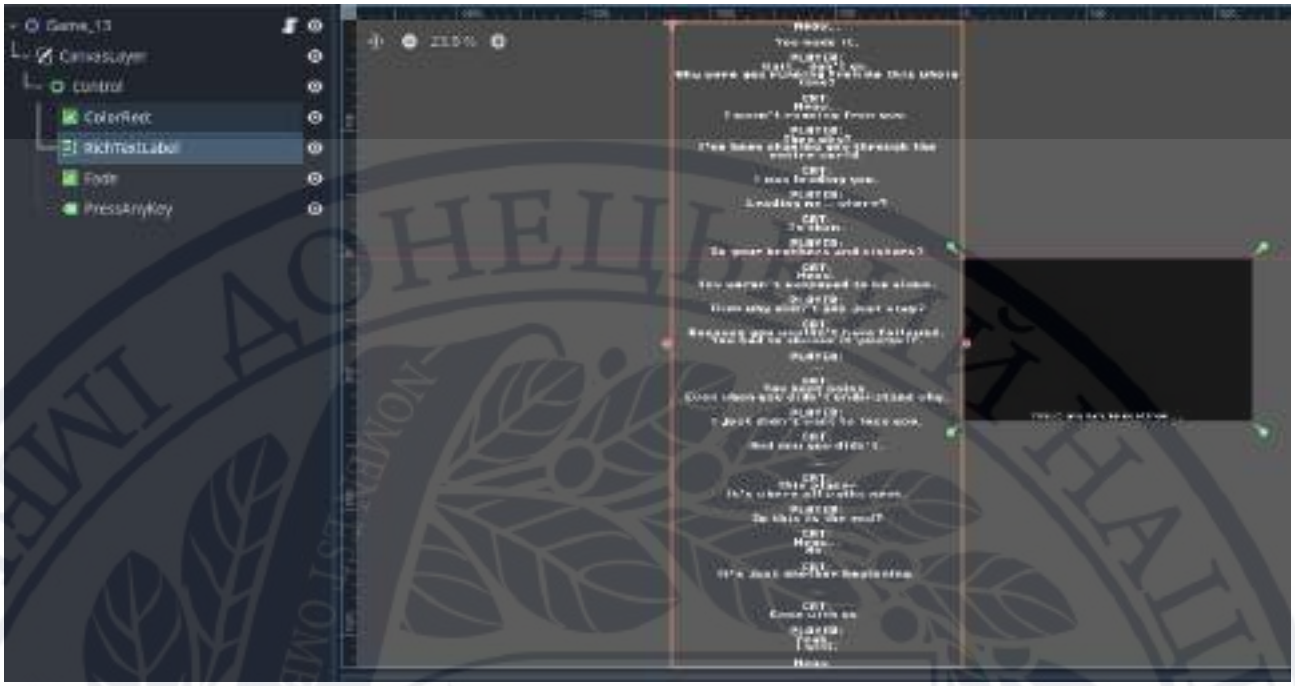


Рисунок В.13 – Сцена титрів

**ДОДАТОК Г**

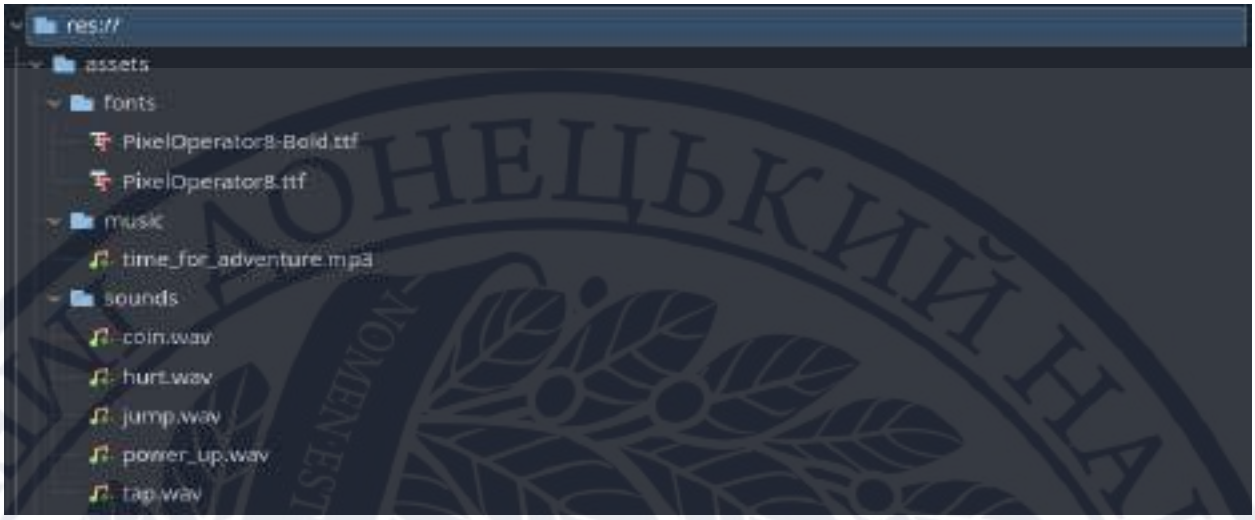


Рисунок Г.1 Папки Fonts, Music та Sounds для організації шрифтів, музики та звукових ефектів.


Рисунок Г.2 Папка Sprites для організації графічних ресурсів гри

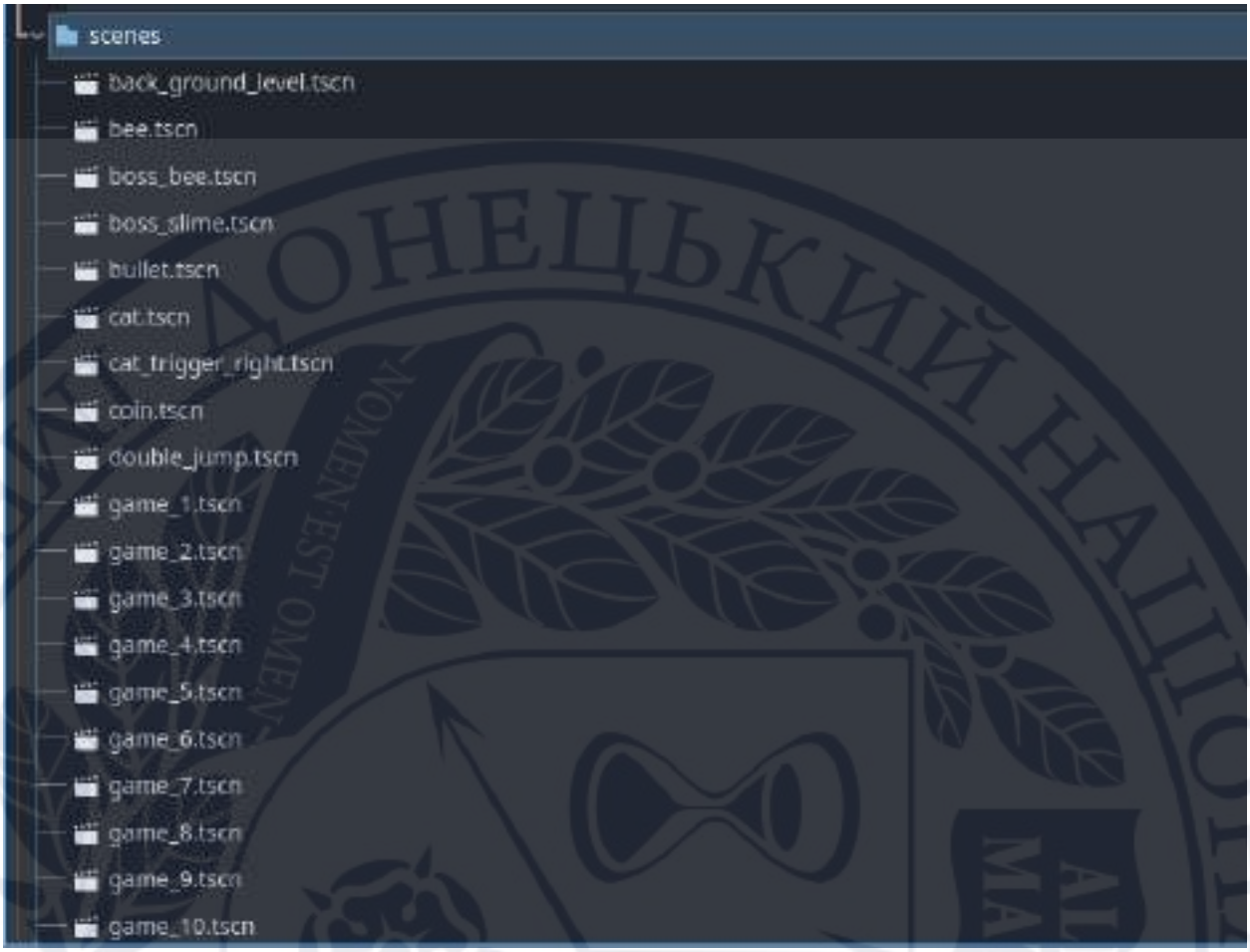


Рисунок Г.3 Папка Scenes для організації ігрових сцен проєкту (частина 1)


Рисунок Г.4 Папка Scenes для організації ігрових сцен проєкту (частина 2)


Рисунок Г.5 Папка Scripts для організації скриптів проєкту (частина 1)


Рисунок Г.6 Папка Scripts для організації скриптів проєкту (частина 2)


Рисунок Г.7 Папка Scripts для організації скриптів проєкту (частина 3)