МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА 2D ГРИ "V-NIGHT:REBOOT" ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ ЗАСОБАМИ GODOT ENGINE»**

Виконав: студент 4 курсу, групи 6.1220
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Ш.Х. Саїфуль Ислам
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., доцент

_____ Шило Г.М.
(підпис)

" ____ " _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Саїфуль Ислам Шамілю Халідовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка 2D гри "V-NIGHT:REBOOT" для мобільних пристроїв засобами Godot Engine

керівник роботи Матвіїшина Надія Вікторівна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » 12 2023 року № 2180-с

2. Строк подання студентом роботи 15.05.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Постановка задачі
2. Основні теоретичні відомості
3. Розробка 2D гри
4. Тестування гри

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 25.12.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	25.01.2024	
2.	Збір вихідних даних.	10.02.2024	
3.	Обробка методичних та теоретичних	12.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	14.04.2024	
5.	Розробка третього розділу.	11.05.2024	
6.	Оформлення та нормоконтроль	05.06.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	23.06.2024	

Студент _____ Ш.Х. Саїфуль Іслам
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.В. Матвіїшина
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка 2D гри "V-NIGHT:REBOOT" для мобільних пристроїв засобами Godot engine»: 35 с., 4 рис., 22 джерела, 2 додатки.

2D ГРА, ДИЗАЙН-ДОКУМЕНТ, КАЗУАЛЬНА ГРА, СЦЕНА ГРИ, GDSCRIPT, GODOT ENGINE.

Об'єкт дослідження – процес розробки казуальних ігор.

Предмет дослідження – ігри для мобільних пристроїв на двигуні Godot Engine.

Методи дослідження: описовий, аналіз, експеримент.

Основною метою кваліфікаційної роботи є розробка казуальної 2D гри для мобільних пристроїв засобами Godot engine.

В рамках кваліфікаційної роботи створено повноцінний ігровий продукт – 2D гру, з використанням сучасних технологій та інструментів розробки. Для реалізації мети використано двигун для розробки гри Godot Engine, мову програмування GDScript, інтегровану в двигун Godot Engine для написання ігрової логіки.

Отримані результати можуть бути цінними для розробників, які працюють над подібними ігровими проєктами, а також особливо корисні в контексті, де необхідно інтегрувати стратегічне планування, креативні підходи до вирішення проблем і захоплюючий геймплей.

SUMMARY

Bachelor's Qualifying Theses «Development of 2D Game " Y-Night: Reboot" for Mobile Devices using Godot Engine»: 35 pages, 4 figures, 22 references, 2 supplements.

2D GAME, DESIGN DOCUMENT, CASUAL GAME, GAME SCENE, GDSCRIPT, GODOT ENGINE.

Object of the study – the process of developing casual games.

Subject of the study – games for mobile devices on the Godot Engine.

Methods of research: descriptive, analysis, experiment.

Aim of the study: development of a casual 2D game for mobile devices using the Godot engine.

As part of the qualification work, a full-fledged 2D game product was created using modern technologies and development tools. To achieve this goal, we used the Godot Engine game development engine, a GDScript programming language integrated into Godot Engine to write game logic.

The findings can be valuable for developers working on similar game projects, and are especially useful in contexts where strategic planning, creative problem-solving, and addictive gameplay need to be integrated.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ 7
1 Аналіз сучасного ринку мобільних ігор 9
1.1 Стан ринку мобільних ігор 9
1.2 Популярні жанри мобільних ігор 9
1.3 Використання двигунів для розробки мобільних ігор 10
1.4 Стан розробки мобільних ігор на godot engine 11
1.5 Тенденції в розробці мобільних ігор 13
2 Організація та логіка проєкту 15
2.1 Організація роботи над проєктом 15
2.1.1 Дизайн-документ гри v-night:reboot 15
2.2 Поняття сцени в godot 17
2.3 Організація збереження даних 19
3 Тестування проєкту 24
3.1 Виправлення логічних помилок коду 24
3.2 Особливості адаптації для мобільних пристроїв 25
Висновки 27
Перелік посилань 28
Додаток А Частина коду гравця 31
Додаток Б Посилання на інші скрипти проєкту 35

ВСТУП

Сучасне середовище розробки мобільних ігор динамічно розвивається і створення захопливих аркадних ігор вже стало невід'ємною частиною індустрії мобільних розваг. Тему щодо розробки мобільної ігри було обрано з метою дослідження повного циклу розробки продукту, прояснення дизайнерських рішень та потенційного застосування винайдених відомостей.

Вибір цієї теми підкріплюється стрімко зростаючим за останні роки попитом на мобільні аркадні ігри. З поширенням смартфонів і планшетних комп'ютерів, відбувається зміна розробницьких трендів в бік ігор категорії В та мобільних аркадних ігор. За статистикою, понад 2.4 мільярди людей грають в ігри на своїх телефонах. При цьому, операційні системи iOS та Anroid встановлені на приблизно 80% активних пристроїв [5]. З цього можна зробити висновок, що випуск мобільної гри на одну з цих платформ може бути потенційно цікавим великій аудиторії.

Вибір саме Godot Engine був обумовлений його доступністю, простотою в користуванні та потужними інструментами розробки ігор. Цей двигун є повністю безкоштовним з відкритим вихідним кодом згідно з дозвільною ліцензією MIT, – на практиці це дозволяє командам розробників, які обрали Godot, модифікувати двигун під особисті потреби без попередніх умов, а також залучати велику кількість ентузіастів, які постійно покращують цей двигун. Наприклад, для випуску оновлення з версією 4.0, було залучено більше ніж 2000 людей, яких в суспільстві Godot називають "contributors" [15]. Наразі цей двигун має великі переваги над конкурентами, серед яких система сцен, які будуються на вузлах, що дозволяє створювати складні ієрархії сцен. Godot підтримує найпоширеніші серед ігрових розробників мови програмування, такі як C#, C та C++, а також надає доступ до власної мови програмування GDScript. В одному проекті можна користуватись різними мовами програмування, що дозволяє розробникам з різним досвідом розробки працювати один з одним без

ускладнень [10, 18]. Простота двигуна та велика кількість навчальних ресурсів, які стали активно створюватись останні 2 роки, дозволяють новачкам без досвіду розробки ігор зробити свою першу гру за декілька годин. В представленій роботі розглянуто цикл розробки мобільних ігор, та надано інформацію про Godot Engine як потужної сили в розробці продуктів та найчастіше повноцінну заміну Unity Engine.

Актуальність дослідження підкреслюється аналізом сучасного стану індустрії мобільних ігор. Глибше вивчаючи наявну літературу, було вирішено виявити основні проблеми та можливості, що існують у цій галузі. У представленій роботі розглядається процес розробки мобільних ігор з акцентом на унікальні виклики та можливості, які надає жанр казуальних ігор. Об'єктом дослідження є процес розробки казуальних ігор. Предмет дослідження – ігри для мобільних пристроїв на Godot Engine.

Основною метою кваліфікаційної роботи є розробка казуальної 2D гри для мобільних пристроїв засобами Godot Engine. Конкретні цілі включають:

Проведення ретельного аналізу проблем, з якими стикається сучасна індустрія мобільних ігор, з використанням як наявної літератури, так і власного досвіду розробників.

Надання обґрунтування ключових дизайнерських рішень, прийнятих під час процесу розробки.

Аналіз різноманітних можливостей застосування результатів, отриманих у цьому дослідженні, з представленням, як висновки, зроблені в кваліфікаційній роботі можуть бути використані для майбутніх спроб розробки ігор.

На практичному рівні висновки, представлені у роботі, можуть бути корисними для колег-розробників, початківців у галузі та установ, що працюють у сфері інформаційних технологій та штучного інтелекту. Висновки за результатами дослідження можуть стати основою для кращих практик та надихнути на нові підходи розробки мобільних ігор.

1 АНАЛІЗ СУЧАСНОГО РИНКУ МОБІЛЬНИХ ІГОР

1.1 Стан ринку мобільних ігор

Сучасний ринок мобільних ігор є одним із найдинамічніших та найбільш конкурентних сегментів геймінгу. За останні десятиліття спостерігається вибуховий ріст популярності мобільних ігор, що обумовлено поширенням смартфонів та планшетних комп'ютерів серед різних верств населення. За даними досліджень, з 2020 року виручка від мобільних ігор перевищила виручку від ігор на персональних комп'ютерах та консолях. Прогнозується, що до 2024 року глобальна виручка в секторі мобільних ігор сягне 100,54 мільярда доларів США, а до 2029 року - 164,81 мільярда доларів США [1].

1.2 Популярні жанри мобільних ігор

Наразі, серед мобільних ігор можна виділити кілька основних жанрів, які користуються високим попитом:

- ігри в жанрі "хенд-хелд" (Handheld Games): Ці ігри орієнтовані на короткий геймплей та простоту управління. Прикладами таких ігор можуть бути "Angry Birds", "Fruit Ninja" та "Cut the Rope";
- ігри в жанрі "головоломки" (Puzzle Games): Цей жанр включає в себе ігри, що вимагають від гравця логічного мислення та розв'язання різноманітних головоломок. Прикладами популярних представників цього жанру є "Monument Valley", "Threes!" та "Candy Crush Saga";
- ігри в жанрі "аркади" (Arcade Games): Ці ігри відрізняються відмінною геймплейною динамікою та високою швидкістю дій. Серед них можна виділити такі хіти, як "Temple Run", "Subway Surfers" та "Crossy Road";

– ігри в жанрі “стратегії” (Strategy Games): Цей жанр орієнтований на розвиток стратегічного мислення та планування. Популярні представники цього жанру включають “Clash of Clans”, “Plants vs. Zombies” та “Hearthstone” [3,7].

1.3 Використання двигунів для розробки мобільних ігор

На сучасному ринку існує безліч інструментів та двигунів для розробки мобільних ігор. Godot Engine набув широкої популярності завдяки своїй потужності, гнучкості та активній спільноті розробників, хоча і не настільки численній, як у конкурентів. Його використання дозволяє створювати якісні та ефективні мобільні ігри з різноманітними геймплейними механіками та візуальним оформленням.

Однак, Godot Engine – це лише один з багатьох двигунів, доступних для розробки мобільних ігор. Наразі є декілька інших популярних двигунів:

– Unity - це один з найпопулярніших двигунів для розробки мобільних ігор. Він використовується для створення ігор різних жанрів і має потужні інструменти для 3D- і 2D-графіки;
– Unreal Engine – це ще один потужний двигун, який використовується для створення високоякісних мобільних ігор. Він відомий своїми візуальними можливостями і гнучкістю;
– Solar2D, також відомий як Corona SDK, дозволяє розробникам ігор створювати мобільні 2D-проекти;
– Buildbox – один з найбільш доступних двигунів для розробки ігор на Android;
– Fusion 2.5 – це ігровий двигун, який використовує систему редактора подій для швидкого створення мобільних ігор або додатків;

– GameMaker Studio 2 - це потужний двигун для розробки 2D-ігор, який пропонує простий у використанні інтерфейс та гнучкі можливості скриптів [8].

За даними різних джерел, Unity та Unreal Engine залишаються найпопулярнішими двигунами для розробки мобільних ігор.

1.4 Стан розробки мобільних ігор на Godot Engine

Godot Engine, хоча й не настільки популярний, як Unity або Unreal Engine, але має ряд переваг, які роблять його повноцінною заміною цих двигунів для розробників:

– однією з ключових особливостей Godot є його система вузлів. Все в Godot, від спрайтів до таймерів, є вузлами. Це дозволяє легко організовувати та перевикористовувати код, а також створювати складні ієрархії об’єктів;
– інтерфейс Godot є дуже інтуїтивним і гнучким. Він дозволяє розробникам легко налаштовувати робоче середовище для відповідності їхнім потребам;
– Godot використовує GDScript, мову програмування, схожу на Python. GDScript досить зручний для навчання, і він ідеально підходить для швидкої ітерації;
– Godot є повністю відкритим двигуном. Це означає, що розробники можуть вільно модифікувати двигун для відповідності їхнім конкретним потребам.

На відміну від багатьох інших двигунів, Godot було спеціально розроблено з урахуванням 2D-геймплею, хоча він також підтримує 3D.

– Godot має активну спільноту розробників, яка постійно працює над поліпшенням двигуна і допомогою новим користувачам;

- Godot має вбудований налагоджувач, який дозволяє розробникам відстежувати помилки в реальному часі під час виконання гри. Налагоджувач надає інформацію про активні об'єкти, виклики функцій, виконання скриптів, використання пам'яті та інші параметри. Це допомагає розробникам швидко знаходити та виправляти помилки у своєму коді;
- Godot також пропонує інструменти для моніторингу продуктивності. Розробники можуть відстежувати різні метрики продуктивності, такі як використання ЦПУ, використання пам'яті, кількість активних об'єктів і багато іншого, що робить його зручним середовищем для розробки, тому що розробники мають можливість оптимізувати свої ігри без ускладнень;
- цей двигун пропонує різні методи оптимізації, щоб допомогти розробникам поліпшити продуктивність своїх ігор. Наприклад, Godot підтримує техніку оклюзії (occlusion culling), яка допомагає поліпшити продуктивність, запобігаючи рендерінгу об'єктів, які не видно гравцеві. Крім того, Godot пропонує підтримку порталів для поліпшення продуктивності у великих 3D-сценах.
- однією з найбільших переваг Godot є вбудована документація. Розробники можуть швидко знайти інформацію про класи, методи та сигнали, просто клацнувши на них у редакторі. Це збільшує продуктивність та спрощує роботу з двигуном;
- сигнали - це потужний механізм в Godot, який дозволяє об’єктам спілкуватися між собою. Вони дозволяють реагувати на події, такі як натискання кнопок, зіткнення об’єктів або зміни стану;
- Godot підтримує різні типи ресурсів, такі як зображення, звуки, шрифти та інші. Імпорт ресурсів в Godot дуже зручний, і ви можете налаштувати параметри імпорту для кожного ресурсу окремо;
- Godot дозволяє тестувати свої проекти на мобільних пристроях прямо через двигун завдяки потужному інструменту під назвою “Remote

Debug”. Розробники можуть в режимі реального часу відстежувати і аналізувати стан своєї гри навіть коли вона виконується на віддаленому пристрої. Це особливо корисно для виявлення проблем, які виникають тільки на мобільних платформах, таких як просідання FPS, витоки пам'яті або неправильна поведінка інтерфейсу [9, 10, 11, 15, 18].

Хоча Godot Engine має безліч переваг, є деякі області, де він може поступатися Unity або Unreal Engine:

- Godot підтримує 3D, але його можливості в цій галузі не так розвинені, як у Unity або Unreal Engine. Це буде проблемою для ігор зі складною 3D-графікою;
- Godot не має вбудованої підтримки розробки для ігрових консолей;
- порівняно з Unity, інструменти Godot для дизайну рівнів в деяких аспектах менш розвинені;
- порівняно з Unity і Unreal Engine, Godot має менше доступних ресурсів і плагінів через меншу кількість активних користувачів. Але кількість нових розробників на Godot активно росте, тому з часом різниця в доступності плагінів буде зменшуватись.

Усі описані функції роблять Godot потужним інструментом для розробки мобільних ігор, даючи змогу розробникам легко знаходити та виправляти помилки, а також оптимізувати продуктивність своїх ігор [9,10,11].

1.5 Тенденції в розробці мобільних ігор

Важливо зазначити, що ринок мобільних ігор продовжує зростати, а тенденції в галузі швидко змінюються. Наприклад, гіпер-казуальні ігри стали дуже популярними в останні роки. Ці ігри характеризуються простими механіками і короткими сеансами гри, що робить їх ідеально підходящими для мобільних пристроїв [7].

Монетизація є важливою частиною розробки мобільних ігор. На сьогоднішній день існує безліч моделей монетизації, включаючи прямі продажі, внутрішньоігрові покупки, рекламу та підписки. Вибір правильної моделі монетизації значно впливає на успіх гри [16].

Розробка мобільних ігор також має свої виклики. Це включає технічні обмеження мобільних пристроїв, високу конкуренцію на ринку, необхідність постійного оновлення контенту для утримання інтересу гравців, а також виклики, пов’язані з вибором моделі монетизації проекту.

Майбутнє мобільних ігор виглядає дуже обіцяючим. З ростом технологій, таких як штучний інтелект, віртуальна та доповнена реальність, мобільні ігри продовжують розвиватися і пропонувати новий та захоплюючий досвід для гравців. Крім того, з розвитком 5G технологій, мобільні ігри, які потребують високої пропускної здатності та низької затримки, такі як мультиплеєрні онлайн-ігри, стануть ще доступнішими [1, 2, 3].

Виходячи з аналізу сучасного ринку мобільних ігор, можна зрозуміти, що Godot Engine відкриває значні можливості для розробки 2D гри “V-NIGHT:REBOOT” для мобільних пристроїв. Цей двигун володіє рядом привабливих особливостей, які роблять його ідеальним вибором для цього проєкту.

Godot Engine відрізняється своєю гнучкістю, що дозволяє легко адаптуватися до різних вимог гри. Його потужність дозволяє реалізувати складні геймплейні механіки, а активна спільнота розробників є великим джерелом підтримки та ресурсів.

Особливо важливо відзначити, що система вузлів, яка дозволяє легко організовувати та перевикористовувати код, а також створювати складні ієрархії об’єктів, робить процес розробки більш ефективним і менш трудомістким.

Крім того, Godot Engine є безкоштовним та відкритим двигуном, що робить його доступним для всіх розробників. Це відкриває можливості для розробки ігор незалежно від бюджету проекту [10, 11, 18].

2 ОРГАНІЗАЦІЯ ТА ЛОГІКА ПРОЄКТУ

2.1 Організація роботи над проєктом

Перед початком розробки гри було вирішено створити дизайн-документ (Game Design Document). Зазвичай, він робиться в довільній формі, тому немає суворих вимог щодо його написання, але є загальні принципи і цілі, для чого він створюється. В ньому мають бути описані загальна ідея проекту, концепція, ключові механіки, сюжет, ігровий цикл, важливі аспекти для розробників. Цей документ допомагає встановити загальний напрям проекту та допомагає розробникам під час роботи над ним. В підрозділі 2.1.1 представлено частину написаного дизайн-документу, створеного для гри “V-Night” [17].

2.1.1 Дизайн-документ гри V-Night:Reboot

Короткий огляд гри:

V-Night:Reboot – передісторія до оригінальної гри V-Night, де головна героїня розслідувала секрети корпорації V-Corp. Сюжет приквела буде коротким, але доповнюватиме основну історію, де було допущено багато недомовленостей. Це буде 2D-платформер із лінійними рівнями, які матимуть розгалуження, що не впливають на сюжет, і секретні кімнати, в яких лежатимуть нагороди. На шляху героя стоятиме 7 видів звичайних супротивників і 3 супротивники-боси, кожен з яких має свої особливості ведення бою з гравцем. Рівні будуть доповнені інтерактивними предметами, коробками з їжею, що відновлює здоров'я гравця, а також різними пастками і перешкодами. Сюжет приквела буде плавно підводити гравця до основних подій оригінальної гри.

Ігровий цикл і ключові механіки:

Біг: Персонаж біжить рівнем, уникаючи перешкод і пасток.

Збір монет: Персонаж збирає монети, стрибаючи і виконуючи інші дії, подібно до гри Маріо.

Руйнування коробок: Персонаж може руйнувати коробки, використовуючи кнопку атаки, з яких вистрибує їжа.

Атака супротивників: Персонаж може атакувати супротивників, використовуючи кнопку удару, так само він може використовувати блок під час атак супротивника.

Вхід у зони діалогу: Персонаж може входити в зони, де програються діалоги.

Механіка бігу: Персонаж може бігати по рівню, уникаючи перешкод і пасток.

Механіка стрибка: Персонаж може стрибати для збору монет і уникнення перешкод.

Механіка атаки і захисту: Персонаж може атакувати, використовуючи кнопку удару. Це може бути використано для руйнування коробок і атаки супротивників. Атаки противника можна блокувати або уникати за допомогою ривка назад.

Механіка збирання предметів: Персонаж може збирати монети та їжу, що вистрибує з коробок.

Механіка діалогу: Персонаж може входити в зони, де програються діалоги.

Механіка ривка: Персонаж має здатність здійснювати різкий стрибок у бік напрямку свого погляду. Це дає йому змогу швидко й ефективно долати перешкоди. Ривок може бути активований певною кнопкою. Ця механіка додає додатковий рівень складності та стратегії в гру, вимагаючи від гравців швидкого реагування та планування своїх дій.

Опис оточення і персонажів:

- місцем дії гри виступає нічне місто недалекого майбутнього, іменоване як Pulse City. Гравець разом з героїнею зможе пройти більшу частину міста, від околиць до самого центру міста;
- головною героїнею виступає молода дівчина на ім'я Кера, яка відома своїм зухвалим характером і залишковим підлітковим максималізмом;
- противником головної героїні виступає головне управління поліції Pulse City, яке безпосередньо управляється корпорацією V-Corp, замішаною в протиправній діяльності.

2.2 Поняття сцени в Godot

Сцена – це основний будівельний блок у Godot. Вона представляє собою віртуальний простір, в якому розміщуються об’єкти, персонажі, камери та інші елементи гри. Кожна гра в Godot складається з однієї або декількох сцен, які можуть бути завантажені та виведені на екран. Сцени можуть містити ієрархію вузлів (Node), яка визначає взаємозв’язки між об’єктами [19].

Після створення сцени можна додавати до неї вузли та налаштовувати їх властивості. Всі об’єкти в сцені є вузлами (Nodes). Вони можуть бути графічними об’єктами, камерами, світлами, колайдерами та іншими елементами. Кожен вузол може мати дочірні вузли, що створює ієрархію. Властивості вузлів можна налаштовувати через інспектор властивостей. Також кожна сцена має один кореневий вузол (Root Node). Він є головним вузлом сцени. Всі інші вузли в сцені є дочірніми вузлами кореневого вузла (рис. 2.1).

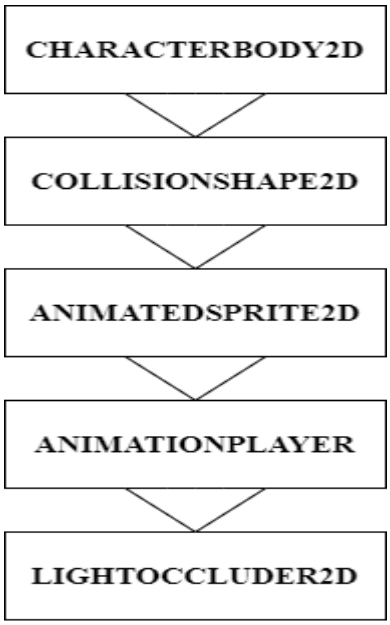

Рисунок 2.1 – Структура сцени головного героя

CharacterBody2D: спеціалізований вузол фізичного тіла для 2D персонажів, який дозволяє обробляти переміщення об’єктів в фізичному просторі. Вбудовані методи цього класу дозволяють гнучко налаштовувати поведінку ігрових героїв. Зазвичай в ігрових сценах цей вузол використовується як головний.

CollisionShape2D: вузол, який надає усім об’єктам абстрактного класу CollisionObject2D форму для виявлення стиків з іншими об’єктами під час гри.

AnimatedSprite2D: Вузол, який дозволяє створювати прості 2D-анімації. Анімації створюються за допомогою ресурсу SpriteFrames, який дозволяє імпортувати файли зображень для надання кадрів анімації спрайту.

AnimationPlayer: вузол для просунутого налаштовування анімацій, наприклад, можна налаштувати час переходу однієї анімації на іншу, змішувати їх для більш плавного переходу та має багато вбудованих методів та властивостей, які дозволяють програмістам працювати над поведінкою ігрових героїв.

LightOccluder2D: вузол, який використовується для затінення світла, що випромінюється від вузлів Light2D, створюючи тіні. Фактично виступає як форма, яка блокує світло [19, 21, 22].

2.3 Організація збереження даних

В Godot Engine нема традиційної бази даних, замість цього, усі дані та логіка проекту організовані в вузли та побудовані на них сцени.

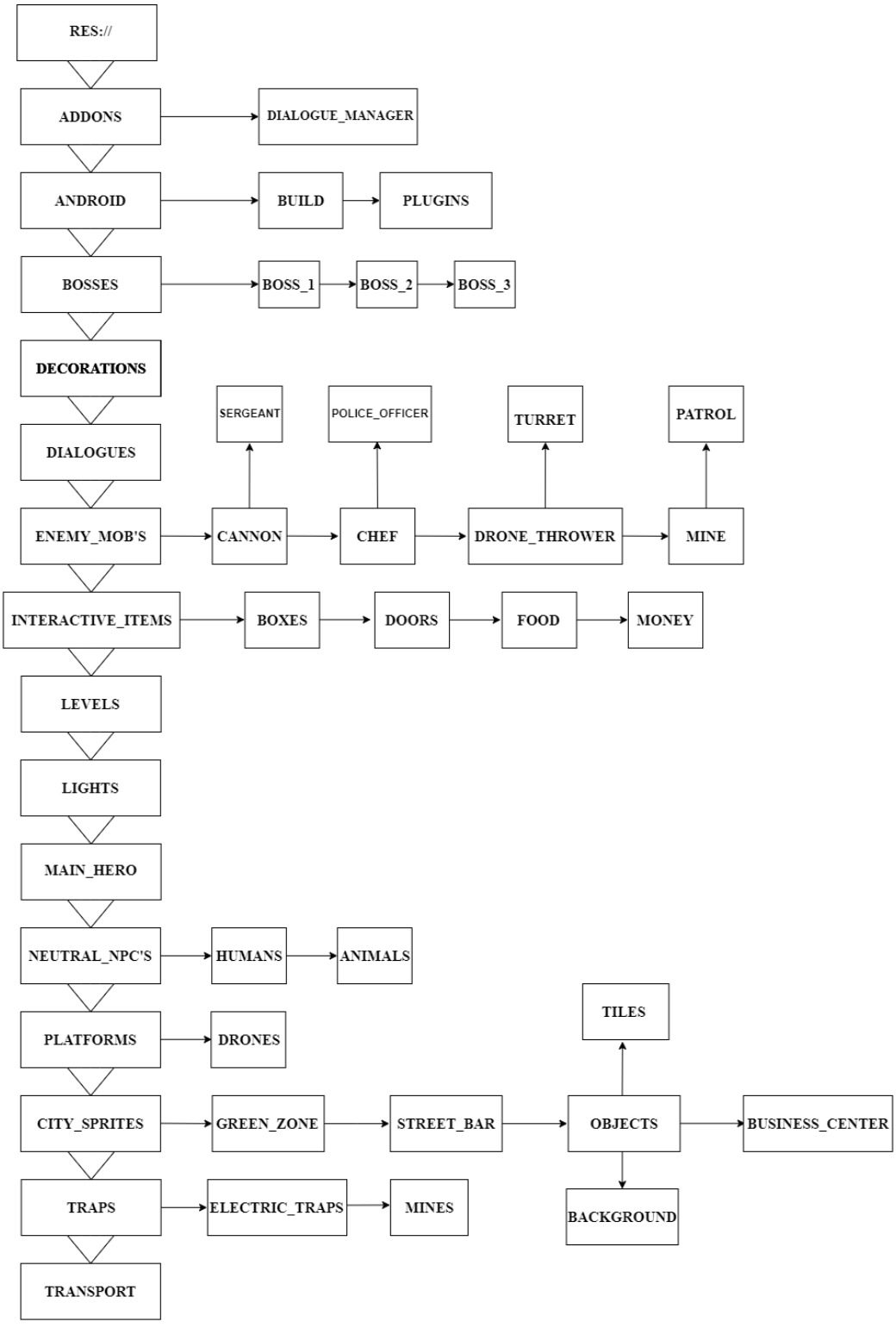

Рисунок 2.2 – Структура даних проєкту

Збережені дані проекту гри “V-Night” розміщені в директоріях, які строго поділені по типам ігрових ресурсів та об’єктів (рис. 2.2). В Цих директоріях розміщені усі ігрові актив:

- директорія Addons містить сховище для доданих активів з бібліотеки AssetLib, де інші розробники та ентузіасти розміщують свої користувацькі розширення та додатки, що дозволяє користувачам отримати доступ до готових технічних рішень. Розширення, яке використовується в проекті “V-Night”, це готова діалогова система, де розробники можуть швидко конфігурувати діалоги між персонажами та виставляти різні параметри, наприклад, швидкість друкування тексту, паузи та інше;
- директорія Android зберігає в собі файли, необхідні для експорту проекту на платформу Android та файли плагінів. Завдяки плагінам розробники можуть налаштувати обробку внутрішньоігрових транзакцій, показ реклами та інше;
- папка Bosses зберігає в собі спрайти та готові сцени для “NPC” (non-player character), які виступають особливо складними суперниками для гравця;
- директорія Enemy_mobs є сховищем для усіх ігрових противників гравця. В цій директорії знаходяться ще 8 папок, за кожною з яких закріплені файли одного з “NPC”;
- директорія Interactive_items зберігає всі ігрові предмети, з якими буде взаємодіяти гравець. Наприклад, двері, які відкриваються за допомогою кнопки, коробки, з яких випадає їжа, що відновлює здоров’я гравця та інше;
- в Levels знаходяться сцени з побудованими ігровими рівнями та іншими сценами, пов’язаними з ними;
- в папці Lights зберігаються сцени з налаштованими джерелами світла;

- в Main_hero знаходиться сцена з персонажем гравця, камера, сцени з візуальними ефектами для головного героя;
- в папці Neutral_NPC’s є дві директорії для людей та тварин. Тварини виступають декораціями та оживляють світ навколо гравця, а не ігрові персонажі людей взаємодіють з гравцем;
- папка Platforms зберігає сцени з платформами, по яким буде стрибати гравець;
- директорія City_sprites містить в собі всі спрайти декорацій міста для ігрових рівнів;
- Traps зберігає сцени з ігровими статичними пастками (рис. 2.3).

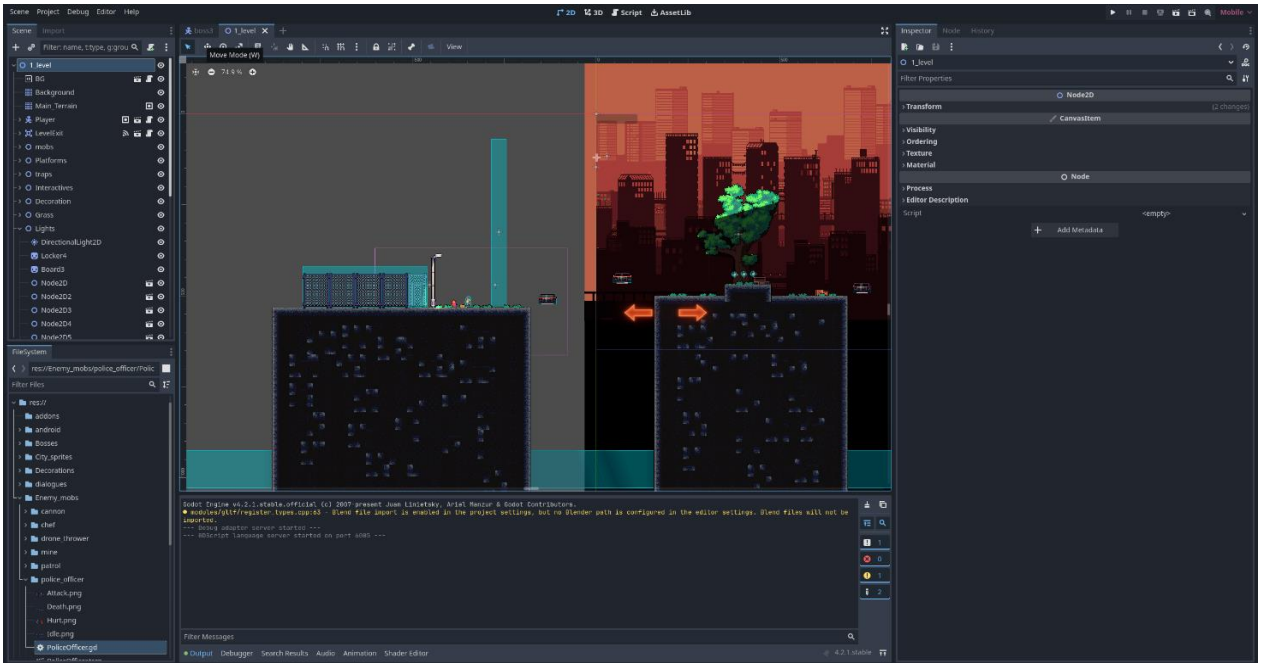

Рисунок 2.3 – Середовище розробки

Після створення сцен персонажів та ігрових об’єктів черга дійшла до програмування. Для 2D-платформера “V-Night” було вирішено обрати мову програмування GDScript. GDScript – це високорівнева, об’єктно-орієнтована, імперативна мова програмування, спеціально створена для двигуна Godot Engine. Вона була розроблена таким чином, щоб бути простою в розумінні для новачків-програмістів, але при цьому ні в чому не поступатися іншим мовам

програмування в потужності та можливісті розширення GDScript власними функціями, класами та модулями.

Також ця інтегрована прямо в двигун мова програмування забезпечує відмінну продуктивність завдяки JIT(Just-In-Time)-компіляції. Прикладом основних методів цієї мови програмування, з якими одразу зустрічаються програмісти-новачки є методи: _ready(), _process(delta) та _physics_process(delta).

Метод _ready() викликається один раз при створенні вузла, за яким закріплений скрипт. Зазвичай його використовують для ініціалізації змінних, завантаження ресурсів або налаштування компонентів.

Метод _process(delta) викликається на кожному кадрі та дозволяє завдяки цьому оновлювати стан вузла. Параметр delta являє собою значення різниці часу між минулим і поточним кадром. Але цей метод потрібно використовувати для обробки логіки, яка не залежить від фізики, наприклад, для обробки програвання анімацій.

Метод _physics_process(delta) також викликається на кожному кадрі, але він синхронізований з фізичними обчисленнями. Тому значення delta в даному методі є постійним. Це дозволяє обробляти фізичну логіку, наприклад, переміщення, стикання об’єктів, гравітацію та інше [23].

Для даної гри скрипт головного героя має містити обробку користувацького введення для переміщень героя; самі переміщення в двомірному просторі; програвання анімацій залежно від того, що робить персонаж в даний проміжок часу; гравітацію, силу якої можна налаштовувати для різних ситуацій, що урізноманітнює гру; кількість одиниць здоров’я героя, яка буде постійно змінюватись протягом проходження рівня; функції для отримання шкоди та відновлення здоров’я; обробку елементів бою із суперниками і обробку взаємодії з навколишнім середовищем. Для скриптів суперників було вирішено зробити зону, в якій гравець буде становитись помітним для них, та буде починатись переслідування гравця з метою почати атакувати його як тільки він виявиться в зоні ураження. Для цього суперникам

також потрібно надати фізичні властивості для взаємодії з оточуючим середовищем, створити обробку усіх ситуацій, в яких можуть опинитися не ігрові персонажі під час сесії.

Механізм атаки гравця створено таким чином, щоб він був досить чутливим, щоб гравець розумів, коли саме відбудеться атака супротивника та мав час на вжиття заходів захисту. Для цього, крім функціональної роботи потрібно провести роботу над візуальною частиною анімації атаки. Для урізноманітнення навколишнього середовища створено декілька скриптів для реалізації функціональних статичних пасток, літаючих дронів та людей, які населяють місто.

3 ТЕСТУВАННЯ ПРОЄКТУ

3.1 Виправлення логічних помилок коду

Після написання коду та створення першого і тестового рівня почались тести в режимі налагодження в самому двигуні. Для цього в скриптах ігрових персонажів були прописані строки print() для виводу інформації в термінал про стан змінних-перемикачів, які контролюють поведінку штучного інтелекту та героя, яким керує гравець. Цей метод налагоджування називається “логуванням” [21,22]. Вже під час перших ігрових тестів виявились проблеми в логіці поведінки не ігрових персонажів та їх переміщенням. Виявилось, що гравець може взагалі уникати контакту з супротивниками та проходити рівень безпосередньо, без зайвих зусиль. Для вирішення цієї проблеми достатньо було підвищити швидкість переміщення супротивників, щоб вони завжди наздоганяли гравця і вимушували його починати з ними бій.

Враховуючи, що гра є платформером, ландшафт в ній виступає в якості багатьох островів різного розміру, між якими знаходяться рухливі платформи, завдяки яким герой може рухатись далі [21]. Але якщо ворожий персонаж знаходився на краю прірви, а гравець своїм ударом виштовхував його в прірву, то ворожий персонаж просто починав летіти в прірву вниз без кінця. Рішенням цієї проблеми стало створення колізії під рівнем, для якої був створений скрипт, який враховував завдяки вбудованому сигналу body_entered(Body: Node2D), що якщо об’єкт, який потрапив в цю область, знаходиться в групі “player”, то рівень починається з останньої точки збереження, а якщо об’єкт відноситься до групи “enemies”, то викликається внутрішній метод queue_free(), який безпечно видаляє вузол супротивника.

Для покращення бойового циклу в грі було вирішено змінити момент, коли гравець отримує шкоду під час атаки ближнього бою супротивника. Завдяки вбудованому сигналу frame_changed() вузла AnimatedSprite2D було

налаштовано, що шкоду гравець буде отримувати тоді, коли в анімації атаки настане 4 кадр, а не перший за замовчуванням, та гравець при цьому буде знаходитись в зоні ураження.

3.2 Особливості адаптації для мобільних пристроїв

Після проведення успішних ігрових тестів в Godot Engine, почалися тести на мобільних пристроях. Для цього був створений тестовий користувацький інтерфейс для можливості гри на смартфонах, та в налаштуваннях Godot був завантажений шаблон експорту для платформи Android. Завдяки інструменту під назвою “Remote Debug ”, описаному в першому розділі, при ініціалізації гри на смартфонах середнього цінового діапазону виявилось, що середня частота кадрів на першому рівні не перевищує 25 одиниць (рис. 3.1).

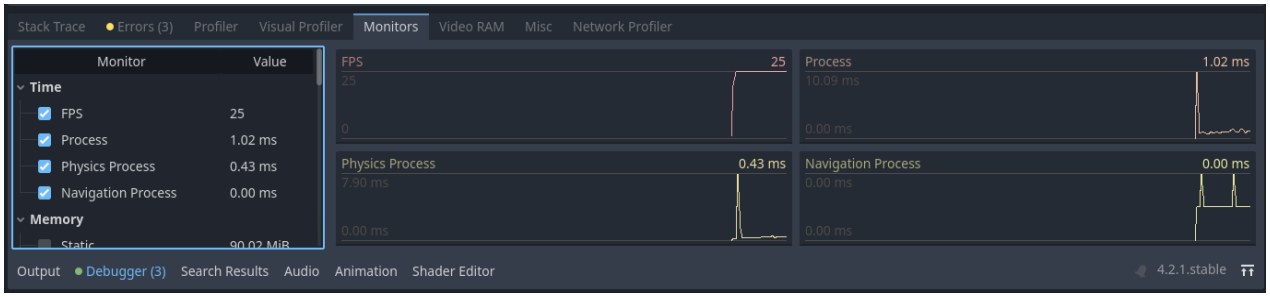

Рисунок 3.1 – Монітори налагодження

Для вирішення проблеми низької частоти кадрів потрібно було проаналізувати всі розміщені на сцені вузли та те, скільки ресурсів вони використовують. Найпростішим методом такого аналізу стало почергове вимкнення вузлів, які розміщені на сцені, запуск гри в режимі налагодження на смартфоні, перегляд показників частоти кадрів та значення функції process. Важливо дивитись не тільки на показник частоти кадрів, а і на інтервальні значення між кадрами, які означають час, за який завершуються функції process

та physics_process. Інтервал між кадрами при частоті кадрів 60 становить приблизно 16,67 мілісекунд.

Шляхом описаних вище тестів було виявлено, що найбільше на частоту кадрів на першому рівні впливав вузол DirectionalLight2D. DirectionalLight2D – це вузол, який моделює нескінчену кількість паралельних променів, які охоплюють всю сцену. Такий тип освітлення застосовується для імітації сонячних або місячних променів. Але через високі графічні налаштування цього вузла, він дуже сильно впливав на продуктивність гри на мобільних пристроях.

Далі було з’ясовано, що побудований перший рівень для гри є занадто великим, тому було прийнято рішення змінити структуру рівнів, зробити їх набагато меншими з постійними безшовними переходами між ними. Описаний в першому розділі метод оптимізації під назвою “occlusion culling” не працює в 2D проектах, тому для покращення продуктивності гри було вирішено змінити скрипти не ігрових персонажів, а саме зробити систему відсікання із застосуванням програмного коду.

Якщо гравець є на певному віддаленні від не ігрового персонажа, наприклад, в 500 одиниць, то прописаний код в функціях process та physics_process цих персонажів просто не буде оброблятись завдяки ключовому слову pass. Зміна налаштувань глобального світла, зміна структури рівнів та оптимізація на рівні коду дозволила досягти потрібного рівня продуктивності гри. Показники в функції process все одно інколи перевищують значення в 16,67 мілісекунд, але такі рідкісні стрибки цього показника є нормальним процесом.

ВИСНОВКИ

У кваліфікаційній роботі розроблено та реалізовано 2D гру "V-NIGHT:REBOOT" для мобільних пристроїв на Godot Engine, який відрізняється своєю гнучкістю, що дозволяє легко адаптуватися до різних вимог гри. Потужність двигуна дозволяє реалізувати складні геймплейні механіки, а активна спільнота розробників є великим джерелом підтримки та ресурсів.

Система вузлів Godot Engine дозволяє легко організовувати та перевикористовувати код, а також створювати складні ієрархії об’єктів, робить процес розробки більш ефективним і менш трудомістким.

У процесі розробки були розглянуті основні концепції та інструменти Godot Engine, що дозволили ефективно реалізувати ігрові механіки та дизайн гри.

Описані в роботі методі оптимізації та налагодження дозволили провести тестування та налагодження гри.

Тож, результатом кваліфікаційної роботи є розроблена гра засобами Godot Engine 4 для мобільних пристроїв.

Ця робота буде корисною для ігрових розробників та студентів, які цікавляться розробкою мобільних ігор.

ПЕРЕЛІК ПОСИЛАНЬ

1. Shergil A. Mobile Game Development Trends To Consider In 2024. 2023. URL: https://www.topdevelopers.co/blog/mobile-game-development-trends/. (дата звернення: 13.02.2024)

2. Cracking the Code: CrazyLabs' Gaming Experts Reveal 2024 Mobile Gaming Trends. 2024. URL: https://www.crazylabs.com/blog/gaming-experts-reveal-mobile-gaming-trends/. (дата звернення: 15.02.2024)

3. Mobile gaming trends of 2024: Top 7 developments set to define the year. 2024. URL: https://www.mistplay+.com/resources/mobile-gaming-trends-2024. (дата звернення: 17.02.2024)

4. Number of mobile app downloads worldwide from 2016 to 2023. URL: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. (дата звернення: 20.02.2024)

5. Usage share of operating System. URL: https://en.wikipedia.org/wiki/Usage_share_of_operating_systems. (дата звернення: 20.02.2024)

6. Mobile & Tablet Android Version Market Share Worldwide - April 2024. URL: https://gs.statcounter.com/os-version-market-share/android/mobile-tablet/worldwide. (дата звернення: 20.02.2024)

7. AppTweak., Most popular mobile game genres in 2022. URL: https://www.businessofapps.com/insights/most-popular-mobile-game-genres-in-2022/. (дата звернення: 25.02.2024)

8. Kriebernegg T. The best 22 mobile game engines and development platforms. 2023, 12 p. URL: https://appradar.com/blog/mobile-game-engines-development-platforms. (дата звернення: 01.03.2024)

9. Wirtz B. Unity vs. Godot Engine: Performance and Community Comparison. 2023. URL: https://www.gamedesigning.org/engines/unity-vs-godot/. (дата звернення: 03.03.2024)

10. Introduction to Godot Engine. URL: https://www.educba.com/godot-engine/. (дата звернення: 05.03.2024)

11. Tkachenko K. Godot Engine's Rise to Prominence in 2024: Revolutionizing Mobile Game Development. URL: https://www.gamelight.io/post/godot-engine-s-rise-to-prominence-in-2024-revolutionizing-mobile-game-development. (дата звернення: 13.03.2024)

12. Linietsky J. A decade in retrospective and future. URL: https://godotengine.org/article/retrospective-and-future/. (дата звернення: 14.03.2024)

13. Galla N. GDC 2024: Retrospective. URL: https://godotengine.org/article/gdc-2024-retrospective/. (дата звернення: 15.03.2024)

14. Zenva., GDScript Basics Tutorial – Complete Guide., URL: https://gamedevacademy.org/gdscript-basics-tutorial-complete-guide/. (дата звернення: 18.03.2024)

15. Godot 4.0 sets sail: All aboard for new horizons. 2023. URL: https://godotengine.org/article/godot-4-0-sets-sail/. (дата звернення: 01.04.2024

16. 5 of the Most Successful Mobile Game Monetization Strategies for 2022. URL: https://newormedia.com/blog/mobile-game-monetization-strategies/. (дата звернення: 03.04.2024)

17. How to Write a Game Design Document (GDD). 2023. URL: https://www.nuclino.com/articles/write-game-design-document. (дата звернення: 27.04.2024)

18. Introduction to Godot. 2024. URL: https://docs.godotengine.org/en/stable/getting_started/introduction/introduction_to_godot.html. (дата звернення: 01.05.2024)

19. Nodes and Scenes. 2024. URL: https://docs.godotengine.org/en/stable/getting_started/step_by_step/nodes_and_scenes.html. (дата звернення: 05.05.2024)

20. GDScript reference. 2024. URL: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html. (дата звернення: 10.05.2024)

21. 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, Edition 2. Packt Publishing Ltd. 2023, 264 с.

22. Buckley D. Godot 4 for Beginners. Zenva Pty Ltd. 2024, 70 с.

ДОДАТОК А

Частина коду гравця

```
extends CharacterBody2D

class_name PlatformerController
signal jumped(is_ground_jump: bool)
signal hit_ground()
signal health_changed(new_health)
# Set these to the name of your action (in the Input Map)
## Name of input action to move left.
@export var input_left : String = "move_left"
## Name of input action to move right.
@export var input_right : String = "move_right"
## Name of input action to jump.
@export var input_jump : String = "jump"
@onready var anim = get_node("AnimationPlayer")
@onready var anim_sprite = get_node("AnimatedSprite2D")
#@onready var anim_effects = get_node("Effects")
@onready var save_game = get_tree().get_nodes_in_group("SaveGame")
#PLAYER PARAMETERS
@export var attack_range = 50
@export var attack_damage = 10
var health
@export var max_health = 100
var hurt = false
var old_health = max_health
var can_punch = true
var can_move = true
var interacting_with_panel = false
var punch_cooldown = 0.5
var punch_timer = 0.0
var is_attacking = false
var fall_timer = 0.0
var fall_delay_timer = 0.0
#bounce variables
@export var bounce_y = 500
@export var bounce_x = 5000
@export var bounce_x_damage_area = 3500
@export var bounce_y_damage_area = 300
@export var shot_bounce_y = 550
var is_bouncing = false
var is_bouncing_y_axis = false
```

```
@onready var enemies = get_tree().get_nodes_in_group("enemies")
@onready var boss = get_tree().get_nodes_in_group("boss")
const DEFAULT_MAX_JUMP_HEIGHT = 70
const DEFAULT_MIN_JUMP_HEIGHT = 35
const DEFAULT_DOUBLE_JUMP_HEIGHT = 35
const DEFAULT_JUMP_DURATION = 0.3
var original_collision_layer
var original_collision_mask
var coins = 0
func _ready():
	health = max_health
	old_health = max_health
	add_to_group("Player")
	if is_coyote_time_enabled:
		add_child(coyote_timer)
		coyote_timer.wait_time = coyote_time
		coyote_timer.one_shot = true

	if is_jump_buffer_enabled:
		add_child(jump_buffer_timer)
		jump_buffer_timer.wait_time = jump_buffer
		jump_buffer_timer.one_shot = true
func _physics_process(delta):
	if health < old_health and not hurt:
		if is_attacking:
			anim.stop()
			is_attacking = false
		anim.play("hurt")
		hurt = true
	old_health = health
	if not can_punch:
		punch_timer += delta
		if punch_timer >= punch_cooldown:
			can_punch = true
			punch_timer = 0.0
	var target_velocity = Vector2.ZERO
	if interacting_with_panel:
		return
	if !can_move:
		return
	if fall_delay_timer > 0:
		fall_delay_timer -= delta
	if not is_attacking:
		if Input.is_action_pressed(input_left):
			target_velocity.x = -max_acceleration
```

```
        elif Input.is_action_pressed(input_right):
                target_velocity.x = max_acceleration
        else:
                if not is_bouncing:
                        target_velocity.x = 0
        velocity.x = lerp(velocity.x, target_velocity.x, 0.6)
velocity.y += acc.y * delta
if not is_bouncing:
        if velocity.x < 0:
                $AnimatedSprite2D.flip_h = true
                #anim_effects.flip_h = true
        elif velocity.x > 0:
                $AnimatedSprite2D.flip_h = false
                #anim_effects.flip_h = false
if is_coyote_timer_running() or current_jump_type == JumpType.NONE:
        jumps_left = max_jump_amount
if is_feet_on_ground() and current_jump_type == JumpType.NONE:
        start_coyote_timer()
# Check if we just hit the ground this frame
if not _was_on_ground and is_feet_on_ground():
        current_jump_type = JumpType.NONE
        if is_jump_buffer_timer_running() and not can_hold_jump:
                jump()
        hit_ground.emit()
        is_bouncing = false
        is_bouncing_y_axis = false

# Cannot do this in _input because it needs to be checked every frame
if Input.is_action_pressed(input_jump):
        if can_ground_jump() and can_hold_jump:
                jump()
var gravity = apply_gravity_multipliers_to(default_gravity)
acc.y = gravity
# Play the animations
if not is_attacking and not hurt and not interacting_with_panel:
        if abs(velocity.x) > 10 and is_feet_on_ground():
                anim.play("run")
                #if not hurt and not is_bouncing:
                        #anim_effects.play("walk_effect")
                        #anim_effects.show()
        if abs(velocity.x) < 50 and is_feet_on_ground():
                anim.play("idle")
                #anim_effects.visible = false
        if not is_feet_on_ground():
                if velocity.y < 0:
```

```
                        anim.play("jump")
                        #anim_effects.visible = false
                else:
                        if anim.current_animation != "hurt" and fall_timer <= 0 and
not is_bouncing:
                                anim.play("fall")
        else:
                # код для обработки получения урона и отскока игрока
                if health < old_health and not hurt:
                        hurt = true
                        #anim_effects.hide()
                        bounce_player(enemies)
                old_health = health
        if health <= 0:
                respawn()
        _was_on_ground = is_feet_on_ground()
        if !can_move:
                return
        move_and_slide()
```

ДОДАТОК Б

Посилання на інші скрипти проєкту

Детально ознайомитись із скриптами гри можна по зазначеному посиланню:
https://drive.google.com/drive/folders/1OtogZDkvweqaLiMZczvbbi7Vb9kvQ0lq?usp=drive_link