МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА 2D ГРИ З ВИКОРИСТАННЯМ ІРОВОГО РУШІЯ GODOT»**

Виконав: студент 4 курсу, групи
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Є. Є. Федотов
(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 доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

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., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Федотову Євгенію Євгеновичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка 2D гри з використанням ігрового рушія Godot

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

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025

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

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

1. Актуальність Godot у сучасній інді-ігровій індустрії.
2. Підготовка до розробки 2D гри.
3. Розробка 2D гри.
4. Тестування 2D 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 23.12.2024

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.	23.12.2024	
2.	Збір вихідних даних.	10.01.2025	
3.	Обробка методичних та теоретичних	06.04.2025	
	джерел.		
4.	Розробка першого та другого розділу.	25.04.2025	
5.	Розробка третього та четвертого розділу.	20.05.2025	
6.	Оформлення та нормоконтроль	02.06.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ Є.Є. Федотов
(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 з використанням ігрового рушія Godot»: 56 с., 16 рис., 8 табл., 16 джерел, 8 додатків.

ВУЗОЛ, ГРА, ІНДІ, ІГРОВИЙ РУШІЙ, СЦЕНА, AUTOLOAD, 2D, 3D, GODOT.

Об’єкт дослідження – процес розробки 2D ігор з використанням ігрового рушія Godot.

Мета роботи: розробити 2D гру, використовуючи ігровий рушій Godot та продемонструвати можливості рушія.

Предмет дослідження – особливості розробки гри з використанням ігрового рушія Godot.

Метод дослідження – аналіз, синтез, абстрагування, порівняння, індукція

Кваліфікаційна робота присвячена дослідженню процесу створення 2D гри з використанням ігрового рушія Godot. Актуальність теми зумовлена зростанням популярності інді-ігор в сучасній ігровій індустрії, які потребують доступних інструментів розробки. В процесі розробки, було детально розглянуто ключові аспекти і особливості використання рушія. У межах розробки, було створено 2D гру, що демонструє основні можливості рушія, зокрема анімацію, фізику, логіку керування подіями, систему сцен та інші інструменти. Проведено тестування гри, що дозволило оцінити стабільність роботи рушія та його придатність для розробки інді-ігор.

Результати роботи можуть бути корисними як для початківців, що вивчають ігрову розробку, так й для досвідчених розробників, що шукають альтернативу традиційним рушіям. Також, отриманий проєкт може бути використаний в освітніх програмах студентів, що навчаються програмуванню та/або розробці ігор.

SUMMARY

Bachelor's Qualification Thesis «Development of a 2D Game Using the Godot Game Engine»: 56 pages, 16 figures, 8 tables, 16 references, 8 supplements.

NODE, GAME, INDIE, GAME ENGINE, SCENE, AUTOLOAD, 2D, 3D, GODOT.

Object of the study – the process of developing 2D games using the Godot game engine.

Aim of the study: develop a 2D game using the Godot game engine and demonstrate the engine's capabilities.

Methods of research – analysis, synthesis, abstraction, comparison, induction.

The qualification work is devoted to the study of the process of creating a 2D game using the Godot game engine. The relevance of the topic is due to the growing popularity of indie games in the modern gaming industry, which require affordable development tools. During the development process, the key aspects and features of the engine were considered in detail. As part of the development, a 2D game was created to demonstrate the main features of the engine, including animation, physics, event management logic, scene system and other tools. The game was tested to evaluate the stability of the engine and its suitability for indie game development.

The results of the work can be useful and illustrative for both beginners studying game development and experienced developers looking for an alternative to traditional engines. Also, the resulting project can be used in the educational programmes of students studying programming or game development.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Скорочення та умовні познаки 8
Вступ..... 9
1 Актуальність Godot у сучасній інді-ігровій індустрії 11
1.1 Актуалізація інді-ігрової індустрії 11
1.2 Загальний огляд Godot..... 12
1.3 Порівняння Godot з іншими рушіями 14
2 Підготовка до розробки 2D гри 20
2.1 Аналіз вимог та постановка задач 20
2.2 Архітектура гри 21
3 Розробка 2D гри..... 26
3.1 Початкове налаштування проєкту..... 26
3.2 Головний персонаж..... 27
3.3 Союзники 30
3.4 Ворог..... 31
3.5 Предмети, що підбираються гравцем 32
3.6 Об’єкти з якими взаємодіє гравець 32
3.7 Створення рівнів 34
3.8 Графічний інтерфейс 35
3.8.1 Меню та налаштування 37
3.8.2 Ігровий інтерфейс 38
3.9 Збереження даних 40
3.10 Експорт на Windows та Android 41
4 Тестування 2D гри..... 43
4.1 Продуктивність на різних пристроях..... 43

Висновки 46
Перелік посилань 47
Додаток А Класова архітектура рушія Godot 49
Додаток Б Функція переходу скінченного автомату ворога 50
Додаток В Заміри продуктивності на ПК1 51
Додаток Г Заміри продуктивності на ПК2 52
Додаток Д Заміри продуктивності на ПК3 53
Додаток Е Заміри продуктивності на ПК4 54
Додаток Ж Заміри продуктивності на Realme 6 Pro в режимі рендеру Mobile 55
Додаток И Заміри продуктивності на Realme 6 Pro в режимі рендеру Compatibility 56

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

2D – двовимірна графіка

3D – тривимірна графіка

ПК – персональний комп’ютер

ВСТУП

Сучасний розвиток ігрової індустрії супроводжується зростанням популярності інді-ігор, [1, 2] які в свою чергу розробляються малими командами розробників або однією людиною. Зростання їх популярності можна пов’язати з невибагливими системними вимогами до комп’ютерів гравців, унікальними графічними стилями, механіками та іншими особливостями на відміну від ігор, що розробляються масштабними компаніями. Розробка подібних проєктів потребує ігрового рушія, який не буде складним в опануванні та не потребуватиме дорогого обладнання. Одним з таких є Godot.

Ігровий рушій Godot бере свій початок з 2007 року, як закритий програмний продукт, в деяких компаніях. Починаючи з 2014 року, починає розповсюджуватися під ліцензією MIT на платформі GitHub. За цей період часу на Godot розроблена незначна кількість проєктів, які дісталися широких мас [3, 4]. Теоретично це можна пов’язати з його невеликою популярністю, порівняно з іншими ігровими рушіями, які вже встигли закріпитися на ринку (Unity, Unreal Engine тощо) з плином часу. Як наслідок, Godot має значно меншу кількість розробленого навчального матеріалу (особливо беручи до уваги останні версії рушія), окрім офіційної документації, що може відштовхнути початківців. Однак, в довго тривалій перспективі Godot може запропонувати значні переваги, порівняно з іншими ігровими рушіями, як наприклад повністю відкритий код та відсутність будь-яких тарифів при будь-якому доході.

Мета кваліфікаційної роботи – розробка 2D гри, використовуючи Godot, тим самим демонструючи можливості рушія.

Тому можна виділити наступні етапи виконання кваліфікаційної роботи:

– дослідити актуальність проблеми;
– дослідити основні положення про ігровий рушій GODOT;
– створення 2D гри;
– оцінка функціональності інструментів рушія;

– проведення тестування гри на різних пристроях та платформах.

Отже, детально розглянуто процес створення 2D гри, яка демонструватиме можливості рушія та матиме перспективи подальшого розширення та публікації на платформах цифрової дистрибуції.

1 АКТУАЛЬНІСТЬ GODOT У СУЧАСНІЙ ІНДІ-ІГРОВІЙ ІНДУСТРІЇ

1.1 Актуалізація інді-ігрової індустрії

Появою інді-ігор фактично можна вважати період з 1970 по 1980 роки, коли вони створювалися індивідуальними розробниками та невеликими студіями, ще для тогочасних комп’ютерів, як ZX Spectrum чи MSX. На початку їх розповсюдження відбувалося за рахунок касет, потім компакт-дисків. Але найбільшим поштовхом, можна вважати появу цифрового магазину ігор Steam [5, 6].

Особливо популярність інді-ігор зросла протягом останнього десятиліття. Опираючись на дослідженнях VG Insights [7] (рис. 1.1), співвідношення продажу інді-ігор і AA та AAA ігор, тривалий час знаходилася на рівні близько 50%, з періодичною перевагою, останніх.

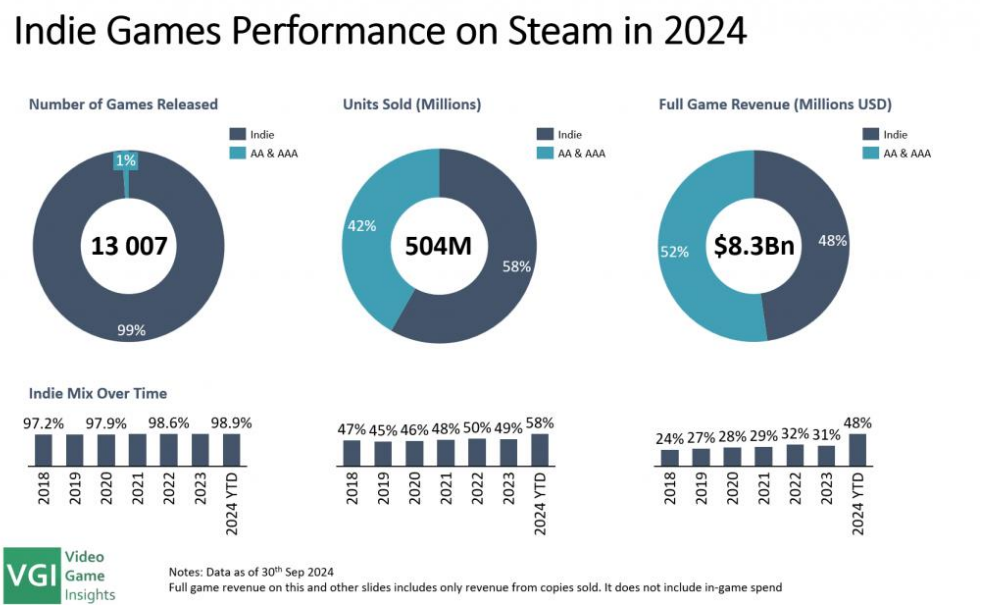


Рисунок 1.1 – Порівняльне дослідження VG Insights

Однак за 2024 рік, показник вперше перейшов на бік інді-проєктів зі значенням в 58%. Не менш важливе значення відіграє показник прибутковості –

інді-ігри становлять 48%, від загального значення прибутку за 2024 рік, що вдвічі більше, ніж за 2018 рік та 17%, порівняно з 2023. До того ж, дослідження не враховує мікро транзакцій, що почали набувати популярності останні роки, відповідно значення прибутку може бути дещо більшим.

Крім загального розростання, прибутковості інді-проєктів, слід відмітити, що показник їх кількості, не ставав меншим від 97% й стрімко наблизився до показника в майже 99% загальної кількості ігор, що виходили за рік.

Однак з часом, поняття інді-проєкту стало доволі широким за визначенням, враховуючи навіть лише численність розробників, число яких може варіюватися від 1-2 осіб до команд, які налічують більше ніж 50 розробників. Проте саме кількість випущених ігор, має найбільші показники, серед розробників, які працюють самостійно. Звісно кількість не є еквівалентною якості, але більшість найбільш відомих та значущих інді-ігор, було створено саме однією людиною. До таких можна віднести Balatro (номінована на нагороду “Гра року” за версією The Game Awards 2024), Braid, Minecraft, Undertale та Stardew Valley.

Кожен з цих проєктів відрізняється своєю креативністю та унікальністю геймплею. Подібні переваги інді-ігор в цілому, також відмічені користувачами, що підтверджується опитуванням, котре проводилося, ще в 2012 році, Робіном Кейзером [8], хоч і на малій групі респондентів (104 особи).

Однак, розробка гри потребує не лише креативної ідеї, а також вибору зручного, функціонального та доступного ігрового рушія. Одним з таких можна назвати Godot.

1.2 Загальний огляд Godot

Godot є сучасним, кросплатформеним ігровим рушієм, розробленим на C++ для створення 2D та 3D ігор з відкритим вихідним кодом, починаючи з 2014 року. З 2007 по 2014, був закритим програмним продуктом.

Однією з основних переваг рушія, є його кросплатформеність, що дозволяє експортувати гру на такі платформи, як Windows, Linux, macOS, Android, iOS та HTML5. Серед консолей, офіційно Godot підтримує лише Steam Deck, оскільки він підтримує офіційні шаблони експорту Linux. Розробка під інші популярні платформи, як Sony PlayStation 4 та 5, Xbox X та S, Nintendo Switch, є можливою, але потребує дозволу особисто від компанії розробника консолі, через те, що Godot поширюється, як рушій з відкритим вихідним кодом.

Godot підтримує декілька мов програмування – GDScript та C#. Використовуючи GDExtension, також стає можливим використання C++ та C. Ці мови офіційно підтримуються розробниками рушія, однак спільнота створює власні розширення, які дозволяють використовувати інші мови.

GDScript є об’єктно-орієнтованою, імперативною мовою програмування високого рівня з послідовною типізацією, розроблено спеціально для Godot. Мова має синтаксис, схожий на мову Python (базується на відступах). Оскільки мова є розробленою, спеціально для рушія, код має достатню швидкість виконання. Звісно, якщо використовувати скомпільований код мовами C# або C++, можна досягти вищої продуктивності, однак більшість розроблених методів, викликають алгоритми розроблені саме на C++. Відповідно, в більшості задач, відсутня явна потреба використовувати іншу мову, замість GDScript, для досягнення вищої продуктивності. До того ж, використовуючи C#, наразі існують певні обмеження на різних версіях Godot, пов’язані з експортом проєктів на різні платформи.

Ще одною технологією Godot є GDExtencion. Він дозволяє розшити функціонал рушія, інтегруючи зовнішній код, написаний мовою програмування з використанням GDExtension API, тим самим надаючи можливість створювати власні модулі та бібліотеки, які взаємодіють з рушієм на низькому рівні, не змінюючи його внутрішнього коду. Відповідно, такий підхід вимагає лише компіляцію розробленої бібліотеки, а не рушія в цілому, що довелося б робити в випадку з модулями C++.

Отже, можна сказати напевно, що Godot є доволі комплексно підготовленим рушієм, який забезпечує широкий спектр можливостей. Проте слід провести мінімальне порівняння з іншими популярними ігровими рушіями на даний час.

1.3 Порівняння Godot з іншими рушіями

Найдоцільніше почати порівняння з мінімальних системних вимог ігрових рушіїв та їх можливостей. Для порівняння обрано 6 ігрових рушіїв за кількістю створених ігор в порядку спадання, за даними SteamDB [9], а саме Unity, Unreal Engine, GameMaker, RPGMaker, RenPy, та Godot.

Зібрані дані представлені у вигляді порівняльної таблиці (табл. 1.3.1) для аналізу даних.

Таблиця 1.1 – Порівняння мінімальних системних вимог ігрових рушіїв та їх можливостей

	Unity	Unreal Engine	GameMaker	RPG Maker	Ren'Py	Godot
Дата виходу	2005	1998	1999	1992	2004	2014
Центральний процесор	Підтримує x86, x64; ARM64; Вимагає SSE2	4 ядерний процесор, 2.5 Ггц	2 ядерний процесор	2 ядерний процесор	Вимагає SSE2	Підтримує x86, x64; ARMv8; SSE2
Об' єм оперативної пам' яті	Дані відсутні	8 ГБ	2 ГБ	4 ГБ	Дані відсутні	4 ГБ

Продовження таблиці 1.1

	Unity	Unreal Engine	GameMaker	RPG Maker	Ren’Py	Godot
Графічний процесор	З підтримкою DirectX 10, OpenGL 3.2, Vulkan	З підтримкою DirectX 11	З підтримкою OpenGL 4 та DirectX 11	З підтримкою OpenGL 4.1 та DirectX 9	З підтримкою OpenGL 2 та DirectX 9	З підтримкою OpenGL 3.3 або Vulkan 1.0
Місце на диску	Дані відсутні	Дані відсутні	3 ГБ	2 ГБ	113 МБ	200 МБ (1.3 ГБ після завантаження шаблону експорту)
Операційна система	Windows 10, macOS 11, Ubuntu 22.04	Windows 10, macOS 13.5, Rocky Linux 8 / Redhat Linux 8	Windows 7, macOS, Linux	Windows 7, macOS 10.10, Debian 8	Дані відсутні	Windows 7, macOS 10.13/10.15 Linux дистрибутиви випущені після 2016 року
Дозволяє розробляти	2D та 3D ігри	2D та 3D ігри	2D та 3D ігри	Лише 2D RPG	Лише 2D візуальні новели	2D та 3D ігри

Дані, наведені в таблиці 1.1, демонструють, що Unreal Engine, є найбільш вимогливим ігровим рушієм, що зумовлено його можливостями в 3D графіці. RPG Maker та Ren’Py, мають невеликі системні вимоги, що зумовлено їх орієнтованістю на розробку виключно 2D ігор та вузько направленістю. Unity має доволі збалансовані системні вимоги, відповідно до можливостей рушія. Game Maker має менші системні вимоги, ніж Godot, але аналізуючи кількість розроблених 3D проєктів [10] та слова старшого аналітика програмного забезпечення Джордана Майнора [11], цілком можна припустити, що Game Maker, гірше підтримує 3D графіку, порівняно з Godot.

Не менш важливим показником є статистика відвідуваності офіційних сторінок рушіїв, що може демонструвати зацікавленість ними та їх популярність на даний момент. За даними Dragonfly [12], рейтинг має наступний вигляд:

– Unity;
– Unreal Engine;
– RPG Maker;
– Godot;
– Ren’Py;
– GameMaker.

Відповідно до цього рейтингу, найбільш відвідувані є сторінки саме тих рушіїв на яких розроблено найбільша кількість ігор. Однак, варто відзначити, що за параметром кількості, Godot уступає всім переліченим рушіям, але знаходиться всередині наведеного рейтингу. Це може свідчити про ріст його популярності.

Для підтвердження цієї теорії, першочергово отримано дані про кількість розроблених ігор за останні 7 років та загалом, використовуючи інструментарій SteamDB [13]. Отримані дані, зведено в таблицю 1.2.

Таблиця 1.2 – Кількість розроблених ігор рушіями по роках

	2017	2018	2019	2020	2021	2022	2023	2024	Всього
Unity	3390	4470	4085	4967	5766	6219	7344	8492	49731
Unreal Engine	718	1023	995	1283	1682	1863	2354	3130	14549
GameMaker	286	443	387	596	795	808	722	757	5186
RPGMaker	223	329	356	351	427	443	463	486	3367
Ren’Py	133	164	186	228	317	420	456	608	2734
Godot	12	34	55	106	180	317	377	720	1873

Для подальшого порівняння, розглядається процентне зростання кількості розроблених ігор, використовуючи формулу 1.1:

$$\text{Процентне зростання} = \left(\frac{\text{Поточний рік} - \text{Попередній рік}}{\text{Попередній рік}}\right) * 100\% \quad (1.1)$$

Після використання формули, отримані дані занесено в таблицю 1.3.

Таблиця 1.3 – Процентне зростання кількості розроблених ігор

Процентне зростання/Рік	Unity	Unreal Engine	GameMaker	RPGMaker	Ren'Py	Godot
2018	31.8%	42.48%	54.90%	47.53%	23.31%	183.33%
2019	-8.61%	-2.74%	-12.64%	8.21%	13.41%	61.76%
2020	21.59%	28.94%	54.01%	-1.40%	22.58%	92.73%
2021	16.09%	31.10%	33.39%	21.65%	39.04%	69.81%
2022	7.86%	10.76%	1.64%	3.75%	32.49%	76.11%
2023	18.09%	26.36%	-10.64%	4.51%	8.57%	18.93%
2024	15.63%	32.97%	4.85%	4.97%	33.33%	90.98%

Базуючись на отриманих значеннях, побудовано графік для аналізу (рис. 1.2):

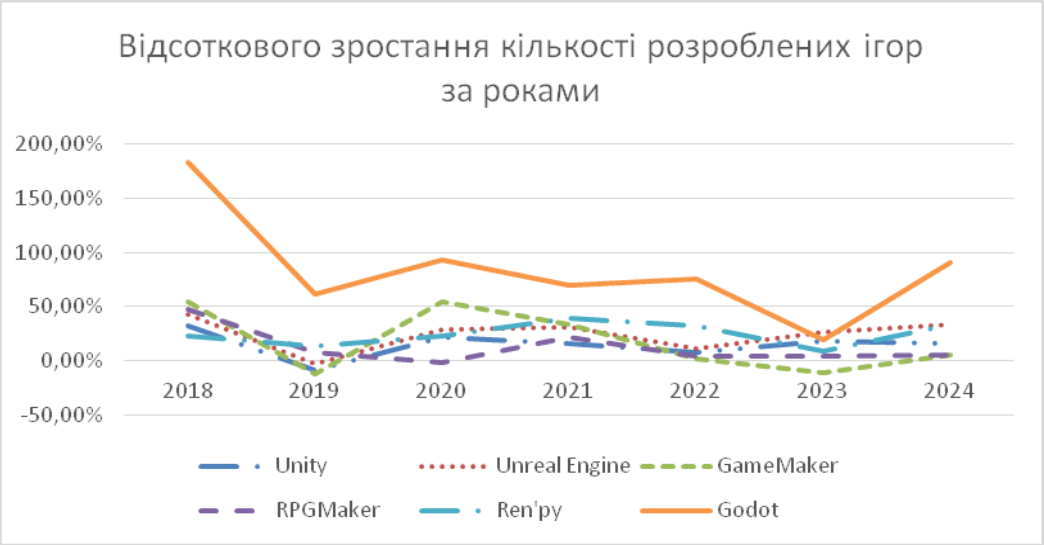


Рисунок 1.2 – Графік відсоткового зростання кількості розроблених ігор за роками

Виходячи з рис. 1.2 та табл. 1.3, можна помітити стрімке зростання кількості розроблених ігор на Godot. Звісно, високі відсотки Godot на рисунку 1.2, пов’язані з першочергово невеликою кількістю ігор, порівняння з іншими ігровими рушіями, що вже мали певну достатню кількість ігор, що виходили в певний рік й мають більш стабільні графіки приросту. Але це не заперечує швидкого приросту кількості ігор, що розроблюються на Godot. Таким чином, він вже на момент 2024 року, випередив за кількістю розроблених за рік ігор RPG Maker та Ren’Py.

Якщо цей тренд збережеться, є ймовірність, що Godot перевищить річний показник розроблених ігор Game Maker в найближчі роки та за декілька років зможе досягти загальної кількості розроблених ігор еквівалентній кількості на RPG Maker та Ren’Py.

Зрозуміло, що конкурувати з Unity чи Unreal Engine, найближчим часом Godot не в змозі, але він вже почав займати нішу серед менших ігрових рушіїв, що дозволяє прогнозувати його подальший розвиток в сегментів інді-ігор.

Ще одним ключовим фактором переваги Godot, відсутність будь-яких тарифів чи сплат за рушій. В табл. 1.4 представлено порівняння цін і ліцензування ігрових рушіїв.

Таблиця 1.4 – Вартість і ліцензування ігрових рушіїв

	Тип ліцензії	Вартість	Податок на прибутки
Unity	Умовно безкоштовний	Безкоштовно до 200.000$ річного прибутку	2.5% від прибутку понад 200.000$ (Pro ліцензія)
Unreal Engine	Умовно безкоштовний	Безкоштовно до 1.000.000$ валового прибутку	5% від прибутку понад 1.000.000$
RPGMaker	Платний	75$	Відсутній
Ren’Py	Безкоштовний	Відсутня	Відсутній
Godot	Безкоштовний	Відсутня	Відсутній

Продовження таблиці 1.4

	Тип ліцензії	Вартість	Податок на прибутки
GameMaker	Платний для комерційного використання	Безкоштовний експорт ігор, але можливість комерційного використання потребує ліцензії за 99.99$	Відсутній

Виходячи з даних, наведених в таблиці 1.4 та для прикладу враховуючи стандартний збір кошт на публікацію однієї гри в Steam у розмірі 100$ та роялті від кожної продажі, в розмірі 30%, Game Maker та PRG Maker є доволі затратними рішеннями для інді-розробників, які розроблюють свою першу гру, яка має ризики не бути фінансово успішною. Unity та Unreal Engine, є більш вигідними для початківців, оскільки не потребуватимуть першочергової оплати. Найбільш вигідними відповідно є Godot та Ren'Py.

Підсумовуючи порівняння ігрових рушіїв можна зазначити, що Godot є доволі перспективним рушієм, який вартий уваги, особливо серед, розробників початківців. Звісно, за масштабністю рушій уступає, більш розвинутим, на кшталт Unity або Unreal Engine, але відповідно має й значно менші вимоги та є повністю безкоштовним. Прямим конкурентом Godot фактично є Game Maker, проте він є платним та виходячи з результатів аналізів, гірше працює з 3D графікою. Ren'Py та RPG Maker, є гарним рішенням, лише у випадку роботи з певним жанром і обмеженням 2D графікою.

2 ПІДГОТОВКА ДО РОЗРОБКИ 2D ГРИ

2.1 Аналіз вимог та постановка задач

Метою розробки є реалізація 2D гри, яка буде в достатньому обсязі демонструвати можливості та структуру проєкту в Godot. Для розробки обрано тематику космосу та жанр платформер, в графічному стилі pixel art. Ключовими вимогами до кінцевої реалізації гри є:

– інтуїтивно простий та зрозумілий геймплей;
– передавати атмосферу обраної тематики;
– вороги не мають бути занадто складними;
– рівні мають мати предмети для збирання та секретні місця;
– налаштування гри;
– цікаві ігрові механіки;
– наявність простого сюжету та його оповідання.

Відповідно до поставлених вимог, виділяються наступні задачі до реалізації:

а) головний персонаж, союзники та ворог:
    1) анімації та переміщення,
    2) фізична взаємодія,
    3) ігрові механіки;
б) рівні:
    1) декорування та наповнення,
    2) перехід між рівнями;
в) користувацький інтерфейс:
    1) головне меню, налаштування, пауза,
    2) інтерфейс гравця;
г) інше:
    1) система магазину,

2) збереження прогресу та налаштувань,
3) звуки та музика.

Таким чином, головним героєм проєкту є космонавт. Його союзниками виступають інші космонавти з якими він може вести діалог, але вони не беруть повноцінної геймплейної участі. Основними антагоністами виступають прибульці. Ігрові рівні містять різноманітні перешкоди, зокрема пастки, а також схованки з ігровою валютою, яку гравець може знайти безпосередньо під час проходження. Ігровий процес передбачає вибір: зберігати зібрану валюту до завершення гри, тим самим отримуючи більшу фінальну оцінку або ж витрачати її в магазині для придбання бонусів, що спрощують проходження рівнів.

Система збереження прогресу дозволяє гравцеві відновити гру з останнього місця збереження. Інтерфейс та діалоги реалізовано з підтримкою багатомовної локалізації (українська та англійська мови), що сприяє розширенню цільової аудиторії. Завдяки можливостям рушія Godot, проєкт експортовано на платформи Windows та Android.

2.2 Архітектура гри

Архітектура ігор на Godot, тісно пов’язана з своєрідною інтерпретацією об’єктно-орієнтованого підходу та самою архітектурою рушія. Архітектура рушія має модульний вигляд та поділяється на чотири рівні: Core, Scene, Servers та Drivers (див. додаток А). Ключова взаємодія розробника з рушієм відбувається на рівні Scene. Для прикладу, 2D персонаж є набором 2D вузлів (Node), кожен з яких відповідає різній цілі, які утворюють сцену (Scene) персонажа (див. рис. 2.1).

Таким чином кожен ігровий елемент є окремим вузлом або їх сукупністю, яка утворює сцену. У свою чергу, сукупність сцен утворює один глобальний рівень або набір рівнів, які є частино Scene Tree, який використовується для керування ієрархією вузлів та сцен, надаючи повний контроль над ними (див.

рис. 2.2). Будь-яка ієрархія вузлів, завжди має на вершині корінний вузол, який створюється автоматично рушієм та є вікном самої гри.

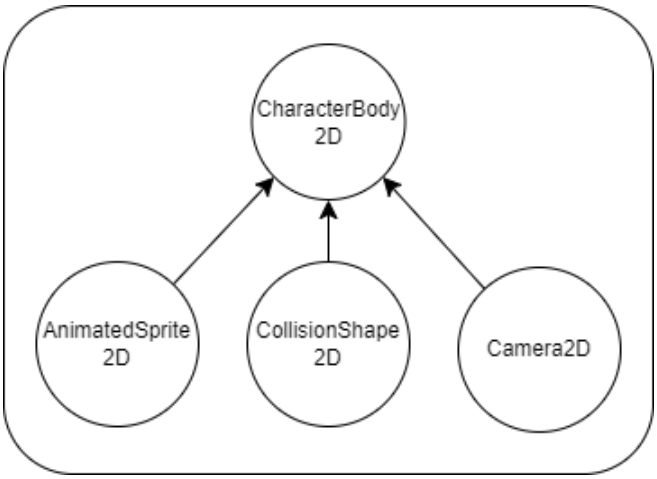


Рисунок 2.1 – Приклад Scene персонажа

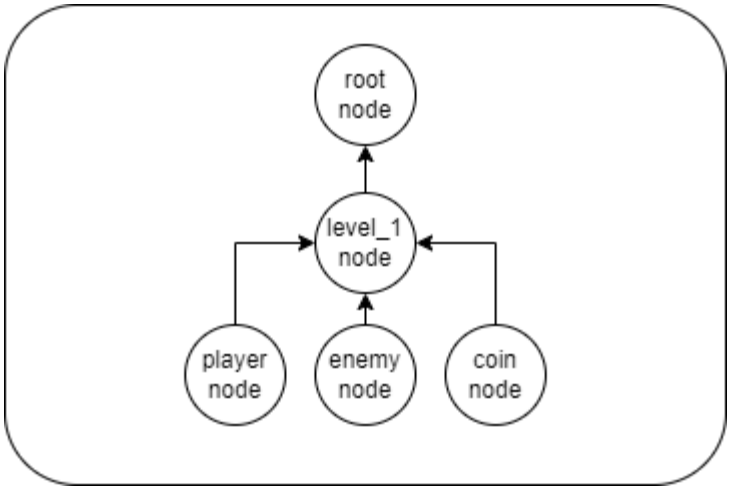


Рисунок 2.2 – Приклад ієрархії Node

Описана структуризація надає модульність та простоту повторного використання окремо розроблених елементів. Однак при використанні підходу з реалізацією окремих рівнів, а не однієї глобальної сцени, виникає проблема збереження прогресу гравця, оскільки на кожному новому рівні відбувається повторна ініціалізація сцени персонажа. Для цього в Godot, активно використовується autoload (шаблон проєктування “Одинак”). В умовах рушія,

він завжди є дочірнім вузлом, корінного вузла, тому може зберігати необхідну інформацію та надавати її іншим вузлам.

Для розробки, також впроваджено використання скінченного детермінованого автомату, який реалізує поведінку штучного інтелекту ворога в грі. Скінченний автомат – це математична модель обчислення, яка може перебувати лише в одному стані з кінцевої множини в будь-який момент часу, а переходи між станами здійснюються у відповідь на зовнішні вхідні дані. Автомат визначається множиною його станів, початковим станом, множиною допустимих вхідних символів і умовами для кожного переходу. Детермінованість скінченого автомату, полягає в переходах між його станами – кожен стан має один перехід для кожного входу.

Формально, скінчений детермінований автомат можна описати п’ятіркою:

$$DFA = (Q, \Sigma, \sigma, q_0, F) \tag{2.1}$$

де Q – кінцева множина станів;

Σ – множина допустимих вхідних символів (алфавіт);

δ: Q × Σ → Q – функція переходу;

$q_0 \in$ Q – початковий стан;

F ⊆ Q – множина кінцевих (приймаючих станів).

Для кращого розуміння концепції, доречно розглянути звичайний приклад автомату, що моделює роботу світлофора з трьома станами, зображений на рисунку 2.3.

У даному прикладі, автомат має 3 стани, перехід між якими здійснюватиметься після закінчення часу таймера, який виступає алфавітом. Початковим станом, може виступати зелений колір. Кінцевого стану, представлений автомат не має, оскільки світлофор працює циклічно.

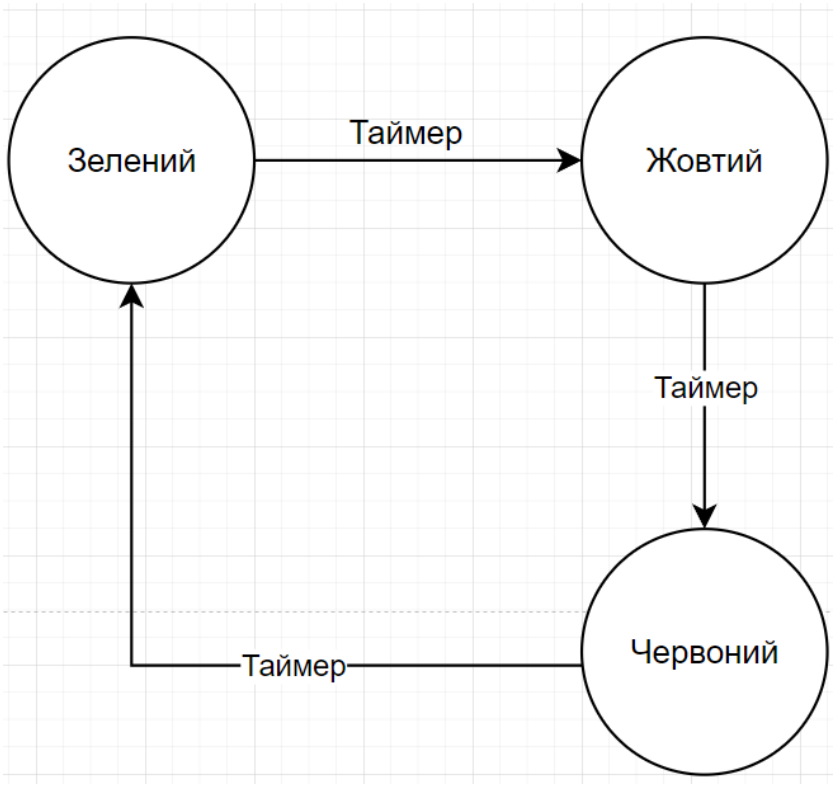


Рисунок 2.3 – Детермінований скінченний автомат світлофора

В розробленій грі ворог має 5 станів – ходьба, переслідування гравця, стрибок та простоювання. Організація цих станів та переходів між ними без використання скінченого автомату, була б значно складнішою через потребу використання великої кількості умов та вкладених перевірок. Скінченний автомат, який використано під час подальшої розробки, зображено на рисунку 2.4.

Опираючись на формулу 2.1 та зображений на рисунку 2.4 автомат ворога починає роботу зі стану WALK, а єдиним приймаючим станом є DEATH. Множина станів ворога і множина вхідних символів, наочно продемонстрована в функції переходу у додатку Б. Фактично, описані вхідні символи, є формалізаціями певних подій, що реалізовано функціями у грі.

Хоча скінченний автомат теоретично можна застосувати й для головного героя, у цій розробці вирішено не використовувати його. Оскільки керування героєм здійснюється напряму гравцем, кількість станів є обмеженою та цілком контрольованою, якщо прийти до їх реалізації. До того ж, реалізація скінченного

автомату лише ускладнила б реалізацію коду без відчутних переваг, тому така структура виявилася недоцільною.

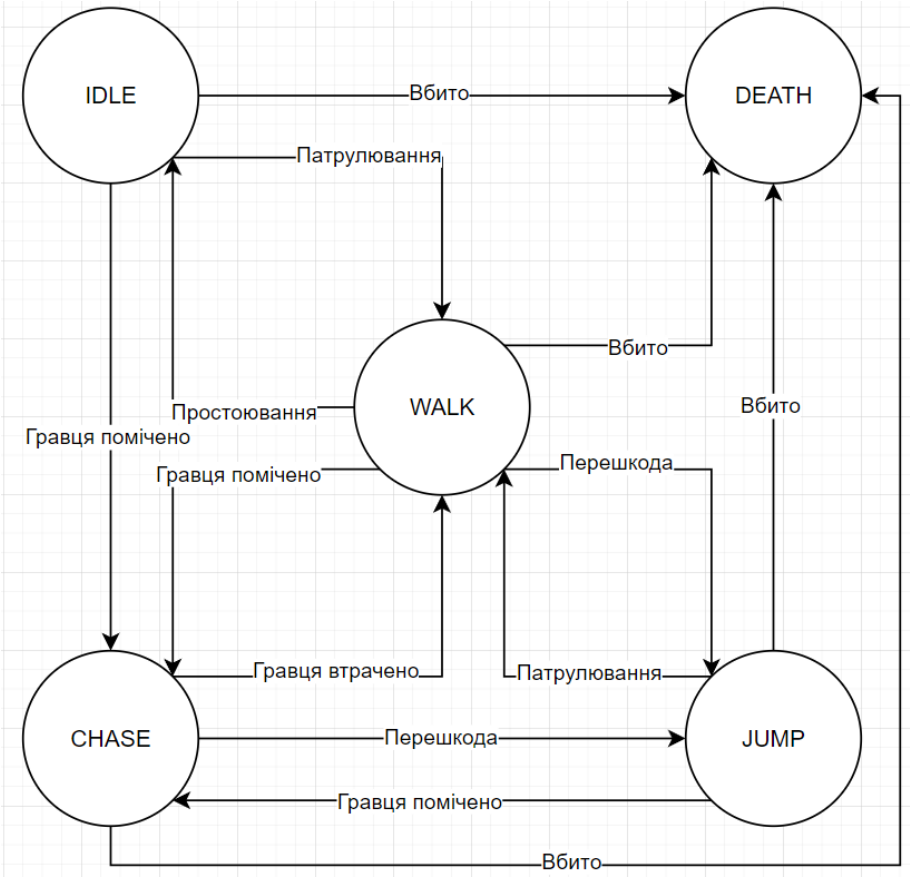


Рисунок 2.4 – Детермінований скінченний автомат ворога

3 РОЗРОБКА 2D ГРИ

3.1 Початкове налаштування проєкту

Першочергово відбувається створення проєкту з задання назви та вибором одного з трьох методів рендеру – Forward+, Mobile та Compatibility. Головна різниця між ними, полягає в підходах до обробки джерел світла [14]. Однак, обрання певного методу, не є глобальним. В таблиці 3.1.1 зображено комбінації методів, залежно від обраного базового. Оскільки гра розроблялася для платформи ПК та Android, за основу було взято Forward+, але в процесі тестування в розділі 4.1, для мобільних пристроїв з огляду на продуктивність, вирішено обрати Compatibility режим рендеру. Хоча Compatibility й може виступати глобальним рішенням для всіх платформ, але також накладає зайві обмеження, як наприклад відсутність підтримки створення частинок графічним процесором (вузол GPUParticles). Його ключовою відмінністю від інших методів рендеру, полягає в використані старішого прикладного програмного інтерфейсу OpenGL 3.3 та OpenGL ES 3.0, в той час, як решта методів, використовує сучасний Vulkan.

Таблиця 3.1 – Порівняння методів ренрдеру

Метод/Платформа	ПК	Мобільні пристрої	Веб-платформа
Forward+	Forward+	Mobile	Compatibility
Mobile	Mobile	Mobile	Compatibility
Compatibility	Compatibility	Compatibility	Compatibility

Після створення нового проєкту, рушій одразу надає змогу перейти до розробки. Проте перед початком основної роботи були виконані базові налаштування з метою спрощення подальшого процесу розробки. Насамперед

було активовано масштабування зображення відповідно до розміру екрану пристрою (смартфона чи вікна на ПК), оскільки за замовчування ця функція вимкнена. Тим самим обрано відповідний метод та аспект, масштабування.

Також відразу відбулося встановлення глобального шрифту для всього проєкту, що значно полегшило створення графічного інтерфейсу користувача. Останнім кроком початкового налаштування стала зміна типу фільтрації текстур з лінійного на найближчий, оскільки гра реалізується у стилі pixel art, а лінійна фільтрація призводить до втрати чіткості піксельної графіки.

3.2 Головний персонаж

Керований персонаж побудований на основі вузла CharacterBody2D, призначеного для створення об'єктів, що рухаються відповідно до скрипту. На відміну від вузла PhysicBody2D, він має спеціалізований метод move_and_slide(), який автоматично обробляє зіткнення з фізичними елементами, такими як підлога, стіни тощо.

Першим дочірнім вузлом персонажа є AnimatedSprite2D. Він дозволяє використовувати набір спрайтів для створення анімацій. Для цього надається спеціалізований інтерфейс для взаємодії з кожним кадром анімації, що зображений на рисунку 3.1.

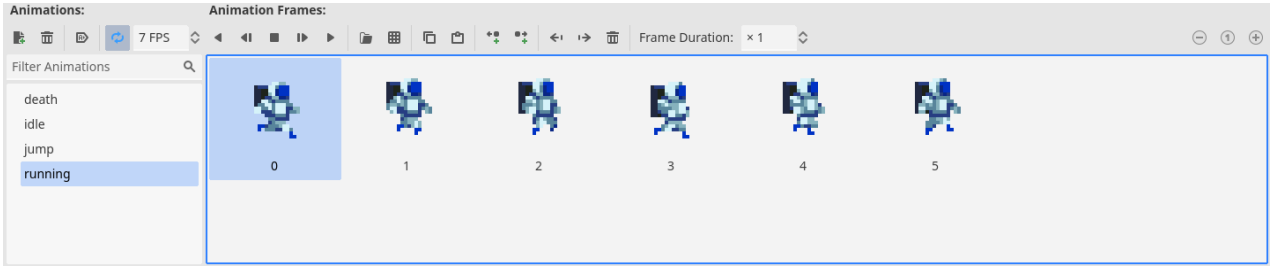


Рисунок 3.1 – Інтерфейс для анімації спрайтів

Кожна анімація має власну назву та набір спрайтів. Наступним вузлом у сцені персонажа є CollisionShape2D, який визначає зону колізії. Початково він не

має колізії, доки не задано відповідну форму. Ще одним вузлом персонажа, є PointLight2D, що реалізує ліхтарик гравця. Форма світла визначається текстурою, що використовується для його створення. Сама ж текстура, може бути створена безпосередньо в рушії наприклад використовуючи 2D градієнт з радіальним заповненням або ж бути завантаженою окремим зображенням. Однак сама тінь від джерела світла, початково відсутня та потребує бути увімкненою окремо. До того ж, тінь має 3 рівні фільтрації – відсутня, PCF5 та PCF13. Залежно від рівня, змінюється гладкість тіні та ресурсна вартість рендеру. Враховуючи стилістику гри, доречно взагалі не використовувати фільтрацію.

Останнім дочірнім вузлом гравця є Camera2D. Початково камера охоплює всю видиму зону, що також задається в налаштування проєкту. Тому, використовується наближення зображення до гравця для обмеження його кругозору. Для запобігання виходу камери за межі ігрового простору на неї встановлюються координатні обмеження, які відповідають межам ігрового рівня. Крім того, дочірніми вузлами камери є графічні елементи інтерфейсу: відображення кількості здоров'я, зібраної валюти, а також попередньо приховане меню паузи.

Далі, перед розробкою логіки персонажа, було підготовлено керування ним. Для цього використовується згаданий в розділі 2.2 autoload, який оброблює всі події вводу, під назвою InputMap. Створюючи подію та надаючи їй назву, обираються кнопки, що будуть її обробляти. Так на рисунку 3.2, наведено введення події стрибку гравця.

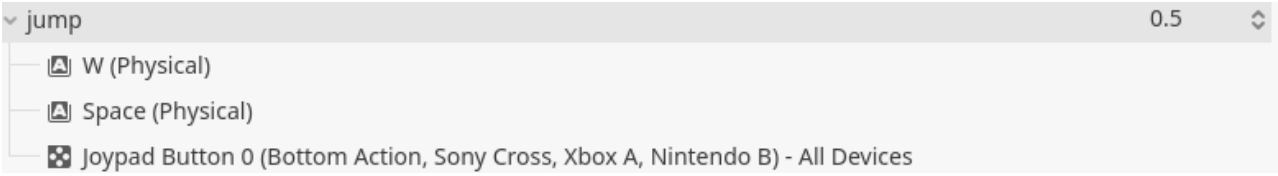


Рисунок 3.2 – Обробка подій натискання клавіш для стрибку

Для надання найкращого користувацького досвіду, реалізовано не лише керування клавіатурою, а й геймпадами.

Ще одним autoload, є PlayerManager, що відповідає за збереження даних гравця між сценами, а також має декілька методів для обробки логіки взаємодії (отримання пошкоджень, збільшення рахунку тощо). Слід відмітити, що в Godot, підчас запуску сцени, є можливість переглянути окремо локальне Scene Tree, що є початково створеною сценою в редакторі та віддалене Scene Tree, яке відображає, саме запущену сцену. Таким чином, в локальній сцені гравця, будуть знаходитися, лише вище перераховані вузли, а PlayerManager буде частиною саме віддаленого Scene Tree. До того ж, autoload, створенні користувачем, задаються вручну в налаштуваннях як глобальні. Вони відображаються в Scene Tree, на відміну від системних, які не можна побачити.

У скрипті сцени персонажа реалізовано переміщення персонажа та використання здібностей (зміни гравітації і активація ліхтарика) за допомогою попередньо налаштованого InputMap. Логіка переміщення знаходиться в функції _physics_process(), а використання здібностей – у функції _process(). Різниця між цими функціями полягає в тому, що _physics_process() викликається фіксовану кількість разів на секунду, що робить її ідеальною для фізичних розрахунків. Водночас, _process() викликається кожен кадр, залежить від кількості кадрів на секунду, тому вона менше підходить для обробки фізичних процесів.

Може здатися, що таке розділення на окрему сцену персонажа та PlayerManager ускладнює логіку, однак насправді цей підхід спрощує її. У сцені персонажа відбувається виключно обробка фізичної взаємодії та анімацій, тоді як PlayerManager відповідає за обробку саме ігрової логіки.

Взаємодія з дочірніми вузлами персонажа, як наприклад для зміни напрямку спрайту, може бути реалізована декількома способами. Перший, найбільш громіздкий – це пряме посилання на шлях до вузла. Такий підхід підходить для статичних сцен, де після створення не передбачається значних змін, як наприклад у меню налаштувань. Однак, такий підхід має недолік: може виникнути ситуація, коли певний дочірній вузол ще не ініціалізовано, що призведе до помилки. Другий підхід полягає в отриманні вузла за назвою після його ініціалізації, за допомогою методу get_node() та привласненні його певній

змінній. Цей підхід є кращим, але при великій кількості вузлів, що зберігаються в змінних, взаємодія з функцією _ready(), яка виконується під час створення вузла, може ускладнитися. Останній, найбільш зручний метод, що й був використаний – це анотація @onready, яка дозволяє відкладено ініціалізувати змінну до моменту, коли вузол увійшов у Scene Tree.

За всі звукові ефекти, персонажа включно, відповідає autoload SFXController, який є AudioStreamPlayer вузлом. Однак, напряму він працює саме з звуками персонажа, оскільки камера гравця завжди знаходиться на ньому. У випадку з ворогами, SFXController передає звук на AudioStreamPlayer2D вузол, який дозволяє реалізувати звук, з врахуванням відстані до гравця.

3.3 Союзники

Союзники має побідну вузлову будову до гравця (за виключенням камери та ліхтарика), а також включає зону взаємодії з ним. Зона взаємодію реалізована за допомогою вузла Area2D, який реагує на об’єкти колізії, що входять в зону, яка задається дочірнім вузлом CollisionShape2D. Після потрапляння до зони взаємодії, гравцеві надається можливість ініціювати діалог із союзником. Про доступність діалогу свідчить поява відповідного графічного індикатора. Для реалізації діалогів, використано плагін розроблений спільнотою під назвою Dialogic 2. Він включає в себе можливість створювати повноцінні візуальні новели та за потреби, систему діалогів для ігор інших жанрів.

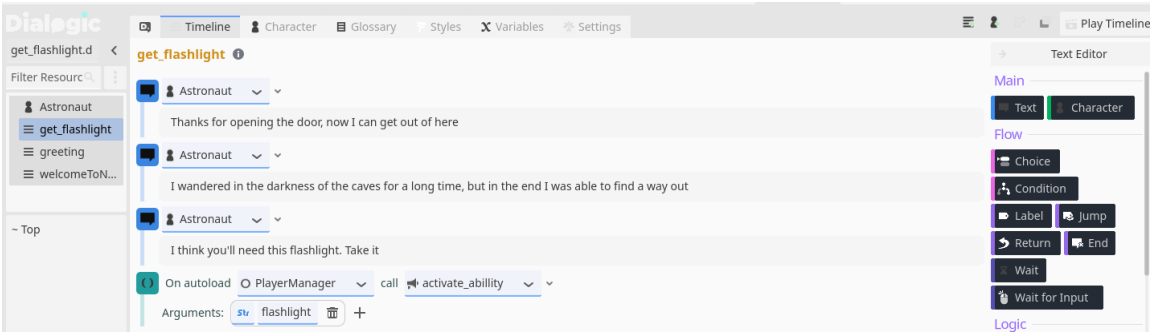


Рисунок 3.3 – Інтерфейс редагування діалогу в Dialogic 2

До того ж, він дозволяє напряму взаємодіяти з кодом: використовувати власні чи вбудовані змінні або викликати функції написані власноруч. Таким чином деякі союзники гравця після діалогу з ними відкривають певні можливості у вигляді використання ліхтарика або зміни гравітації.

3.4 Ворог

Ворог аналогічно має подібну вузлову будову до союзника, але має 3 зони (вузол Area2D) взаємодії, кожна з яких відповідає за обробку своєї логіки, а саме:

– зона виявлення гравця,
– зона отримання пошкоджень,
– зона нанесення пошкоджень.

Зона отримання пошкоджень працює за методом класичних платформерів, на кшталт Super Mario, коли потрібно стрибнути на ворога, щоб подолати його. Якщо гравець потрапляє в зону виявлення, ворог почне переслідування, наближаючись до гравця. У випадку, якщо ворог наблизиться вдосталь близько до гравця, спрацює зона нанесення пошкоджень, перебуваючи в якій гравець отримуватиме пошкодження з певною періодичністю.

Також сцена ворога має в собі дочірні вузли RayCast2D та AudioStreamPlayer2D. Останній, як зазначалося у розділі 3.1, використовується передання звуку з урахуванням відстані до гравця. RayCast2D застосовано для фіксації перешкод (об’єктів з колізією, що не є гравцем) і у разі їх виявлення виконується стрибок для подолання.

Спочатку, доки гравець не увійшов в жодну з зон, ворог виконує патрулювання певної ділянки, яка реалізована двома точками, між якими здійснюється переміщення. Досягнувши однієї з них, ворог затримується на кілька секунд, після чого продовжує патрулювання.

3.5 Предмети, що підбираються гравцем

Реалізовано 2 предмета, що можуть бути підібрані гравцем – діаманти, що виступають одночасно ігровою валютою і показником рахунку та батареї, що надають додаткові життя.

Їх сцени є доволі схожими за структурою, складаючись з спрайта та зони підняття предмету. Батареї, додатково мають вузол PontLight2D, що робить їх більш помітними в темряві.

Також, обидві сцени включають в себе взаємодію з PlayerManager, для збільшення відповідних показників.

3.6 Об’єкти з якими взаємодіє гравець

В грі реалізовано 3 об’єкта з якими гравець може взаємодіяти на пряму натисканням відповідної клавіши. Першим, є поєднання дверей та терміналу. Коли гравець входить в зону взаємодії з терміналом, він отримує можливість активувати його для відкриття зачиненого проходу. Коли відбувається відкриття дверей, здійснюється відповідна анімація і відключається колізія дверей. Однак, варто відмітити, що двері та термінал є двома окремими спрайтами, які вже мають певне розміщення в сцені свого поєднання. Відповідно коли ця сцена додається на рівень, виникає потреба розмістити їх в інших місцях та новими положеннями. Для цього Godot дозволяє редагувати вузли однієї сцени безпосередньо в іншій, незалежно від початкової.

Наступним об’єктом є ліфт. Порівняно з усіма попередніми сценами, він використовує так званий користувацький сигнал. До цього моменту, використовувалися лише вбудовані сигнали, які не оголошуються фактично в коді, а лише їх обробка (як наприклад потрапляння гравця в зону переслідування ворогом). Коли гравець активує ліфт, перебуваючи у відповідній зоні,

спрацьовує сигнал про активацію ліфта. Таким чином, на одному з рівнів використано вузол AnimationPlayer для анімації переміщення ліфта (рисунок 3.4).



Рисунок 3.4 – Інтерфейс AnimationPlayer

На самому рівні прописаний прослуховувач сигналу, тож у разі його спрацювання запускається циклічна анімація переміщення.

Останнім об’єктом взаємодії є магазин, який дозволяє гравцеві витрачати накопичену валюту для придбання додаткового життя чи поповнення шкали здоров’я. Однак основна логіка взаємодії з PlayerManager розподілена між частинами інтерфейсу магазину, які більш детально розглянуто у розділі 3.8.2.

Непряма взаємодія гравця відбувається з певними зона та пастками, які фактично також є зонами, але з анімованими спрайтами. В грі впроваджено чотири типи зони взаємодії, серед яких:

– зона зміни рівня,
– зона смерті,
– секретна зона,
– зона перемоги.

Потрапляючи у зону зміни рівня, відбувається перенесення гравця на наступний рівень та збереження його прогресу, а сам перехід супроводжується анімацією затухання екрану. Зона смерті розташована на окремих рівня. При її активації гравець втрачає частину здоров’я та повертається на початок рівня. Секретна зона прихована у стінах та стелях без колізії. Потрапивши до неї, частково розкривається секретне місце, яке стає прозорішим для візуального виявлення. Зона перемоги зустрічається лише наприкінці гри. Вона зберігає фінальний рахунок гравця в окремий файл і очищує дані в основному файлі з показниками гравця.

Пастки, аналогічно до ворога, описаного у розділі 3.4., завдають пошкоджень гравцеві у разі потрапляння до відповідної зони. Проте шкода завдається лише в певний момент анімації активності пастки.

Оскільки більшість пасток розміщено на затемнених рівнях, під час завдання шкоди вони випромінюють світло за допомогою вузла PointLight2D.

3.7 Створення рівнів

Загалом в грі реалізовано 6 рівнів. Основним вузлом для реалізації рівнів є TileMapLayer. Хоча в рушії можна поміти інший вузол, під назвою TileMap, але на ньому присутня мітка, що він застарілий та рекомендація використовувати, саме TileMapLayer.

Першочергово для використання TileMapLayer, створено TileSet, який може містити необхідну кількість зображень, які фактично і складають TileSet, який за потреби може бути збережено для повторного використання.

Колізію на рівні можна задати власноруч, додаючи CollisionShape2D, проте рушій надає можливість додавати колізію на вже розбиті блоки TileSet. Приклад цього наведено на рисунку 3.5.

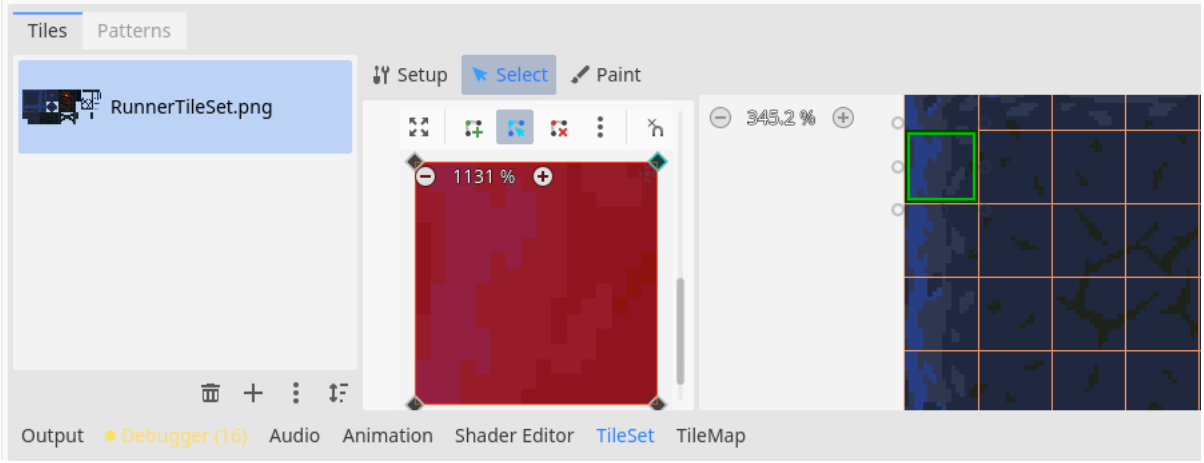


Рисунок 3.5 – Додавання колізії на блок TileSet

У кількох рівнях гри реалізовано візуальний ефект космічного сяйва за допомогою 2D шейдерів, застосованих до вузла ColorRect. Базовий код ефекту було запозичено з відкритого ресурсу, що спеціалізується на шейдерах для рушія Godot, однак через відмінність у версіях рушія, код потребував адаптації до поточної версії [15]. Шейдери в Godot реалізуються на основі мови програмування шейдерів GLSL, однак у спрощеному варіанті, що абстрагує низькорівневу ініціалізацію та спрощує створення складних візуальних ефектів.

Також реалізовано візуалізацію вітру за допомогою вузла GPUParticles2D. Частинки імітують рух повітря завдяки спрямованому емісійному потоку з налаштованою швидкістю, розміром та прозорістю.

На додачу підземні рівні мають затемнення, що змушує гравця використовувати ліхтар. Ефект затемнення реалізовано за допомогою вузла CanvasModulate.

Самі рівні є базовими вузлами Node2D, що містять у собі всі підготовлені сцени: персонажа, ворогів тощо.

3.8 Графічний інтерфейс

У більшості попередніх розділів розглядалася побудова сцен за допомогою вузлів, що успадковувалися від вузла Node2D, який використовується для 2D елементів гри. Елементи графічного інтерфейсу в Godot успадковуються від іншого батьківського вузла – Control. В його основі лежить прозорий прямокутник, який за заданою прив’язкою заповнює екранний простір користувача. Таким чином, доцільно використовувати вузол Control як основу будь-якого елементу інтерфейсу. Він має мати базову прив’язку до всього екрану гравця, а всі інші елементи варто розмішувати як його дочірні – відповідно до бажаних правил розміщення. Такий підхід забезпечує централізоване керування всім інтерфейсом, що знаходиться всередині цього вузла та спрощує масштабування під різні роздільні здатності екранів.

Також Control вузол дозволяє застосовувати користувацькі теми до елементів інтерфейсу. Кожну тему можна зберегти для повторного використання в інших сценах. Рушій надає гнучке налаштування кожного окремого елементу теми, який може бути, як видозміненим базовим елементом Godot, змінюючи його шрифт та кольори при різних видах взаємодії, так і використовуючи зовнішній розроблений набір текстур.

Під час реалізації інтерфейсу, використано панель для налаштування субрегіону текстури. Вона автоматично виявляє окремі елементи на зображенні, надаючи змогу ручному корегуванню виділення та дозволяє визначити області масштабування: динамічну (всередині умовних ліній, яка розтягується) та фіксована (для країв). Такий підхід до масштабування відомий, як 9-slice scaling [16]. Приклад його використання наведено на рисунку 3.6.

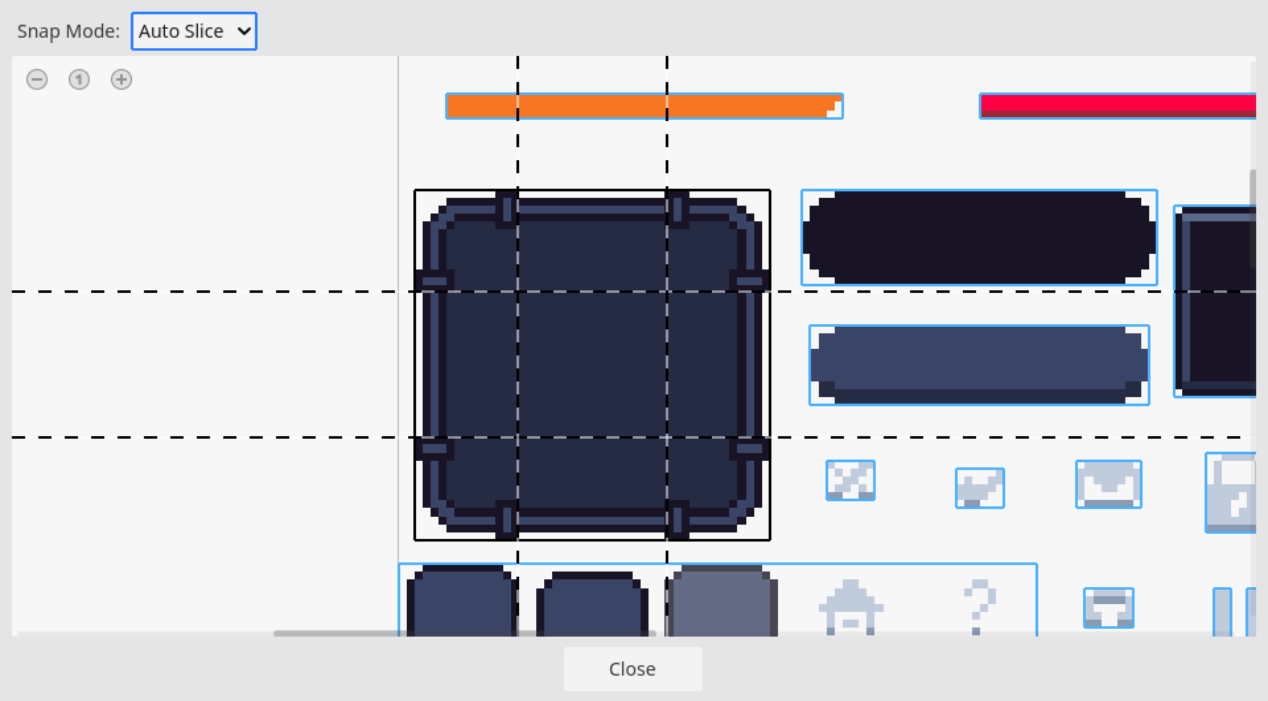


Рисунок 3.6 – Інтерфейс інструменту Auto Slice для налаштування текстур інтерфейсу

Відповідно до задач поставлених в пункті 2.1, реалізовано переклад діалогів в Dialogic 2 та решти інтерфейсу. Першочергово файл перекладу зберігається у CSV форматі, в якому записується переклад з певним ключем,

який буде використовуватися в коді для перекладу, відповідно до локалізації, що увімкнена. Рушій автоматично, розділить CSV файл на декілька локалізацій, якщо він вірно заповнений. Dialogic 2 має схожий підхід до перекладу, однак кожен діалог потребує окремого файлу.

3.8.1 Меню та налаштування

Головне меню складається з декількох елементів – панелі головного меню, налаштувань, запису максимального рахунку гравця та заднього фону, що реалізовано за допомогою вузла ParallaxBackground, тим самим створюючи ефект руху. На момент версії рушія 4.3, загальноприйнятою реалізацією ParallaxBackgrond є приклад використання наведений на рисунку 3.7. Однак, в рушії зазначається наявність експериментального вузла Parallax2D, який має в майбутньому замінити ParallaxBackground та ParallaxLayer для спрощення реалізації.

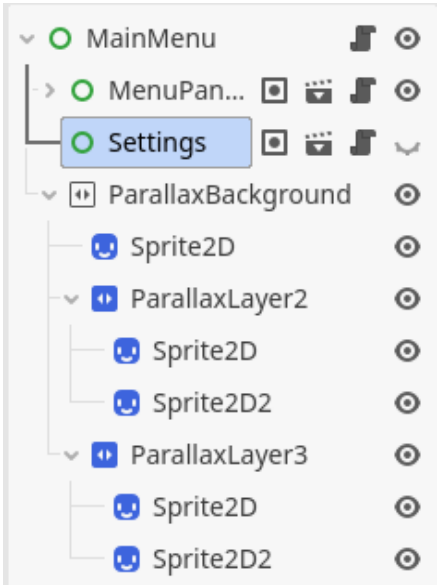


Рисунок 3.7– Сцена головного меню з ParallaxBackground вузлом

Структурно, панель головного меню та меню налаштувань побудовані на основі схожих вузлів інтерфейсу, зокрема Button, Label, VBoxContainer та додаткових елементів, специфічних для параметрів конфігурації. Панель

головного меню забезпечує базову навігацію: запуск нової гри чи продовження, перехід до налаштувань та завершення сеансу. У меню налаштувань реалізовано фунціонал перемикання повноекранного режиму, зміни мови, вибору режиму вертикальної синхронізації, а також аудіо налаштування для трьох окремих каналів – музики, звукових ефектів та загального рівня гучності. Варто зазначити, що рушій підтримує вбудовану обробку звуку, зокрема еквалізацію, реверберацію та інші аудіо ефекти. Проте в межах даного проєкту, потреби у використанні цього інструментарію непередбаченою. Також на цьому етапі створено локальну групу (сцени) графічного інтерфейсу, що є своєрідним тегом в рушії. Звертаючись за цим тегом, реалізовано функцію для оновлення всього інтерфейсу, який відноситься до головного меню, що є обов'язковим при зміні локалізації. На додачу до цього, впроваджено глобальну групу інтерфейсу на Android пристроях, оскільки кількість доступних налаштувань в меню різнитиметься, залежно від платформи. Окрім цього, в пункті 3.8.2 описано інтерфейс керування на мобільних пристроях, який також належить до зазначеної групи.

Також підчас потрапляння гравця в головне, починає грати музика, яка супроводжуватиме його й на рівнях. Керування музикою здійснюється за допомогою autoload MusicController.

3.8.2 Ігровий інтерфейс

В процесі гри, користувач має змогу спостерігати за своїми показниками здоров'я та кількістю балів. Хоча Godot надає зручні вузли ProgressBar та TextureProgressBar, у грі показник здоров'я реалізовано не у вигляді індикатора прогресу, а окремими графічними сегментами, що змінюють текстуру залежно від кількості здоров'я гравця. Такий підхід також є поширеною практикою, однак потребує складнішої реалізації в рушії, порівно з використанням стандартних індикаторів. Показники рахунку та життів мають числове відображення.

Якщо ж гравець гратиме на Android пристрої, він матиме адаптоване до цього керування, зображене на рисунку 3.8.

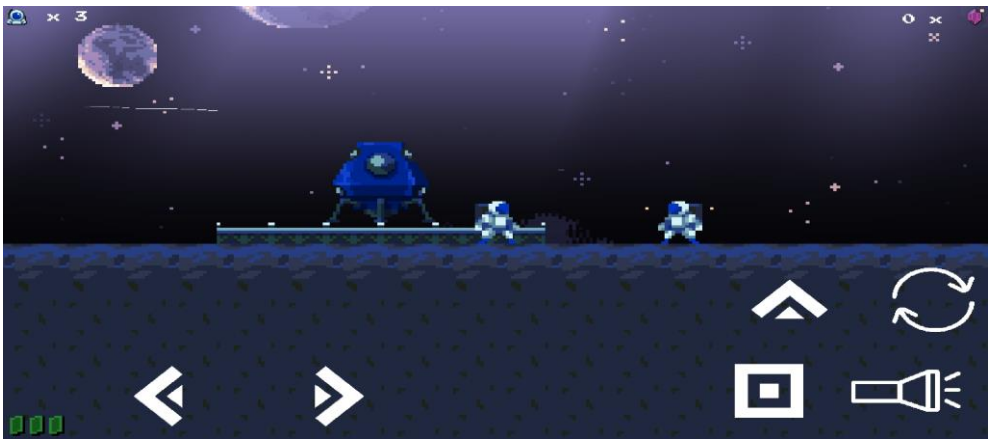


Рисунок 3.8 – Керування на Android пристрої

Клавіши керування на смартфоні є TouchScreenButton вузлами, що на відміну від звичайних вузлів кнопок, підтримують натискання декількома пальцями одночасно. Однак, сама дія не задається через InputMap, а лише записується назва дії у спеціальне поле, яке зазначено в InputMap.

Також гравець під час гри, має змогу викликати меню паузи для переміщення в головне меню або продовження гри.

Магазин реалізовано двома окремими вузлами. Основний вузол виконує роль контейнера, який відповідає за відображення всіх доступних товарів у магазині. Кожен товар представлений окремим дочірнім вузлом, що містить інформацію про товар та можливість його придбання. Узагальнене представлення магазину представлене на рисунку 3.9.

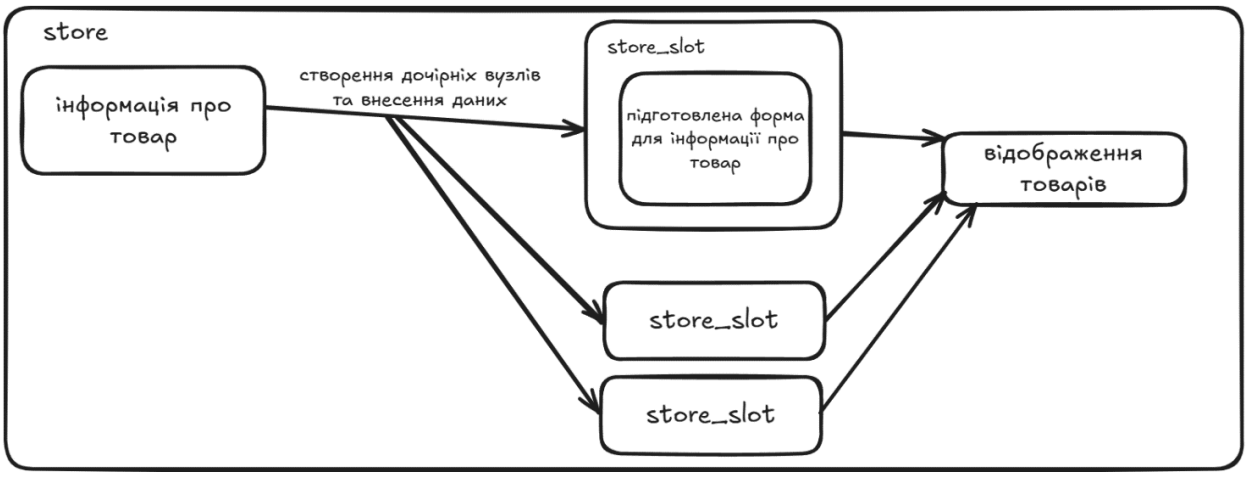


Рисунок 3.9 – Представлення наповнення магазину

3.9 Збереження даних

За збереження ігрової інформації, напряму відповідає SaveManager та частково NavigationManager. Загалом, Godot пропонує 3 основні методи збереження інформації: конфігураційний файл, JSON та бінарна серіалізація. Хоча JSON й виступає легшим для читання форматом, бінарна серіалізація даних дозволяє зберігати менші за обсягом файли, обробляти більше поширених типів даних та відповідно, у випадку створення користувацького класу, JSON вимагатиме написання власної логіки для кодування та декодування класу. Зберігання даних гравця, відбувається під час переходу між рівнями, оскільки при такому підході не доведеться зберігати додаткову інформацію, про вже переможених ворогів на рівні та зібрані ресурси. Під час завантаження даних, гравця відправить на рівень на якому він був в останнє. Кодова реалізація цього функціоналу наведена на рисунку 3.10.

```
func save():
    var file = FileAccess.open(SAVE_FILE_PATH,FileAccess.WRITE)

    file.store_var(PlayerManager.total_coins)
    file.store_var(PlayerManager.total_hp_points)
    file.store_var(PlayerManager.total_life_count)
    file.store_var(PlayerManager.player_abilities)
    file.store_var(NavigationManager.saved_level)

    file.close

func load():
    var file = FileAccess.open(SAVE_FILE_PATH,FileAccess.READ)

    PlayerManager.total_coins = file.get_var()
    PlayerManager.total_hp_points = file.get_var()
    PlayerManager.total_life_count = file.get_var()
    PlayerManager.player_abilities = file.get_var()
    NavigationManager.saved_level = file.get_var()

    file.close()

    NavigationManager.navigate_to(NavigationManager.saved_level)
```

Рисунок 3.10 – Збереження та завантаження даних гравця

Додатково, відбувається збереження налаштувань гри, що зазначив користувач у відповідному меню, після натискання клавіши збереження. Для збереження налаштувань, використано конфігураційний файл.

Згідно з офіційною документацією рушія, файли збережень зберігаються у спеціальній директорії “user://”.

3.10 Експорт на Windows та Android

Godot дозволяє з легкістю здійснювати експорт проєкту на Windows, що потребує виконання кількох підготовчих кроків. Насамперед, завантажуються шалони експорту через рушій або вручну, з GitHub репозиторію. Далі виконується фінальне налаштування проєкту, зокрема: вказується назва гри, головна сцена запуску і встановлення іконки.

Ще одним елементом підготовки до експорту, є додавання шляху до інструменту цифрового підпису, використання якого, орієнтовано на поліпшення сумісності з політикою безпеки Windows та є необхідним при поширені гри публічно на цифрових платформах. Втім, у процесі тестування, описаного у розділі 4.1, цифровий підпис не використовувався й жодних проблем із запуском гри, не було помічено.

У меню експорту, додано шаблон експорту для Windows, де було вимкнено виведення debug-консолі. Гра була експортована у форматі “.exe”, з можливістю вбудувати всі пакетні файли у виковуваний файл. Варто зазначити, що у разі вбудовування “.pkc” у “.exe”, цифровий підпис стає неможливим.

Для експорту на Android, попередньо встановлено OpenJDK та Android SDK, після чого шляхи до них були вказані в налаштуваннях Godot. Також було створено цифровий ключ, який є обов’язковим – без нього Android пристрій, не дозволяє встановлення застосунку.

На відміну від Windows, налаштування Android експорту, є більш широкими і потребують обрання архітектури процесору, зазначити категорію застосунку, адаптацію під різні розміри екранів та налаштування запитів на дозвіл використання певного функціоналу пристрою. Однак, слід відмітити, що збереження даних гравця, не потребує додаткових дозволів, від системи.

Експорт виконано в форматі “.apk”, придатному для встановлення на пристрій вручну. У разі публікації в Google Play Store, слід використовувати “.aab”, відповідно до вимоги платформи.

4 ТЕСТУВАННЯ 2D ГРИ

4.1 Продуктивність на різних пристроях

Для тестування, використано декілька різних пристроїв з різними технічними характеристиками. Основними показниками для замірювання будуть такі показники, як кількість кадрі в секунду, затримку кадру, використання оперативної та відео пам’яті і кількість об’єктів у сцені. Фіксація замірювань відбувалася з інтервалом в 5 секунд. Технічні характеристики та назви пристроїв наведено в таблиці 4.1.

Таблиця 4.1 – Пристрої на яких проводилося тестування

Характеристика/назва	ПК1	ПК2	ПК3	ПК4	Realme 6 Pro
Процесор	AMD FX-4300	Intel i5-3330	AMD Ryzen 5 1600	Intel i3-2370M	Qualcomm Snapdragon 720G
Графічна карта	AMD Radeon RX580 8 Гб	GeForce GTX 1650 4 Гб	GeForce GTX 1060 3 Гб	GeForce 610M 1 Гб	Adreno 618
Оперативна пам’ять (ГБ)	12	8	16	4	8
Операційна система	Windows 10	Windows 11	Windows 10	Windows 7	Android 11
Дисплей	1920x1080, 60 Hz	1920x1080, 60 Hz	1920x1080, 60 Hz	1366x768, 60Hz	1080x2400, 60 Hz

Для цілей тестування, було створено окремий autoload для збору інформації. Повна інформація замірювань наведена у додатках В-И. Всі замірювання проводилися з використанням вертикальної синхронізації для явного відображення проблем з оптимізацією, оскільки без її використання,

кількість кадрів завжди перевищує, можливість відображення екрану, а відповідно не вплине на ігровий досвід. Для наочного перегляду показників, їх середні показники занесено до таблиці 4.2.

Таблиця 4.2 – Середньостатистичні показники замірювань

Середній показник/пристрій	ПК1	ПК2	ПК3	ПК4
Кількість кадрів в секунду	59.94	59.94	59.94	59.93
Час кадру	33.47528	23.77500	17.91944	18.19427
Використання відео пам’яті (МБ)	65.42	58.27	58.95	60.90
Використання оперативної пам’яті (МБ)	94.18	94.72	93.03	94.70
Кількість об’єктів в сцені	165	161	160	161

Підчас тестування на ПК, було явлено певні просідання кількості кадрів в момент запуску першого діалогу, з використанням Dialogic 2. Хоча для оптимізації цього процесу й було використано попереднє завантаження всіх діалогів в головному меню, повністю усунути проблему не вдалося, однак втрата кадрів є не суттєвою й складає 1-2 кадри в секунду. Обсяг використання оперативної пам’яті, також знаходиться в допустимих межах, що свідчить про відсутність витоку пам’яті чи інших проблем. Обсяг відео пам’яті, також напряму залежав від поточної сцени і залишався в межах припустимих значень.

Тестування продуктивності на Android пристрої демонструє суттєве зниження кількості кадрів, порівняно з версією для ПК. Аналіз показав, що рушій Godot, демонструє недостатню оптимізацію при використанні джерел світла у Mobile режимі рендерингу. Ймовірно причино цього є використання прикладного програмного інтерфейсу Vulkan, який є більш вибагливим до пристрою. Очевидним вирішенням цієї проблеми, є використання Compatibility

режиму рендеру. Перехід до цього режиму, дозволив помітно підвищити стабільність і частоту кадрів навіть при збільшені кількості об’єктів на сцені, без істотної втрати візуальної якості. Очевидним недоліком такого підходу є обмеження функціональності, а саме унеможливлення використання вузла GPUParticles2D, який відповідав за візуалізацію ефектів частинок вітру.

Відповідне порівняння продуктивності між режимами Mobile та Compatibility занесено у таблицю 4.3.

Таблиця 4.3 – Середньостатистичні показники замірювань на Realme 6 Pro в різних режимах рендеру

Середній показник/режим	Compatibility	Mobile
Кількість кадрів в секунду	59.95	58.50
Час кадру	17.66125	22.47911
Використання відео пам’яті (МБ)	56.47	57.03
Використання оперативної пам’яті (МБ)	75.08	80.34
Кількість об’єктів в сцені	168	151

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи, було досягнуто поставлену мету – розроблено 2D гру з використанням рушія Godot, що дозволило на практичному рівні продемонструвати його технічні можливості та підтвердити доцільність використання в інді-розробці.

У процесі дослідження було проаналізовано актуальність рушія Godot у сучасній ігровій індустрії, зокрема в інді-сегменті, який продовжує активно розвиватися. Здійснено порівняльний огляд Godot з іншими, популярними ігровими рушіями, під час визначено ключові переваги рушія – відкритий код, повна безкоштовність та активне зростання популярності.

Підчас виконання проєкту, було сформульовано та виконано низку цілей, серед яких, створення ігрових рівнів, реалізація головного героя, ворогів, союзників з системою діалогів, впровадження користувацького інтерфейсу та його локалізації, а також експорт гри для платформи Windows та Android. Додаткову увагу, було приділено вузловій архітектурі рушія, що є тісно пов’язаною з об’єктно-орієнтованим підходом. Для керування поведінкою ворогів застосовано скінченний автомат, що забезпечив передбачувану логіку взаємодії.

У ході тестування проєкту встановлено, що на ПК гра працює стабільно, за винятком незначних втрат продуктивності, пов’язаних з використанням плагіна Dialogic 2. На мобільному пристрої, у режиму рендеру Mobile, спостерігалися суттєві проблеми з оптимізацію, поруч з джерелами світла, що супроводжувалися зменшенням частоти кадрів. Перехід у режим Compatibility, дозволи досягти кращої продуктивності, але з певними обмеженнями, пов’язаними з структурою проєкту.

Реалізований проєкт, підтвердив ефективність рушія для створення 2D ігор та продемонстрував його придатність, як інструменту для інді-розробників, студентських проєктів та навчальних цілей, попри помічені проблеми в Mobile режимі рендеру.

ПЕРЕЛІК ПОСИЛАНЬ

1. Steam Charts by Tag: Idie. SteamDB. URL : https://steamdb.info/charts/?tagid=492 (accessed 26.12.2024)

2. Top Indie Games. Steam250. URL : https://steam250.com/tag/indie (accessed 26.12.2024)

3. Showcase. Godot Engine. URL : https://godotengine.org/showcase/ (accessed 26.12.2024)

4. Godot (game engine) – Notable Games made with Godot. Wikipedia. URL : https://shorturl.at/KRVI8 (accessed 26.12.2024)

5. José Luis Sanz. Pequeños que aspiran ser grandes: todo sobre los juegos. ElOutput. URL : https://eloutput.com/videojuegos/reportajes/juegos-indie/ (fecha de consulta 08.01.2025)

6. Інді-ігри. Вікіпедія. URL : https://shorturl.at/EPnCg (дата звернення 08.01.2025)

7. VGI Global Indie Games Market Report 2024. VG Insights. URL : https://vginsights.com/assets/reports/VGI_Global_Indie_Games_Market_Report_2024.pdf (accessed 08.01.2025)

8. Robin Keijser. Popularity of Indie Games: Bachelor's thesis. Akademin för teknik och miljö, Högskolan i Gävle, 2012. 28 p. URL : https://www.diva-portal.org/smash/get/diva2:729367/FULLTEXT01.pdf (accessed 08.01.2025)

9. Engine Technology Overview. SteamDB. URL : https://steamdb.info/tech/ (accessed 10.01.2025)

10. GameMaker Engine Games. SteamDB. URL : https://steamdb.info/tech/Engine/GameMaker/?tagid=4191(accessed 01.03.2025)

11. Jordan Minor. GameMaker Review. PCMag. URL : https://www.pcmag.com/reviews/gamemaker (accessed 05.03.2025)

12. Top Game Engine for Indie Developers: DragonflyDB. URL : https://www.dragonflydb.io/game-dev/engines (accessed 08.03.2025)

13. Godot Releases on Steam. SteamDB. URL : https://steamdb.info/stats/releases/?tech=Engine.Godot (accessed 11.03.2025)

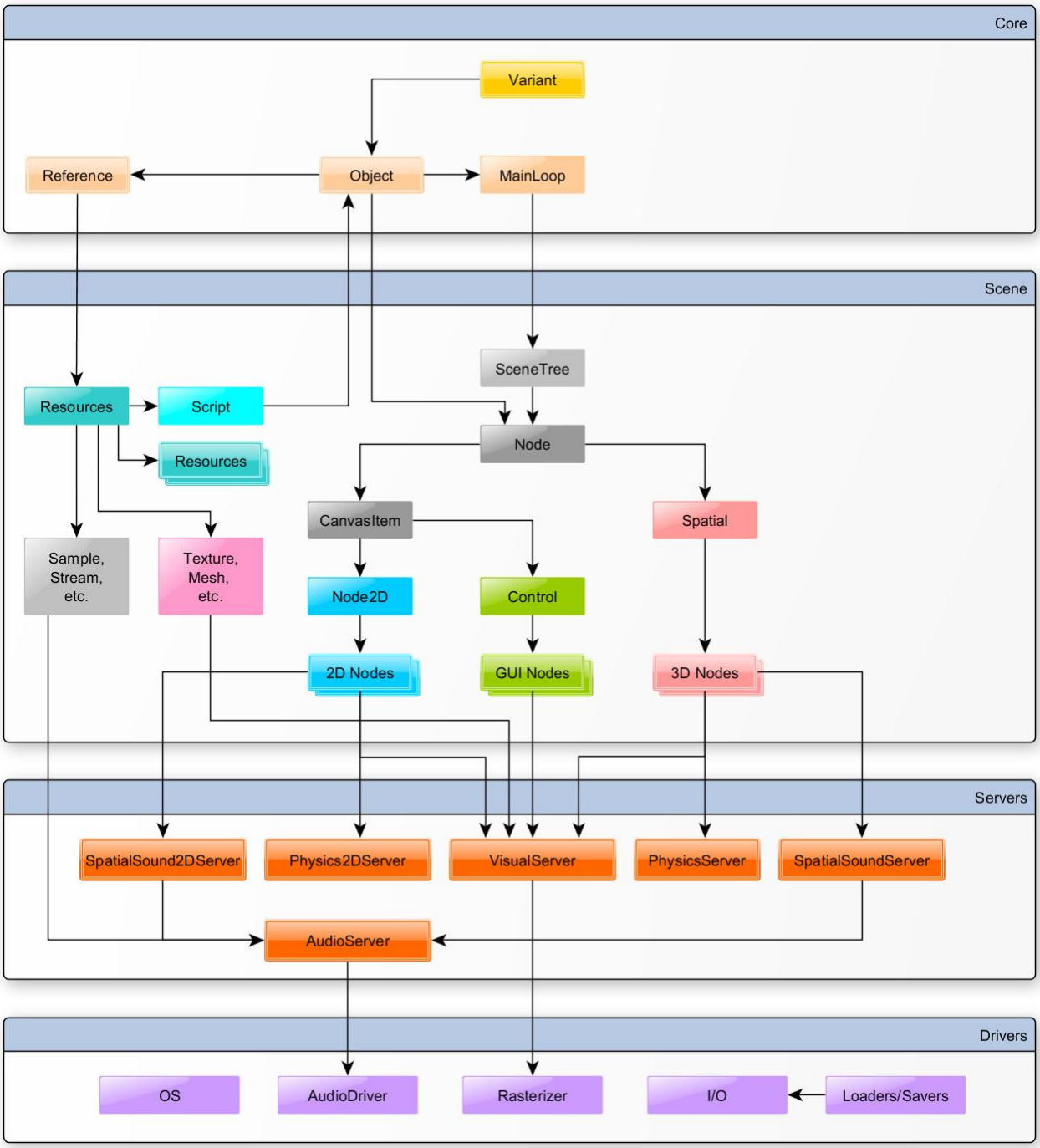
14. Internal Rendering Architecture. Godot Docs. URL : https://docs.godotengine.org/en/stable/contributing/development/core_and_modules/internal_rendering_architecture.html (accessed 20.03.2025)

15. Godot Rays Shader. Godot Shaders. URL : https://godotshaders.com/shader/god-rays/ (accessed 25.03.2025)

16. 9-slice scaling. Wikipedia. URL : https://en.wikipedia.org/wiki/9-slice_scaling (accessed 31.03.2025)

ДОДАТОК А

Класова архітектура рушія Godot



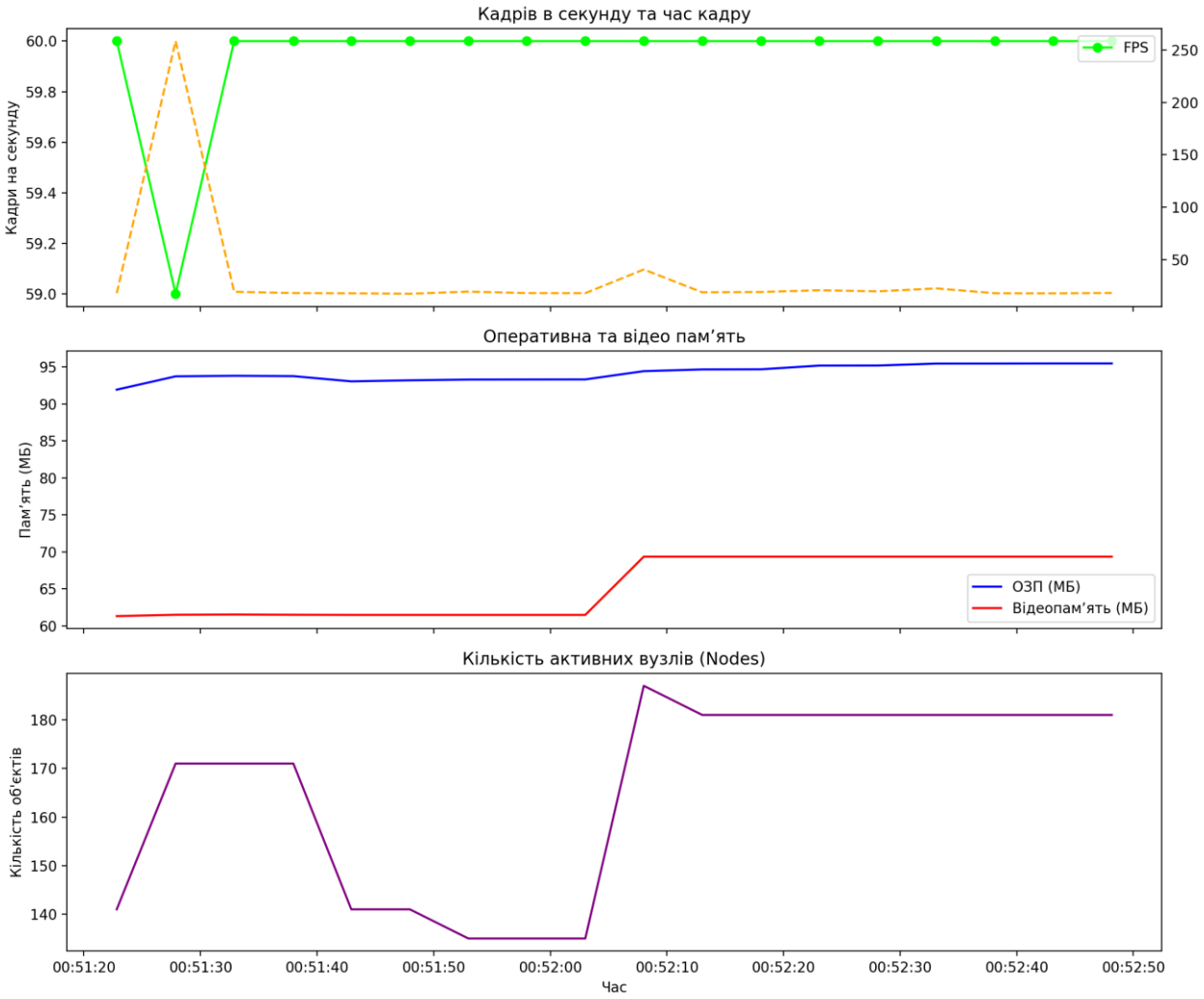
ДОДАТОК Б

Функція переходу скінченого автомату ворога

Поточний стан	Вхідний символ	Наступний стан
IDLE	Гравця помічено	CHASE
IDLE	Патрулювання	WALK
IDLE	Вбито	DEATH
WALK	Гравця помічено	CHASE
WALK	Перешкода	JUMP
WALK	Простоювання	IDLE
WALK	Вбито	DEATH
JUMP	Гравця помічено	CHASE
JUMP	Патрулювання	WALK
JUMP	Вбито	DEATH
CHASE	Гравця втрачено	WALK
CHASE	Перешкода	JUMP
CHASE	Вбито	DEATH

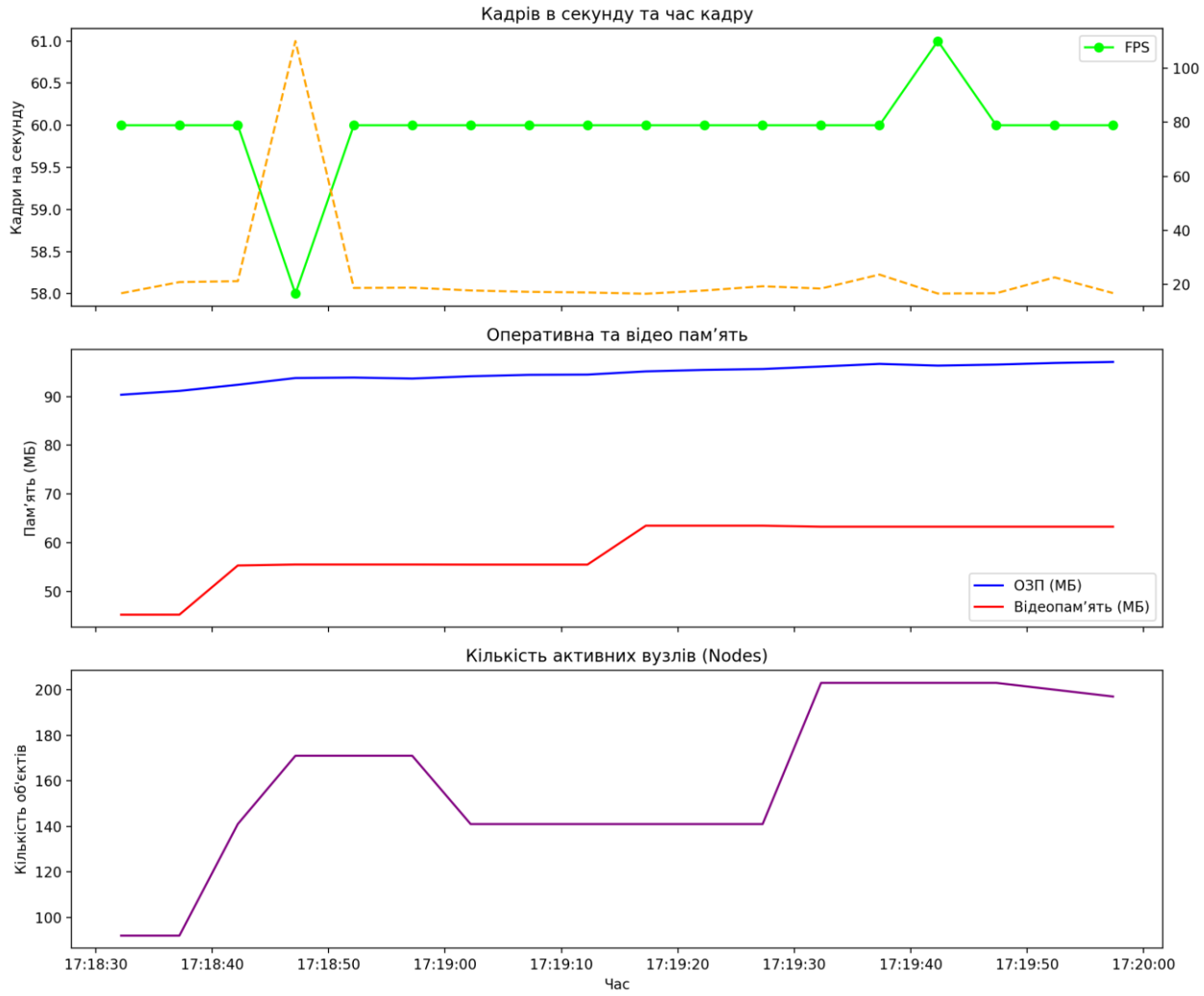
ДОДАТОК В

Заміри продуктивності на ПК1



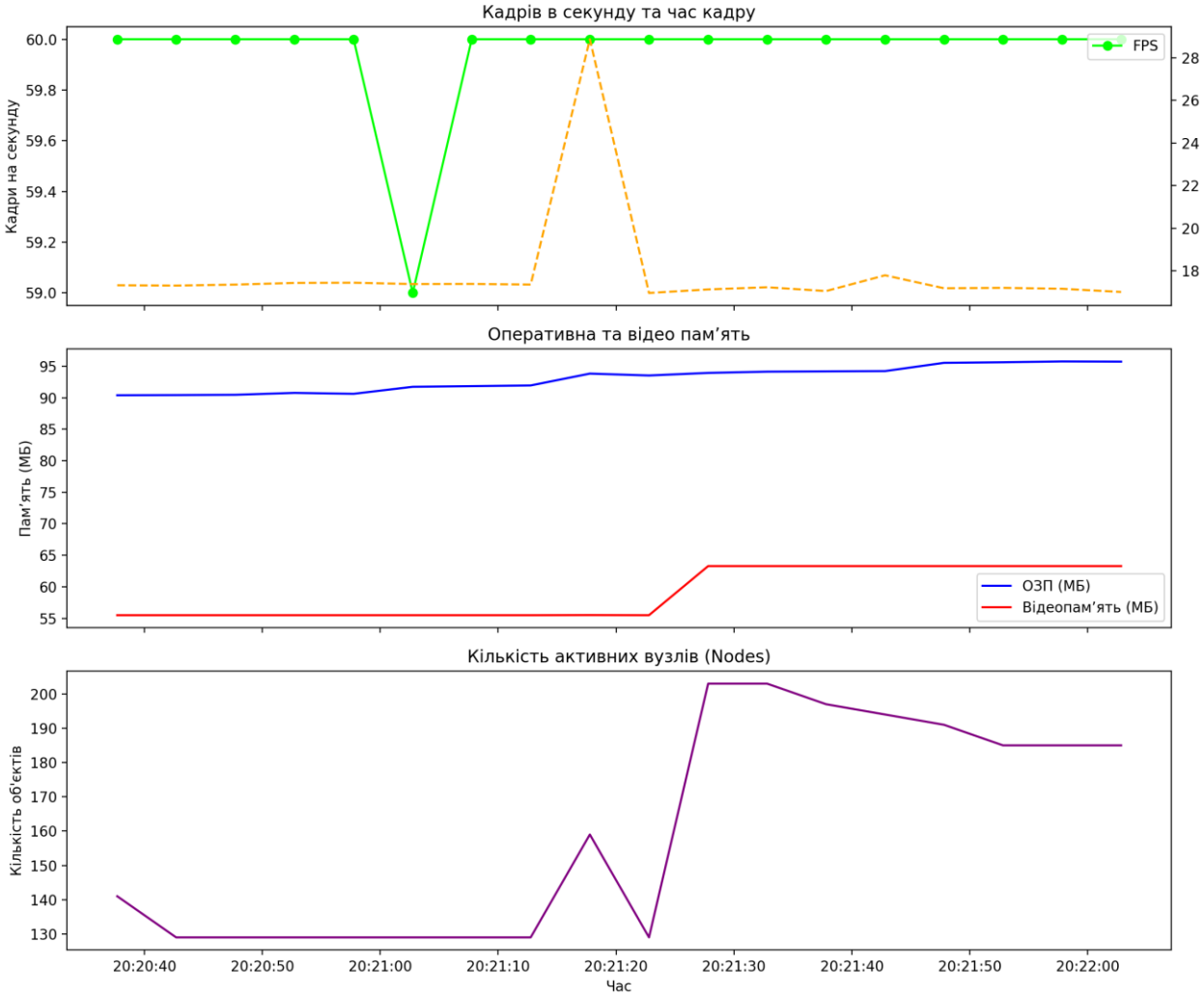
ДОДАТОК Г

Заміри продуктивності на ПК2



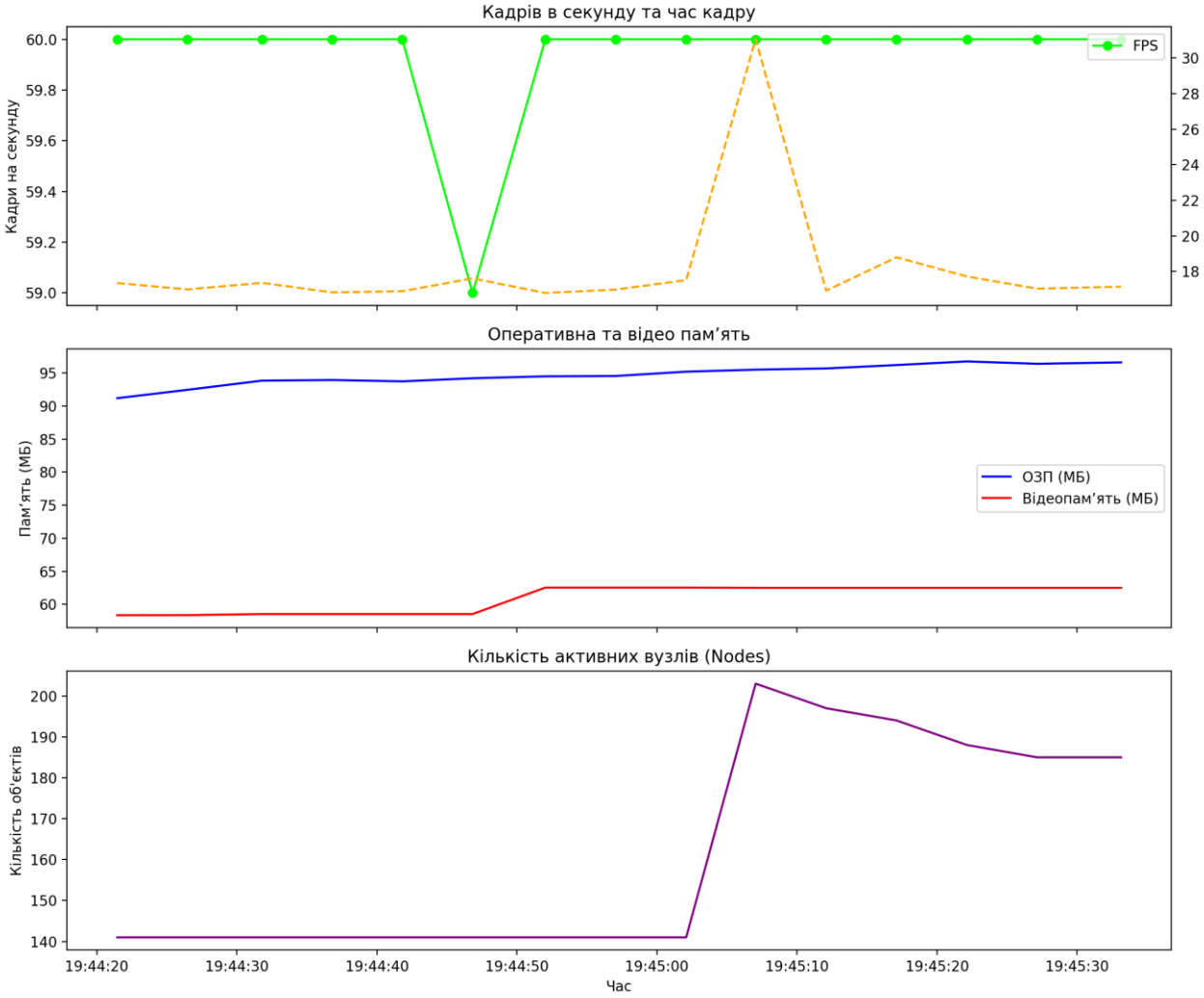
ДОДАТОК Д

Заміри продуктивності на ПК3



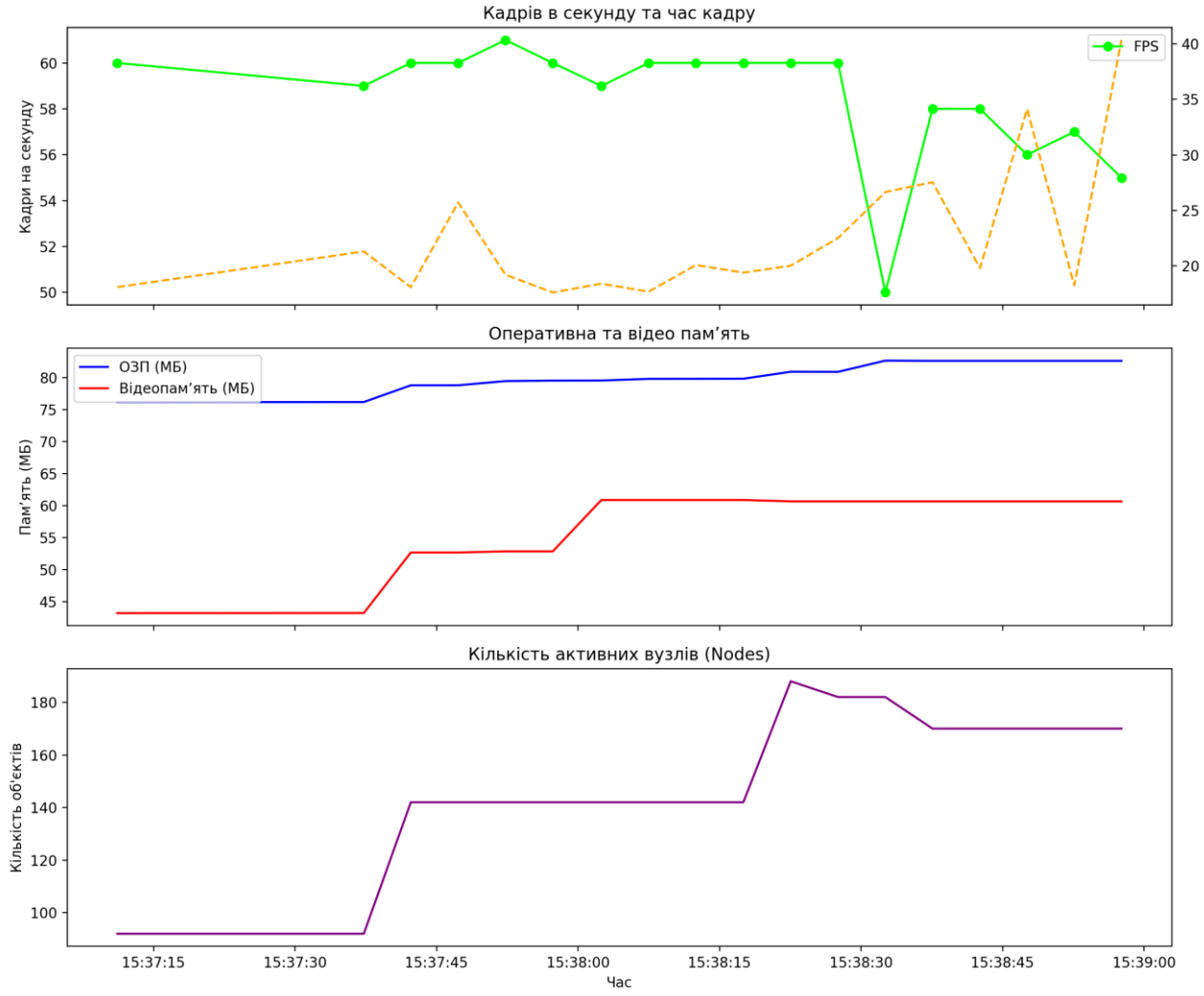
ДОДАТОК Е

Заміри продуктивності на ПК4



ДОДАТОК Ж

Заміри продуктивності на Realme 6 Pro в режимі рендеру Mobile



ДОДАТОК И

Заміри продуктивності на Realme 6 Pro в режимі рендеру Compatibility

