Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії)

Пояснювальна записка

до магістерської кваліфікаційної роботи

магістр
(освітньо-кваліфікаційний рівень)

на тему: "Ігрові механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D"

08-54.МКР.004.00.000 ПЗ

Виконала: студентка групи КІ-22мз
спеціальності
123 – Комп`ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Колеснікова Н.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф.ОТ
Городецька О.С.
(прізвище та ініціали)
Опонент: к.т.н., доц.каф.МБІС
Карпінець В.В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«15» 06 2024 року

Вінниця 2024

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень —магістр

Спеціальність 123 — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

«_6_» __лютого__ 2024 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту _____Колесніковій Наталії Сергіївні_____
(прізвище, ім'я, по батькові)

1 Тема роботи: "Ігрові механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D", керівник роботи: к.т.н., доцент кафедри ОТ Городецька О. С. затверджена наказом Вінницького національного технічного університету від 11.03.2024 року № 111.

2 Строк подання студентом роботи: 24.05.2024.

3 Вихідні дані до роботи: ігрові механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D; підтримка різних платформ (Windows, macOS, Linux, Android, iOS); використання Photon для реалізації мультиплеєра; тестування ігрових механік на різних пристроях; методика тестування та перевірки продуктивності гри.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ; огляд і аналіз предметної області; аналіз проблеми та постановка задачі; аналіз операційних систем для розробки ігор; порівняння операційних систем для ПК, мобільних пристроїв та консолей; виклики та

переваги кросплатформенної розробки; теоретичні основи кросплатформенної розробки на Unity3D; огляд можливих інструментів для розробки гри; кросплатформне середовище розробки Unity; практична реалізація кросплатформенного проєкту; тестування кросплатформенного проєкту; економічна частина; висновки; додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схема прототипу гри; структура ігрових механік; інтерфейсне вікно Unity3D; порівняння продуктивності гри на різних платформах; блок-схема лобі.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3,4	Крупельницький Л. В., к.т.н., доц. каф.ОТ		
5	Небава М.І., к.е.н., професор каф. ЕПВМ		

7 Дата видачі завдання 12.03.2024. Календарний план виконання етапів роботи наведено в таблиці 2.

Таблиця 2 — Календарний план

№з/п	Назва етапів дипломної магістерської роботи	Строк	Примітка
1	Огляд і аналіз предметної області	29.03.2024	Виконано
2	Теоретичні основи кросплатформенної розробки на Unity3D	15.04.2024	Виконано
3	Практична реалізація кросплатформенного проєкту на Unity3D	17.05.2024	Виконано
4	Тестування кросплатформенного проєкту	22.05.2024	Виконано
5	Економічна частина	23.05.2024	Виконано

Студент _____ Колеснікова Н.С.

Керівник магістерської кваліфікаційної роботи _____ Городецька О.С.

Консультант з економічної частини _____ Небава М.І.

АНОТАЦІЯ

УДК 004.4

Колеснікова Н.С. "Ігрові механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D" Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2024. 108 с

На укр. мові. Бібліогр. 20 назв; рис: 29; таб.:10.

В магістерській роботі було адаптовано ігрові механіки для кросплатформенних ігор, з використанням Unity3D. Актуальність обумовлена розвитком технологій у галузі відеоігор та потребою в універсальних рішеннях для розробки ігор для різних платформ. Метою є розробка методів адаптації ігрових механік для залучення широкої аудиторії шляхом оптимізації ігрового процесу для різних платформ.

Дослідження охоплює аналіз проблематики кросплатформенної розробки, вивчення апаратної структури ігрових платформ, розробку і тестування проекту. Використані методи включають аналіз літератури, порівняльний аналіз, експериментальну розробку та тестування.

Основні висновки підкреслюють важливість інтегрованого підходу до розробки кросплатформенних ігор, включаючи аналіз цільових платформ, адаптацію ігрових механік та інтерфейсів для забезпечення продуктивності та задоволення гравців. Результати важливі для розвитку галузі відеоігор, сприяючи створенню гнучких і доступних рішень.

Ключові слова: кросплатформенність, Unity3D, адаптація ігрових механік, оптимізація, відеоігри, розробка ігор.

ABSTRACT

Koliesnikova N.S. "Game Mechanics Adapted to the Hardware Structure of a Cross-Platform Project on Unity3D" Master's Thesis in the specialty 123 - Computer Engineering, Vinnytsia: VNTU, 2024.

In Ukrainian. Bibliography: 108 pages, 5 chapters; 29 figures, 10 tables, 20 sources listed in references.

The master's thesis focuses on adapting game mechanics for cross-platform games using Unity3D. The relevance is due to the rapid development of video game technologies and the need for universal solutions for game development on various platforms. The goal is to develop methods for adapting game mechanics to engage a broad audience by optimizing the gaming process for different platforms.

The research covers the analysis of cross-platform development issues, the study of the hardware structure of gaming platforms, and the development and testing of the project. Methods used include literature analysis, comparative analysis, experimental development, and testing.

The main conclusions emphasize the importance of an integrated approach to cross-platform game development, including the analysis of target platforms, adaptation of game mechanics, and interfaces to ensure performance and player satisfaction. The results are significant for the further development of the video game industry, contributing to the creation of flexible and accessible solutions.

Keywords: cross-platform, Unity3D, game mechanics adaptation, optimization, video games, game development.

ВСТУП

У сучасному світі індустрія відеоігор постійно розвивається, пропонуючи все нові та інноваційні технології, що змінюють уявлення про ігровий процес та його можливості. Однією з ключових тенденцій останніх років стала кросплатформенна розробка ігор, яка дозволяє гравцям насолоджуватися улюбленими іграми незалежно від використовуваної платформи. **Актуальність** теми дослідження обумовлена зростаючим попитом на кросплатформенні ігрові рішення та необхідністю оптимізації ігрового процесу для різноманітних апаратних платформ.

Робота виконується у зв'язку з науковими програмами, планами, темами, що вивчають технології розробки відеоігор, зокрема кросплатформенність як ключову концепцію сучасної ігрової індустрії. Вона спрямована на розробку методологічних та практичних рішень для ефективної адаптації ігрових механік під особливості різних платформ, використовуючи потужності рушія Unity3D.

Мета дослідження полягає у визначенні оптимальних підходів до адаптації ігрових механік для кросплатформенних проектів, розроблених на Unity3D, з метою забезпечення високої якості ігрового досвіду для користувачів різних платформ.

Задачі дослідження:

— аналізувати існуючі проблеми та виклики, пов'язані з кросплатформенною розробкою ігор;

— дослідити особливості апаратної структури різних ігрових платформ;

— розробити методики адаптації ігрових механік для ефективної роботи на різноманітних платформах;

— протестувати та оцінити ефективність запропонованих рішень на прикладі кросплатформенного проекту.

Об'єкт дослідження — кросплатформенні ігрові проекти.

Предмет дослідження — адаптація ігрових механік до апаратної структури різних платформ з використанням Unity3D.

Методи дослідження включають аналіз наукової літератури, порівняльний аналіз, експериментальну розробку та тестування.

Новизна одержаних результатів полягає в комплексних рекомендацій щодо оптимізації ігрових механік для кросплатформенних ігор, що враховують особливості апаратного забезпечення та забезпечують високу якість ігрового процесу.

Практичне значення одержаних результатів виражається у підвищенні якості розробки кросплатформенних ігрових проектів, зниженні витрат часу та ресурсів на адаптацію ігор для різних платформ, а також у покращенні якості кінцевого продукту для користувачів.

Ця магістерська робота спрямована на вирішення актуальних задач кросплатформенної розробки, сприяючи подальшому розвитку індустрії відеоігор і покращенню ігрового досвіду гравців на різноманітних платформах.

Публікація за темою роботи – "Адаптація ігрових механік до апаратної структури кросплатформенного проєкту на Unity3D"

Колеснікова Н.С., Крупельницький Л.В., Городецька О.С. (2024). . в НТКП ВНТУ. Факультет інформаційних технологій та комп'ютерної інженерії. Отримано з https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21186/17564

1. ОГЛЯД І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз проблеми та постановка задачі

Кросплатформенна розробка ігор є однією з найбільш стрімко розвиваючихся галузей сучасної ігрової індустрії, що ставить перед розробниками складні виклики та вимагає комплексного підходу до вирішення задач. Адаптація ігор під різноманітні апаратні платформи та операційні системи не тільки відкриває доступ до широкої аудиторії, але й сприяє створенню універсальних ігрових рішень, які можуть задовольнити вимоги та очікування гравців з різними уподобаннями.

Ключові виклики кросплатформенної розробки включають:

— фрагментація апаратних та програмних платформ, розмаїття графічних і обчислювальних можливостей, екранів різних роздільних здатностей, а також відмінності між операційними системами вимагають розробки адаптивних ігор, здатних ефективно працювати в широкому спектрі умов;

— відмінності у механізмах вводу та контролі розробники стикаються з необхідністю інтеграції різноманітних способів взаємодії з іграми, від традиційних клавіатур і мишей до сенсорних екранів та спеціалізованих ігрових контролерів;

— сумісність з операційними системами постійне оновлення ОС та їхніх версій ставить під загрозу стабільність і сумісність ігор, вимагаючи регулярних оновлень та патчів.

Для вирішення цих викликів визначено наступні задачі:

— створення універсальної архітектури, необхідно розробити архітектуру, яка дозволяє легко масштабувати та адаптувати ігровий контент та механіку до різних платформ, мінімізуючи зусилля на переробку та оптимізацію;

— глибока оптимізація ресурсів, розробка ефективних алгоритмів та використання передових технік оптимізації дозволить забезпечити високу продуктивність ігор навіть на пристроях з обмеженими можливостями;

— інноваційні рішення в інтерфейсах управління необхідно створити адаптивні та інтуїтивно зрозумілі інтерфейси, що забезпечують зручне управління в різних ігрових середовищах;

— розширене тестування та якісне забезпечення, впровадження комплексних методів тестування на ранніх етапах розробки допоможе виявити та виправити потенційні проблеми сумісності та продуктивності, забезпечуючи високу якість кінцевого продукту;

— забезпечення безперервної інтеграції та доставки (CI/CD) розвиток методологій безперервної інтеграції та безперервної доставки стає ключовим елементом у спрощенні процесів розробки та випуску ігор на різноманітні платформи, автоматизація процесів тестування, збірки та розгортання дозволяє забезпечити високу якість продукту і швидке впровадження змін;

— врахування особливостей цільових ринків розуміння культурних та регіональних особливостей аудиторії важливо для створення ігор, які будуть добре сприйняті на різних ринках, це може впливати на вибір тематики ігор, елементів інтерфейсу, а також стратегій монетизації;

— адаптація до технологічних трендів швидкий розвиток технологій, таких як штучний інтелект, машинне навчання, та віртуальна та доповнена реальність, відкриває нові можливості для кросплатформенної розробки використання цих технологій може допомогти створювати більш інтерактивні та занурюючі ігрові досвіди;

— оптимізація для мережевих технологій, зростаюча популярність мультиплеєрних ігор та ігор, які вимагають постійного з'єднання з Інтернетом, вимагає від розробників глибокого розуміння мережевих технологій та оптимізації мережевого коду для забезпечення стабільності та низької затримки;

— юзабіліті та доступність, створення ігор, які є зручними для користувачів з різними фізичними можливостями та вподобаннями, є важливим аспектом кросплатформенної розробки, це включає розробку адаптивних інтерфейсів.

Реалізація цих задач вимагає не лише технічної компетентності, але й креативного підходу до процесу розробки, а також глибокого розуміння потреб та вподобань цільової аудиторії. Успішна кросплатформенна розробка залежить від здатності команди адаптуватися до швидко змінюваних технологічних умов і потреб

користувачів, створюючи ігри, які вирізняються якістю, доступністю та ігровим досвідом на будь-якій платформі.

1.2 Доцільність розробки

У контексті сучасної ігрової індустрії, кросплатформенна розробка ігор набуває особливої актуальності. Ця стратегія відіграє ключову роль у досягненні максимально широкої аудиторії, оптимізації витрат та ресурсів на розробку та підтримку продукту на різних платформах [1]. Доцільність кросплатформенної розробки може бути аргументована кількома важливими факторами:

Розширення ринку та аудиторії. Забезпечення доступності ігор на різних платформах, включаючи мобільні пристрої, ПК, консолі та навіть веб, дозволяє залучити найширшу можливу аудиторію. Такий підхід не тільки збільшує потенційний ринок збуту продукту, але й сприяє формуванню більш лояльної спільноти користувачів.

Оптимізація виробничих витрат. Розробка однієї кросплатформеної бази коду знижує загальні витрати порівняно з розробкою окремих версій гри для кожної платформи. Це дозволяє більш ефективно використовувати ресурси та час команди, а також спрощує процес оновлення та випуску патчів.

Підвищення конкурентоспроможності продукту. У світі, де користувачі очікують можливості грати у свої улюблені ігри на будь-якому пристрої, кросплатформенність стає значною конкурентною перевагою. Ігри, які пропонують безшовний досвід на різних пристроях, збільшують свою привабливість та вартість для користувачів.

Гнучкість у монетизації. Кросплатформенна розробка відкриває ширші можливості для монетизації, дозволяючи використовувати різні моделі доходу відповідно до особливостей кожної платформи. Це включає внутрішньо ігрові покупки, підписки, рекламу та інші стратегії, які можуть бути оптимізовані для досягнення максимальної ефективності.

Сприяння інноваціям — кросплатформенний підхід стимулює пошук інноваційних технологічних та дизайнерських рішень. Розробники мають можливість

експериментувати з новими ідеями та інтегрувати передові технології, такі як штучний інтелект, машинне навчання, віртуальна та доповнена реальність, що може підвищити якість ігрового досвіду.

Враховуючи вище вказані фактори, доцільність кросплатформенної розробки ігор стає очевидною. Цей підхід не тільки сприяє розширенню ринкових можливостей і оптимізації витрат, але й відкриває нові горизонти для інновацій та покращення користувацького досвіду в ігровій індустрії.

1.3 Аналіз операційних систем для розробки ігор

Операційна система - це програмне забезпечення, яке керує різними аспектами роботи комп'ютера або іншого пристрою [2]. Вона відповідає за управління апаратними ресурсами, забезпечення взаємодії між користувачем та пристроєм, запуск та виконання програм, а також забезпечення безпеки та стабільності системи.

У сучасній індустрії ігрової розробки вибір операційної системи має велике значення. Різні ОС можуть мати відмінні характеристики та інструменти для розробки ігор, що впливає на їхню розповсюдженість та популярність серед розробників.

Загальний рейтинг операційних систем в Україні, включаючи десктопи, мобільні, планшети та ігрові приставки, показує, що лідером є Windows.

На рисунку 1 наведена повна статистики використання операційних систем в Україні.

Windows залишається однією з найбільш популярних платформ для розробки ігор завдяки широкому набору інструментів та технологій, доступних для розробників [3]. macOS, зі своєю високою якістю та оптимізацією, залишається важливим вибором для тих, хто працює у середовищі Apple. Крім того, зростання популярності мобільних ігор робить Android та iOS привабливими платформами для розробки, забезпечуючи доступ до великого ринку гравців. Також Linux набирає популярність серед розробників, особливо завдяки підтримці геймерської платформи Steam та розширенню ігрової індустрії на цю операційну систему.

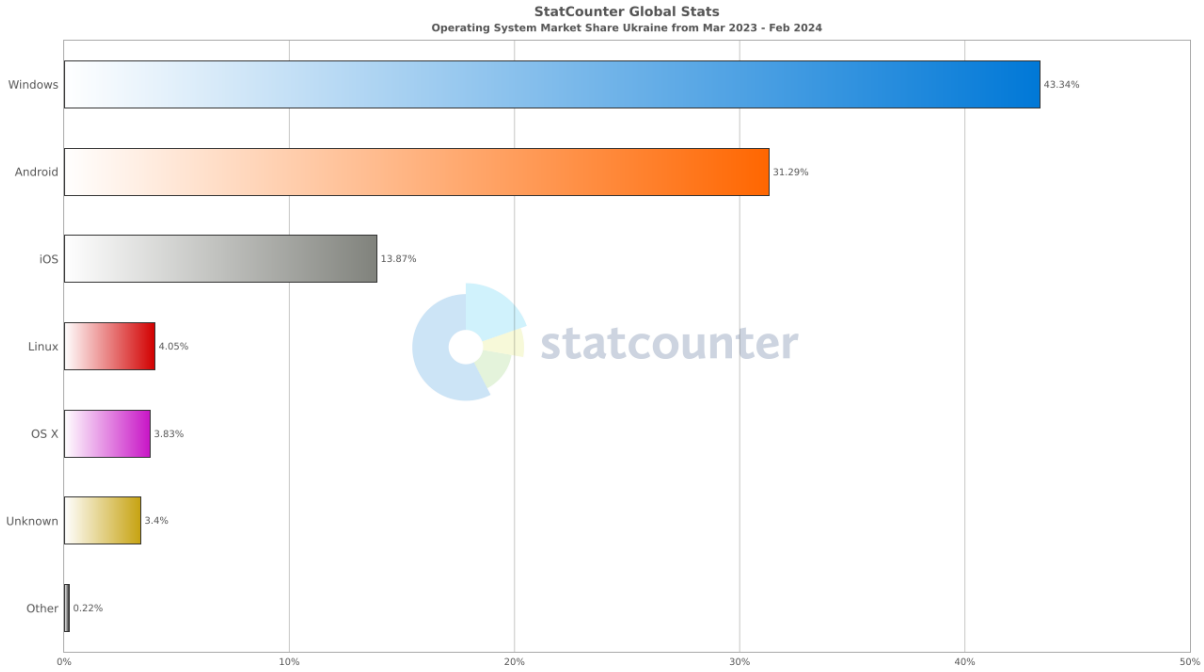


Рисунок 1 — Статистика використання операційних систем в Україні

1.3.1 Детальний аналіз операційних систем для ПК

В рамках аналізу операційних систем для розробки ігор, ми розглянемо трьох ключових гравців на ринку: Windows, macOS та Linux. Кожна з цих систем має свої унікальні характеристики, переваги та обмеження, які впливають на процес розробки ігор. Аналіз базується на кількох критично важливих аспектах, включаючи доступність інструментів розробки, сумісність з апаратним забезпеченням, розповсюдження ігор, а також спільноту розробників. Оцінюючи кожну операційну систему через ці параметри, ми отримуємо цілісне уявлення про те, як вони впливають на ігрову індустрію та процес розробки ігор зокрема.

Для наочного представлення основних відмінностей між операційними системами, розглянутими з боку розробки ігор, складено таблицю порівняння 3.

Ця таблиця допомагає візуалізувати ключові відмінності та переваги кожної операційної системи, надаючи цінну інформацію для розробників ігор при виборі платформи для своїх проектів.

Для більш глибокого розуміння та аналізу впливу операційних систем на процес розробки ігор, перейдемо до детального розгляду кожної з них, враховуючи специфіку їх застосування у цій сфері.

Таблиця 3 — Порівняння операційних систем

Характеристика	Windows	macOS	Linux
Підтримка інструментів	Широка підтримка (Unity, Unreal)	Обмежене, але стабільне (Unity)	Обмежена підтримка
База користувачів	Найбільша серед геймерів	Менша, переважно не геймерська	Найменша, але лояльна
Сумісність з апаратним забезпеченням	Широка сумісність	Обмежена до Apple апаратури	Висока, але може потребувати налаштувань
Оптимізація ігор	Легше оптимізувати за рахунок однорідності	Потребує додаткової роботи	Може бути складною без підтримки
Ліцензування	Платна ОС	Платна ОС, включена в апаратуру	Безкоштовна, відкрите ПЗ
Розповсюдження ігор	Широкі можливості	Обмежене, Mac App Store	Переважно через репозиторії або Steam
Спільнота розробників	Велика та різноманітна	Менша, але високо кваліфікована	Технічно нахилені та досвідчені

Windows домінує на ринку як найпопулярніша операційна система для розробки ігор. Її широка підтримка інструментів розробки, включаючи Unity та Unreal Engine, робить її першим вибором для багатьох розробників. Велика база користувачів означає велику потенційну аудиторію для розроблених ігор, а сумісність

з широким спектром апаратного забезпечення забезпечує гнучкість у виборі компонентів для розробки та тестування. Основним обмеженням може бути лише витрати на ліцензію ОС, хоча це рідко стає суттєвою перешкодою для професійних розробників.

MacOS пропонує унікальні переваги через тісну інтеграцію з екосистемою Apple. Висока стабільність та оптимізація для апаратури Apple роблять цю ОС привабливою для розробників, які цінують якість та надійність. Розробка ігор під macOS може також відкрити доступ до аудиторії користувачів продуктів Apple, що характеризуються високою платоспроможністю. Проте, обмежена частка ринку серед геймерів та менш потужне апаратне забезпечення в порівнянні з ігровими ПК на Windows можуть обмежувати потенціал розповсюдження ігор.

Linux виступає як ідеальна платформа для розробників, що цінують відкритість, гнучкість та повний контроль над процесом розробки. Відкрите програмне забезпечення та безкоштовне використання роблять Linux привабливим вибором для стартапів та незалежних розробників. Водночас, обмежена підтримка ігрових рушіїв та інструментів може вимагати додаткових зусиль для адаптації та оптимізації ігор. Менша аудиторія геймерів на Linux також є важливим фактором, який впливає на комерційну успішність проектів.

1.3.2 Порівняння мобільних операційних систем

У світі мобільних технологій домінують дві основні операційні системи: Android та iOS. Ці системи є серцем мільйонів мобільних пристроїв по всьому світу, від смартфонів та планшетів до розумних годинників та домашніх розважальних систем [4]. Розробка ігор для мобільних платформ вимагає від розробників глибокого розуміння особливостей кожної з цих систем, їх сильних та слабких сторін, а також знання їх апаратних можливостей.

Android, розроблений Google, є відкритою операційною системою, що дозволяє виробникам легко адаптувати її до своїх пристроїв. Це призвело до великої фрагментації ринку, з безліччю пристроїв різних форм і розмірів, кожен з яких має свої специфікації.

iOS, розроблена Apple, є закритою системою, яка працює виключно на апаратному забезпеченні Apple, включаючи iPhone, iPad та iPod Touch. Ця екосистема надає більшу однорідність та стабільність, але з меншою гнучкістю у порівнянні з Android.

При розробці ігор для мобільних платформ, важливо враховувати наступні аспекти:

— доступність інструментів розробки: Android пропонує Android Studio, в той час як iOS розробники використовують Xcode, обидва набори інструментів мають свої переваги, включаючи емулятори, дебагери та інтегровані середовища розробки;

— мови програмування, для Android переважно використовують Java та Kotlin, у той час як iOS розробки базуються на Objective-C та Swift. Вибір мови може значно вплинути на продуктивність ігрового проекту та його сумісність з операційною системою;

— фрагментація апаратної платформи, Android-пристрої мають широкий спектр екранних розмірів, роздільних здатностей та апаратних специфікацій, що створює додаткові виклики для розробників у плані оптимізації та адаптації ігор, натомість, iOS має обмежений набір пристроїв, що полегшує цільову оптимізацію;

— ринкова доля та аудиторія, розуміння цільової аудиторії та розподілу ринкових часток між різними платформами допоможе визначити потенційний дохід від ігрового проекту;

— процес публікації Google Play та Apple App Store мають різні вимоги до публікації додатків, включаючи перевірку якості, безпеки та відповідність правилам, ці вимоги можуть вплинути на час випуску та доступність ігор на різних платформах.

Розуміння цих ключових відмінностей та викликів є критично важливим для розробки успішних мобільних ігор, що можуть ефективно функціонувати на різних платформах.

1.3.3 Порівняння Консолей

В аналізі консолей для розробки ігор ми зосередимося на трьох ключових платформах: PlayStation, Xbox, і Nintendo Switch. Кожна з цих консолей має унікальні

характеристики, які впливають на процес розробки ігор, їх розповсюдження, а також на стратегії оптимізації. Для кращого розуміння цих відмінностей ми використовуємо таблицю для порівняння основних аспектів розробки на цих платформах.

Для наочності, представимо ключові аспекти розробки ігор для PlayStation, Xbox і Nintendo Switch у формі таблиці 4.

Таблиця 4 — Порівняння консолей

Категорія	PlayStation	Xbox	Nintendo Switch
Ліцензійні вимоги	Строгі, потрібна ліцензія від Sony	Строгі, потрібна ліцензія від Microsoft	Потрібна ліцензія, але доступніша
Апаратні обмеження	Висока продуктивність, фокус на графіці	Висока продуктивність, сумісність з PC	Портативність, обмеження продуктивності
Аудиторія	Вимогливі геймери, широка аудиторія	Широка аудиторія, ігри на PC	Різноманітна аудиторія, сімейні ігри
Оптимізація ігор	Вимагає високої деталізації, використання ексклюзивних функцій	Оптимізація під потужне апаратне забезпечення, хмарні технології	Адаптація під портативність, інноваційне керування

Ця таблиця наглядно демонструє ключові різниці між платформами, які мають бути враховані при розробці ігор. Розробка ігор для консолей вимагає глибокого розуміння їхніх унікальних вимог та можливостей. Вибір між PlayStation, Xbox, та Nintendo Switch залежить від цілей розробки, бажаної аудиторії, а також від технічних та креативних обмежень проекту. Кожна платформа пропонує свої унікальні переваги

та виклики, тому важливо враховувати ці аспекти при плануванні розробки ігрового проекту.

PlayStation від Sony є однією з найпопулярніших консолей на ринку, відомою своїми потужними технічними характеристиками та великою бібліотекою ексклюзивних ігор. Розробка ігор для PlayStation вимагає отримання ліцензії від Sony, що може бути процесом з високим порогом входження. Водночас, доступ до великої та активної аудиторії може виправдати ці зусилля.

Xbox від Microsoft має тісну інтеграцію з Windows, що дозволяє легше портувати ігри між PC та консоллю. Ліцензійні вимоги схожі на PlayStation, але платформа пропонує додаткові можливості через хмарні сервіси та Game Pass. Оптимізація ігор для Xbox часто спрямована на використання потужного апаратного забезпечення.

Nintendo Switch виділяється на тлі інших консолей завдяки своїй унікальності, поєднуючи стаціонарну та портативну ігрові платформи в одному пристрої. Ліцензування та розробка ігор для Switch є доступнішими, хоча розробники стикаються з викликами оптимізації під менш потужне апаратне забезпечення. Втім, інноваційне керування та унікальні можливості Switch пропонують нові шляхи для креативності у геймдизайні.

Розробка ігор для консолей вимагає глибокого розуміння їхніх унікальних вимог та можливостей. Вибір між PlayStation, Xbox, та Nintendo Switch залежить від цілей розробки, бажаної аудиторії, а також від технічних та креативних обмежень проекту. Кожна платформа пропонує свої унікальні переваги та виклики, тому важливо враховувати ці аспекти при плануванні розробки ігрового проекту [5].

1.3.4 Аналіз операційних систем для розробки VR-проєктів

У світі віртуальної реальності (VR) вибір операційної системи та платформи є ключовим для розробки ефективних і залучаючих VR-досвідів. Від роботи з мобільними VR-гарнітурами до потужних настільних VR-систем, розробники повинні враховувати унікальні особливості та обмеження кожної платформи. Наступний аналіз зосереджений на порівнянні операційних систем і платформ, які

підтримують розробку VR, з метою надати цінну інформацію для вибору оптимального шляху розробки.

Ринок VR-технологій швидко розвивається, пропонуючи розробникам широкий спектр операційних систем і платформ, таких як Windows для настільних VR-систем, Android для мобільних VR-пристроїв, а також пропрієтарні системи, які використовуються у таких пристроях, як Oculus Quest. Кожна з цих систем має свої сильні та слабкі сторони, які впливають на процес розробки, розподіл та взаємодію з кінцевим користувачем.

Порівняння VR-операційних систем:

— Windows домінуюча операційна система для настільних VR-пристроїв, Windows підтримує лідируючі VR-гарнітури, включаючи Oculus Rift, HTC Vive та Valve Index, розробники мають доступ до потужних інструментів та API для створення високоякісних VR-досвідів з використанням таких платформ, як Unity або Unreal Engine;

— Android операційна система, яка часто використовується у мобільних VR-пристроях, таких як Samsung Gear VR та Google Daydream, она пропонує гнучкість у розробці VR-додатків для широкого спектра мобільних пристроїв, але з обмеженнями у продуктивності та якості досвіду порівняно з настільними системами;

— пропрієтарні системи, пристрої, такі як Oculus Quest, використовують власні операційні системи, оптимізовані для бездротового VR-досвіду, це дає розробникам можливість створювати інтуїтивно зрозумілі та високоефективні VR-досвіди, хоча й із певними обмеженнями в сумісності та розподілі.

При виборі операційної системи для розробки VR-проєктів важливо враховувати цілі проєкту, цільову аудиторію та ключові вимоги до продуктивності та якості досвіду. Настільні системи на базі Windows забезпечують найвищу продуктивність і якість, тоді як мобільні та бездротові VR-пристрої пропонують гнучкість і доступність за рахунок деякого зниження продуктивності. Вибір правильної операційної системи і платформи є ключовим для успіху будь-якого VR-проєкту.

1.4 Виклики та переваги кросплатформенної розробки ігор

Кросплатформенна розробка ігор стає все більш популярною у світі відеоігор, дозволяючи гравцям насолоджуватися своїми улюбленими іграми на різних пристроях, включаючи ПК, консолі та мобільні пристрої. Цей підхід має як унікальні виклики, так і значні переваги, які впливають на стратегії розробки, випуску і підтримки ігрових проектів.

Виклики кросплатформенної розробки:

— фрагментація апаратного забезпечення, різноманітність апаратних специфікацій між пристроями може створювати складнощі в оптимізації ігрового процесу та графіки, щоб забезпечити стабільну продуктивність на всіх платформах;

— відмінності в системах управління, кожна платформа має свою систему управління, що може вимагати розробки специфічних інтерфейсів для кожного типу пристроїв;

— оновлення та підтримка, підтримка одночасних оновлень і функцій через різні платформи може бути складною, особливо коли мова йде про виправлення помилок і додавання нового вмісту;

— маркетингові та ліцензійні обмеження, різні платформи можуть мати унікальні вимоги до публікації і маркетингу, що вимагає додаткових зусиль та ресурсів для дотримання правил кожного магазину додатків.

Переваги кросплатформенної розробки

— ширший охоплення аудиторії, розробка ігор для кількох платформ дозволяє досягти більшої аудиторії, збільшуючи потенційний дохід та популярність проекту;

— спільнота гравців, єдина кросплатформенна екосистема сприяє формуванню спільноти гравців, які можуть взаємодіяти та змагатися між собою, незалежно від використовуваної платформи;

— економія ресурсів, використання кросплатформених інструментів та рушіїв, таких як Unity або Unreal Engine, може знизити час і витрати на розробку, дозволяючи використовувати один і той же кодовий базис для різних платформ;

— легше оновлення та підтримка, одна база коду дозволяє розробникам легше випускати оновлення та підтримувати гру, забезпечуючи консистентний досвід для всіх користувачів.

Узагальнюючи, кросплатформенна розробка ігор пропонує величезний потенціал для розширення охоплення аудиторії та створення більш гнучких і доступних ігрових проектів. Незважаючи на виклики, пов'язані з оптимізацією та сумісністю, переваги, як правило, переважують складнощі, особливо з розвитком технологій та зростанням інструментів для кросплатформенної розробки.

1.5 Аспекти оптимізації ігрового процесу для різних платформ

Оптимізація ігрового процесу для різних платформ є ключовим аспектом кросплатформенної розробки ігор. Вона вимагає врахування апаратних обмежень, варіативності систем управління та відмінностей у користувацькому інтерфейсі між різними пристроями. Ефективна оптимізація забезпечує гладкий і залучаючий ігровий досвід для всіх користувачів, незалежно від того, на якій платформі вони грають. Нижче представлені ключові аспекти оптимізації ігрового процесу для різних платформ.

Продуктивність та апаратні ресурси:

— адаптація графічних налаштувань, необхідно налаштувати роздільну здатність, текстури, ефекти та інші графічні параметри залежно від апаратних можливостей кожної платформи, забезпечуючи оптимальне співвідношення якості та продуктивності;

— ефективне використання пам'яті, оптимізація використання пам'яті є важливою для забезпечення стабільності ігрового процесу, особливо на мобільних пристроях з обмеженою кількістю оперативної пам'яті.

Системи управління:

— адаптація до різних систем управління, розробка інтуїтивно зрозумілих і зручних систем управління для кожної платформи, враховуючи особливості сенсорних екранів, геймпадів та клавіатур;

— конфігурація управління, надання гравцям можливості налаштовувати елементи управління під особисті переваги може підвищити зручність ігрового процесу.

Адаптація інтерфейсу:

— масштабування інтерфейсу, оптимізація UI для різних роздільних здатностей та розмірів екранів, забезпечуючи зручне відображення на всіх пристроях;

— інтуїтивно зрозумілий дизайн, створення інтерфейсу, який є інтуїтивно зрозумілим для користувачів незалежно від платформи, з чіткими інструкціями та зворотнім зв'язком.

Мережева оптимізація:

— синхронізація даних між платформами, забезпечення ефективного обміну даними та стабільності мережевої гри у кросплатформенному середовищі;

— оптимізація мережевого коду, мінімізація затримки та оптимізація передачі даних для підтримки плавної мультиплеєрної гри.

Тестування та якість:

— платформоспецифічне тестування, ретельне тестування ігрового процесу на всіх цільових платформах для виявлення та виправлення помилок та проблем сумісності;

— збір відгуків, аналіз відгуків від користувачів різних платформ для виявлення аспектів, що потребують оптимізації або вдосконалення.

Ефективна оптимізація ігрового процесу для різних платформ не лише покращує досвід користувачів але й сприяє розширенню охоплення гри, залученню більшої аудиторії та підвищенню загальної якості проекту [6]. Завдяки зосередженню уваги на ключових аспектах оптимізації, розробники можуть успішно подолати виклики, пов'язані з кросплатформенною розробкою, та використовувати її переваги на повну.

Перший розділ магістерської роботи присвячений глибокому аналізу предметної області кросплатформенної розробки ігор, включаючи визначення основних проблем, постановку задач, обгрунтування доцільності розробки, а також детальний огляд різноманітних операційних систем для ПК, мобільних пристроїв та

консолей. В ході аналізу були виявлені ключові виклики та переваги, які супроводжують процес кросплатформенної розробки, та розглянуті аспекти оптимізації ігрового процесу для забезпечення високоякісного досвіду на різних платформах.

Основні виклики, які були виявлені, включають фрагментацію апаратного забезпечення, відмінності в системах управління, складнощі оновлень та підтримки, а також маркетингові та ліцензійні обмеження. Водночас, кросплатформенна розробка ігор відкриває значні переваги, такі як розширення охоплення аудиторії, формування спільноти гравців, економія ресурсів та спрощення процесу оновлення та підтримки ігор.

Оптимізація ігрового процесу для різних платформ вимагає уваги до таких аспектів, як адаптація графічних налаштувань, ефективне використання пам'яті, адаптація до різних систем управління, масштабування інтерфейсу, мережева оптимізація, а також ретельне тестування та збір відгуків від користувачів.

Розділ підкреслює важливість комплексного підходу до розробки кросплатформенних ігор, з акцентом на необхідність балансу між технічною оптимізацією та забезпеченням якісного ігрового досвіду. Висновки цього розділу закладають фундамент для подальшого дослідження та розробки кросплатформенного ігрового проекту, враховуючи всі виклики та можливості, які були ідентифіковані в ході дослідження.

2 ТЕОРЕТИЧНІ ОСНОВИ КРОСПЛАТФОРМЕННОЇ РОЗРОБКИ НА UNITY3D

2.1 Огляд можливих інструментів для розробки гри

У сучасному світі розробка кросплатформенних ігор стає все більш актуальною завдяки зростанню кількості пристроїв, які підтримують ігри: ПК, мобільні пристрої, консолі, VR/AR. Для ефективної реалізації таких проєктів розробники використовують різні ігрові рушії.

Згідно з опитуванням серед розробників ігор у 2023 році, найбільш популярними ігровими рушіями є Unity3D та Unreal Engine. Розподіл популярності різних ігрових рушіїв можна побачити на рисунку 2.

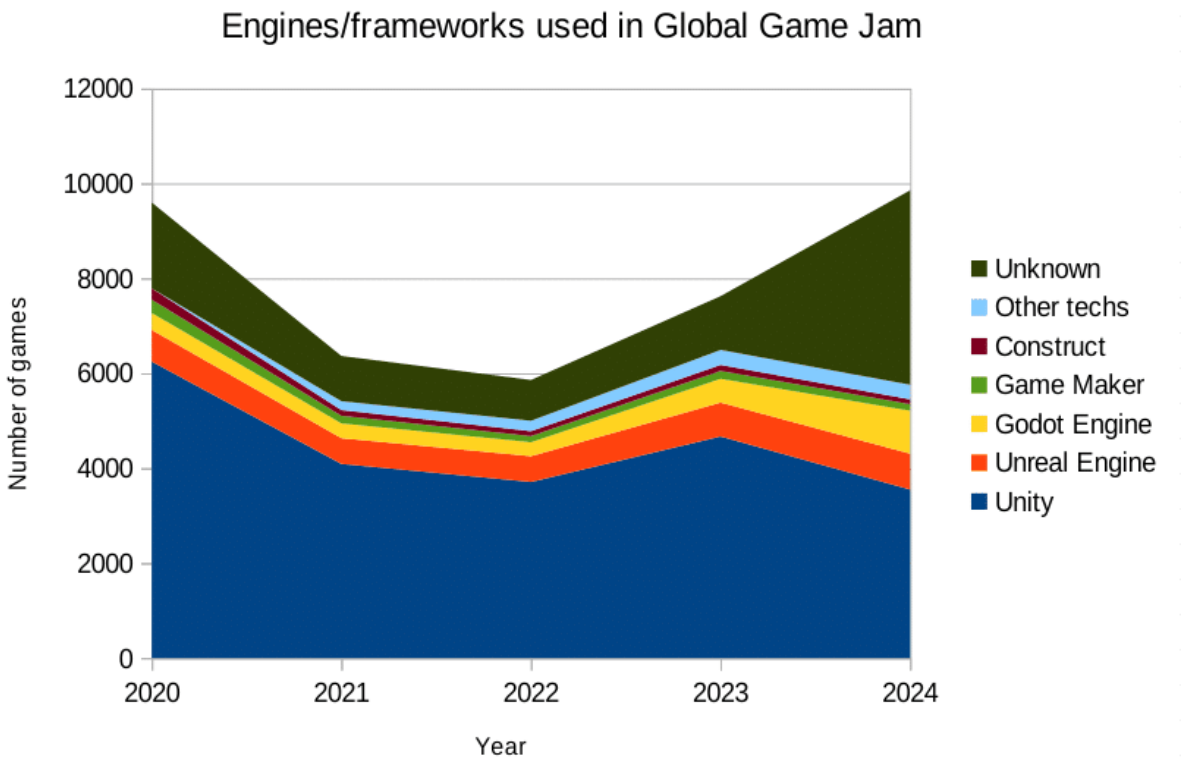


Рисунок 2 — Розподіл популярності ігрових рушіїв серед розробників

Зараз ми розглянемо дані для визначення значень 2023 та 2024 років на рисунку 3.

GGJ 2023 total	**7625**	With data:	**6493**
Engine	**Games**	**Percentage**	**Percent. known**
Unity	4669	61,23 %	71,91 %
Unreal Engine	715	9,38 %	11,01 %
Godot Engine	500	6,56 %	7,70 %
Game Maker	167	2,19 %	2,57 %
Construct 3	93	1,22 %	1,43 %
YAHAHA Studio	86	1,13 %	1,32 %
Ren'Py	48	0,63 %	0,74 %
GGJ 2024 total	**9857**	With data:	**5759**
Engine	**Games**	**Percentage**	**Percent. known**
Unity	3549	36,00 %	61,63 %
Godot Engine	905	9,18 %	15,71 %
Unreal Engine	757	7,68 %	13,14 %
Game Maker	140	1,42 %	2,43 %
Construct	99	1,00 %	1,72 %
Ren'Py	54	0,55 %	0,94 %
Gdevelop	42	0,43 %	0,73 %

Рисунок 3 — Розподіл ігрових рушіїв за 2023 та 2024 роки

Ігровий рушій Godot обігнав Unreal Engine як другий за популярністю серед ігрових рушій, в той час як Unity, можливо, через фіаско з спробою впровадження нової політики, зазнав досить значного падіння, впавши з вражаючих 61% до 36% у порівнянні з минулим роком. Тим не менш, Unity все ще залишається найпопулярнішим ігровим рушієм з дуже великим відривом.

Для порівняння основних характеристик популярних ігрових рушіїв використаємо таблицю 5 нижче.

2.1.1 Огляд рушія Unity3D

Unity3D займає лідируючу позицію серед ігрових рушіїв завдяки простому та інтуїтивному інтерфейсу, який підходить як для новачків, так і для професіоналів. Рушій підтримує розробку ігор для ПК (Windows, macOS, Linux), мобільних

пристроїв (Android, iOS), консолей (Xbox, PlayStation, Nintendo) та VR/AR-пристроїв [7].

Таблиця 5 — Порівняння основних характеристик популярних ігрових рушіїв

Параметр	Unity3D	Unreal Engine	Godot Engine	CryEngine
Простота використання	Висока	Середня	Висока	Середня
Графіка	Середня	Висока	Середня	Висока
Мова програмування	C#	C++	GDScript, C#	C++, Lua
Кросплатформеність	Висока	Висока	Висока	Середня
Анімація	Середня	Висока	Середня	Середня
Фізика	Середня	Висока	Середня	Висока
Ліцензія	Freemium	Freemium	MIT (безкоштовно)	Freemium

Основною мовою програмування є C#, а для графічного рендерингу підтримуються як вбудований рендерер, так і Scriptable Render Pipeline (SRP). Unity3D також забезпечує потужну фізику (2D та 3D) з інтегрованим редактором, систему анімації та вбудований магазин активів (Asset Store). Рушій інтегровано з Visual Studio та іншими сторонніми інструментами.

Переваги Unity3D:

— інтуїтивний інтерфейс;

— підтримка більшості платформ;

— велика спільнота розробників та багата база знань.

Недоліки Unity3D:

— обмежена оптимізація для великих AAA-проєктів;

— вбудована система рендерингу поступається за якістю іншим рушіям.

2.1.2 Огляд рушія Unreal Engine

Unreal Engine — це потужний ігровий рушій з підтримкою високоякісної графіки та реалістичної фізики. Основною мовою програмування є C++, але також доступний візуальний скриптинг Blueprints, який дозволяє створювати ігрову логіку без написання коду [8]. Unreal Engine підтримує кросплатформенну розробку для ПК, консолей, мобільних пристроїв та VR/AR-пристроїв, а також має потужну систему анімації з підтримкою риггінгу та кінематографії.

Переваги Unreal Engine:

— високоякісна графіка та фізика;

— можливість розробки AAA-проєктів;

— візуальний скриптинг Blueprints.

Недоліки Unreal Engine:

— складність освоєння та високі вимоги до обладнання розробників;

— довший цикл розробки у порівнянні з Unity3D.

2.1.3 Огляд рушія Godot Engine

Godot Engine є відкритим та безкоштовним ігровим рушієм, який підтримує розробку ігор для ПК, мобільних пристроїв та консолей. Основною мовою програмування є GDScript, але також підтримуються C#, C++ та візуальне програмування. Модульна архітектура Godot Engine дозволяє легко створювати ігрові сцени з гнучкою системою компонентів. Рушій підтримує як 2D, так і 3D графіку, хоча можливості 3D графіки тут обмежені у порівнянні з іншими рушіми.

Переваги Godot Endine:

— відкритий та безкоштовний код;

— модульна архітектура та підтримка візуального програмування;

— підтримка різних мов програмування.

Недоліки Godot Endine:

— обмежені можливості для 3D графіки;

— менша спільнота та обмежена база знань.

2.1.4 Огляд рушія CryEngine

CryEngine — це потужний ігровий рушій для створення високоякісних графічних ефектів та реалістичної фізики. Основними мовами програмування є C++ та Lua.

Рушій підтримує реалістичну фізику з можливістю руйнування об'єктів, але кросплатформеність обмежена підтримкою тільки ПК та консолей.

Переваги CryEngine:

— високоякісна графіка та фізика;

— підтримка високоякісних AAA-проєктів.

Недоліки CryEngine:

— складний в освоєнні;

— вимогливий до обладнання;

— обмежена підтримка мобільних платформ.

Отже, Unity3D є універсальним інструментом для кросплатформенної розробки завдяки простому інтерфейсу та широкому спектру можливостей. У порівнянні з іншими рушіями, Unity пропонує інтуїтивний інтерфейс та доступну екосистему. Unreal Engine забезпечує високоякісну графіку, але складніший в освоєнні та має високі вимоги до обладнання. Godot Engine є відкритим та безкоштовним, але має обмежену підтримку великих проєктів у порівнянні з іншими рушіями. CryEngine підходить для високоякісних AAA-проєктів, але складний в освоєнні та обмежений у кросплатформенності.

2.2 Кросплатформне середовище розробки Unity

Unity — одне з найпопулярніших кросплатформенних середовищ розробки завдяки своїй універсальності, простоті використання та багатому набору функцій. Засноване у 2005 році компанією Unity Technologies, середовище стрімко набуло популярності й стало вибором номер один для багатьох розробників.

Unity забезпечує розробку ігор для різних платформ із мінімальними змінами. Для розробників ігор операційні системи мають особливе значення, оскільки вони

визначають середовище, в якому гра буде працювати, і впливають на її продуктивність та можливості. Вибір операційної системи для розробки ігор вимагає врахування багатьох факторів, включаючи продуктивність, сумісність з апаратним забезпеченням, наявність інструментів розробки та вподобання кінцевих користувачів. Розподіл популярності різних операційних систем можна побачити на рисунку 4.

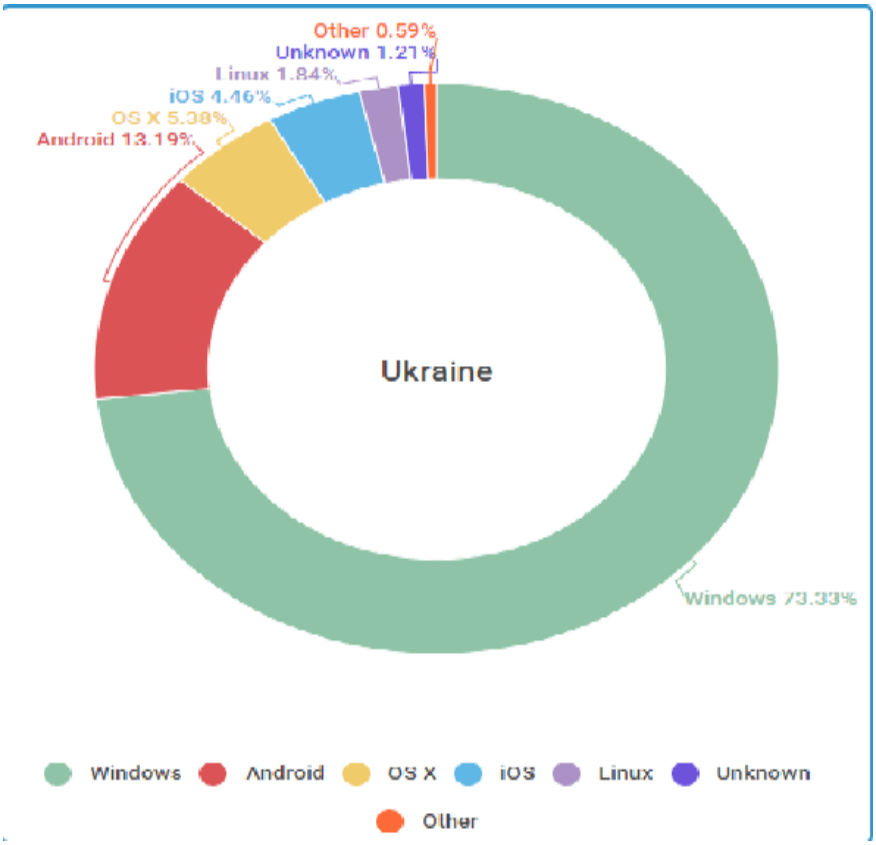


Рисунок 4 — Статистика використання операційних систем в Україні

В Україні найпопулярнішою операційною системою є Windows, яка використовується на 73.33% всіх пристроїв, від настільних комп'ютерів до портативних гаджетів. Це свідчить про домінуюче становище Windows на ринку операційних систем в країні та робить її ключовою платформою для розробки ігор. Багато розробників обирають Windows через її широку поширеність та підтримку потужних інструментів для створення ігор.

З розвитком мобільних технологій значну частку ринку зайняла операційна система Android, яка використовується на 13.19% пристроїв. Це демонструє

зростаючу популярність мобільних пристроїв серед українських користувачів та вимагає від розробників адаптації своїх ігор для цієї платформи. Android забезпечує відкриту екосистему та широкий спектр інструментів для розробки, що дозволяє створювати ігри для великої аудиторії.

Операційна система від Apple, OS X, посідає третє місце з 5.38% ринку. Навіть додавши до цього показник використання iOS, який складає 4.46%, загальна частка ринку Apple не перевищить популярність Android в Україні. Проте, розробка ігор для платформи Apple залишається важливою через високий рівень монетизації та лояльну аудиторію користувачів.

Такі дані підкреслюють важливість і різноманітність використання операційних систем в Україні, відображаючи поточні тенденції та переваги користувачів. Розробники ігор повинні враховувати ці тенденції при виборі платформ для своїх проєктів, забезпечуючи сумісність та оптимальну продуктивність своїх ігор на різних операційних системах.

Далі розглянемо основні можливості, особливості та переваги Unity.

Переваги Unity: Unity підтримує понад 25 платформ, включаючи ПК (Windows, macOS, Linux), мобільні пристрої (Android, iOS), консолі (Xbox, PlayStation, Nintendo) та VR/AR-пристрої. Ця кросплатформеність дозволяє розробникам охоплювати різні ринки та пристрої. Unity пропонує інтуїтивний інтерфейс, який забезпечує зручний доступ до всіх основних інструментів розробки. Основною мовою програмування в Unity є C#, що робить його доступним для розробників різного рівня досвіду. Крім того, середовище підтримує візуальний скриптинг через систему Bolt.

Unity також має потужну систему фізики (2D та 3D), інтегровану систему анімації Mecanim, Scriptable Render Pipeline (SRP), а також вбудований магазин активів (Asset Store). Окрім цього, Unity підтримує інтеграцію з багатьма сторонніми інструментами, такими як Visual Studio, Plastic SCM та інші [9].

2.2.1 Опис рушія Unity3D

Unity3D забезпечує потужну основу для створення кросплатформенних ігор завдяки ряду функціональних можливостей.

Кросплатформеність Unity підтримує розробку ігор для ПК (Windows, macOS, Linux), мобільних пристроїв (Android, iOS), консолей (Xbox, PlayStation, Nintendo) та VR/AR-пристроїв. Ця універсальність дозволяє розробникам створювати ігри, які можуть працювати на різних платформах з мінімальними змінами.

Основною мовою програмування в Unity є C#, що робить його доступним для розробників різного рівня досвіду. Рушій також підтримує візуальний скриптинг через систему Bolt.

Рендеринг Unity3D пропонує два основних рендерери:

— вбудований рендерер для базових проєктів;

— Scriptable Render Pipeline (SRP), який забезпечує високий рівень гнучкості складних проєктів.

Рушій підтримує як 2D, так і 3D фізику завдяки інтеграції з технологіями PhysX та Box2D. Це дозволяє розробникам створювати реалістичні фізичні симуляції та взаємодії між об'єктами.

Інтегрована система анімації Mecanim підтримує як скелетну анімацію, так і Blend Tree для плавного переходу між анімаційними станами.

Інтеграція з іншими інструментами Unity підтримує інтеграцію з Visual Studio, Plastic SCM та іншими сторонніми інструментами [10]. Крім того, вбудований магазин активів (Asset Store) дозволяє розробникам швидко знаходити й використовувати готові матеріали та інструменти.

2.2.2 Інтерфейсне вікно Unity3D

Інтерфейсне вікно Unity3D забезпечує зручний доступ до всіх основних інструментів розробки. Основні компоненти інтерфейсного вікна наведено на рисунку 5:

— сцена (Scene) це головне вікно для створення та редагування ігрових сцен, воно дозволяє переміщувати, масштабувати та обертати об'єкти, а також керувати камерами та освітленням, перехід між 2D та 3D режимами забезпечує додаткову гнучкість;

— ієрархія (Hierarchy) це список всіх об'єктів на сцені, тут розробники можуть організовувати ієрархію об'єктів, групувати їх у вкладені структури та швидко переміщатися між об'єктами сцени;

— інспектор (Inspector) це вікно, яке дозволяє редагувати властивості обраного об'єкта, залежно від типу об'єкта, інспектор відображає різні параметри, включаючи компоненти об'єкта, скрипти та анімацію;

— проєкт (Project) це структура файлів та папок проєкту, розробники можуть легко керувати ресурсами проєкту, імпортувати нові активи та організовувати їх у логічні групи;

— консоль (Console) це виведення повідомлень, попереджень та помилок під час розробки, вікно консолі дозволяє відслідковувати проблеми в коді та налагоджувати їх.

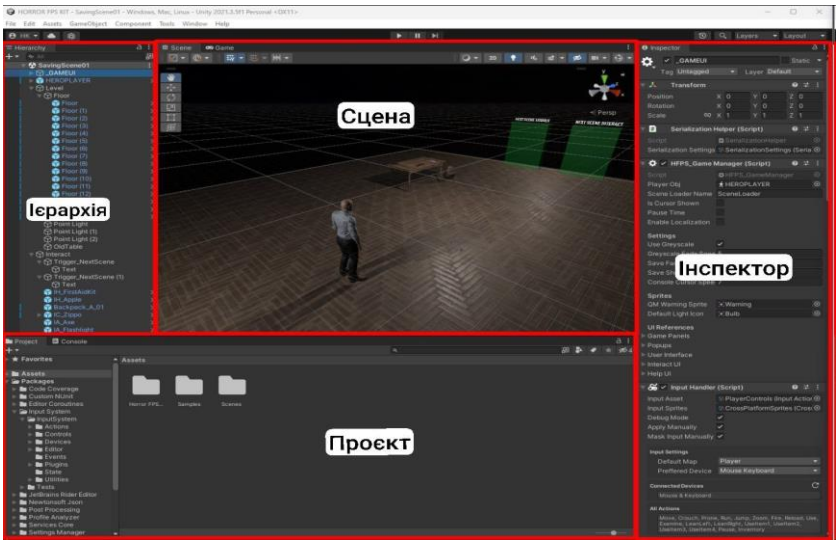


Рисунок 5 — Інтерфейсне вікно Unity3D

2.2.3 Інтеграція між Visual Studio і Unity

Visual Studio є головним середовищем розробки коду для Unity. Інтеграція між Visual Studio та Unity забезпечує такі можливості:

— налагодження (debugging) дозволяє ставити точки зупину, відстежувати значення змінних та виконання коду в реальному часі;

— підсвічування синтаксису C# та автодоповнення, підтримка підсвічування синтаксису C# та автодоповнення, включаючи стандартні бібліотеки Unity;

— інтеграція з Unity Editor, можливість швидко перемикатися між Unity Editor та Visual Studio для оперативного внесення змін до проєкту.

У порівнянні з іншими ігровими рушіями Unity має свої переваги та недоліки. Переваги Unity забезпечують йому інтуїтивний інтерфейс, широкий спектр можливостей та активну спільноту розробників. Рушій підтримує понад 25 платформ та має вбудований магазин активів (Asset Store). Однак Unity має свої недоліки. Вбудована система рендерингу поступається за якістю іншим рушіями, а також має обмежену оптимізацію для високоякісних AAA-проєктів.

Порівняння з Unreal Engine: Unreal Engine пропонує кращу графіку, але є складнішим в освоєнні. Unity має перевагу в легкості створення прототипів та невеликих проєктів.

Порівняння з Godot Engine: Godot є повністю безкоштовним, але має менше можливостей у 3D графіці. Unity пропонує більш інтуїтивний інтерфейс та ширший набір інструментів.

Unity3D є одним із найбільш універсальних та доступних інструментів для кросплатформенної розробки. Рушій забезпечує інтуїтивний інтерфейс, широкий спектр можливостей та активну спільноту розробників. У порівнянні з аналогами, Unity має свої переваги та недоліки, але залишається привабливим вибором для розробників різного рівня.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ КРОСПЛАТФОРМЕННОГО ПРОЄКТУ НА UNITY3D

3.1 Розробка концепції та прототипу гри

Концепція гри мультиплеєрна головоломка з внутрішньою валютою та конкуренцією між гравцями, передбачає створення кросплатформенного проєкту, де гравці повинні розв'язувати головоломки, заробляти внутрішню валюту та змагатися між собою за допомогою мультиплеєрного режиму. Основні особливості гри:

— мультиплеєрність, гравці можуть об'єднуватися або змагатися між собою у режимі реального часу за допомогою технології Photon;

— головоломки, гра складається з низки головоломок різних типів, які стають дедалі складнішими з кожним новим рівнем;

— внутрішня валюта, гравці можуть заробляти віртуальні гроші, які потім використовуються для придбання предметів або підвищення рейтингу;

— конкуренція, можливість змагатися з іншими гравцями через таблиці лідерів або безпосередньо в мультиплеєрному режимі.

Сучасні комп'ютерні ігри потребують ретельної розробки та адаптації до різних платформ і операційних систем. Це вимагає врахування багатьох факторів, включаючи продуктивність, сумісність з апаратним забезпеченням, наявність інструментів розробки та вподобання кінцевих користувачів [11]. У цьому контексті розробка прототипу гри стає складним, але захоплюючим завданням. Представлений прототип гри охоплює всі ключові аспекти, необхідні для створення багатокористувацької гри з інтерактивними можливостями, що забезпечують захоплюючий ігровий досвід. Схема прототипу гри зображена на рисунку 6.

Прототип гри починається з лобі, яке є центральною частиною, де гравці можуть підключитися до матчу або запросити інших приєднатися до них. Головною особливістю лобі є те, що лідер лобі (гравець, який створив лобі або якого вибрали інші гравці) має повний контроль над запуском гри. Він може вирішити, коли розпочати матч, даючи гравцям час підготуватися та налаштуватися.

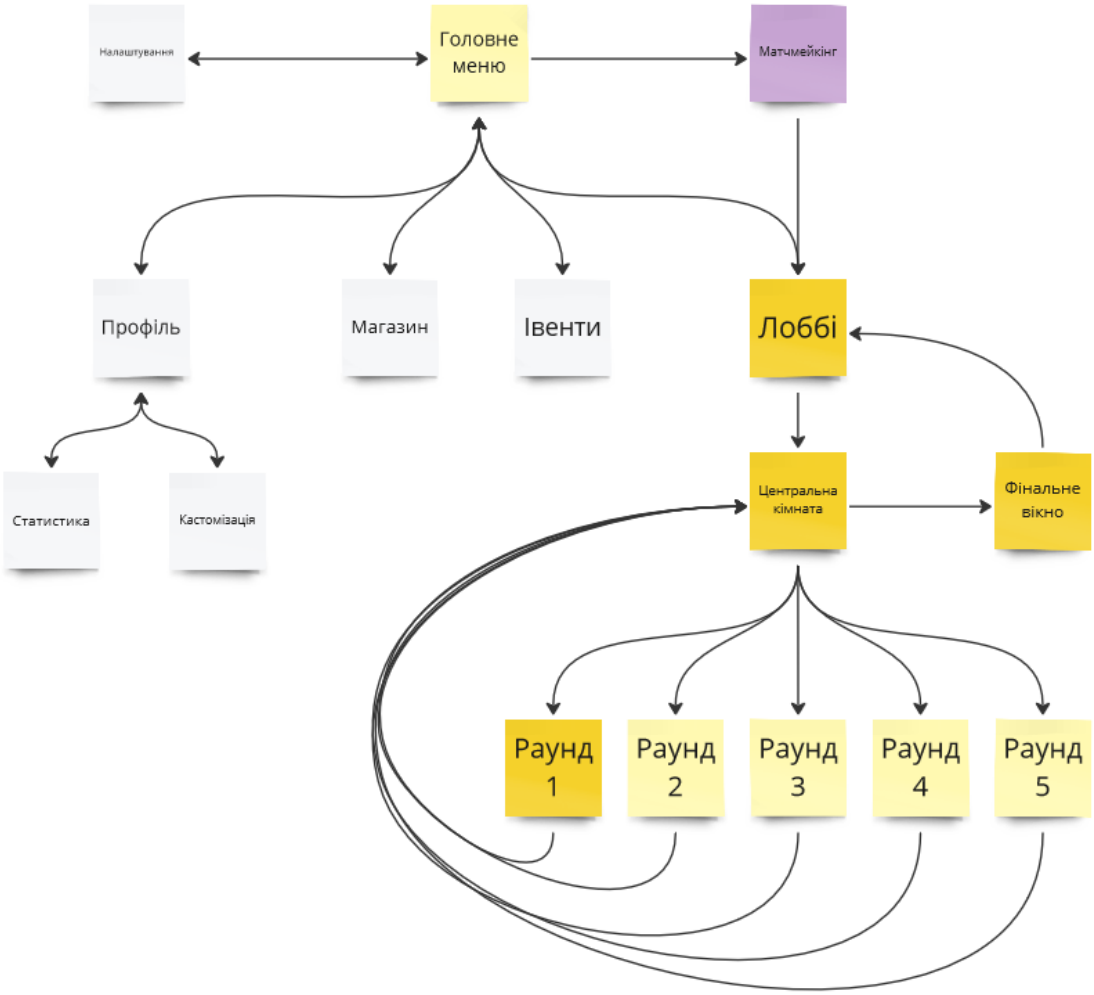


Рисунок 6 — Схема прототипу гри

У лобі також може бути доступний інвентар для кожного гравця, де вони можуть змінювати зовнішній вигляд своїх персонажів, обирати екіпіровку та налаштовувати різні аспекти свого персонажа. Це додає глибину гри та дозволяє гравцям відчувати себе унікальними. Кожен гравець має доступ до персонального магазину, де він може купувати різноманітні предмети та бонуси за зароблені у грі кошти, що дозволяє покращувати свої можливості та робити гру ще цікавішою.

На кожному рівні гравці можуть зустрічати спеціальні бонуси. Щоб активувати ці бонуси, необхідно прийняти певні рішення, наприклад, вирішити головоломку або виконати завдання. Активовані бонуси можуть відкрити двері на наступний рівень.

Для початку рівня всі гравці мають одночасно активувати двері, що може вимагати співпраці та координації між гравцями, додаючи елемент командної гри.

Перед початком кожного рівня гравці бачать інформаційні екрани, які показують назву рівня та інші ключові деталі, допомагаючи їм краще орієнтуватися та підготуватися до майбутніх завдань. Після запуску матчу гравцями з лобі, вони потрапляють на різні рівні. На рівнях неможливо померти, що дозволяє гравцям зосередитися на виконанні завдань. Умови рівнів можуть змінюватися, але основною метою може бути, наприклад, досягнення точки Б з точки А. Після завершення всіх раундів гравці переходять до фінального вікна.

Після завершення всіх раундів відображається фінальне вікно, де показується переможець або переможці, сума зароблених грошей та різні досягнення за рівень. Також відображаються результати кожного гравця окремо, надаючи можливість оцінити свої досягнення та побачити, як вони виступили у порівнянні з іншими. Після перегляду фінального вікна гравці повертаються назад у лобі, де вони можуть підготуватися до наступного матчу, придбати нові предмети в магазині або змінити налаштування свого персонажа.Цей прототип гри забезпечує захоплюючий ігровий досвід з елементами співпраці, індивідуальної кастомізації та цікавою механікою прогресії через рівні.

3.2 Реалізація мультиплеєра через Photon

Photon є популярною платформою для розробки мультиплеєрних ігор завдяки своїй простоті у використанні та широким можливостям. Вона пропонує готові рішення для обробки мережевої взаємодії, синхронізації станів гравців і управління лобі, що значно спрощує процес розробки мультиплеєрних функцій [12].

Перший крок у реалізації мультиплеєра через Photon полягає у встановленні підключення до сервера Photon рисунок 7. Це досягається шляхом ініціалізації Photon Network в коді гри та встановлення з'єднання з сервером.

Після підключення до сервера гравці переходять до лобі, де вони можуть взаємодіяти один з одним, запрошувати друзів або приєднуватися до існуючих ігор. Лідер лобі має можливість розпочати гру, коли всі гравці готові.

```
using Photon.Pun;
using Photon.Realtime;

public class NetworkManager : MonoBehaviourPunCallbacks
{
    void Start()
    {
        PhotonNetwork.ConnectUsingSettings();
    }

    public override void OnConnectedToMaster()
    {
        Debug.Log("Connected to Photon server");
        PhotonNetwork.JoinLobby();
    }
}
```

Рисунок 7 — Підключення до сервера Photon

Photon забезпечує зручні методи для створення, приєднання та управління лобі рисунок 8.

```
public void CreateRoom()
{
    RoomOptions options = new RoomOptions();
    options.MaxPlayers = 4;
    PhotonNetwork.CreateRoom("RoomName", options);
}

public override void OnJoinedRoom()
{
    Debug.Log("Joined room");
    // Додатковий код для налаштування лобі
}
```

Рисунок 8 — Реалізація лобі

Одним з важливих аспектів мультиплеєрної гри є синхронізація станів гравців, що включає їхнє положення, дії та взаємодію з об'єктами. Photon надає зручний механізм для цієї синхронізації через PhotonView компоненти, які передають інформацію про стан об'єктів між клієнтами рисунок 9.

Після того, як всі гравці зібралися в лобі та готові до початку гри, лідер лобі запускає матч. Photon дозволяє керувати процесом запуску та переходу між рівнями, забезпечуючи синхронізацію подій для всіх гравців рисунок 10.

Використання Photon для реалізації мультиплеєра в нашій грі дозволяє

забезпечити стабільну та ефективну взаємодію між гравцями.

```
using Photon.Pun;

public class PlayerController : MonoBehaviourPun, IPunObservable
{
    private Vector3 networkedPosition;

    void Update()
    {
        if (photonView.IsMine)
        {
            // Локальне управління гравцем
        }
        else
        {
            transform.position = Vector3.Lerp(transform.position, networkedPosition, Time.deltaTime * 10);
        }
    }

    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.IsWriting)
        {
            stream.SendNext(transform.position);
        }
        else
        {
            networkedPosition = (Vector3)stream.ReceiveNext();
        }
    }
}
```

Рисунок 9 — Синхронізація станів гравців

```
}
public void StartGame()
{
    if (PhotonNetwork.IsMasterClient)
    {
        PhotonNetwork.LoadLevel("GameLevel");
    }
}
```

Рисунок 10 — Управління рівнями

Простота інтеграції та широкий спектр можливостей Photon робить його ідеальним вибором для розробки багатокористувацьких ігор. Завдяки Photon ми можемо реалізувати такі функції, як підключення до серверів, управління лобі, синхронізація станів гравців та управління матчами, забезпечуючи захоплюючий ігровий досвід для всіх користувачів.

3.3 Розробка основних механік

Основні механіки гри є серцем будь-якого ігрового проекту. Вони визначають, як гравці взаємодіють з ігровим світом, які дії вони можуть виконувати та як розгортається ігровий процес [13]. У цьому розділі буде детально розглянуто розробку основних механік нашої гри, включаючи створення персонажів, управління рухом, взаємодію з об'єктами та виконання завдань.

Для реалізації функцій гравця було створено контролер, що забезпечує можливість ходьби, бігу, стрибків, присідання, упору лежачи та лазіння по сходах. Усі ці дії реалізовані за допомогою анімованого тіла гравця, що відображає відповідні рухи. Звуки кроків генеруються на основі текстур та тегів, що додає реалістичності ігровому процесу. Додатково було реалізовано виявлення нахилу гравця та стін, що дозволяє уникати перешкод і адаптувати рух. Гравець може використовувати зброю, наприклад, сокиру. Реалізовано перемикання між предметами за допомогою спеціального механізму.

Було впроваджено систему збереження та завантаження, яка включає збереження даних гравця, даних сцени, інвентарю та шифрування для забезпечення безпеки даних. Враховуючи кросплатформенність, гра підтримує введення для ПК, PS4 та XONE. Розширена система меню забезпечує кросплатформенну підтримку. Система інвентарю дозволяє додавати, видаляти, переміщувати, використовувати, викидати, досліджувати предмети та створювати ярлики для швидкого доступу. Система цілей включає можливість додавання нових цілей, завершення існуючих, а також організацію подій. Система сцен керує чергою сцен, забезпечуючи плавні переходи. Система обстеження дозволяє гравцям оглядати об’єкти з можливістю повороту, подвійного огляду та перегляду документів. Система перетягування жорстких тіл дозволяє обертати, масштабувати та кидати об’єкти [14]. Менеджер взаємодії включає функції підбирання предметів та виведення повідомлень. Система плаваючих іконок забезпечує візуалізацію важливих об’єктів.

Було створено користувацькі об'єкти захоплення, включаючи легкі предмети, такі як ліхтарик, ліхтарик на петлях та свічка. Інвентар рюкзака розширюється з

можливістю підбирання предметів.

Розроблено динамічні об'єкти, включаючи двері, шухляди, важелі, клапани та інші рухомі взаємодії. Було реалізовано різні типи взаємодії, такі як миша, анімація, заблоковані або застряглі об'єкти. Додатково були створені об'єкти з блокуванням клавіатури, блокуванням картки та навісним замком.Було реалізовано можливість переприв'язування клавіш під час гри. Інтегровано II зомбі, який забезпечує поведінкові моделі, такі як сон, блукання, крик, агонія, голод, напад та притягнення.

Створено попередній завантажувач сцени, який дозволяє змінювати фон та виводити підказки.

Реалізовано переляк за допомогою анімацій, ефектів переляку та налаштовуваного переляканого дихання. Розроблено меню інтерфейсу, включаючи головне меню, меню паузи та меню завантаження. Впроваджено систему сповіщень, що включає підказки та повідомлення.

Створено реалістичний VHS-плеєр та систему відеоспостереження, які додають атмосферності. Реалізовано інтерактивне світло, що дозволяє керувати лампами, перемикачами та анімаціями. Плавучість на воді та атмосферні зони додають реалістичності ігровому світу.

3.4 Налаштування проєкту

Проєкт було налаштовано з імпортом необхідних ресурсів у порожній проект та імпортом усіх тегів та шарів. Було забезпечено імпорт системи вводу, постобробки та Timeline з Менеджера пакетів. Налаштування було проведено шляхом встановлення параметра Активна обробка вводу на значення Пакунок системи вводу та активації InputSystem.inputsettings.

Було створено головне меню гри, що включає об'єкт _MAINMENU, доданий до сцени. Додатково було додано об'єкт Camera для забезпечення візуального інтерфейсу рисунок 11.

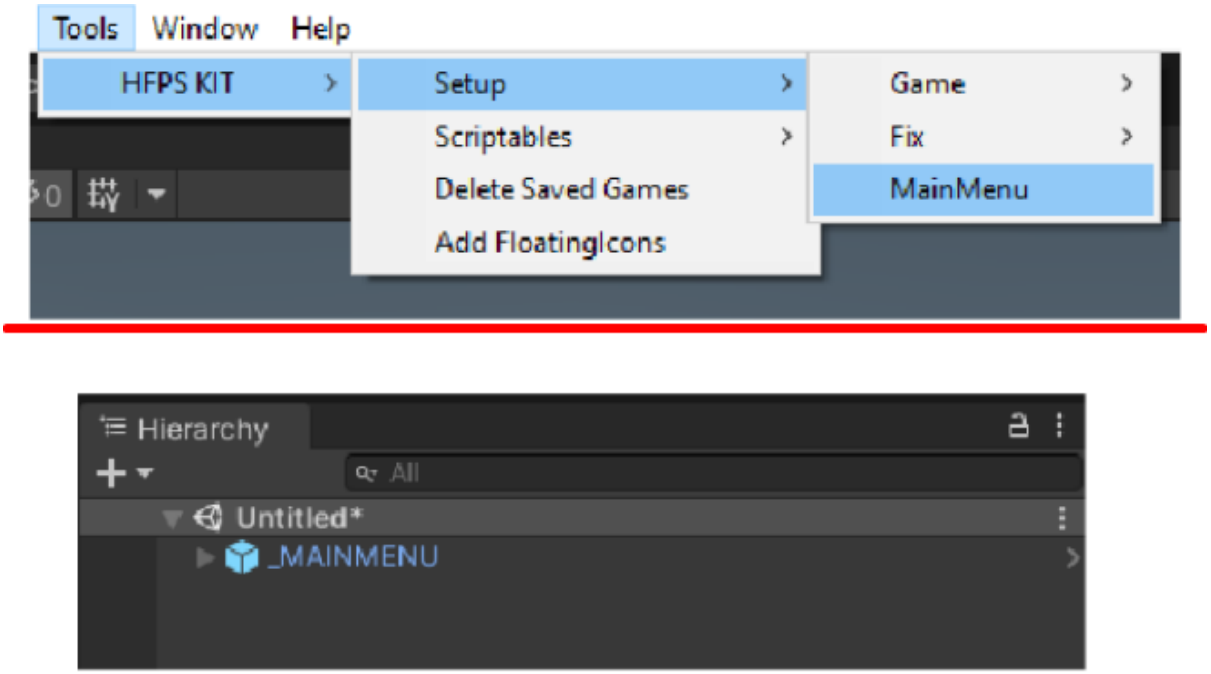


Рисунок 11 — Створення меню гри

Було налаштовано ігрову сцену, включаючи об'єкти _GAMEUI та HEROPLAYER/FPSPLAYER, які були додані до сцени для забезпечення основного ігрового процесу на рисунку 12.

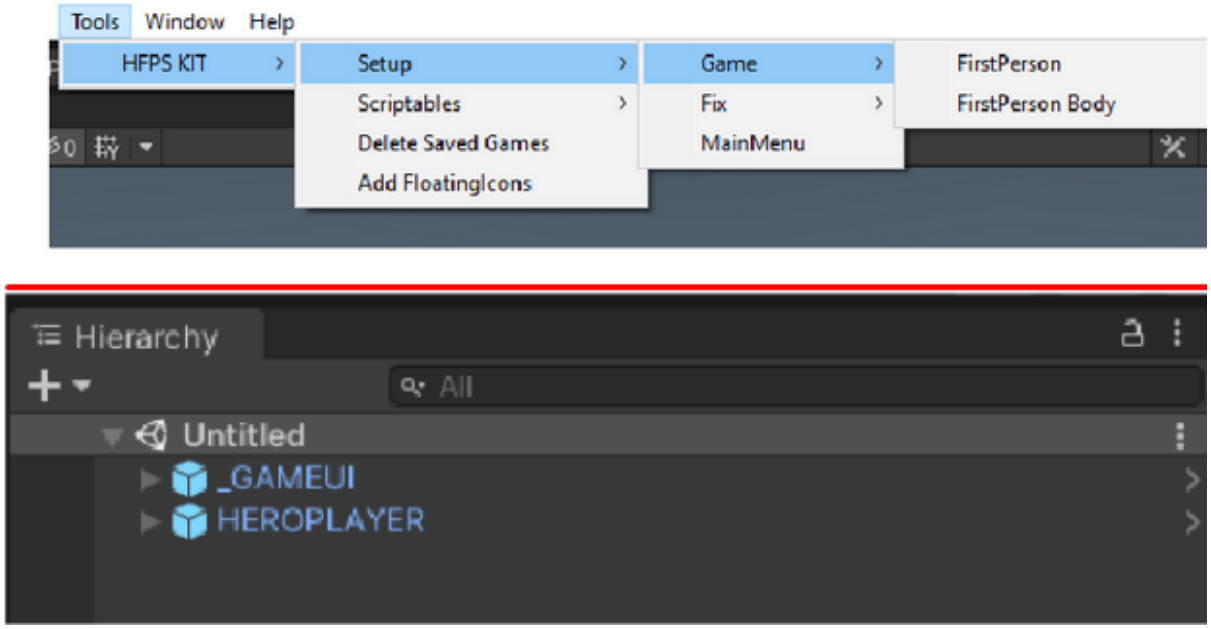


Рисунок 12 — Налаштування головної сцени

Було розроблено нові вхідні дані для гри шляхом налаштування схеми керування та визначення типів дій та шляхів прив'язки.

Це забезпечило інтерактивний та зручний інтерфейс для гравців:

— відкрили вікно ресурсу PlayerControls, рисунок 13;

— натиснули кнопку +, щоб додати нову дію;

— вибрали схему керування на верхній панелі;

— після натискання на дію вибираємо Тип дії;

— вибираємо шлях прив'язки, який змінюватиметься залежно від типу дії;

— натискаємо Зберегти актив, щоб зберегти всі зміни;

— після додавання нової дії використовуємо її за допомогою функцій обробника вводу (рисунок 14 та рисунок 15).

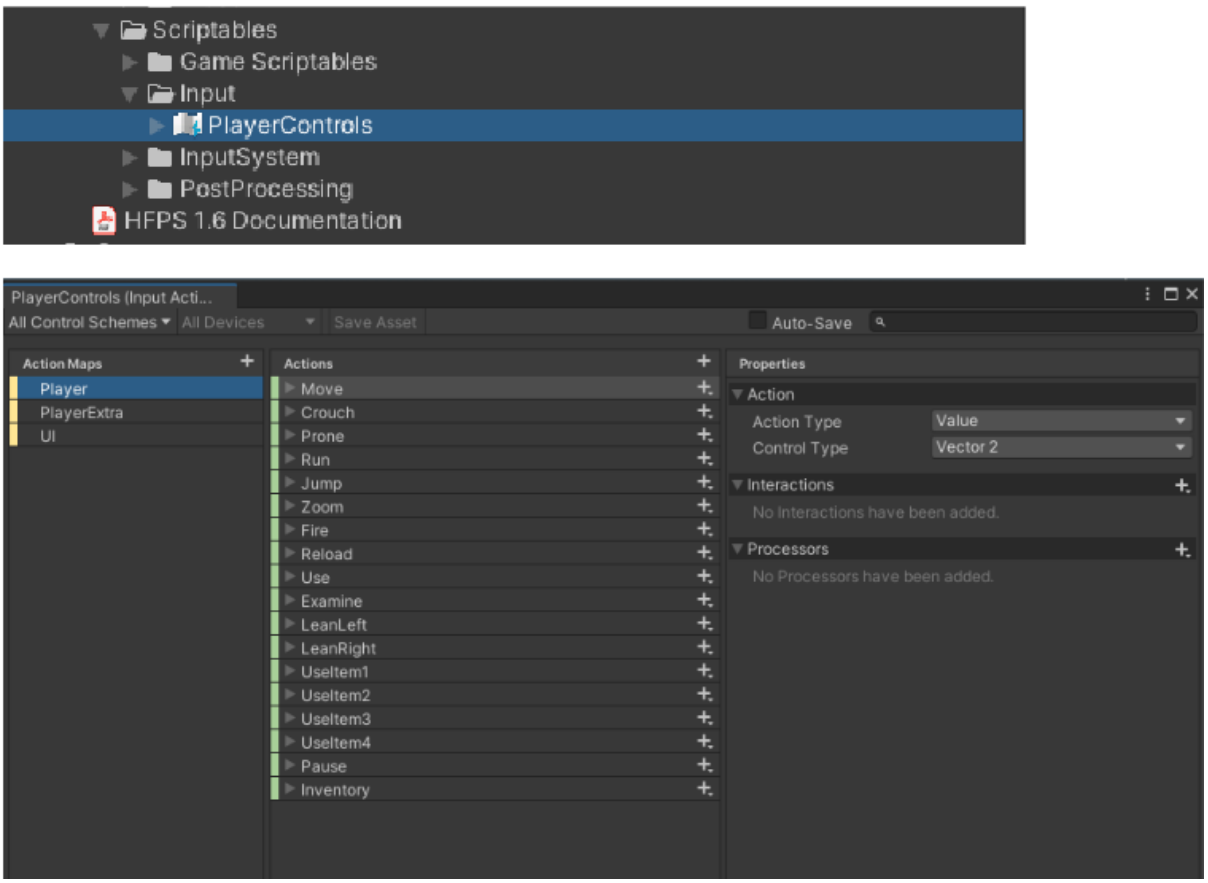


Рисунок 13 — Вікно ресурсу PlayerControls

Якщо ми створюємо новий актив дії входу, призначте його обробнику входу.

```
if (InputHandler.ReadButtonOnce(this, ActionName))
 { Debug.Log($"Дія {ActionName} натиснута один раз!");}
 Переміщення Vector2;
```

if ((movement = InputHandler.ReadInput<Vector2>(ActionName)) != null)
{ Debug.Log($"Дія {ActionName} оновлює {movement}!"); }

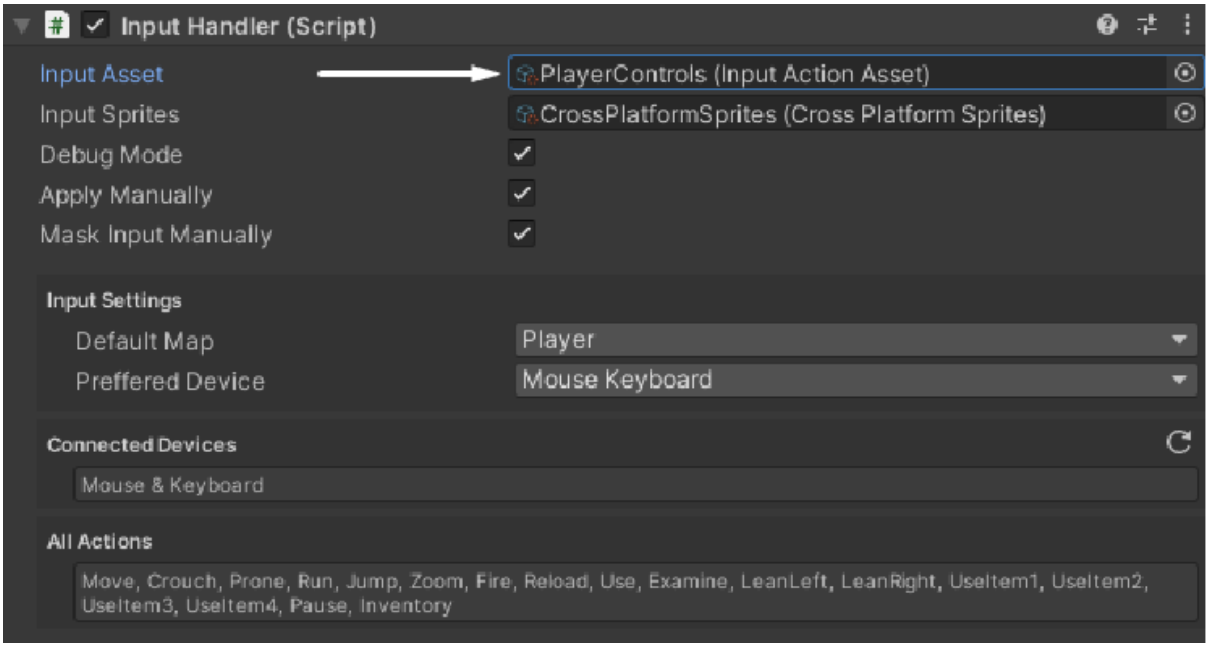


Рисунок 14 — Створення нового активу дії

```
1.  if (InputHandler.ReadButtonOnce(this, ActionName))
2.  {
3.      Debug.Log($"Action {ActionName} is pressed once!");
4.  }
```

```
1.  Vector2 movement;
2.  if ((movement = InputHandler.ReadInput<Vector2>(ActionName)) != null)
3.  {
4.      Debug.Log($"Action {ActionName} is updating {movement}!");
5.  }
```

Рисунок 15 — Функція обробника

Використовуючи функцію ReadInput<T>, звернемо увагу на використання типу повернення, що належить до дії, визначеної в Action Type.

Було впроваджено контролер налаштувань, який надає методи для визначення нових налаштувань гри, залежно від поточної платформи. Було забезпечено можливість дізнаватися про зміни параметрів за допомогою події OnOptionsUpdated.

OptionsController.OnOptionsUpdated += OnOptionsUpdated;

Після додавання функції для події ви можете використовувати функцію GetOptionValue() щоб отримати значення опції рисунок 16.

```
1.  private void OnOptionsUpdated()
2.  {
3.      if (OptionsController.GetOptionValue("KEY", out float value))
4.      {
5.          //your code goes here
6.      }
7.  }
```

Рисунок 16 — Реалізація функції GetOptionValue()

private void OnOptionsUpdated()
{if (OptionsController.GetOptionValue("KEY", out float value))
{ //your code goes here}}

Було реалізовано менеджер збереження/завантаження, що включає створення ресурсу Налаштування серіалізації та призначення його у сценарії Помічника серіалізації. Було додано інтерфейс ISaveable до об'єктів для створення та завантаження збережених даних гри. Було створено нові сцени шляхом створення анімацій на монтажному столі та додавання тригера покадрового переходу. Нові Cutscene'и були призначені сценарію Cutscene Manager та налаштовані відповідно до потреб гри. Було реалізовано нові цілі для гри, шляхом створення або відкриття існуючої сцени об'єктів та додавання нових цілей до ресурсу об'єктів сцени. Цілі були призначені у скрипті Objectives Manager, а також створено тригери для їх запуску.

Було реалізовано плаваючі іконки для визначених об'єктів, що дозволяє відображати важливі об'єкти у грі. Це забезпечило кращу видимість та взаємодію з ігровим світом рисунок 17.

В процесі розробки гри було створено об'єкт бази даних інвентарю, що дозволяє керувати всіма елементами, доступними гравцеві. Для цього було відкрито або створено новий об'єкт бази даних інвентаризації. Доступ до управління елементами бази даних здійснювався через вікно бази даних запасів, яке відкрилося після подвійного клацання на активі або натискання кнопки "Відкрити редактор бази даних

запасів". У цьому вікні ми маємо змогу додавати, видаляти та редагувати елементи бази даних (рисунок 18).

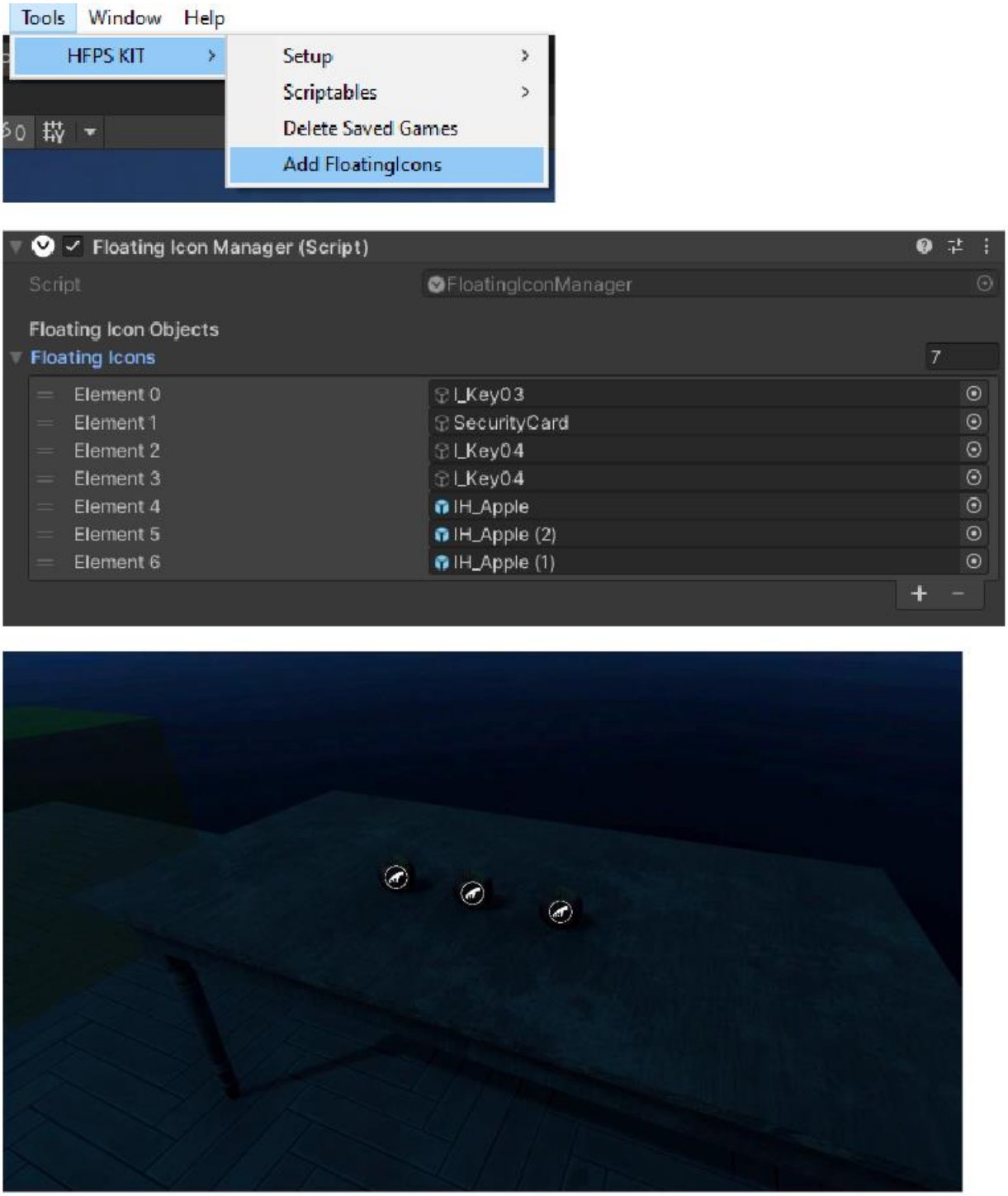


Рисунок 17 — Додавання нових плаваючих об'єктів

Після створення бази даних інвентарю її було призначено у головному скрипті інвентаризації, що знаходиться в об'єкті _GAMEUI.

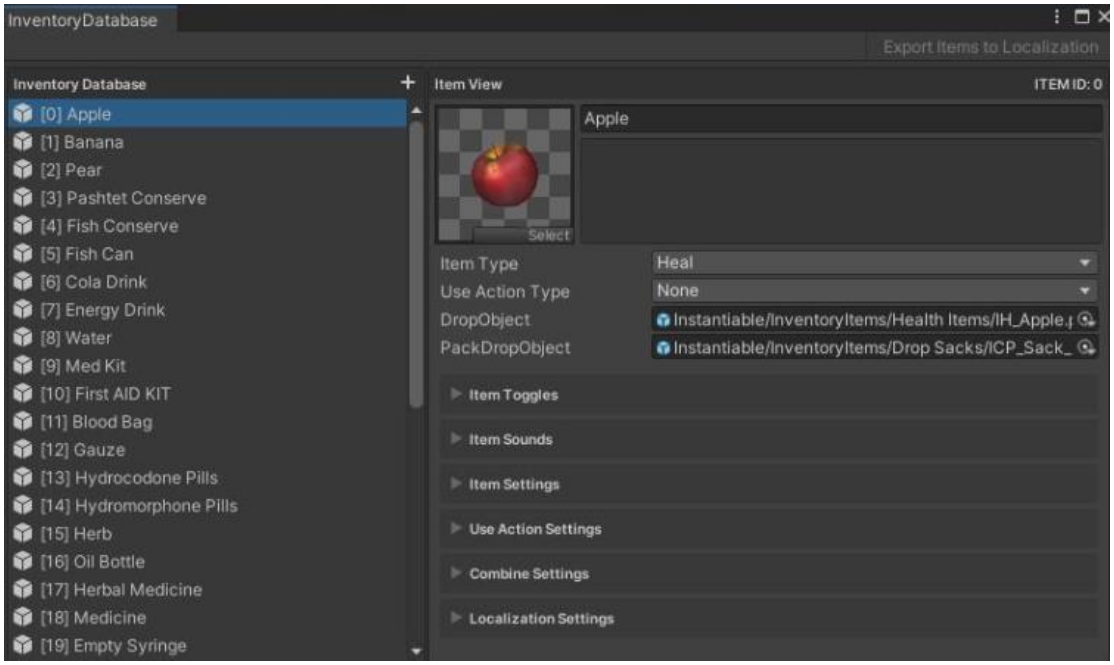


Рисунок 18 — Вікно бази даних інвентаря

Це дозволило інтегрувати інвентар у загальну систему гри та забезпечити його коректне функціонування під час ігрового процесу.

Для забезпечення можливості взаємодії з новоствореними предметами до об'єктів було додано скрипт InteractiveItem.cs. У браузері предметів інвентарю було вибрано необхідні предмети, що дозволило гравцям взаємодіяти з ними під час гри. Ідентифікатори товарів були доступні лише для читання і визначалися автоматично скриптом бази даних. Це забезпечило унікальність кожного предмета та його правильне відображення у системі інвентаризації.

При зміні порядку будь-якого об'єкта в базі даних, всі ідентифікатори об'єктів автоматично оновлювалися відповідно до поточного порядку позицій. Це забезпечило збереження цілісності даних та уникнення можливих конфліктів при взаємодії з інвентарем.

В результаті було реалізовано гнучку і функціональну систему інвентарю, що дозволяє гравцям ефективно керувати своїми предметами, взаємодіяти з ними та використовувати їх під час гри (рисунок 19). Ця система є важливим елементом загального ігрового процесу, забезпечуючи додаткову глибину та інтерактивність.

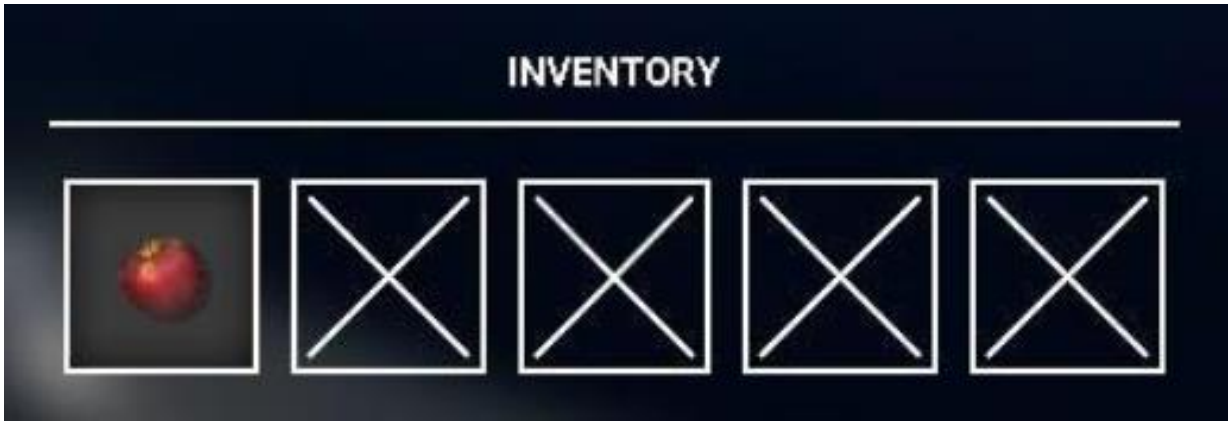


Рисунок 19 — Інвентар

Усі елементи з бази даних інвентарю можна зберігати у спеціальних контейнерах (рисунок 20).

Рисунок 20 — Контейнери інвентарю.

Для цього було реалізовано систему інвентарних контейнерів, яка дозволяє організовувати та зберігати предмети ефективно. Щоб зберегти елемент у контейнері, до нього додавали скрипт InventoryContainer.cs, що забезпечувало можливість інтеграції предмета в контейнерну систему.

Окрім звичайних контейнерів, було реалізовано фіксовані контейнери, які можуть містити ті самі елементи, що й інші контейнери. Це досягалося шляхом додавання скрипта InventoryFixedContainer.cs, який дозволяє визначати конкретні набори елементів для зберігання в таких контейнерах рисунок 21.

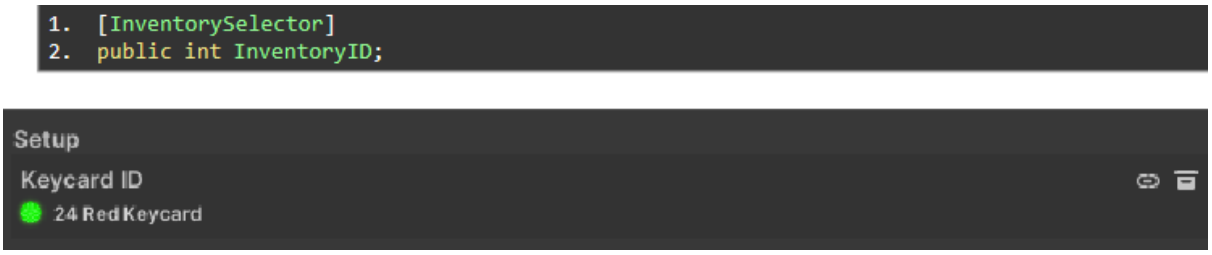


Рисунок 21 — Реалізація скрипта InventoryFixedContainer.cs

Цей підхід забезпечив можливість створення спеціалізованих контейнерів для певних типів предметів, покращуючи організацію інвентарю.

Для зручності роботи з інвентарем було реалізовано можливість перетворення поля типу int на поле селектора інвентарю. Це досягалося за допомогою команди [InventorySelector] у полі int, що дозволило розробникам легко вибирати та управляти елементами інвентарю.

Впровадження системи інвентарних контейнерів значно покращило керування предметами у грі, дозволяючи гравцям ефективно організовувати свій інвентар. Це забезпечило не лише зручність використання, але й додаткову функціональність, що підвищує загальний ігровий досвід.

Було впроваджено функцію комбінування предметів, що дозволяє об'єднувати два різних предмети для отримання нового корисного предмета. Це було реалізовано через визначення параметрів об'єднання в редакторі бази даних Інвентарю рисунок 22.

Було створено динамічні об'єкти з використанням скрипта DynamicObject.cs, що дозволяє визначати інтерактивні двері, шухляди, важелі, клапани та інші рухомі взаємодії. Було визначено типи використання та взаємодії для кожного об'єкта.

Було розроблено об'єкти, які можна перетягувати, шляхом зміни шару об'єкта на Interact та додавання скрипта DraggableObject.cs. Це дозволило забезпечити різні типи взаємодії з об'єктами.

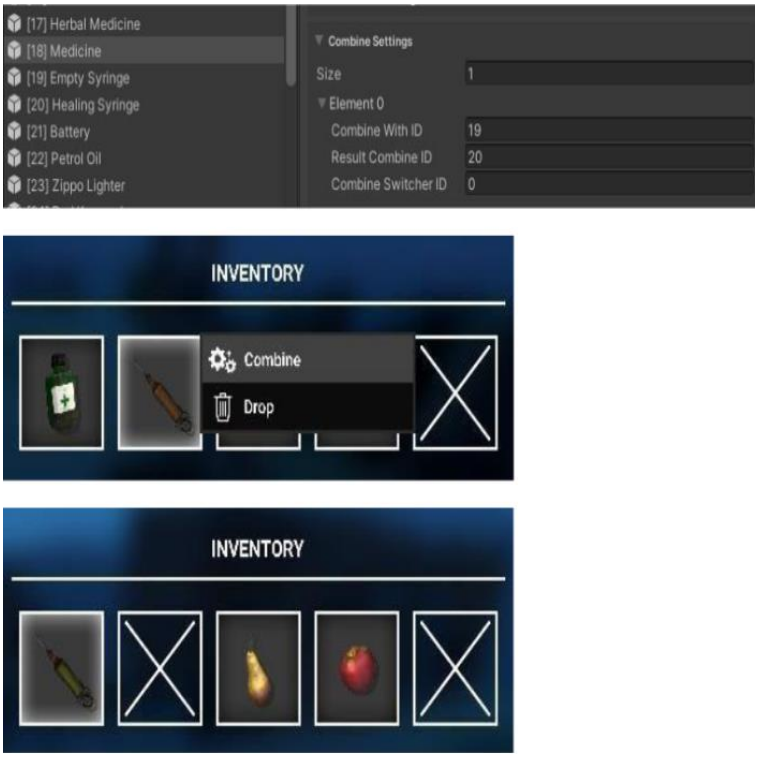


Рисунок 22 — Комбіновані предмети

Було створено нові зомбі AI, які включають поведінкові моделі, такі як сон, блукання, крик, агонія, голод, напад та притягнення. Було налаштовано колайдери, анімації та додано необхідні скрипти для забезпечення реалістичної поведінки зомбі у грі.

Було розроблено нову зброю для гравців, яка включає налаштування анімацій, додавання скрипта WeaponController.cs та додавання нової зброї до ItemManager. Це дозволило гравцям використовувати різні види зброї під час гри.

Було налаштовано реалістичний VHS-плеєр, який підтримує відтворення відео у грі. Це включало переміщення об'єктів VHS Player та VHS Monitor на сцену, підключення їх та додавання відеокасет для відтворення.

Було реалізовано можливість додавання нової моделі гравця, що включало

заміну об'єкта BodyRoot, налаштування аніматора та визначення параметрів кроків та анімацій. Це дозволило забезпечити унікальність персонажа кожного гравця.

У розділі 3 було детально розглянуто практичну реалізацію кросплатформенного проєкту на Unity3D, який охоплює розробку концепції гри, впровадження мультиплеєрного режиму, створення основних ігрових механік та налаштування проєкту.

Розробка концепції гри з акцентом на мультиплеєрні головоломки, внутрішню валюту та конкуренцію між гравцями дозволила створити захоплюючий ігровий досвід. Прототип гри включає лобі, де гравці можуть взаємодіяти, налаштовувати свої персонажі, вирішувати головоломки та заробляти внутрішню валюту, що сприяє співпраці та змаганню між гравцями.

Використання Photon для реалізації мультиплеєрних функцій значно спростило процес створення мережевої взаємодії, забезпечуючи стабільне та ефективне підключення до серверів, управління лобі та синхронізацію станів гравців. Це дозволило реалізувати багатокористувацькі ігри з високою якістю мережевої взаємодії.

Основні механіки гри були реалізовані з акцентом на інтерактивність та реалістичність, включаючи можливість виконання різних дій, використання зброї та ефективне керування інвентарем. Реалізація системи цілей, сцен та динамічних об'єктів додала глибини ігровому процесу.

Проєкт було налаштовано з урахуванням кросплатформенності, що забезпечило підтримку різних пристроїв, включаючи ПК, PS4 та XONE. Було створено зручний інтерфейс для гравців, систему збереження та завантаження, а також інтегровано різні інструменти та ресурси для забезпечення стабільного та захоплюючого ігрового досвіду.

Загалом, проведена робота показала, що використання Unity3D та Photon дозволяє створювати високоякісні кросплатформенні ігри, забезпечуючи захоплюючий ігровий процес, ефективну мережеву взаємодію та підтримку різних платформ.

4 ТЕСТУВАННЯ КРОСПЛАТФОРМЕННОГО ПРОЄКТУ

4.1 Підготовка до тестування

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, особливо кросплатформенних ігор, де забезпечення стабільної та оптимальної роботи на різних пристроях і операційних системах стає ключовим завданням [15]. У цьому розділі ми детально розглянемо процес тестування проєкту "Ігрові механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D", включаючи використані методи, інструменти та результати.

4.1.1 Визначення цілей тестування

Основні цілі тестування нашого проєкту включають:

— забезпечення стабільної роботи гри на різних платформах (Windows, macOS, Linux, Android, iOS);

— перевірка правильності роботи ігрових механік на різних пристроях;

— оптимізація продуктивності гри для забезпечення плавного геймплею;

— виявлення та виправлення помилок, які можуть вплинути на користувацький досвід.

4.1.2 Вибір інструментів для тестування

Для тестування нашого кросплатформенного проєкту ми використовували наступні інструменти:

— Unity Test Framework для юніт-тестування та інтеграційного тестування;

— Unity Profiler для аналізу продуктивності гри;

— Automated Test Scripts для автоматизованого тестування основних функцій гри;

— Cloud Build Services для тестування гри на різних пристроях і платформах.

4.2 Методи тестування

У процесі розробки ігор важливою складовою є забезпечення якості кінцевого

продукту. Для досягнення цього необхідно використовувати різноманітні методи тестування, які дозволяють виявити та усунути помилки, а також забезпечити стабільність та продуктивність гри на різних платформах [16]. У цьому розділі буде розглянуто основні методи тестування, які застосовуються у розробці ігор.

4.2.1 Юніт-тестування

Юніт-тестування проводилося для перевірки окремих компонентів та функцій гри. Ми використовували Unity Test Framework для написання тестів, що перевіряли коректність роботи основних ігрових механік, таких як пересування персонажа, взаємодія з об'єктами та реакція на дії гравця.

Приклад юніт-тесту:

```
[Test]
public void PlayerMovementTest()
{// Arrange
var player = new GameObject().AddComponent<Player>();
var initialPosition = player.transform.position;
// Act
 player.Move(Vector3.right);
 // Assert
Assert.AreEqual(initialPosition + Vector3.right, player.transform.position);}
```

4.2.2 Інтеграційне тестування

Інтеграційне тестування проводилося для перевірки взаємодії між різними модулями гри. Ми тестували, як ігрові механіки працюють разом, наприклад, взаємодія між фізикою та анімацією, а також мультиплеєрні функції за допомогою Photon.

Приклад інтеграційного тесту:

```
[Test]
public void PlayerAndObjectInteractionTest()
```

```
{// Arrange
var player = new GameObject().AddComponent<Player>();
 var interactableObject = new GameObject().AddComponent<InteractableObject>();
// Act
player.InteractWithObject(interactableObject);
// Assert
Assert.IsTrue(interactableObject.HasBeenInteractedWith);}
```

4.3.3 Функціональне тестування

Функціональне тестування перевіряло відповідність гри вимогам та специфікаціям. Ми перевіряли, як ігрові механіки адаптовані до різних платформ, зокрема, сенсорний екран для мобільних пристроїв, геймпади для консолей та клавіатура з мишею для ПК.

4.3.4 Автоматизоване тестування

Автоматизоване тестування дозволило скоротити час на перевірку гри та забезпечити високу якість продукту. Ми використовували автоматизовані скрипти для перевірки основних функцій гри та інтегрували їх у процес CI/CD.

Приклад автоматизованого тесту:

```
[Test]
public void CoinCollectionTest()
{// Arrange
 var player = new GameObject().AddComponent<Player>();
var coin = new GameObject().AddComponent<Coin>();
// Act
player.CollectCoin(coin);
// Assert
Assert.AreEqual(1, player.Coins);}
```

4.4 Оптимізація та профілювання

Оптимізація ресурсів є ключовим аспектом розробки високоякісних ігор, особливо в контексті кросплатформених проектів. Вона полягає у максимально ефективному використанні апаратних можливостей різних пристроїв для забезпечення стабільної та продуктивної роботи гри [17]. Недостатня оптимізація може призвести до проблем з продуктивністю, таких як низька частота кадрів, довгий час завантаження та перегрівання пристроїв, що негативно впливатиме на користувацький досвід. Особливої уваги буде приділено оптимізації ресурсів у середовищі Unity3D, яка є однією з найпоширеніших платформ для створення кросплатформених ігор. Ми розглянемо специфічні функції та налаштування Unity3D, які дозволяють оптимізувати використання ресурсів.

4.4.1 Використання Unity Profiler

Для аналізу продуктивності гри на різних платформах ми використовували Unity Profiler. Це дозволило нам виявити вузькі місця в продуктивності, такі як високе споживання пам'яті, навантаження на процесор та GPU, а також тривалість кадру.

4.4.2 Оптимізація ресурсів

На основі даних з Unity Profiler ми проводили оптимізацію ресурсів гри, таких як текстури, моделі та анімації. Це включало зменшення роздільної здатності текстур, використання менш складних моделей та оптимізацію анімаційних циклів.

4.5 Результати тестування

Після проведення всіх видів тестування ми досягли наступних результатів:

— гра стабільно працює на всіх цільових платформах (Windows, macOS, Linux, Android, iOS);

— основні ігрові механіки правильно функціонують на різних пристроях;

— продуктивність гри оптимізована для забезпечення плавного геймплею;

— всі виявлені помилки були виправлені, що значно покращило користувацький досвід.

Тестування є критично важливим етапом у розробці кросплатформенних ігор на Unity3D. Використання різноманітних методів тестування, таких як юніт-тестування, інтеграційне тестування, функціональне та автоматизоване тестування, дозволило нам забезпечити високу якість та стабільність гри на різних платформах. Адаптація ігрових механік до апаратної структури кожної платформи та оптимізація продуктивності є ключовими факторами успішної кросплатформенної розробки.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нової інформаційної технологія ігрової механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D. Особливістю розробки є оптимізації ігрових механік для кросплатформенних ігор, що враховують особливості апаратного забезпечення та забезпечують високу якість ігрового процесу [18].

Аналогом може бути Lethal Company, ціна 50000 $.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 6.

Таблиця 6 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 6

Ринкові переваги					
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у ВПК	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 6

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 7

Таблиця 7 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	4	4
Наявність аналогів на ринку	4	4	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	4	4
Експлуатаційні витрати	4	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	4	4	4
Фахівці з технічної і комерційної реалізації	3	4	4
Фінансування	3	4	4
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	3	4
Супровідна документація	4	3	3
Сума	45	44	45
Середньоарифметична сума балів	(45+44+45) / 3 = 44,67		

За даними таблиці 7 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 8.

Таблиця 8 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, оптимізації ігрових механік для кросплатформенних ігор, що враховують особливості апаратного забезпечення та забезпечують високу якість ігрового процесу. Зростаючий попит на кросплатформенні ігрові рішення та необхідністю оптимізації ігрового процесу для різноманітних апаратних платформ також обумовив високий комерційного потенціалу розроблюваного нового програмного продукту [19].

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \quad , \tag{5.1}$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ — число робочих днів за місяць, 21 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 9.

Таблиця 9 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	41000	1952,38	42	82000,000
Програміст	37000	1761,90	42	74000,000
Всього				156000,00

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 12 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 12\ \% / 100\ \% \qquad (5.2)$$

$$З_д = (156000{,}00 \cdot 12\ \% / 100\ \%) = 18720{,}00\ \text{(грн.)}$$

Нарахування на заробітну плату розробників, згідно діючого законодавства складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$$Н_з = (156000{,}00 + 18720{,}00) \cdot 22\ \% / 100\ \% = 38438{,}40\ \text{(грн.)}$$

Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ — термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 200000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 2 міс.

$$A_{обл} = \frac{200000}{2} \times \frac{2}{12} = 16666{,}67 \ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 9. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів сплачується по підписці, а підписка на них складає Unity Pro License 1500 $ за рік та Microsoft Visual Studio Enterprise 2500 $ за рік, то в перерахунку на 2 робочі місяці використання сума, яка включається в розробку складе ((4000*40/12)*2) приблизно $B_{нем.ак.}$ = 26700 грн.

Таблиця 9 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	200000	2	2,00	16666,667
Офісне обладнання (меблі)	50000	4	2,00	2083,333
Приміщення	1500000	20	2,00	12500,000
Всього				31250,00

Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \qquad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = (5635,47/1000)*1,2 = 6,76 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,3 кВт;

Ф — фактична кількість годин роботи обладнання, годин.

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,3 \cdot 8 \cdot 42 \cdot 6,76 = 613,2672 \text{ (грн.)}$$

Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{\text{ів}}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$I_в = 156000{,}00 * 77\% / 100\% = 120120 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{\text{нзв}}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 156000{,}00 * 122\ \% / 100\ \% = 190320 \text{ (грн.)}$$

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$B_{заг} = 156000,00+18720,00+38438,40+31250,00+26700+613,27+120120+$$
$$+190320 = 582161,67 \text{ грн.}$$

Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta} \text{ (грн),} \qquad (5.8)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 582161,67 / 0,5 = 1164323 \text{ грн.}$$

Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку [20]. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.9)$$

де $\pm\Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 125000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *5000* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 1000 шт., протягом другого року – на 1500 шт., протягом третього року на 2500 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$\Delta П_1$ = (0*5000 + (125000 + 5000)*1000)* 0,8333* 0,15) * (1 - 0,18) = 12812499,488 грн.

$\Delta П_2$ = (0*5000 + (125000 + 5000)*(1000+1500)* 0,8333* 0,15) * (1 - 0,18) = 33312498,668 грн.

$\Delta П_3$ = (0*5000 + (125000 + 5000)*(1000+1500+2500)* 0,8333* 0,15) * (1 - 0,18) = 66624997,335 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 112749995,49 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності. Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

ПП = (12812499,488/(1+0,1)[1])+(33312498,668/(1+0,1)[2])+(66624997,335/
/(1+0,1)[3]) = 11647726,81 + 27530990,63 + 50056346,61 = 89235064,05 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, це можуть бути витрати на підготовку

приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 1164323 = 2328646{,}67 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{5.12}$$

$$E_{абс} = 89235064{,}05 - 2328646{,}67 = 86906417{,}38 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розра-хувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

$T_{ж}$ – життєвий цикл наукової розробки, роки,

$E_{в} = \sqrt[3]{(1 + 86906417{,}38/2328646{,}67} - 1 = 2{,}371.$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

Де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_{в}}, \tag{5.15}$$

$$T_{ок} = 1 / 2{,}371 = 0{,}42 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,42 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 1164323 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу

розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,42 роки.

ВИСНОВКИ

Дана магістерська робота є всебічним дослідженням, що охоплює теоретичні та практичні аспекти розробки ігор для різних платформ. У ході виконання роботи були вирішені основні завдання, пов'язані з адаптацією ігрових механік під особливості різноманітних апаратних структур, що дозволяє досягти високої якості ігрового досвіду на різних пристроях.

Розглянуто та проаналізовано сучасні тенденції у кросплатформенній розробці ігор, що є однією з найбільш актуальних тем у сучасній ігровій індустрії. Виявлено ключові проблеми, з якими стикаються розробники, зокрема фрагментація апаратних та програмних платформ, відмінності у механізмах вводу та сумісність з операційними системами. Ці проблеми значно ускладнюють процес розробки універсальних ігор, які можуть працювати однаково добре на всіх платформах.

Проведено огляд різних ігрових рушіїв, таких як Unity3D, Unreal Engine, Godot Engine та CryEngine, і визначено їх основні переваги та недоліки. Unity3D виявився найбільш оптимальним вибором для кросплатформенної розробки завдяки його універсальності, простоті використання та підтримці широкого спектру платформ. Це забезпечує можливість розробникам створювати ігри, які можуть працювати на різних пристроях з мінімальними змінами коду, що значно знижує витрати на розробку та підтримку.

Розроблено концепцію та прототип мультиплеєрної головоломки, яка включає внутрішню валюту та можливість змагання між гравцями. Використано Photon для реалізації мультиплеєрного режиму, що дозволяє гравцям взаємодіяти в реальному часі. Проведено детальне тестування гри на різних платформах для виявлення та усунення проблем продуктивності. Використано Unity Profiler для оптимізації ресурсів гри, що дозволило забезпечити стабільну роботу на різних апаратних платформах.

Робота підтверджує важливість комплексного підходу до розробки ігор, який включає ретельний аналіз цільових платформ, адаптацію ігрових механік та інтерфейсів, а також забезпечення оптимальної продуктивності та задоволення

користувачів. Комплексний підхід дозволяє враховувати особливості кожної платформи, забезпечуючи високу якість ігрового досвіду незалежно від використовуваного пристрою.

Результати дослідження є важливими для подальшого розвитку індустрії відеоігор, сприяючи створенню більш гнучких і доступних ігрових рішень. Це дозволяє розширити аудиторію користувачів, забезпечуючи доступ до якісних ігор на різних платформах. Завдяки цьому розробники можуть залучати нових гравців, підвищувати рівень задоволення існуючих користувачів та забезпечувати стабільне зростання своїх проектів.

Отже, дана магістерська робота не лише вирішує актуальні проблеми кросплатформенної розробки ігор, але й надає цінні рекомендації та інструменти для подальшого розвитку цієї галузі. Вона є важливим внеском у сучасну ігрову індустрію, сприяючи підвищенню якості та доступності відеоігор на різних платформах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Unity Technologies. Unity User Manual [Electronic resource] – Access mode: https://docs.unity3d.com/Manual/index.html

2. Stroustrup B. The C++ Programming Language, 4th Edition. – Addison-Wesley Professional, 2013. – 1366 p.x

3. Microsoft. Visual Studio Documentation [Electronic resource] – Access mode: https://docs.microsoft.com/en-us/visualstudio/

4. Resig J., Bibeault B., Maras Y. Secrets of the JavaScript Ninja, 2nd Edition. – Manning Publications, 2016. – 368 p.

5. Hevner A., Chatterjee S. Design Science Research in Information Systems. – Springer, 2010. – 320 p.

6. Sutherland J., Schwaber K. The Scrum Guide. – Scrum.org, 2020. – 19 p.

7. Unity Technologies. Unity Performance Optimization Guide [Electronic resource] – Access mode: https://learn.unity.com/tutorial/performance-optimization

8. Gregory J. Game Engine Architecture, Third Edition. – A K Peters/CRC Press, 2018. – 1240 p.

9. Freeman E., Robson E. Head First Design Patterns: A Brain-Friendly Guide. – O'Reilly Media, 2020. – 694 p

10. Shneiderman B., Plaisant C., Cohen M., Jacobs S., Elmqvist N., Diakopoulos N. Designing the User Interface: Strategies for Effective Human-Computer Interaction, 6th Edition. – Pearson, 2016. – 624 p.

11. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

12. Photon Engine. Photon Unity Networking [Electronic resource] – Access mode: https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro

13. Schell J. The Art of Game Design: A Book of Lenses, Third Edition. – A K Peters/CRC Press, 2019. – 610 p.

14. Fowler M. Refactoring: Improving the Design of Existing Code. – Addison-Wesley Professional, 2018. – 448 p.

15. Troelsen A., Japikse P. Pro C# 8 with .NET Core: Foundational Principles and Practices in Programming. – Apress, 2020. – 1376 p.

16. Brooks F. The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition. – Addison-Wesley Professional, 1995. – 322 p.

17. Sommerville I. Software Engineering, 10th Edition. – Pearson, 2015. – 816 p.

18. Berners-Lee T., Fischetti M. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. – HarperOne, 2000. – 248 p.

19. National Institute of Standards and Technology (NIST). Secure Software Development Framework (SSDF) [Electronic resource] – Access mode: https://csrc.nist.gov/publications/detail/white-paper/2020/04/23/secure-software-development-framework-ssdf/final

20. Albahari J., Albahari B. C# 8.0 in a Nutshell: The Definitive Reference. – O'Reilly Media, 2020. – 1088 p.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., професор Азаров О. Д.

«29» лютого 2024 року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

"Ігрові механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D".

08-54.МКР.004.00.000 ПЗ

Науковий керівник: доцент к.т.н.

_____ Городецька О.С.

Виконав: студентка групи КІ-22мз

_____ Колеснікова Н.С.

ВНТУ 2024

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Важливим є актуальність дослідження у напрямку магістерської роботи, яка обумовлена стрімким розвитком технологій у галузі відеоігор та потребою в універсальних рішеннях для розробки ігор, оптимізованих під різноманітні платформи.

1.2 Наказ про затвердження теми МКР по ВНТУ № 81 від 11.03.2024.

2 Мета МКР і призначення розробки

2.1 Мета роботи — полягає у визначенні оптимальних підходів до адаптації ігрових механік для кросплатформенних проектів, розроблених на Unity3D, з метою забезпечення високої якості ігрового досвіду для користувачів різних платформ.

2.2 Призначення розробки — виражається у підвищенні якості розробки кросплатформенних ігрових проектів, зниженні витрат часу та ресурсів на адаптацію ігор для різних платформ, а також у покращенні якості кінцевого продукту для користувачів.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих принципів та технологій кросплатформенної розробки ігор.

3.2 Розробка структури та функціональної схеми кросплатформенного проєкту на Unity3D.

3.3 Написання програмного коду та реалізація мультиплеєрного режиму з використанням Photon.

3.4 Оптимізація ігрового процесу з використанням Unity Profiler.

3.5 Проведення розрахунків для доведення доцільності нової розробки з економічної точки зору.

4 Вимоги до виконання МКР

4.1 Використання сучасних методів розробки ігор та програмних засобів.

4.2 Дотримання високого рівня теоретичного обґрунтування дослідження.

4.3 Практична реалізація розробок, що демонструє застосування теоретичних знань.

4.4 Робота повинна бути виконана самостійно, з дотриманням академічної доброчесності.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ з/п	Назва етапів дипломної магістерської роботи	Початок виконання	Кінець виконання	Очікувані результати	Примітка
1	Огляд і аналіз кросплатформенних ігрових рушіїв	01.03.2024	30.03.2024	Звіт про аналіз ігрових рушіїв	Виконано
2	Вибір технологій розробки ігор	01.04.2024	20.04.2024	Документ з вибором технологій	Виконано
3	Розробка концепції та прототипу гри	21.04.2024	10.05.2024	Концепція та прототип гри	Виконано
4	Реалізація мультиплеєрного режиму	11.05.2024	20.05.2024	Мультиплеєрний режим реалізовано	Виконано
5	Тестування та оптимізація гри	21.05.2024	24.05.2024	Звіт про тестування та оптимізацію	Виконано
6	Розрахунок економічних показників проекту	25.05.2024	30.05.2024	Звіт з економічними показниками	Виконано

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам. Порядок контролю виконання та захисту МКР Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення т правила складання»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт студентами спеціальності 123 «Комп’ютерна інженерія». / Укладачі О.Д. Азаров, О.В. Дудник, С.І. Швець − Вінниця : ВНТУ, 2023. − 57 с.

8.2 Порядок виконання МКР в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Блок-схема лобі

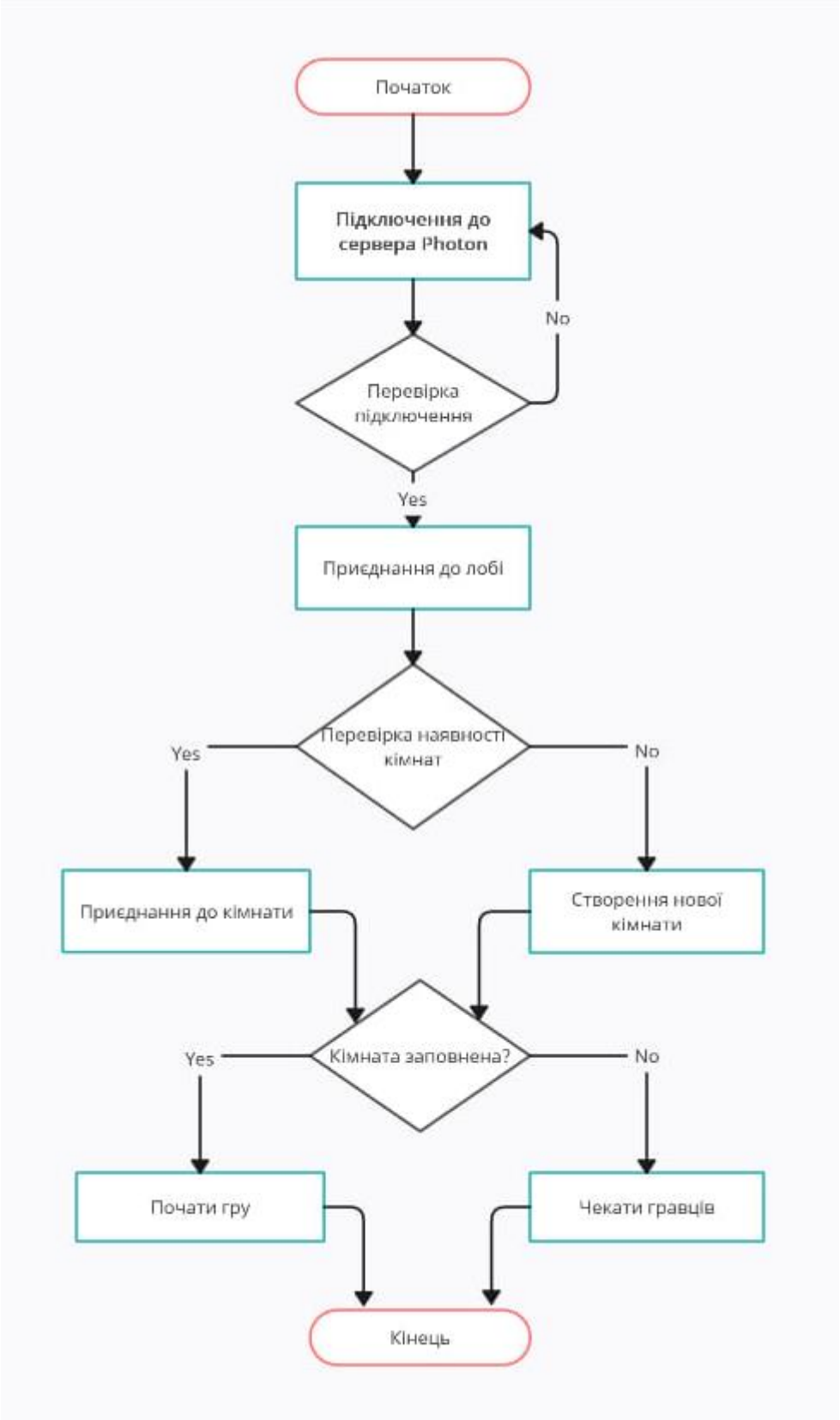


Рисунок Б.1 — Блок - схема лоббі

ДОДАТОК В

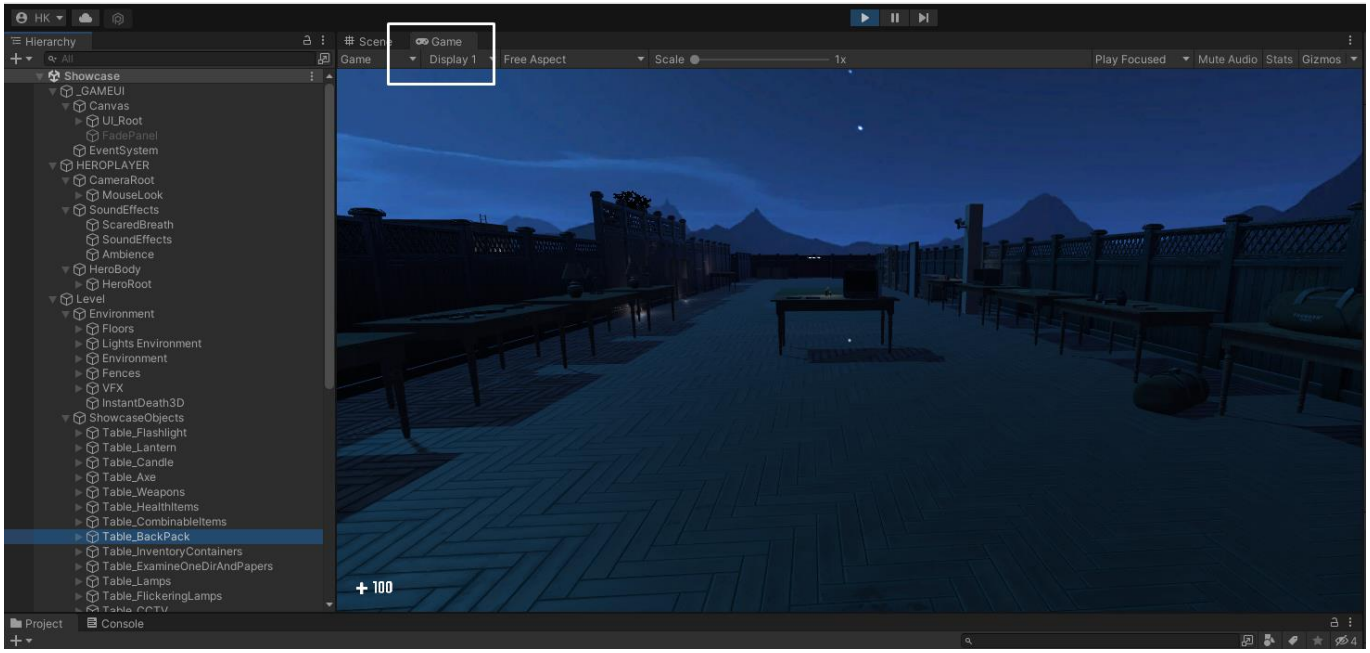
Приклад інвентаря та колекції предметів

Рисунок В.1 — Приклад інвентаря

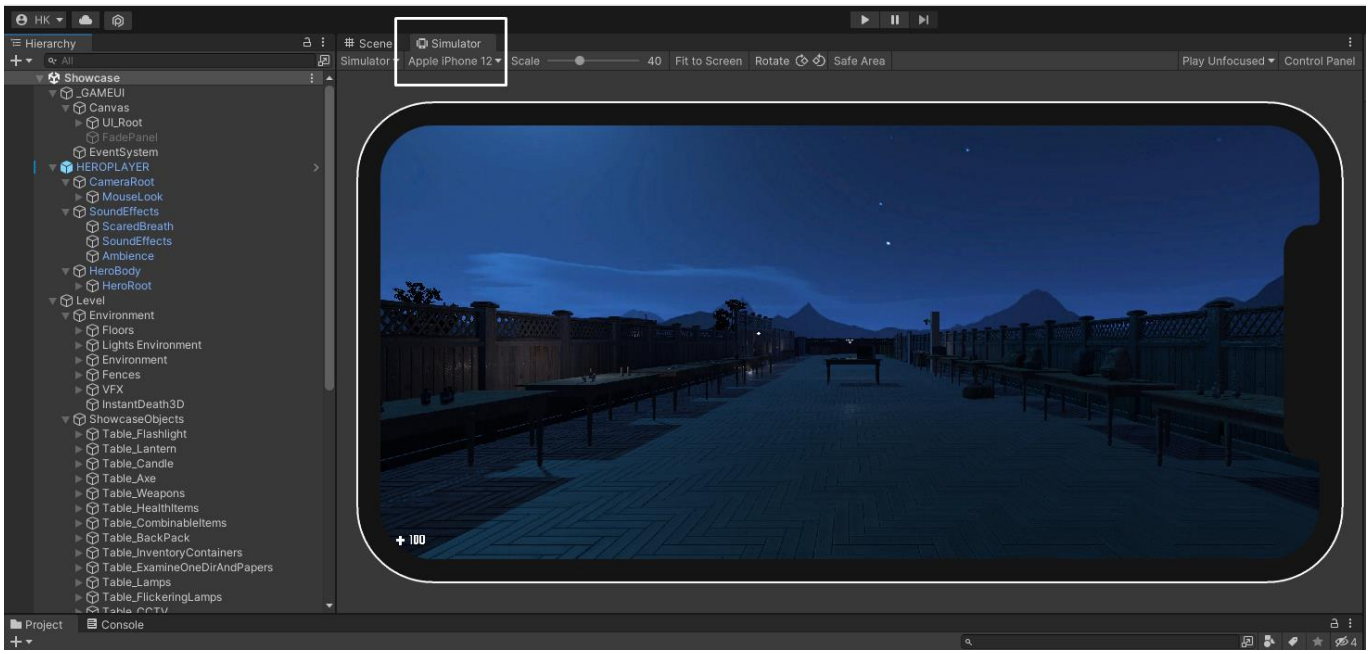
Рисунок В.2 — Приклад колекціонування

ДОДАТОК Г

Демонстрація симулятора Apple iPhone 12



Додаток Г.1 — Звичайне вікно рендеру

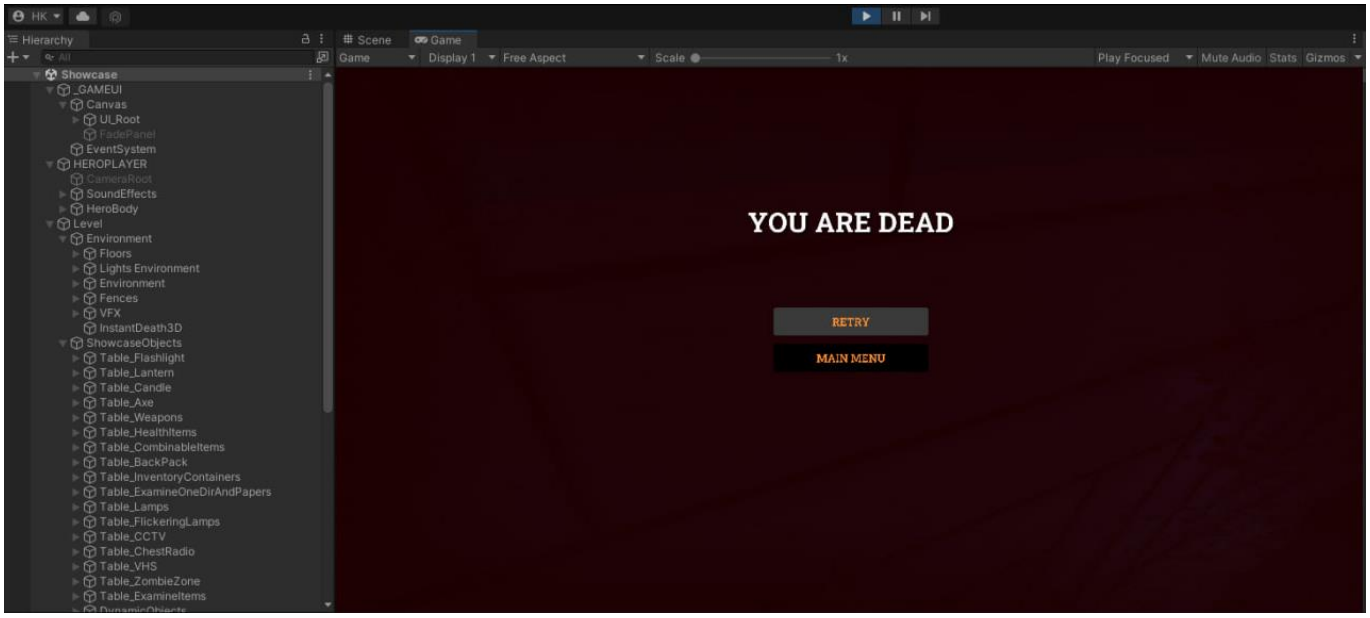


Додаток Г.2 — Симуляція Apple iPhone 12

ДОДАТОК Д

Приклад реалізації тригера

Додаток Д.1 — Розміщення тригеру



Додаток Д.2 — Робота тригеру

ДОДАТОК Е

Лістинг програми

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;

public class PlayerController : MonoBehaviourPunCallbacks
{
    public float speed = 5.0f;
    public float jumpForce = 5.0f;
    public int maxHealth = 100;
    private int currentHealth;
    private Rigidbody2D rb;
    private bool isGrounded;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        currentHealth = maxHealth;
    }

    void Update()
    {
        if (photonView.IsMine)
        {
            Move();
            Jump();
            CheckHealth();
        }
    }
```

```
void Move()
{
    float moveInput = Input.GetAxis("Horizontal");
    rb.velocity = new Vector2(moveInput * speed, rb.velocity.y);
}

void Jump()
{
    if (isGrounded && Input.GetKeyDown(KeyCode.Space))
    {
        rb.velocity = Vector2.up * jumpForce;
    }
}

void CheckHealth()
{
    if (currentHealth <= 0)
    {
        Die();
    }
}

void Die()
{
    PhotonNetwork.Destroy(gameObject);
}

public void TakeDamage(int damage)
{
    currentHealth -= damage;
    Debug.Log("Player health: " + currentHealth);
}
```

```
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Ground"))
        {
            isGrounded = true;
        }
    }

    private void OnCollisionExit2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Ground"))
        {
            isGrounded = false;
        }
    }
}

public class Interactable : MonoBehaviour
{
    private bool isPlayerNearby;

    void Update()
    {
        if (isPlayerNearby && Input.GetKeyDown(KeyCode.E))
        {
            Interact();
        }
    }

    void Interact()
    {
        Debug.Log("Interacted with object!");
```

```
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            isPlayerNearby = true;
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            isPlayerNearby = false;
        }
    }
}

public class GameManager : MonoBehaviourPunCallbacks
{
    public static GameManager instance;
    public int playerScore = 0;

    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(gameObject);
```

```
        }
    }

    public void IncreaseScore(int amount)
    {
        playerScore += amount;
        Debug.Log("Score: " + playerScore);
    }
}

public class Collectible : MonoBehaviour
{
    public int scoreValue = 10;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            GameManager.instance.IncreaseScore(scoreValue);
            Destroy(gameObject);
        }
    }
}

public class Enemy : MonoBehaviour
{
    public float speed = 2.0f;
    public int damage = 10;
    public Transform[] waypoints;
    private int currentWaypointIndex = 0;

    void Update()
    {
```

```
        Move();
    }

    void Move()
    {
        if (waypoints.Length == 0)
            return;

        transform.position = Vector2.MoveTowards(transform.position,
waypoints[currentWaypointIndex].position, speed * Time.deltaTime);

        if (Vector2.Distance(transform.position, waypoints[currentWaypointIndex].position)
< 0.1f)
        {
            currentWaypointIndex = (currentWaypointIndex + 1) % waypoints.Length;
        }
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            PlayerController player =
collision.gameObject.GetComponent<PlayerController>();
            if (player != null)
            {
                player.TakeDamage(damage);
            }
        }
    }
}

public class CameraFollow : MonoBehaviour
```

```
{
    public Transform player;
    public float smoothSpeed = 0.125f;
    public Vector3 offset;

    void LateUpdate()
    {
        Vector3 desiredPosition = player.position + offset;
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition,
smoothSpeed);
        transform.position = smoothedPosition;
    }
}

public class LevelManager : MonoBehaviour
{
    public static LevelManager instance;

    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }

    public void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

```
    }

    public void LoadNextLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LobbyManager : MonoBehaviourPunCallbacks
{
    public InputField playerNameInput;
    public InputField roomNameInput;
    public Text statusText;
    public GameObject lobbyPanel;
    public GameObject roomPanel;

    void Start()
    {
        PhotonNetwork.ConnectUsingSettings();
    }

    public override void OnConnectedToMaster()
    {
        statusText.text = "Connected to master server.";
        PhotonNetwork.JoinLobby();
    }

    public override void OnJoinedLobby()
```

```
{
    statusText.text = "Joined lobby.";
    lobbyPanel.SetActive(true);
}

public void CreateRoom()
{
    string roomName = roomNameInput.text;
    if (!string.IsNullOrEmpty(roomName))
    {
        PhotonNetwork.CreateRoom(roomName);
    }
}

public void JoinRoom()
{
    string roomName = roomNameInput.text;
    if (!string.IsNullOrEmpty(roomName))
    {
        PhotonNetwork.JoinRoom(roomName);
    }
}

public override void OnJoinedRoom()
{
    statusText.text = "Joined room: " + PhotonNetwork.CurrentRoom.Name;
    lobbyPanel.SetActive(false);
    roomPanel.SetActive(true);
}

public override void OnCreateRoomFailed(short returnCode, string message)
{
    statusText.text = "Create room failed: " + message;
```

```
    }

    public override void OnJoinRoomFailed(short returnCode, string message)
    {
        statusText.text = "Join room failed: " + message;
    }

    public void LeaveRoom()
    {
        PhotonNetwork.LeaveRoom();
    }

    public override void OnLeftRoom()
    {
        statusText.text = "Left room.";
        roomPanel.SetActive(false);
        lobbyPanel.SetActive(true);
    }
}

public class EventManager : MonoBehaviour
{
    public static EventManager instance;

    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else
        {
            Destroy(gameObject);
```

```
        }
    }

    public delegate void PlayerDied();
    public static event PlayerDied OnPlayerDied;

    public void PlayerHasDied()
    {
        if (OnPlayerDied != null)
        {
            OnPlayerDied();
        }
    }
}

public class UIManager : MonoBehaviour
{
    public GameObject pauseMenu;

    public void PauseGame()
    {
        pauseMenu.SetActive(true);
        Time.timeScale = 0f;
    }

    public void ResumeGame()
    {
        pauseMenu.SetActive(false);
        Time.timeScale = 1f;
    }

    public void QuitGame()
    {
```

```
        Application.Quit();
    }

    public void RestartGame()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}

//Файл PlayerController.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerController : MonoBehaviour {
public Rigidbody2D theRB;
public float moveSpeed;
public Animator myAnim;
public static PlayerController instance;
public string areaTransitionName;
private Vector3 bottomLeftLimit;
private Vector3 topRightLimit;
public bool canMove = true;
// Use this for initialization
void Start () {
if (instance == null)
{
instance = this;
} else
{
if (instance != this)
{
Destroy(gameObject);
```

```
}
}
DontDestroyOnLoad(gameObject);
}
// Update is called once per frame
void Update () {
if (canMove)
{
theRB.velocity = new Vector2(Input.GetAxisRaw("Horizontal"),
Input.GetAxisRaw("Vertical")) * moveSpeed;
} else
{
theRB.velocity = Vector2.zero;
}
myAnim.SetFloat("moveX", theRB.velocity.x);
myAnim.SetFloat("moveY", theRB.velocity.y);
if (Input.GetAxisRaw("Horizontal") == 1 || Input.GetAxisRaw("Horizontal") == -1
|| Input.GetAxisRaw("Vertical") == 1 || Input.GetAxisRaw("Vertical") == -1)
{
if (canMove)
{
myAnim.SetFloat("lastMoveX", Input.GetAxisRaw("Horizontal"));
myAnim.SetFloat("lastMoveY", Input.GetAxisRaw("Vertical"));
}
}
transform.position = new Vector3(Mathf.Clamp(transform.position.x,
bottomLeftLimit.x, topRightLimit.x), Mathf.Clamp(transform.position.y,
bottomLeftLimit.y,
topRightLimit.y), transform.position.z);
}
public void SetBounds(Vector3 botLeft, Vector3 topRight)
{
bottomLeftLimit = botLeft + new Vector3(.5f, 1f, 0f);
```

```
topRightLimit = topRight + new Vector3(-.5f, -1f, 0f);
}
}
//Файл DialogManager
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class DialogManager : MonoBehaviour {
public Text dialogText;
public Text nameText;
public GameObject dialogBox;
public GameObject nameBox;
public string[] dialogLines;
public int currentLine;
private bool justStarted;
public static DialogManager instance;
private string questToMark;
private bool markQuestComplete;
private bool shouldMarkQuest;
// Use this for initialization
void Start () {
instance = this;
//dialogText.text = dialogLines[currentLine];
}
// Update is called once per frame
void Update () {
if(dialogBox.activeInHierarchy)
{
if(Input.GetButtonUp("Fire1"))
{
if (!justStarted)
{
```

```
currentLine++;
if (currentLine >= dialogLines.Length)
{
dialogBox.SetActive(false);
GameManager.instance.dialogActive = false;
if(shouldMarkQuest)
{
shouldMarkQuest = false;
if(markQuestComplete)
{
QuestManager.instance.MarkQuestComplete(questToMark);
} else
{
QuestManager.instance.MarkQuestIncomplete(questToMark);
}
}
}
else
{
CheckIfName();
dialogText.text = dialogLines[currentLine];
}
} else
{
justStarted = false;
}
}
}
}
public void ShowDialog(string[] newLines, bool isPerson)
{
dialogLines = newLines;
currentLine = 0;
```

```
CheckIfName();
dialogText.text = dialogLines[currentLine];
dialogBox.SetActive(true);
justStarted = true;
nameBox.SetActive(isPerson);
GameManager.instance.dialogActive = true;
}
public void CheckIfName()
{
if(dialogLines[currentLine].StartsWith("n-"))
{
nameText.text = dialogLines[currentLine].Replace("n-", "");
currentLine++;
}
}
public void ShouldActivateQuestAtEnd(string questName, bool markComplete)
{
questToMark = questName;
markQuestComplete = markComplete;
shouldMarkQuest = true;
}}
//Файл CameraController.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;
public class CameraController : MonoBehaviour {
public Transform target;
public Tilemap theMap;
private Vector3 bottomLeftLimit;
private Vector3 topRightLimit;
private float halfHeight;
private float halfWidth;
```

```
public int musicToPlay;
private bool musicStarted;
void Start () {
target = FindObjectOfType<PlayerController>().transform;
halfHeight = Camera.main.orthographicSize;
halfWidth = halfHeight * Camera.main.aspect;
theMap.CompressBounds();
bottomLeftLimit = theMap.localBounds.min + new Vector3(halfWidth, halfHeight,
0f);
topRightLimit = theMap.localBounds.max + new Vector3(-halfWidth, -halfHeight,
0f);
PlayerController.instance.SetBounds(theMap.localBounds.min,
theMap.localBounds.max);
}
// LateUpdate is called once per frame after Update
void LateUpdate () {
transform.position = new Vector3(target.position.x, target.position.y,
transform.position.z);
transform.position = new Vector3(Mathf.Clamp(transform.position.x,
bottomLeftLimit.x, topRightLimit.x), Mathf.Clamp(transform.position.y,
bottomLeftLimit.y,
topRightLimit.y), transform.position.z);
if(!musicStarted)
{
musicStarted = true;
AudioManager.instance.PlayBGM(musicToPlay);
}
}
}
//Файл MainMenu.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class GameMenu : MonoBehaviour {
public GameObject theMenu;
public GameObject[] windows;
private CharStats[] playerStats;
public Text[] nameText, hpText, mpText, lvlText, expText;
public Slider[] expSlider;
public Image[] charImage;
public GameObject[] charStatHolder;
public GameObject[] statusButtons;
public Text statusName, statusHP, statusMP, statusStr, statusDef, statusWpnEqpd,
statusWpnPwr, statusArmrEqpd, statusArmrPwr, statusExp;
public Image statusImage;
public ItemButton[] itemButtons;
public string selectedItem;
public Item activeItem;
public Text itemName, itemDescription, useButtonText;
public GameObject itemCharChoiceMenu;
public Text[] itemCharChoiceNames;
public static GameMenu instance;
public Text goldText;
public string mainMenuName;
// Use this for initialization
void Start () {
instance = this;
}
// Update is called once per frame
void Update () {
if(Input.GetButtonDown("Fire2"))
{
if(theMenu.activeInHierarchy)
{
```

```
CloseMenu();
} else
{
theMenu.SetActive(true);
UpdateMainStats();
GameManager.instance.gameMenuOpen = true;
}
AudioManager.instance.PlaySFX(5);
}
}
public void UpdateMainStats()
{
playerStats = GameManager.instance.playerStats;
for(int i = 0; i < playerStats.Length; i++)
{
if(playerStats[i].gameObject.activeInHierarchy)
{
charStatHolder[i].SetActive(true);
nameText[i].text = playerStats[i].charName;
hpText[i].text = "HP: " + playerStats[i].currentHP + "/" +
playerStats[i].maxHP;
mpText[i].text = "MP: " + playerStats[i].currentMP + "/" +
playerStats[i].maxMP;
lvlText[i].text = "Lvl: " + playerStats[i].playerLevel;
expText[i].text = "" + playerStats[i].currentEXP + "/" +
playerStats[i].expToNextLevel[playerStats[i].playerLevel];
expSlider[i].maxValue =
playerStats[i].expToNextLevel[playerStats[i].playerLevel];
expSlider[i].value = playerStats[i].currentEXP;
charImage[i].sprite = playerStats[i].charIamge;
} else
{
charStatHolder[i].SetActive(false);
```

```
}
}
goldText.text = GameManager.instance.currentGold.ToString() + "g";
}
public void ToggleWindow(int windowNumber)
{
UpdateMainStats();
for(int i = 0; i < windows.Length; i++)
{
if(i == windowNumber)
{
windows[i].SetActive(!windows[i].activeInHierarchy);
} else
{
windows[i].SetActive(false);
70
}
}
itemCharChoiceMenu.SetActive(false);
}
public void CloseMenu()
{
for(int i = 0; i < windows.Length; i++)
{
windows[i].SetActive(false);
}
theMenu.SetActive(false);
GameManager.instance.gameMenuOpen = false;
itemCharChoiceMenu.SetActive(false);
}
public void OpenStatus()
{
UpdateMainStats();
```

```
//оновлення інформації
StatusChar(0);
for(int i = 0; i < statusButtons.Length; i++)
{
statusButtons[i].SetActive(playerStats[i].gameObject.activeInHierarchy);
statusButtons[i].GetComponentInChildren<Text>().text =
playerStats[i].charName;
}
}
public void StatusChar(int selected)
{
statusName.text = playerStats[selected].charName;
statusHP.text = "" + playerStats[selected].currentHP + "/" +
playerStats[selected].maxHP;
statusMP.text = "" + playerStats[selected].currentMP + "/" +
playerStats[selected].maxMP;
statusStr.text = playerStats[selected].strength.ToString();
statusDef.text = playerStats[selected].defence.ToString();
if(playerStats[selected].equippedWpn != "")
{
statusWpnEqpd.text = playerStats[selected].equippedWpn;
}
statusWpnPwr.text = playerStats[selected].wpnPwr.ToString();
if (playerStats[selected].equippedArmr != "")
{
statusArmrEqpd.text = playerStats[selected].equippedArmr;
}
statusArmrPwr.text = playerStats[selected].armrPwr.ToString();
statusExp.text =
(playerStats[selected].expToNextLevel[playerStats[selected].playerLevel] -
playerStats[selected].currentEXP).ToString();
statusImage.sprite = playerStats[selected].charIamge;
}
```

```
public void ShowItems()
{
GameManager.instance.SortItems();
71
for(int i = 0; i < itemButtons.Length; i++)
{
itemButtons[i].buttonValue = i;
if(GameManager.instance.itemsHeld[i] != "")
{
itemButtons[i].buttonImage.gameObject.SetActive(true);
itemButtons[i].buttonImage.sprite =
GameManager.instance.GetItemDetails(GameManager.instance.itemsHeld[i]).itemSprite;
itemButtons[i].amountText.text =
GameManager.instance.numberOfItems[i].ToString();
} else
{
itemButtons[i].buttonImage.gameObject.SetActive(false);
itemButtons[i].amountText.text = "";
}
}
}
public void SelectItem(Item newItem)
{
activeItem = newItem;
if(activeItem.isItem)
{
useButtonText.text = "Use";
}
if(activeItem.isWeapon || activeItem.isArmour)
{
useButtonText.text = "Equip";
}
itemName.text = activeItem.itemName;
```

```
itemDescription.text = activeItem.description;
}
public void DiscardItem()
{
if(activeItem != null)
{
GameManager.instance.RemoveItem(activeItem.itemName);}}
public void OpenItemCharChoice()
{itemCharChoiceMenu.SetActive(true);
for(int i = 0; i < itemCharChoiceNames.Length; i++)}
public void CloseItemCharChoice()
{itemCharChoiceMenu.SetActive(false);}
public void UseItem(int selectChar)
{activeItem.Use(selectChar);
CloseItemCharChoice();}
public void SaveGame()
{GameManager.instance.SaveData();
QuestManager.instance.SaveQuestData();}
public void PlayButtonSound()
{AudioManager.instance.PlaySFX(4);}
```

ДОДАТОК Ж

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Ігрові механіки, адаптовані до апаратної структури кросплатформенного проєкту на Unity3D

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,5% Схожість 1,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Колеснікова Н.С.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Городецька О.С.
(підпис) (прізвище, ініціали)