МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)
магістра
(ступінь вищої освіти)

на тему АРКАДНА ГРА ДЛЯ ОС ANDROID З МОНЕТИЗАЦІЄЮ

Виконав: студент 2 курсу, групи КНТ-513м

Спеціальності 123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні системи та мережі

СУШКО О.С.
(ПРІЗВИЩЕ та ініціали)

Керівник ТЯГУНОВА М.Ю.
(ПРІЗВИЩЕ та ініціали)

Рецензент МАЛИЙ О.Ю.
(ПРІЗВИЩЕ та ініціали)

2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра Комп'ютерні системи та мережі
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.

" 18 " жовтня 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

СУШКА Олександра Сергійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Аркадна гра для ОС Android з монетизацією
керівник проєкту (роботи) к.т.н., доцент, ТЯГУНОВА Марія Юріївна
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від 18 жовтня 2024 року №149
2. Строк подання студентом проєкту (роботи) 10 грудня 2024 року
3. Вихідні дані до проєкту (роботи) наявні методи та технології реалізації Аркадної гри для ОС Android з монетизацією
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) Аналіз інструментів та технологій розробки
2) Аналіз та вибір стратегій монетизації
3) Розробка концепції та проектування
4) Реалізація
5) Тестування та оптимізація

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ТЯГУНОВА М.Ю., к.т.н., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних інструментів та технологій розробки мобільних ігор, перегляд отриманих результатів, виділення їх переваг та недоліків	09.10.2024 р.	
2	Аналіз стратегій монетизації, наявних рекламних мереж та форматів реклами, перегляд отриманих результатів, виділення їх переваг та недоліків	18.10.2024 р.	
3	Розробка концепції гри та її проєктування	29.10.2024 р.	
4	Розробка дизайну гри	03.11.2024 р.	
5	Реалізація функціоналу гри	18.11.2024 р.	
6	Інтеграція внутрішньоігрової реклами	23.11.2024 р.	
7	Тестування гри	28.11.2024 р.	
8	Оптимізація гри	02.12.2024 р.	
9	Оформлення отриманих результатів у ПЗ	07.12.2024 р.	
10	Оформлення графічного матеріалу	09.12.2024 р.	
11	Оформлення допоміжного матеріалу	10.12.2024 р.	

Студент _____ **Олександр СУШКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ **Марія ТЯГУНОВА**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 141 с., 96 рис., 15 лістингів, 57 джерел.

GOOGLE ADMOB, UNITY, UNITY PROFILER, АРКАДНА ГРА, МОБІЛЬНА ГРА, МОНЕТИЗАЦІЯ, ОС ANDROID

Об’єкт дослідження — мобільні аркадні ігри для ОС Android.

Предмет дослідження — ефективні методи монетизації для мобільних аркадних ігор.

Мета роботи — забезпечити стабільну взаємодію з користувачами та високий рівень їх утримання в аркадній грі для ОС Android з реалізацією сучасних та ефективних методів монетизації.

У першому розділі проведено аналіз інструментів та технологій розробки мобільних ігор, зокрема обрано ігровий двигун Unity та мову програмування C#.

Другий розділ зосереджений на дослідженні стратегій монетизації мобільних ігор, аналізі рекламних мереж та форматів внутрішньоігрової реклами. Як результат обрано стратегію внутрішньоігрової реклами через мережу Google AdMob, використовуючи проміжну рекламу та рекламу з винагородою.

У третьому розділі було розроблено концепцію гри, а саме: визначено її жанр (аркада), основну ідею (заповнення споруди у заданий час), цільову аудиторію (казуальні гравці) та стратегію монетизації, а також виконано проєктування гри.

У четвертому розділі відбувається реалізація гри, включаючи дизайн, програмний код, функціонал, інтеграція внутрішньоігрової реклами та інше.

П'ятий розділ охоплює проведення тестування гри, оцінку її продуктивності та функціоналу, а також оптимізацію для покращення користувацького досвіду.

Результатом роботи є аналіз інструментів розробки та стратегій монетизації, розроблено мобільну аркадну гру, інтегровано внутрішньоігрову рекламу та виконано тестування та оптимізацію для стабільної роботи на платформі Android.

ABSTRACT

EN: 141 p., 96 figures, 15 listings, 57 references.

GOOGLE ADMOB, UNITY, UNITY PROFILER, ARCADE GAME, MOBILE GAME, MONETIZATION, ANDROID OS

The object of research is mobile arcade games for Android OS.

The subject of the study is effective monetization methods for mobile arcade games.

The purpose of the study is to ensure stable interaction with users and a high level of their retention in an arcade game for Android OS with the implementation of modern and effective monetization methods.

The first section analyzes the tools and technologies for developing mobile games, in particular, the Unity game engine and the C# programming language.

The second section focuses on the study of mobile game monetization strategies, analysis of advertising networks and in-game advertising formats. As a result, the strategy of in-game advertising through the Google AdMob network using interstitial and rewarded advertising was chosen.

In the third section, the game concept was developed, namely, its genre (arcade), the main idea (filling the building in a given time), the target audience (casual players) and the monetization strategy were determined, and the game design was performed.

The fourth section covers the implementation of the game, including design, program code, functionality, integration of in-game advertising, etc.

The fifth section covers testing the game, evaluating its performance and functionality, and optimizing it to improve the user experience.

The result of the work is the analysis of development tools and monetization strategies, the development of a mobile arcade game, the integration of in-game advertising, and testing and optimization for stable operation on the Android platform.

ЗМІСТ

ВСТУП

З кожним роком кількість користувачів смартфонів у світі продовжує зростати, і ці пристрої стають невіддільною частиною повсякденного життя мільйонів людей. Смартфони використовуються не тільки для зв’язку чи роботи, але і як основне джерело розваг. За останні десятиліття мобільні ігри стали надзвичайно популярними серед користувачів різних вікових категорій, охоплюючи як дітей, так і дорослих. Вони допомагають відірватися від буденних турбот або скоротати час, наприклад, під час поїздок або перерв на роботі.

Розробка мобільних ігор саме на платформі Android є одним із найбільш перспективних напрямків у сфері цифрових розваг. Платформа Android охоплює значну частину світового ринку мобільних пристроїв, що надає можливість залучати широку аудиторію користувачів. Серед усіх жанрів мобільних ігор аркадні ігри є одними з найпопулярніших завдяки простому управлінню, коротким ігровим сесіям та можливості швидкого занурення у геймплей. Водночас важливо не лише створити цікавий продукт, але й забезпечити його економічну життєздатність. У сучасному ринку мобільних ігор монетизація є ключовим компонентом, що дозволяє не лише повернути вкладені ресурси, але й забезпечити стабільний прибуток.

Актуальність дослідження зумовлена не лише зростанням ринку мобільних ігор, а й необхідністю впровадження ефективних моделей монетизації. Сучасні користувачі частіше надають перевагу безкоштовним іграм, що призводить до збільшення попиту на різноманітні методи непрямої монетизації, такі як реклама, внутрішньоігрові покупки та преміумконтент. Тому дослідження оптимальних підходів до монетизації мобільних ігор на платформі Android може сприяти успішному розвитку не лише окремих проєктів, а й усього ринку мобільних ігор.

Мета роботи — забезпечити стабільну взаємодію з користувачами та високий рівень їх утримання в аркадній грі для ОС Android з реалізацією сучасних та ефективних методів монетизації.

Відповідно до мети дослідження необхідно виконати такі завдання:

– проаналізувати програмні засоби та технології розробки мобільних ігор;

– проаналізувати наявні мережі та методи монетизації мобільних ігор;

– сформувати вимоги до розробки мобільної гри на платформі Android;

– спроєктувати структуру та розробити концепцію гри;

– реалізувати гру;

– інтегрувати засоби монетизації;

– провести тестування розробки та оцінити ефективність обраних методів монетизації;

– провести оптимізацію гри.

1 АНАЛІЗ ІНСТРУМЕНТІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ МОБІЛЬНИХ ІГОР

У цьому розділі ми проаналізуємо інструменти та технології, які використовуються у процесі розробки мобільних ігор. Цей етап має важливе значення для створення успішного продукту.

Метою даного розділу є надання всебічного огляду інструментів і методів, що формують основу для розробки мобільних ігор.

Вибір технологій має важливе значення для успішної розробки мобільних ігор, оскільки він впливає на всі ключові аспекти процесу. Обрана технологія визначає продуктивність гри, включаючи її графіку, швидкість і стабільність. Неоптимальні рішення можуть призвести до зниження якості геймплею, що негативно позначиться на загальному враженні гравців. Крім того, правильно підібрана технологія може суттєво скоротити час розробки, адже ігровий двигун та інструменти, які максимально відповідають потребам проєкту, роблять процес створення гри швидшим і ефективнішим.

Технології також мають вплив на витрати: деякі з них потребують додаткових ліцензій, плагінів або спеціального обладнання, що підвищує бюджет. Зважений вибір інструментів дозволяє уникнути зайвих витрат та забезпечити фінансову ефективність розробки. Мобільні ігри мають працювати на різних операційних системах і пристроях, тому для максимального охоплення аудиторії важливо обрати технологію, яка підтримує кросплатформенність. Це дозволяє значно спростити перенесення гри під нові платформи, як, наприклад iOS, що своєю чергою розширює потенційну аудиторію.

Крім того, обрана технологія впливає на можливість майбутнього розширення гри та додавання нових функцій, що важливо для підтримки оновлень і збереження інтересу користувачів. Отже, зважений вибір технологій є важливим етапом, від якого залежить продуктивність, ефективність розробки, бюджетні витрати, сумісність із платформами та потенціал для подальшого розвитку проєкту.

1.1 Ігрові двигуни

Ігровий двигун — це спеціальне програмне забезпечення, що значно спрощує процес розробки комп'ютерних ігор. Він об’єднує в собі різні інструменти та бібліотеки, що допомагають створювати ігровий контент, контролювати ігровий процес, працювати з графікою, звуком, фізикою, а також обробляти введення й виведення інформації. Ігровий двигун дозволяє реалізовувати не тільки логіку гри та поведінку об'єктів, але й забезпечує інтеграцію різноманітних елементів, що робить його незамінним у створенні ігрових проєктів різних жанрів та платформ.

Завдяки йому розробники можуть використовувати готові засоби для роботи з графікою, що включають різні методи рендерингу, роботу з 3D-моделями, текстурами та спецефектами. Крім того, він забезпечує фізичні обчислення, зокрема, обробку зіткнень, гравітації та симуляцію різноманітних процесів, таких як поведінка рідин і тканин. Також двигун обробляє введення від користувача, забезпечує виведення на екран і інтеграцію зі звуком, надає інструменти для управління ігровим процесом, включно зі створенням рівнів, збереженням і завантаженням гри, а також керуванням інтерфейсом і камерою [1].

Завдяки широкому функціоналу ігрового двигуна час на розробку значно скорочується, адже більшість базових компонентів для створення гри вже готові до використання.

Для розробки мобільних ігор доступна велика кількість ігрових двигунів, серед яких найбільш популярні:

– Defold;

– Cocos2D;

– CryEngine;

– Construct 3;

– Unreal Engine;

– Godot Engine;

– Unity.

1.1.1 Ігровий двигун Defold

Defold — це безкоштовний ігровий двигун, орієнтований на 2D-графіку, створений King, відомим своїми іграми на кшталт Candy Crush Saga. Логотип цього двигуна зображено на рисунку 1.1. Defold підтримує різні платформи, включаючи Windows, macOS, Android, iOS та HTML5. Двигун активно використовується розробниками, оскільки має низькі системні вимоги та зручний інтерфейс [2].

Рисунок 1.1 — Логотип «Defold» [3]

Переваги Defold:

– простота та легкість. Defold має компактний розмір і мінімальні вимоги до системних ресурсів, що дозволяє працювати навіть на слабших комп’ютерах. Крім того, двигун має інтуїтивний інтерфейс, який робить його зручним для початківців;

– підтримка Lua. Defold використовує Lua як мову програмування для скриптів, що є простою для вивчення, особливо для розробників з базовими знаннями в програмуванні;

– підтримка кросплатформенності. Defold дозволяє розгортати ігри на різних платформах без необхідності значного доопрацювання коду, що є великим плюсом для розробників, які хочуть охопити широку аудиторію;

– вбудовані інструменти для монетизації. Defold має можливість інтеграції

рекламних мереж і систем для внутрішньоігрових покупок, що робить його зручним для створення комерційних проєктів;

– вбудоване управління ресурсами. Defold забезпечує оптимальне управління ресурсами та пам’яттю, що дозволяє створювати добре оптимізовані ігри;

– активна спільнота та документація. Defold має активну спільноту розробників, а також багату документацію та приклади, що допомагають швидко освоїти основи роботи з двигуном.

Недоліки Defold:

– обмежені можливості для 3D-графіки. Defold основно орієнтований на 2D-графіку, а його 3D-можливості обмежені. Це може стати обмеженням для розробників, які хочуть створити тривимірні ігри;

– менш популярний, ніж конкуренти. Defold не такий популярний, як Unity чи Unreal Engine, тому кількість ресурсів і плагінів, створених спільнотою, менша. Це може ускладнити пошук специфічних ресурсів;

– менша підтримка API сторонніх сервісів. Деякі розробники можуть зіткнутися з труднощами при інтеграції специфічних сервісів (наприклад, спеціалізованих рекламних мереж або аналітики), оскільки Defold не завжди має готові рішення для цього;

– відсутність візуального редактора для скриптів. Defold не має вбудованого редактора для візуального програмування, як у випадку з такими двигунами, як Godot чи Unity. Це недолік для тих, хто не має досвіду написання коду [4, 5].

1.1.2 Ігровий двигун Cocos2D

Cocos2D — це безкоштовний, відкритий ігровий двигун, який здебільшого використовується для створення 2D-ігор, інтерактивних програм та інших мультимедійних додатків. На рисунку 1.2 зображено логотип цього двигуна. Двигун підтримує мови програмування, такі як C++, Lua та JavaScript, і здатний експортувати проєкти на основні платформи, включаючи Android, iOS, Windows, macOS та веб-браузери. Cocos2D часто обирають для мобільних ігор завдяки його легкій вазі, хорошій продуктивності та спрощеній структурі [7].

Рисунок 1.2 — Логотип «Cocos2D» [6]

Основні переваги Cocos2D:

– відкритий вихідний код. Повністю безкоштовний і відкритий, що дає можливість змінювати код під потреби конкретного проєкту;

– платформна незалежність. Підтримка експорту на кілька платформ (Android, iOS, веб, ПК), що дозволяє охопити велику аудиторію;

– легка вага та продуктивність. Оптимізований для роботи з невеликим обсягом пам'яті, що добре підходить для мобільних ігор і забезпечує хорошу продуктивність;

– зручні інструменти для розробки UI. Забезпечує потужний набір інструментів для анімації, рендерингу та створення графіки інтерфейсу користувача;

– спільнота та підтримка. Має досить велику спільноту, яка надає ресурси, плагіни, розширення та підтримку.

Основні недоліки Cocos2D:

– обмежена підтримка 3D. Cocos2D здебільшого орієнтований на 2D-ігри, тому 3D-функціонал обмежений і не підходить для повноцінних 3D-проєктів;

– складність освоєння для новачків. Відсутність інтуїтивно зрозумілих інструментів для початківців, що може ускладнити процес навчання для нових розробників;

– відсутність повноцінного редактора сцени. Інтерфейс для візуального

редагування менш розвинений, ніж у конкурентів, таких як Unity, що може уповільнити розробку;

– менше ресурсів та навчальних матеріалів. У порівнянні з більш популярними двигунами, такими як Unity або Unreal, Cocos2D має менше навчальних ресурсів та курсів [7].

1.1.3 Ігровий двигун CryEngine

CryEngine — це ігровий двигун, розроблений німецькою компанією Crytek, що став відомим завдяки дивовижним візуальним можливостям, високій якості графіки та здатності обробляти великі відкриті простори. На рисунку 1.3 зображено логотип цього двигуна. Спочатку CryEngine використовувався для створення таких відомих ігор, як серія Far Cry та Crysis, які задали новий стандарт графічної якості для ігрової індустрії [9].

Рисунок 1.3 — Логотип «CryEngine» [8]

Переваги CryEngine:

– якість графіки. CryEngine забезпечує високоякісну графіку, яка виглядає дивовижно навіть у великих масштабах;

– масштабованість. Підтримує як великі відкриті світи, так і більш компактні та деталізовані ігрові простори;

– фізика в реальному часі. Двигун має добре оптимізовану фізику, що забезпечує реалістичну поведінку об’єктів;

– можливості рендерингу у віртуальній реальності (VR). Двигун добре підходить для VR-ігор та застосунків;

– сильний інструментарій для AI та сценарного менеджменту. CryEngine дозволяє створювати складні взаємодії та поведінкові сценарії для NPC.

Недоліки CryEngine:

– складність у навчанні. CryEngine вважається одним зі складніших двигунів для освоєння, тому вивчення та використання його вимагає більше часу та ресурсів;

– високі вимоги до обладнання. Для повноцінного використання його потужностей потрібне потужне обладнання, що може бути обмеженням для інді-розробників;

– менша спільнота. Порівняно з Unity або Unreal Engine, CryEngine має меншу спільноту, що ускладнює пошук прикладів коду, підказок або готових рішень;

– ліцензування та комерційна підтримка. Вартість використання двигуна та підтримка від Crytek можуть бути досить дорогими для невеликих студій або незалежних розробників [9, 10].

1.1.4 Ігровий двигун Construct 3

Construct 3 — це ігровий двигун для 2D-ігор, який працює в браузері та орієнтований на простоту використання, особливо для початківців і розробників без глибоких знань програмування. На рисунку 1.4 зображено логотип цього двигуна. В основі Construct 3 лежить інтерфейс, що дозволяє створювати ігри без необхідності писати код завдяки використанню візуального скриптування [11].

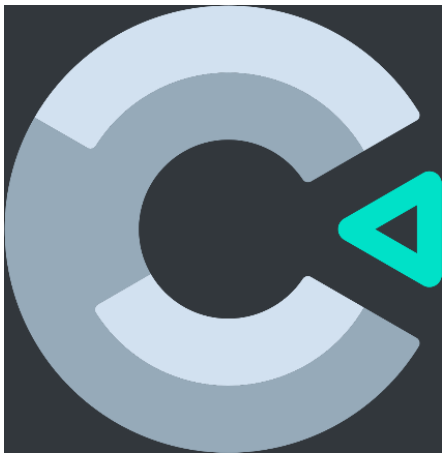

Рисунок 1.4 — Логотип «Construct3» [12]

Переваги Construct 3:

– простота використання. Відсутність потреби у глибоких знаннях програмування дозволяє швидко розпочати роботу, що робить двигун популярним серед початківців;

– швидка розробка. Візуальний підхід до створення гри допомагає значно скоротити час розробки;

– підтримка кросплатформеності. Завдяки кросплатформеній підтримці, створену гру легко адаптувати для різних платформ, включно з мобільними пристроями;

– велика спільнота і документація. Construct має широку базу користувачів та багато навчальних матеріалів, що може бути корисним для розв'язання проблем під час розробки;

– інтеграція з монетизацією. Construct 3 дозволяє додавати рекламу та внутрішньоігрові покупки, що особливо корисно для мобільних ігор з монетизацією [13].

Недоліки Construct 3:

– обмеження в роботі з 3D. Construct 3 більше підходить для 2D-ігор, тому для складних 3D-проєктів він малоефективний;

– обмеження в масштабі. Двигун більше підходить для невеликих ігор, оскільки для великих, складних проєктів може бракувати продуктивності та функціоналу;

– обмежені можливості для кастомізації. Хоча візуальний скриптінг дозволяє швидко створювати основну логіку, для тонкої настройки гри можливостей може не вистачати, особливо якщо потрібен доступ до низькорівневих налаштувань;

– вартість. Construct 3 має підписку, що може бути недешево для розробників-ентузіастів чи студентів [14].

1.1.5 Ігровий двигун Unreal Engine

Unreal Engine — це потужний ігровий двигун, створений компанією Epic Games, що використовується для розробки ігор, анімацій, віртуальної реальності

(VR), архітектурної візуалізації, фільмів та інших інтерактивних додатків. На рисунку 1.5 зображено логотип цього двигуна. Першу версію було випущено у 1998 році, а з роками він отримав численні вдосконалення, перетворившись на один з найпопулярніших інструментів для розробників ігор [15, 16].

Рисунок 1.5 — Логотип «Unreal Engine» [17]

Переваги Unreal Engine:

– висока якість графіки. Unreal Engine забезпечує дивовижну візуальну якість, яка використовується у багатьох AAA-іграх. Він пропонує можливості рейтрейсингу (Ray Tracing), що забезпечує реалістичне освітлення та відображення;

– блєкпринти (Blueprints). Завдяки візуальній мові програмування Blueprints, Unreal Engine дозволяє створювати складні механіки без написання коду, що підходить для новачків та швидкого прототипування;

– потужний інструментарій для розробки VR та AR. Unreal Engine підтримує розробку проєктів для віртуальної та доповненої реальності, що робить його популярним вибором у галузях VR-ігор та архітектурної візуалізації;

– масштабованість та кросплатформеність. Двигун дозволяє створювати проєкти для різних платформ — ПК, консолей, мобільних пристроїв, VR та AR, а

також підтримує високий рівень оптимізації для великих сцен та відкритих світів;

– Unreal Marketplace. Магазин Unreal Marketplace пропонує тисячі готових моделей, текстур, анімацій, звуків і кодів, що може пришвидшити розробку гри;

– спільнота та ресурси для навчання. Unreal Engine має величезну спільноту розробників і ресурси для навчання, що включають документацію, відеоуроки, курси, та активний форум підтримки [18].

Недоліки Unreal Engine:

– складність. Хоча Blueprints спрощують розробку, для глибшого налаштування гри потрібно знання C++, а сам двигун може бути складним для початківців;

– вимогливість до ресурсів. Unreal Engine відомий своєю вимогливістю до апаратного забезпечення, що може ускладнити розробку на менш потужних комп'ютерах. Також деякі ігри на Unreal Engine можуть вимагати більше ресурсів на користувацькому пристрої;

– розмір фінального продукту. Ігри на Unreal Engine часто мають великий обсяг пам'яті, що може вплинути на їх розмір при завантаженні та на завантажувальний час;

– ліцензування. Epic Games надає Unreal Engine безкоштовно, проте вимагає роялті в розмірі 5% від доходу, якщо гра перевищує $1 мільйон виручки. Для великих проєктів це може бути значною сумою [19].

1.1.6 Ігровий двигун Godot Engine

Godot Engine — це відкритий ігровий двигун для розробки 2D та 3D ігор. На рисунку 1.6 зображено логотип цього двигуна. Основний акцент зроблено на простоту, кросплатформеність та підтримку інді-розробників і невеликих команд. Godot активно розвивається та має повністю відкритий код, що дозволяє будь-кому вносити зміни й додавати функції. Він безкоштовний для комерційного використання та не має ліцензійних зборів чи роялті [20].

Переваги Godot:

– простота та зручність для новачків. Godot пропонує інтуїтивно зрозумілий інтерфейс та систему вузлів, що полегшує побудову ігрових сцен. Це дає змогу

легко налаштовувати ігровий процес та взаємодію об’єктів навіть для новачків, чого не завжди можна досягти в Unity чи Unreal Engine, де потрібне більше досвіду;

– підтримка 2D-графіки. У роботі з 2D-іграми Godot є одним із найкращих рішень, завдяки виділеному 2D-рендеру, що дозволяє уникнути помилок глибини та оптимізувати продуктивність. Unity, наприклад, може бути менш ефективним для чисто 2D-проєктів [20];

– легке кросплатформене розгортання. Godot дозволяє експортувати ігри на безліч платформ — від настільних і мобільних до веббраузерів. Хоча Unity та Unreal також підтримують кросплатформеність, відсутність ліцензійних платежів у Godot робить його особливо привабливим для інді-розробників;

– безкоштовна ліцензія (MIT License). Godot повністю безкоштовний, і навіть комерційне використання не передбачає додаткових платежів, що вигідно відрізняє його від Unity, де комерційне використання може вимагати ліцензії;

– активна спільнота та відкритий код. Відкрите ліцензування дозволяє легко брати участь у вдосконаленні двигуна та створювати плагіни. Це робить розробку прозорою та гнучкою, чого часто не вистачає у великих комерційних платформах.

Рисунок 1.6 — Логотип «Godot Engine» [21]

Недоліки Godot:

– менше підтримки для 3D-проєктів. Хоча Godot розвиває свої 3D-можливості, він все ще відстає від Unity та Unreal у складних 3D-проєктах. Реалізація освітлення, рендеру та деяких ефектів не така потужна, як у конкурентів, що робить його менш підхідним для ресурсомістких 3D-ігор;

– менше навчальних ресурсів. Попри наявність активної спільноти, ресурси для навчання та туторіали, доступні для Godot, суттєво поступаються кількістю та глибиною матеріалів популярних движків, в яких існує велика кількість офіційних документацій, форумів та онлайн-курсів, які дозволяють швидше освоїти роботу з цими платформами. В свою чергу, обмеженість навчальних матеріалів Godot може створювати додаткові труднощі для нових користувачів, хто не має досвіду в розробці ігор, що може призвести до довшого періоду адаптації та освоєння;

– мова GDScript має обмеження. Хоча GDScript легкий у використанні, його продуктивність та можливості для великих проєктів не завжди відповідають рівню C# або C++ в інших двигунах. Це іноді вимагає переходу на C#, що поки підтримується не так широко;

– продуктивність у великих проєктах. Godot, будучи спрямованим на простіші проєкти, може відставати за продуктивністю на великих, насичених графікою і складною логікою проєктах, де Unity чи Unreal мають більше інструментів для оптимізації [22].

1.1.7 Ігровий двигун Unity

Unity — це один із найпопулярніших ігрових двигунів для створення 2D, 3D, AR та VR контенту. На рисунку 1.7 зображено його логотип. Його використовують як інді-розробники, так і великі студії для створення ігор, інтерактивних додатків та віртуальних середовищ. Unity був вперше випущений у 2005 році компанією Unity Technologies і відтоді став одним із провідних інструментів для розробки інтерактивного контенту завдяки широким можливостям, високій продуктивності та багатій екосистемі [23, 24].

Рисунок 1.7 — Логотип «Unity» [25]

Основні переваги Unity:

– кросплатформність. Unity підтримує понад 25 платформ, зокрема Windows, macOS, iOS, Android, веббраузер (WebGL), VR та AR-платформи (Oculus, Vive, HoloLens) тощо. Це означає, що один і той самий проєкт можна запускати на різних пристроях із мінімальними змінами;

– інтерфейс та інтуїтивність. Інтерфейс Unity відносно легкий у вивченні, що робить його популярним серед початківців. Unity має візуальний редактор, що дозволяє налаштовувати параметри об’єктів у сцені без написання коду;

– багата екосистема. Двигун Unity пропонує величезну кількість ресурсів, які можуть бути використані в ігровому проєкті — від 3D-моделей до кодових пакетів та плагінів. Це значно прискорює розробку та економить час, особливо для інді-розробників;

– широка спільнота. Unity має одну з найбільших спільнот серед ігрових двигунів, тому є багато безкоштовних та платних уроків, форумів, документації та інших навчальних ресурсів. Це допомагає розв'язувати проблеми та швидко вчитися;

– підтримка AR/VR. Unity є одним з найкращих рішень для створення контенту для AR та VR завдяки вбудованим інструментам та підтримці необхідних бібліотек для цих технологій;

– безкоштовна версія. Unity має безкоштовну версію (Unity Personal), яка дозволяє використовувати майже всі функції, зокрема для навчання та невеликих комерційних проєктів, що також сприяє популярності цього інструменту;

– гнучкість у монетизації. Unity має вбудовані можливості для монетизації мобільних ігор, такі як Unity Ads, що дає змогу легко інтегрувати рекламу у свої додатки [24, 26].

Недоліки Unity:

– продуктивність для високоякісної графіки. Unity добре підходить для середніх ігор або 2D/3D проєктів, однак у проєктах з надвисокою якістю графіки (наприклад, AAA-ігри) може поступатися Unreal Engine, оскільки останній має потужніший рендеринг графіки;

– розмір виконуваного файлу. Unity створює більші за розміром файли, ніж деякі інші двигуни. Це може бути проблемою для мобільних платформ, де розмір програми має значення;

– ліцензування. Unity має безкоштовну версію для невеликих команд, але для великих студій або проєктів з доходом понад $100 000 потрібно використовувати платну ліцензію (Unity Pro);

– менша потужність для графіки у реальному часі. Unity поки що трохи відстає у створенні реалістичної графіки реального часу порівняно з Unreal Engine, хоча розробники активно працюють над цим [24, 26].

Після аналізу доступних ігрових двигунів, для цього проєкту було обрано саме Unity. Ось основні причини, що вплинули на вибір:

– важливим аргументом на користь Unity є його простота у вивченні. Освоїти цей двигун для розробки ігор легше, ніж багато інших конкурентів, що створює більше можливостей для нових розробників;

– також Unity чудово адаптований для мобільних пристроїв. Завдяки йому розробники можуть створювати візуально привабливі ігри, які інтегруються з можливостями, що підтримуються більшістю смартфонів, такими як GPS, акселерометри та гіроскопи, що відкриває ширший функціонал. Окрім того, Unity має вбудовані інструменти для аналітики, внутрішньоігрових покупок та реклами, які сприяють монетизації ігор;

– ще одна вагома причина вибору Unity — це його кросплатформеність. Коли гра набере популярності, її легко можна буде адаптувати для інших платформ, розширивши її присутність на ринку мобільних ігор. Ця універсальність значно полегшує досягнення більшої аудиторії;

– також важливим аргументом на користь Unity є його простота у вивченні. Освоїти цей двигун для розробки ігор легше, ніж багато інших конкурентів, що створює більше можливостей для нових розробників та збільшує кількість ігор, створених саме на Unity;

– одна з ключових переваг Unity над іншими двигунами — це його потужна, можна навіть сказати найбільша спільнота розробників. Unity має свою офіційну

платформу для обговорення та спілкування спільноти розробників Unity — Unity Discussions [27]. Вона створена для обміну знаннями, пошуку рішень та отримання підтримки від інших користувачів. Вона функціонує як форум, де можна ставити питання, ділитися порадами, обговорювати новини Unity і нові функції, а також брати участь у дискусіях, пов’язаних із різними аспектами розробки ігор. Unity Discussions підходить як новачкам, так і досвідченим розробникам, адже тут можна знайти як базові поради, так і поглиблену інформацію з просунутих аспектів використання Unity. Крім цієї платформи, існує безліч інтернет-ресурсів: статті, відеоуроки, де можна знайти готові рішення для багатьох питань, які можуть виникнути в процесі розробки на цьому прекрасному двигуні [28]. Тому був вибраний саме Unity.

1.2 Вибір мови програмування

Вибір мови програмування залежить від ігрового двигуна, на якому буде вестись розробка. Для цього проєкту ми обрали Unity. Раніше Unity підтримував такі мови програмування як Boo та UnityScript, але з часом вони відмовилися від них на користь такої сучасної та універсальної мови програмування як C#. Тому на цей час C# є одною та основною мовою програмування в Unity. Тому у цьому проєкті, для написання скриптів ми будемо використовувати C#.

C# — це об'єктно-орієнтована мова програмування, вона з'явилася у 2000 році під керівництвом Андерса Гейлсберга в рамках платформи .NET. Зараз C# є однією з найпопулярніших мов для розробки різноманітних застосунків, зокрема ігор, бізнес-додатків, мобільних додатків, а також вебсервісів. Вона дозволяє створювати ієрархію об'єктів, використовувати наслідування, поліморфізм і інкапсуляцію. Завдяки строгій структурі, C# є досить безпечною і передбачуваною мовою, що сприяє кращому управлінню пам'яттю та уникненню багатьох помилок.

Таким чином, для розробки мобільної гри в рамках цієї роботи було обрано

Unity як ігровий двигун, а C# — як основну мову програмування. Вибір Unity обумовлений його простотою у вивченні, що робить його доступним для нових розробників, а також потужною адаптованістю для мобільних пристроїв, завдяки якій можна створювати візуально привабливі ігри з розширеним функціоналом. Крім того, Unity забезпечує кросплатформеність, що дозволяє в майбутньому адаптувати гру для інших платформ. Найважливішим фактором є також велика спільнота Unity, де можна отримати підтримку та швидко знайти готові рішення. Оскільки Unity підтримує тільки C#, цей вибір був очевидним для написання всіх необхідних скриптів у проєкті.

2 АНАЛІЗ ТА ВИБІР СТРАТЕГІЙ МОНЕТИЗАЦІЇ МОБІЛЬНИХ ІГОР

Під монетизацією розуміють методи та стратегії, що використовуються для отримання доходу від гри. У мобільній ігровій індустрії ефективна монетизація є ключовою частиною успіху. Правильний вибір монетизаційної стратегії визначає не тільки прибутковість гри, а й користувацький досвід, який безпосередньо впливає на відданість гравців і їхнє бажання повертатися до гри знову і знову. Чимало перспективних проєктів натрапляють на труднощі саме через невдало підібрану модель монетизації, яка або не приносить достатнього доходу, або викликає негативне сприйняття з боку користувачів.

Великий вибір платформ та методів монетизації ставить нелегке завдання: необхідно оцінити переваги та недоліки кожного з них, а також зважити, як ці методи поєднуються з концепцією ігрового проєкту. Ігри різних жанрів, цільових аудиторій і механік потребують індивідуального підходу, оскільки один і той самий метод монетизації може по-різному вплинути на різні типи ігор.

Ми проведемо аналіз стратегій монетизації, а також огляд сучасних рекламних мереж та форматів реклами і виберемо найефективніший підхід для розроблюваної гри. Розуміння сильних та слабких сторін допоможе прийняти

правильне рішення, яке забезпечить оптимальний баланс між доходом і позитивним користувацьким досвідом.

2.1 Стратегії монетизації

Стратегія монетизації — це конкретний підхід, для оптимального отримання доходу від своєї гри. Вибір правильної стратегії дуже важливий, оскільки вона безпосередньо впливає на потенційний дохід гри й може суттєво вплинути на досвід гравців. Вдало підібрана стратегія балансує між отриманням прибутку та позитивним користувацьким досвідом, підвищуючи утримання гравців та їхню задоволеність грою. Неправильна стратегія може призвести до розчарування гравців, зниження рівня утримання та поганих відгуків, що в кінцевому підсумку негативно вплине на дохід [29].

Існує декілька основних стратегій монетизації мобільних ігор:

– платні додатки;

– стратегія підписки;

– внутрішньоігрові покупки;

– внутрішньоігрова реклама;

– гібридна стратегія.

Далі проаналізуємо кожну з них окремо.

2.1.1 Платні додатки

Користувачі сплачують певну суму за завантаження та доступ до гри. Ця стратегія проста і вимагає від гравців одноразового платежу для того, щоб насолоджуватися грою в повному обсязі [29].

Переваги:

– дохід отримується одразу з кожним завантаженням;

– ніяких переривань гри або реклами, що забезпечує безперебійний користувацький досвід.

Недоліки:

– обмежує базу гравців, оскільки багато користувачів надають перевагу безкоштовним іграм;

– високі очікування щодо якості гри можуть призвести до поганих відгуків, якщо гра не відповідає стандартам гравців.

2.1.2 Стратегія підписки

Гравці платять регулярну плату за преміум-контент, додаткові функції або без рекламну гру [29].

Види підписок:

– баттл-пасс. Користувачі купують пропуск, який надає їм доступ до нагород, які вони повинні заробити в ігровому процесі протягом певного періоду, що зазвичай триває 15 або 30 днів. Ця стратегія може бути доступна як у безкоштовній, так і в платній версіях, надаючи гравцям різноманітний досвід;

– підписка «Видалення реклами». Ігри, які значною мірою покладаються на внутрішньоігрову рекламу, можуть надавати користувачам можливість усунути її (тимчасово або назавжди), сплативши відповідну плату. Таким чином, користувачі можуть насолоджуватися безперервним ігровим процесом, а ми не втрачаємо дохід;

– VIP-підписка. Підписники отримують спеціальний набір ексклюзивних переваг, включаючи нагороди, видалення реклами.. Це покращує користувацький досвід на певний час, створюючи відчуття VIP-обслуговування в ігровій спільноті.

Переваги:

– забезпечує стабільний і передбачуваний потік доходів завдяки регулярній абонентській платі;

– передплатники більше інвестують у гру, що збільшує утримання гравців.

Недоліки:

– підписка може не сподобатися випадковим гравцям, що обмежує потенційну аудиторію;

– для утримання підписників потрібне якісне оновлення контенту.

2.1.3 Внутрішньоігрові покупки

Внутрішньоігрові покупки, також відомі як мікротранзакції, є однією з моделей монетизації в мобільних іграх. Вони дозволяють користувачам здійснювати платні транзакції в межах додатка після його завантаження. Це означає, що розробники можуть зробити свій додаток безкоштовним для завантаження, що значно полегшує процес залучення нових користувачів, оскільки багато людей надають перевагу безкоштовним іграм. Після того, як гравець стає активним користувачем, можна почати монетизувати додаток через різні мікротранзакції, такі як покупки в грі для поліпшення ігрового досвіду, покупки косметичних предметів, підписки або доступ до додаткового контенту [29].

Типи внутрішньоігрових покупок:

– розхідні матеріали. Розхідні матеріали — це продукти, які можна використати або «спожити» один раз, а потім їх потрібно купляти знову. Наприклад віртуальні валюти, що використовуються для покупки внутрішньоігрових предметів, а також різні бустери, які покращують ігровий процес, наприклад, бонуси або спеціальні здібності;

– не розхідні матеріали. Це товари, придбані один раз, без терміну придатності, які залишаються доступними назавжди. Наприклад, косметичні предмети (скіни, теми), бонуси, додаткові життя, контент-паки.

Переваги:

– пропонує прямий дохід від покупок, орієнтованих на залучених користувачів;

– персоналізує досвід, дозволяючи гравцям контролювати свої витрати в грі.

Недоліки:

– низький відсоток гравців, які зазвичай здійснюють покупки;

– може призвести до сприйняття середовища «плати, щоб виграти», якщо предмети суттєво впливають на ігровий процес.

2.1.4 Внутрішньоігрова реклама

Внутрішньоігрова реклама — це стратегія монетизації, за якої мобільні додатки показують рекламу для отримання доходу. Ця реклама може

відображатися у різних форматах які дозволяють монетизувати свою гру у збалансований, гнучкий спосіб, зберігаючи при цьому позитивний користувацький досвід [29].

Формати внутрішньоігрової реклами:

– нативна реклама;

– проміжна реклама;

– банерна реклама;

– реклама з винагородою;

– стіна пропозицій;

– ігрові оголошення.

Далі, а саме після вибору мережі реклами ми детально поговоримо про кожний з цих форматів внутрішньоігрової реклами.

Переваги внутрішньоігрової реклами:

– дозволяє безкоштовно завантажувати ігри, залучаючи більше гравців;

– реклама з винагородою пропонує внутрішньоігрові стимули, збільшуючи кількість переглядів реклами, не створюючи при цьому конфліктних ситуацій.

Недоліки внутрішньоігрової реклами:

– надмірне використання реклами може призвести до розчарування користувачів і змусити гравців покинути гру;

– втома від реклами може з часом знизити її ефективність.

2.1.5 Гібридна стратегія

Гібридна стратегія вирізняється тим, що поєднує в собі кілька стратегій монетизації одночасно. Можна поєднувати внутрішньоігрові покупки, внутрішньоігрову рекламу та підписки, залучаючи різні сегменти гравців і максимізуючи потоки доходів. Таким чином, одна модель слугує основним джерелом доходу, а інші моделі доповнюють її [29].

Переваги:

– гнучкість, що дозволяє задовольнити різні вподобання гравців і максимізувати дохід;

– зменшує залежність від одного методу монетизації, збалансовуючи потоки доходів.

Недоліки:

– складність балансування між рекламою, покупками та оновленням контенту;

– може відштовхнути гравців, якщо не враховувати досвід гравців.

Проаналізувавши всі стратегії монетизації та їх переваги та недоліки, для цієї гри було обрано саме внутрішньоігрову рекламну стратегію монетизації.

Основні причини цього вибору наступні:

– низький бар'єр для входу. Внутрішньоігрова реклама вимагає менших налаштувань порівняно, наприклад зі стратегією внутрішньоігрових покупок у грі, що полегшує інтеграцію та початок отримання прибутку без необхідності використання складних бекенд-систем;

– ідеально підходить для вибраного жанру — аркади. Аркадні ігри мають короткі, повторювані ігрові сесії, що робить їх добре відповідними для проміжної реклами або реклами з винагородою, яка може з'являтися під час природних перерв або заохочувати гравців, не перериваючи ігровий процес;

– доступність для гравців. На відміну від внутрішньоігрових покупок, які можуть обмежувати доступ для гравців, які не бажають або не можуть витрачати гроші, внутрішньоігрова реклама дозволяє кожному насолоджуватися грою без обмежень, підвищуючи рівень утримання користувачів та їх задоволеність;

– потенціал доходу. Внутрішньоігрова реклама, особливо з винагородою, може приносити стабільний дохід, особливо в регіонах, де прямі витрати можуть бути нижчими, таким чином розширюючи охоплення та прибутковість на різних ринках;

– масштабованість. Внутрішньоігрова реклама дає можливість гнучко регулювати частоту та розміщення реклами на основі зворотного зв'язку, оптимізуючи користувацький досвід та баланс доходів у міру зростання бази гравців.

Стратегія внутрішньоігрової реклами — лише початок. Зі збільшенням

кількості гравців, у майбутньому, можна буде впровадити додаткові стратегії монетизації, наприклад, стратегію внутрішньоігрових покупок, щоб збільшити потенційний дохід. Внутрішньоігрові покупки також добре вписуються в жанр аркади, пропонуючи гравцям можливість купувати ексклюзивний контент, косметику, бонуси — все це може збагатити ігровий досвід гравців, не порушуючи основного ігрового процесу. Такий поетапний підхід дозволить відстежувати вподобання гравців та їхню зацікавленість, не ускладнюючи початковий реліз.

2.2 Аналіз рекламних мереж

У сучасному світі мобільних ігор рекламні мережі відіграють вирішальну роль у монетизації ігор, забезпечуючи міст між розробниками та рекламодавцями, які прагнуть охопити цільову аудиторію. Вони дають змогу розробникам заробляти гроші, показуючи рекламу користувачам. Зазвичай мережа надає рекламу, керує ставками та оптимізує доставку реклами, щоб підвищити залученість і дохід. Розробникам платять за покази, кліки або дії, залежно від типу оголошення та цілей рекламодавця [30].

Ось деякі з найпоширеніших рекламних мереж:

– AppLovin;

– InMobi;

– Unity Ads;

– IronSource;

– Google AdMob.

Далі проаналізуємо кожну з них окремо.

2.2.1 Рекламна мережа AppLovin

AppLovin — відома мережа мобільної реклами, яка широко використовується розробниками та видавцями, особливо в індустрії мобільних ігор. На рисунку 1.8 зображено її логотип. Заснована у 2012 році, AppLovin надає інструменти для

монетизації та розширення своєї користувацької бази. Вона значно зросла завдяки придбанню інших рекламних платформ, ігрових студій та аналітичних компаній, що зробило її надійною екосистемою для монетизації додатків [31].

Рисунок 2.1 — Логотип «AppLovin» [32]

Основні особливості AppLovin:

– платформа медіації MAX. MAX від AppLovin — це її рішення медіації, яке дозволяє керувати кількома рекламними мережами, максимізуючи дохід, використовуючи конкурентні торги в режимі реального часу (in-app bidding). Воно оптимізує розміщення оголошень і час проведення аукціонів, збільшуючи eCPM (ефективну ціну за тисячу показів);

– інструменти для залучення користувачів. AppLovin пропонує рішення для залучення користувачів, які використовують машинне навчання для ефективного націлювання та залучення високоякісних користувачів. Розробники отримують доступ до аналітики в реальному часі та набору інструментів для оптимізації;

– AppDiscovery. Інструмент залучення користувачів, з'єднує рекламодавців з високоякісними користувачами, таргетуючи потенційну аудиторію в широкій мережі додатків та допомагає розширювати своє охоплення [33].

Його основні переваги:

– комплексні рішення. AppLovin пропонує інтегровану платформу, що поєднує в собі інструменти для залучення користувачів, монетизації та аналітики, забезпечуючи універсальне рішення для розробників;

– передові технології. Двигун AXON використовує велику кількість даних для підвищення ефективності реклами та залучення користувачів;

– велика мережа. Доступ до широкої мережі рекламодавців і паблішерів забезпечує широке охоплення та різноманітність рекламних оголошень [34].

Його основні недоліки:

– складність. Широкі можливості платформи можуть бути складними для освоєння новими користувачами;

– залежність доходу від ефективності реклами. Оскільки AppLovin покладається на конкурентні торги, коливання доходу можуть відбуватися через зміну попиту на рекламу, що впливає на стабільність потоку доходу;

– конкуренція на ринку. Працює в конкурентному просторі з іншими великими рекламними мережами, такими як Google AdMob та Meta Audience Network [35];

– плата за послуги. Залежно від використовуваних послуг і продуктів, ціни та збори AppLovin можуть збільшуватися, що впливає на невеликі студії або інді-розробників, які не мають великих бюджетів.

2.2.2 Рекламна мережа InMobi

InMobi — це глобальна мережа мобільної реклами та одна з найбільших незалежних платформ для реклами в мобільних додатках. На рисунку 1.9 зображено її логотип. Заснована у 2007 році, вона пропонує рішення для рекламодавців, розробників додатків та видавців для налагодження зв'язку з мобільною аудиторією та її монетизації. InMobi надає рекламні послуги в різних форматах, включаючи банерну рекламу, нативну рекламу, інтерстиціальну рекламу, відеорекламу та мультимедійні матеріали [36].

Рисунок 2.2 — Логотип «InMobi» [37]

Деякі особливості InMobi:

– розширений таргетинг аудиторії. InMobi пропонує детальні можливості таргетингу на основі демографічних даних, місцезнаходження, інтересів, поведінки та типу пристрою. Це допомагає рекламодавцям охопити конкретну аудиторію;

– рекламні формати та креативність. InMobi підтримує широкий спектр форматів оголошень;

– монетизація на основі штучного інтелекту. InMobi використовує машинне навчання та штучний інтелект для оптимізації розміщення реклами, підвищуючи релевантність та ефективність оголошень шляхом аналізу поведінки та тенденцій користувачів;

– торги в режимі реального часу. Біржа InMobi пропонує торги в режимі реального часу, що дозволяє встановлювати конкурентні ціни та максимізувати потенційний дохід;

– інтеграція з SDK. InMobi SDK розроблений для легкої інтеграції з Android, iOS та Unity для безперебійної роботи користувачів.

Переваги InMobi:

– глобальне охоплення. InMobi працює у понад 190 країнах, що робить його ефективною платформою для рекламодавців та розробників додатків, орієнтованих на глобальну аудиторію;

– різноманітні формати оголошень. Від стандартних банерів до мультимедійних та інтерактивних оголошень, InMobi підтримує різноманітні формати оголошень, покращуючи користувацький досвід та залучення користувачів;

– розширені можливості таргетингу. Завдяки широким можливостям таргетингу InMobi, рекламодавці можуть охопити конкретні демографічні групи та поведінкові моделі, що допомагає максимізувати ефективність кампанії;

– прозорість та аналітика. Глибока аналітика допомагає паблішерам і рекламодавцям вимірювати ефективність кампаній, оптимізувати доходи та приймати рішення на основі даних.

Недоліки InMobi:

– розмір та продуктивність SDK. Деякі розробники відзначають, що InMobi SDK може бути важчим за інші, що може дещо вплинути на продуктивність програми, особливо на пристроях нижчого класу;

– проблеми з конфіденційністю. У минулому InMobi вже стикався з перевірками щодо дотримання конфіденційності. Хоча з того часу вони підвищили прозорість і дотримання вимог, конфіденційність залишається проблемою для деяких користувачів і розробників;

– складна звітність. Деякі рекламодавці вважають функції звітності складними, особливо якщо вони не знайомі з цифровими рекламними метриками [38].

2.2.3 Рекламна мережа Unity Ads

Unity Ads є частиною екосистеми Unity і дає змогу показувати рекламу у своїх іграх на різних платформах, включаючи iOS, Android і навіть деякі десктопні платформи. На рисунку 1.10 зображено її логотип. Вона спеціалізується на мобільних іграх, пропонуючи безперешкодний спосіб інтеграції реклами, яка виглядає природно в ігровому середовищі. Мережа в основному фокусується на відеорекламі, рекламі з винагородою та інтерактивному досвіді [39].

Рисунок 2.3 — Логотип «UnityAds» [40]

Основні переваги UnityAds:

– простота інтеграції. Нативна інтеграція Unity Ads в ігровий двигун Unity робить його надзвичайно зручним у плані впровадження реклами та керування нею без потреби у зовнішніх плагінах чи SDK;

– якість реклами та утримання гравців. Реклама з винагородою ефективно підвищує залучення та утримання, оскільки гравці добровільно взаємодіють з рекламою за винагороди, що сприяє позитивному досвіду від реклами;

– ефективна аналітика та звітність. Панель Unity пропонує комплексний аналіз продуктивності реклами, залучення користувачів і дані щодо доходів, що допомагає оптимізувати розташування реклами та стратегії;

– призначена для ігор. Оскільки Unity Ads є частиною платформи Unity, вона спеціально орієнтована на розробників ігор, а її функції більш пристосовані до ігрового середовища;

– підтримка кросплатформності. Unity Ads працює на iOS, Android та навіть деяких настільних платформах, що робить її універсальною для кросплатформного розроблення ігор [41].

Недоліки UnityAds:

– обмежена підтримка для ігор не на Unity. Хоча Unity Ads можна інтегрувати в ігри, створені не на Unity, це менш інтуїтивно та може вимагати додаткових кроків або зовнішніх SDK;

– нижчий дохід. Деякі розробники повідомляють про нижчий дохід за показ у порівнянні з іншими рекламними мережами, такими як AdMob, особливо в регіонах з низькими eCPM (ефективна вартість за тисячу показів);

– контроль якості реклами. У розробників обмежений контроль над типами показуваної реклами, і інколи можуть з’являтися оголошення, що не відповідають тематиці гри або цільовій аудиторії, що може порушити досвід користувача;

– залежність від Unity. Оскільки Unity Ads тісно інтегрована з рушієм Unity, розробникам, які переходять на інший рушій, може бути важко підтримувати налаштування реклами [41].

2.2.4 Рекламна мережа IronSource

IronSource — це рекламна мережа, що спеціалізується на мобільних іграх та додатках і пропонує рішення для монетизації й просування. На рисунку 1.11 зображено її логотип. Вона популярна серед розробників мобільних ігор завдяки широкому функціоналу для інтеграції рекламних форматів та аналітики, що

дозволяє підвищити дохід від реклами та залучення користувачів. IronSource була заснована у 2010 році та зарекомендувала себе як один із провідних постачальників рішень для монетизації. У 2021 році IronSource об’єдналася з компанією Unity, що дало змогу пропонувати ще ширші можливості інтеграції реклами безпосередньо в середовищі розробки Unity [42].

Рисунок 2.4 — Логотип «IronSource» [43]

Переваги IronSource:

– зручність у використанні. Інтуїтивно зрозумілий інтерфейс, який полегшує інтеграцію реклами та управління кампаніями;

– підтримка медіації. Можливість роботи з кількома рекламними мережами з однієї панелі;

– інтеграція з Unity. Для розробників на Unity це рішення зручне, оскільки підтримує нативну інтеграцію;

– вищий дохід завдяки торгам у додатку. Використовуючи аукціонну модель, IronSource забезпечує кращі ціни на рекламу, що часто призводить до більшого доходу;

– надійні рекламні формати. Відеореклама з винагородою, вставки та банери оптимізовані для покращення користувацького досвіду та максимального залучення.

Недоліки IronSource:

– висока конкуренція. У популярних жанрах мобільних ігор конкуренція за покази реклами може зменшити дохід;

– можливі проблеми з модерацією. Іноді реклама може не підходити за змістом для певної аудиторії, через що потрібний додатковий контроль;

– нерівномірність заповнення реклами в різних регіонах. IronSource може мати обмежений інвентар реклами в певних регіонах, що призводить до низьких коефіцієнтів заповнення або менш оплачуваних оголошень [44].

2.2.5 Рекламна мережа Google AdMob

Google AdMob — це рекламна мережа, розроблена спеціально для мобільних додатків, яка дозволяє монетизувати свої застосунки шляхом показу реклами. На рисунку 1.12 зображено її логотип. Завдяки AdMob розробники можуть заробляти, інтегруючи рекламу безпосередньо в мобільні додатки на платформах Android та iOS. Платформа також надає функції медіації реклами, аналітики користувачів та доступу до великого інвентарю рекламодавців через Google Ads. Google придбала AdMob у 2010 році, і з того часу він став однією з найпопулярніших мобільних рекламних мереж у світі. AdMob пропонує різні типи реклами, такі як банерна реклама, міжсторінкова реклама, реклама з винагородою та нативна реклама. AdMob дозволяє розробникам інтегрувати рекламу через SDK (Software Development Kit), що спрощує розміщення реклами в додатках з використанням єдиної платформи для управління рекламою та аналітики [45].

Рисунок 2.5 — Логотип «Google AdMob» [46]

Основні можливості Google AdMob:

– формати реклами. Підтримує декілька форматів реклами, включаючи банери, міжсторінкову рекламу, реклама з винагородою та нативну рекламу, що дозволяє гнучко налаштовувати подання реклами;

– медіація реклами. AdMob дозволяє працювати з різними рекламними мережами, а не тільки Google Ads. Через медіацію AdMob заповнює рекламу з інших мереж, оптимізуючи дохід шляхом пріоритету найвищої ціни за показ;

– аналітика користувачів. Забезпечує доступ до Firebase Analytics для отримання даних про поведінку користувачів, демографію, залучення та утримання в додатку;

– оптимізація доходу. Алгоритми AdMob використовують машинне навчання для оптимізації розміщення реклами та доходу;

– крос-платформенна підтримка. Підтримує Android та iOS, що дозволяє монетизувати додатки на різних операційних системах через єдину платформу [47].

Переваги Google AdMob:

– легка інтеграція та використання. SDK зручний для користувача і добре інтегрується з Android Studio, Xcode, Unity та іншими, тому розробники можуть легко налаштувати рекламу у своїх додатках;

– великий інвентар реклами. Оскільки Google AdMob підтримується Google Ads, він має доступ до одного з найбільших інвентарів реклами у світі. Це означає більше реклам, вищі коефіцієнти заповнення та більше отримання вищого доходу;

– розширене таргетування та персоналізація. Google використовує дані користувачів і машинне навчання для показу максимально релевантної реклами, що покращує показник кліків та залучення;

– безкоштовні інструменти аналітики. Інтеграція з Firebase забезпечує потужні аналітичні та трекінгові інструменти для розуміння поведінки користувачів та покращення продуктивності додатка;

– медіація реклами та оптимізація доходу. Можливості медіації AdMob дозволяють працювати з кількома рекламними мережами, збільшуючи коефіцієнти заповнення та потенціал доходу [47].

Недоліки Google AdMob:

– поділ доходу та дохід від реклами. AdMob утримує частину доходу від реклами, тому розробники не отримують 100% доходу від реклами;

– вплив на користувацький досвід. Реклама, особливо нав’язливі формати, такі як міжсторінкова, може негативно впливати на досвід користувача, що може призвести до видалення або відмови від використання додатка;

– вимоги щодо дотримання політики. Google має суворі правила щодо контенту та конфіденційності користувачів. Розробники повинні дотримуватися правил Google щодо збору даних, персоналізованої реклами та контенту додатка, що може обмежити показ реклами та типи додатків, які можуть брати участь;

– обмежена можливість кастомізації. Хоча AdMob пропонує різні формати реклами, є певні обмеження в налаштуванні, особливо з нативною рекламою, що може бути проблемою для розробників, які прагнуть унікального дизайну реклами.

2.2.6 Медіація

Кожна рекламна мережа працює самостійно, а це означає, що якщо треба інтегрувати кілька мереж, доводиться керувати кожною мережею окремо. Це містить у собі роботу з декількома SDK (наборами для розробки програмного забезпечення), наповнення рекламних місць контентом, відстеження ефективності реклами та забезпечення безперебійного рекламного потоку в грі. Попри свою ефективність, такий процес може бути громіздким, особливо через те, що мережі іноді не надають рекламу за кожним запитом, що призводить до втрачених можливостей для отримання прибутку.

Саме тут медіація стає корисною. Медіація — це рівень, який знаходиться між грою та кількома рекламними мережами, працюючи як хаб для з'єднання та управління ними з однієї платформи. Інтегруючи платформу медіації (наприклад, AdMob Mediation, MAX від AppLovin або IronSource Mediation), можна оптимізувати дохід від реклами та уникнути проблем, пов'язаних з роботою безпосередньо з окремими мережами.

Медіація створює конкурентне середовище серед рекламних мереж. Замість того, щоб обирати одну рекламну мережу для показу оголошень, платформи медіації дають змогу кільком мережам змагатися за цю можливість у режимі реального часу. Мережа, яка запропонувала найвищу ціну, розміщує оголошення, гарантуючи найвищий можливий дохід за кожне розміщення [48].

Ось як платформа медіації оптимізує доставку реклами та максимізує дохід:

– запит на ставку та рекламні аукціони. Коли з'являється місце для розміщення реклами, платформа медіації надсилає запит до підключених рекламних мереж. Кожна мережа робить ставку на показ оголошення, а платформа обирає мережу, яка пропонує найвищу ціну за показ (CPM — ціна за 1000 показів);

– резервна реклама. Якщо мережа не може надати оголошення (через проблеми з інвентарем або інші фактори), медіація швидко переходить до наступного учасника, який запропонував найвищу ціну, гарантуючи, що оголошення завжди буде показано, зменшуючи час простою та недоотриманий дохід;

– дані про ефективність реклами. Платформи медіації надають інформацію про ефективність роботи кожної мережі, включаючи коефіцієнт заповнення, eCPM (ефективну ціну за клік) та показники залученості користувачів. Ці дані допомагають оптимізувати розміщення оголошень і визначати пріоритети мереж, які стабільно показують хороші результати;

– автоматизоване управління рекламною мережею. Замість того, щоб інтегрувати SDK для кожної рекламної мережі окремо, посередницькі платформи часто виконують інтеграцію в межах свого інтерфейсу. Це економить час і спрощує налаштування [48, 49].

Медіацію добре використовувати, коли:

– є високий показник DAU (Daily Active Users). Ігри з великою базою користувачів отримують більше користі від конкурентних торгів між мережами, що максимізує потенціал доходу від реклами;

– глобальне охоплення. Медіація дозволяє мережам спеціалізуватися на певних регіонах. Якщо гра має міжнародну аудиторію, ця платформа може використовувати мережі, популярні в кожному регіоні, збільшуючи заповнюваність і релевантність реклами;

– потреби в оптимізації доходів. Якщо дохід від реклами є основним пріоритетом, медіація може суттєво вплинути на ситуацію, створюючи конкурентне середовище для торгів між рекламними мережами.

Однак, якщо гра має низьку базу користувачів або більше покладається на внутрішні покупки, ніж на рекламу, медіація може не принести суттєвих переваг. У таких випадках буде достатньо зосередитися на одній високопродуктивній мережі, оскільки зусилля та ресурси, необхідні для управління медіацією, можуть переважати над вигодами від доходу.

Після аналізу наявних мереж реклами, їх переваг та недоліків, та медіації, для цього проєкту було обрано рекламну мережу Google AdMob без використання медіації.

Основні причини та аргументи, що вплинули на вибір саме цієї рекламної мережі:

– простота інтеграції та надійність. Google AdMob є дружнім до розробників, з простою інтеграцією SDK в Unity, що робить його ідеальним для тих, хто вперше починає займатися монетизацією. Google також надає потужну, послідовну підтримку з регулярними оновленнями, які забезпечують сумісність SDK з останніми вимогами Android та мінімізують потенційні проблеми зі стабільністю роботи додатків, що є важливим для безперебійної роботи гравців;

– високий коефіцієнт заповнення та конкурентоспроможна ціна за клік. Google AdMob пропонує один з найвищих коефіцієнтів заповнення у світі, що гарантує, що більшість рекламних запитів генерують рекламу, особливо цінну для високочастотного розміщення оголошень в аркадних іграх. Крім того, ціна за клік в AdMob є конкурентоспроможною, що може бути особливо корисним для аркадних ігор з частими ігровими сесіями. Без медіації ми уникаємо додаткових складнощів в управлінні кількома рекламними мережами, водночас отримуючи надійний дохід від високоякісного рекламного інвентарю Google;

– якість реклами та користувацький досвід. Google дотримується суворих стандартів якості реклами, що допомагає забезпечити відповідність реклами, яка показується у вашій грі, очікуванням користувачів і стандартам безпеки, забезпечуючи бездоганний досвід. Наявність лише однієї мережі зменшує ймовірність появи небажаної, що сприяє утриманню гравців, а це дуже важливо в аркадних іграх з короткими та частими ігровими сесіями;

– потенціал майбутнього зростання з Google. Якщо згодом буде потрібно розширитися до медіації, AdMob пропонує простий спосіб інтегрувати інші мережі у свою платформу медіації. Однак поки що відмова від медіації спрощує розвиток і дозволяє зосередити зусилля на розумінні основних принципів монетизації.

Такий вибір особливо підходить для першої гри, оскільки поєднує в собі ефективність монетизації з низькою складністю та мінімальним впливом на ігровий процес. Це чудовий вибір, щоб збалансувати простоту та ефективність при монетизації мобільної аркади.

2.3 Аналіз форматів внутрішньоігрової реклами Google AdMob

У минулому розділі для монетизації нашої мобільної гри ми обрали рекламну мережу Google AdMob.

Google AdMob підтримує наступні формати внутрішньоігрової реклами:

– банерна реклама (banner);

– проміжна реклама (interstitial);

– реклама з винагородою (rewarded);

– проміжна реклама з винагородою (rewarded interstitial);

– просунута нативна (native advanced);

– при відкритті додатка (app open) [50].

Розглянемо кожний з цих форматів внутрішньоігрової реклами окремо.

Банерна реклама (banner).

Банерна реклама — прямокутне оголошення, яке зазвичай відображається у верхній або нижній частині екрана в додатку. Воно залишається на екрані, поки користувачі взаємодіють з додатком.

Переваги:

– ненав'язлива. Не заважає діям користувача, оскільки є статичною;

– постійна експозиція. Видима для користувачів протягом тривалого часу, не вимагаючи взаємодії.

Недоліки:

– низький рівень залучення. Користувачі часто ігнорують банерну рекламу, що призводить до зниження кількості кліків;

– може перекривати інтерфейс, особливо на невеликих екранах.

Проміжна реклама (interstitial).

Проміжна реклама — це повноекранна реклама, яка з'являється під час природних перерв, наприклад, після проходження рівня або коли гравець повертається до головного меню. Гравці можуть закрити їх через кілька секунд.

Переваги:

– вищий дохід за показ, ніж у банерної реклами, завдяки повноекранному формату;

– залучення, як правило, вище, оскільки гравці повинні взаємодіяти з рекламою, щоб продовжити гру.

Недоліки:

– нав'язлива, якщо розміщується занадто часто, оскільки перериває ігровий процес.

Реклама з винагородою (rewarded interstitial).

Реклама з винагородою — це необов'язкова реклама, яку гравці можуть вибрати для перегляду в обмін на ігрову винагороду (наприклад, додаткові життя, монети, бонуси).

Переваги:

– висока залученість, оскільки гравці добровільно дивляться цю рекламу;

– створює позитивний досвід, надаючи гравцям щось цінне натомість;

– вищий дохід на залучення завдяки формату згоди з переглядом.

Недоліки:

– обмежена кількість розміщень, їх можна показувати лише тоді, коли гравці, ймовірно, захочуть отримати винагороду;

– потребує збалансування винагород, щоб уникнути знецінення внутрішньоігрової валюти або нагород.

Проміжна реклама з винагородою (rewarded interstitial).

Поєднання реклами з винагородою та проміжної реклами. Проміжна реклама з винагородою не вимагає згоди гравця. Вона автоматично винагороджує гравця після перегляду реклами, не перериваючи при цьому потік.

Переваги:

– високий рівень залученості, оскільки гравці отримують винагороду без необхідності згоди;

– вищий дохід, ніж від банерної реклами, і може бути стратегічно розміщена без надмірного використання.

Недоліки:

– може заплутати гравців, які очікують контролю над переглядом;

– часте використання може призвести до втоми гравця, якщо винагороди занадто мінімальні або реклама занадто нав'язлива.

Просунута нативна (native advanced).

Ця реклама розроблена відповідно до візуального стилю гри, плавно інтегруючись з користувацьким інтерфейсом, що дозволяє їй не порушувати загальну атмосферу ігрового процесу. Таке поєднання робить рекламу менш нав'язливою для користувача, знижуючи ризик відчуття переривання гри.

Переваги:

– вища залученість, оскільки вони інтегровані в гру, що робить їх менш деструктивними;

– забезпечує більшу творчу гнучкість, покращуючи користувацький досвід і підтримуючи послідовність бренду.

Недоліки:

– складніша в розробці та реалізації, ніж стандартні формати реклами;

– ефективність може змінюватися, якщо гравці не помічають їх через непомітне розміщення.

При відкритті додатка (app open).

Ця реклама з'являється одразу, як тільки користувач відкриває додаток, і відображається у повноекранному режимі на весь початковий екран, доки користувач не закриє рекламу.

Переваги:

– хороший дохід від перших показів, оскільки вони залучають користувачів одразу після входу в додаток;

– не заважає ігровому процесу, оскільки з'являється лише один раз під час запуску програми.

Недоліки:

– може викликати певне розчарування, оскільки затримує доступ гравців до гри;

– слід використовувати обережно, щоб уникнути високого показника відмов, особливо для нетерплячих гравців.

Після огляду підтримуваних форматів внутрішньоігрової реклами Google AdMob, аналізу їх переваг та недоліків, для цієї гри було обрано два формати внутрішньоігрової реклами: проміжна реклама та реклама з винагородою. Вибір проміжної реклами та реклами з винагородою для мобільної аркади має очевидні переваги, пов'язані з максимізацією доходу при збереженні позитивного користувацького досвіду.

Вибір проміжної реклами та реклами з винагородою для мобільної аркади є стратегічним, оскільки пріоритетними є як користувацький досвід, так і генерування прибутку. Проміжна реклама природно вписується в ігровий процес, з'являючись в ідеальні моменти, наприклад, після проходження рівня або завершення гри, не перериваючи при цьому ігровий процес. Цей формат дозволяє демонструвати рекламу в повноекранному режимі, максимізуючи дохід за показ порівняно з банерами, які приносять низький дохід при частих коротких сесіях, характерних для аркадних ігор. Оскільки проміжна реклама з'являється лише під час пауз, вона підтримує чистий, не засмічений інтерфейс, що має вирішальне значення для збереження занурення в гру на невеликих мобільних дисплеях.

З іншого боку, реклама з винагородою пропонує гравцям можливість переглядати рекламу в обмін на винагороду, наприклад, для можливості перегравання рівня або для отримання додаткової внутрішньоігрової валюти, покращуючи ігровий досвід, надаючи гравцям контроль над взаємодією з рекламою. Такий підхід не лише підвищує залученість, але й сприяє позитивному сприйняттю реклами, оскільки гравці отримують відчутні ігрові переваги. В аркадних іграх, де гравці часто повторно проходять гру, реклама з винагородою органічно вписується, пропонуючи винагороду, яка сприяє утриманню гравців, оскільки користувачі заохочуються до тривалої гри.

Ці два формати є оптимальними для цієї аркадної гри, оскільки вони балансують між ефективною монетизацією та бездоганним користувацьким досвідом, максимізуючи як залученість, так і прибуток.

У підсумку проведеного аналізу у цьому розділі, для мобільної аркадної гри, що розробляється в рамках цієї роботи, була обрана стратегія внутрішньоігрової реклами з використанням рекламної мережі Google AdMob без медіації. Це рішення обумовлено низьким бар'єром для входу, надійністю та простотою інтеграції, що особливо важливо для першої монетизованої гри. Вибір Google AdMob також підтримується високим коефіцієнтом заповнення, конкурентною ціною за клік і дотриманням високих стандартів якості реклами, що забезпечує стабільний дохід без ускладнень від управління кількома мережами. Також було обрано два основних формати реклами: проміжну рекламу та рекламу з винагородою, які ідеально підходять для коротких і частих сесій у жанрі аркадних ігор. Проміжна реклама інтегрується у гру природно, з’являючись під час пауз, наприклад після завершення рівня, не порушуючи ігровий процес. Реклама з винагородою, зі свого боку, дозволяє гравцям обирати взаємодію з рекламою в обмін на винагороди, наприклад на такі як додаткові спроби чи бонуси, що покращує ігровий досвід і сприяє підвищенню залученості. Такий підхід дозволяє зберігати баланс між ефективною монетизацією і позитивним користувацьким досвідом, що є особливо важливим для зростання аудиторії та утримання гравців.

3 ПРОЄКТУВАННЯ ГРИ

Розробка концепції та проєктування є першими та найважливішими етапами в процесі розробки гри, адже саме тут закладаються фундаментальні принципи, які визначатимуть її унікальність, ігровий процес, та цільову аудиторію. На цьому етапі формується чітке уявлення про те, якою має бути гра, які емоції вона має викликати у гравця, які механіки будуть залучені та як вони інтегруватимуться одна з одною. Від того, наскільки детально і продумано буде розроблена концепція, залежить те, чи зможе гра привернути увагу аудиторії, чи матиме вона високу конкурентоспроможність і, чи буде гравцям цікаво повертатися до неї знову і знову.

Проєктування гри є надзвичайно важливим, оскільки воно задає чіткий напрямок і структуру. На цьому етапі проєктуються основні механіки, інтерфейс, функціонал проєкту. Чітко спроєктована ідея, механіки, функціонал і інтерфейс користувача допомагають уникнути плутанини в процесі створення та дозволяють ефективно розподіляти час і ресурси. Крім того, проєктування дає змогу краще планувати монетизацію та просування гри, що важливо для фінансового успіху. Усе це дозволяє зберегти організованість і впевненість у правильності обраного напрямку.

3.1 Розробка концепції гри

Розробка концепції гри — це критично важливий перший етап у створенні будь-якої гри. Вона закладає фундамент для всіх подальших етапів і визначає загальне бачення проєкт, його мету, цільову аудиторію, жанр, ключові механіки та стилістику. Грамотно пророблена концепція допомагає уникнути суттєвих помилок та витрат на пізніших етапах розробки.

3.1.1 Жанр гри

Оскільки метою створюваної гри є не тільки приємне проведення часу, а і певний розвиток навичок, то було обрано жанр аркади. Жанр аркадних мобільних ігор вирізняється простотою геймплею та короткими ігровими сесіями, що робить його ідеальним для мобільних платформ. У цих іграх зазвичай легко розібратися, але складність поступово зростає, викликаючи в гравця бажання грати ще раз і вдосконалювати свої навички. Багато аркадних ігор фокусуються на механіках швидкого реагування, майстерності або збирання очок.

Приклади популярних мобільних аркадних ігор: Subway Surfers, Fruit Ninja, Flappy Bird, Crossy Road та інші.

Основні особливості цього жанру:

– простий геймплей. Управління зазвичай інтуїтивно зрозуміле, з мінімальною кількістю дій, які гравець може виконувати;

– короткі сесії. Кожна сесія триває недовго, зазвичай кілька хвилин, що дозволяє грати в гру у будь-який зручний момент;

– поступове підвищення складності. Складність гри часто підвищується, мотивуючи гравців повернутися для нових спроб;

– мотивація для рекордів. Багато аркадних ігор мають системи рекордів, що спонукають гравців змагатися.

3.1.2 Основна ідея гри

Основа ідея гри полягає в тому, що гравцю генерується випадкова споруда з кубів і йому необхідно її заповнити у заданий час, розміщуючи куби у відповідних позиціях, і при цьому не перевищуючи допустимої кількості помилок. А саме, існує ліміт допустимих помилок — кількість кубів, розміщених в неправильних позиціях. Якщо гравець перевищить цей ліміт — він програє. Гра побудована на концепції рівнів. Після кожного успішного заповнення споруди, гравець переходить на новий рівень. З кожним наступним рівнем кількість кубів у споруді збільшується, а кількість допустимих помилок і час на заповнення споруди зменшуються, що підвищує складність гри та у той же самий час приносять гравцеві виклик, мотивацію, і робить ігровий процес більш захоплюючим. За

правильно розміщені куби гравець отримує очки, а швидкість виконання рівня приносить додаткові бонусні очки. Мета гри — набрати якомога більше очок, оскільки гра не має кінцевого рівня.

Крім того, гра містить внутрішньоігровий магазин, який надає гравцям елемент персоналізації та урізноманітнює ігровий процес. А саме, гравці можуть змінювати візуальний вигляд кубів за допомогою скинів. Деякі скіни відкриваються за досягнуті під час гри очки, інші — відкриваються за дорогоцінні камені, які можна збирати під час гри. Таким чином, гра не потребує внесення реальних коштів для купівлі внутрішньоігрових предметів, все можна відкрити просто граючи у гру. Внутрішньоігровий магазин додає додаткової мотивації для гравців грати в гру, мотивуючи гравців досягати якомога більшої кількості очок і збираючи дорогоцінні камені.

Загалом, гра розрахована на покращення різних когнітивних та фізичних навичок, таких як увага, концентрація, реакція та логічне мислення. Вона допомагає гравцям тренувати здатність швидко аналізувати ситуацію.

3.1.3 Цільова аудиторія гри

Цільова аудиторія гри — це група людей, на яку розрахована гра, з урахуванням їхнього віку, інтересів, переваг і рівня досвіду в іграх. Розуміти цільову аудиторію дуже важливо, оскільки це один з основних чинників успіху гри. Від розуміння цільової аудиторії залежать ключові аспекти розробки та просування гри.

Цільова аудиторія розроблюваної гри має широкий діапазон, а саме це казуальні мобільні гравці віком від 10 до 40 років, які цінують швидкі, захоплюючі ігри для коротких сеансів, але також люблять змагатися та досягати нових рекордів. Гра приваблює людей, які шукають викликів для розвитку уважності, концентрації, реакції та логічного мислення, а також підлітків і молодь, які цінують персоналізацію ігрового процесу через скіни та внутрішньоігрові нагороди. Основна мотивація аудиторії — покращувати свої результати, збирати очки, відкривати нові скіни та досягати високих показників, не витрачаючи реальні кошти.

3.1.4 Концепція інтеграції внутрішньоігрової реклами в ігровий процес

Для цього проєкту була обрана стратегія внутрішньоігрової реклами з використанням рекламної мережі Google AdMob без медіації застосовуючи два основних формати реклами: проміжну рекламу та рекламу з винагородою, які ідеально підходять для коротких і частих сесій у вибраному жанрі аркади.

Проміжна реклама буде показуватися гравцю відразу після програшу, але лише після того, як гравець зазнає п'яти програшів. Такий підхід розміщення проміжної реклами є хорошим рішенням з кількох причин:

– збереження безперервності гри. Показ рекламного оголошення після програшу, а не під час ігрового процесу, мінімізує втручання в ігровий процес. Гравець не втрачає концентрації, коли фокус повністю на грі. Це підвищує шанси, що він не покине гру через часте та нав'язливе втручання реклами.

– знижений рівень роздратування у користувача. Оскільки реклама показується лише після п’яти програшів, вона не сприймається як надокучлива. Таке налаштування дозволяє гравцю вільно поринути в гру і лише зрідка взаємодіяти з рекламою, що значно покращує користувацький досвід. До того ж п'ять спроб створюють своєрідний ритм, де гравець сприймає рекламний блок як природну паузу;

– емоційний ефект після програшу. Після кількох невдалих спроб, багато гравців роблять короткі перерви для відпочинку або аналізу своїх дій. Показ реклами в цей момент може слугувати цією паузою, дозволяючи гравцю перепочити й не відчувати, що реклама є надокучливою. В результаті, гравець має більше шансів сприйняти її позитивно або нейтрально;

– підвищення ефективності реклами. Коли гравець бачить рекламу у момент, коли він готовий зробити перерву, він, швидше за все, зверне на неї більше уваги. Це покращує імовірність взаємодії з рекламним контентом, що може збільшити прибуток від реклами;

– мінімальний ризик втрати гравця. Проміжна реклама, яка показується рідко і лише після кількох програшів, не викликає відторгнення у гравця. Вона не створює враження, що гра «переслідує» рекламою, тому гравець з меншою

ймовірністю покине гру через негативний досвід від нав’язливих оголошень.

Така стратегія показу проміжної реклами працює як компроміс між бажанням монетизувати гру і необхідністю підтримувати позитивний користувацький досвід, забезпечуючи гравцю безперервну взаємодію з грою та комфортні паузи з рекламою.

У сцені ігрового процесу, перегляд реклами з винагородою виступатиме можливістю для гравця перегратипрограний рівень. Після програшу гравцем (вийшов час або гравець перевищив ліміт допустимих помилок) перед вікном програшу, спочатку буде показуватися спеціальне вікно з пропозицією гравцеві перегратипоточний рівень в обмін на перегляд реклами. Гравець зможе обрати сам, чи хоче він дивитися рекламу, щоб продовжити чи ні. Якщо гравець захоче продовжити гру, щоб не втратити свій рекорд і продовжити підвищувати його далі, він може погодитися на перегляд реклами тапнувши на кнопку. Після перегляду реклами, гравцеві буде перезапущено поточний рівень, на якому гравець зупинився і гравець зможе продовжувати досягати безмежного рекорду. Якщо гравець відмовиться, спеціальне вікно з пропозицією закриється і з'явиться звичайне вікно з інформацією про програш. Для балансу гри, таке вікно з пропозицією може з'явитися тільки один раз за одну спробу.

Такий тип має декілька переваг для користувацького досвіду та залученості гравця:

– позитивний вплив на користувацький досвід. Гравці отримують вибір — продовжити програний рівень або відмовитися від перегляду реклами. Така можливість надає гравцям контроль, що є важливим аспектом хорошого користувацького досвіду.

– зниження розчарування після програшу. Програш зазвичай викликає розчарування у гравця, особливо якщо він наблизився до високого результату. Пропозиція переглянути рекламу в обмін на продовження рівня знижує цей негативний ефект і дає гравцю шанс продовжити свою гру, що сприяє емоційній прив’язаності до гри та довготривалому інтересу;

– мотивація досягати кращого результату. Можливість перегратирівень і

покращити свій рекорд додає стимулу для гравців. Це збільшує загальну залученість і мотивує до тривалих ігрових сесій, підвищуючи ігровий досвід без втрати ритму;

– економія ресурсів. Оскільки реклама з’являється лише один раз за спробу, це допомагає підтримувати баланс і уникати надмірної залежності від рекламних бонусів. Гравці сприйматимуть цю можливість як цінний ресурс, використовуючи його стратегічно і не відчуваючи надокучливості.

У внутрішньоігровому магазині, перегляд реклами з винагородою буде давати можливість гравцю отримати безкоштовні дорогоцінні камені. У магазині буде розміщена спеціальна кнопка, яка буде оформлена належним чином і даватиме розуміння гравцеві, що при натисканні на цю кнопку гравець може отримати додаткові дорогоцінні камені в обмін на перегляд реклами. Ця кнопка буде розміщена у комфортному місці, не перекриваючи основний інтерфейс гри й не створюючи візуальний безлад. Тобто якщо, наприклад, гравець зайшов у магазин і йому дуже сподобався якийсь скін, але йому трошки не вистачає дорогоцінних каменів для його покупки, але він хоче його придбати саме зараз, він може тапнути на цю спеціальну кнопку та подивитися рекламу в обмін на певну кількість (яка саме кількість, буде вирішено під час балансу гри) безкоштовних дорогоцінних каменів, щоб потім придбати бажаний скін.

Впровадження реклами з винагородою у внутрішньоігровому магазині, де гравці можуть отримати додаткові дорогоцінні камені, також є стратегічним кроком для підвищення монетизації і залученості:

– створення можливостей для прогресу. Коли гравець бачить бажаний скін, але йому не вистачає кількох дорогоцінних каменів, можливість отримати ці камені за перегляд реклами стає привабливою. Це дозволяє гравцям отримати те, що їм потрібно, не витрачаючи реальні гроші, що робить гру доступнішою та підвищує рівень задоволення від ігрового процесу;

– вибір без тиску. Розміщення спеціальної кнопки в магазині, яка пропонує безкоштовні дорогоцінні камені за перегляд реклами, не створює тиску на гравця. Це — опціональний спосіб отримати ресурс, який підсилює позитивний

користувацький досвід і сприяє лояльності до гри;

– покращення взаємодії з внутрішньоігровим магазином. Гравці частіше взаємодіятимуть з магазином, оскільки знають, що завжди є можливість отримати безкоштовні дорогоцінні камені. Це допомагає не тільки збільшити продажі, але й робить внутрішньоігровий магазин більш важливою частиною гри;

– інтуїтивний дизайн і комфорт для користувача. Спеціальна кнопка з пропозицією реклами розташована зручно, не перекриваючи основний інтерфейс, що забезпечує акуратний та органічний вигляд. Гравці з легкістю розуміють, де саме вони можуть скористатися рекламною можливістю, що сприяє зручності використання інтерфейсу.

Таке комбіноване рішення покращує загальний баланс реклами у грі та посилює ефективність монетизації, не створюючи відчуття нав’язливості:

– надає різноманітність способів взаємодії з рекламою. Завдяки обом типам реклами гравці отримують більше можливостей вибору, як вони можуть використати перегляд реклами для свого прогресу в грі, що підтримує їхню залученість та інтерес;

– позитивний емоційний вплив. Реклама з винагородою сприймається позитивніше, оскільки гравець отримує цінний бонус, особливо коли ця винагорода допомагає досягти кращих результатів або отримати бажані предмети. Це сприяє формуванню позитивного ставлення до реклами у грі.

Загалом, така стратегія сприяє гармонійному поєднанню монетизації та користувацького досвіду, зберігаючи задоволення від гри та дозволяючи гравцям насолоджуватися процесом без відчуття тиску.

3.2 Проєктування гри

Проектування гри — це один із найважливіших етапів у розробці ігрового продукту. Саме на цьому етапі вже формується ця унікальна ідея гри,

встановлюються її основні механіки, створюються правила, які визначають, як гравець буде взаємодіяти з ігровим світом, і розробляється концепція візуального стилю та атмосфери. Процес проектування охоплює широкий спектр завдань — від взаємодії гравця зі сценами до визначення ідеології й стилістичних аспектів, які допоможуть створити незабутній ігровий досвід.

Важливість проектування складно переоцінити: на цьому етапі закладається фундамент гри, який впливатиме на кожний наступний крок розробки, від програмування до тестування. Грамотно спроектована гра допоможе уникнути багатьох помилок і зберегти час, зусилля і ресурси.

3.2.1 Визначення загальної архітектури та структури проєкту

У цьому проєкті буде використовуватися поєднання Менеджерно-контролерної архітектури (Manager-Controller Architecture) та компонентної архітектури (Component-Based Architecture) Unity.

Менеджерно-контролерна архітектура (Manager-Controller Architecture) — це архітектура, у якій менеджери відповідають за глобальні аспекти гри (наприклад, керування рівнями, звуками, обробкою реклами), тоді як контролери здійснюють управління конкретними елементами чи групами об'єктів. Менеджери часто є єдиними і глобально доступними (зазвичай існують як Singleton-об'єкти), щоб надавати єдиний контроль над важливими аспектами гри. Контролери, у свою чергу, керують певними функціональними частинами, зазвичай тими, що залежать від стану гри чи потребують інтеракції з іншими об'єктами.

Компонентна архітектура (Component-Based Architecture) — ця архітектура підходить для модульності та повторного використання. Головна ідея полягає в тому, щоб розбити логіку об'єктів на незалежні компоненти. Кожен компонент відповідає за одну функцію (наприклад, рух, анімацію, фізику, інтерактивність) і може бути доданий або вилучений без впливу на інші компоненти.

Поєднання цих архітектур дозволяє досягти більшої гнучкості. Менеджери забезпечують централізоване керування ключовими аспектами гри, тоді як компоненти надають гнучкість і можливість швидкої адаптації конкретних об'єктів. Контролери, які керують групами об'єктів або певними ігровими

механіками, можна реалізувати як набори компонентів. Таким чином, контролери й менеджери задають контекст, а компоненти надають кожному об’єкту окремі функціональні частини.

Ефективність такої комбінації:

– висока масштабованість і легкість у внесенні змін;

– менеджери організують гру на глобальному рівні, уникаючи дублювання логіки, тоді як компоненти забезпечують модульність для об’єктів;

– спрощення тестування і внесення змін, оскільки компоненти можна легко заміняти й комбінувати.

Структура проєкту.

Проєктування будь-якої гри в Unity починається з організовування сцен і як вони будуть взаємодіяти між собою. Сцена в Unity — це основна одиниця для організації контенту та елементів гри. Це своєрідне «полотно», на якому розміщуються всі об'єкти, з якими взаємодіє гравець. Сцена в Unity — це місце, де зберігаються всі елементи конкретного рівня, меню чи етапу гри, і саме вона слугує основним будівельним блоком у структурі проєкту.

Сцени дозволяють організувати проєкт, розділивши його на логічні частини, кожна з яких має свій зміст, призначення і структуру. Це допомагає в управлінні складними ігровими процесами та забезпечує зручне налаштування окремих компонентів, з яких складається гра. На кожній сцені можна знайти різноманітні об’єкти, що створюють оточення, інтерфейс та інші елементи, які визначають взаємодію гравця з грою.

Об'єкти в Unity (GameObjects) є основними елементами будь-якої сцени та основними блоками, з яких складається гра. Це універсальні контейнери, які можуть представляти практично будь-який елемент ігрового світу: персонажів, предмети, звукові ефекти, освітлення, камери, точки спавну та інші елементи. Кожен об'єкт у Unity має певний набір компонентів, що визначають його поведінку, вигляд і взаємодію з іншими елементами гри.

Компонент в Unity — це модуль, який додає об'єкту певні функції, характеристики або поведінку. Компоненти є основним механізмом, що дозволяє

налаштовувати об’єкти в Unity та надає їм ті чи інші властивості, такі як можливість відображатися на екрані, взаємодіяти з іншими об’єктами або виконувати певну поведінку. Кожен об’єкт у Unity може мати один або кілька компонентів, які працюють разом, утворюючи комплексні ігрові елементи.

Приклади деяких основних компонентів Unity:

– Transform. Кожен об'єкт має компонент Transform, який визначає його позицію, обертання та масштаб у світі або локальному просторі. Це базовий компонент, який присутній на всіх об'єктах і не може бути видалений;

– Mesh Renderer. Використовується для відображення 3D-моделей в сцені. Він працює разом з компонентом Mesh Filter, який містить саму модель, і Material, який визначає її вигляд (текстури, кольори, шейдери);

– Rigidbody. Додає фізичні властивості об'єкту, дозволяючи йому підкорятися законам фізики. Це дозволяє об'єктам рухатися, обертатися, падати під дією сили тяжіння та взаємодіяти з іншими фізичними об'єктами;

– Collider. Використовується для визначення області, в межах якої об'єкт може взаємодіяти з іншими об'єктами, наприклад, при зіткненнях. Існують різні типи колайдерів, зокрема Box Collider, Sphere Collider, Mesh Collider;

– Audio Source. Відповідає за відтворення звуків. З ним можна налаштувати 2D або 3D-звуки, їхню гучність, просторові параметри, циклічність та інші ефекти;

– Camera. Компонент, що відповідає за рендеринг сцени на екрані. Він визначає точку зору гравця, з якої відображається світ гри, і може бути налаштований для різних типів камери (наприклад, перспективної або ортографічної);

– Script. Компоненти, що додаються для реалізації поведінки об'єкта за допомогою коду на C#. Вони можуть змінювати властивості об'єкта, реагувати на події та взаємодіяти з іншими елементами гри.

Цей проєкт буде поділений на чотири сцени. навчальна сцена (сцена з інструктажем) та три основні сцени:

– навчальна сцена (tutorial scene);

– початкова сцена або меню (menu scene);

– сцена ігрового процесу (gameplay or main scene);

– сцена внутрішньоігрового магазину або ігровий магазин (shop scene).

На рисунку 3.1 зображена загальна структура сцен, на якій показано, як гравець може взаємодіяти між ними.

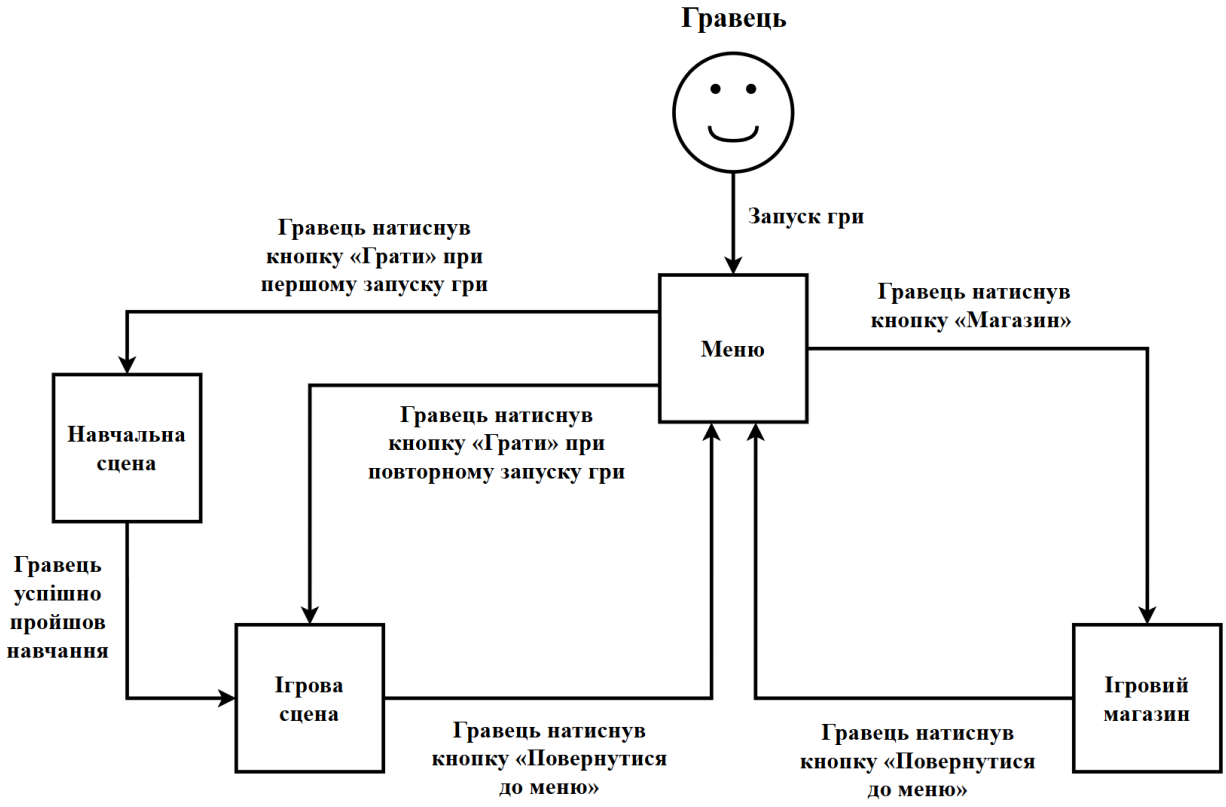

Рисунок 3.1 — Структура сцен

При найпершому запуску гри, при натисканні кнопки «Грати» зі сцени меню, гравець буде потрапляти до навчальної сцени, де він познайомиться з усіма основними аспектами гри: від управління до механік ігрового процесу. Після успішного ознайомлення з основними аспектами гри, гравець потраплятиме одразу у сцену ігрового процесу, щоб закріпити отримані навички. При наступних запусках гри гравець буде потрапляти в меню. З меню гравець може перейти до ігрової сцени (щоб почати гру) або до ігрового магазину, де він може подивитися наявні скіни на куби і вибрати бажаний. З ігрової сцени та ігрового магазину, гравець може повернутися назад до меню.

3.2.2 Навчальна сцена

Навчальна сцена буде зістрічати гравця при натисканні кнопки «Грати» зі сцени меню при найпершому запуску гри. На рисунку 3.2 зображено макет навчальної сцени.

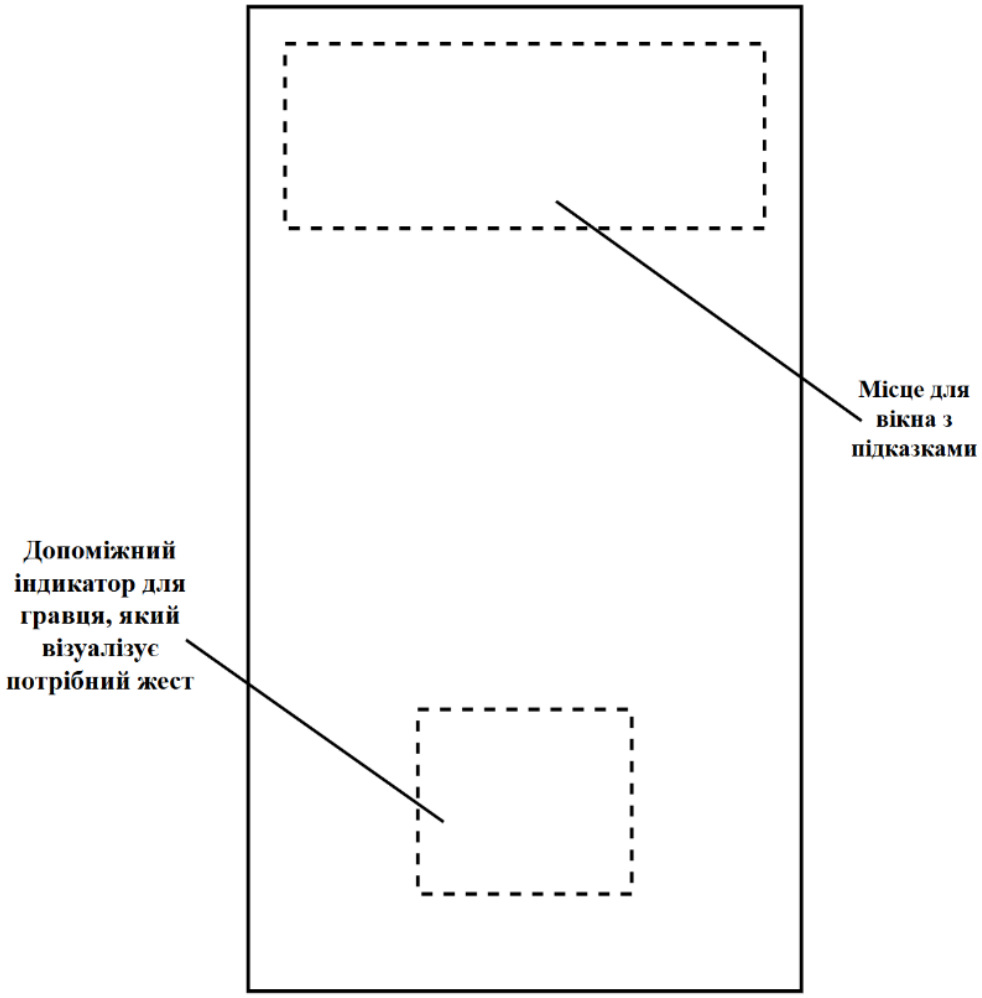

Рисунок 3.2 — Макет навчальної сцени

Знизу (див. рисунок 3.2) з'являтиметься інтуїтивно зрозумілий індикатор, який демонструватиме гравцеві потрібний жест для виконання якоїсь дії. А зверху (див. рисунок 3.2) показуватимуться вікна з підказками, спрощуючи процес навчання гравця. Буде два типи вікна з підказками: які вимагають підтвердження гравця (для засвідчення, що гравець точно розібрався) для закриття вікна і які не вимагають підтвердження гравця. Прототипи цих вікон зображено на рисунках 3.3 та 3.4 відповідно.

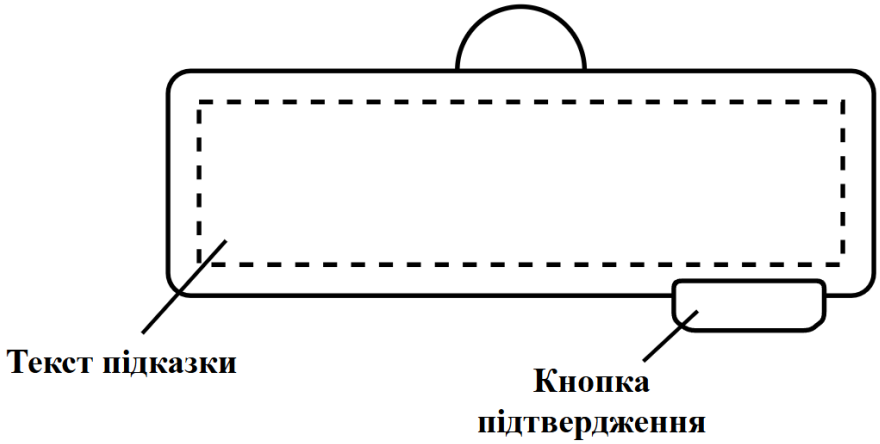

Рисунок 3.3 — Макет вікна з підказкою з кнопкою підтвердження

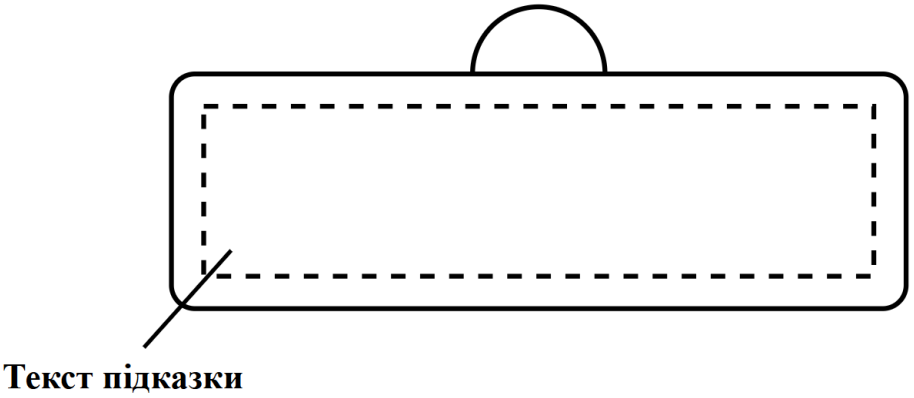

Рисунок 3.4 — Макет вікна з підказкою без кнопки підтвердження

Навчальна сцена створена для м'якого занурення гравця в ігровий процес, поступово знайомлячи його з основними елементами управління грою, основними механіками та правилами гри, щоб дати розуміння всіх дій, які можна виконувати під час гри. Кожен етап навчання буде супроводжуватися підказками які допоможуть гравцю без зусиль засвоювати інформацію. Завершивши навчання, гравець потрапляє у справжній ігровий процес, де може закріпити свої навички в умовах реальної гри. Надалі, при кожному запуску, гравець починатиме з меню.

3.2.3 Сцена меню

Сцена меню буде першою сценою для гравця, яку він зустрічатиме під час заходу в гру. меню являється головним хабом гри, через який гравець зможе легко переходити між основими сценами: ігровою сценою й ігровим магазином. На рисунку 3.5 зображено макет сцени меню.

Меню буде містити 4 кнопки (див. рисунок 3.5):

– кнопка з налаштуваннями;

– кнопка «Почати гру»;

– кнопка «Перейти у ігровий магазин»;

– кнопка «Про гру».

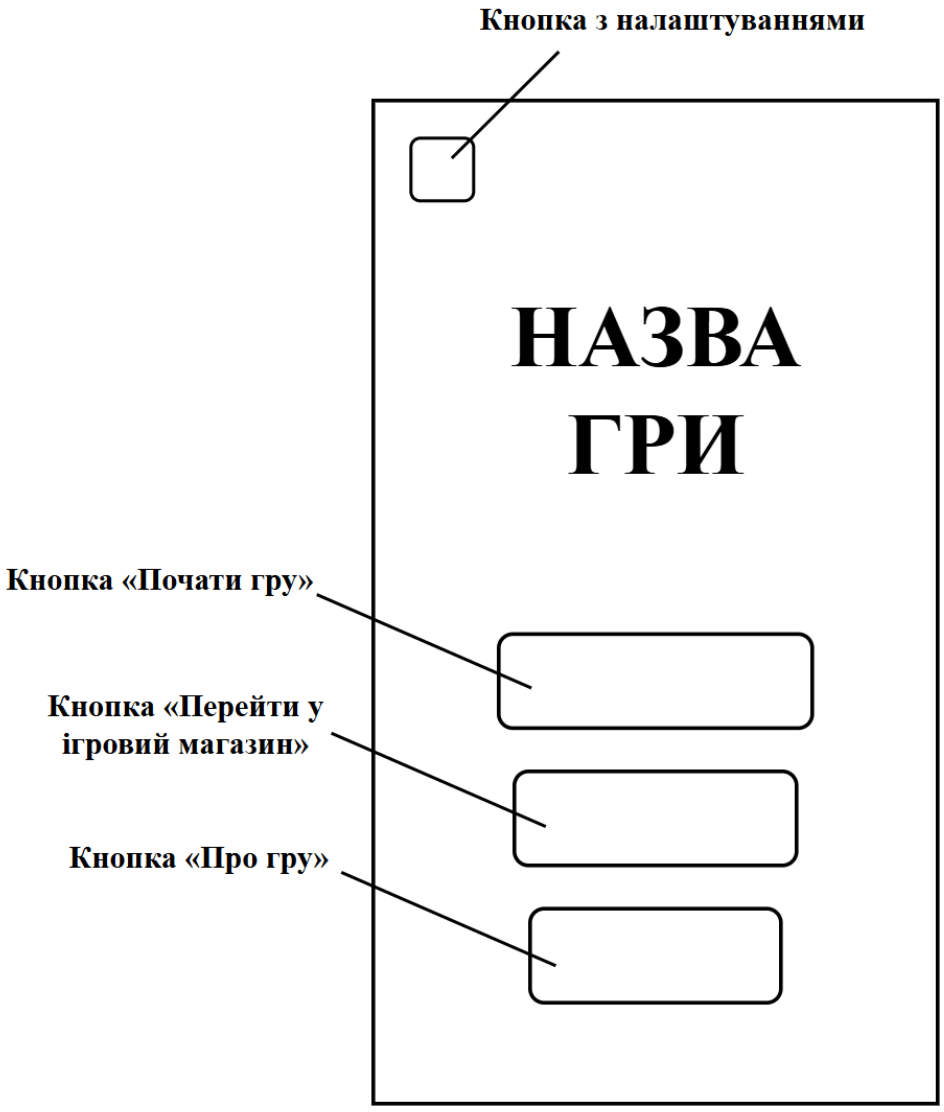

Рисунок 3.5 — Макет сцени меню

Кнопка налаштувань (див. рисунок 3.5) буде відкривати вікно з налаштуваннями, де гравець зможе за допомогою слайдерів окремо регулювати гучність «музики» та «звукових ефектів». Макет такого вікна зображений на рисунку 3.6.

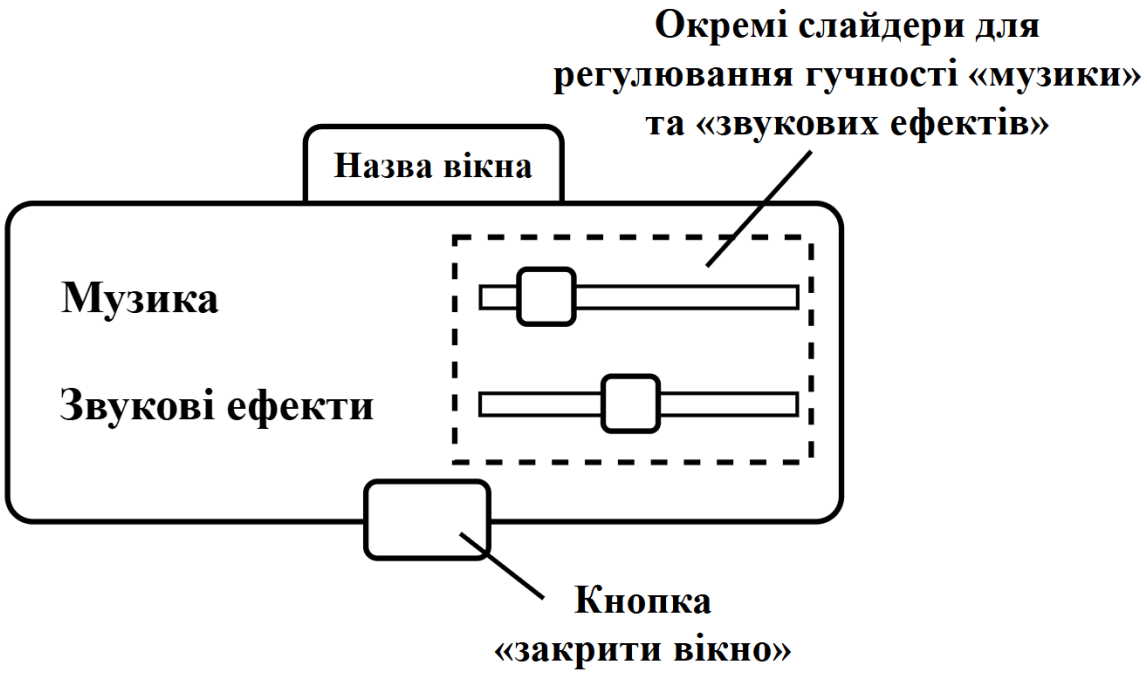

Рисунок 3.6 — Макет вікна з налаштуваннями гучності

Кнопка «Про гру» (див. рисунок 3.5) буде відкривати вікно з основною інформацією про гру, а саме з ім'ям розробника гри та автора музики. Макет вікна «Про гру» зображений на рисунку 3.7.

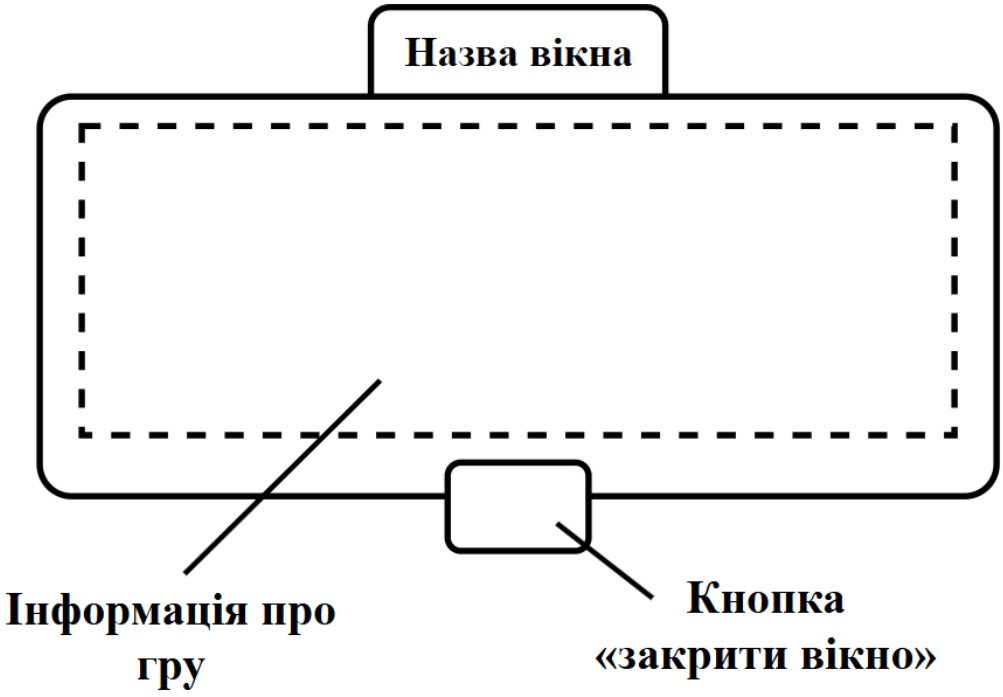

Рисунок 3.7 — Макет вікна «Про гру»

А за допомогою кнопки «Почати гру» (див. рисунок 3.5), гравець зможе зануритися в ігровий процес, а кнопка «Перейти в ігровий магазин» (див. рисунок 3.5) спрямує гравця у прекрасний світ магазину, де на нього чекає величезна кількість барвистих скінів, здатних прикрасити ігровий процес та надати йому унікального стилю.

Оскільки сцена меню відіграє важливу роль у формуванні першого враження, важливо приділити увагу її дизайну та додати унікальності. Для цього була придумана наступна ідея — фоном меню буде виступати довільна споруда з кубів, скін яких буде змінюватися випадковим чином кожен раз, як гравець буде переходити до меню. Також буде розроблена анімація для камери, за допомогою якої, вона буде красиво пролітати навколо цієї споруди, надаючи динамічності до сцени. Крім того, у меню буде розміщено назву гри, для якої теж буде розроблена анімація для додаткової динамічності. Таких підхід надасть унікальності, зацікавить гравця і створить приємне перше враження.

3.2.4 Сцена ігрового процесу

Сцена ігрового процесу — це основна сцена гри, в якій гравець буде проводити найбільшу кількість часу. На початку, на сцені буде розміщений один куб і навколо цього куба буде згенерована випадкова споруда. Далі починатиметься відлік до початку гри, під час якого гравець зможе ознайомитися зі згенерованою спорудою. Після відліку буде запущено генератор позицій навколо початкового куба, а також таймер для поточного рівня, за який гравцеві потрібно заповнити згенеровану споруду. Гравцю потрібно вибирати правильну позицію та тапати на екран для того, щоб розмістити новий куб. Як ми говорили під час обдумування концепції, задача гравця заповнити згенеровану споруду у визначений час, та на перевищуючи допустиму кількість кубів розміщених в неправильних позиціях. Після успішного заповнення споруди, гравець буде переходити на наступний рівень. З кожним рівнем складність буде збільшуватися, а саме кубів у споруді буде генеруватися більше, а допустима кількість кубів розміщених в неправильних позиціях та час на заповнення споруди — менше.

На рисунку 3.8 зображено макет сцени ігрового процесу.

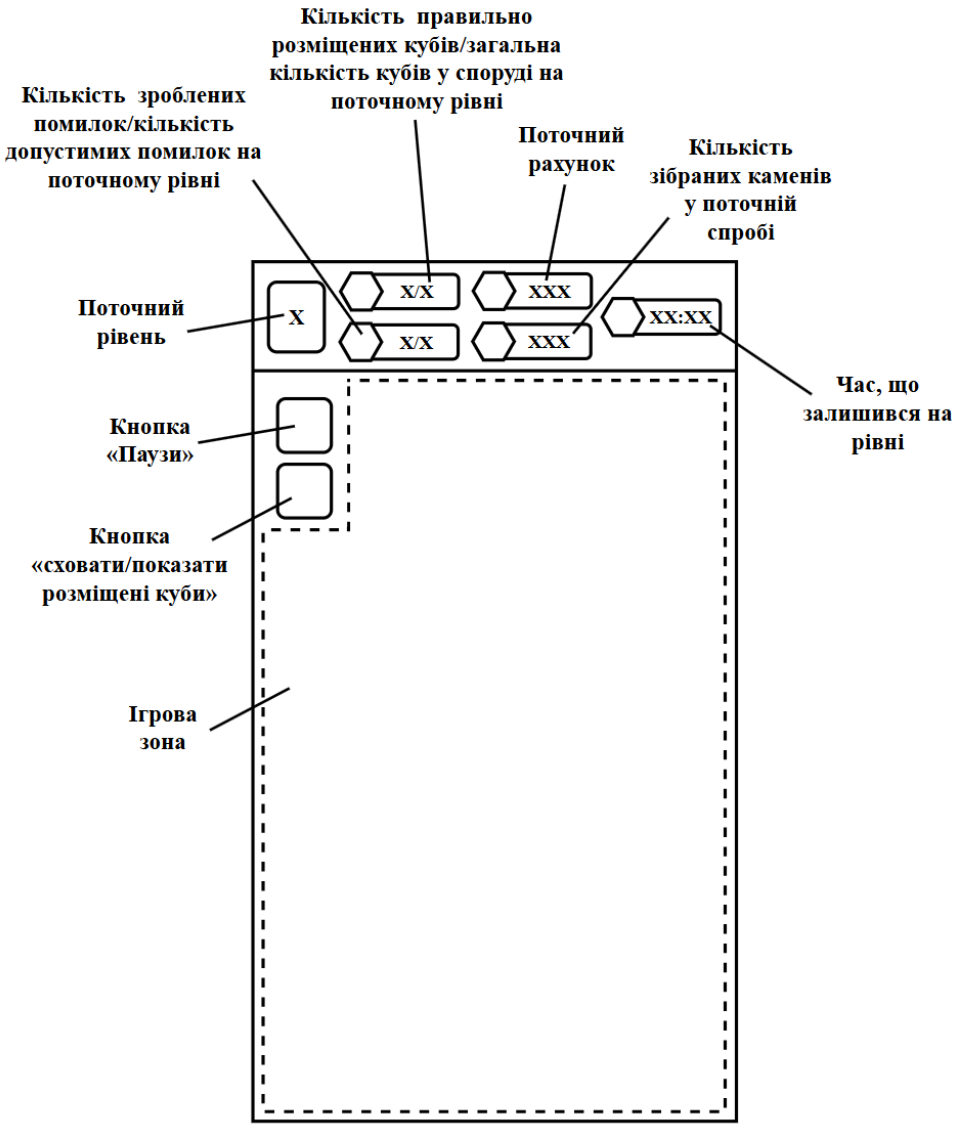

Рисунок 3.8 — Макет сцени ігрового процесу

Зверху на панелі буде розташований основний інтерфейс гри (див. рисунок 3.8), а саме там буде показано:

– поточний рівень;

– кількість правильно розміщених кубів та загальна кількість кубів у споруді на поточному рівні;

– кількість зроблених помилок (кубів розміщених в неправильних позиціях) та допустима кількість помилок на поточному рівні;

– поточний рахунок гравця;

– кількість зібраних дорогоцінних каменів у поточній спробі;

– час (таймер), який залишився гравцю на заповнення споруди.

Всі ці значення будуть динамічно змінюватися під час гри, чи то гравець розмістив куб у правильній позиції, чи зібрав дорогцінне каміння і т.д. — все буде відображено для розуміння, що зробив гравець.

Трохи нижче, зліва, буде розміщено дві кнопки (див. рисунок 3.8): кнопка «Паузи» та кнопка «сховати або показати розміщені куби».

При натисканні кнопки «Паузи», буде з'являтися вікно паузи, макет якого зображено на рисунку 3.9 та гра буде ставитися на паузу, і весь геймплей буде завмирати поки гравець не натисне кнопку «Повернутися до гри».

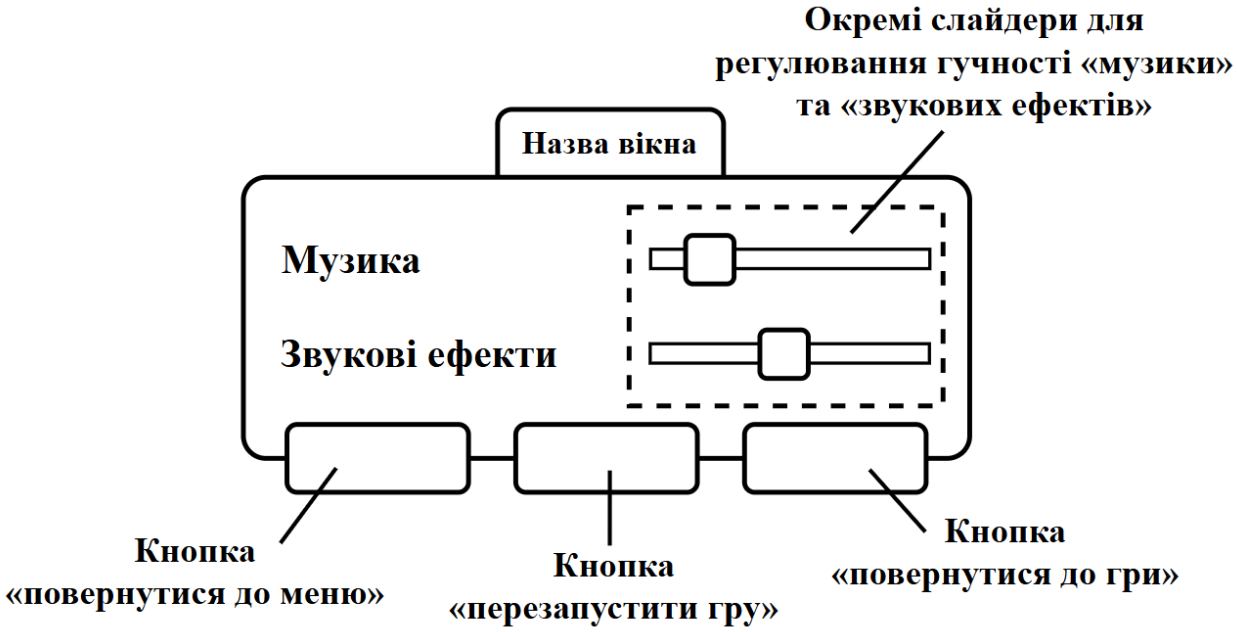

Рисунок 3.9 — Макет вікна паузи

У цьому вікні гравець може окремо відрегулювати гучність «музики» та «звукових ефектів». Також із цього вікна гравець може повернутися до меню або перезапустити гру натиснувши на відповідні кнопки (див. рисунок 3.9).

Кнопка «сховати або показати розміщені куби» (див. рисунок 3.9) буде слугувати допоміжним інструментом, який допомагатиме гравцеві виконувати своє основне завдання. А саме, при натисканні цієї кнопки, всі розміщені куби гравцем, окрім крайнього будуть сховані (лише візуально), для того, щоб полегшити гравцю видимість решти споруди яку треба заповнити. Це полегшить гравцеві заповнення

споруди, особливо на вищих рівнях, де кількість кубів у споруді буде досить високою. При повторному натисканні на цю кнопку, сховані куби будуть знову показані.

Усе інше місце, що залишилося на екрані (див. рисунок 3.8), слугуватиме так званим ігровим полем, у якому гравець зможе керувати грою і з комфортом заповнювати споруду.

При програші (вийшов час або гравець перевищив ліміт допустимих помилок), як ми говорили у розділі концепту, спочатку буде показуватися спеціальне вікно з пропозицією гравцеві перегратиточний рівень в обмін на перегляд реклами. Гравець зможе обрати сам, чи хоче він дивитися рекламу, щоб продовжити чи ні. Макет такого вікна зображено на рисунку 3.10.

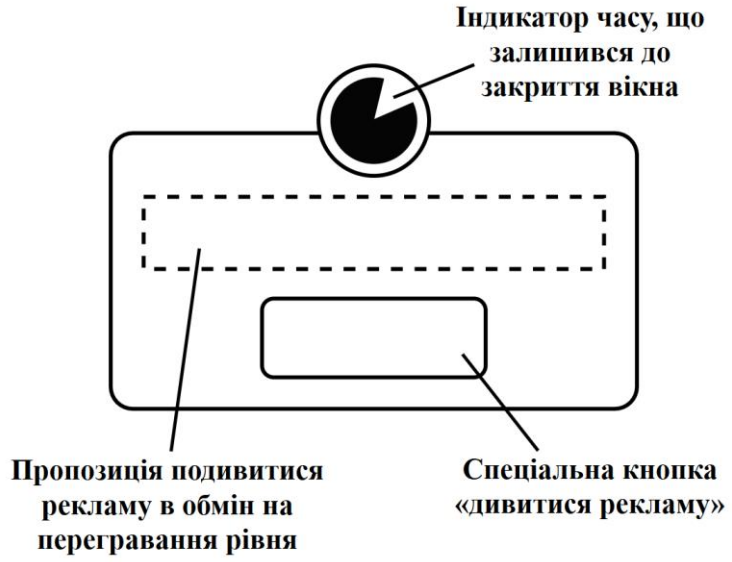

Рисунок 3.10 — Макет спеціального вікна з пропозицією

Зверху цього вікна буде розміщений індикатор часу (див. рисунок 3.10) видимості цього вікна. Як тільки час вийде, вікно закриється автоматично. Якщо гравець відмовиться від перегляду реклами та не захоче чекати автоматичного закриття вікна, він може тапнути в будь-якому місці поза цим вікном для того, щоб закрити його вручну. Якщо гравець захоче подивитися рекламу, щоб переграти рівень, він може тапнути на спеціальну кнопку, яка розміщена знизу вікна (див. рисунок 3.10).

Якщо гравець вирішив перегратирівень за перегляд реклами і програв після перегравання рівня або якщо гравець відмовився від перегляду реклами, зразу після цього гравцю буде показано вікно програшу. Макет цього вікна зображено на рисунку 3.11.

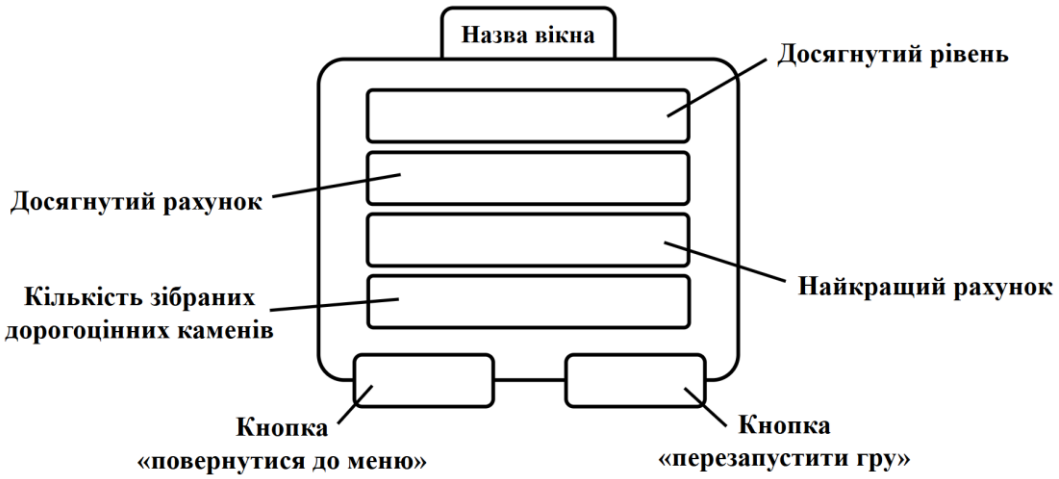

Рисунок 3.11 — Макет вікна програшу

Вікно програшу містить інформацію (див. рисунок 3.11) про максимальний досягнутий рівень за поточну спробу, досягнутий рахунок у поточній спробі, а також найкращий рахунок який гравець досягав граючи у гру та кількість зібраних дорогоцінних каменів за цю спробу. Гравець може почати гру заново або перейти до меню, натиснувши відповідні кнопки, які розміщені знизу вікна.

3.2.5 Сцена ігрового магазину

Внутрішньоігровий магазин є важливою частиною досвіду гравця. Це не лише місце, де можна придбати або відкрити додатковий контент, але й можливість для гравця виразити свою індивідуальність, персоналізуючи об’єкти, з якими він взаємодіє. У цьому проєкті внутрішньоігровий магазин дозволяє гравцям змінювати візуальний вигляд кубів за допомогою скинів, що заохочує гравців повертатися до гри, підвищуючи залученість. Як ми говорили у розділі концепції, магазин буде містити два типи скинів: які відкриваються за досягнуті під час гри очки, інші — відкриваються за дорогоцінні камені, які можна збирати під час гри. Така можливість підтримує інтерес до гри, адже гравці прагнуть досягати нових

висот або накопичувати більше дорогоцінного каміння, аби відкрити бажані елементи.

На рисунку 3.12 зображено макет сцени ігрового магазину. Зверху на панелі буде розташовано:

– кнопку для «виходу з ігрового магазину»;

– найвищий досягнутий рахунок, який гравець досягав граючи у гру;

– кількість наявного дорогоцінного каміння;

– спеціальну кнопку, тапнувши на яку, гравець зможе отримати додаткове дорогоцінне каміння за перегляд реклами.

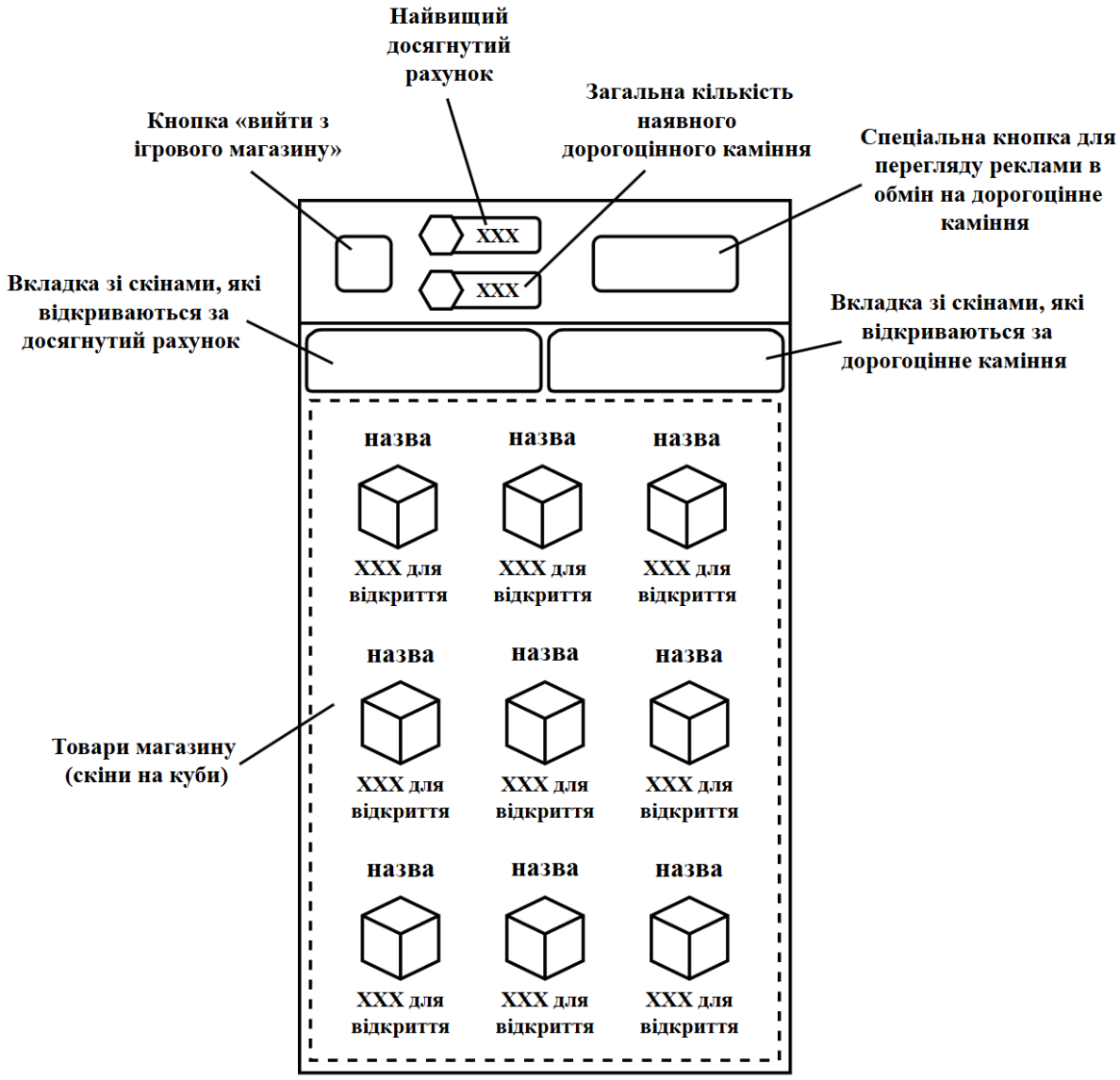

Рисунок 3.12 — Макет сцени ігрового магазину

Тапнувши на спеціальну кнопку (див. рисунок 3.12), гравцю буде показана реклама. Після того як гравець додивиться рекламу до кінця і закриє її, гравцеві буде відкриватися вікно з інформацією про успішне отримання винагороди, та кількість отриманих дорогоцінних каменів. Макет такого вікна зображено на рисунку 3.13.

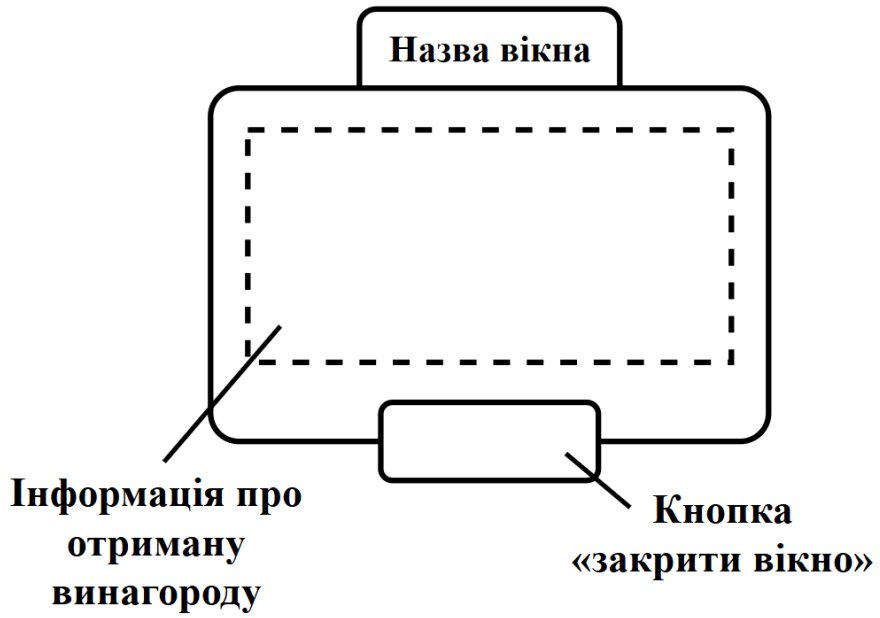

Рисунок 3.13 — Макет вікна успішного отримання винагороди

Трохи нижче буде розташована панель з двома кнопками (див. рисунок 3.12), за допомогою яких гравець зможе переключатися між двома вкладками зі скінами: які відкриваються за досягнутий рахунок і які відкриваються за дорогоцінне каміння. Усе інше місце на сцені, слугуватиме місцем для розміщення найрізноманітніших скінів та їхньою інформацією. А саме гравцеві буде показано назву скіна і який рахунок йому потрібно досягнути або яку кількість дорогоцінних каменів йому потрібно зібрати для того що відкрити той чи інший скін.

Скіни які відкриваються за досягнутий рахунок, будуть відкриватися автоматично при досяганні необхідного рахунку. А для того, щоб, відкрити скіни які відкриваються за дорогоцінні камені, гравцю потрібно просто тапнути на бажаний скін, та підтвердити покупку у з'явившомуся вікні, макет якого зображено на рисунку 3.14.

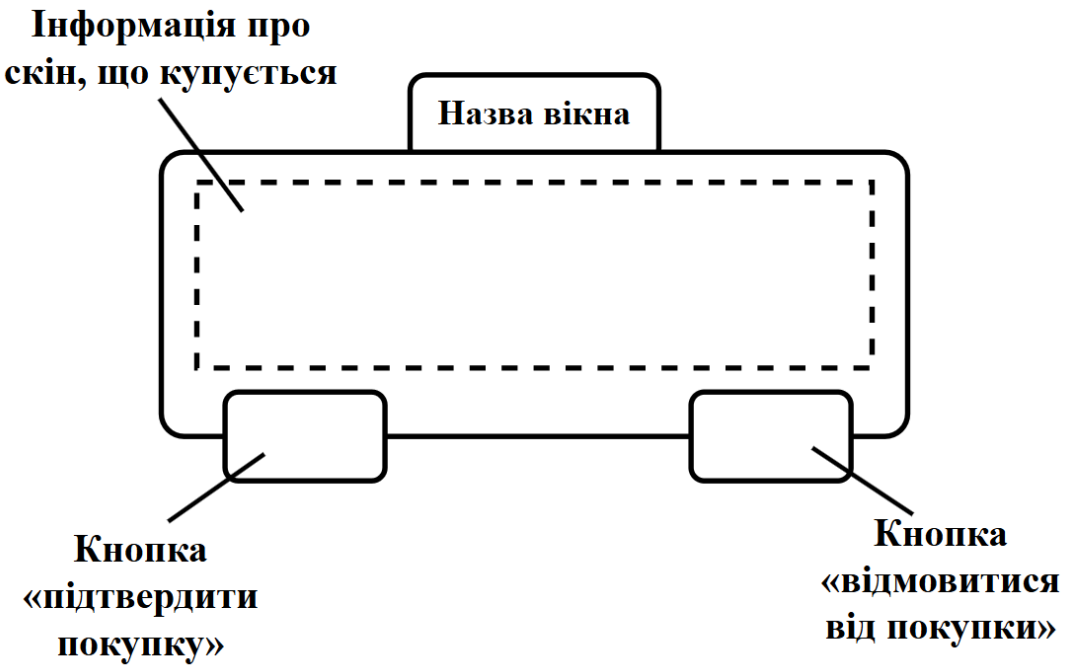

Рисунок 3.14 — Макет вікна з підтвердженням покупки

У цьому вікні (див. рисунок 3.14) міститься основна інформація про скін, що купується, а саме, його ціна (кількість дорогоцінного каміння яке буде списано з балансу гравця), а також його назва. Гравець може підтвердити або відмовитися від покупки (якщо наприклад гравець ненавмисно тапнув не на той скін який він хотів відкрити) тапнувши на відповідну кнопку (див. рисунок 3.14).

4 РЕАЛІЗАЦІЯ ГРИ

У цьому розділі буде проходити ключовий етап роботи над грою — реалізація всіх елементів, які перетворюють концепцію на функціональний продукт. Тут будуть втілюватися основні ідеї, сформульовані на етапі проєктування: буде розроблюватися дизайн, який включає інтерфейс, ефекти та анімації, також буде реалізовуватися основний ігровий процес, ігрові механіки, звукові ефекти та інше. Крім того, буде інтегруватися внутрішньоігрова реклама.

4.1 Розробка дизайну гри

Розробка дизайну гри є фундаментальним аспектом, що визначає успіх проєкту. Дизайн не лише формує візуальну привабливість, а й впливає на те, як гравці сприймають, взаємодіють із грою та наскільки глибоко вони в неї занурюються. Хороший дизайн створює перше враження, яке часто є вирішальним. Якщо гра виглядає привабливо, гравець схильний дати їй шанс навіть до того, як оцінить її механіку. Це особливо важливо на конкурентному ринку, де велика кількість ігор бореться за увагу користувачів. Привабливий дизайн допомагає встановити емоційний зв’язок із гравцем. Правильно підібрані кольори, стиль графіки, анімації, візуальні ефекти та звукове оформлення викликають емоції, що сприяють створенню унікальної атмосфери гри.

Крім естетики, дизайн визначає зручність гри. Інтуїтивно зрозумілий інтерфейс, чітка навігація, комфортна розстановка елементів управління та відсутність зайвого візуального шуму дозволяють гравцям фокусуватися на процесі гри, а не на боротьбі з незрозумілими елементами.

4.1.1 Розробка інтерфейсу гри

В Unity створення інтерфейсу користувача (UI) здійснюється за допомогою Unity UI System, який надає всі необхідні інструменти для створення як простих, так і складних інтерфейсів. Основні елементи UI Unity, без яких не обійдеться ніякий інтерфейс це: Canvas, Event System (Система подій), Rect Transform.

Canvas — це основний компонент для розміщення елементів інтерфейсу. Усі UI-елементи мають бути його дочірніми об'єктами. Є три режими відображення:

– Screen Space — Overlay. Елементи UI завжди знаходяться поверх усіх інших об’єктів;

– Screen Space — Camera. UI прив’язується до камери, що дає змогу інтегрувати його в 3D-простір;

– World Space. UI існує як об’єкт у 3D-просторі і взаємодіє з іншими об'єктами сцени.

У цьому проєкті буде використовуватися режим відображення — Screen Space — Overlay, так як нам не потрібно інтегрувати UI у 3D-простір.

Event System керує обробкою подій (натискання кнопок, перетягування тощо). Вона автоматично додається до сцени при створенні Canvas.

Для UI-елементів використовується Rect Transform замість звичайного Transform, який дозволяє працювати з положенням, розмірами та якірними точками (anchors).

Основні компоненти Unity UI які будуть використані у цьому проєкті:

– компонент Image. Дозволяє відображати спрайти або кольорові фони. Використовується для фону, іконок, рамок тощо;

– компонент Button. Це інтерактивний елемент інтерфейсу, який використовується для виконання дій під час натискання. Він складається з фону, заданого компонентом Image, і тексту, що є його дочірнім об'єктом;

– компонент Text. Використовується для відображення тексту. Має налаштування шрифтів, кольорів, стилів та вирівнювання;

– компонент Slider. Слайдер для налаштувань (наприклад, гучність звуку). Складається з рухомого маркера і заповнення;

– компонент Panel. Це базовий елемент інтерфейсу в Unity, який використовується для створення фонових областей або контейнерів для інших UI-елементів.

Перед тим як приступити до створення (малювання) всіх спрайтів для нашого спроєктованого інтерфейсу, який ми робили в минулому розділі, для початку варто визначитися з палітрою кольорів для проєкту. Кольорова палітра — це набір кольорів, які створюють її стиль та атмосферу. Правильний вибір кольорів допомагає викликати потрібні емоції у гравців, підкреслити важливі елементи й зробити текст читабельним. Гармонійна палітра, яка відповідає жанру та стилю гри, значно покращує враження користувачів і сприяє її впізнаваності.

Для цього проєкту було обрано наступну кольорову палітру із 5-ти кольорів: 8c5af4, 9b70ff, d4dcff, a396f9, b97cff. Саму палітру зображено на рисунку 4.1.

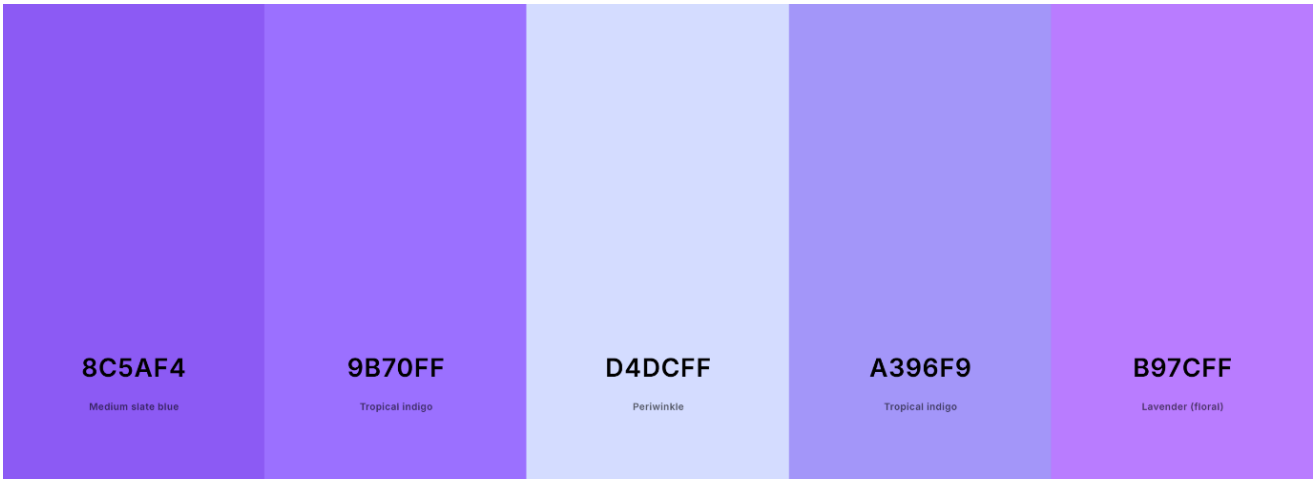

Рисунок 4.1 — Підібрана кольорова палітра [51]

Перейдемо до створення спрайтів для інтерфейсу гри на основі підібраної кольорової палітри. Вони будуть створюватися у програмі Adobe Photoshop. Adobe Photoshop являється професійною програмою для редагування растрової графіки, розроблена компанією Adobe. Використовується для обробки фотографій, створення цифрових ілюстрацій, дизайну, ретуші та роботи із зображеннями. Photoshop підтримує роботу з шарами, має широкий набір інструментів для редагування, фільтри, плагіни та функції штучного інтелекту для автоматизації складних завдань. Це один із найпопулярніших інструментів у сфері дизайну, фотографії та цифрового мистецтва.

Нижче зображено спрайти які були створені за допомогою Adobe Photoshop для навчальної сцени, а саме спрайт для вікна з підказкою (див. рисунок 4.2), а також спрайт кнопки для такого ж вікна тільки з кнопкою (див. рисунок 4.3).

Рисунок 4.2 — Створений спрайт вікна з підказкою

Рисунок 4.3 — Створений спрайт кнопки вікна з підказкою

На рисунку 4.4 зображено готове вікно з підказкою з кнопкою у Unity. На цьому вікні буде зображуватися потрібний текст підказки.

Рисунок 4.4 — Готове вікно з підказкою з кнопкою

Після створення потрібних спрайтів, було розроблено і налаштовано ієрархію об'єктів UI у Canvas, яка зображена на рисунку 4.5.

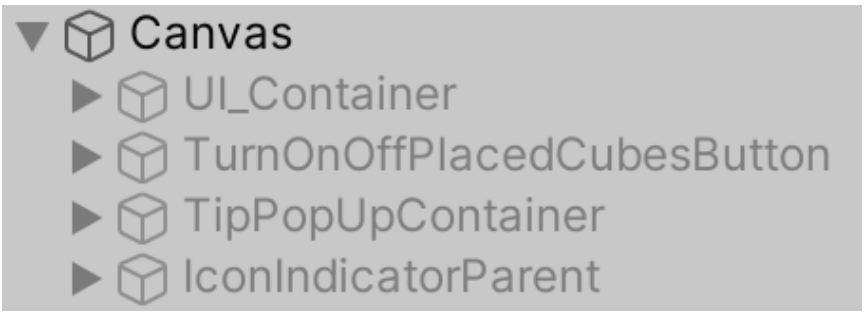

Рисунок 4.5 — Ієрархія об'єктів інтерфейсу навчальної сцени

UI_Container та TurnOnOffPlacedCubesButton є основними UI об'єктами сцени ігрового процесу. Вони тут потрібні, для того щоб пояснити гравцю їх призначення. TipPopUpContainer являється контейнером вікна з підказкою, а IconIndicatorParent являється контейнером для індикаторів (іконок), які будуть демонструвати гравцеві потрібний жест для виконання якоїсь дії.

Подібним чином для сцени меню було створено декілька спрайтів кнопок, а також спрайти для вікна з налаштуваннями та вікна «Про гру». Деякі з цих спрайтів зображено на рисунках 4.6 та 4.7.

Рисунок 4.6 — Створений спрайт вікна з налаштуваннями

Рисунок 4.7 — Створений спрайт кнопок для переходу у інші сцени

Після створення всіх потрібних спрайтів, було розроблено і налаштовано ієрархію об'єктів UI у Canvas для сцени меню, яка зображена на рисунку 4.8. А на рисунку 4.9 зображено повністю реалізований вигляд інтерфейсу сцени меню.

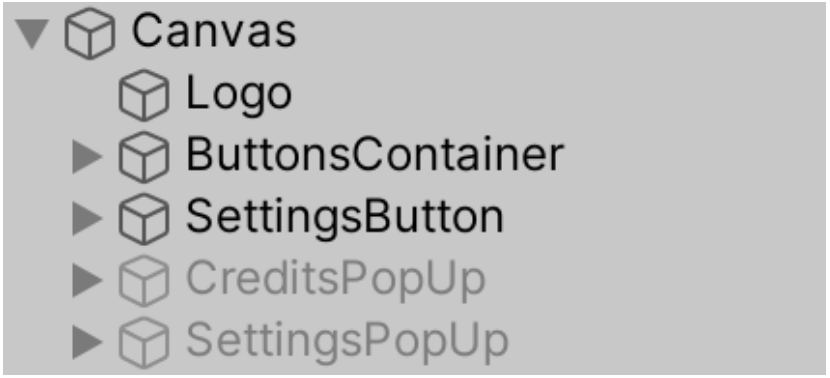

Рисунок 4.8 — Ієрархія об'єктів інтерфейсу сцени меню

Рисунок 4.9 — Реалізований інтерфейс сцени меню

Для сцени ігрового процесу було свторено безліч спрайтів як для кнопок так і для вікон. Деякі створені спрайти зображно на рисунках 4.10, 4.11, 4.12.

Рисунок 4.10 — Створений спрайт вікна програшу

Рисунок 4.11 — Створений спрайт кнопки паузи

Рисунок 4.12 — Створений спрайт кнопок для вікон

Також на сайті зі стоковими іконками були підібрані відповідні іконки для інтерфейсу гри, деякі з них зображено на рисунку 4.13 [1].

Рисунок 4.13 — Деякі підібрані іконки для проєкту

Після створення всіх потрібних спрайтів, було розроблено і налаштовано ієрархію об'єктів UI у Canvas для сцени ігрового процесу, яка зображена на рисунку 4.14. А на рисунку 4.15 зображено повністю реалізований вигляд інтерфейсу сцени ігрового процесу.

Рисунок 4.14 — Ієрархія об'єктів інтерфейсу сцени ігрового процесу

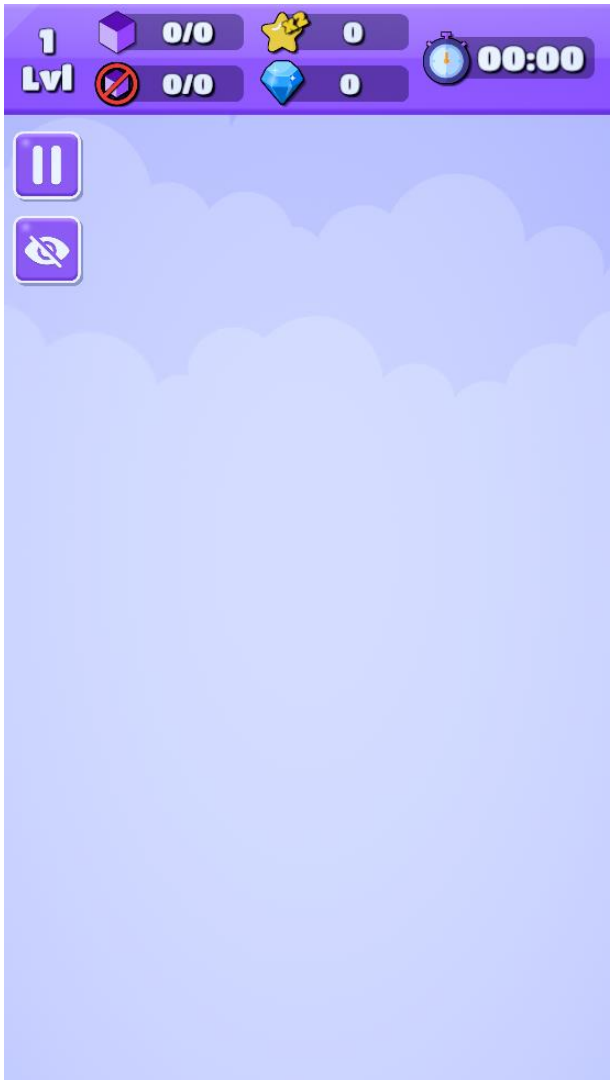

Рисунок 4.15 — Реалізований інтерфейс сцени ігрового процесу

Для сцени внутрішньоігрового магазину також була створена певна кількість спрайтів, деякі з яких зображно на рисунках 4.16, 4.17 та 4.18.

Рисунок 4.16 — Створений спрайт кнопки «вийти з ігрового магазину»

Рисунок 4.17 — Створений спрайт кнопок на панелі вкладок

Рисунок 4.18 — Створений спрайт вікна «підтвердження покупки»

Також після створення всіх потрібних спрайтів для цієї сцени, було розроблено і налаштовано ієрархію об'єктів UI у Canvas для сцени ігрового магазину, яка зображена на рисунку 4.20. А на рисунку 4.19 зображено повністю реалізований вигляд інтерфейсу сцени ігрового магазину.

Рисунок 4.19 — Реалізований інтерфейс сцени ігрового магазину

Рисунок 4.20 — Ієрархія об'єктів інтерфейсу сцени ігрового магазину

4.1.2 Розробка візуальних ефектів гри

Візуальні ефекти відіграють ключову роль у мобільних іграх, створюючи не лише естетичну привабливість, а й глибший зв’язок між гравцем і ігровим процесом. Вони допомагають передати атмосферу, підсилюють емоційне залучення і забезпечують зворотний зв'язок, що є критично важливим для якісного ігрового досвіду. Окрім естетичної складової, візуальні ефекти виконують функцію інтуїтивного навігаційного інструменту. Вони можуть привертати увагу до важливих об’єктів, підказувати напрямок руху чи вказувати на небезпеки. У швидкому темпі мобільної гри це особливо важливо, адже гравець часто діє інтуїтивно, спираючись на візуальні сигнали.

Для створення візуальних єфектів у цьому проєкті ми будемо використовувати вбудовану систему частинок (Particle System) у Unity.

Система частинок (Particle System) в Unity — це потужний інструмент для створення візуальних ефектів, таких як дим, вогонь, сніг, дощ, вибухи або магічні ефекти. Вона працює шляхом створення маленьких графічних об'єктів (частинок), які рухаються, змінюються і зникають відповідно до заданих параметрів.

Система частинок у Unity базується на компоненті Particle System, який додається до об'єкта сцени. Particle System має багато модулів для деталізації роботи частинок:

– Emission. Контролює кількість частинок, які випускаються;

– Shape. Форма області, звідки з'являються частинки;

– Velocity over Lifetime. Зміна швидкості частинок протягом життя;

– Color over Lifetime. Градієнт зміни кольору;

– Size over Lifetime. Зміна розміру протягом життя;

– Noise. Додає турбулентності до руху частинок;

– Collision. Взаємодія частинок з об'єктами у світі;

– Renderer. Відповідає за візуалізацію частинок, дозволяючи налаштовувати їхній матеріал, текстуру, форму та порядок відображення в кадрі;

– Lights. Використання частинок для створення джерел світла;

– Trails. Додає сліди до частинок для створення ефекту шлейфу.

Основні параметри налаштовуються у Main Module:

– Lifetime (тривалість життя). Як довго частинка існує;

– Start Speed. Початкова швидкість;

– Start Size. Розмір частинки;

– Start Color. Колір частинки;

– Gravity Modifier. Вплив гравітації.

Для цієї гри було розроблено велику кількість ефектів: ефект під час розміщення нового куба, ефекти підбирання дорогоцінних каменів, ефект розміщення куба в неправильній позиції, ефект (частинки) для навколишнього середовища (для додавання більшої привабливості та динаміки під час ігрового процесу) та інші.

Розглянемо приклад одного розробленого візуального ефекту, який з'являтиметься в разі правильного розміщення куба. Спочатку для нього було розроблено потрібний спрайт у програмі Adobe Photoshop, про який ми говорили вище. Цей розроблений спрайт зображено на рисунку 4.21.

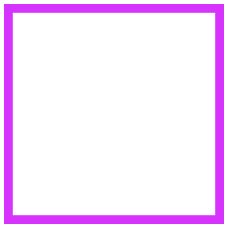

Рисунок 4.21 — Розроблений спрайт для ефекту розміщення куба

Потім був створений пустий 3D-об'єкт і для цього об'єкта був прикріплений компонент Particle System. Далі для розробленого спрайта (див. рисунок 4.21) був створений матеріал. Цей матеріал було присвоєно в комірку material модулю Renderer у компоненті Particle System. І потім було проведено налаштування цього Particle через його найрізноманітніші модулі та їхні параметри для досягнення бажаного результату. Результатом детального налаштування та експериментами з різними параметрами було розроблено ефект розміщення куба у правильній позиції. Суть ефекту полягає в тому, що під час розміщення куба, у місці, де з'єднуються новий розміщений куб і попередній куб, буде розміщуватися цей об'єкт (префаб) з розробленим і налаштованим компонентом Particle System, це буде відбуватися у скрипті який буде написаний у відповідному розділі. На старті програвання цього ефекту, particle буде мати розмір, що дорівнює розміру розміщеного куба, і далі він буде плавно збільшуватися (розширюватися) навколо куба та паралельно разом із цим він буде плавно згасати (прозорість particle буде зменшуватися поки не досягне 0). На рисунку 4.22 показано як виглядає цей розроблений ефект.

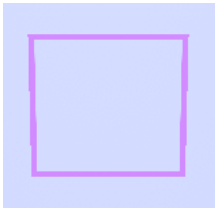

Рисунок 4.22 — Розроблений ефект розміщення куба у правильній позиції

Подібним чином було розроблено безліч інших ефектів. Ще один із прикладів зображений на рисунку 4.23.

Рисунок 4.23 — Розроблений ефект підбирання дорогоцінного каменя

4.1.3 Розробка анімацій гри

Анімації у мобільній грі є ключовим елементом, який впливає на загальне враження гравця та якість взаємодії з грою. Перш за все, анімації допомагають створити атмосферу та занурення у гру. Вони оживляють середовища та ігрові події. Якісні анімації є частиною загального враження про гру. Вони сприяють відчуттю професійності та уваги до деталей, що може стати вирішальним фактором у тому, чи захоче гравець продовжувати гру або рекомендувати її іншим. На конкурентному ринку мобільних ігор естетичне оформлення та плавність анімацій можуть виділити проєкт серед безлічі інших.

У Unity є своє вбудоване рішення (анімаційний двигун) для створення анімацій. Але це вбудоване рішення анімацій Unity яке будується навколо компонента Animator не є оптимальним вибором для простіших анімацій у мобільних іграх. Основні причини чому:

– перевантаженість Animator;

– менша гнучкість і контроль;

– відсутність інтерполяції для простих задач.

Пговоримо детальніше про кожну з цих причин.

Перевантаженість Animator.

Animator Controller підходить для складних анімаційних систем (наприклад, персонажів із різними станами, переходами та параметрами). Однак для простих анімацій, таких як переміщення об'єкта, зміна кольору чи обертання, створення Animator Controller є надмірним. Також Animator споживає більше пам'яті й викликає додаткові витрати на обробку, навіть якщо він використовується для простих завдань. Це може бути критичним для мобільних пристроїв із обмеженими ресурсами.

Анімації в Animator часто пов'язані з конкретними об'єктами й сценами, що ускладнює їх динамічне створення або модифікацію через код. Якщо анімації потрібно запускати або налаштовувати програмно, Animator вимагає більше зусиль, ніж альтернативи, які надають інтуїтивно зрозумілі API.

Animator створений для керування записаними анімаціями. У ситуаціях, де

потрібна проста інтерполяція (наприклад, переміщення об'єкта з точки А в точку Б), використання Animator не є інтуїтивним.

Тому у цьому проєкті для створення анімацій була використана безкоштовна бібліотека анімацій PrimeTween від українського розробника. PrimeTween — це високопродуктивна бібліотека анімації для Unity, що не виділяє пам'ять під час свого виконання на відміну від вбудованного рішення Unity. Він надає всі необхідні інструменти для того щоб анімувати будь-що одним рядком коду, налаштовувати всі властивості анімації безпосередньо з інспектора та надає можливість створювати складні анімаційні послідовності [53].

Для того, щоб створювати прекрасні анімації за допомогою цієї бібліотеки, все, що потрібно зробити — це імпортувати цю бібліотеку та просто ввести `Tween.` і IDE запропонує всі підтримувані анімації. Якщо немає бажаної анімації, можна використовувати Tween.Custom(), щоб анімувати будь-що і будь-як. Також важливим компонентом цієї бібліотеки є Sequence. Sequence — це група твінів, зворотних викликів та інших послідовностей. Анімації у Sequence можуть накладатися одна на одну, виконуватися послідовно або паралельно, а також мати будь-яку комбінацію. Керувати послідовністю можна так само, як і окремими твінами.

Для цієї гри було розроблено багато анімацій, включаючи: різноманітні анімації для інтерфейсу користувача (UI), анімації для розміщення та руйнування споруди, анімації для дорогцінних каменів, анімації переходу між сценами та багато інших. У лістингу 4.1 наведено один із прикладів розроблених анімацій, а саме код анімації для камери у сцені меню, за допомогої якої камера красиво пролітає навколо споруди, надаючи динамічності до сцени.

Лістинг 4.1 — Код анімації для камери у сцені меню

```
Vector3 YRotation = new Vector3 (0, -360f, 0);
Sequence.Create(cycles: -1, CycleMode.Restart)
    .Group(
    Tween.LocalEulerAngles(
    transform,
    startValue: Vector3.zero,
    endValue: YRotation,
```

```
duration: 20f,
Ease.Linear))
.Group(
Tween.PositionY(
transform,
startValue: 0f,
endValue: 5f,
duration: 10f,
Ease.Linear,
cycles: 2,
cycleMode: CycleMode.Yoyo));
```

4.1.4 Створення звукового дизайну гри

Звуковий дизайн, включаючи звукові ефекти та музику, є важливим елементом мобільних ігор, оскільки він значно підвищує якість ігрового досвіду, впливаючи на емоційне сприйняття та взаємодію гравця з грою. Грамотно створений звуковий супровід здатен створювати атмосферу, яка занурює гравця у світ гри, допомагаючи краще сприймати її механіки та естетику. Звукові ефекти додають інтерактивності ігровому процесу. Вони сигналізують про важливі події в грі, такі як досягнення рівня, отримання нагороди чи попереджають гравця про неправильно розміщений куб.

Unity надає потужні інструменти для роботи зі звуком, дозволяючи створювати інтерактивні аудіо-сцени, застосовувати ефекти та налаштовувати поведінку звуку залежно від ігрових подій.

Основні компоненти звукового дизайну у Unity включають:

– Audio Listener;

– Audio Source;

– Audio Clip;

– Audio Mixer;

– 3D-Audio;

– Audio Effects.

Поговоримо про кожний з них.

Audio Listener. Цей компонент «чує» всі звуки у сцені. Його можна порівняти з мікрофоном, що фіксує звукові події. Зазвичай його додають до об'єкта камери.

– в сцені має бути лише один Audio Listener, інакше виникають конфлікти;

– використовується разом із компонентами Audio Source, щоб відтворювати 3D-звуки залежно від позиції об'єкта.

Audio Source. Це основний компонент для відтворення звуків. Додається до об'єкта, який має створювати звук (наприклад, персонажа, машини, чи об'єкта довкілля). Основні властивості:

– AudioClip. Звуковий файл, який буде відтворюватися;

– Output. Аудіо-канал (Audio Mixer), до якого прив'язаний звук;

– Loop. Опція повторення звуку;

– Play On Awake. Звук автоматично відтворюється при запуску сцени;

– Spatial Blend. Регулює змішування 2D та 3D звуку;

– Doppler Level . Імітує ефект Доплера (зміна висоти звуку при русі джерела).

Audio Clip. Це сам звуковий файл, який може бути у форматах WAV, MP3, OGG тощо. Може бути моно або стерео. Підтримуються як короткі ефекти, так і довгі музичні треки.

Audio Mixer. Це інструмент для управління кількома аудіо-каналами, що дозволяє змішувати та коригувати різні звукові елементи, створюючи більш точний і збалансований звук. За допомогою цього інструменту можна регулювати гучність, частотні характеристики для кожного каналу, що дає можливість досягти бажаного звукового ефекту і підвищити якість звукового супроводу. Основні функції:

– групування аудіо-джерел (наприклад, окремі канали для музики, звукових ефектів і діалогів);

– регулювання гучності, ефектів і фільтрів у реальному часі;

– застосування ефектів, таких як реверберація, еквалізація, компресія тощо.

3D-Audio. Unity підтримує тривимірний звук, який залежить від просторового розташування Audio Source і Audio Listener. Радіус прослуховування задається через властивості Min Distance та Max Distance у компоненті Audio Source. Звук стає гучнішим чи тихішим залежно від відстані до слухача. Функція Spatial Blend дозволяє точно контролювати, як звук взаємодіє з простором.

Audio Effects. Unity дозволяє додавати спеціальні аудіо-ефекти (через компоненти або Audio Mixer):

– Reverb Zone. Зона реверберації, створює ефект відлуння;

– Audio Low Pass Filter. Знижує високі частоти звуку;

– Audio High Pass Filter. Знижує низькі частоти;

– Distortion. Додає спотворення звуку.

Першим кроком до створення системи звукового дизайну є створення та налаштування Аудіо-мікшера (Audio Mixer). Цей інструмент дозволяє впорядкувати і розділити всі аудіо-сигнали на логічні групи, що значно спрощує управління звуковими ефектами (SFX) та музикою (BGM) у грі. Для цього у вікні проєкту був створений аудіо-мікшер, який зображено на рисунку 4.24.

Рисунок 4.24 — Створений аудіо-мікшер

Далі у вікні аудіо-мікшера було створено дві групи: одна для звукових ефектів та інша для музики, які зображено на рисунку 4.25. Це потрібно для того, щоб, гравець міг окремо коригувати гучність «музики» та «звукових ефектів» під час гри у вікні налаштувань, яке розташоване у сцені меню (див. рисунок 3.6) та у вікні паузи, яке розташоване у сцені ігрового процесу (див. рисунок 3.9).

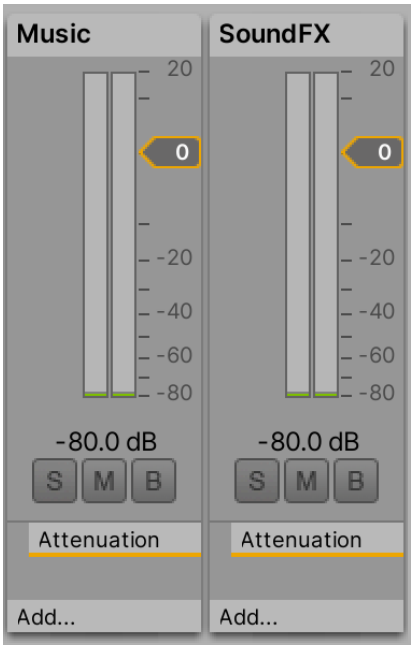

Рисунок 4.25 — Створені групи для для звукових ефектів та музики

Для програвання фонової музики під час гри, у сцені меню був створений об'єкт, до якого був доданий компонент AudioSource й у комірці Output цього компоненту був вибраний попередньо створений аудіо-мікшер та його група Music, яку ми теж створили перед цим. А також були включені його властивості PlayOnAwake та Loop, для того, щоб музика починала програватися зразу на початку гри, а Loop для того, щоб музика програвался безперервно. Цей об'єкт зображено на рисунку 4.26. Потім для цього об'єкту ми напишемо та прикріпимо спеціальний скрипт, за допомогою якого, цей об'єкт буде залишатися активним під час всієї гри, тобто не буде знищуватися під час переходу між сценами, зберігаючи постійне м'яке програвання фонової музики під час гри.

Для звукових ефектів теж був створений окремий об'єкт у сцені меню, до якого теж був доданий компонент AudioSource й у комірці Output цього компоненту був вибраний попередньо створений аудіо-мікшер та його група SoundFX, яку ми теж створили перед цим, цей створений об'єкт зображено на рисунку 4.27. Надалі для цього об'єкту буде написано скрипт, а саме менеджер звукових ефектів, який буде реалізовувати функціонал для програвання звукових ефектів з будь-якого місця у будь-якій сцені за допомогою цього створеного об'єкта. Цей об'єкт теж буде залишатися активним під час всієї гри, тобто не буде

знищуватися під час переходу між сценами, для того, щоб у будь-який потрібний момент часу ми могли програти потрібний звуковий ефект.

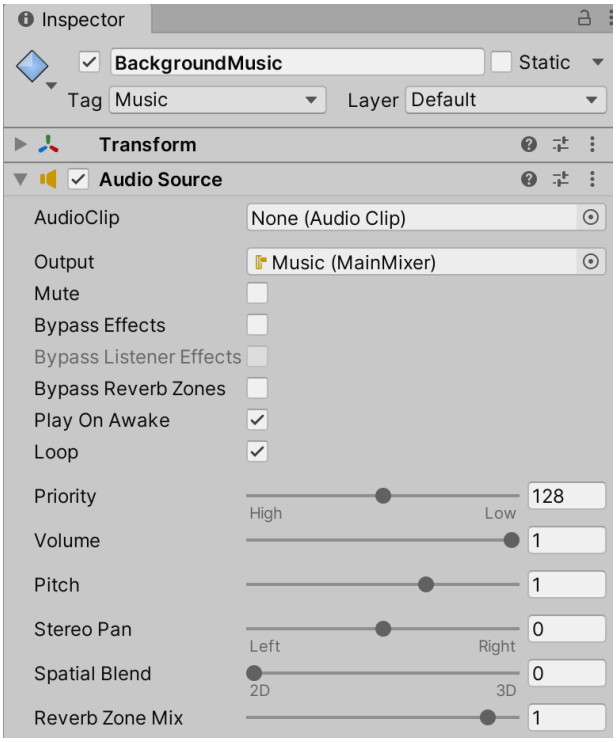

Рисунок 4.26 — Створений об'єкт для управління фоновою музикою у грі

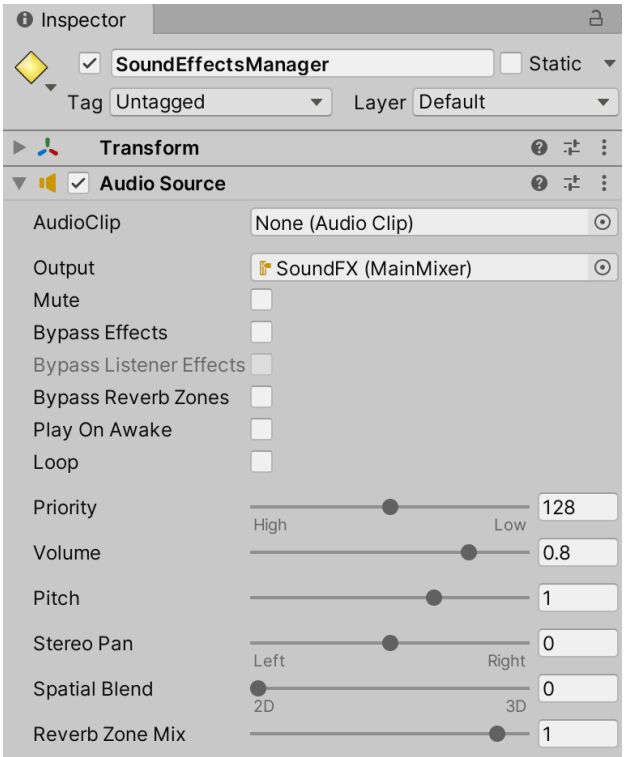

Рисунок 4.27 — Створений об'єкт для управління звуковими ефектами у грі

Звуки й музика для цього проєкту підбиралися із безкоштовних асетів звукового дизайну з офіційного сайту асетів Unity AssetStore. Вони підбиралися дуже ретельно, дотримуючись загальної естетики та атмосфери розроблюваної гри. Було підібрано величезну кількість звуків як для інтерфейсу користувача, так і для всіх подій які можуть відбутися у грі.

4.1.5 Розробка скинів для внутрішньоігрового магазину

Гарний дизайн скінів відіграє важливу роль у формуванні позитивного досвіду гравця та підтримці його зацікавленості. Коли скіни виглядають привабливо, детально опрацьовані й відповідають загальній естетиці гри, вони стають не лише елементом кастомізації, а й цінним досягненням для гравця. У випадках, коли доступ до скінів відкривається за рекорд чи зібрані дорогоцінні камені, їх візуальна привабливість стає ще більш важливою, оскільки це мотиваційний фактор для гравців продовжувати гру та прагнути досягати нових висот.

Для розробки скинів, у нашому випадку на куби, спочатку потрібно отримати UV-розгортку моделі, яка визначить, як текстура накладатиметься на об’єкт. Далі, поверх отриманої UV-розгортки буде малюватися сама текстура скіна на куб. І вже потім готова текстура і модель куба з якого була отримана UV-розгортка імпортується в Unity, для намальованої текстури створюється матеріал, і зрештою цей матеріал застосовується до імпортованої моделі куба.

На прикладі одного скіна розглянемо процес створення скінів для цього проєкту. Спочатку треба отримати UV-розгортку звичайного куба та її модель, для того, щоб потім намальована текстура поверх розгортки правильно накладалася на куб. UV-розгортка — це двовимірне представлення поверхні 3D-моделі, яке показує, як текстура накладатиметься на об'єкт, розгортуючи його геометрію у площину. Для створення моделі куба та отримання її UV-розгортки, у цьому проєкті буде використовуватися програма Blender.

Blender — це потужна та безкоштовна програма для тривимірного моделювання, анімації, рендерингу, відеомонтажу та створення візуальних ефектів. Її розробляє та підтримує спільнота відкритого програмного забезпечення під

керівництвом Blender Foundation. Ця програма доступна для Windows, macOS і Linux, що робить її універсальним інструментом для професіоналів та аматорів у сфері комп'ютерної графіки. Blender дозволяє створювати тривимірні моделі з високим рівнем деталізації, застосовувати текстури, матеріали та освітлення для досягнення реалістичних ефектів. Завдяки вбудованому рушію рендерингу Cycles можна отримати фотореалістичні зображення, тоді як рушій Eevee забезпечує швидкий візуалізаційний процес у реальному часі. Інструменти для анімації дозволяють створювати складні сцени з рухом персонажів, об'єктів та камери. Blender також підтримує симуляцію фізики, як динаміка рідин, газів, тканин і частинок, що робить його корисним для створення реалістичних ефектів.

У Blender був створений базовий куб, який зображено на рисунку 4.28, а потім за допомогою функції Smart UV Project була зроблена автоматична UV-розгортка цього куба, яка зображена на рисунку 4.29. Потім, модель цього куба була експортована у форматі FBX і так само була експортована його зроблена UV-розгортка у форматі PNG.

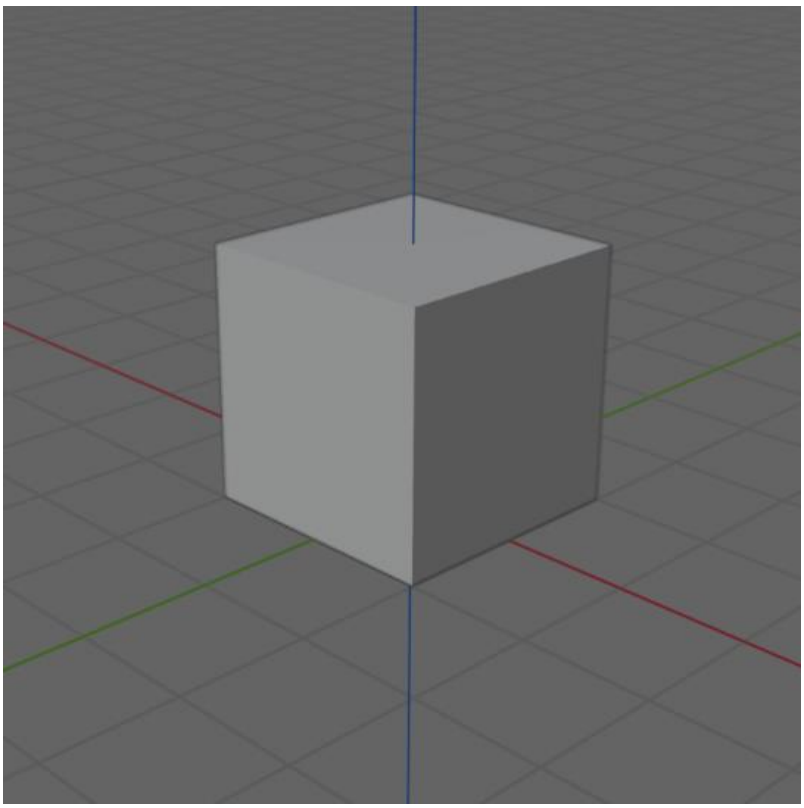

Рисунок 4.28 — Створений базовий куб у Blender

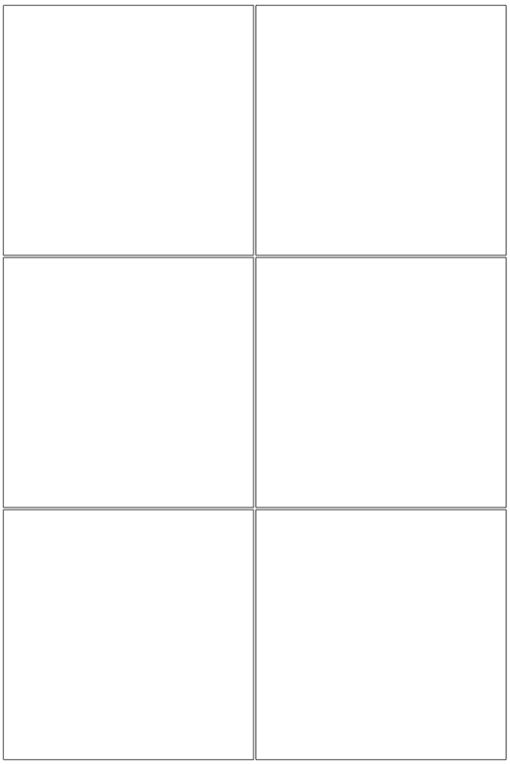

Рисунок 4.29 — Зроблена розгортка базового куба у Blender

Далі, за допомогою програми Adobe Photoshop, поверх зробленої UV-розгортки було намальовано текстуру скіна, яку зображено на рисунку 4.30.

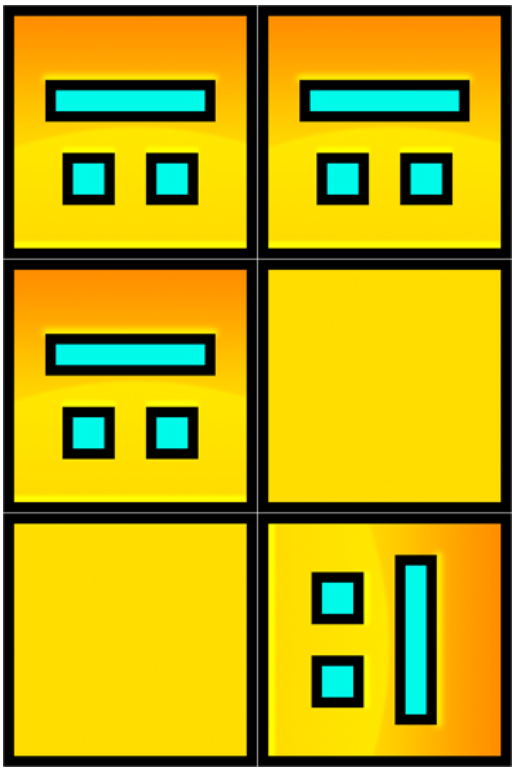

Рисунок 4.30 — Намальована текстура скіна поверх UV-розгортки

На завершальному етапі створення скіна, модель куба (у FBX форматі, яка була експортована з Blender) і намальована текстура в Adobe Photoshop були імпортовані в Unity. Потім в Unity був створений матеріал, до кого було застосовано намальовану текстуру і цей матеріал був застосований до імпортованої моделі куба. На рисунку 4.31 зображено отриманий результат створення скіна.

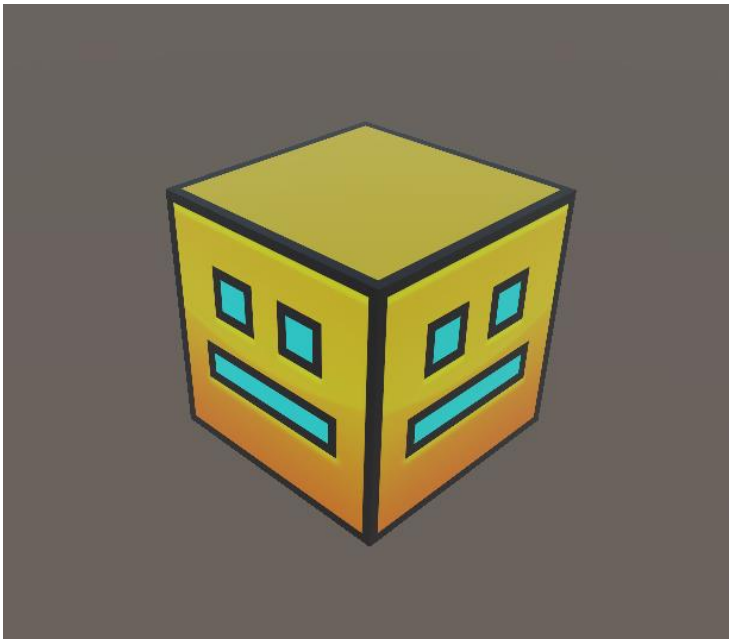

Рисунок 4.31 — Розроблений скін на куб

Подібним чином загалом було розроблено 17 скінів для внутрішньоігрового магазину. 8 скінів, які відкриватимуться за досягнутий рекорд (без урахування базового куба, що використовує звичайний матеріал з фіолетовим кольором) та 9 скінів, що відкриватимуться за дорогоцінне каміння.

4.2 Реалізація функціоналу гри

Цей розділ присвячений практичній реалізації гри: створенню ієрархії ігрових об'єктів, реалізації її основного функціоналу, ключових механік та написання коду, що формують ігровий досвід.

4.2.1 Створення та конфігурація об'єктів на сценах

Для початку створимо всі потрібні об'єкти та налаштуємо їх і утворимо загальну ієрархію об'єктів на кожній сцені, щоб потім написати потрібні скрипти для цих об'єктів для перетворення їх у функціональний продукт.

У навчальній сцені була сформована та налаштована наступна ієрархія об'єктів, яка зображена на рисунку 4.32.

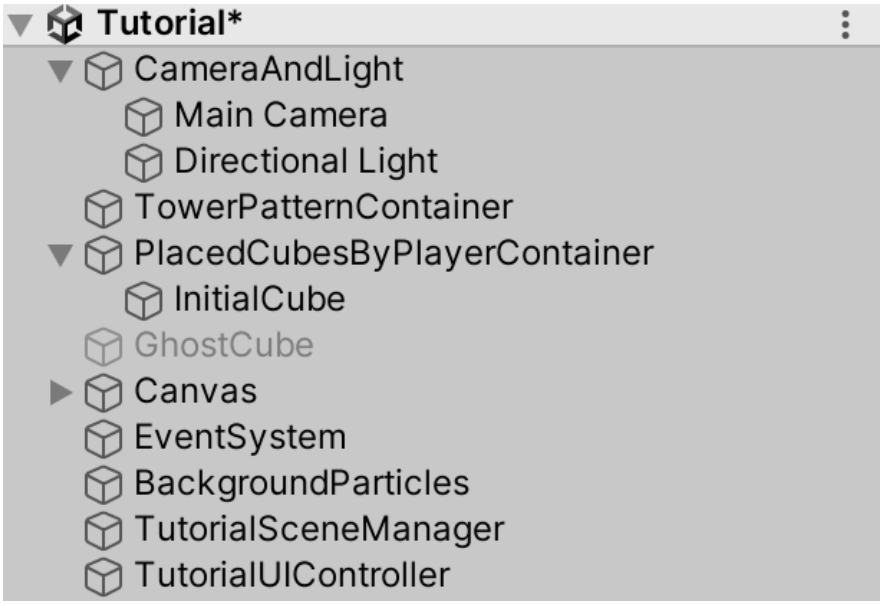

Рисунок 4.32 — Створена ієрархія ігрових об'єктів на навчальній сцені

На самому початку на сцені буде розташовано початковий куб (InitialCube) який знаходиться у контейнері PlacedCubesByPlayerContainer, у цьому контейнері будуть розташовуватися всі розміщенні куби під час навчання гравця. Об'єкт GhostCube являється кубом, який буде показувати гравцю позиції для розміщення кубів. На початку він буде вимкнений, а потім коли гравець дійде до місця коли він буде потрібен, він буде увімкнений. Об'єкт-контейнер CameraAndLight містить Основну камеру сцени, а також її світло. У наступному розділі для цього об'єкта будуть написані скрипти для повороту та зуму камери, а також скрипт для відстеження тапу, для того, щоб навчити гравця цих функцій. Об'єкт-контейнер TowerPatternContainer буде виступати чистим контейнером, у якому будуть розміщуватися всі об'єкти (куби) споруди, яку треба буде заповнити гравцю під час

навчання. Об'єкт Canvas містить весь попередньо створений інтерфейс цієї сцени, об'єкт EventSystem, як ми говорили вище, потрібен для обробки вводу користувача та управління подіями в UI, а об'єкт BackgroundParticles, який містить компонент ParticleSystem, виступає ефектом для фону, який генерує випадкові частинки на фоні, для створення більшої динаміки та привабливості зовнішнього вигляду сцени. TutorialSceneManager та TutorialUIController об'єкти для яких у наступному розділі будуть написані відповідні скрипти, TutorialSceneManager — менеджер для загального керування сценою, та TutorialUIController — для контролю користувацького інтерфейсу цієї сцени.

У сцені меню була сформована та налаштована наступна ієрархія об'єктів, яка зображена на рисунку 4.33.

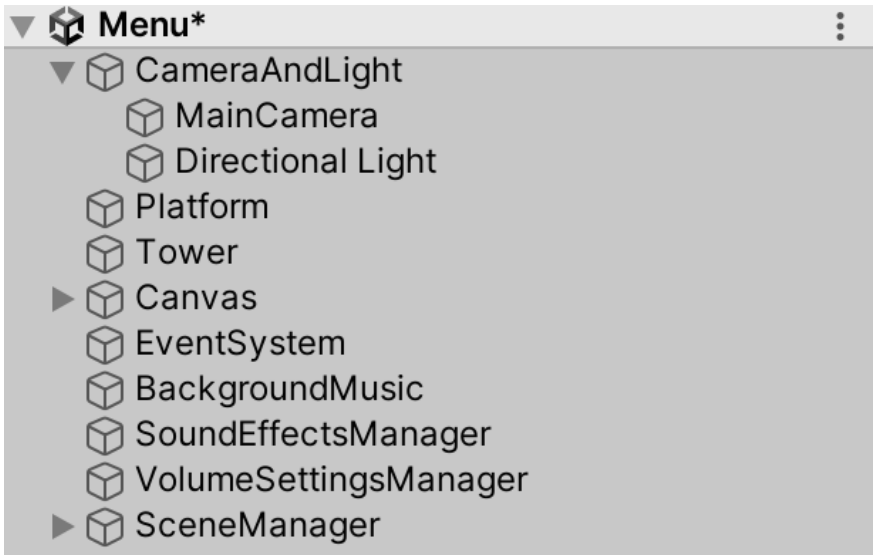

Рисунок 4.33 — Створена ієрархія ігрових об'єктів на сцені меню

У сцені меню на початку буде розташована споруда на платформі, які виступають об'єктами Tower та Platfrom в ієрархії. Як і в навчальній сцені, у меню теж був створений об'єкт-контейнер CameraAndLight, який містить основну камеру сцени та її світло. Цей об'єкт-контейнер містить скрипт з анімацією, за допомогою якої камера красиво пролітає навколо споруди, надаючи динамічності до сцени. Canvas містить інтерфейс цієї сцени, EventSystem обробляє введення користувача. Об'єкти BackgroundMusic та SoundEffectsManager будуть містити відповідні

скрипти для управління фоновою музикою та звуковими ефектами у грі відповідно. Об'єкт VolumeSettingsManager буде містити скрипт-менеджер, який під час заходу у гру, буде регулювати початкову гучність музики та звукових ефектів гри до тих значень, які встановлював гравець під час своєї крайньої гри. А об'єкт SceneManager, буде відповідати за менеджмент сцен, а саме, він буде містити спеціальний скрипт-менеджер, який буде відповідати за перехід між сценами. Цей об'єкт також містить скрипт-анімацію та компонент Canvas, для створення красивої та плавної анімації переходу між сценами.

У сцені ігрового процесу була сформована та налаштована наступна ієрархія об'єктів, яка зображена на рисунку 4.34.

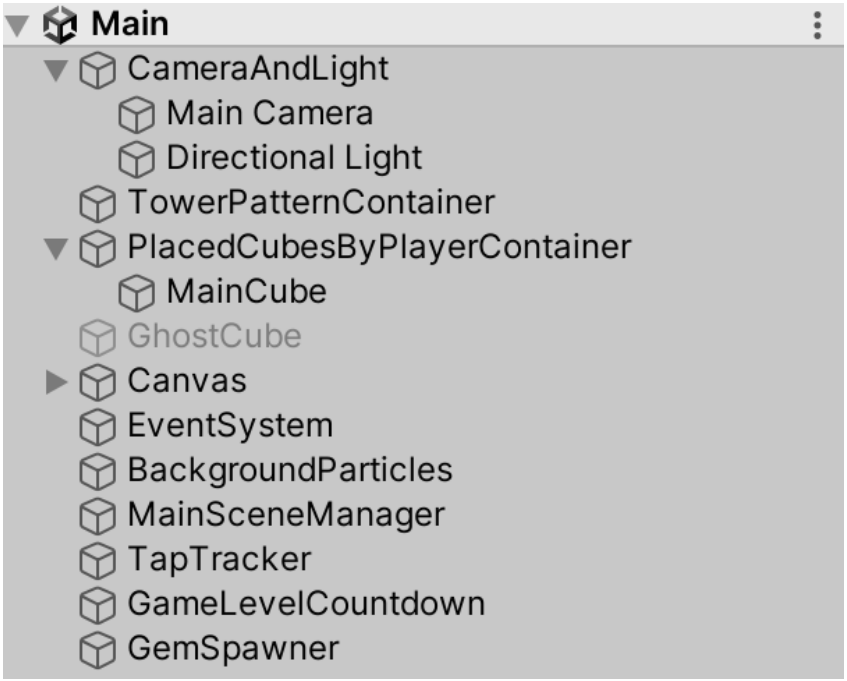

Рисунок 4.34 — Створена ієрархія ігрових об'єктів на сцені ігрового процесу

На самому початку на сцені ігрового процесу буде розташовано початковий куб (InitialCube), з якого буде починатися генерація позицій для майбутніх кубів, він знаходиться у контейнері PlacedCubesByPlayerContainer, у цей контейнер будуть поміщатися всі розміщенні куби гравцем під час гри. Об'єкт GhostCube це звичайний куб, який буде виступати так званим візуалізатором позицій, він буде показувати гравцю позицію у яку гравець може розмістити куб. На початку гри він

буде вимкнений, до тих пір поки не згенерується споруда та не пройде відлік перед стартом гри. Він буде вимикатися після кожного успішного завершення рівня, та вмикатися на старті наступного рівня. Об'єкт-контейнер CameraAndLight містить камеру сцени та її світло. Як вже говорилося вище, далі для цього об'єкта будуть написані скрипти для повороту та зуму камери, для того, щоб гравцю було зручно розуміти обстановку в ігровому полі. Об'єкт-контейнер TowerPatternContainer являється контейнером для зберігання всіх об'єктів (кубів) споруди, які будуть генеруватися та спавнитися перед стартом кожного рівня. Після кожного успішного заповнення гравцем якоїсь частини (куба) цієї споруди, цей об'єкт (куб) буде знищуватися. Об'єкт Canvas містить весь інтерфейс сцени. Далі для об'єкта Canvas буде написано та прикріплено скрипт, який буде відповідати за контроль користувацького інтерфейсу цієї сцени. Об'єкт BackgroundParticles, який містить компонент ParticleSystem, буде мати таку ж саму роль як і у Навчальній сцені, він буде генерувати частинки на фоні ігрового процесу, для створення більшої привабливості та естетики. Об'єкт TapTracker буде мати скрипт, який буде точно відстежувати тап гравця, за допомогою якого гравець буде мати можливість розміщувати куби. Об'єкт MainSceneManager буде менеджером ігрового процесу. Він буде мати два скрипти, один буде контролювати механіку рівнів, а інший буде виконувати загальний контроль ігрового процесу. Об'єкт GameLevelCountdown буде мати скрипт, який буде обробляти механіку часу яке залишилось на рівні. Об'єкт GemSpawner буде виступати спавнером дорогоцінних каменів під час гри, для цього для нього буде написано спеціальний скрипт, який буде мати свою логіку для цього.

У сцені ігрового магазину також була створена та налаштована невелика ієрархія об'єктів, яка показана на рисунку 4.35. Ігровий магазин як і усі інші сцени, має основну камеру, світло та EventSystem значення яких не треба пояснювати. Об'єкт Canvas містить весь інтерфейс ігрового магазину, включаючи його товари (скінами на куби), які теж являються частиною інтерфейсу. А об'єкт ShopManager буде мати скрипт-менеджер, який буде керувати основним функціоналом ігрового магазину.

Рисунок 4.35 — Створена ієрархія ігрових об'єктів на сцені ігрового магазину

4.2.2 Розробка програмного коду гри

Розробка програмного коду гри є одним із ключових етапів створення інтерактивного проєкту, що перетворює ідеї, концепції та просто статичні об'єкти у працюючий продукт.

Нижче наведені деякі основні скрипти які були розроблені для цього проєкту.

У лістингу 4.2 наведений код, який прикріплений до об'єкта SceneManager у сцені меню. Цей менеджер відповідає за функціонал переходу між всіма сценами гри, а також за красиву анімацію переходу. Цей об'єкт не буде знищуватися при переході між сценами. А у лістингах 4.3 та 4.4 наведений допоміжний код для повноцінної та структурованої роботи цього менеджеру. А саме, на лістингу 4.3 наведений абстрактний клас, який полегшує роботу з менеджментом потрібних анімацій для входу до сцени та для виходу зі сцени. А на лістингу 4.4 наведений код функціоналу анімації «згасання», яка буде використовуватися під час виходу зі сцени та під час заходу. Завдяки такій реалізації, в майбутньому, ми зможемо в будь-який момент розробити іншу анімацію, наприклад тільки для входу на сцену і легко інтегрувати її з цим менеджером.

Лістинг 4.2 — SceneManager.cs

```
using System.Collections;
using System.Linq;
using UnityEngine;

public class SceneManager : MonoBehaviour
{
    [SerializeField] private Canvas _thisCanvas;
```

```
        public static SceneManager Instance;
        public GameObject transitionsContainer;
        private SceneTransition[] transitions;

        private void Awake()
        {
            if (Instance == null)
            {
                Instance = this;
                DontDestroyOnLoad(gameObject);
            }
            else
            {
                Destroy(gameObject);
            }
        }

        private void Start()
        {
            transitions                                                    =
transitionsContainer.GetComponentsInChildren<SceneTransition>();
        }

        public    void    LoadScene(string    sceneName,    string
transitionName)
        {
            StartCoroutine(LoadSceneAsync(sceneName,
transitionName));
        }

        private IEnumerator LoadSceneAsync(string sceneName, string
transitionName)
        {
            _thisCanvas.enabled = true;
            SceneTransition  transition  =  transitions.First(t  =>
t.name == transitionName);
            AsyncOperation                    scene                   =
UnityEngine.SceneManagement.SceneManager.LoadSceneAsync(sceneName);
            scene.allowSceneActivation = false;

            yield return transition.AnimateTransitionIn();
            do
            {
                yield return null;
            }
            while (scene.progress < 0.9f);

            scene.allowSceneActivation = true;
            yield return transition.AnimateTransitionOut();
            _thisCanvas.enabled = false;
        }
    }
```

Лістинг 4.3 — SceneTransition.cs

```
using UnityEngine;

public abstract class SceneTransition : MonoBehaviour
{
    public abstract IEnumerator AnimateTransitionIn();
    public abstract IEnumerator AnimateTransitionOut();
}
```

Лістинг 4.4 — CrossFadePrimeTween.cs

```
using PrimeTween;
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

public class CrossFadePrimeTween : SceneTransition
{
    [SerializeField] private CanvasGroup _canvasGroup;
    private Image _image;

    private void Awake()
    {
        _image = GetComponent<Image>();
    }

    public override IEnumerator AnimateTransitionIn()
    {
        _image.raycastTarget = true;
        yield return Tween.Alpha(_canvasGroup, 1f, 0.25f,
Ease.InOutSine).ToYieldInstruction();
    }

    public override IEnumerator AnimateTransitionOut()
    {
        _image.raycastTarget = false;
        yield return Tween.Alpha(_canvasGroup, 0f, 0.25f,
Ease.InOutSine).ToYieldInstruction();
    }
}
```

У лістингу 4.5 та 4.6 наведені скрипти, які прикріплені до об'єкта SoundEffectsManager у сцені меню. Ці скрипти відповідають за управління звуковими ефектами у грі. Скрипт SoundEffectsManager (див. лістинг 4.5) — це основний скрипт для менеджменту звукових ефектів у грі. Він являється сінглтоном, а також має метод DontDestroyOnLoad(gameObject), який робить об'єкт SoundEffectsManager не знищуваним при переході між сценами, бо цей менеджер

призначений для управління звуковими ефектами у всіх сценах. А скрипт у лістингу 4.6 являється допоміжним для нього, який виступає так званою бібліотекою для звукових ефектів (Аудіо кліпів), які можна з легкістю додавати у цю бібліотеку через Інспектор Unity.

Лістинг 4.5 — SoundEffectsManager.cs

```
using UnityEngine;

public class SoundEffectsManager : MonoBehaviour
{
    public static SoundEffectsManager Instance;

    [SerializeField] private SoundEffectsLibrary _soundLibrary;
    [SerializeField] private AudioSource _audioSource;

    private void Awake()
    {
        if (Instance != null)
        {
            Destroy(gameObject);
        }
        else
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
    }

    public void PlaySoundEffectOneShot(string soundEffectName)
    {

_audioSource.PlayOneShot(_soundLibrary.GetAudioClipByName(soundEffec
tName));
    }
}
```

Лістинг 4.6 — SoundEffectsLibrary.cs

```
using UnityEngine;

[System.Serializable]
public struct AudioClips
{
    public string name;
    public AudioClip clip;
}

public class SoundEffectsLibrary : MonoBehaviour
```

```
{
    public AudioClips[] _audioClips;

    public AudioClip GetAudioClipByName(string soundEffectName)
    {
        foreach (var soundEffect in _audioClips)
        {
            if (soundEffect.name == soundEffectName)
            {
                return soundEffect.clip;
            }
        }
        return null;
    }
}
```

Для сцени ігрового процесу були написані основні скрипти для відстежування та обробки всього введення гравця під час гри. А саме, у лістингу А.1 наведений скрипт який відповідає за відстежування таких жестів пальців як swipe та pinch-zoom гравця та управляє поворотом камери по вертикалі, горизонталі, а також її функціоналом наближення та віддалення від споруди. Завдяки цьому скрипту, гравець зможе простими жестами пальців, управляти видом на споруду. А у лістингу А.2 наведений скрипт, який відстежує саме жест тапу по екрану, за допомогою якого гравець зможе розміщувати нові куби. Скрипт лістингу А.1 прикріплений до об'єкта CameraAndLight, а скрипт лістингу А.2 до об'єкта-контейнера TapTracker.

У лістингу Б.1 наведений скрипт який відповідає за весь функціонал спавну дорогоцінних каменів під час гри. Він прикріплений до об'єкта GemSpawner сцени ігрового процесу. А у лістингу Б.2 наведений скрипт який прикріплений до об'єкта GhostCube у сцені ігрового процесу, він відповідає за контроль цього об'єкта, а саме, він працює як візуалізатор позицій для гравця, у які він може розмістити наступний куб.

У лістингу В.1 наведений скрипт, який прикріплений до об'єкта Canvas в ігровій сцені. Цей скрипт (див. лістинг В.1) реалізовує функціонал відліку перед початком кожного рівня, який буде показувати цифри відліку, а також починати наступний рівень.

Скрипти, які наведено у лістингу 4.7 та 4.8 належать до сцени Магазину, а саме, вони формують функціонал системи вкладок у магазині, за допомогою якої, гравець може зручно та швидко переходити між вкладками з різними типами скинів.

Лістинг 4.7 — TabGroup.cs

```
using System.Collections.Generic;
using UnityEngine;

public class TabGroup : MonoBehaviour
{
    [HideInInspector] public List<TabButton> tabButtons;
    public Sprite tabIdle, tabActive;
    public List<GameObject> PagesToSwap;

    public void Subscribe(TabButton button)
    {
        if(tabButtons == null)
            tabButtons = new List<TabButton>();
        tabButtons.Add(button);
    }

    public void OnTabSelected(TabButton button)
    {
        ResetTabs();


        SoundEffectsManager.Instance.PlaySoundEffectOneShot("But
        ton");

        button.background.sprite = tabActive;

        int index = button.transform.GetSiblingIndex();
        for (int i = 0; i < PagesToSwap.Count; i++)
        {
            if (i == index)
                PagesToSwap[i].SetActive(true);
            else
                PagesToSwap[i].SetActive(false);
        }
    }

    private void ResetTabs()
    {
        foreach (TabButton button in tabButtons)
            button.background.sprite = tabIdle;
    }
}
```

Лістинг 4.8 — TabButton.cs

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

[RequireComponent(typeof(Image))]
public class TabButton : MonoBehaviour, IPointerDownHandler
{
    public TabGroup tabGroup;
    [HideInInspector] public Image background;

    void Start()
    {
        background = GetComponent<Image>();
        tabGroup.Subscribe(this);
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        tabGroup.OnTabSelected(this);
    }

}
```

4.3 Інтеграція внутрішньоігрової реклами

Першим етапом в інтеграції внутрішньоігрової реклами Google AdMob в Unity, було створення облікового запису на цій платформі. Після цього треба було підтвердити особисту інформацію й адресу проживання.

На рисунку 4.36 зображено створений та налаштований додаток з його згенерованим ідентифікатором у Google AdMob.

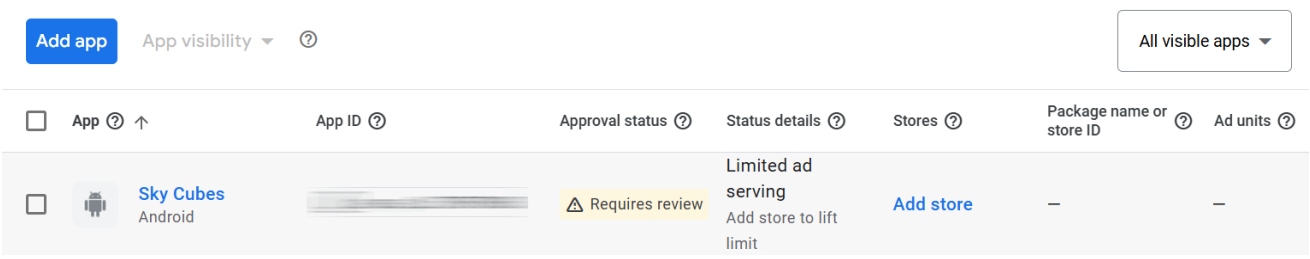

Рисунок 4.36 — Створений та налаштований додаток у Google AdMob

Далі на сайті Google AdMob, був створений та налаштований новий додаток для розроблюваної гри, а також був згенерований унікальний ідентифікатор цього додатка, який потрібен для ідентифікації додатка у системі AdMob, щоб реклама відображалася правильно, і AdMob міг збирати статистику про її ефективність. Цей ідентифікатор буде далі інтегруватися в Unity.

Наступним етапом було створення та налаштування рекламних блоків для цього додатка у Google AdMob. Як ми вже говорили вище, у цьому проєкті буде три місця для показу внутрішньоігрової реклами, а саме, 2 місця для показу реклами з винагородою й одне місце для показу проміжної реклами. Для кожного такого місця треба було створити по рекламному блоку та налаштувати його. Загалом було створено 3 рекламних блоки, які зображено на рисунку 4.37.

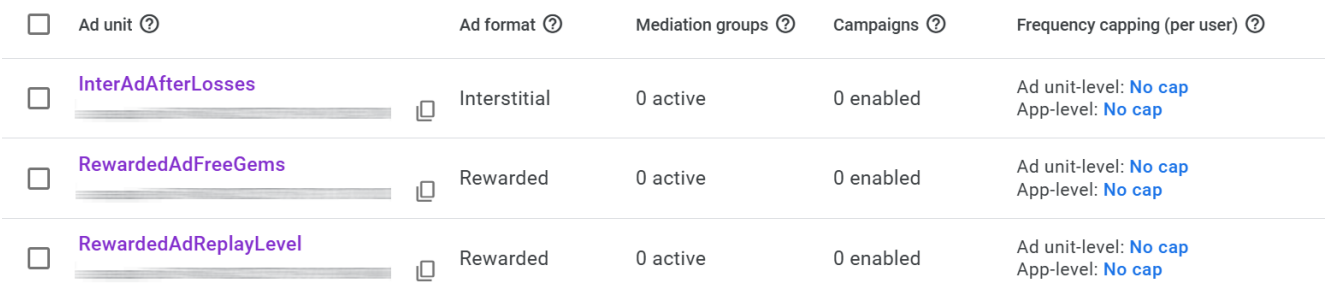

Рисунок 4.37 — Створені та налаштовані рекламні блоки у додатку на Google AdMob

На рисунку 4.38 зображено приклад налаштованого рекламного блоку. Цей рекламний блок був створений та налаштований для показу проміжної реклами в Ігровій сцені зразу після програшу, якщо гравець зазнав п'яти програшів. У налаштуваннях цього блоку, в комірці формату реклами був вибраний формат проміжної реклами, також було придумано йому назву, вибрано тип реклами (текст, картинка, мультимедійні засоби, відео), а також вибрано eCPM floor — Google Optimized, тобто Google буде динамічно встановлювати поверхи на основі даних користувачів. eCPM floor — це мінімальна ціна за клік, якій повинна відповідати мережа AdMob і рекламні джерела, щоб показати оголошення. Подібним чином були налаштовані інші створені рекламні блоки.

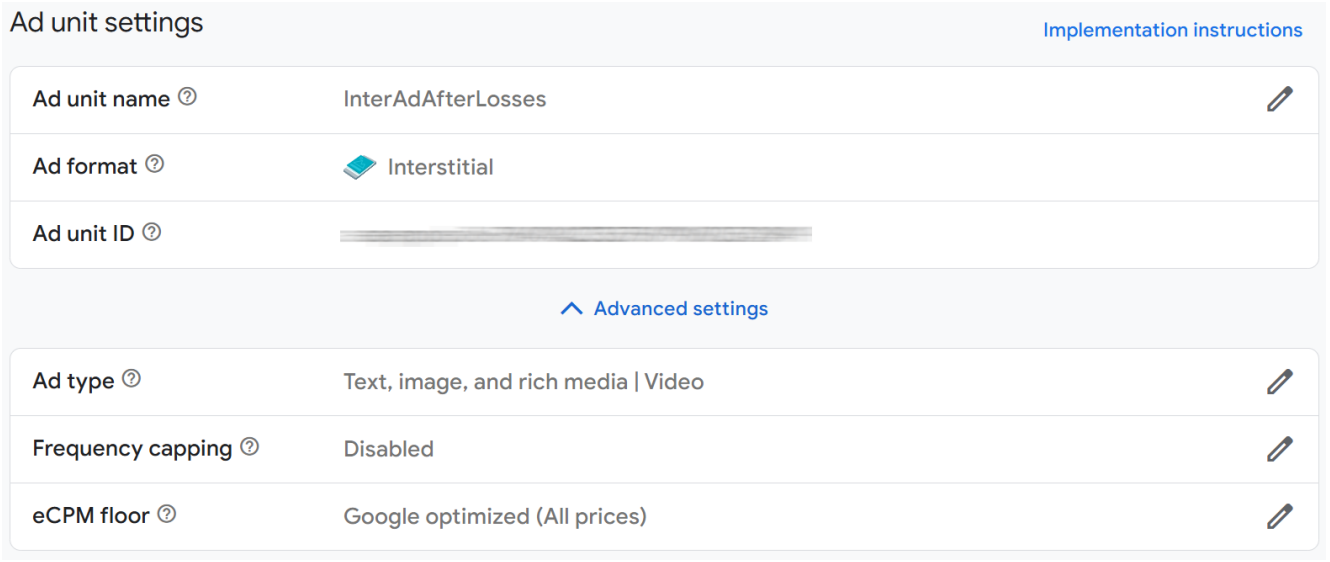

Рисунок 4.38 — Створений та налаштований рекламний блок для проміжної реклами після програшу

Після створення додатка, його рекламних блоків та їхнього налаштування, настав час до самої інтеграції реклами в Unity. Як ми говорили в розділі аналізу рекламних мереж, Google AdMob дуже просто інтегрується до Unity. Для цього у Google AdMob є плагін спеціально призначений для Unity, його можна знайти на офіційному акаунті GoogleAds у GitHub [54]. Після звантаження, його було імпортовано в Unity. Потім були налаштовані деякі залежності у вкладці Publishing Settings проєкту Unity, для правильної роботи Google AdMob, це зображено на рисунку 4.39 [55].

Рисунок 4.39 — Налаштовані залежності проєкту для правильної роботи AdMob

Далі у налаштуваннях плагіну Google Mobile Ads в Unity був введений згенерований ідентифікатор додатка, попередньо створеного на сайті Google AdMob, це зображено на рисунку 4.40.

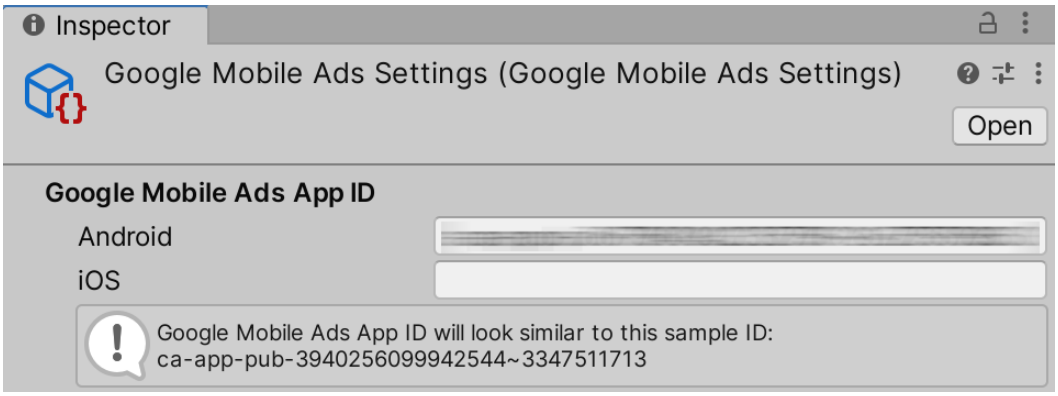

Рисунок 4.40 — Введений Android AdMob app ID у налаштуваннях плагіну

Після виконання всіх попередніх дій, Google AdMob повністю імпортований, налаштований та готовий до використання в Unity. Далі, для показу реклами у грі за допомогою імпортованого плагіну, було створено декілька об'єктів на сценах, а також написано кілька скриптів, які будуть реалізовувати логіку та функціонал показу всієї внутрішньоігрової реклами у грі.

У сцені меню, був створений об'єкт AdManager, для якого був написаний скрипт-менеджер реклами, код якого наведений у лістингу Г.1. Цей менеджер являє собою клас-сінглтон, який не буде знищуватися під час переходу між сценами, завдяки виклику метода DontDestroyOnLoad() при його ініціалізації. По-перше, цей клас-менеджер ініціалізує мережу Google AdMob у момент запуску гри, бо це необхідно для її роботи, це потрібно зробити лише один раз під час роботи гри. По-друге, він керує підвантаженням всіх типів реклами, для створення кешу реклами, для їх більш швидкого показу в момент коли це буде потрібно, керує їх готовністю до показу (перевіряє чи є інтернет-з'єднання, чи підвантажилася вона повністю і так далі), а також керує звільненням пам'яті вже використаної реклами. У цьому класі також містяться змінні, які зберігають спеціальні ідентифікатори рекламних блоків, ці ідентифікатори використовуються під час запиту на завантаження реклами в Google AdMob. Тимчасово, під час розробки, використовуються

спеціальні тестові ідентифікатори, для показу тестової реклами. Кожний тип реклами (наприклад тип реклами з винагородою), має свій спеціальний тестовий ідентифікатор. Всі ці тестові ідентифікатори можна знайти в офіційній документації Google AdMob. Це потрібно для того, щоб не порушувати правила Google і не запитувати реальну рекламу під час розробки, тому, що за правилами Google, це може призвести до блокування акаунту Google AdMob. Далі, після повного завершення розробки проєкту, для реального тестування реклами, будуть використовуватися реальні ідентифікатори рекламних блоків які можна знайти в акаунті Google AdMob, але для цього, ще потрібно буде налаштувати тестовий-пристрій. Реальні ідентифікатори рекламних блоків, також використовуються під час випуску гри у світ. Цей скрипт був прикріплений до об'єкта AdManager у сцені меню.

Також для сцени ігрового процесу та сцени ігрового магазину, були написані класи-контролери та їх допоміжні скрипти, які мають логіку для показу внутрішньоігрової реклами на цих сценах відповідно. Наприклад, у сцені ігрового процесу, після програшу, спочатку буде відкриватися вікно з пропозицією подивитися рекламу, в обмін на перегравання рівня (якщо вже є наявна підвантажена реклама), якщо гравець погодиться, то буде показана реклама з винагородою, а якщо гравець не погодиться, то буде перевірятися чи гравець зазнав 5 програшів, якщо так, то буде показана проміжна реклама. Після показу проміжної реклами, цей лічильник буде скинутий і реклама не буде показуватися до тих пір, поки гравець знову не програє 5 разів.

У сцені ігрового магазину все набагато простіше, якщо гравець захоче отримати безкоштовні дорогоцінні камені, йому треба просто тапнути на спеціальну кнопку і гравцю буде показано рекламу з винагородою, після перегляду якої, він отримає свої бажані безкоштовні дорогоцінні камені. Для цієї кнопки був написаний спеціальний скрипт, який наведено у лістингу Г.2. Він відповідає за зовнішній вигляд та функціонування кнопки. Бо, для показу реклами, потрібне інтернет з'єднання, у цьому скрипті (див. лістинг Г.2), кожні пару секунд, перевіряється інтернет з'єднання, і якщо воно пропаде, або якщо його не буде під

час заходу на цю сцену, кнопка буде вимкнена (гравець не зможе на неї тапнути), а також спрайт кнопки буде змінено на спеціальний спрайт, який буде давати знати гравцю, що для того, щоб скористатися цією кнопкою й отримати безкоштовні дорогоцінні камені за перегляд реклами, гравцю спочатку потрібно перевірити інтернет з'єднання. Якщо інтернет з'єднання відновиться, кнопку буде увімкнено назад і спрайт кнопки буде змінено на основний. Такий підхід покращує користувацький досвід, адже гравець одразу бачить, чому кнопка недоступна, і може швидко виправити ситуацію, відновивши інтернет-з’єднання. Це не лише робить гру зручнішою, але й стимулює гравців користуватися функцією, що може збільшити дохід від реклами й підвищити лояльність до гри.

В результаті, після виконання всіх вище перерахованих дій та написання коду, інтеграція внутрішньоігрової реклами Google AdMob у гру була успішно завершена. На рисунку 4.41 зображений приклад успішного показу тестової проміжної реклами у сцені ігрового процесу, після того, як гравець зазнав 5 програшів.

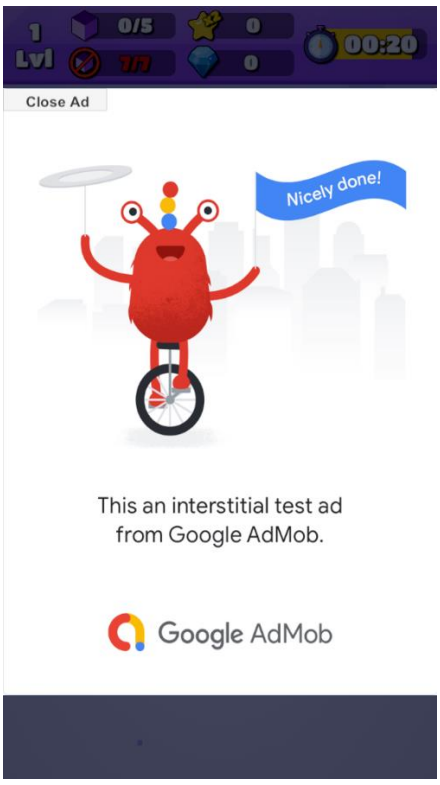

Рисунок 4.41 — Успішний показ тестової проміжної реклами після програшу у сцені ігрового процесу

5 ТЕСТУВАННЯ ТА ОПТИМІЗАЦІЯ ГРИ

Важливо розуміти, що процес розробки гри не завершується після написання коду. Тестування та оптимізація є невід'ємною частиною, що забезпечує стабільну роботу гри, її зручність для користувача та відповідність технічним вимогам. Цей розділ охоплює основні етапи тестування і виявлення помилок, а також процеси оптимізації, які дозволяють покращити продуктивність гри, зменшити час завантаження та забезпечити безперебійну гру навіть на менш потужних пристроях. Тільки після цих етапів можна досягти максимальної якості продукту, готового до випуску.

5.1 Тестування гри

Спочатку буде проводитися тестування гри, для того, щоб розуміти, що і як треба оптимізувати.

5.1.1 Тестування основного функціоналу гри та ігрового процесу

Тестування гри починається з тестування основних механік, її ігрового процесу, балансу гри, а також виявлення помилок, якщо вони є. Таке тестування проводилось у реальних умовах, тобто гра була побудована (builded) під платформу Android і тестувалася на реальному смартфоні.

Під час тестування основного ігрового процесу та балансу гри, гра відчувалася трошки складнішою, ніж була задумана. Це було пов'язано з загальним балансом гри на рівнях, а саме, кількість кубів у вежі та кількість допустимих помилок занадто швидко збільшувалася з досягненням кожного нового рівня, це призводило до того, що, наприклад, уже на 7 рівні, гравцеві потрібно було заповнити вежу з 20 кубів, водночас маючи лише 2 допустимі помилки. Це було збалансовано, для досягнення більш повільного збільшення складності гри під час

переходу з рівня на рівень. Це було досягнуто шляхом зменшення константних значень які відповідають за збільшення кількості кубів і зменшення кількості помилок під час переходу з певного рівня на наступний.

А під час тестування функціоналу гри та її механік, було помічено, що наближення та віддалення камери жестом pinch-zoom було трохи повільним. Це було виправлено, шляхом збільшення значення змінної _zoomSpeed скрипта CameraInputController.cs (див. лістинг А.1).

Після коригування всіх виявлених недоліків, гру було протестовано ще раз. Результати повторного тестування виявилися дуже позитивними, і жодних нових проблем чи недоліків виявлено не було.

5.1.2 Тестування інтегрованої внутрішньоігрової реклами

Для досягнення максимально реалістичних результатів при тестуванні інтегрованої реклами в мобільну гру, важливо використовувати реальні рекламні блоки в поєднанні з тестовим пристроєм. Це дозволяє не тільки перевірити правильність роботи рекламних інтеграцій, але й забезпечити їх оптимальну продуктивність і безперебійність у реальних умовах. Важливо зазначити, що для тестування з Google AdMob необхідно використовувати тестовий пристрій, щоб уникнути порушення політики реклами та отримати найбільш точні й об'єктивні дані щодо відображення реклами, її завантаження та взаємодії з користувачем. Тільки так можна гарантувати, що реклама працює належним чином і не спричиняє проблем у досвіді користувачів.

Для цього, спочатку, у створеному обліковому записі Google AdMob, у вкладці Test devices, треба було додати тестовий пристрій: для цього треба було ввести назву пристрою, вибрати платформу (у нашому випадку Android), вибрати бажаний жест для відкриття Ad Inspector під час тестування на смартфоні, який має деяку інформацію про показані рекламні блоки та іншу корисну інформацію, а також найважливіше, треба було ввести рекламний ідентифікатор, який знаходиться у налаштуваннях пристрою, на якому збирається проводитися тестування реклами. На рисунку 5.1 зображено результат створення та налаштування тестового пристрою. Тестові пристрої дозволяють безпечно

тестувати виробничі оголошення без ризику порушення політики Google. Завдяки тестовому пристрою, оголошення в мережі AdMob працюватимуть у спеціальному тестовому режимі та відображатимуть бейдж «Test Ad», на яку можна спокійно клікати, не боячись блокування акаунту.

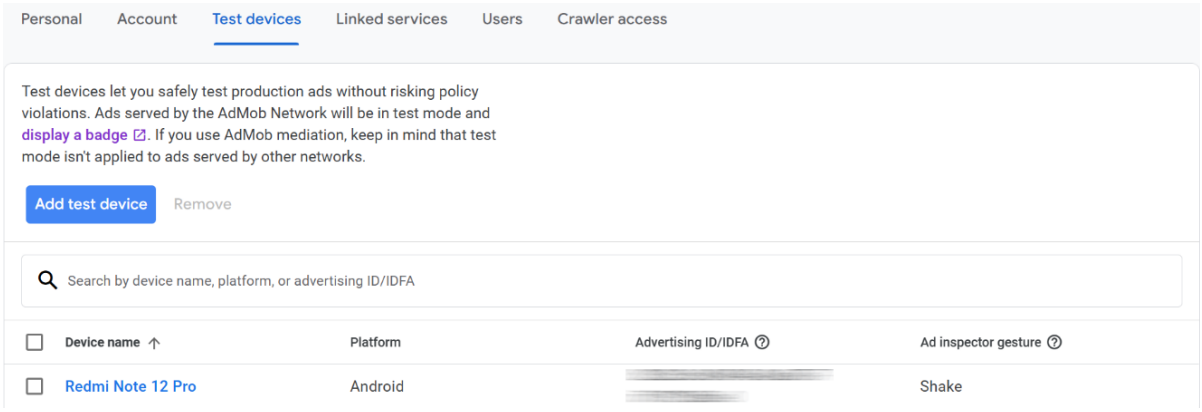

Рисунок 5.1 — Створений та налаштований тестовий пристрій у Google AdMob

Далі у скрипті AdManager.cs (див. лістинг Г.1), тестові ідентифікатори рекламних блоків були змінені на реальні ідентифікатори, які знаходяться у вкладці створених рекламних блоків додатка в обліковому записі Google AdMob. Після цього гра була зібрана та запущена на тестовому пристрої для тестування інтегрованої реклами у гру.

Вся інтегрована внутрішньоігрова реклама, включаючи проміжну рекламу та рекламу з винагородою у сцені ігрового процесу та реклама з винагородою у сцені ігрового магазину була успішна протестована та ніяких помилок, або проблем виявлено не було. Нижче наведені результати успішного тестування інтегрованої внутрішньоігрової реклами.

На рисунку 5.2 зображено успішне відкриття вікна з пропозицією переглянути рекламу в обмін на перегравання рівня після програшу у сцені ігрового процесу. А на рисунку 5.3 зображений результат успішного показу реклами з винагородою після натискання кнопки «дивитися рекламу» у цьому вікні з пропозицією. Після перегляду цієї реклами до кінця, рівень на якому ми зупинилися — був успішно перезапущений.

Рисунок 5.2 — Успішне відкриття вікна з пропозицією переглянути рекламу в обмін на перегравання рівня

Рисунок 5.3 — Результат успішного показу реклами з винагородою у сцені ігрового процесу

А на рисунку 5.4 зображено результат успішного показу проміжної реклами після 5-ти програшів.

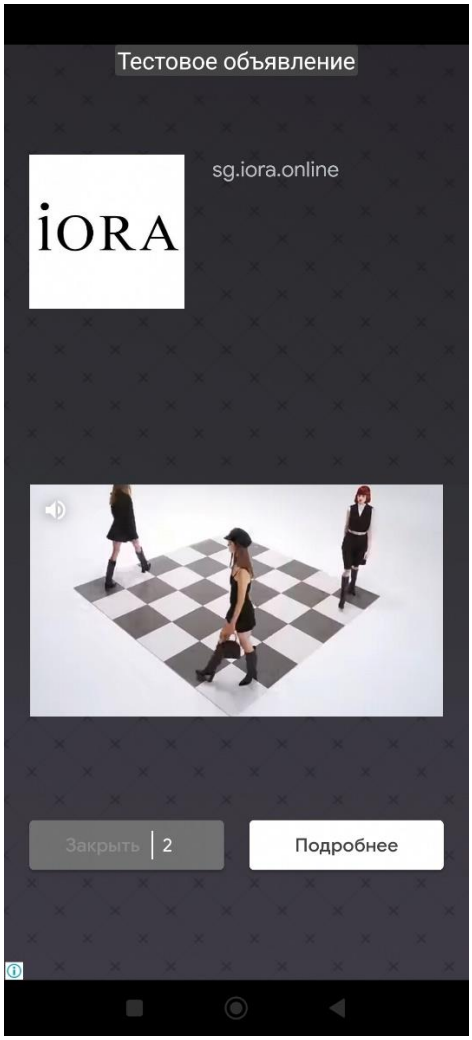

Рисунок 5.4 — Результат успішного показу проміжної реклами після 5-ти програшів у сцені ігрового процесу

На рисунку 5.5 зображено активну кнопку для перегляду реклами в обмін на безкоштовні дорогоцінні камені у сцені ігрового магазину. А на рисунку 5.6 зображено вигляд кнопки, після того, як було перервано інтернет-з'єднання.

Рисунок 5.5 — Активна кнопка для перегляду реклами у сцені ігрового магазину

Рисунок 5.6 — Неактивна кнопка для перегляду реклами у сцені ігрового магазину

На рисунку 5.7 зображено результат успішного показу реклами з винагородою, після натискання спеціальної кнопки у сцені ігрового магазину.

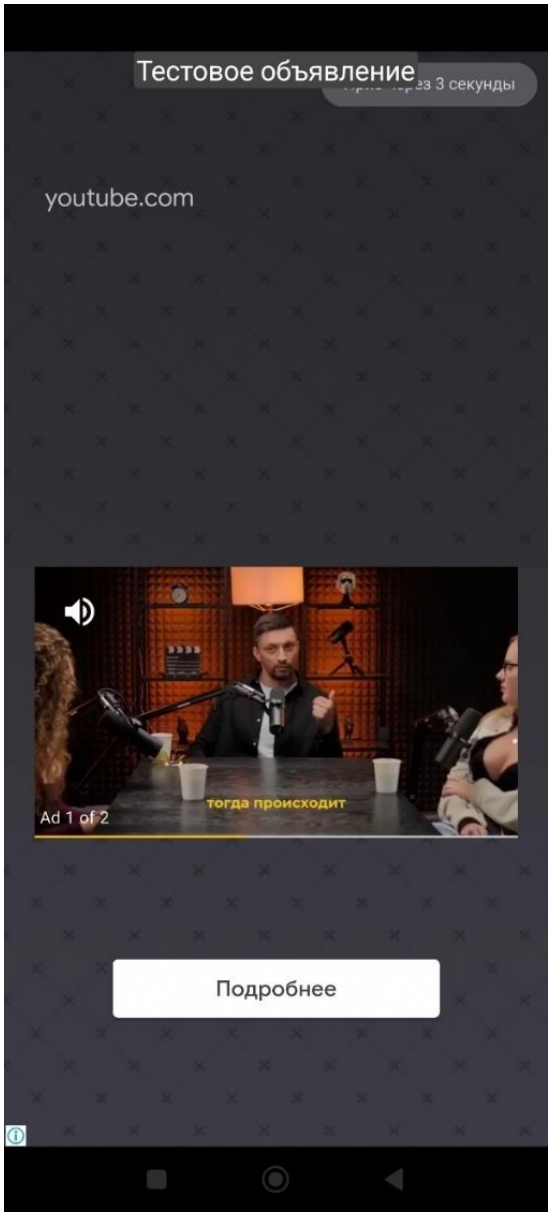

Рисунок 5.7 — Результат успішного показу реклами з винагородою у сцені ігрового магазину

А на рисунку 5.8 зображено результат успішного перегляду цієї реклами до кінця (див. рисунок 5.7), а саме, там показано вікно успішного отримання п'яти дорогоцінних каменів після перегляду реклами.

Рисунок 5.8 — Результат успішного отримання п'яти дорогоцінних каменів після перегляду реклами з винагородою у сцені ігрового магазину

5.1.3 Тестування перформансу гри

Тестування перформансу гри це дуже важливий етап тестування, адже він дозволяє виявляти та усувати затримки, збої, високий frame time чи проблеми з завантаженням, та гарантує стабільну роботу і позитивний досвід для гравців. У сучасних умовах високих очікувань гравців, якість роботи гри стає вирішальним фактором її конкурентоспроможності.

Тестування перформансу гри проводилось на смартфоні Xiaomi Redmi Note 12 Pro, за допомогою потужного, вбудованого у Unity інструменту — Unity Profiler в парі з Profile Analyzer.

Основні технічні характеристики смартфону Xiaomi Redmi Note 12 Pro:

– роздільна здатність екрану (screen resolution): 1080x2400 pixels, 20:9 ratio;

– частота оновлення екрану: 120Hz;

– операційна система (OS): Android 12, MIUI 13;

– оперативна пам'ять (RAM): 8GB;

– процесор (CPU): Octa-core (2x2.6 GHz Cortex-A78 & 6x2.0 GHz Cortex-A55);

– графічний процесор (GPU): Mali-G68 MC4.

Інструмент Unity Profiler, який знаходиться у вкладці Window-Analysis — це вбудований інструмент в Unity, який дозволяє аналізувати перформанс гри під час її виконання. Він допомагає виявляти проблеми в різних аспектах гри, таких як робота процесора, пам’яті, графіки, фізики, аудіо та інших систем. Основна мета профайлера — надати детальну інформацію про те, як використовуються ресурси, і допомогти знайти причини уповільнень. Профайлер показує, як різні компоненти гри, включаючи скрипти, рендеринг та фізику, впливають на продуктивність. Наприклад, можна побачити, скільки часу витрачається на обробку кадру, які функції викликаються найчастіше та які ресурси займають найбільше пам'яті. Для глибшого аналізу можна увімкнути «Deep Profiling», щоб побачити деталі викликів методів у коді.

Однією з ключових функцій Unity Profiler є можливість віддаленого профілювання, яка дозволяє аналізувати гру на різних пристроях, наприклад, на смартфонах або консолях. Це особливо корисно для оптимізації під специфічні платформи. Unity Profiler також підтримує розширення, що дає змогу створювати власні модулі для збору специфічних даних.

A Profile Analyzer в Unity — це інструмент, який дозволяє глибше аналізувати дані зібрані за допомогою Unity Profiler. Його основна мета — допомогти оптимізувати продуктивність гри або застосунку, пропонуючи зручний спосіб вивчення продуктивності через порівняння та агрегацію знімків профайлу (Profiler captures). Його основні можливості: об'єднання інформації з кількох кадрів для аналізу середніх, мінімальних, максимальних значень і варіацій виконання, а також можливість порівнювати дві різні сесії профілювання, щоб зрозуміти вплив змін у коді або налаштуваннях.

Тестування проводилося з функцією віддаленого профілювання (Remote Profiling) з увімкненою функцією «Deep Profiling». Тобто, Unity Profiler встановив зв’язок між редактором Unity на комп’ютері та цільовим пристроєм, на якому запускається гра, у нашому випадку зі смартфоном Redmi Note 12 Pro. Підключення

здійснювалося через USB 4.0, але також можна підключатися через локальну мережу Wi-Fi або IP-адресу. Unity Profiler збирав дані у реальному часі під час роботи гри на пристрої. Це дало можливість зрозуміти, як різні компоненти (скрипти, фізика, графіка тощо) використовують ресурси смартфона. Тестування гри за допомогою Remote Profiling має перевагу над звичайним профайлінгом в Unity Editor. При тестуванні у редакторі, гра виконується на потужному комп'ютері, а Unity Editor додає накладні витрати, які не характерні для фінальної гри. Віддалене профілювання дозволяє побачити, як гра працює на цільовій платформі (наприклад, смартфоні чи консолі), де апаратні ресурси можуть бути значно обмеженішими. Тобто спочатку за допомогою Unity Profiler збиралася інформація про продуктивність гри на сценах на цільовому пристрої, потім був зроблений знімок профілю (зібраних даних) кожної сцени та імпортований у Profile Analyzer для безпосереднього детального аналізу.

На рисунку 5.9 зображено приклад збирання інформації перформансу в Unity Profiler на навчальній сцені.

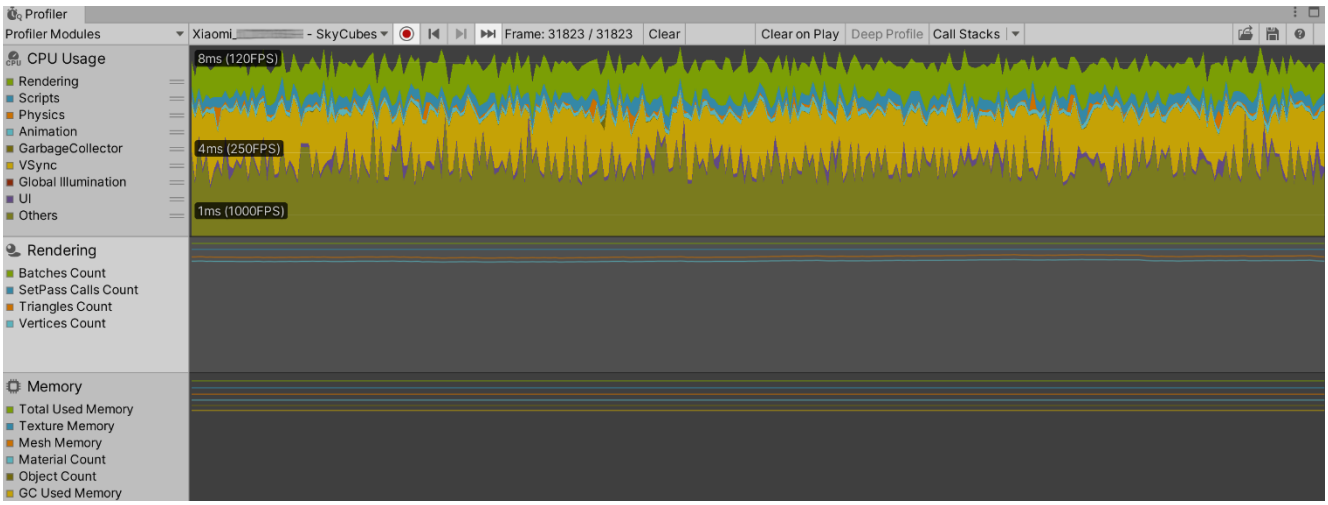

Рисунок 5.9 — Збирання інформації перформансу за допомогою Unity Profiler на навчальній сцені

Таким чином за допомогою Unity Profiler, була зібрана інформація перформансу на кожній сцені на цільовому пристрої. Потім зібрана інформація в Unity Profiler на кожній сцені була імпортована та проаналізована у Profile Analyzer.

На рисунку 5.10 зображено приклад аналізу попередньо зібраної інформації перформансу навчальної сцени у Profile Analyzer.

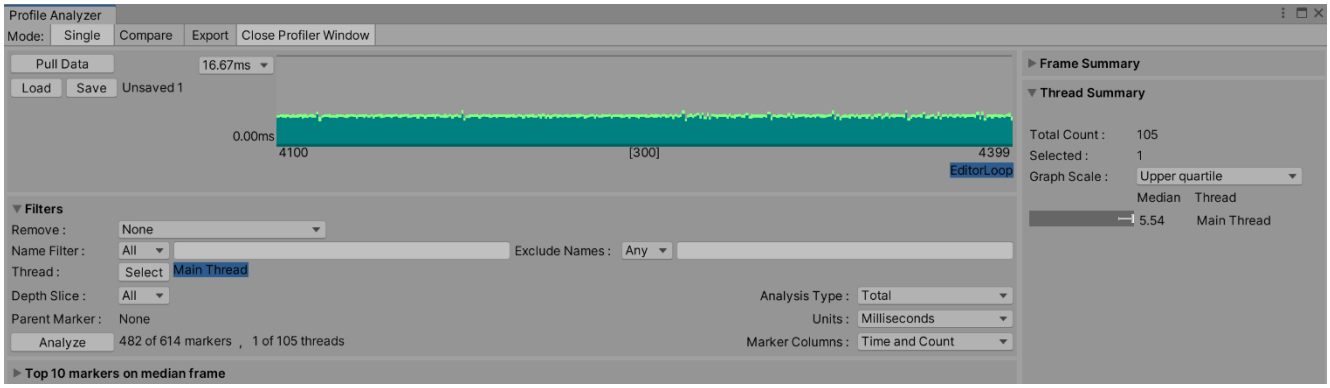

Рисунок 5.10 — Аналіз попередньо зібраної інформації перформансу навчальної сцени у Profile Analyzer

Нижче наведені результати аналізу перформансу на кожній сцені. А саме, на рисунках 5.11, 5.12, 5.13 та 5.14 наведено графіки часу кадрів (frame time) на основні зібраних даних для 300 кадрів (frames) на кожній сцені.

Час кадру (frame time) — це час, який потрібен системі для обробки та рендерингу одного кадру під час виконання гри. Зазвичай цей показник вимірюється в мілісекундах (ms). Наприклад, якщо гра працює зі швидкістю 120 FPS (кадрів на секунду), то середній frame time повинен бути близько 8,33 мс.

Аналіз frame time є важливішим і точнішим показником для оцінки продуктивності гри, ніж просто FPS, оскільки FPS показує лише середнє значення кадрів за секунду і не відображає стабільності. Нестабільний frame time призводить до «заїкання» (stuttering), навіть якщо FPS високий. Графіки часу кадру дозволяють виявляти вузькі місця (наприклад, проблеми з CPU або GPU), пов’язувати затримки з конкретними ігровими подіями та об’єктивно оцінювати оптимізацію. Стабільний frame time забезпечує плавний геймплей і кращий користувацький досвід.

Під час тестування перформансу на навчальній сцені (див. рисунок 5.11), сцені ігрового процесу (див. рисунок 5.12) та ігрового магазину (див. рисунок 5.13) не було помічено якихось проблем з продуктивністю гри, frame time був стабільним та гра йшла дуже плавно та без заїкань, навіть у найбільш інтенсивних моментах.

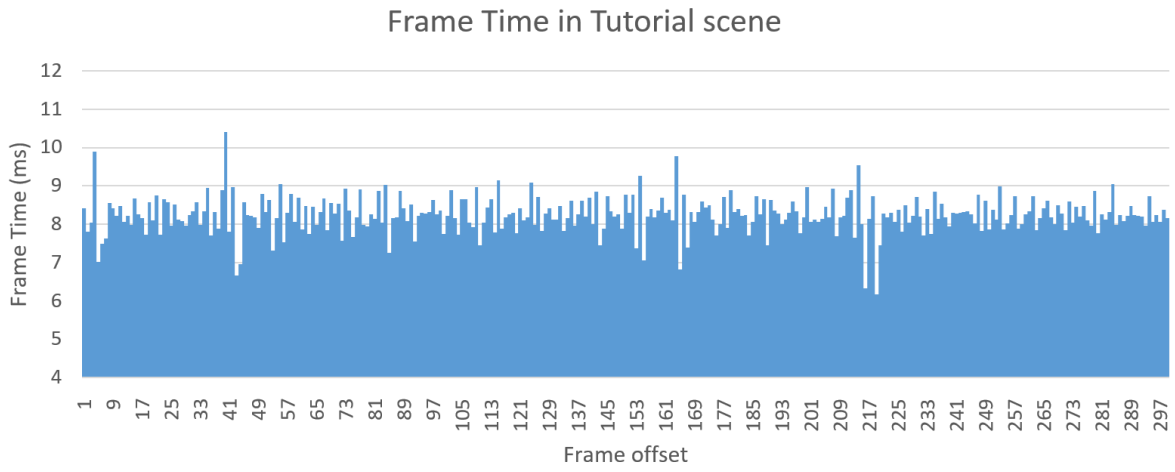

Рисунок 5.11 — Графік часу кадру для 300 кадрів на навчальній сцені

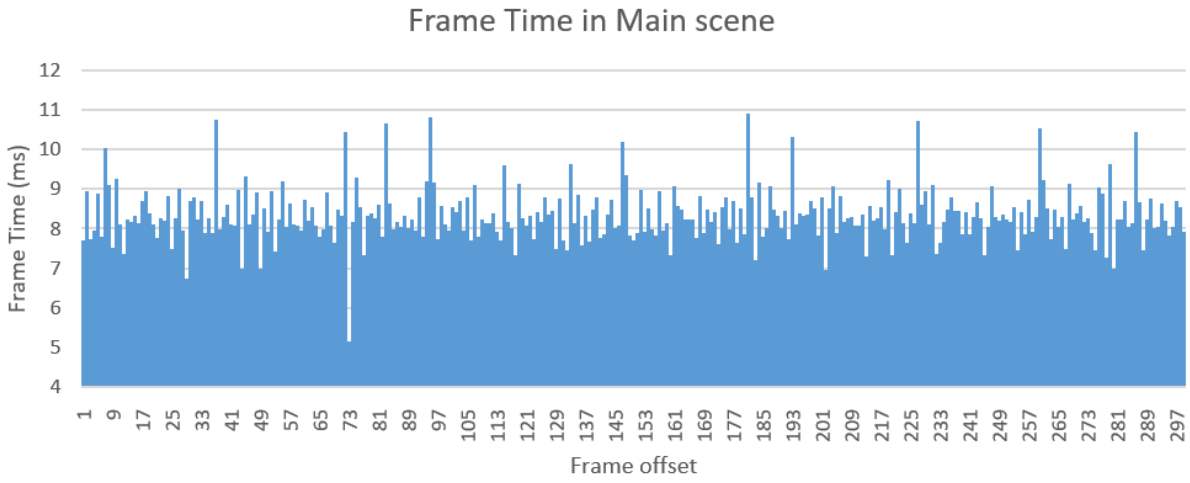

Рисунок 5.12 — Графік часу кадру для 300 кадрів на сцені ігрового процесу

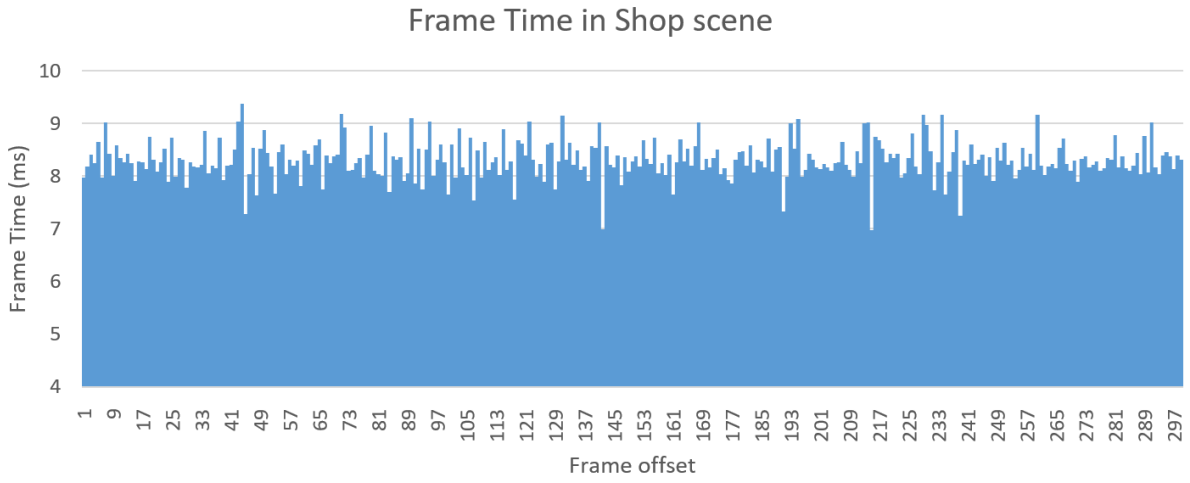

Рисунок 5.13 — Графік часу кадру для 300 кадрів на сцені ігрового магазину

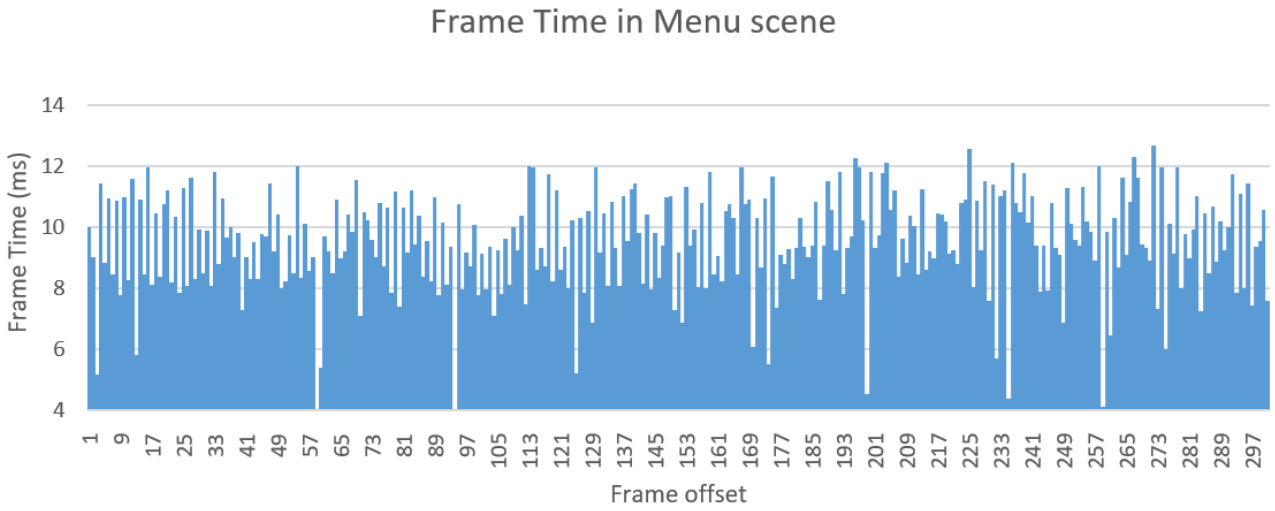

Рисунок 5.14 — Графік часу кадру для 300 кадрів на сцені меню

Також, на рисунку 5.15 зображена стовпчаста діаграма середнього часу кадру (frame time) на основі аналізу даних для 300 кадрів для кожної сцени.

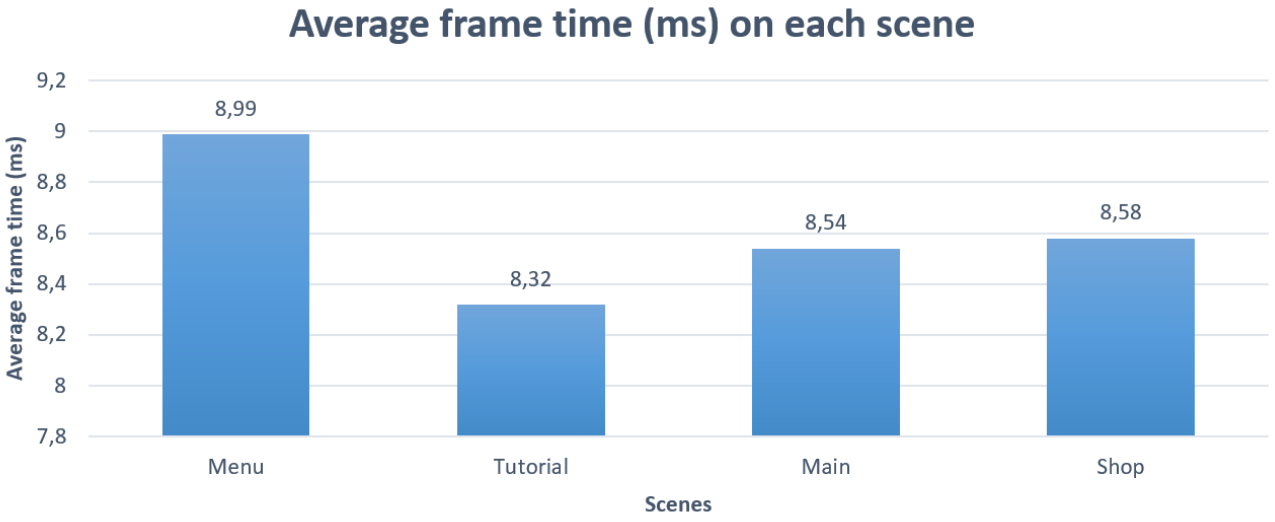

Рисунок 5.15 — Середній час кадру на основі даних 300 кадрів на кожній сцені

По діаграмі (див. рисунок 5.15) можна сказати, що на всіх сценах, окрім сцени Меню, середній час кадру досить низький (чим нижче, тим краще), наприклад у сцені Ігрового процесу, середній час кадру на основі аналізу даних для 300 кадрів дорівнює 8,54 мілісекунди й для більш простого розуміння, якщо перевести цей середній час кадру у FPS (frame per second), за допомогою відповідної формули

(5.1), ми отримаємо ≈ 117 FPS. З огляду на те, що максимальний можливий FPS на цільовому пристрої, на якому проходив тест (Xiaomi Redmi Note 12 Pro), дорівнює 120 FPS, то це відмінний результат. Тобто, якщо піти зворотним шляхом, оптимальний frame time для цільового пристрою, на якому проходив тест перформансу, буде 8,33 мс.

$$\text{FPS} = \frac{1000\ \text{мс}}{\text{Frame Time (мс)}} = \frac{1000\ \text{мс}}{8{,}54\ \text{мс}} \approx 117\ \text{FPS}, \qquad (5.1)$$

де 1000 мс — це кількість мілісекунд в одній секунді;

Frame Time (мс) — час, потрібний для обробки одного кадру, у мілісекундах.

Під час тестування перформансу гри у сцені меню, було помічено стрибки та нестабільний frame time (див. рисунок 5.14), який більш помітний під час опускання камери до самого низу споруди, а на діаграмі, яка зображена на рисунку 5.15, можна побачити, що середній час кадру на цій сцені взагалі дорівнює аж 8,99 мілісекунди, та якщо порахувати по формулі вище (див. формулу 5.1), це ≈ 111 FPS, це вже незадовільний результат.

За допомогою увімкненої функції «Deep Profiling», яка дозволяє бачити деталі викликів методів у коді, було виявлено, що це просідання викликано промальовуванням мешей та текстур кубів у споруді. А причиною того, чому це маленьке просідання більш помітне коли камера опускається до самого низу споруди є те, що момент коли камера опущена до самого низу, вона бачить найбільшу кількість кубів споруди й графічному рушію потрібно промальовувати текстуру кожного цього куба.

Після того, коли було точно з'ясовано, що проблема в промальовуванні, було використано ще один більш спеціалізований вбудований інструмент Unity під назвою Frame Debugger, для перегляду окремих викликів промальовування (draw calls), які було викликано для візуалізації цих кадрів.

Frame Debugger в Unity — це інструмент, який допомагає аналізувати процес

рендерингу кадру у грі. Він дозволяє детально переглядати кожен етап обробки графіки, від першого виклику рендеру до фінального вигляду зображення на екрані. З його допомогою можна зрозуміти, які об'єкти й шейдери були використані, як накладаються текстури, як обробляються світло і тіні, а також як працює постобробка. Це корисно для виявлення зайвих викликів рендеру (overdraw), оптимізації продуктивності та розуміння того, як конкретні елементи сцени впливають на загальне навантаження графічного процесора.

На рисунку 5.16 зображено використання інструменту Frame Debugger у сцені Меню на одному з кадрів. Ми можемо побачити, що кожен куб споруди промальовується окремо, та що для провалювання всіх мешей кубів цієї споруди, було викликано аж 44 виклики промальовування (draw calls).

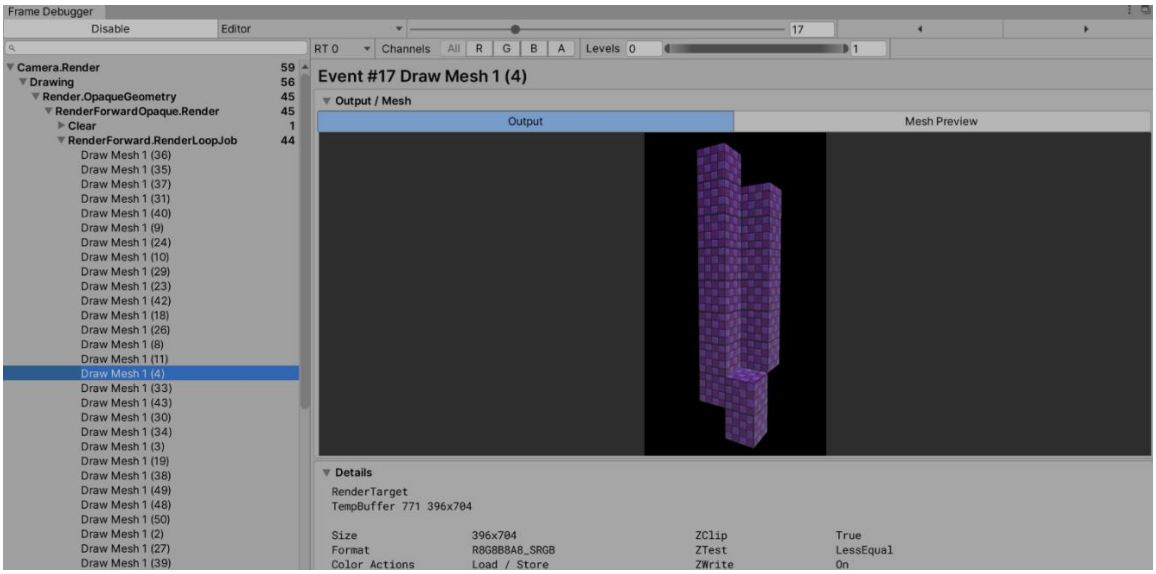

Рисунок 5.16 — Результати використання інструменту Frame Debugger

Draw call (виклик промальовування) — це команда, яку центральний процесор (CPU) надсилає графічному процесору (GPU), щоб той намалював певний об'єкт, текстуру чи групу об'єктів у сцені. Наприклад, кожен окремий об'єкт у грі, може потребувати окремого draw call, особливо якщо вони мають різні матеріали, шейдери або текстури.

Кількість draw call безпосередньо впливає на продуктивність гри, оскільки кожен виклик вимагає координації між CPU і GPU. CPU витрачає час на підготовку

цих команд, а GPU — на їх виконання. Якщо draw calls занадто багато, це може створити так званий bottleneck «вузьке місце» на рівні CPU, яке обмежує швидкість оновлення кадрів (FPS), навіть якщо сам GPU здатен обробляти графіку швидше.

Таким самим чином, за допомогою інструмента Frame Debugger було проаналізовано кількість викликів промальовування (draw calls) для одного кадру у кожній сцені. На рисунку 5.17 зображено результат цього аналізу у вигляді стовпчастої діаграми.

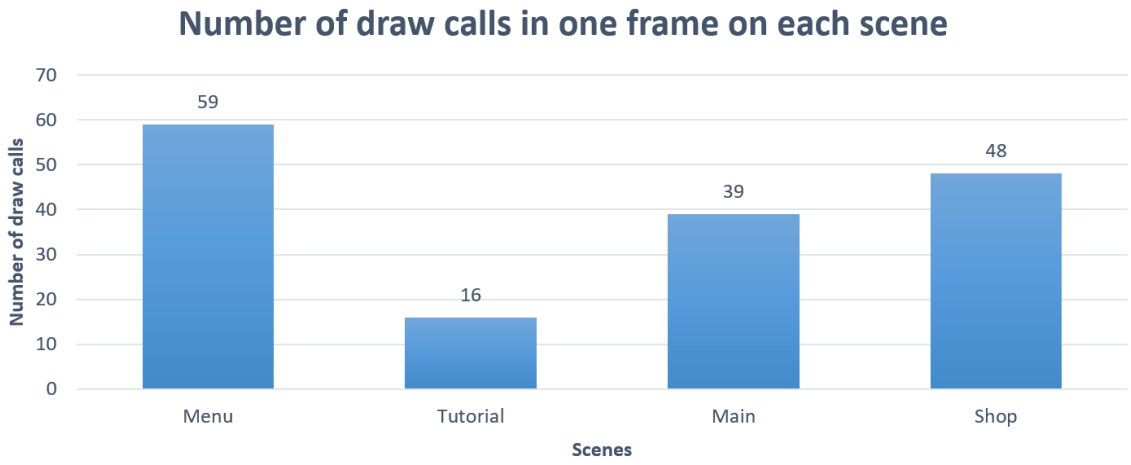

Рисунок 5.17 — Кількість викликів промальовування для одного кадру у кожній сцені гри

Як видно по діаграмі зображеній на рисунку 5.17, сцена Меню «лідирує» за кількістю викликів промальовування (чим більше — тим гірше), основну причину якої ми вже з'ясували раніше. Далі, у розділі оптимізації, буде проводитися оптимізація цього моменту.

5.2 Оптимізація гри

Оптимізація допомагає забезпечити стабільну та плавну роботу гри, мінімізуючи збої та технічні проблеми, які можуть негативно вплинути на досвід

користувача. Оптимізації забезпечує безперервний і приємний ігровий процес, та усуває ймовірність втрати інтересу користувача через технічні проблеми.

Під час тестування перформансу гри в минулому розділі, було виявлено стрибки та нестабільний frame time у сцені меню, особливо коли камера опускається до самого низу і за допомогою інструменту Frame Debugger було з'ясовано, що це викликано тим, що ігровому двигуну треба промальовувати кожний меш куба цієї споруди окремо.

Для того, щоб оптимізувати цей момент, потрібно об'єднати всі меші кубів цієї споруди в одну велику меш, для того, щоб зі свого боку, зменшити кількість викликів промальовування. Для цього був використаний безкоштовний ассет «MeshCombiner» в Unity Asset Store [56]. За допомогою якого, в один клік, всі меші кубів цієї споруди були об'єднанні в одну цільну меш. Після цього, був проведений повторний аналіз викликів промальовування за допомогою Frame Debugger. На рисунку 5.18 зображено результати перевірки викликів промальовування після оптимізації за допомогою об'єднання мешів у одну цільну меш. За результатами у Frame Debugger, видно, що тепер для промальовування споруди, графічному процесу потрібен лише один виклик промальовування.

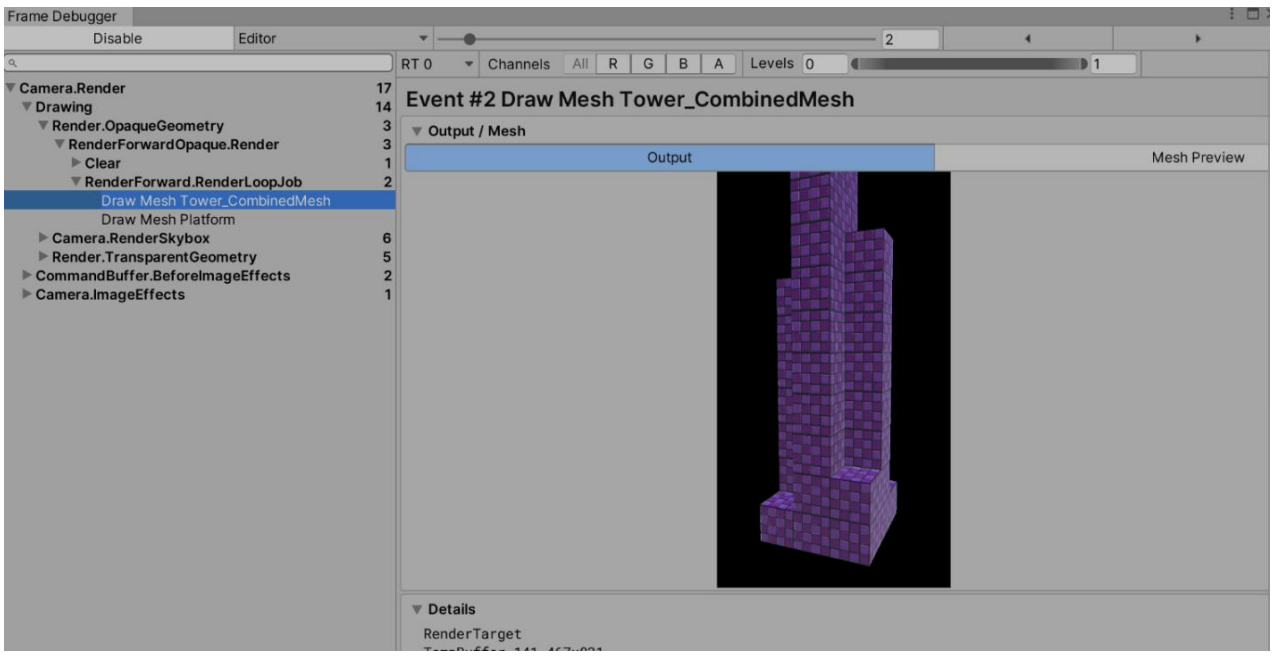

Рисунок 5.18 — Результати повторного використання інструменту Frame Debugger після оптимізації

Ще одним етапом оптимізації, було позбавлення від Normal Map текстур або зменшення його властивості bumpiness (нерівність), які використовують матеріали скинів, для їх більш гарного вигляду. Normal Map в Unity — це текстура, яка зберігає інформацію про напрямки нормалей поверхні моделі. Вона використовується для імітації дрібних деталей (наприклад, виступів або вм'ятин) без додавання геометрії. Завдяки цьому освітлення взаємодіє з поверхнею, створюючи ефект рельєфу, що робить модель візуально складнішою та реалістичнішою, зберігаючи продуктивність.

Попри те, що Normal Map наче й призначений для оптимізації перформансу гри, тому що вона не змінює геометрію моделі (тобто кількість полігонів залишається незмінною), використання текстур Normap Map на більш старих моделях смартфонів, з більш старими графічними процесорами, може привести до падіння загального перформансу гри. Це пов'язано з тим, що використання Normal Maps викликає підвищене навантаження на графічний процесор на цих пристроях, адже він обчислює освітлення з урахуванням нормалей текстури для кожного пікселя. Це також вимагає більше пам’яті для зберігання текстур, що може стати проблемою для пристроїв з обмеженими ресурсами. Шейдери з Normal Maps є складнішими, що збільшує час рендерингу (що у свою чергу збільшує frame time), а додаткове навантаження на GPU підвищує енергоспоживання, спричиняючи швидший розряд батареї.

Тому, використання Normap Map для мобільних ігор не рекомендується. Це не означає, що їх не можна використовувати взагалі, але якщо мета — охопити якомога більшу аудиторію, яка містить користувачів, що мають більш слабкі моделі смартфонів, у такому разі краще цього уникати, або дуже сильно зменшувати його властивість bumpiness. Під час розробки гри, було створено текстури Normal Map для деяких скинів, один із прикладів створеної Normal Map зображено на рисунку 5.19.

А на рисунку 5.20 зображено різницю вигляду скіна, з використанням Normal Map та без, зліва показано вигляд скіна з використання Normal Map, а праворуч без використання Normal Map. Як видно на цьому рисунку (див. рисунок 5.20), Normal

Map додав цьому скіну виступи та впадини, та зробив його більше деталізованим.

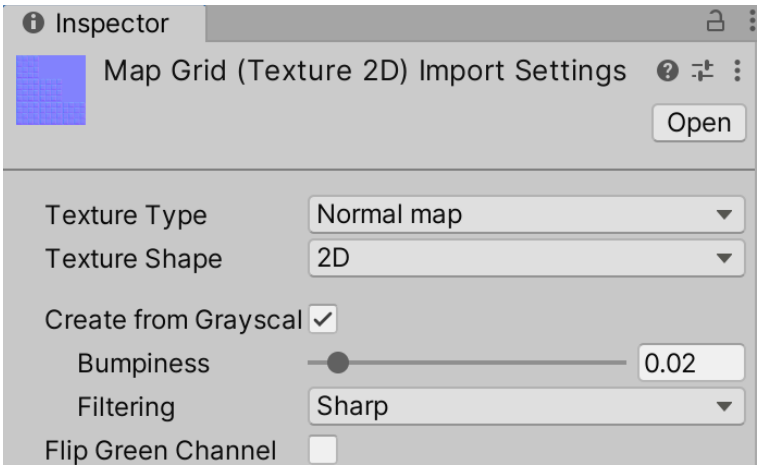

Рисунок 5.19 — Створена текстура Normal Map для скіна

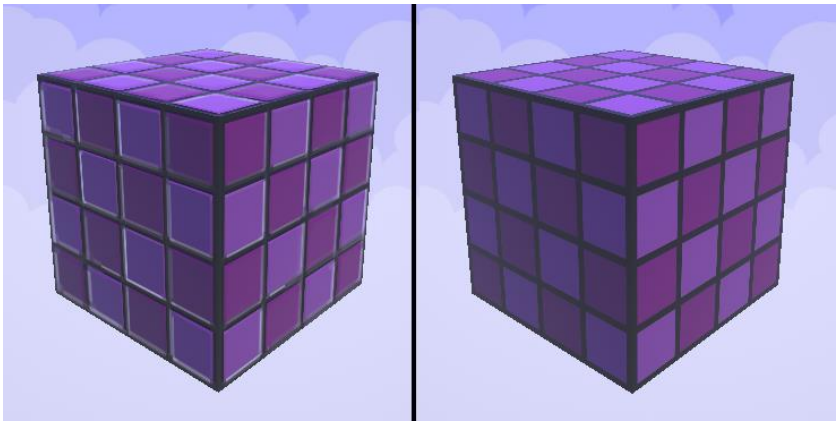

Рисунок 5.20 — Вигляд скіна з використанням Normal Map та без Normal Map

Попри те, що вони роблять скін більш красивим і деталізованим, для досягнення максимальної плавності гри на слабших пристроях, у більшості скинів Normal maps було прибрано, а для деяких скинів, значення властивості Normal maps Bumpiness було сильно зменшено.

Під час тестування перформансу гри в минулому розділі, ми також провели аналіз кількості викликів промальовування для кожної сцени за допомогою інструменту Frame Debugger та побудували діаграму (див. рисунок 5.17). З діаграми було зрозуміло, що у сцені Меню найбільша кількість викликів промальовування (draw calls), а саме аж 59 draw calls в одному кадрі. Основну причину такої високої кількості викликів промальовування у цій сцені ми вже виправили, за допомогою

об'єднання мешей споруди, що в свою чергу посприяло зменшенню викликів промальовування (draw calls) аж на 42 виклики. Для інших сцен, ніякої оптимізації для зменшення draw calls не було, а як ми зрозуміли у минулому розділі, кількість draw call безпосередньо впливає на продуктивність гри, оскільки кожен виклик вимагає координації між CPU і GPU, що на менш слабких пристроях, може призвести до маленького падіння загального перформансу гри. Тому для гри було проведено ще одну більш загальну оптимізацію.

Один із найрозповсюдженіших методів зменшення числа викликів промальовування (draw calls) — це використання Атласів спрайтів (Sprite atlases). Атлас спрайтів (Sprite Atlas) — це зображення, яке містить декілька графічних елементів (спрайтів) в одній текстурі. Використання атласів є поширеною практикою в розробці ігор для оптимізації. Ідея атласу спрайтів полягає в тому, щоб замість того, щоб завантажувати й обробляти кожен елемент інтерфейсу (спрайт) окремо, використовувати одне велике зображення, що містить всі необхідні елементи інтерфейсу (спрайти) гри на сцені. Кожен спрайт в атласі розташовується у певній області цього зображення, а під час рендерингу гри визначається, яку частину атласу потрібно відобразити в цей момент. Оскільки всі елементи містяться в одному файлі, це значно зменшує кількість викликів промальовування (draw calls) та покращує перформанс, це особливо корисно на мобільних пристроях з обмеженими ресурсами. Цей метод також дозволяє значно зменшити час завантаження гри, оскільки завантажується один великий файл замість багатьох дрібних.

Для створення спрайт атласів у цьому проєкті був використаний вбудований інструмент Unity для роботи з атласами спрайтів. Це один із найзручніших способів, оскільки Unity автоматично оптимізує роботу атласів. Все що потрібно зробити для створення спрайт атласу в Unity, це клацнути у створеній папці для зберігання атласів та вибрати Create → 2D → Sprite Atlas, а потім перетягнути всі необхідні спрайти у поле Objects for Packing. У цьому проєкті було створено декілька спрайт атласів для кожної сцени. На рисунку 5.21 зображено один із прикладів створеного спрайт атласу для сцени меню.

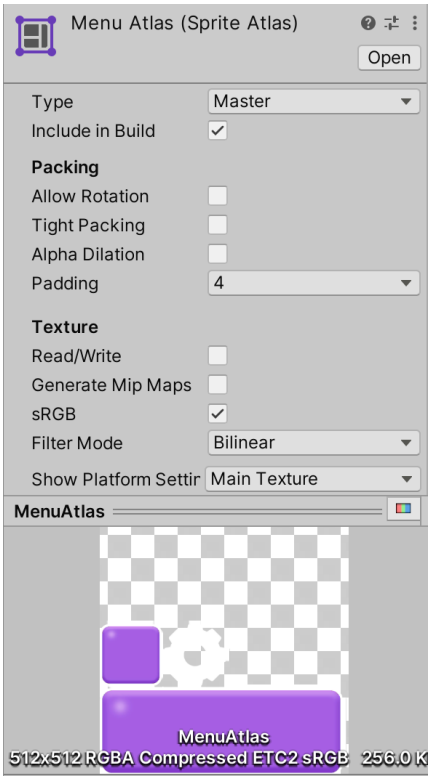

Рисунок 5.21 — Створений спрайт атлас для основного інтерфейсу для сцени меню

На рисунках 5.22 та 5.23 показано різницю у кількості викликів промальовування (draw calls) для основного інтерфейсу у сцені меню без створеного спрайт атласу та зі спрайт атласом. Для цього знову був використаний дуже корисний вбудований інструмент Unity Frame Debugger.

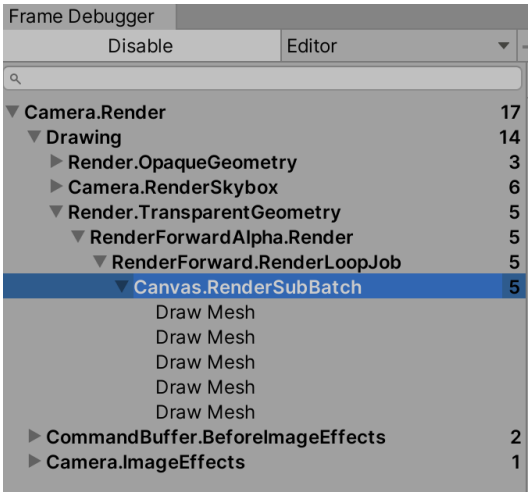

Рисунок 5.22 — Кількість викликів промальовування інтерфейсу у сцені меню без Спрайт атласу

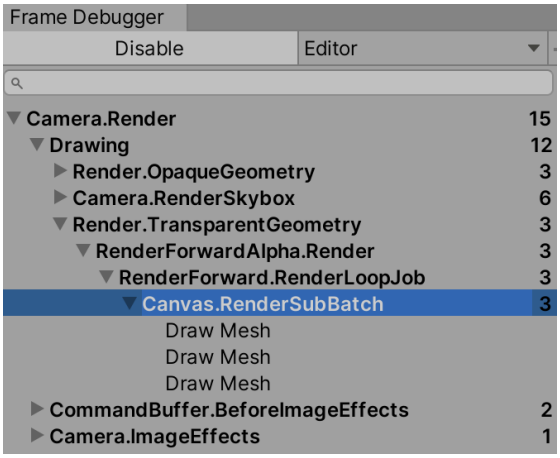

Рисунок 5.23 — Кількість викликів промальовування інтерфейсу у сцені меню з використанням Спрайт атласу

Як ми бачимо з результатів тестування у Frame Debugger, без використання спрайт атласу, тільки для основного інтерфейсу у сцені меню відбувається 5 викликів промальовування, а з використанням спрайт атласу лише 3. У цьому одному прикладі ця різниця невелика, але потрібно враховувати, що у проєкті 4 різні сцени й для кожної було створено кілька спрайт атласів для використовуваних спрайтів у них і в такому разі різниця вже значна. А саме, на рисунку 5.24 зображено різницю у кількості викликів промальовування (draw calls) у кожній сцені, до оптимізації та після оптимізації у вигляді стовпчастої діаграми.

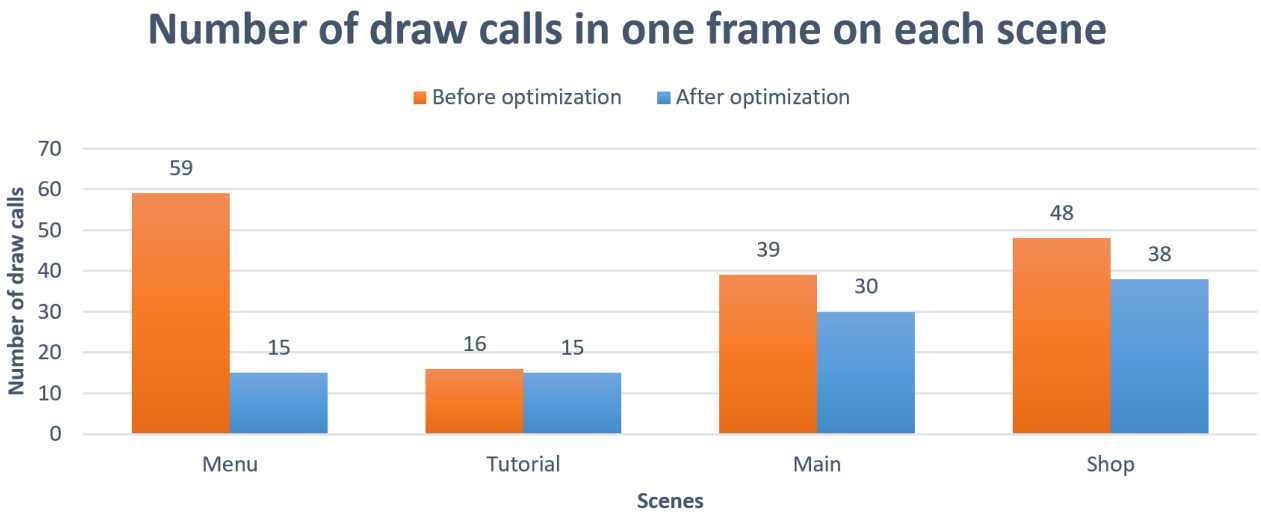

Рисунок 5.24 — Різниця кількості викликів промальовування у кожній сцені, до оптимізації та після оптимізації

Після всіх проведених етапів оптимізації, гру було протестовано знову і на рисунках 5.25, 5.26, 5.27 та 5.28, наведено результати повторного тестування кожної сцени окремо, а саме зображено графіки часу кадрів на основні зібраних даних для 300 кадрів (frames) на кожній сцені після оптимізації.

Як видно на графіку часу кадру, який зображений на рисунку 5.25, після всіх проведених етапів оптимізації, у сцені меню, на якій до цього були проблеми з нестабільним frame time, тепер все стабільно.

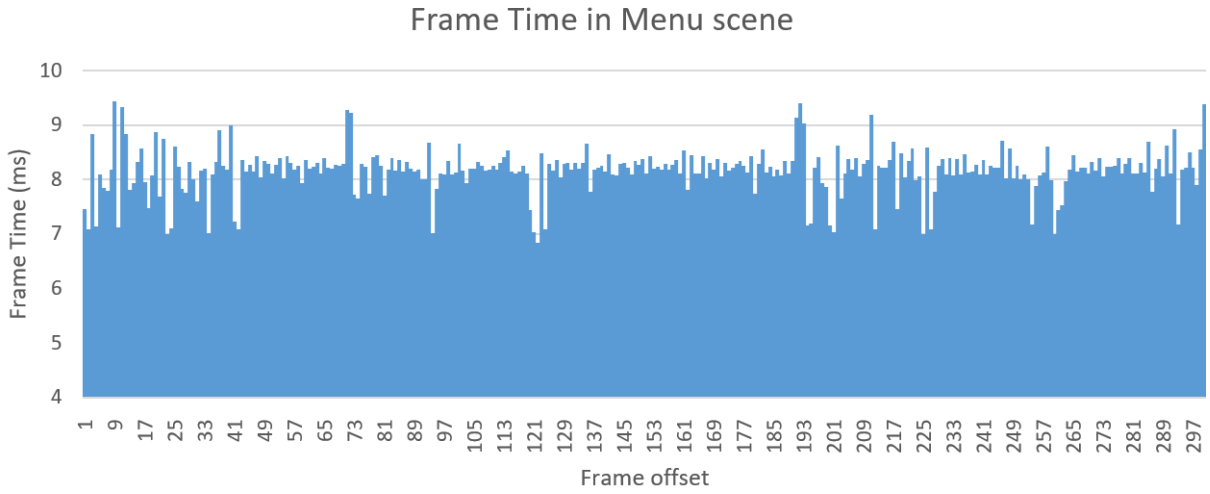

Рисунок 5.25 — Графік часу кадру для 300 кадрів на сцені меню після оптимізації

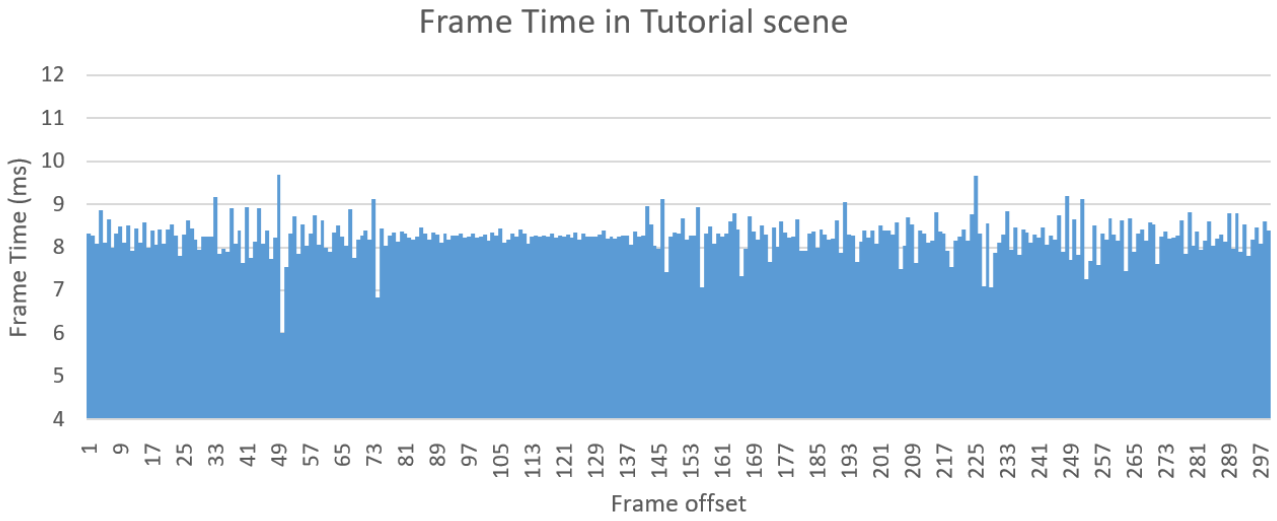

Рисунок 5.26 — Графік часу кадру для 300 кадрів на навчальній сцені після оптимізації

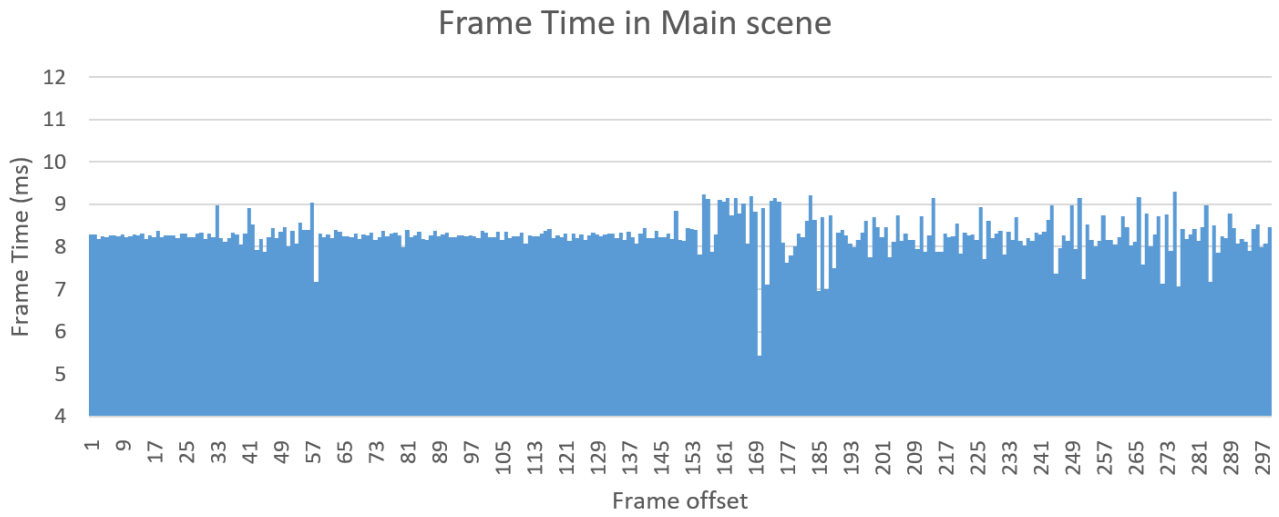

Рисунок 5.27 — Графік часу кадру для 300 кадрів на сцені ігрового процесу після оптимізації

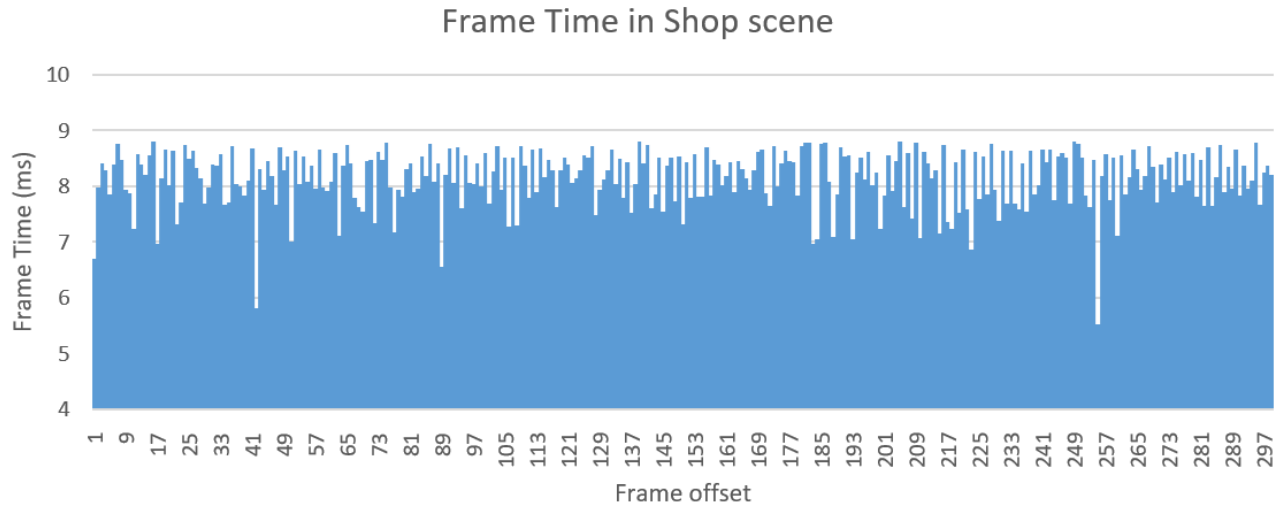

Рисунок 5.28 — Графік часу кадру для 300 кадрів на сцені ігрового магазину після оптимізації

А на рисунку 5.29 зображено різницю середнього часу кадру (avg frame time) на основі аналізу даних для 300 кадрів у кожній сцені, до всіх проведених етапів оптимізації та після оптимізації у вигляді стовпчастої діаграми. Як видно з цього графіку (див. рисунок 5.29), на сцені меню спостерігається просто колосальне зменшення середнього часу кадру, аж на 0,61 мс. На інших сценах, після оптимізації, frame time теж зменшився, що свідчить про успішну оптимізацію гри.

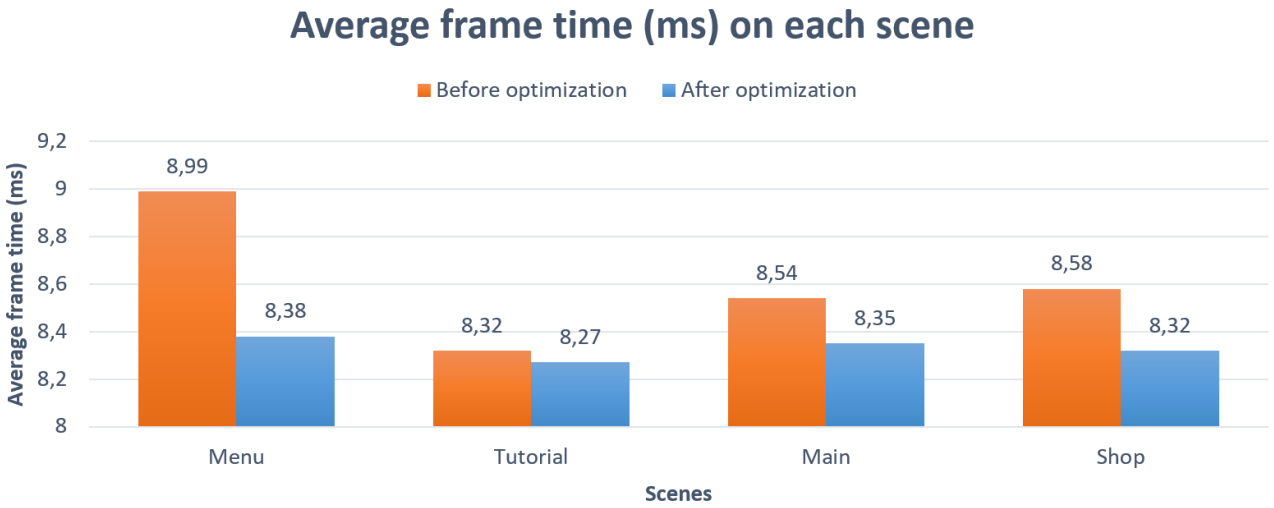

Рисунок 5.29 — Різниця середнього часу кадру для 300 кадрів у кожній сцені, до всіх проведених етапів оптимізації та після оптимізації

Загалом, після повторного тестування гри після всіх виконаних етапів оптимізації та аналізу їхніх результатів, можна сказати, що оптимізація була більш ніж успішна, та дозволила значно покращити її перформанс. Завдяки чому, гравці зможуть насолоджуватися більш плавним і захоплюючим геймплеєм, що значно підвищує їхній рівень задоволення від гри.

ВИСНОВКИ

Таким чином, у дипломній роботі магістра була розроблена аркадна гра для ОС Android з інтегрованою системою монетизації. Впроваджені механізми монетизації, забезпечують баланс між ефективною монетизацією і позитивним користувацьким досвідом, що є особливо важливим для зростання аудиторії та утримання гравців. Гра протестована та оптимізована для досягнення високого рівня продуктивності та задоволення гравців. Результати підтверджують доцільність обраного підходу до створення мобільних ігор для Android та ефективність використаних стратегій монетизації [57].

У процесі дослідження було проведено детальний аналіз інструментів та технологій розробки мобільних ігор, виділення їх переваг та недоліків, що дозволило обрати оптимальний ігровий двигун, а також мову програмування для розробки проєкту.

Для забезпечення стабільного та ефективного прибутку від гри, був проведений глибокий аналіз наявних стратегій монетизації мобільних ігор, детально проаналізовано рекламні мережі, типи внутрішньоігрової реклами, визначено їх переваги та недоліки, що дозволило визначити найефективніші підходи, зокрема використання внутрішньоігрової стратегії монетизації, Google AdMob як рекламної мережі, а також два основних формати реклами, таких як: проміжна реклама та реклама з винагородою. Такий підхід дозволив досягти балансу між прибутковістю гри та позитивним користувацьким досвідом. Також була розроблена концепція гри, яка включала визначення її жанру, основної ідеї та цільової аудиторії. Відштовхуючись від концепції, було здійснено проєктування гри, включаючи проєктування інтерфейсу користувача та основного функціоналу кожної зі сцен. На основі розробленої концепції та розробленого проєкту гри, було проведено реалізацію гри, яка включала розробку візуальних та звукових ефектів, розробку інтерфейсу гри, розробку скинів для внутрішньоігрового магазину, налаштування ігрових сцен, написання програмного коду та інтеграцію

внутрішньоігрової реклами. Далі було проведено тестування ігрового процесу, функціоналу гри, інтегрованої внутрішньоігрової реклами та перформансу гри. Завершальним етапом була оптимізація продуктивності гри на основі результатів тестування, для забезпечення стабільної та плавної роботи на різних пристроях, що у свою чергу покращує користувацький досвід та гарантує безперервний ігровий процес.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Що таке ігровий двигун. Короткий огляд для початківців. URL: https://drawandcode.com/learning-zone/what-is-a-game-engine/ (дата звернення: 15.10.2024).

2) Офіційний веб-сайт Defold. URL: https://defold.com/ (дата звернення: 16.10.2024).

3) Логотип Defold. URL: https://defold.com/logo-and-trademark/ (дата звернення: 16.10.2024).

4) Відгуки про ігровий двигун Defold на веб-сайті Reddit. URL: https://www.reddit.com/r/gamedev/comments/x4zvv5/whats_your_opinion_about_defold/ (дата звернення: 16.10.2024).

5) Обговорення відмінностей Defold і Godot Engine на офіційному форумі Defold. URL: https://forum.defold.com/t/is-defold-still-a-good-choice-compared-with-godot-4/69941 (дата звернення: 16.10.2024).

6) Логотип Cocos2D. URL: https://archive.org/details/cocos2d-2014_2020 (дата звернення: 16.10.2024).

7) Огляд ігрового двигуна Cocos2D на веб-сайті eternitech. URL: https://eternitech.com/technologies/cocos2d/ (дата звернення: 16.10.2024).

8) Логотип CryEngine. URL: https://www.cryengine.com/brand (дата звернення: 17.10.2024).

9) Огляд ігрового двигуна CryEngine на офіційному сайті Crytek. URL: https://www.crytek.com/cryengine (дата звернення: 17.10.2024).

10) Відгуки про ігровий двигун CryEngine на веб-сайті Capterra. URL: https://www.capterra.com/p/210664/CRYENGINE/reviews/ (дата звернення: 17.10.2024).

11) Офіційний веб-сайт Construct 3. URL: https://www.construct.net/en (дата звернення: 17.10.2024).

12) Логотип Construct 3. URL: https://www.pngaaa.com/detail/4891115 (дата

звернення: 17.10.2024).

13) Відгуки та огляд ігрового двигуна Construct 3 на веб-сайті Capterra. URL: https://www.capterra.com/p/201543/Construct-3/ (дата звернення: 17.10.2024).

14) Відгуки про ігровий двигун Construct 3 на веб-сайті Reddit. URL: https://www.reddit.com/r/gamedev/comments/niufij/how_good_is_construct_3_for_game_development/ (дата звернення: 17.10.2024).

15) Офіційний веб-сайт Unreal Engine. URL: https://www.unrealengine.com/en-US (дата звернення: 17.10.2024).

16) Огляд Unreal Engine на веб-сайті Bairesdev. URL: https://www.bairesdev.com/blog/what-is-unreal-engine/ (дата звернення: 17.10.2024).

17) Логотип Unreal Engine. URL: https://www.unrealengine.com/en-US/branding (дата звернення: 17.10.2024).

18) Відгуки про Unreal Engine на веб-сайті Capterra. URL: https://www.capterra.com/p/158599/Unreal-Engine/reviews/ (дата звернення: 17.10.2024).

19) Огляд Unreal Engine та його плюсів і мінусів на веб-сайті Coursera. URL: https://www.coursera.org/articles/what-is-unreal-engine (дата звернення: 17.10.2024).

20) Детальний огляд ігрового двигуна Godot Engine. URL: https://dev.to/opensauced/game-development-and-multimedia-godot-engine-a-game-changer-in-game-development-20l7 (дата звернення: 17.10.2024).

21) Логотип Godot Engine. URL: https://godotengine.org/press/ (дата звернення: 17.10.2024).

22) Відгуки про Godot Engine на веб-сайті Capterra. URL: https://www.capterra.com/p/234843/Godot-Engine/ (дата звернення: 18.10.2024).

23) Офіційний веб-сайт Unity. URL: https://unity.com/ (дата звернення: 19.10.2024).

24) Повний огляд ігрового двигуна Unity на веб-сайті gamedevacademy. URL: https://gamedevacademy.org/what-is-unity/ (дата звернення: 19.10.2024).

25) Логотип Unity. URL: https://unity.com/legal/branding-trademarks (дата звернення: 19.10.2024).

26) Відгуки про Unity на веб-сайті Capterra. URL: https://www.capterra.com/p/158591/Unity/ (дата звернення: 19.10.2024).

27) Офіційний веб-сайт Unity Discussions. URL: https://discussions.unity.com/ (дата звернення: 19.10.2024).

28) Офіційний посібник Unity. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 19.10.2024).

29) Стаття про стратегії монетизації у мобільних іграх на веб-сайті appsflyer. URL: https://www.appsflyer.com/blog/tips-strategy/mobile-game-monetization/ (дата звернення: 20.10.2024).

30) Знайомство з мережами реклам та огляд найпопулярніших. URL: https://www.gamebizconsulting.com/blog/best-ad-networks-mobile-game-app (дата звернення: 21.10.2024).

31) Офіційний веб-сайт AppLovin. URL: https://www.applovin.com/ (дата звернення: 21.10.2024).

32) Логотип AppLovin. URL: https://www.vhv.rs/viewpic/TRiixbT_applovin-logo-png-transparent-png/ (дата звернення: 21.10.2024).

33) Огляд AppLovin на веб-сайті thestrategystory. URL: https://thestrategystory.com/2023/01/18/how-does-applovin-work-and-make-money-business-model/ (дата звернення: 21.10.2024).

34) Огляд AppLovin на веб-сайті mavendigital. URL: https://www.mavendigital.ae/blog/applovin-review/ (дата звернення: 21.10.2024).

35) Частка ринку AppLovin. URL: https://6sense.com/tech/mobile-ad-network/applovin-market-share (дата звернення: 21.10.2024).

36) Офіційний веб-сайт InMobi. URL: https://www.inmobi.com/ (дата звернення: 22.10.2024).

37) Логотип InMobi. URL: https://logovtor.com/inmobi-logo-vector-svg/ (дата звернення: 22.10.2024).

38) Огляд та відгуки про InMobi на веб-сайті trustradius. URL: https://www.trustradius.com/products/miip/reviews?qs=pros-and-cons#overview (дата звернення: 22.10.2024).

39) Офіційний сайт UnityAds. URL: https://unity.com/products/unity-ads (дата звернення: 23.10.2024).

40) Логотип UnityAds. URL: https://logowik.com/unity-ads-logo-vector-svg-pdf-ai-eps-cdr-free-download-15998.html (дата звернення: 23.10.2024).

41) Огляд UnityAds та його переваг та недоліків на веб-сайті mavendigital. URL: https://www.mavendigital.ae/blog/unity-ads-review/ (дата звернення: 23.10.2024).

42) Офіційний сайт IronSource. URL: https://www.is.com/ (дата звернення: 24.10.2024).

43) Логотип IronSource. URL: https://companieslogo.com/ironsource/logo/ (дата звернення: 23.10.2024).

44) Огляд та відгуки про IronSource на веб-сайті trustradius. URL: https://www.trustradius.com/products/ironsource/reviews#overview (дата звернення: 24.10.2024).

45) Офіційний сайт Google AdMob. URL: https://admob.google.com/home/resources/what-is-admob/ (дата звернення: 24.10.2024).

46) Логотип Google AdMob. URL: https://worldvectorlogo.com/logo/google-admob (дата звернення: 24.10.2024).

47) Огляд Google AdMob та його переваг та недоліків на веб-сайті geeksforgeeks. URL: https://www.geeksforgeeks.org/overview-of-google-admob/ (дата звернення: 24.10.2024).

48) Що таке медіація. URL: https://www.appsflyer.com/glossary/ad-mediation/ (дата звернення: 24.10.2024).

49) Огляд що таке медіація та її особливості. URL: https://xenoss.io/blog/mobile-ad-mediation-for-game-monetization (дата звернення: 24.10.2024).

50) Офіційний сайт Google Support з інформацією про формати реклами Google AdMob та їх особливості. URL: https://support.google.com/admob/answer/6128738 (дата звернення: 25.10.2024).

51) Сайт Coolors для підбору кольорової палітри. URL: https://coolors.co/8c5af4-9b70ff-d4dcff-a396f9-b97cff (дата звернення: 15.11.2024).

52) Сайт Freepik зі стоковими іконками. URL: https://www.freepik.com/vectors/icon (дата звернення: 15.11.2024).

53) Бібліотека анімацій на сайті Unity AssetStore. URL: https://assetstore.unity.com/packages/tools/animation/primetween-high-performance-animations-and-sequences-252960 (дата звернення: 15.11.2024).

54) Офіційний плагін Google AdMob для інтеграції у Unity. URL: https://github.com/googleads/googleads-mobile-unity (дата звернення: 25.11.2024).

55) Офіційна документація Google присвячена інтеграції Google AdMob у Unity. URL: https://developers.google.com/admob/unity/quick-start#import-from-github (дата звернення: 25.11.2024).

56) Безкоштовний асет Mesh Combiner для комбінування мешей на офійіному сайті Unity Asset Store. URL: https://assetstore.unity.com/packages/tools/modeling/mesh-combiner-157192 (дата звернення: 27.11.2024).

57) Сушко О.С., Тягунова М.Ю. Аркадна гра для ОС Android з монетизацією. Експериментальні та теоретичні дослідження в контексті сучасної науки: VII всеукраїнська студентська наукова конф., м. Львів, 22 лист. 2024 р. Львів, 2024. С. 422-424.

ДОДАТОК А

Лістинг А.1 — CameraInputController.cs

```
using UnityEngine;
using UnityEngine.EventSystems;

public class CameraInputController : MonoBehaviour
{
    private Vector3 _firstPoint;
    private Vector3 _secondPoint;
    private float _xAngle = 0.0f;
    private float _yAngle = 0.0f;
    private float _xAngleTemp = 0.0f;
    private float _yAngleTemp = 0.0f;

    [SerializeField] private float _maxRotationAngle = 120.0f;
    [SerializeField] private float _zoomSpeed = 0.008f;
    [SerializeField] private float _minZoom = -25f;
    [SerializeField] private float _maxZoom = -5f;
    [SerializeField] private float _minVerticalAngle = -30f;
    [SerializeField] private float _maxVerticalAngle = 30f;

    private bool _isZooming = false;
    private float _singleTouchDelay = 0.1f;
    private float _lastTouchTime = 0.0f;
    private Transform _camera;

    private void Start()
    {
        _xAngle = 0.0f;
        _yAngle = 0.0f;
        transform.rotation = Quaternion.Euler(_yAngle, _xAngle,
0.0f);
        _camera = Camera.main.transform;
    }

    private void Update()
    {
        if (Input.touchCount == 1 &&
!IsPointerOverGameObjectTouch() && !_isZooming)
        {
            Touch touch = Input.GetTouch(0);

            if (Time.time - _lastTouchTime > _singleTouchDelay)
            {
                if (touch.phase == TouchPhase.Began)
                {
                    _firstPoint = touch.position;
                    _xAngleTemp = _xAngle;
```

```
                        _yAngleTemp = _yAngle;
                    }

                    if (touch.phase == TouchPhase.Moved)
                    {
                        _secondPoint = touch.position;
                        _xAngle = _xAngleTemp + (_secondPoint.x -
_firstPoint.x) * _maxRotationAngle / Screen.width;
                        _yAngle = _yAngleTemp - (_secondPoint.y -
_firstPoint.y) * _maxRotationAngle / Screen.height;

                        _yAngle       =       Mathf.Clamp(_yAngle,
_minVerticalAngle, _maxVerticalAngle);

                        transform.rotation                          =
Quaternion.Euler(_yAngle, _xAngle, 0.0f);
                    }
                }
            }
            else if (Input.touchCount == 2)
            {
                _isZooming = true;

                Touch touchZero = Input.GetTouch(0);
                Touch touchOne = Input.GetTouch(1);

                Vector2  touchZeroPrevPos  =  touchZero.position  -
touchZero.deltaPosition;
                Vector2   touchOnePrevPos   =   touchOne.position   -
touchOne.deltaPosition;

                float   prevTouchDeltaMag   =   (touchZeroPrevPos   -
touchOnePrevPos).magnitude;
                float   touchDeltaMag   =   (touchZero.position   -
touchOne.position).magnitude;

                float   deltaMagnitudeDiff   =   touchDeltaMag   -
prevTouchDeltaMag;

                Vector3 cameraPos = _camera.localPosition;
                cameraPos.z += deltaMagnitudeDiff * _zoomSpeed;
                cameraPos.z  =  Mathf.Clamp(cameraPos.z,  _minZoom,
_maxZoom);
                _camera.localPosition = cameraPos;
            }
            else if (Input.touchCount == 0 && _isZooming)
            {
                _isZooming = false;
                _lastTouchTime = Time.time;
            }
        }

        private bool IsPointerOverGameObjectTouch()
```

```
        {
            return
EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).finger
Id);
        }
    }
```

Лістинг А.2 — TapTracker.cs

```
    using UnityEngine;
    using UnityEngine.EventSystems;

    public class TapTracker : MonoBehaviour
    {
        [SerializeField]        private        MainSceneController
_mainSceneController;

        public float TapThreshold = 0.15f;
        public float TapCooldown = 0.2f;

        private float tapStartTime;
        private bool isTap;
        private int tapCount;
        private float lastTapTime;
        private bool isCooldownActive;
        private bool isActionPerformed;

        private void PerformTapAction()
        {
            _mainSceneController.PlaceCube();
        }

        void Update()
        {
            if        (Input.touchCount        >        0        &&
!IsPointerOverGameObjectTouch())
            {
                Touch touch = Input.GetTouch(0);

                switch (touch.phase)
                {
                    case TouchPhase.Began:
                        if (!isCooldownActive)
                        {
                            tapStartTime = Time.time;
                            isTap = true;
                            tapCount = 1;
                        }
                        break;

                    case TouchPhase.Moved:
                    case TouchPhase.Stationary:
```

```
                        if (isTap && Time.time - tapStartTime >
TapThreshold)
                        {
                            isTap = false;
                        }
                        break;

                    case TouchPhase.Ended:
                        if (isTap && tapCount == 1 &&
!isCooldownActive)
                        {
                            if (Time.time - tapStartTime <=
TapThreshold)
                            {
                                if (!isActionPerformed)
                                {
                                    PerformTapAction();
                                    isActionPerformed = true;
                                    isCooldownActive = true;
                                    lastTapTime = Time.time;
                                }
                            }
                        }
                        isTap = false;
                        break;
                }

                for (int i = 1; i < Input.touchCount; i++)
                {
                    if (Input.GetTouch(i).phase == TouchPhase.Began)
                    {
                        tapCount++;
                    }
                }
            }

            if (isCooldownActive && Input.touchCount == 0 &&
Time.time - lastTapTime >= TapCooldown)
            {
                isCooldownActive = false;
                isActionPerformed = false;
            }
            else if (isCooldownActive && Input.touchCount > 0)
            {
                lastTapTime = Time.time;
            }
        }

        private bool IsPointerOverGameObjectTouch()
        {
            if
(EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).finge
rId))
```

```
            return true;
        else
            return false;
    }
}
```

ДОДАТОК Б

Лістинг Б.1 — CurrencySpawner.cs

```
using System.Collections;
using UnityEngine;

public class CurrencySpawner : MonoBehaviour
{
    [SerializeField] private UIManager _UIManager;
    [SerializeField] private MainGameController
_mainSceneController;
    [SerializeField] private GameObject[] _currency;
    [SerializeField] private Material[]
_towerPatternCubeMaterial;
    [SerializeField] private Material
_towerPatternDefaultMaterial;
    [SerializeField] private float _megaGemSpawnChance = 0.10f;

    private WaitForSeconds
_lifeTimeExpiredDelayBeforeSpawningNewCurrencyWFS;
    private WaitForSeconds
_pickedUpDelayBeforeSpawningNewCurrencyWFS;
    [SerializeField] private float
_lifeTimeExpiredDelayBeforeSpawningNewCurrency = 2f;
    [SerializeField] private float
_pickedUpDelayBeforeSpawningNewCurrency = 7f;
    private GameObject _currentCurrency;
    private bool _isAllowedToSpawnNewCurrency = true;
    private float _randMegaGem;

    private Vector3 _currentCurrencyPosition;
    private GameObject _currentTowerPatternCubeWithCurrency;

    [HideInInspector] public int TotalNumberOfGemsCollected = 0;

    private Coroutine _spawnCurrencyCoroutine = null;
    private Coroutine _pickedUpDelayCoroutine = null;
    private Coroutine _lifeTimeExpiredDelay = null;

    private void Start()
    {
        _lifeTimeExpiredDelayBeforeSpawningNewCurrencyWFS = new
WaitForSeconds(_lifeTimeExpiredDelayBeforeSpawningNewCurrency);
        _pickedUpDelayBeforeSpawningNewCurrencyWFS = new
WaitForSeconds(_pickedUpDelayBeforeSpawningNewCurrency);
    }

    public void ActivateGemsGeneration()
    {
```

```
            if (_spawnCurrencyCoroutine != null)
            {
                StopCoroutine(_spawnCurrencyCoroutine);
                _spawnCurrencyCoroutine = null;
            }

            if (_pickedUpDelayCoroutine != null)
            {
                StopCoroutine(_pickedUpDelayCoroutine);
                _pickedUpDelayCoroutine = null;
            }

            if (_lifeTimeExpiredDelay != null)
            {
                StopCoroutine(_lifeTimeExpiredDelay);
                _lifeTimeExpiredDelay = null;
            }

            _isAllowedToSpawnNewCurrency = true;

            _spawnCurrencyCoroutine = StartCoroutine(SpawnGemsIE());
        }

        public void DeactivateGemsGeneration(bool isOnLoss = false)
        {
            StopCoroutine(_spawnCurrencyCoroutine);

            if (isOnLoss && _currentCurrency != null &&
_currentTowerPatternCubeWithCurrency != null)
            {

_currentCurrency.GetComponentInChildren<SimpleCollectibleScriptV1>()
.DestroyGem();

_currentTowerPatternCubeWithCurrency.GetComponent<MeshRenderer>().ma
terial = _towerPatternDefaultMaterial;
            }
        }

        private IEnumerator SpawnGemsIE()
        {
            while
(_mainSceneController._towerPatternAllCubePositions.Count >= 5)
            {
                if(_isAllowedToSpawnNewCurrency)
                {
                    Vector3           generatedPosition           =
RandPositionOfSpawnCurrency();
                    _currentCurrency                              =
Instantiate(CurrencyTypeToSpawn(),                generatedPosition,
Quaternion.identity);
```

```
                    SimpleCollectibleScriptV1
currentCurrencyCollectible                                          =
_currentCurrency.GetComponentInChildren<SimpleCollectibleScriptV1>()
;
                    currentCurrencyCollectible.OnCurrencyDestroyed
+= HandleCurrencyDestroyed;
                    _isAllowedToSpawnNewCurrency = false;
                }
                yield return null;
            }
        }

        private void HandleCurrencyDestroyed(bool wasPickedUp, int
numberOfGemsCollected)
        {
            if (wasPickedUp)
            {
                UpdateNumberOfGemsCollected(numberOfGemsCollected);
                _pickedUpDelayCoroutine                             =
StartCoroutine(StartPickedUpDelayAndSpawnNewCurrency());
            }
            else
            {
                if(_currentTowerPatternCubeWithCurrency != null)

_currentTowerPatternCubeWithCurrency.GetComponent<MeshRenderer>().ma
terial = _towerPatternDefaultMaterial;
                _lifeTimeExpiredDelay                               =
StartCoroutine(StartLifeTimeExpiredDelayAndSpawnNewCurrency());
            }
        }

        private IEnumerator StartPickedUpDelayAndSpawnNewCurrency()
        {
            yield return _pickedUpDelayBeforeSpawningNewCurrencyWFS;
            _isAllowedToSpawnNewCurrency = true;
        }

        private                                          IEnumerator
StartLifeTimeExpiredDelayAndSpawnNewCurrency()
        {
            yield                                                return
_lifeTimeExpiredDelayBeforeSpawningNewCurrencyWFS;
            _isAllowedToSpawnNewCurrency = true;
        }

        private GameObject CurrencyTypeToSpawn()
        {
            if (IsNeededToSpawnSuperGem())
            {

ChangeTowerPatternCubeMaterial(_towerPatternCubeMaterial[3]);
                return _currency[3];
```

```
            }

            if   (_mainSceneController.CurrentLevelData.currentLevel
<= 5)
            {

ChangeTowerPatternCubeMaterial(_towerPatternCubeMaterial[0]);
                return _currency[0];
            }
            if   (_mainSceneController.CurrentLevelData.currentLevel
<= 11)
            {

ChangeTowerPatternCubeMaterial(_towerPatternCubeMaterial[1]);
                return _currency[1];
            }


ChangeTowerPatternCubeMaterial(_towerPatternCubeMaterial[2]);
            return _currency[2];
        }


        private bool IsNeededToSpawnSuperGem()
        {
            _randMegaGem = Random.value;
            if (_randMegaGem < _megaGemSpawnChance)
                return true;
            return false;
        }

        private Vector3 RandPositionOfSpawnCurrency()
        {
            int          maxIndex          =          Mathf.Min(10,
_mainSceneController._towerPatternAllCubePositions.Count);
            int randomIndex = Random.Range(4, maxIndex);

            if                      (randomIndex                      <
_mainSceneController._towerPatternAllCubePositions.Count)
            {
                _currentCurrencyPosition                              =
_mainSceneController._towerPatternAllCubePositions[randomIndex];
                return
_mainSceneController._towerPatternAllCubePositions[randomIndex];
            }
            else
                return
_mainSceneController._towerPatternAllCubePositions[0];
        }

        private     void     ChangeTowerPatternCubeMaterial(Material
cubeMaterial)
        {
```

```
            for (int i = 0; i < _mainSceneController._towerPatternAllCubeObjects.Count; i++)
            {
                if (_mainSceneController._towerPatternAllCubeObjects[i].transform.position == _currentCurrencyPosition)
                {
                    _currentTowerPatternCubeWithCurrency = _mainSceneController._towerPatternAllCubeObjects[i];

                    _mainSceneController._towerPatternAllCubeObjects[i].GetComponent<MeshRenderer>().material = cubeMaterial;
                    break;
                }
            }
        }

        private void UpdateNumberOfGemsCollected(int collectedNumberOfGems)
        {
            TotalNumberOfGemsCollected += collectedNumberOfGems;
            PlayerPrefs.SetInt("AmountOfGems", PlayerPrefs.GetInt("AmountOfGems") + collectedNumberOfGems);

            _UIManager.UpdateGemsNumberUI(TotalNumberOfGemsCollected, collectedNumberOfGems);
        }
    }
```

Лістинг Б.2 — GhostCubeController.cs

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;

    public class GhostCubeController : MonoBehaviour
    {
        [SerializeField] private MainGameController _mainSceneController;
        [SerializeField] private Transform _ghostCubeTransform;
        [SerializeField] private MeshRenderer _ghostCubeMeshRenderer;
        [SerializeField] private Material _rightPlacementPositionMaterial;
        [SerializeField] private Material _wrongPlacementPositionMaterial;

        private WaitForSeconds _amountOfTimeUntilGhostCubePositionChangesWFS = null;
        private List<Vector3> _availablePositionsForGhostCube = new List<Vector3>(5);
    private List<Vector3> _neededPositionsForPlayer = new
```

```
List<Vector3>(2);
        [SerializeField] private float _timeToShowRightPosition = 5f;
        Vector3[] _directionsOfGhostCube = new Vector3[]
        {
            new Vector3(1, 0, 0),
            new Vector3(-1, 0, 0),
            new Vector3(0, 1, 0),
            new Vector3(0, -1, 0),
            new Vector3(0, 0, 1),
            new Vector3(0, 0, -1)
        };

        public void ActivateGhostCubeGeneration()
        {
            _amountOfTimeUntilGhostCubePositionChangesWFS    =    new
WaitForSeconds(_mainSceneController.CurrentLevelData.amountOfTimeUnt
ilGhostCubePositionChanges);
            _ghostCubeTransform.gameObject.SetActive(true);
            StartCoroutine(GenerateAndChangeGhostCubePosition());
        }

        public void DeactivateGhostCubeGeneration()
        {
            _ghostCubeTransform.gameObject.SetActive(false);
            StopCoroutine(GenerateAndChangeGhostCubePosition());
        }

        private IEnumerator GenerateAndChangeGhostCubePosition()
        {
            while (true)
            {
                GenerateGhostCubePositionFromAvailablePositions();
                yield                                          return
_amountOfTimeUntilGhostCubePositionChangesWFS;
            }
        }

        public void SortAvailablePositionsForGhostCubeGenerator()
        {
            _availablePositionsForGhostCube.Clear();
            foreach (var direction in _directionsOfGhostCube)
            {
                Vector3                    newPos                    =
_mainSceneController._currentCubePositionPlacedByPlayer + direction;
                if (IsPositionEmpty(newPos))
                {
                    _availablePositionsForGhostCube.Add(newPos);
                }
            }

            _neededPositionsForPlayer.Clear();
            foreach        (var        availablePosition        in
_availablePositionsForGhostCube)
```

```
            {
if(_mainSceneController._towerPatternAllCubePositions.Contains(avail
ablePosition))
                {
_neededPositionsForPlayer.Add(availablePosition);
                }
            }
            UniversalTimer.Instance.ResetTimer();
        }

        public bool IsPositionEmpty(Vector3 TargetPos)
        {
            foreach            (Vector3            pos            in
_mainSceneController._allCubePositionsPlacedByPlayer)
            {
                if (pos == TargetPos)
                    return false;
            }
            return true;
        }

        public                                                  void
GenerateGhostCubePositionFromAvailablePositions()
        {
            if      (UniversalTimer.Instance.GetElapsedTime()      >
_timeToShowRightPosition && _neededPositionsForPlayer.Count != 0)
            {
                _ghostCubeTransform.position                        =
_neededPositionsForPlayer[Random.Range(0,
_neededPositionsForPlayer.Count)];
                _ghostCubeMeshRenderer.material                     =
_rightPlacementPositionMaterial;
                UniversalTimer.Instance.ResetTimer();
            }
            else
            {
                List<Vector3>     availablePositions     =     new
List<Vector3>(_availablePositionsForGhostCube);

                if (availablePositions.Count == 1)
                    _ghostCubeTransform.position                    =
availablePositions[0];
                else if (availablePositions.Count == 0)
                {
                    _mainSceneController.HandleLoss("NoPositions");
                }
                else
                {
                    if
(availablePositions.Contains(_ghostCubeTransform.position))
                    {
```

```
availablePositions.Remove(_ghostCubeTransform.position);
                    }
                    _ghostCubeTransform.position                    =
availablePositions[Random.Range(0, availablePositions.Count)];

                    if
(_neededPositionsForPlayer.Contains(_ghostCubeTransform.position))
                    {
                        _ghostCubeMeshRenderer.material             =
_rightPlacementPositionMaterial;
                        UniversalTimer.Instance.ResetTimer();
                    }
                    else
                    {
                        _ghostCubeMeshRenderer.material             =
_wrongPlacementPositionMaterial;
                    }
                }
            }
        }
    }
```

ДОДАТОК В

Лістинг В.1 — GameStartCountdown.cs

```
using PrimeTween;
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

public class GameStartCountdown : MonoBehaviour
{
    [SerializeField] private CurrencySpawner _currencySpawner;
    [SerializeField] private UIManager _UIManager;
    [SerializeField]        private        GhostCubeController
_ghostCubeController;
    [SerializeField] private Text _countdownDisplay;
    [SerializeField] private GameObject _currentLevelDisplay;
    [SerializeField] private AudioSource _audioSource;
    private int countdownValue = 3;

    private float _delay = 0.5f;
    private WaitForSeconds _delayWFS;

    private void Start()
    {
        _delayWFS = new WaitForSeconds(_delay);
    }

    public IEnumerator StartCountdown()
    {
        _currentLevelDisplay.GetComponent<Text>().text = "Level
" + MainGameController.Instance.CurrentLevelData.currentLevel;
        _currentLevelDisplay.SetActive(true);
        yield                                                return
Tween.Scale(_currentLevelDisplay.GetComponent<RectTransform>(),
startValue:     1f,     endValue:     1.3f,     duration:     0.8f,
Ease.InOutSine).ToYieldInstruction();
        yield                                                return
Tween.Alpha(_currentLevelDisplay.GetComponent<CanvasGroup>(),
startValue:     1f,     endValue:     0f,     duration:     0.5f,
Ease.InOutSine).ToYieldInstruction();
        _currentLevelDisplay.SetActive(false);

        _countdownDisplay.gameObject.SetActive(true);

        while (countdownValue > 0)
        {
            _countdownDisplay.text = countdownValue.ToString();
            _countdownDisplay.gameObject.SetActive(true);
            yield                                            return
```

```
Tween.Scale(_countdownDisplay.rectTransform,     startValue:     1f,
endValue: 1.3f, duration: 0.3f, Ease.Linear).ToYieldInstruction();
                yield                                              return
Tween.Alpha(_countdownDisplay.GetComponent<CanvasGroup>(),
startValue:     1f,     endValue:     0f,     duration:     0.2f,
Ease.Linear).ToYieldInstruction();
                _countdownDisplay.gameObject.SetActive(false);
                _countdownDisplay.GetComponent<CanvasGroup>().alpha
= 1f;

                yield return _delayWFS;
                countdownValue--;
            }

            _countdownDisplay.gameObject.SetActive(false);
            countdownValue = 3;
            _currentLevelDisplay.GetComponent<CanvasGroup>().alpha =
1f;


_ghostCubeController.SortAvailablePositionsForGhostCubeGenerator();
            _ghostCubeController.ActivateGhostCubeGeneration();
            _currencySpawner.ActivateGemsGeneration();

            MainGameController.Instance.isCubePlacingAllowed = true;
            GameLevelCountdown.Instance.StartCountdown();


_UIManager.EnableDisableTurnOnOffPlacedCubesButton(true);
        }
    }
```

ДОДАТОК Г

Лістинг Г.1 — AdManager.cs

```
using UnityEngine;
using GoogleMobileAds.Api;
using UnityEngine.Networking;
using System.Collections;

public class AdManager : MonoBehaviour
{
    public static AdManager Instance;

    public InterstitialAd interstitialAd;
    public RewardedAd rewardedAd;

#if UNITY_ANDROID
    private string _interstitialAdUnitId = "ca-app-pub-
3940256099942544/1033173712";
    private string _rewardedAdUnitId = "ca-app-pub-
3940256099942544/5224354917";
#endif

    private void Awake()
    {
        if (Instance != null)
        {
            Destroy(gameObject);
        }
        else
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            MobileAds.Initialize((InitializationStatus
initStatus) => { });
        }
    }

    private void Start()
    {

StartCoroutine(CheckInternetConnectionRequest("atAppStart"));
    }

    public void LoadInterstitialAd()
    {
        if (interstitialAd != null)
        {
            interstitialAd.Destroy();
            interstitialAd = null;
```

```
            }

            var adRequest = new AdRequest();

            InterstitialAd.Load(_interstitialAdUnitId, adRequest,
                (InterstitialAd ad, LoadAdError error) =>
                {
                    if (error != null || ad == null)
                    {
                        Debug.LogError("Interstitial  ad  failed  to
load an ad " + "with error: " + error);
                        return;
                    }
                    interstitialAd = ad;
                });
        }

        public void LoadRewardedAd()
        {
            if (rewardedAd != null)
            {
                rewardedAd.Destroy();
                rewardedAd = null;
            }

            var adRequest = new AdRequest();

            RewardedAd.Load(_rewardedAdUnitId, adRequest,
                (RewardedAd ad, LoadAdError error) =>
                {
                    if (error != null || ad == null)
                    {
                        Debug.LogError("Rewarded ad failed to load an
ad " + "with error : " + error);
                        return;
                    }
                    rewardedAd = ad;
                });
        }

        public void CheckConnectionAndLoadInterstitialAd()
        {

StartCoroutine(CheckInternetConnectionRequest("InterstitialAd"));
        }

        public void CheckConnectionAndLoadRewardedAd()
        {

StartCoroutine(CheckInternetConnectionRequest("RewardedAd"));
        }

        IEnumerator            CheckInternetConnectionRequest(string
```

```
adTypeToLoad)
        {
            UnityWebRequest          request          =          new
UnityWebRequest("https://www.google.com/");
            yield return request.SendWebRequest();

            if (request.error != null)
            {
                Debug.Log("No Internet Connection!");
            }
            else
            {
                if (adTypeToLoad == "atAppStart")
                {
                    LoadInterstitialAd();
                    LoadRewardedAd();
                }

                if (adTypeToLoad == "InterstitialAd")
                    LoadInterstitialAd();
                else if (adTypeToLoad == "RewardedAd")
                    LoadRewardedAd();
            }
        }
    }
```

Лістинг Г.2 — FreeGemsButtonController.cs

```
    using System.Collections;
    using UnityEngine;
    using UnityEngine.Networking;
    using UnityEngine.UI;

    public class FreeGemsButtonController : MonoBehaviour
    {
        [SerializeField]               private               Sprite
_connectionProblemButtonImage;
        [SerializeField] private Sprite _idleButtonImage;
        [SerializeField] private GameObject _loadingIcon;

        private WaitForSeconds _requestDelay;
        private Button _button;
        private GameObject _childGameObject;
        private Image _buttonSprite;
        private bool _isNotCalledError = true;
        private bool _isNotCalled = true;

        private void Start()
        {
            _button = gameObject.GetComponent<Button>();
            _childGameObject = transform.GetChild(0).gameObject;
            _buttonSprite = gameObject.GetComponent<Image>();
```

```
            _requestDelay = new WaitForSeconds(3);

            StartCoroutine(CheckInternetConnectionRequest());
        }

        IEnumerator CheckInternetConnectionRequest()
        {
            while (true)
            {
                UnityWebRequest request = new
UnityWebRequest("https://www.google.com/");
                yield return request.SendWebRequest();

                if (request.error != null)
                {
                    Debug.Log("No Internet Connection!");
                    _isNotCalled = true;
                    if (_isNotCalledError)
                    {
                        _button.enabled = false;
                        _childGameObject.SetActive(false);
                        _buttonSprite.sprite =
_connectionProblemButtonImage;
                        _isNotCalledError = false;
                        _loadingIcon.SetActive(false);
                    }
                }
                else
                {
                    _isNotCalledError = true;
                    if (_isNotCalled)
                    {
                        _button.enabled = true;
                        _buttonSprite.sprite = _idleButtonImage;
                        _childGameObject.SetActive(true);
                        _isNotCalled = false;
                    }
                }
                yield return _requestDelay;
            }
        }
    }
```