

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування інституту, факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка
до дипломного проєкту (роботи)
бакалавра
(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА МОБІЛЬНОЇ АРКАДНОЇ ГРИ ДЛЯ ОС АНДРОЇД**

Виконав: студент 4 курсу, групи **КНТ-519**
спеціальності

123 «Комп'ютерна інженерія»
(код і найменування спеціальності)

Освітня програма (спеціалізація)
«Комп'ютерна інженерія»

СУШКО О.С.
(ПРИЗВИЩЕ та ініціали)

Керівник **ГРУШКО С.С.**
(ПРИЗВИЩЕ та ініціали)

Рецензент **СТЕПАНЕНКО О.О.**
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

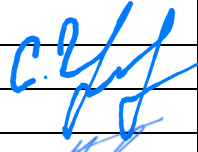
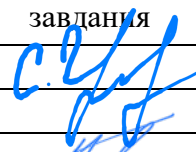
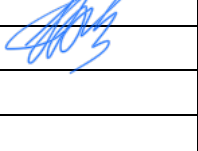
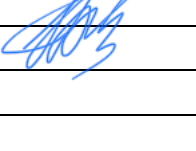
ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
“ ” 2023 року

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

СУШКО Олександр Сергійович
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільної аркадної гри для ос андроїд
керівник проєкту (роботи) к. т. н., доцент, Грушко Світлана Сергіївна
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
- затверджені наказом вищого навчального закладу від “24” березня 2023 року № 71
2. Строк подання студентом проєкту (роботи) 10 травня 2023 року
3. Вихідні дані до проєкту (роботи) наявні методи та технології реалізації мобільної аркадної гри на базі операційної системи Android
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1)Аналіз технологій та технічних вимог, розробка концепції гри
2)Проектування
3)Реалізація
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схема видів програшу;
Схема інтерфейсу сцени меню;
Схема інтерфейсу ігрової сцени;
Схема інтерфейсу сцени магазину.

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ГРУШКО С.С., к.т.н., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.03.2023 р.

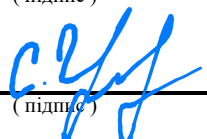
КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів та алгоритмів пошуку об'єктів, перегляд отриманих результатів, виділення	01.03.2023 р.	
	недоліків та переваг		
2	Розробка структури системи	15.03.2023 р.	
3	Розробка алгоритму роботи системи	20.03.2023 р.	
4	Реалізація системи уточненого розпізнавання	01.04.2023 р.	
5	Дослідження системи уточненого розпізнавання	10.04.2023 р.	
6	Оформлення отриманих результатів у ПЗ	01.05.2023 р.	
7	Оформлення графічного матеріалу	10.05.2023 р.	
8	Оформлення допоміжного матеріалу	17.05.2023 р.	

Студент (ка)

 Олександр СУШКО
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

 Світлана ГРУШКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 83 с., 45 рис., 53 джерела.

UNITY, VISUAL STUDIO, АРКАДА, ІГРОВИЙ ДВИГУН, МОБІЛЬНА ГРА, ОПЕРАЦІЙНА СИСТЕМА ANDROID

Об'єкт розробки — розробка мобільної аркадної гри на платформі Android.

Метою роботи є розробка мобільної аркадної гри на базі операційної системи Android, аналіз програмних засобів та технологій розробки, перегляд та аналіз технічних вимог до мобільних ігор на платформі Android, розробка концепції, проектування, розробка структури, реалізація та тестування гри.

Проект складається з трьох розділів.

У першому розділі буде розглянуто різні технології для розробки мобільних ігор, такі як ігрові двигуни, середовище сценаріїв, мови програмування та інші, буде проведений аналіз технічних вимог до мобільних ігор на платформі Android. Також буде розроблена концепція гри, її ігровий процес, механіка та інші аспекти, буде розглянуто жанр та цільову аудиторію гри.

У другому розділі буде розглянуто архітектуру гри, включаючи структуру проекту, модулі, компоненти та їх взаємодію. Також буде спроектований дизайн гри, включаючи визначення інтерфейсу користувача, об'єктів та інших елементів гри.

У третьому розділі описується реалізація гри, а саме розробка інтерфейсу та дизайну гри, реалізація ігрового процесу, механік гри, реалізація звукового дизайну та ефектів, а також розробка програмного коду і тестування гри.

В результаті виконання дипломної роботи проведено аналіз, моделювання та розробка мобільної аркадної гри на платформі Android. Також приведені рекомендації для подальшого розвитку та вдосконалення гри.

ЗМІСТ

Вступ.....	7
1 Аналіз технологій та технічних вимог, розробка концепції гри	9
1.1 Огляд і вибір технологій розробки (ігрові двигуни, мови програмування, середовище сценаріїв, тощо) на основі обґрунтування	9
1.1.1 Аналіз і вибір ігрового двигуна	10
1.1.2 Аналіз і вибір мови програмування.....	24
1.1.3 Аналіз і вибір інтегрованого середовища розробки сценаріїв.....	28
1.2 Огляд технічних вимог до мобільних ігор на платформі Android	31
1.2.1 Характеристики мобільного пристрою	32
1.2.2 Версії операційної системи Android	32
1.3 Розробка концепції гри, включаючи жанр, цільову аудиторію, геймплей, механіку та інші аспекти.....	36
1.3.1 Жанр.....	36
1.3.2 Цільова аудиторія	37
1.3.3 Основна концепція гри.....	37
2 Проектування гри.....	38
2.1 Розробка архітектури програмного продукту, включаючи структуру проекту, модулі, компоненти та їх взаємодію	38
2.1.1 Сцена «Меню».....	40
2.1.2 Сцена ігрового процесу.....	41
2.1.3 Сцена «Магазин».....	43
2.2 Проектування дизайну гри, включаючи визначення інтерфейсу користувача, об'єктів та інших елементів гри.....	44
2.2.1 Проектування дизайну та інтерфейсу сцени «Меню».....	44
2.2.2 Проектування дизайну та інтерфейсу у ігровій сцені	46
2.2.3 Проектування дизайну та інтерфейсу у сцені магазину.....	49
3 Реалізація гри.....	50

3.1 Реалізація інтерфейсу та дизайну гри, реалізація геймплею, механік гри, взаємодії гравця з грою, реалізація звукового дизайну, ефектів та іншого	51
3.1.1 Реалізація дизайну та інтерфейсу користувача.....	51
3.1.2 Розробка анімацій.....	59
3.1.3 Розробка ефектів.....	61
3.1.4 Розробка звукового дизайну	62
3.1.5 Розробка основних об'єктів геймплею та механік гри	66
3.2 Розробка програмного коду гри відповідно до вибраної технології та концепції гри	69
3.3 Тестування гри	75
3.3.1 Тестування ігрового процесу та виявлення несправностей	76
3.3.2 Тестування продуктивності гри.....	76
Висновки.....	78
Перелік джерел посилання	80
Додаток А	84
Додаток Б.....	88
Додаток В	94

Графічний матеріал:

Плакат 1 - Схема видів програшу

Плакат 2 - Схема інтерфейсу сцени меню

Плакат 3 - Схема інтерфейсу ігрової сцени

Плакат 4 - Схема інтерфейсу сцени магазину

ВСТУП

Смартфони стали невіддільною частиною сучасного життя, і для багатьох людей вони є важливим інструментом. В наш час, смартфони стали не тільки зручним засобом зв'язку, а й незамінним інструментом для роботи, навчання, розваги та багатьох інших завдань. З ними ми можемо спілкуватися з друзями та близькими, надсилати повідомлення та дзвінки, дивитися новини та залишатися в курсі подій у світі. І також разом з цим, смартфони стали однією з найбільших платформ для ігор у сучасному світі. Мобільні ігри допомагають відволіктися і скоротати час. Багато людей грають у мобільні ігри під час щоденних поїздок, чекаючи в черзі або під час перерв на роботі. Вони пропонують швидкий і простий спосіб розслабитися та відпочити від стресу повсякденного життя.

За останні роки, індустрія мобільних ігор розширилася до неймовірних розмірів. Зі зростанням популярності смартфонів і планшетів все більше людей грають в ігри на своїх мобільних пристроях. Згідно зі звітом Newzoo, у 2020 році світовий ринок мобільних ігор оцінювався в 77,2 мільярда доларів, а до 2024 року очікується, що він зросте до 116,3 мільярда доларів [1]. Однією з основних причин популярності мобільних ігор є їх доступність. Мобільні ігри можна легко завантажити та грати будь-де та будь-коли. Вони пропонують різноманітні жанри, включаючи аркади, ігри-головоломки, стратегії, рольові ігри та інші. Багато мобільних ігор також є безкоштовними, хоча покупки в додатку та реклама є звичайними способами отримання прибутку для розробників.

Для смартфонів існує безліч операційних систем, серед них трьома найпопулярнішими мобільними ОС є Android, iOS і KaiOS. Для цього проекту була обрана операційна система Android. Одні із головних причин, чому була вибрана саме ця операційна система це — її популярність та розвиненість. Вона є найпопулярнішою мобільною операційною системою у світі, частка ринку якої становила понад 70% станом на 2021 рік. Розроблена Google Android є платформою з відкритим кодом, яка використовується багатьма виробниками смартфонів і

планшетів, такими як Xiaomi, Samsung, та інших. Ця операційна система сумісна з широким спектром пристроїв, бо вона була розроблена для роботи на пристроях різного розміру та специфікацій, від недорогих смартфонів до високоякісних планшетів, що дозволяє користувачам вибрати пристрій, який відповідає їх бюджету та вимогам. Це розширює спектр юзерів, які зможуть грати в гру.

Мета дослідження дипломної роботи — розробити мобільну аркадну гру на базі операційної системи Android.

Завдання дослідження дипломної роботи:

- переглянути програмні засоби та технології розробки мобільних ігор;
- проаналізувати технічні вимоги до мобільної гри на платформі Android;
- розробити концепцію гри;
- спроектувати та розробити структуру гри;
- реалізувати гру;
- протестувати розроблену гру.

1 АНАЛІЗ ТЕХНОЛОГІЙ ТА ТЕХНІЧНИХ ВИМОГ, РОЗРОБКА КОНЦЕПЦІЇ ГРИ

Розробка мобільних ігор — це процес створення ігор, в які можна грати на мобільних пристроях, таких як смартфони та планшети. Мобільні ігри зазвичай менші за розміром і доступніші, ніж ігри для консолей або ПК (персональний комп'ютер), і в них можна грати на ходу. Розробка гри є складним і трудомістким процесом, який містить у собі різні етапи, такі як розробка концепції, підготовка до виробництва, виробництво, тестування та випуск. Процес може тривати від кількох місяців до кількох років, залежно від обсягу та складності гри. Для розробки потребується безліч технологій, включаючи ігрові двигуни, мови програмування, ігрові механізми, комплекти розробки програмного забезпечення та інші. Вибір технологій має дуже важливе значення для розробки, оскільки це має значний вплив на стабільність гри, час розробки та загальний успіх [2].

Також дуже важливо брати до уваги системні вимоги мобільної гри, оскільки вони визначають пристрої та версії операційної системи, на яких гра може працювати безперебійно. Якщо гра потребує більше ресурсів, ніж може надати пристрій, або вона використовує версію операційної системи яку не підтримує пристрій вона може не працювати взагалі або працювати погано, що призведе до поганої взаємодії з користувачем.

1.1 Огляд і вибір технологій розробки (ігрові двигуни, мови програмування, середовище сценаріїв, тощо) на основі обґрунтування

Аналіз і вибір технологій має вирішальне значення для розробки. Нижче наведено основні причини, чому вибір технології важливий у розробці мобільних ігор:

- перформанс. Вибір технології може вплинути на перформанс гри, зокрема на її графіку, швидкість та стабільність. Використання неефективної технології може призвести до низької продуктивності, що може негативно вплинути на досвід гравця;

- час розробки. Вибір правильної технології може значно скоротити час розробки гри. Наприклад від вибору ігрового двигуна та інструментів розробки, може залежати швидкість та ефективність розробки;

- вартість. Для деяких технологій можуть знадобитися додаткові ліцензії, плагіни або апаратне забезпечення, що може збільшити витрати на розробку. Вибір правильної технології може допомогти зберегти витрати на розробку в рамках бюджету;

- сумісність з платформами. Мобільні ігри мають бути сумісні з різними операційними системами та конфігураціями обладнання. Наприклад, якщо ми у майбутньому захочемо вийти на ринок iOS, щоб збільшити охоплення гри, то в такому випадку, нам дуже важливий вибір технології, яка підтримує кросплатформну розробку, яка може спростити перенесення гри на інші платформи;

- майбутня масштабованість. Вибір технології може вплинути на майбутню масштабованість гри, включаючи можливість додавати нові функції та підтримку майбутніх оновлень.

Таким чином, вибір технології має вирішальне значення для розробки мобільних ігор, оскільки це може вплинути на продуктивність гри, час розробки, вартість, сумісність з різними платформами та майбутню масштабованість. Тому необхідно ретельно оцінити та вибрати правильну технологію.

1.1.1 Аналіз і вибір ігрового двигуна

Ігровий двигун (game engine) — це програмне забезпечення, яке полегшує процес створення комп'ютерних ігор. Він являє собою набір інструментів та бібліотек, які дозволяють розробникам створювати ігровий контент, керувати ігровим процесом, працювати з графікою, звуками та фізикою, обробляти введення

та виведення, а також реалізовувати та керувати ігровою логікою та поведінкою об'єктів.

Ігровий двигун надає розробникам готовий до використання набір інструментів, в який входить:

- рендеринг графіки. Двигуни надають різні методи рендерингу, такі як трасування променів та інші, а також інструменти для роботи з 3D-моделями, текстурами та ефектами;

- фізика. Двигуни надають інструменти для роботи з фізикою, такі як зіткнення об'єктів, гравітація, а також симуляція рідин, тканин та інших фізичних процесів;

- обробка введення та виведення. Двигуни дозволяють обробляти введення від гравців і виводити інформацію на екран, а також працювати зі звуком;

- управління ігровим процесом. Двигуни надають інструменти для управління ігровим процесом, такі як створення рівнів, системи збереження та завантаження гри, керування камерою та інтерфейсом користувача [3].

Використання ігрового двигуна дозволяє сильно скоротити час розробки ігор, оскільки багато базових функцій вже реалізовані в ньому.

Існує безліч ігрових двигунів для створення мобільних ігор, найпопулярніші з них:

- Godot Engine;
- Construct 3;
- Cocos2D;
- Corona SDK;
- GameMaker Studio;
- Unreal Engine;
- Unity.

«Godot Engine»

Godot Engine — це ігровий двигун з відкритим вихідним кодом, який був випущений у 2014 році. На рисунку 1.1 зображено його логотип. Він підтримується спільнотою Godot Engine і ліцензується згідно з ліцензією MIT, що робить його

безкоштовним як для комерційного, так і для некомерційного використання. Двигун використовує систему на основі вузлів, що дозволяє розробникам легко створювати та організовувати ігрові об'єкти [4, 6].

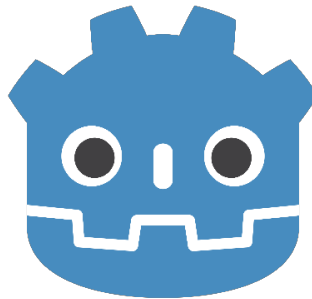


Рисунок 1.1 — Логотип ігрового двигуна «Godot Engine» [5]

Розглянемо його переваги:

- система на основі вузлів. Система Godot на основі вузлів є однією з її найважливіших переваг. Це дозволяє легко створювати складні ігрові об'єкти, комбінуючи кілька менших. Ця функція також полегшує організацію ігрових об'єктів і керування ними, що особливо корисно для великих проектів;
- сценарії. Godot підтримує кілька мов сценаріїв, включаючи GDScript, C# і Visual Scripting, що дозволяє розробникам вибирати мову, яка найкраще відповідає їхнім потребам;
- кросплатформенність. Godot Engine підтримує широкий спектр платформ, включаючи Windows, macOS, Linux, Android, iOS і такі вебплатформи, як HTML5;
- 2D і 3D. Godot підтримує розробку як 2D, так і 3D ігор, що робить його універсальним двигуном для розробників ігор;
- активна спільнота. Спільнота Godot є дуже активною та підтримує, з великою кількістю посібників, документації та плагінів, які допомагають розробникам створювати свої ігри.

Розглянемо його недоліки:

- крива навчання. Хоча система Godot на основі вузлів є значною перевагою, вона також може ускладнити навчання тим, хто з нею не знайомий;

- обмежена документація. Хоча спільнота Godot є активною та підтримує, документація двигуна все ще дещо обмежена, особливо якщо порівнювати з іншими популярними ігровими двигунами, такими як Unity;
- підтримка сторонніх плагінів. Підтримка плагінів Godot не така широка, як у інших ігрових двигунів, що може бути обмеженням для деяких розробників;
- продуктивність. Хоча Godot здатний створювати високоякісні ігри, він може працювати не так добре, як інші ігрові двигуни, особливо коли йдеться про складну 3D-графіку [7, 8].

«Construct 3»

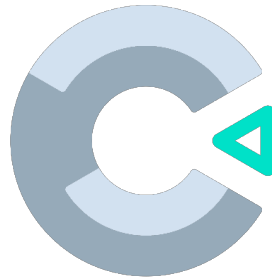


Рисунок 1.2 — Логотип ігрового двигуна «Construct 3» [9]

Construct 3 — це механізм веброзробки ігор, розроблений компанією Scirra Ltd. На рисунку 1.2 зображено його логотип. Він дозволяє користувачам створювати 2D-ігри та інший інтерактивний вміст, включаючи анімацію, симуляції та навчальні матеріали, не вимагаючи знань програмування [10].

Його переваги:

- знання програмування не потрібні. Construct 3 пропонує інтерфейс перетягування, що дозволяє користувачам створювати ігри без написання коду;
- кросплатформенність. Ігри, створені за допомогою Construct 3, можна публікувати на різних платформах, включаючи Windows, macOS, Linux, iOS, Android і в Інтернеті;
- спільна розробка. Він дозволяє співпрацювати між кількома користувачами, дозволяючи розробникам ігор працювати разом над проектами в режимі реального часу;

- розширюваність. Construct 3 підтримує плагіни та поведінку, що дозволяє користувачам розширювати функціональні можливості двигуна.

Його недоліки:

- обмежена підтримка 3D. Хоча Construct 3 підтримує 3D-графіку, вона в основному розроблена для розробки 2D-ігор;

- модель підписки. Construct 3 вимагає підписки, щоб отримати доступ до повних функцій і функцій, що робить її менш доступною для розробників-любителів з обмеженим бюджетом;

- обмежені параметри експорту. Параметри експорту у Construct 3 обмежені, оскільки користувачі можуть публікувати лише на певних платформах [11].

«Cocos2D»



Рисунок 1.3 — Логотип ігрового двигуна «Cocos2D» [12]

Cocos2D — це популярний ігровий двигун із відкритим кодом, який широко використовується для розробки 2D-ігор. На рисунку 1.3 зображено його логотип. Спочатку він був розроблений на Python, але тепер він підтримує різні мови програмування, такі як Objective-C, Swift, Java, JavaScript і C++ [13].

Розглянемо його основні переваги:

- простий у використанні. Cocos2D простий у використанні та має інтуїтивно зрозумілий інтерфейс, який спрощує розробку гри;

- кросплатформеність. Він підтримує кілька платформ, таких як iOS, Android, Mac, Windows і Linux, що робить його ідеальним вибором для розробки кросплатформних ігор;

- велика аудиторія. Цей двигун має велике та активне співтовариство розробників, які роблять внесок у розвиток двигуна та діляться своїми знаннями та досвідом;

- розширюваність. Його можна розширювати та він дозволяє розробникам додавати власні функції та функціональні можливості до двигуна;

- повна документація. Є велика документація, яка включає навчальні посібники та приклади, що полегшують вивчення та використання двигуна розробниками.

Розглянемо його основні недоліки:

- обмежені 3D-можливості. Cocos2D в основному призначений для розробки 2D-ігор і має обмежені 3D-можливості;

- крута крива навчання. Хоча він простий у використанні, він має круту криву навчання для початківців, які тільки починають розробляти ігри;

- обмежена підтримка спільноти для деяких мов програмування. Хоча Cocos2D підтримує кілька мов програмування, підтримка спільноти для деяких із цих мов може бути обмеженою [14].

«Corona SDK»



Рисунок 1.4 — Логотип ігрового двигуна «Corona SDK» [15]

SDK розшифровується як «Software Development Kit», який являє собою набір програмних засобів, бібліотек і документації, які використовуються

розробниками для створення програмних програм для певної платформи чи операційної системи. SDK зазвичай включає різноманітні ресурси, які можуть допомогти розробникам у таких завданнях, як написання, тестування та налагодження коду, а також створення інтерфейсів користувача, доступ до даних тощо. SDK зазвичай надається розробником або постачальником платформи чи операційної системи, щоб дозволити розробникам створювати програми, які можуть працювати на цій платформі чи системі. Наприклад Google надає Android SDK, який розробники можуть використовувати для створення програм Android.

Ключові компоненти SDK:

- інструменти програмування. SDK може містити редактор коду, компілятор та інші інструменти, які допомагають розробникам писати та тестувати код;
- бібліотеки. Також тут можуть бути попередньо створені бібліотеки коду, які можна використовувати для виконання звичайних завдань, таких як обробка введених користувачем даних, відображення графіки або доступ до даних;
- документація. Зазвичай вони містять документацію, яка пояснює, як використовувати різні компоненти SDK, включаючи інструменти програмування та бібліотеки;
- зразки та приклади. Він може мати у собі зразки коду та приклади, які демонструють, як використовувати SDK для створення програм [16].

Corona SDK — це кросплатформний механізм розробки ігор, який дозволяє розробникам створювати ігри для мобільних пристроїв (iOS і Android), а також для настільних комп'ютерів (Windows і macOS) за допомогою мови програмування Lua. На рисунку 1.4 зображено логотип цього ігрового двигуна. Його розробила компанія Corona Labs Inc., розташована в Пало-Альто, Каліфорнія. Він доступний як у безкоштовній, так і в платній версіях. Безкоштовна версія дозволяє розробникам створювати та публікувати програми з певними обмеженнями, такими як заставка та обмежений доступ до певних функцій. Платна версія, доступна за передплатою, надає повний доступ до всіх функцій двигуна та дозволяє розробникам публікувати без будь-яких обмежень [17].

Ось переваги цього двигуна:

- легко освоїти. Як основна мова програмування, тут використовується Lua, яку відносно легко освоїти навіть для новачків. Механізм також містить обширну документацію та навчальні посібники, що полегшує початок роботи;

- кросплатформеність. Corona SDK дозволяє розробникам створювати ігри, які можна запускати на кількох платформах, включаючи iOS, Android, Windows і macOS;

- швидко та ефективно. Цей двигун дуже оптимізований для продуктивності, що дає змогу створювати ігри, які безперебійно працюють на мобільних пристроях із недорогим апаратним забезпеченням;

- велика аудиторія. Його аудиторія є активною та корисною, з багатьма форумами та доступними ресурсами, щоб допомогти розробникам у їхніх проектах;

- вбудований фізичний двигун. Corona SDK має вбудований фізичний двигун, що полегшує створення реалістичних рухів і зіткнень в іграх.

Недоліки Corona SDK:

- обмежена підтримка 3D. Corona SDK розроблено в основному для розробки 2D-ігор, і хоча він пропонує деякі базові функції 3D, він є не повноцінним 3D-двигуном;

- обмежена підтримка сторонніх розробників. Порівняно з іншими ігровими двигунами, Corona SDK має відносно невелику бібліотеку сторонніх плагінів і ресурсів;

- відсутність візуального редактора. На відміну від деяких інших ігрових двигунів, Corona SDK не має візуального редактора, що може ускладнити роботу з ним деяким розробникам;

- платні функції. Деякі з функцій Corona SDK, як, наприклад аналітика та хмарні служби потребують платної підписки [18].

«GameMaker Studio»

GameMaker Studio — це популярний 2D-ігровий двигун. Вперше він був створений Марком Овермарсом і випущений у 1999 році як інструмент програмування під назвою «Animo», але пізніше був придбаний YoYo Games у 2007 році. На рисунку 1.5 зображено логотип цього ігрового двигуна. GameMaker

Studio підтримує створення ігор для широкого спектра платформ, включаючи Windows, macOS, Linux, iOS, Android тощо [19].



Рисунок 1.5 — Логотип ігрового двигуна «GameMaker Studio» [20]

Переваги GameMaker Studio:

- простий у використанні. GameMaker Studio має інтерфейс перетягування, що дозволяє легко створювати ігри без будь-якого досвіду програмування;
- кросплатформеність. Двигун підтримує створення ігор для широкого діапазону платформ, включаючи мобільні пристрої та консолі;
- чудово підходить для 2D-ігор. Він був розроблений спеціально для розробки 2D-ігор і надає інструменти для створення анімації, спрайтів і плиткових карт;
- активна аудиторія. Аудиторія GameMaker дуже активна, з багатьма ресурсами, навчальними посібниками та підтримкою, доступною в Інтернеті;

Недоліки GameMaker Studio:

- обмежені 3D-можливості. GameMaker Studio в основному розроблено для розробки 2D-ігор;
- проблеми з оптимізацією. Залежно від складності вашої гри, у вас можуть виникнути проблеми з оптимізацією під час її запуску на певних платформах. Це можна пом'якшити шляхом ручної, ретельної оптимізації, але це все одно може викликати занепокоєння у деяких користувачів;
- погана підтримка багатокористувацьких ігор. Він підтримує багатокористувацьку функціональність, але вона дещо обмежена порівняно з іншими ігровими двигунами. Якщо багатокористувацька гра є ключовою частиною дизайну вашої гри, вам доведеться шукати альтернативні рішення;

- мала кількість бібліотек активів. Міститься відносно невелика бібліотека активів порівняно з іншими ігровими двигунами, що може ускладнити пошук ресурсів, необхідних для створення гри;

- лімітована підтримка платформ. Хоча GameMaker Studio підтримує низку платформ, включаючи ПК, мобільні пристрої та консолі, але не підтримує платформи VR або AR, якщо порівнювати з іншими ігровими двигунами [21].

«Unreal Engine»



Рисунок 1.6 — Логотип ігрового двигуна «Unreal Engine» [22]

Unreal Engine — це ігровий двигун, розроблений і підтримуваний Epic Games. На рисунку 1.6 зображено його логотип. Це потужний і популярний двигун, який використовується для створення високоякісних ігор, а також віртуальної та доповненої реальності. Він являє собою повне середовище розробки, що містить інтегрований редактор, інструменти програмування та набір інших функцій, щоб допомогти розробникам ігор створювати складні та захопливі ігри. Він широко використовується в індустрії розробки ігор, багато ігор класу AAA розробляються за допомогою Unreal Engine [23, 24].

Основні особливості:

- надійна та потужна система візуальних сценаріїв під назвою Blueprint, яка дозволяє розробникам створювати складні ігрові механізми без написання жодного рядка коду;

- широкий набір інструментів і редакторів для створення, імпорту та маніпулювання ресурсами, включаючи редактор ландшафту, редактор матеріалів і редактор частинок;

- підтримка багатьох платформ, включаючи ПК, консолі, мобільні пристрої та пристрої віртуальної реальності;
- можливості рендерингу у реальному часі, включаючи динамічне освітлення та тіні, глобальне освітлення та вдосконалені ефекти постобробки;
- багата екосистема плагінів і інструментів від сторонніх розробників, які розширюють функціональні можливості двигуна;
- вбудована підтримка багатокористувацької онлайн-ігри [25, 26].

Переваги Unreal Engine:

- високоякісна графіка та візуальні ефекти завдяки можливостям рендерингу у реальному часі;
- проста у використанні система візуальних сценаріїв, яка дозволяє розробникам швидко створювати прототипи та впроваджувати механізми ігрового процесу;
- велика та активна спільнота розробників і користувачів, яка надає підтримку, ресурси та можливості для навчання;
- багатоплатформна підтримка, що дозволяє розробникам орієнтуватися на широкий спектр платформ з мінімальними зусиллями;
- повний набір інструментів і редакторів для створення ігрових ресурсів і керування ними.

Недоліки Unreal Engine:

- може потребувати багато ресурсів, особливо під час роботи з великими сценами або ресурсами з високою деталізацією;
- крута крива навчання для початківців, особливо тих, хто не має попереднього досвіду розробки ігор або програмування;
- може вимагати значних витрат часу та зусиль для повного освоєння;
- деякі функції можуть бути обмежені певними платформами або вимагати додаткових ліцензійних зборів;
- обмежена підтримка розробки 2D-ігор [26, 27].

«Unity»

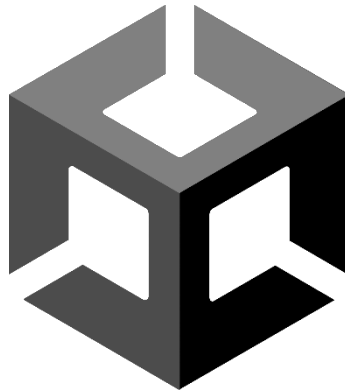


Рисунок 1.7 — Логотип ігрового двигуна «Unity» [28]

Unity — це найпопулярніший кросплатформений ігровий двигун, який широко використовується для розробки ігор, симуляцій та інших інтерактивних програм. На рисунку 1.7 зображено логотип цього ігрового двигуна. Він був вперше випущений у 2005 році компанією Unity Technologies і з того часу став одним із найпоширеніших ігрових двигунів у галузі, який працює над багатьма популярними іграми та програмами на різних платформах, включаючи ПК, мобільні пристрої та консолі.

Unity відомий своїм зручним інтерфейсом, потужним графічним процесором і великою бібліотекою інструментів і ресурсів. Він використовує C# як основну мову програмування і пропонує різноманітні функції, такі як симуляція фізики, анімація, підтримка мережі та віртуальної реальності [29, 30].

Основні переваги Unity:

- кросплатформеність. Однією з найбільших переваг Unity є можливість створювати ігри для багатьох платформ, включаючи ПК, мобільні пристрої, консолі та навіть пристрої VR/AR. Розробники можуть створювати свої ігри один раз і розгортати їх на кількох платформах, заощаджуючи час і ресурси;

- простий у використанні та навчанні. Unity зручний і має простий інтерфейс, що полегшує початківцям навчання та створення ігор. Його інтерфейс із функцією перетягування, система візуальних сценаріїв та інтуїтивно зрозумілий робочий

процес роблять розробку ігор доступною для кожного, незалежно від рівня кваліфікації;

- має одне з найбільших ком'юніті. Unity має величезну спільноту розробників, митців та ентузіастів, що полегшує пошук навчальних посібників, документації та ресурсів, які допоможуть вам створити свою гру. Спільнота також пропонує підтримку, поради та відгуки, що полегшує розв'язання проблем і покращення гри. Крім того, Unity пропонує низку варіантів підтримки, включаючи документацію, форуми та технічну підтримку, щоб допомогти вам вирішити будь-які проблеми, з якими ви зіткнетеся;

- перформанс. Unity оптимізовано для підвищення перформансу з такими функціями, як багатопотоковий рендеринг і підтримка графічних карт високого класу. Це означає, що ви можете створювати ігри, які працюють плавно та чудово виглядають навіть на пристроях низького класу;

- надійне сховище активів (asset store). Сховище активів Unity надає розробникам величезну бібліотеку ресурсів, включаючи моделі, текстури, анімації та аудіозаписи, які вони можуть використовувати у своїх іграх.

Недоліки Unity:

- незручність роботи з великими командами. Розробка ігор з великою командою може стати складним процесом через незручність роботи з Unity в умовах колаборації, тому потрібне хороше планування та організація роботи;

- обмежений візуальний сценарій. Хоча Unity пропонує візуальний сценарій за допомогою інструменту Bolt, він не такий надійний, як деякі інші ігрові двигуни;

- Unity має величезний каталог зручних ресурсів (asset store) як ми сказали вище, але кількість не гарантує якість. Примітно, що деякі важливі активи, такі як скрипти, можуть зіткнутися з проблемами щоразу, коли Unity оновлює свій механізм. Крім того, Unity трохи не вистачає пропозиції, коли мова заходить про багатополігональні 3D-активи. Цю сферу краще обслуговує інший інтенсивно використовуваний ігровий двигун на ринку, Unreal [31];

- висока вартість. Хоча Unity доступний для безкоштовного використання, розробка великих та складних ігор може вимагати додаткових інструментів та послуг, що може збільшити загальну вартість проекту [32, 33].

Проаналізувавши всі перераховані вище ігрові двигуни, для цього проекту був вибраний Unity. Нижче приведені основні аргументи, чому був вибраний саме цей двигун:

- Unity просто створений для мобільних пристроїв. За допомогою Unity візуально красиві ігри можна легко інтегрувати з готовими функціями наявних у кожному смартфоні, такими як GPS, акселерометри, гіроскопи — щоб розблокувати безліч можливостей для розробників ігор. Також у нього в наявності вже є такі функції як аналітика, внутрішньоігрові платежі та реклама, які спрощують стратегії монетизації;

- наступною причиною вибору саме цього двигуна є його кросплатформеність. У будь-який момент, коли гра стане набирати обертів, ми можемо захотіти розширити доступність нашої гри на різних платформах, щоб збільшити охоплення гри на ринку мобільного геймінгу. І тут нам в пригоді стає кросплатформеність;

- ще одним аргументом вибору саме цього двигуна є його простота вивчення. Навчитися розробці ігор на цьому двигуні значно легше, ніж у більшості основних конкурентів Unity, що збільшує кількість розробників Unity на ринку, а отже, ігор, створених Unity;

- одним із головних аргументів і основною причиною вибору саме цього ігрового двигуна Unity є його найбільше ком'юніті розробників, митців та ентузіастів, які створюють і діляться ресурсами, інструментами та знаннями. У Unity є дві основні платформи для взаємодії ком'юніті як між собою, так і з Unity розробниками, це Unity Answers і Unity Forums. Unity Answers — це платформа запитань і відповідей, керована спільнотою, де розробники можуть ставити запитання й отримувати відповіді, пов'язані з розробкою ігор на однойменному двигуні. Вона розроблена, щоб ділитися своїми знаннями та допомагати один одному в проектах. Unity Answers дозволяє розробникам ставити запитання з

широкого кола тем, включаючи функції редактора Unity, скрипти, фізику, графіку, мережу тощо. Запитання можна позначати відповідними ключовими словами, щоб іншим розробникам було легше їх знаходити. Потім інші учасники спільноти можуть переглядати запитання, давати відповіді та голосувати за найбільш корисні відповіді. Unity Answers функціонує як спільна база знань, де розробники можуть знайти відповіді на свої запитання та поділитися власним досвідом з іншими. Ця платформа є дуже цінним ресурсом як для новачків, так і для досвідчених розробників [34]. Unity Forums являється дуже схожою платформою для допомоги розробникам, але є деякі відмінності, які полягають у різниці формату обговорення. Unity Forums організовані як дискусії з гілками, де розробники можуть створювати нові теми та відповідати на вже існуючі, проводити більш відкриті дискусії на ширше коло тем. Unity Answers, з іншого боку, організовані навколо конкретних питань і відповідей, які можна шукати та фільтрувати за темою, тегом і ключовим словом. Загалом і Unity Forums, і Unity Answers є цінними ресурсами для розробників, щоб навчатися, ділитися знаннями та спілкуватися з іншими членами спільноти Unity. Розробники можуть використовувати одну або обидві платформи залежно від своїх конкретних потреб і вподобань. Крім цих офіційних платформ, є просто безліч статей та відео в Інтернеті. Можна навіть сказати так, що якщо у вас виникне якась проблема чи питання, ви можете не переживати, тому що ви з величезною ймовірністю знайдете вже готове рішення або відповідь на ваше питання [35].

1.1.2 Аналіз і вибір мови програмування

Вибір мови програмування є важливим, оскільки це дуже впливає на процес розробки, перформанс гри та на підтримку кодової бази. Ось деякі основні фактори, які залежать від вибору мови програмування:

- досвід розробника. Розробники можуть мати різний рівень досвіду та навичок роботи з різними мовами програмування. Вибір мови, з якою знайомі розробники, може допомогти забезпечити більш плавний процес розробки та зменшити ймовірність помилок;

- перформанс. Деякі мови програмування краще підходять для високопродуктивних додатків, наприклад тих, що вимагають обробки в реальному часі або великомасштабної обробки даних. Вибір оптимізованої для продуктивності мови може допомогти забезпечити безперебійну та ефективну роботу гри;

- сумісність. Вибір мови програмування також може залежати від сумісності мови з іншими інструментами та технологіями, які використовуються в процесі розробки, такими як бібліотеки, фреймворки та бази даних;

- підтримка мови. Вибір мови, яка широко використовується та має велику спільноту розробників, може допомогти забезпечити наявність достатньо ресурсів і підтримки, доступних для підтримки гри з часом.

Ігровий двигун Unity, який був вибраний вище, підтримує три мови програмування:

- UnityScript (також відомий як JavaScript для Unity);
- Boo;
- C#.

UnityScript, також відомий як JavaScript для Unity, був мовою програмування, яка використовувалася в ігровому двигуні Unity. Він базувався на мові JavaScript, але з деякими змінами синтаксису та доповненнями для кращої підтримки розробки ігор. Він був представлений у версії Unity 1.0 у 2005 році як альтернатива C#, яка на той час була основною мовою програмування для Unity. UnityScript був розроблений, щоб бути більш доступним для розробників, які знайомі з веброзробкою та JavaScript, але, можливо, не мали досвіду роботи з C# або іншими мовами об'єктноорієнтованого програмування.

Деякі ключові функції UnityScript:

- динамічна типізація. Це мова з динамічною типізацією, тобто тип даних змінної визначається під час виконання, а не під час компіляції. Це може зробити мову більш гнучкою, але також може призвести до зниження продуктивності;

- об'єктноорієнтоване програмування. UnityScript є об'єктноорієнтованою мовою, тобто підтримує класи, об'єкти та інші концепції ООП. Це полегшує організацію та структурування коду для великих проектів;

- легка інтеграція з Unity. Оскільки UnityScript було розроблено спеціально для використання з ігровим механізмом Unity, він включає вбудовану підтримку багатьох специфічних функцій Unity, таких як ігрові об'єкти, компоненти та фізика;

- більше не підтримується. Однак важливо зазначити, що UnityScript застарів з Unity 2017 і більше офіційно не підтримується Unity. Тепер розробникам рекомендується використовувати C# як основну мову програмування для Unity.

Таким чином, UnityScript була мовою програмування, яка була розроблена спеціально для використання з ігровим двигуном Unity на основі мови JavaScript і спрямована на те, щоб зробити розробку ігор більш доступною для розробників. попри те, що UnityScript вже застарів, він відіграв важливу роль в еволюції Unity як ігрового двигуна [36, 37].

Воо — це мова програмування, яка була спеціально розроблена для створення програмних додатків на платформі .NET. Він був створений Родріго Б. де Олівейра і вперше випущений у 2003 році. Він є статично типізованою мовою, що означає, що змінні мають певний тип даних, який визначається під час компіляції. Це також об'єктноорієнтована мова, що означає, що вона підтримує принципи інкапсуляції, успадкування та поліморфізму.

Деякі ключові особливості цієї мови програмування:

- синтаксичний цукор. Воо відомий своїм лаконічним і виразним синтаксисом, який полегшує читання та написання коду. Він містить низку синтаксичних функцій, які спрощують типові завдання програмування;

- виведення типу. Він використовує визначення типу, що дозволяє компілятору визначати тип даних змінної на основі її використання. Це зменшує потребу в явних оголошеннях типів і робить код більш лаконічним;

- метапрограмування. Ця мова містить потужні функції метапрограмування, які дозволяють розробникам маніпулювати кодом під час виконання. Це можна

використовувати для таких завдань, як генерація коду та динамічна модифікація коду;

- інтеграція з .NET. Оскільки Boo побудований на платформі .NET, він має повний доступ до .NET framework і може взаємодіяти з іншими мовами .NET, такими як C# і VB.NET.

Хоча Boo колись була популярною мовою для створення додатків .NET, останніми роками вона втратила популярність і більше не підтримується. Зараз багато розробників вважають, що краще використовувати C# [37].

C# — це сучасна об'єктноорієнтована мова програмування, розроблена компанією Microsoft. Вперше він був випущений у 2000 році як частина .NET framework і з того часу став популярною мовою для розробки широкого спектра програм, включаючи вебпрограми, настільні програми та ігри. Це строго типізована мова, що означає, що змінні мають бути явно оголошені з типом даних. Він також підтримує принципи інкапсуляції, успадкування та поліморфізму, які є ключовими концепціями об'єктноорієнтованого програмування. Як мова скриптів для Unity, C# широко використовується та рекомендована командою розробників Unity. Це найпопулярніша мова сценаріїв, що використовується в Unity, і пропонує багато переваг, зокрема:

- перформанс. C# є компільованою мовою, що означає, що код перед виконанням транслюється в машинний код. Це дозволяє швидше та ефективніше виконувати код, що особливо важливо для розробки ігор;

- популярність. Він є популярною мовою серед розробників і має велику спільноту користувачів. Це означає, що для розробників, які вивчають або працюють з C# в Unity, є багато ресурсів і підтримки;

- сумісність. Також він сумісний із платформою .NET, яку використовує Unity. Це означає, що C# може легко інтегруватися з іншими мовами .NET, такими як Visual Basic .NET, і може використовувати бібліотеки та фреймворки .NET;

- безпека та захист. Ця мова програмування містить такі функції, як безпека типів і керування пам'яттю, які допомагають запобігти поширеним помилкам програмування та гарантують безпечність і стабільність програми.

Таким чином, C# — це потужна та універсальна мова програмування, яка широко використовується для розробки програм, включаючи ігри в Unity. Його перформанс, сумісність і популярність роблять його чудовим вибором для розробників, які прагнуть створювати надійні та ефективні додатки [38, 39].

Для цього проекту була вибрана мова програмування C#. Це аргументується тим, що C# являється рідною мовою для Unity, він має багато ресурсів і підтримки для розробників, також він має дуже гарний перформанс, а це дуже важливо для ігор. А інші мови, які ми розглянули вище, за допомогою яких можна писати скрипти в Unity такі як UnityScript і Boo вже застаріли та більше не підтримуються.

1.1.3 Аналіз і вибір інтегрованого середовища розробки сценаріїв

Інтегроване середовище розробки (IDE) — це програмний додаток, який надає розробникам комплексний набір інструментів для написання, тестування та налагодження програмного коду. IDE зазвичай містить редактор вихідного коду, налагоджувач, компілятор або інтерпретатор та інші інструменти, такі як контроль версій, автоматизація збірки та функції керування проектами. Мета IDE — зробити процес розробки програмного забезпечення ефективнішим і продуктивнішим, надаючи централізований робочий простір, де розробники можуть виконувати всі свої завдання без необхідності перемикатися між різними програмами [40].

Unity підтримує такі IDE: MonoDevelop, JetBrains Rider, Visual Studio Code, Visual Studio [41].

«MonoDevelop»



Рисунок 1.8 — Логотип «MonoDevelop» [42]

MonoDevelop — це IDE з відкритим вихідним кодом, яка постачається в комплекті з Unity. На рисунку 1.8 зображено логотип цієї IDE. Він підтримує кілька мов програмування, включаючи C#, і пропонує такі функції, як підсвічування синтаксису, доповнення коду та налагодження [43].

«JetBrains Rider»



Рисунок 1.9 — Логотип «JetBrains Rider» [44]

JetBrains Rider — це кросплатформне інтегроване середовище розробки (IDE) для розробки додатків .NET, включаючи ASP.NET, .NET Core і Xamarin. На рисунку 1.9 зображено логотип цієї IDE. Він розроблений компанією JetBrains, відомою створенням потужних інструментів розробки, таких як IntelliJ IDEA, PyCharm і ReSharper.

Основні функції JetBrains:

- кросплатформеність. JetBrains Rider працює на Windows, macOS і Linux, що робить його чудовим вибором для розробників, яким потрібно працювати на різних платформах;
- аналіз коду та рефакторинг. Він містить розширені інструменти аналізу коду, які пропонують пропозиції щодо покращення якості коду та виявлення потенційних проблем. Він також містить ряд інструментів рефакторингу коду, які можуть допомогти розробникам покращити структуру та організацію коду;
- інтелектуальне завершення коду. Є функції інтелектуального завершення коду, які можуть допомогти пришвидшити кодування, передбачаючи, який код розробник, ймовірно, напише наступним;

- інтегроване налагодження. Міститься потужний налагоджувач, який можна використовувати для налагодження як локального, так і віддаленого коду, і надає розширені функції, такі як точки зупинки, спостереження та стеки викликів;

- підтримка плагінів. Підтримується широкий спектр плагінів і розширень, які можна використовувати для покращення його функціональності, включаючи плагіни для популярних фреймворків, таких як Unity, React і Angular [45].

«Visual Studio» і «Visual Studio Code»

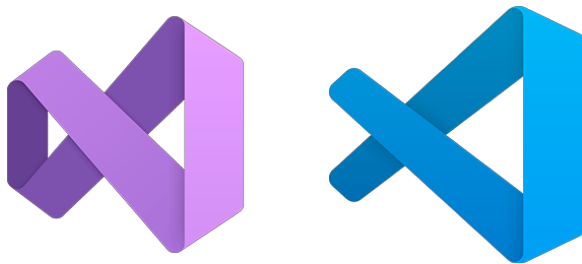


Рисунок 1.10 — Логотипи «Visual Studio» і «Visual Studio Code» відповідно [46]

Visual Studio (VS) і Visual Studio Code (VS Code) — це інтегровані середовища розробки (IDE), розроблені Microsoft, але вони відрізняються своїми функціями та можливостями. На рисунку 1.10 зображено логотипи цих IDE відповідно.

Ось деякі з основних відмінностей між VS Code і Visual Studio:

- призначення. VS Code в основному розроблено як легкий і універсальний редактор коду, який підтримує кілька мов програмування та платформ. З іншого боку, VS — це повнофункціональна IDE, яка орієнтована на створення програм для платформи Microsoft Windows;

- підтримка мов програмування. Хоча обидві IDE підтримують широкий спектр мов програмування, VS більш тісно інтегрована з технологіями Microsoft, такими як C#, .NET і Visual Basic. VS Code, з іншого боку, підтримує більш широкий спектр мов, включаючи Python, JavaScript і Ruby;

- налагодження. Visual Studio включає більш розширені функції налагодження, ніж VS Code, такі як можливість налагодження багатопоточних програм і віддаленого налагодження на інших машинах;

- інтерфейс користувача. VS має більш складний і багатofункціональний інтерфейс користувача, ніж VS Code, з більшою кількістю параметрів і меню для налаштування. VS Code, з іншого боку, має простіший і зручніший інтерфейс користувача;

- розширення. Обидві IDE підтримують розширення, але VS Code має більшу та активнішу екосистему розширень, ніж VS. Розширення VS Code зазвичай невеликі та зосереджені на конкретних завданнях, тоді як розширення VS, як правило, більш комплексні та інтегровані;

- системні вимоги. Visual Studio потребує потужнішого комп'ютера та більше дискового простору, ніж VS Code, через більший набір функцій і складнішу архітектуру.

Отже, VS Code — це легкий і універсальний редактор коду, який підходить для широкого спектра завдань програмування, а Visual Studio — це повнофункціональна IDE, для створення програм Windows за допомогою технологій Microsoft. Обидві мають свої сильні та слабкі сторони [46, 47].

У цьому проекті для написання скриптів, я буду використовувати Visual Studio. Вибір аргументується тим, що я вже знайомий з цією IDE та використовував її в минулому і вже маю великий досвід використання VS для інших проектів. Це допоможе мені працювати ефективніше та результативніше.

1.2 Огляд технічних вимог до мобільних ігор на платформі Android

Ігри для Android, як і будь-яке інше програмне забезпечення, мають певні системні вимоги, які повинні бути виконані для їх належної роботи. Ці вимоги зазвичай визначаються розробниками ігор і можуть відрізнятися залежно від складності гри та її графіки, а також версії Android, на якій працює пристрій.

Загалом системні вимоги до ігор для Android можна розділити на дві основні категорії:

- характеристики мобільного пристрою;
- версія операційної системи Android.

1.2.1 Характеристики мобільного пристрою

Характеристики мобільного пристрою включають такі компоненти:

- процесор. Процесор у вашому пристрої Android є ключовим компонентом, коли мова йде про запуск ігор. Для складніших ігор потрібні потужніші процесори, щоб відповідати вимогам гри. Для деяких ігор може знадобитися певний тип процесора, наприклад ARMv7 або ARMv8;

- оперативна пам'ять. Обсяг оперативної пам'яті (RAM) у вашому пристрої є ще одним важливим фактором. Для безперебійної роботи ігор із високоякісною графікою та складним геймплеєм зазвичай потрібно більше оперативної пам'яті;

- місце для зберігання. Для встановлення та запуску ігор потрібен простір для зберігання на вашому пристрої. Обсяг необхідного простору може відрізнятися залежно від гри, але для багатьох ігор потрібно кілька сотень мегабайт або навіть кілька гігабайтів пам'яті;

- графічний процесор (GPU). Багато ігор значною мірою покладаються на GPU вашого пристрою для відтворення графіки та забезпечення плавного ігрового процесу. Для деяких ігор може знадобитися певний тип GPU, як Adreno, Mali або PowerVR.

Важливо зауважити, що ці вимоги не є абсолютними, і деякі ігри можуть працювати на пристроях, які не відповідають рекомендованим характеристикам. Однак загалом відповідність рекомендованим системним вимогам забезпечить найкращий досвід під час гри.

1.2.2 Версії операційної системи Android

Android — це мобільна операційна система, розроблена компанією Google. Вона була випущена у 2008 році та з того часу стала однією з найпопулярніших операційних систем для мобільних пристроїв, з понад 2,5 мільярдами активних користувачів у всьому світі. Android базується на ядрі Linux і призначений

переважно для мобільних пристроїв із сенсорним екраном, таких як смартфони та планшети. Він також підтримує інші пристрої, такі як розумні годинники, телевізори та автомобілі. Android є платформою з відкритим вихідним кодом, що означає, що її вихідний код є у вільному доступі для будь-кого, хто хоче змінити або поширити його. Однією з ключових особливостей Android є його гнучкість і можливість налаштування. Користувачі Android можуть завантажувати програми з магазину Google Play, який наразі містить понад 3 мільйони програм, або з інших сторонніх магазинів програм. Android також підтримує віджети, які є невеликими програмами, які можна розмістити на головному екрані для швидкого доступу до інформації та функцій. Також він відомий своєю інтеграцією з такими сервісами Google, як Google Search, Gmail, Google Maps і Google Assistant. Ці служби глибоко інтегровані в операційну систему, і до них можна отримати доступ безпосередньо з головного екрана. Що стосується безпеки, Android має вбудовані функції безпеки, такі як Google Play Protect, яка сканує програми на наявність зловмисного програмного забезпечення перед їх завантаженням, і регулярні оновлення системи безпеки від Google. Ця операційна система підтримує шифрування та біометричну автентифікацію, наприклад розпізнавання відбитків пальців і обличчя.

Протягом багатьох років було випущено багато версій Android. Ось найзначніші з них:

- Android 1.0. Це була перша версія Android, випущена у вересні 2008 року. Вона була простою та не мала багатьох функцій, які ми сьогодні сприймаємо як належне, наприклад віртуальної клавіатури;

- Android 1.5 (Cupcake). Ця версія Android була випущена в квітні 2009 року та представила кілька нових функцій, включаючи екранну клавіатуру, запис відео та підтримку Bluetooth стерео;

- Android 2.0/2.1 (Eclair). Eclair був випущений у жовтні 2009 року та представив такі функції, як підтримка кількох облікових записів, підтримка HTML5 у браузері та новий додаток для камери;

- Android 2.2 (Froyo). Froyo було випущено в травні 2010 року та принесло з собою значні покращення продуктивності, включаючи введення компілятора JIT для швидшого виконання програми;

- Android 2.3 (Gingerbread). Gingerbread був випущений у грудні 2010 року та зосереджений на покращенні інтерфейсу користувача та додаванні нових функцій, таких як підтримка NFC та SIP;

- Android 4.0 (Ice Cream Sandwich). Його було випущено в жовтні 2011 року та внесло значні візуальні покращення в інтерфейс Android, а також нову систему багатозадачності та розблокування розпізнавання обличчя;

- Android 4.4 (KitKat). Він був випущений у жовтні 2013 року та представив новий, спрощений дизайн для інтерфейсу Android, а також значні покращення телефонного набору номера та програм обміну повідомленнями;

- Android 5.0/5.1 (Lollipop). Lollipop був випущений у листопаді 2014 року та представив нову мову «Material Design» для Android, а також покращив сповіщення та час автономної роботи;

- Android 6.0 (Marshmallow). Ця версія була випущена у жовтні 2015 року та зосереджений на збільшенні часу роботи батареї з новою функцією «Doze», яка переводила телефон у стан низького енергоспоживання, коли він не використовувався;

- Android 7.0/7.1 (Nougat). Nougat було випущено в серпні 2016 року та представило кілька нових функцій, включаючи режим розділеного екрана, нову систему сповіщень і підтримку платформи Google Daydream VR;

- Android 8.0/8.1 (Oreo). Oreo було випущено в серпні 2017 року та представило низку вдосконалень інтерфейсу Android, включаючи новий режим «картинка в картинці» та підтримку автозаповнення паролів;

- Android 9.0 (Pie). Pie був випущений у серпні 2018 року та представив кілька нових функцій, включаючи навігацію на основі жестів, адаптивну систему батареї та нову інформаційну панель «Цифрового добробуту», щоб допомогти користувачам відстежувати використання свого телефону;

- Android 10. Android 10 було випущено у вересні 2019 року та принесло значні покращення конфіденційності та безпеки, а також нові функції, такі як темний режим для всієї системи та покращена навігація жестами;

- Android 11. Android 11 було випущено у вересні 2020 року та представило кілька нових функцій, зокрема вдосконалені засоби керування медіа, новий розділ «Розмови» на панелі сповіщень і вдосконалення запису екрана;

- Android 12. Android 12 було випущено в жовтні 2021 року та представило суттєво оновлений дизайн інтерфейсу Android, а також кілька нових функцій, зокрема нову панель конфіденційності, оновлений екран блокування та покращення тактильного зворотного зв'язку;

- Android 13. Він був випущений у серпні 2022 року. Android 13 представляє абсолютно новий дизайн інтерфейсу як для планшетів, так і для складних телефонів, з поновленим фокусом на створенні гарного досвіду роботи з великим екраном у самій операційній системі та в програмах. Удосконалення в цій області включають нову структуру та низку вказівок щодо оптимізації додатків, а також більш потужний режим розділеного екрана для багатозадачності та панель завдань у стилі робочого столу ChromeOS, яка полегшує доступ до часто використовуваних програм з будь-якого місця;

- Android 14. Android 14 вийшов на початку лютого 2023 року. Основні зміни спрямовані на покращення продуктивності системи та конфіденційності, а також на продовження роботи з попередньої версії Android, щоб створити більш оптимізовану роботу для всіх, хто використовує планшет або складаний телефон. Також, він містить декілька більш помітних удосконалень на поверхневому рівні — наприклад, нову систему для створення візуальних сповіщень навколо вхідних сповіщень. Google також тестує нову функцію в Android 14, яка дозволить вам «клонувати» програму на своєму пристрої та запускати дві її окремі версії одночасно — це може бути зручним способом використання служби як з особистим обліковим записом, так і з професійним обліковим записом. [48]

У даному проекті, ми будемо розробляти гру з підтримкою версії Android 4.4 KitKat і вище. Такий вибір був зроблений, тому що ми хочемо охопити якомога більшу аудиторію людей.

1.3 Розробка концепції гри, включаючи жанр, цільову аудиторію, геймплей, механіку та інші аспекти

Створення концепту гри є першим і ключовим кроком у розробці мобільної гри. Важливо мати чітке уявлення, про що буде гра, який буде жанр, на яку цільову аудиторію вона спрямована, а також розуміти, які функції та механізми будуть в ній реалізовані. Концепція може розвиватися з часом, але наявність чіткого бачення на початку допоможе забезпечити міцну основу гри.

1.3.1 Жанр

Жанр розроблюваної гри — Аркада.

Аркадні ігри — це популярний жанр мобільних ігор, який існує з перших днів мобільних ігор. Аркадні ігри зазвичай включають швидкий ігровий процес, та просте керування. Деякі популярні приклади аркадних ігор для мобільних пристроїв включають класику, як Pac-Man і Space Invaders, а також новіші ігри, як Temple Run, Subway Surfers, Helix Jump та інші. Аркадні ігри часто розробляються так, щоб їх було легко взяти та грати, а короткі ігрові сеанси ідеально підходять для ігор на ходу. Однією з переваг аркадних ігор як мобільного жанру є їх доступність. У них можна грати на різноманітних пристроях, від смартфонів високого класу до бюджетних пристроїв, і часто вони не вимагають великої обчислювальної потужності апарата чи багато пам'яті. Загалом, аркадні ігри залишаються популярним жанром на мобільних пристроях, надаючи гравцям веселі та захопливі враження, якими можна насолоджуватися будь-коли та будь-де.

1.3.2 Цільова аудиторія

Визначення та розуміння цільової аудиторії має вирішальне значення для успіху будь-якої мобільної гри. Цільова аудиторія — це група людей, яким найімовірніше сподобається та зацікавить гра.

Моя гра орієнтована на широку аудиторію людей і не містить контенту, який би її обмежував. Орієнтований вік гравців може починатися з 5 років та обмежуватися віком 35+ років. Такий низький поріг входу обумовлений тим, що мобільні ігри останнім часом стають все більш популярними серед молоді. Завдяки зручності та доступності мобільних пристроїв грати в ігри в дорозі стало легше, ніж будь-коли. Так само, моя гра буде дуже простою для освоєння та розуміння і не буде містити якісь важкі механіки.

1.3.3 Основна концепція гри

Проект являє собою мобільну гру жанру аркада, яка немає певної мети для перемоги, а призначена для відпочинку від стресу повсякденного життя та приємного проведення часу.

Основна концепція полягає в тому, що у нас є платформа, на якій знаходиться опорний-початковий блок, навколо якого генеруються позиції з 6 можливих для наступного блока. Гравець вибирає потрібну йому позицію та натисканням по екрану розміщує новий блок у цій позиції. Основне завдання гравця це побудувати якомога більшу вежу. Ця вежа має гравітацію, це означає, що при побудові вежі, гравцеві потрібно дотримуватися рівноваги та правильно підбирати позиції для розміщення нових блоків.

Також у цій грі є механіка ракет, яка не дає гравцю повністю розслабитися і додає маленької концентрації. Механіка ракет полягає у наступному. У будь-який момент часу, зверху екрану може вилетіти ракета, яка спрямована у бік вежі та при попаданні цієї ракети у вежу, вона руйнується. Також при побудові гравцем певної кількості блоків, може з'явитися хвиля ракет, яка складається вже з кількох ракет. Гравцю потрібно знищувати ракети натискаючи на них пальцем.

У цій грі немає певної переможної мети. Негласна мета гравця це побудувати якомога більшу вежу, дотримуючись рівноваги та захищаючи її від ракет.

Наступна механіка — це механіка магазину. У якому буде розміщено різні скіни на блоки. Скіни будуть поділені на 2 типи: одні можна буде відкрити за рекорди, а ще одні, можна буде придбати за внутрішньоігрову валюту

У грі буде внутрішньоігрова валюта, яка буде з'являтися під час гри. Гравцеві буде надано можливість збирати цю валюту. Для цього потрібно правильно розставляти блоки, щоб її дістатися.

2 ПРОЕКТУВАННЯ ГРИ

Проектування гри є одним із найважливіших етапів у розробці будь-якої ігрової ідеї. Цей процес включає в себе кілька ключових кроків, таких як вибір архітектури, розробка дизайну та функціональності гри.

Архітектура служить основою, на якій будується вся гра, і визначає, як різні компоненти та системи взаємодіють один з одним. Вона забезпечує ефективне використання ресурсів, плавний ігровий процес, спрощує оновлення та виправлення помилок, а також дає змогу вдосконалюватись у майбутньому.

Розробка та визначення інтерфейсу користувача, об'єктів та інших елементів у мобільній грі є важливими аспектами. Ці елементи відіграють важливу роль у залученні та утриманні гравців, покращенні зручності використання та передачі передбачуваної механіки та естетики гри.

2.1 Розробка архітектури програмного продукту, включаючи структуру проекту, модулі, компоненти та їх взаємодію

У цьому проекті ми будемо використовувати компонентну архітектуру (component-based architecture) Unity. Компонентна архітектура двигуна Unity — це

шаблон проектування, який дозволяє розробникам створювати складні ігрові об'єкти, поєднуючи менші компоненти, які можна використовувати багаторазово. У цій архітектурі ігрові об'єкти складаються з кількох компонентів, кожен із яких представляє певну поведінку чи функцію. Наприклад, ігровий об'єкт, який представляє гравця, може мати компоненти для руху, виявлення зіткнень, здоров'я та зброї. Кожен із цих компонентів має власний код і може бути доданий або видалений з об'єкта гри за потреби.

Приклади компонентів в Unity:

- Transform. Керує позицією, поворотом і масштабом об'єкта в просторі;
- Rigidbody. Додає фізичну поведінку об'єкта, таке як гравітація, зіткнення та інші;
- Collider. Додає колізію до об'єкта для виявлення зіткнень з іншими об'єктами в сцені;
- Script. Дозволяє створювати користувацькі компоненти, написані мовою C#.

Ця архітектура надає кілька переваг для розробки ігор. По-перше, це забезпечує більшу гнучкість і модульність. Розробники можуть легко додавати, видаляти або змінювати компоненти, не зачіпаючи весь ігровий об'єкт, що полегшує ітерацію та експерименти з різними ігровими механіками. По-друге, це заохочує повторне використання коду, оскільки розробники можуть створювати загальні компоненти, які можна використовувати в кількох ігрових об'єктах. Це допомагає скоротити час розробки та підвищити ефективність коду.

В Unity всі об'єкти розташовуються у сценах. Сцени — це контейнери для всіх об'єктів, активів та інших компонентів, які складають певний рівень або область гри.

Деякі основні елементи, які можуть міститися у сценах:

- ігрові об'єкти. Це основні будівельні блоки будь-якої сцени Unity. Ігровий об'єкт — це об'єкт, який має положення, обертання та масштаб і може містити різноманітні компоненти та ресурси, наприклад 3D-моделі, сценарії та інші;

- камери. Камери використовуються для визначення точки зору гравця або перспективи, з якої переглядається сцена. В Unity можна використовувати кілька камер для створення різних видів однієї сцени;

- світло. Світло використовується для освітлення сцени та створення тіней та інших ефектів. Unity підтримує кілька типів світла, включаючи спрямоване світло, точкове світло та прожектори;

- інтерфейс користувача. Систему інтерфейсу користувача (UI) Unity можна використовувати для створення меню, кнопок та інших елементів на екрані, які дозволяють користувачеві взаємодіяти з грою.

Це лише деякі з багатьох елементів, які можна знайти в сцені Unity. Сцени використовуються для створення різних рівнів, меню та інтерфейсів гри. Наприклад, у грі може бути одна сцена для головного меню, інша для першого рівня і третя для другого рівня. Кожну сцену можна спроектувати та сконструювати незалежно, а потім з'єднати разом. Крім того, сцени можна використовувати для тестування та налагодження. Створюючи окрему сцену для певної функції чи функції гри, розробники можуть ізолювати й усувати потенційні проблеми, не впливаючи на решту проекту.

У цьому проекті буде 3 сцени:

- сцена «Меню»;
- основна сцена з ігровим процесом;
- сцена «Магазин».

2.1.1 Сцена «Меню»

Сцена «Меню» слугує так званими воротами гри. Це перше, з чим стикаються гравці, коли запускають гру. Вона задає тон і створює перше враження про естетику, стиль і загальну якість гри. Тому дуже важливо, зробити сцену меню візуально привабливою і цікавою для гравців, щоб викликати захоплення у гравців, коли вони вперше запускають нашу гру. Меню діє як центральний вузол, де гравці можуть легко переміщатися між різними сценами гри, такими як початок гри, доступ до магазину зі скінами та регулювання налаштувань. Це допомагає гравцям швидко знаходити те, що їм потрібно, і підвищує загальну доступність гри.

На рисунку 2.1 показана взаємодія гравця зі сценами. За допомогою меню, гравці зможуть починати гру, переходити у магазин зі шкінами на блоки та регулювати налаштування гри. Для створення привабливості та цікавості до гри, буде розроблена цікава ідея, а саме, у меню буде побудована вежа і навколо цієї вежі буде плавний проліт камери, яка показуватиме всю вежу з кількох боків. А скін блоків, з яких буде побудована ця вежа, буде змінюватися на один з наявних у магазині кожний раз, коли гравці будуть переходити у меню. Це додасть привабливості до гри та також, гравці зможуть побачити, як виглядатиме той чи інший скін у грі.

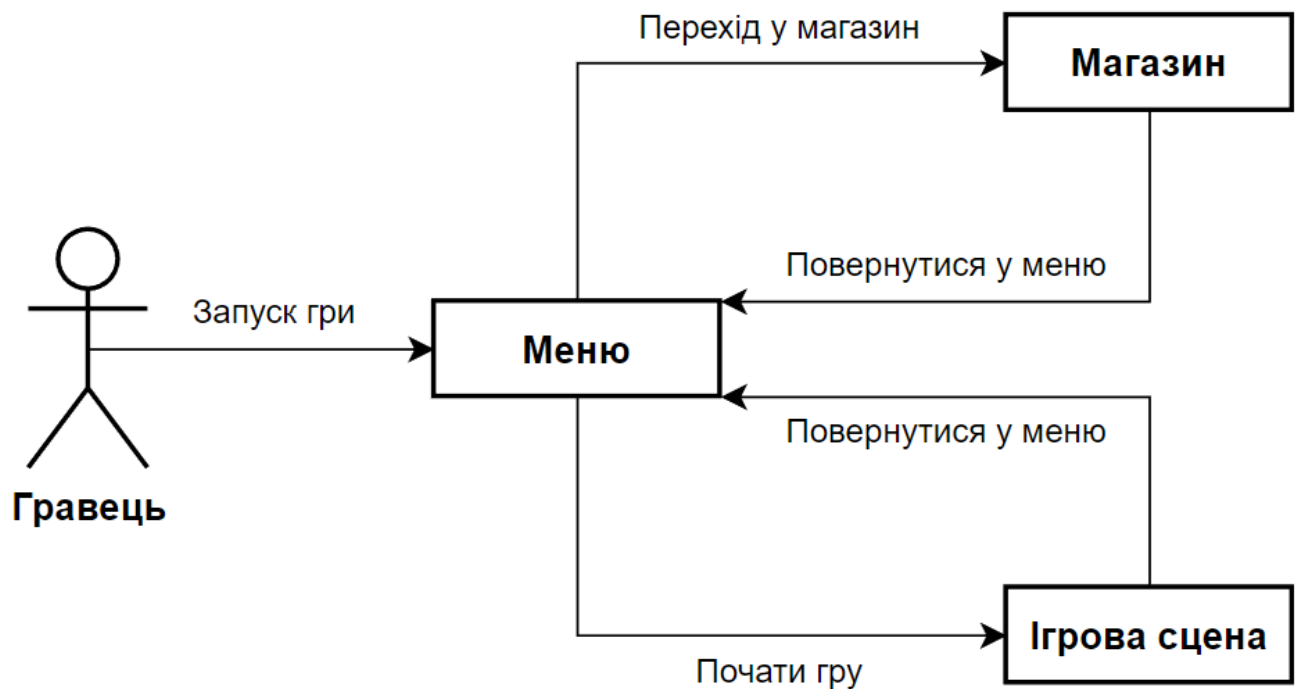


Рисунок 2.1 — Варіанти взаємодії гравця зі сценами

2.1.2 Сцена ігрового процесу

У ігровій сцені, в яку гравець зможе потрапити з меню (див. рисунок 2.1), буде сам геймплей. У цій сцені, у нас буде розташована платформа, на якій буде стояти початковий-опорний блок, з якого гравець буде починати свою гру, навколо якого будуть генеруватися позиції з 6 можливих для наступного блока, гравцю потрібно вибирати потрібну йому позицію та розмішувати новий блок у цій позиції. Для того, щоб розмістити блок, гравцю потрібно буде у будь-якому місці на екрані

тапнути пальцем. Всі ці блоки, які буде розміщувати гравець, поєднуються в одну велику вежу, яка має гравітацію і якщо гравець не буде дотримуватись гравітації та не буде правильно розміщувати блоки, вежа може впасти й гра буде програна. Це перший вид програшу (див. рисунок. 2.2). Також, якщо гравець не правильно розмістить блоки та вийде ситуація, в якій всі позиції навколо крайнього поставленого куба будуть заблоковані, гра також буде програна. Це другий вид програшу (див. рисунок. 2.2). Також, гравець зможе повертати камеру навколо вежі, для того, щоб оцінювати ситуацію і ставити блоки в потрібне місце і збивати ракети, які летять з іншого боку вежі.

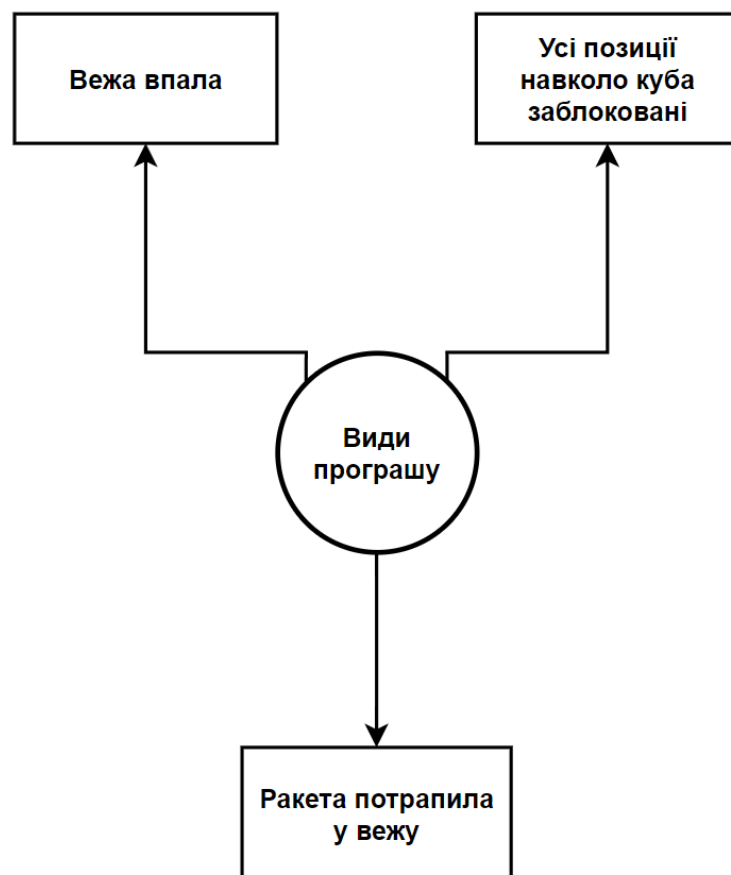


Рисунок 2.2 — Види програшу у грі

Також при старті гри, у деякому радіусі від крайнього поставленого блоку буде з'являтися валюта. У розроблюваній грі, у вигляді валюти виступатиме дорогоцінний камінь, модель якого ми візьмемо в Unity Asset Store [31]. Цей дорогоцінний камінь матиме свій час життя, після закінчення якого він буде

зникати та з'являтиметься новий в іншій позиції. Ці камені будуть мати кілька видів, за кожен з яких гравцеві буде нараховуватися різна кількість валюти. Залежно від кількості блоків, яку поставить гравець під час гри, буде з'являтися більш цінна валюта, за підбір якої гравцеві буде нараховуватися більше. Для того, щоб підібрати це каміння, гравцю потрібно правильно розставляти блоки, щоб дістатися до них. При підборі каміння, гравцю зразу буде нарахована певна кількість валюти, яка відповідає типу підбраного дорогоцінного каміння та також негайно після цього, з'явиться нове каміння в іншій позиції.

Після першого поставленого куба почнуть вилітати ракети. Механіка ракет буде працювати наступним чином. У будь-який момент часу, зверху поза видимістю гравця, будуть з'являтися ракети, які будуть летіти перпендикулярно вниз. При зіткненні цієї ракети з вежею, буде відбуватися вибух і вежа руйнуватиметься та гравець буде програвати. Це третій вид програшу у грі (див. рисунок. 2.2). Також при побудові гравцем певної кількості блоків, може з'явитися хвиля ракет, під час якої одночасно вилітатиме відразу 3 ракети. Гравцю потрібно захищати вежу та знищувати ці ракети. Для того, щоб знищувати ракети, потрібно натискати на ракету пальцем.

2.1.3 Сцена «Магазин»

У сцені «Магазин», в яку гравець також зможе потрапити з меню (див. рисунок 2.1), будуть розміщені скіни на блоки. Скіни будуть поділені на 3 типи:

- скіни які будуть відкриватися за рекорд (кількість поставлених блоків);
- скіни які будуть відкриватися за внутрішньоігрову валюту;
- скіни-прапори, які також будуть відкриватися за внутрішньоігрову валюту.

Гравець зможе купувати скіни які йому подобаються, і на які йому достатньо валюти, яка буде відображатися зверху магазину. Для вибору скіна, гравцю всього-на-всього треба натиснути на бажаний скін і він буде вибраний та буде використовуватися у грі. Якщо гравець забажає вибрати інший, або купити новий, все що треба гравцю, це знову перейти у магазин і вибрати інший клацнувши на нього пальцем. Якщо це скін, який відкривається за рекорд, гравцеві просто потрібно буде натиснути на нього, для вибору. Якщо це скін, який відкривається за

внутрішньоігрову валюту, гравцеві потрібно натиснути на бажаний скін і підтвердити покупку, натиснувши "Так" або "Ні". Одразу після підтвердження покупки, скін автоматично буде обраний як основний і буде використовуватися у грі.

2.2 Проектування дизайну гри, включаючи визначення інтерфейсу користувача, об'єктів та інших елементів гри

Проектування дизайну та інтерфейсу користувача (UI) є важливим аспектом розробки мобільної гри. Дизайн та інтерфейс гри є першим, з чим стикаються користувачі під час запуску гри. Якісний та привабливий дизайн допомагає створити позитивне перше враження та зацікавити користувачів. Якщо користувач побачить неакуратний або заплутаний інтерфейс, це може негативно вплинути на його думку про якість гри. Хороший дизайн і інтерфейс допомагають гравцям швидко орієнтуватися в грі та розуміти, як використовувати різні елементи інтерфейсу. Інтуїтивна навігація та зрозумілі іконки скорочують час, який потрібно користувачеві для освоєння гри, та роблять її привабливішою для широкої аудиторії.

2.2.1 Проектування дизайну та інтерфейсу сцени «Меню»

На рисунку 2.3 зображено прототип інтерфейсу сцени меню. Тут будуть розташовані 4 кнопки, а саме, кнопка налаштувань, кнопка «Почати гру», кнопка переходу в магазин і кнопка «Про гру». Також зверху буде розташовано так зване лого гри, а саме її назва. У майбутньому ми додамо анімацію до цього лого, щоб додати більше динаміки до нашого меню.

При натисканні кнопки налаштування (див. рисунок 2.3), буде відкриватися вікно з налаштуваннями гучності, прототип якого показано на рисунку 2.4. Тут гравець за допомогою слайдерів, зможе окремо налаштовувати гучність музики та звукових ефектів, як йому забажається.

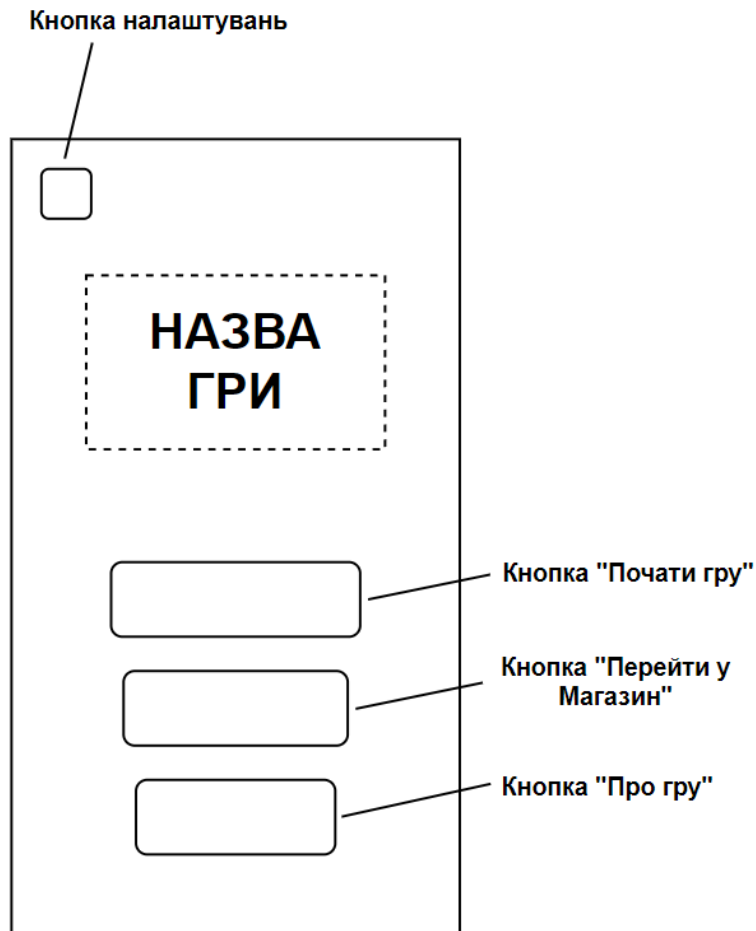


Рисунок 2.3 — Прототип інтерфейсу сцени Меню

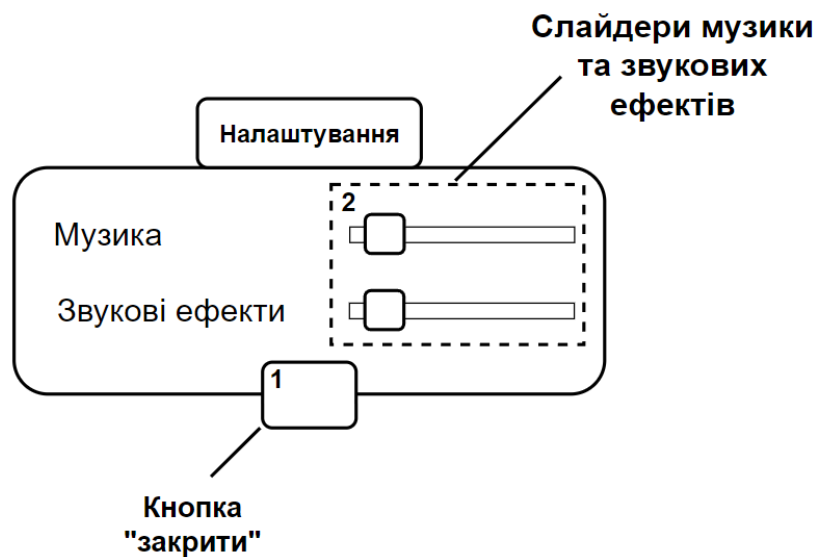


Рисунок 2.4 — Прототип вікна налаштувань

При натисканні кнопки «Про гру», (див. рисунок 2.3), буде відкриватися вікно з інформацією про гру, прототип якого зображено на рисунку 2.5. Тут буде

інформація про розробника та автора музики, яка буде використовуватися у розроблюваній грі.

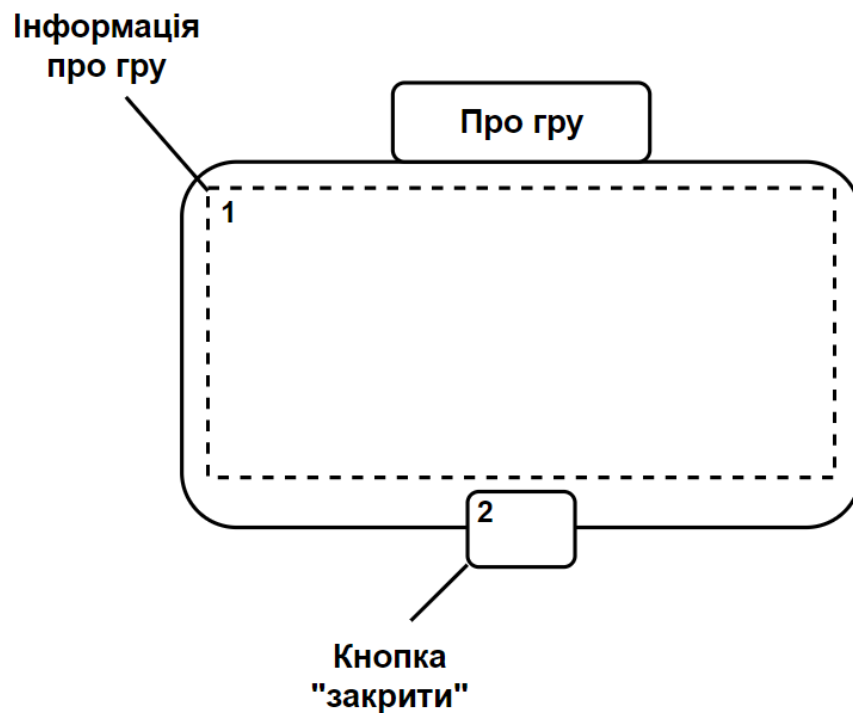


Рисунок 2.5 — Прототип вікна налаштувань

Кнопки «Почати гру» та «Магазин» будуть переносити гравця у ігрову сцену та сцену з магазином відповідно.

2.2.2 Проектування дизайну та інтерфейсу у ігровій сцені

На рисунку 2.6 зображено прототип інтерфейсу основної сцени. У верхньому лівому кутку буде розміщена кнопка паузи гри. У верхньому правому куті буде розміщено 2 контейнери. У верхньому контейнері буде відображатися поточний рахунок гравця та найкращий рахунок гравця. А у нижньому контейнері буде показуватися кількість зібраної ігрової валюти в поточній грі. Ігрове поле (див. рисунок 2.6) буде являти собою основний ігровий простір гравця у якому він керуватиме ігровим процесом, а саме повертати камеру свайпами, ставити блоки та збивати ракети.

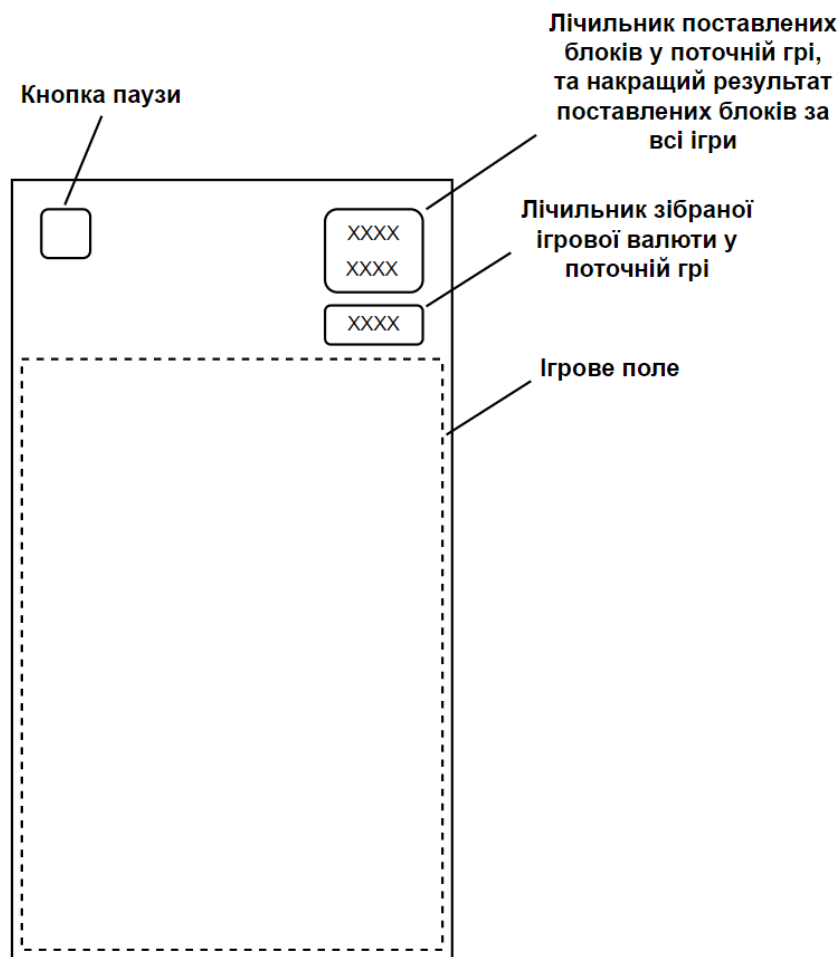


Рисунок 2.6 — Прототип інтерфейсу ігрової сцени

Кнопка паузи слугує зупинкою гри, при натисканні на яку, буде зупинятися весь ігровий процес та відкриватися вікно, яке зображено на рисунку 2.7.

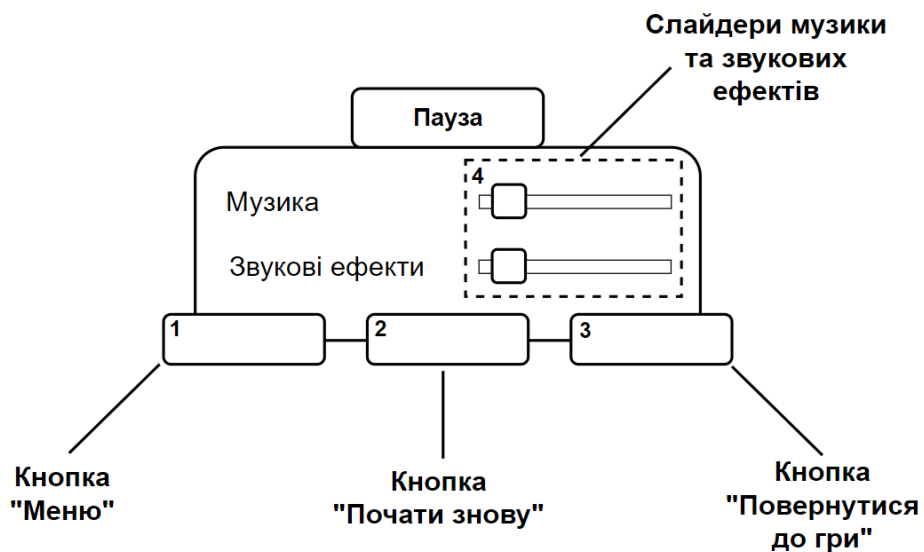


Рисунок 2.7 — Прототип вікна паузи

У цьому вікні (див. рисунок 2.7), гравець зможе підкоригувати гучність музики та звукових ефектів прямо під час гри. Також тут міститься 3 кнопки: «Меню», «Почати знову», «Повернутися до гри». За допомогою яких, гравець може покинути ігровий процес і перейти в меню, перезапустити гру, повернутися до гри відповідно.

Одразу після програшу з'являтиметься вікно з програшем, прототип якого зображено на рисунку 2.8.

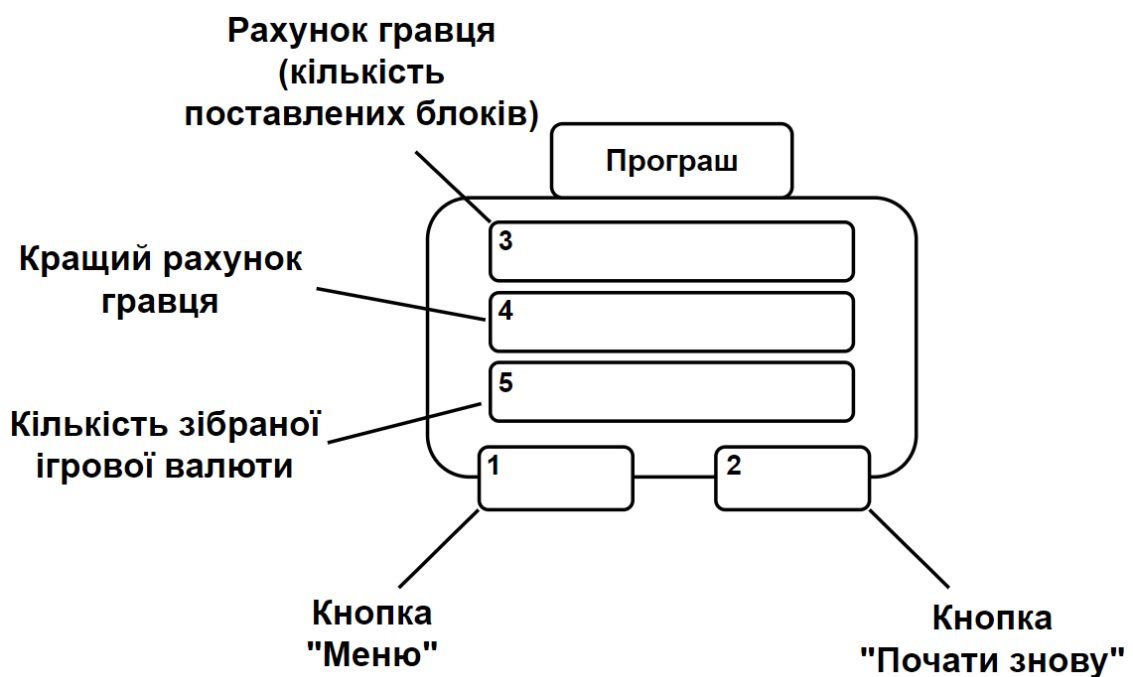


Рисунок 2.8 — Прототип вікна після програшу

У цьому вікні буде основна інформація про зіграну гру. В контейнері, який позначений номером 3 (див. рисунок 2.8) буде показаний рахунок гравця, який він набрав у грі, у контейнері нижче буде показано кращий рахунок гравця і найнижчий контейнер міститиме інформацію про кількість зібраної валюти в крайній грі. Це вікно міститиме дві кнопки. Кнопка, яка розташована зліва буде вести гравця до меню, а за допомогою кнопки справа, яка позначена номером 2 (див. рисунок 2.8) можна почати гру заново.

2.2.3 Проектування дизайну та інтерфейсу у сцені магазину

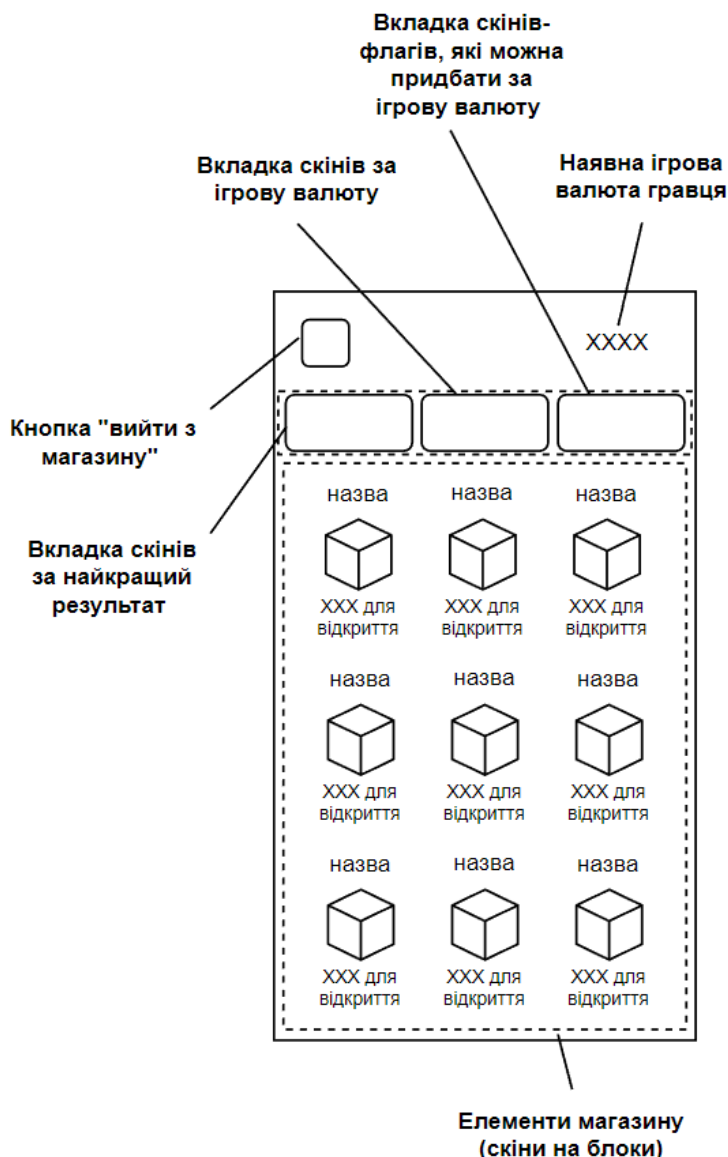


Рисунок 2.9 — Прототип інтерфейсу у магазині

На рисунку 2.9 зображено прототип інтерфейсу магазину. У лівому верхньому кутку розміщена кнопка виходу з магазину, яка перемістить гравця у меню. У правому верхньому кутку буде відображена наявна валюта гравця. Нижче буде розташована панель вкладок, у якій буде розміщено 3 вкладки. За допомогою цих вкладок, гравець зможе швидко відкрити потрібну йому вкладку з певним типом скінів, який йому потрібен. А саме, в першій вкладці буде розташовано скіни за рекорди, у другій вкладці буде розташовано скіни за ігрову валюту, а у третій

вкладці буде розташовано скіни-прапори які теж можна буде придбати за ігрову валюту.

Під час купівлі скінів за ігрову валюту, буде відкриватися вікно з підтвердженням купівлі, прототип якого зображено на малюнку 2.10.

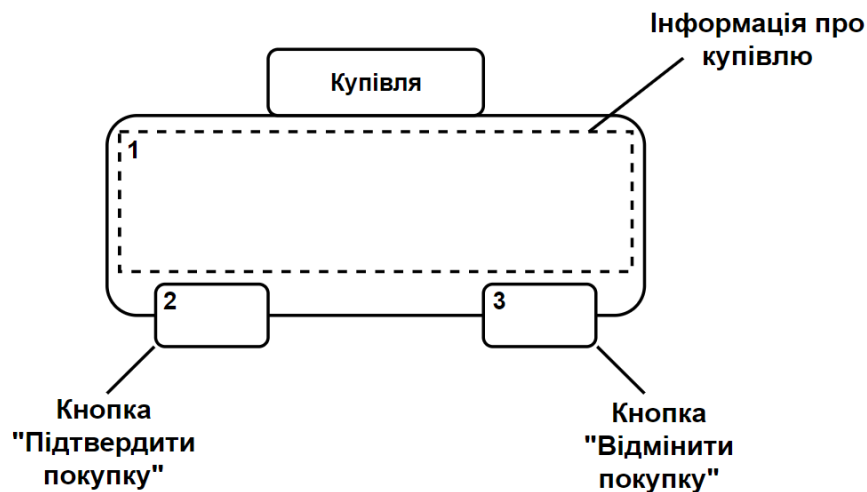


Рисунок 2.10 — Прототип вікна для підтвердження покупки

У полі яке позначено номером 1 (див. рисунок 2.10) буде розміщена інформація про купівлю, а саме назва скіна який купує гравець і його ціна. Гравець зможе підтвердити покупку натиснувши кнопку яка позначена номером 2, або скасувати покупку натиснувши кнопку яка позначена номером 3.

3 РЕАЛІЗАЦІЯ ГРИ

Після завершення аналізу і проектування архітектури, дизайну та інтерфейсу гри, настав час перейти до реалізації гри, а саме реалізувати спроектований інтерфейс та дизайн гри, реалізувати геймплей, механіки гри, ефекти, анімації та інше. А також розробити програмний код гри відповідно до вибраних технологій та протестувати гру.

3.1 Реалізація інтерфейсу та дизайну гри, реалізація геймплею, механік гри, взаємодії гравця з грою, реалізація звукового дизайну, ефектів та іншого

3.1.1 Реалізація дизайну та інтерфейсу користувача

Користувацький інтерфейс реалізовано за допомогою системи Unity UI, яка дає змогу створювати та налаштовувати користувацькі інтерфейси прямо в редакторі. Усі об'єкти UI є ігровими об'єктами, але розташовуються вони в площині прямокутника камери, вони містять у собі компонент RectTransform, який відповідає за положення і масштабування об'єкта в прямокутній площині камери. Всі ці об'єкти є дочірніми по відношенню до основного об'єкта Canvas. В Unity Engine об'єкт Canvas є фундаментальним компонентом системи інтерфейсу користувача (UI). Він слугує контейнером для елементів інтерфейсу, таких як кнопки, текстові поля, зображення та панелі, дозволяючи створювати інтерактивні та візуально привабливі інтерфейси для ваших ігор чи додатків.

Основні ключові моменти об'єкту Canvas в Unity:

- ієрархія. Canvas — це спеціальний тип GameObject, який існує в ієрархії сцени. Він діє як кореневий елемент для всіх елементів інтерфейсу користувача, що містяться в ньому;
- порядок сортування. У певному режимі зображування Canvas може мати певний порядок сортування, який визначає пріоритет його зображування. Вищі порядки сортування візуалізуються поверх нижчих порядків сортування;
- режим масштабування інтерфейсу. Canvas може мати різні режими масштабування інтерфейсу, які керують його масштабуванням при різних роздільних здатностях екрана;
- система подій. Щоб увімкнути інтерактивність інтерфейсу, Canvas потребує компонента Event System. Система подій обробляє події введення, такі як натискання кнопок, переміщення миші та введення з клавіатури;
- трансляція променів. Canvas обробляє трансляцію променів, яка є процесом виявлення взаємодії користувача з елементами інтерфейсу. Рейкасти

використовуються, щоб визначити, чи натиснув користувач на елемент інтерфейсу, і забезпечити відповідну реакцію.

Основні види об'єктів інтерфейсу, які були використані в реалізації користувацького інтерфейсу:

- Panel — це базовий контейнер інтерфейсу, який може містити інші елементи інтерфейсу. Вона часто використовується як фоновий або групувальний елемент для інших компонентів інтерфейсу;

- Text — дозволяє відображати текст у інтерфейсі. Можна встановити шрифт, розмір, колір, вирівнювання та інші властивості тексту;

- Image — використовується для відображення 2D-зображень у інтерфейсі. Можна призначити спрайт або текстуру для компонента Image;

- Button — це інтерактивні елементи інтерфейсу, які реагують на кліки або дотики користувача;

- Slider — це елементи інтерфейсу, які дозволяють користувачам вибирати значення з діапазону, перетягуючи руку вздовж доріжки. Вони зазвичай використовуються для таких налаштувань, як регулювання гучності або налаштування атрибутів символів.

Для створення спрайтів інтерфейсу (кнопки, вікна, тощо) буде використано програму Photoshop. Photoshop — це популярна і потужна програма, яка широко використовується для редагування та маніпулювання зображеннями. Розроблена компанією Adobe Systems, вона пропонує широкий спектр функцій та інструментів, які дозволяють користувачам покращувати, змінювати та трансформувати цифрові зображення у творчий спосіб. За допомогою Photoshop користувачі можуть виконувати базові завдання, такі як обрізання, зміна розміру, регулювання яскравості та контрастності зображень. Він також надає розширені інструменти для ретушування та реставрації фотографій, дозволяючи користувачам легко видаляти плями, зморшки або небажані об'єкти з зображень. Він широко використовується для створення графічного дизайну. Він пропонує різноманітні інструменти для малювання, включаючи пензлі, ручки та фігури, що дозволяють художникам і дизайнерам створювати чудові ілюстрації, логотипи та веб-графіку. Програма

підтримує широкий спектр форматів файлів, що робить її сумісною з іншими дизайнерськими та видавничими програмами. Photoshop став галузевим стандартом для редагування зображень, широко використовуваним фотографами, дизайнерами та творчими професіоналами. Завдяки своїй універсальності та широкому набору інструментів він підходить для різних сфер застосування, включаючи друковані видання, веб-дизайн, цифрову фотографію та виробництво мультимедійних матеріалів.

На рисунку 3.1 зображена палітра кольорів, яка була вибрана для цього проекту.

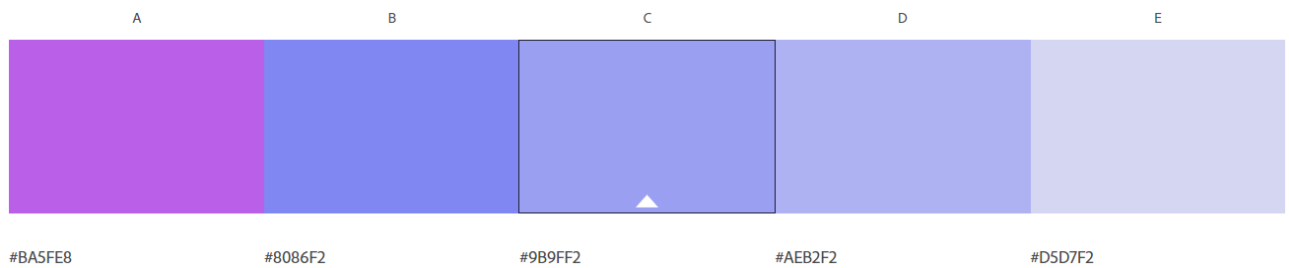


Рисунок 3.1 — Палітра кольорів [49]

На рисунках 3.2, 3.3, 3.4, зображені декілька прикладів створених спрайтів інтерфейсу.



Рисунок 3.2 — Спрайт основних кнопок у сцені меню

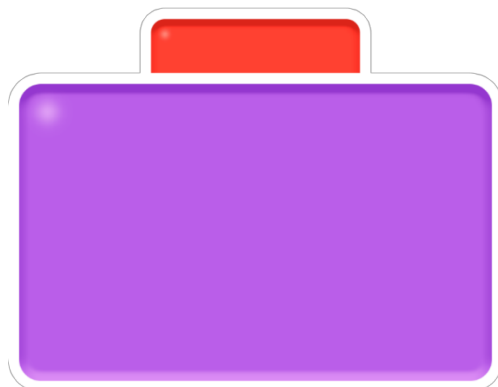


Рисунок 3.3 — Спрайт вікна «Програш» основної сцени



Рисунок 3.4 — Спрайт вікна «Налаштування» сцени меню

На рисунку 3.5 показаний готовий інтерфейс сцени меню.

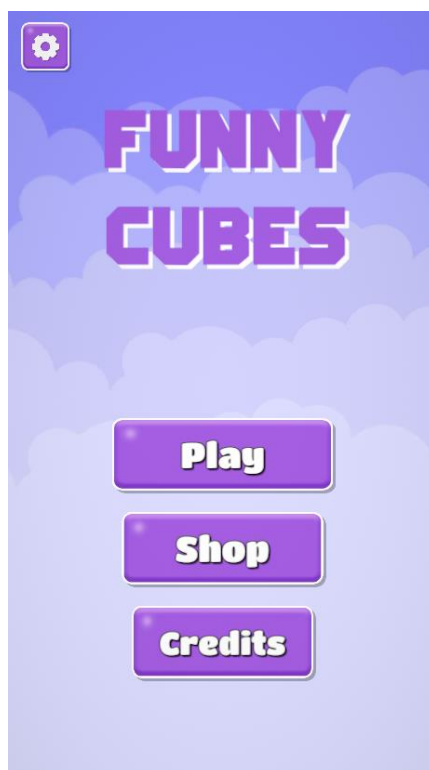


Рисунок 3.5 — Готовий інтерфейс сцени меню

На рисунку 3.6 показане готове вікно з налаштуваннями.



Рисунок 3.6 — Реалізоване вікно налаштувань

На рисунку 3.7 показаний готовий інтерфейс головної сцени.



Рисунок 3.7 — Готовий інтерфейс ігрової сцени

На рисунках 3.8 і 3.9 показані реалізовані вікна паузи і програшу.



Рисунок 3.8 — Реалізоване вікно паузи



Рисунок 3.9 — Реалізоване вікно програшу

На рисунку 3.10 зображений готовий інтерфейс магазину.

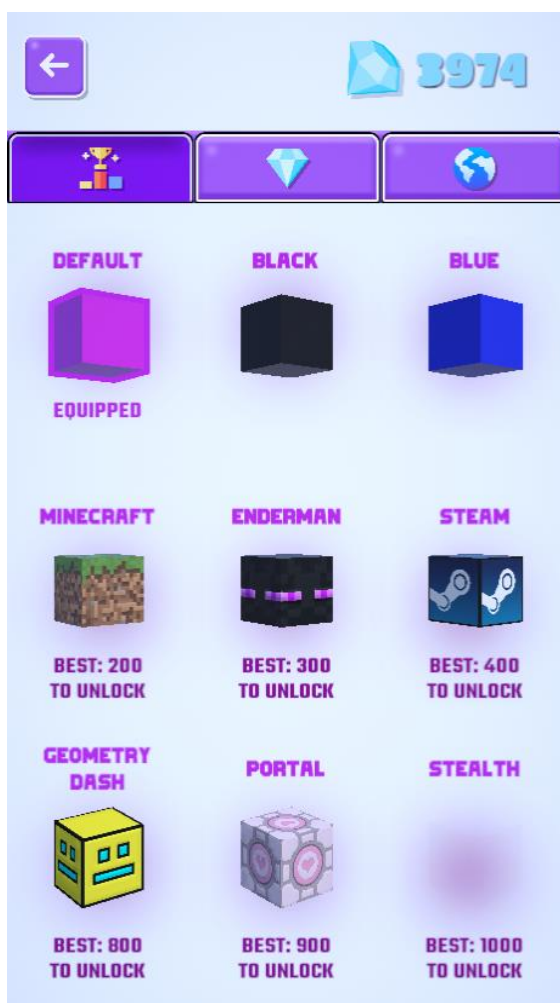


Рисунок 3.10 — Готовий інтерфейс магазину

На рисунку 3.11 зображене готове вікно підтвердження покупки.



Рисунок 3.11 — Реалізоване вікно підтвердження покупки

Іконки, які були використані в нашому інтерфейсі, для надання більшої привабливості, були знайдені на сайті безкоштовних іюкнок [50].

Для того, щоб розробити скіни кубів для магазину, нам треба було зробити розгортку моделі, поверх якої ми потім малювали текстури для наших скінів. Для отримання розгортки звичайного куба, була використана програма Blender.

Blender — це універсальне та потужне програмне забезпечення з відкритим вихідним кодом для створення тривимірних зображень. Його широко використовують художники, дизайнери та аніматори для різних цілей, таких як створення візуальних ефектів, 3D-моделей, анімації та інтерактивних 3D-додатків. Однією з ключових особливостей Blender є його повний набір інструментів моделювання, які дозволяють користувачам створювати складні 3D-об'єкти з точністю і легкістю. Він підтримує полігональне моделювання, NURBS-поверхні та поверхні розбиття, забезпечуючи гнучкість у створенні як органічних, так і механічних форм [51].

На рисунку 3.12 ви можете побачити розгортку звичайного куба, яка була отримана за допомогою Blender, поверх якої ми потім малювали текстури для скінів.

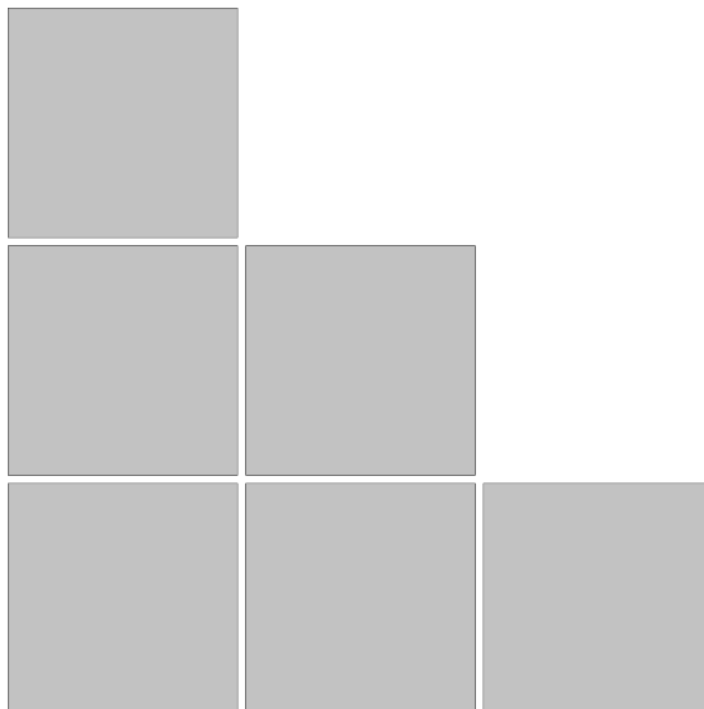


Рисунок 3.12 — Розгортка моделі звичайного куба

Приклад розробленої текстури для шкіни, яка була розроблена на основі розгортки, зображена на рисунку 3.13.

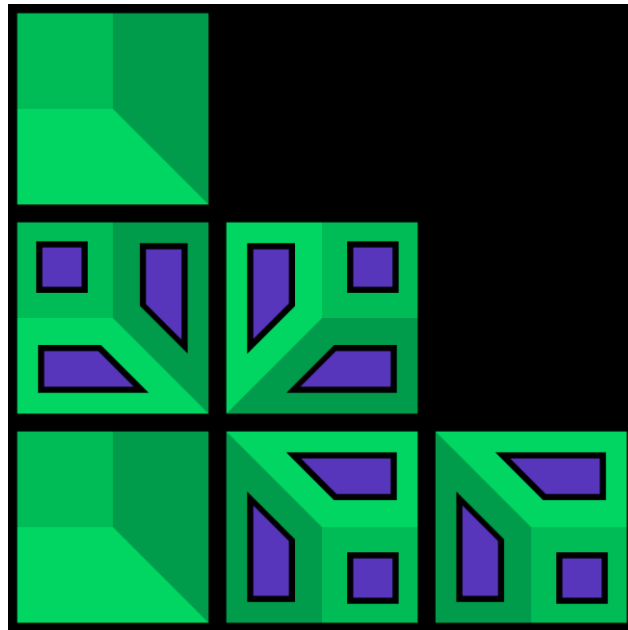


Рисунок 3.13 — Текстура для шкіни

Для того, щоб розробити шкіну, нам треба було імпортувати модель звичайного куба, розгортку якого ми зробили вище (див. рисунок 3.13) з програми Blender до Unity, щоб потім створити матеріал з нашою текстурою і застосувати цей матеріал до імпортованої моделі куба. В Unity матеріал — це тип ресурсу, який визначає візуальний вигляд об'єктів у 3D-сцені. Він контролює взаємодію світла з поверхнею об'єкта, включаючи такі властивості, як колір, текстура, прозорість, відбивна здатність тощо. Матеріали необхідні для створення реалістичної та візуально привабливої графіки у Unity. Unity може працювати з різними форматами 3D-моделей, такими як FBX, OBJ, MAX, Blender, та інші. Можна імпортувати 3D-моделі в Unity і використовувати їх для створення об'єктів, персонажів, довкілля та інших елементів гри.

У результаті, після створення розгортки, далі створення текстури на основі цієї розгортки, імпортування моделі в Unity, створення матеріалу на основі текстури та застосування цього матеріалу на імпортовану модель було розроблено шкіну, який зображений на рисунку 3.14.

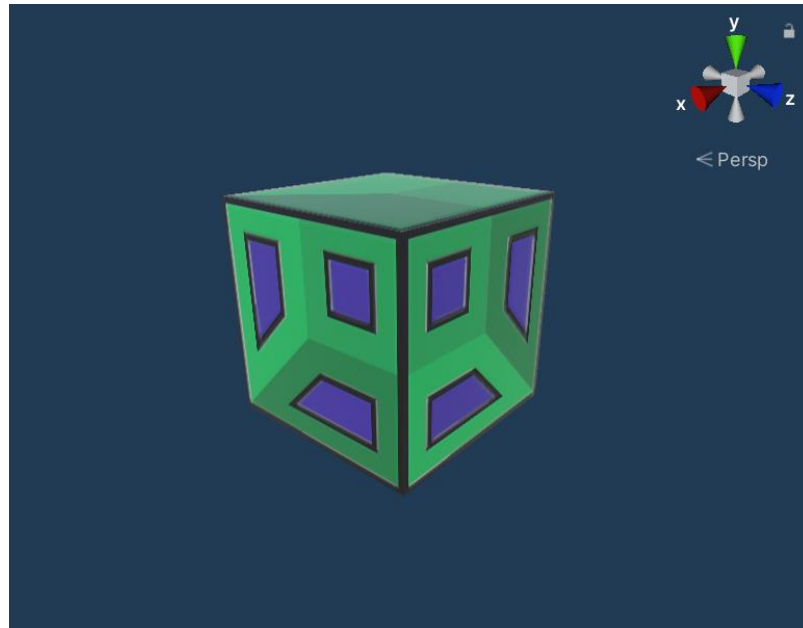


Рисунок 3.14 — Готовий скін у Unity

Далі з цього куба буде створено префаб. В Unity префаб це багаторазово використовуваний шаблон для створення екземплярів GameObjects. Він дозволяє створювати, налаштовувати та зберігати GameObject разом з пов'язаними з ним компонентами, властивостями та дочірніми об'єктами як єдиний ресурс. Заготовки забезпечують зручний спосіб керування та оновлення GameObject'ів у кількох сценах або екземплярах у межах однієї сцени. Модифікація префабу автоматично оновлює всі екземпляри цього префабу, дозволяючи вам внести зміни один раз і відобразити їх у всіх об'єктах, що використовують цей префаб. Ця функція робить префаби особливо корисними для підтримання узгодженості та ефективного керування великомасштабними проектами.

Точно такі самі дії було виконано для всіх інших скінів. У підсумку було розроблено 27 скінів. По 9 скінів для кожного типу.

3.1.2 Розробка анімацій

В Unity анімація — це процес створення та керування рухом і поведінкою об'єктів, персонажів та інших ігрових елементів у часі. Анімація додає життя та інтерактивності проектам Unity, дозволяючи визначати та встановлювати послідовність різних дій, перетворень та візуальних ефектів. Unity надає потужну систему анімації, яка дозволяє створювати як прості, так і складні анімації.

Основні компоненти, що використовуються в анімації Unity:

- компонент Animator відповідає за керування та змішування різних станів анімації. Він керує відтворенням анімації та переходами між ними на основі визначених умов;

- контролер анімації — це машина станів, яка визначає, як анімація відтворюється і змішується на основі вхідних умов і тригерів. Це дозволяє створювати складну логіку анімації та контролювати потік анімації у відповідь на різні події;

- анімаційний кліп — це багаторазовий ресурс, який містить дані ключових кадрів, що визначають рух або поведінку об'єкта в часі. Він представляє певну анімацію, наприклад, ходьбу, стрибки або атаку, і може бути приєднаний до аніматора;

- ключові кадри — це певні точки в анімації, де визначаються значення певних властивостей, таких як положення, обертання або масштаб. Unity інтерполює між ключовими кадрами, щоб створити плавний рух між ними.

За допомогою системи анімації Unity ви можете анімувати різні властивості ігрових об'єктів, такі як положення, обертання, масштаб, колір та властивості матеріалів. Ви також можете керувати анімацією за допомогою сценаріїв, що дозволяє створювати динамічні та інтерактивні анімації, засновані на подіях ігрового процесу та користувацькому введенні.

Для цього проекту було створено декілька анімацій: для логотипа у меню, анімація появи вікон у всіх сценах, проліт камери навколо вежі у меню, анімація підбору валюти та анімація розміщення куба в ігровій сцені.

Приклад анімації «пульсація» для логотипа гри показаний на рисунку 3.15.

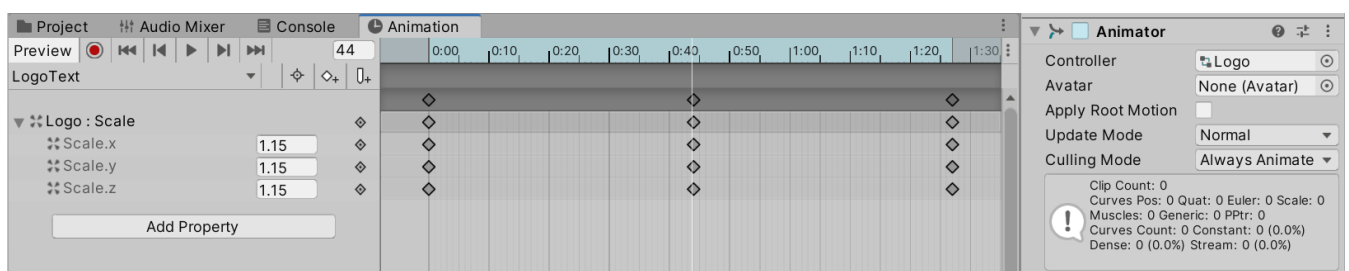


Рисунок 3.15 — Розроблена анімація для логотипу гри у меню

3.1.3 Розробка ефектів

В Unity, у більшості випадків, для створення найрізноманітніших ефектів, використовується вбудована система частинок Unity.

Система частинок (Particle system) в Unity — це потужний інструмент для створення та імітації різноманітних візуальних ефектів, таких як вогонь, дим, вибухи, іскри тощо. Вона дозволяє розробникам генерувати та керувати великою кількістю маленьких рухомих графічних елементів, які називаються частинками. Система частинок розроблена таким чином, щоб її можна було легко налаштовувати та ефективно використовувати. Вона надає ряд параметрів та опцій, які дозволяють контролювати появу, поведінку та загальний вигляд частинок. Деякі ключові аспекти та особливості системи частинок в Unity:

- емісія частинок. Ви можете визначити швидкість і характер емісії частинок, наприклад, безперервну емісію, сплески або контрольовану емісію, виходячи з конкретних умов;

- форма частинок. Unity підтримує різноманітні форми частинок, включно з точками, білбордами (2D спрайтами, що завжди обернені до камери), сітками та користувацькими формами;

- час життя частинок. Кожна частинка має тривалість життя, яка визначає, як довго вона залишається видимою, перш ніж буде знищена. Ви можете вказати тривалість або використовувати випадкові значення в заданому діапазоні;

- розмір і колір частинок. Ви можете керувати розміром і кольором частинок протягом їхнього життя. Це дозволяє застосовувати такі ефекти, як вицвітання, масштабування та зміна кольору, коли частинки старіють або рухаються;

- сили та прискорення. Unity дозволяє застосовувати сили та прискорення до частинок, впливаючи на їхній рух та поведінку. Це дозволяє створювати такі ефекти, як гравітація, вітер та турбулентність;

- зіткнення та взаємодія. Частинки можуть взаємодіяти з іншими об'єктами в сцені, наприклад, колайдерами або поверхнями. Ви можете визначити поведінку частинок, коли вони зіштовхуються, відскакують або прилипають до поверхонь;

- текстури та анімація. Ви можете застосовувати текстури до частинок, щоб надати їм певного вигляду, і навіть анімувати текстури з часом для створення більш динамічних ефектів;

- доріжки та стрічки. Unity підтримує створення шлейфів та стрічок частинок, які є корисними для ефектів, таких як дим або смуги світла, що тягнуться за ними;

- звукові ефекти. Системи частинок можна синхронізувати з джерелами звуку, щоб викликати звуки в певні моменти, посилюючи загальний ефект занурення;

- написання сценаріїв та керування. Unity надає API для написання сценаріїв, що дозволяє динамічно керувати системою частинок. Ви можете змінювати параметри, вмикати чи вимикати емісію та програмно маніпулювати частинками для створення складних та інтерактивних ефектів [52].

У цьому проекті було створено декілька ефектів. Ефект розміщення куба, який також використовується для вибору шкіни в магазині, ракетний слід, ефект вибуху під час потрапляння ракети у вежу і під час знищення ракети тапом та інші.

Приклад створеного сліду ракети, за допомогою системи частинок Unity показаний на рисунку 3.16.



Рисунок 3.16 — Розроблений ефект сліду ракети

3.1.4 Розробка звукового дизайну

Звуковий дизайн в Unity передбачає створення та впровадження звукових елементів для покращення загального ігрового досвіду. Він включає створення, редагування та розміщення звуків, а також керування їх відтворенням та інтеграцію з іншими ігровими системами. Unity надає декілька функцій та інструментів для

полегшення звукового дизайну, що дозволяє розробникам створювати захоплююче та інтерактивне звукове середовище.

Ключові аспекти звукового дизайну в Unity:

- імпорт аудіо. Unity підтримує різні аудіо формати, такі як WAV, MP3 та OGG. Ви можете імпортувати аудіофайли до вашого проекту та організувати їх у редакторі Unity. Крім того, Unity дозволяє налаштовувати такі параметри, як стиснення, потокове передавання та налаштування 3D-звуку, щоб оптимізувати якість звуку;

- джерела звуку. В Unity ви розміщуєте джерела звуку у вашій сцені для випромінювання звуку. Джерело звуку представляє собою об'єкт, що виробляє звук і може бути приєднаний до будь-якого ігрового об'єкту на сцені. Кожне джерело звуку може мати такі властивості, як гучність, висота тону, просторове змішування та призначені до нього аудіокліпи;

- аудіокліпи. Аудіокліп — це фрагмент попередньо записаного або синтезованого звуку, який може бути відтворений джерелом звуку. Unity підтримує як 2D, так і 3D аудіокліпи. Ви можете створювати та редагувати аудіокліпи безпосередньо у Unity або імпортувати їх із зовнішнього програмного забезпечення;

- 3D аудіо та просторова візуалізація. Unity пропонує розширені можливості просторового звуку, що дозволяють створювати реалістичні 3D звукові ландшафти. Увімкнувши просторове відтворення, звуки здаватимуться такими, що походять з певних точок у тривимірному просторі відносно позиції гравця. Це створює відчуття глибини та занурення в ігровий світ;

- аудіо мікшери. Unity надає систему аудіо мікшерів, яка дозволяє вам контролювати та налаштовувати рівні звуку та ефекти у вашій грі. За допомогою мікшера ви можете створювати складні аудіоієрархії, застосовувати ефекти та керувати аудіогрупами незалежно. Аудіомікшери забезпечують гнучкий та потужний спосіб формування загального звукового супроводу;

- аудіоефекти. Unity підтримує різноманітні аудіо ефекти, такі як фільтри, реверберації та модулятори. Ви можете застосувати ці ефекти до окремих джерел

звуку або глобально за допомогою аудіо мікшера. Аудіоефекти можна використовувати для покращення певних звуків, створення атмосфери або додавання особливої поведінки звуку;

- сценарії та події. Можливості скриптів у Unity дозволяють динамічно керувати відтворенням звуку та створювати інтерактивні звукові системи. Ви можете використовувати скрипти, щоб запускати звуки на основі ігрових подій, змінювати властивості джерел звуку в реальному часі або створювати адаптивні аудіосистеми, які реагують на дії гравця;

- сховища ресурсів та плагіни. Магазин ресурсів Unity пропонує широкий спектр ресурсів, пов'язаних з аудіо, включаючи музичні доріжки, звукові ефекти та інструменти для процедурної генерації звуку. Ви також можете знайти плагіни та проміжне програмне забезпечення, які надають додаткову аудіо-функціональність або інтегруються із зовнішніми аудіо-інструментами та бібліотеками.

Для створення загальної системи звуків та музики, був створений Аудіомікшер. За допомогою аудіомікшера можна налаштувати гучність різних звукових джерел у грі. У самому мікшері було створено дві групи: «Музика» та «Звукові ефекти» для того, щоб гравець міг через налаштування, окремо регулювати гучність музики та звукових ефектів, це показано на рисунку 3.17.

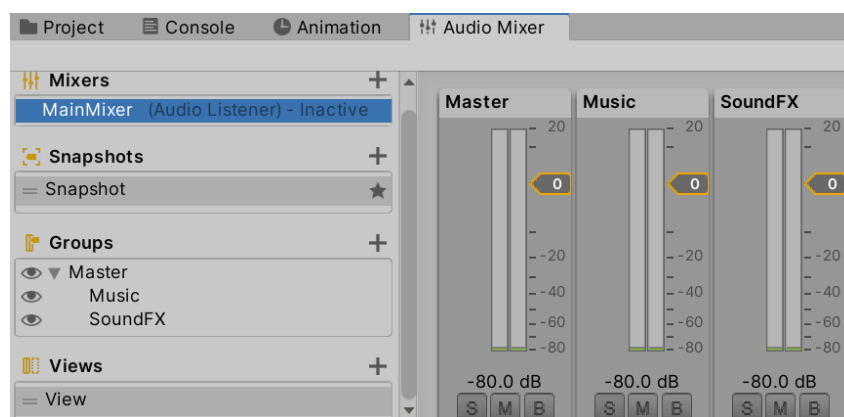


Рисунок 3.17 — Створений аудіомікшер з групами «Музика» та «Звукові ефекти»

До кожного об'єкта, який повинен відтворювати звук, був прикріплений компонент AudioSource, який дає можливість відтворювати звуки у тривимірному середовищі. У властивість AudioClip передається посилання на звуковий фрагмент,

який буде програватися під час виклику методу `Play()` компонента `AudioSource()` у скрипті в майбутньому. А у властивість `Output` передається посилання на відповідну групу аудіомікшера, який буде регулювати гучність цього звукового фрагмента. Наприклад, для кнопки налаштувань у меню ми виконали попередні дії, щоб у майбутньому можна було програвати звук натискання на цю кнопку, це показано на рисунку 3.18.

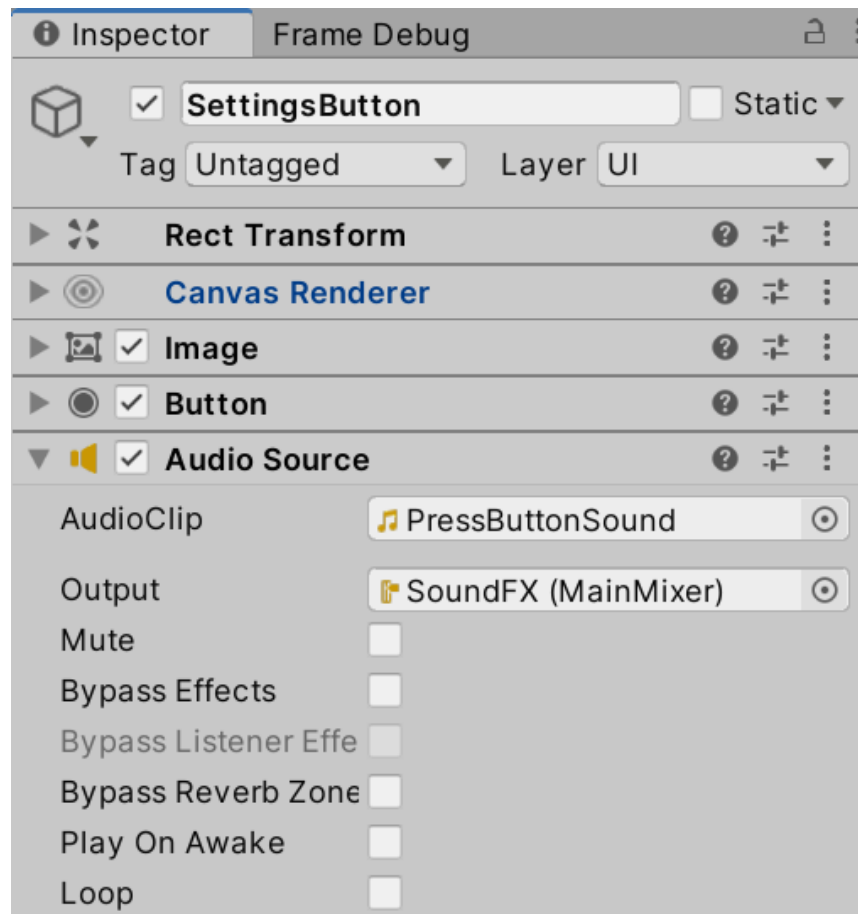


Рисунок 3.18 — Кнопка налаштувань з компонентом `AudioSource`

Для музики було створено окремий об'єкт, до якого теж було прикріплено `AudioSource`, але тут також у компонента `AudioSource` властивість `Loop` було виставлено в `true`, для того, щоб музика програвалася в циклі, тобто постійно. До цього об'єкта в майбутньому ми створимо скрипт, який заборонить знищення цього об'єкта під час переходу з однієї сцени в іншу, що дасть змогу програвати фонову музику під час усієї гри. Це показано на рисунку 3.19.

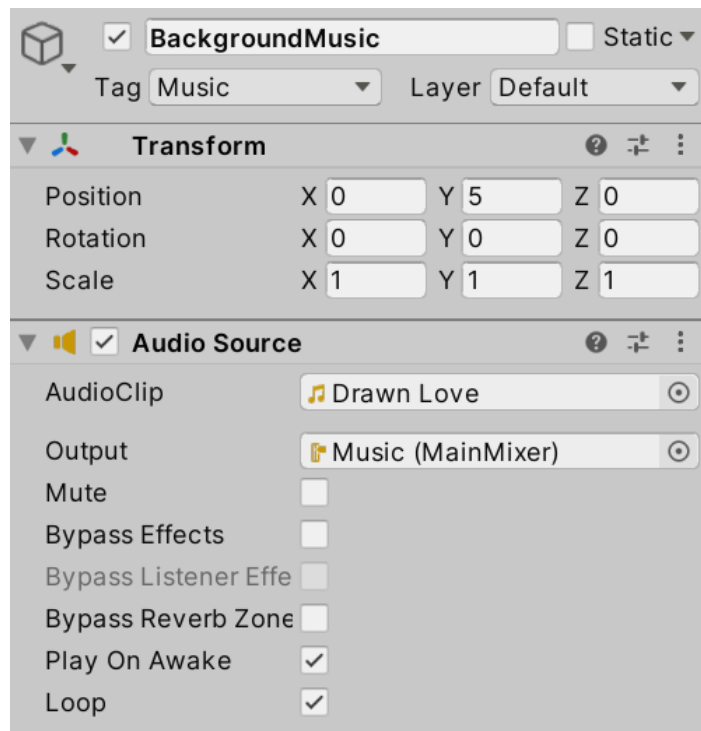


Рисунок 3.19 — Об'єкт з компонентом AudioSource, який програватиме фонову музику

У майбутньому ми напишемо скрипти для того, щоб гравець міг за допомогою слайдерів регулювати гучність через аудіомікшер.

3.1.5 Розробка основних об'єктів геймплею та механік гри

Сцена меню. У сцені меню, було створено вежу з деякої кількості кубів, які були поміщені в порожній об'єкт під назвою Tower. У майбутньому ми створимо скрипт і прикріпимо його до об'єкта Tower, у якому обиратимемо випадковий матеріал, а потім перебиратимемо його дочірні об'єкти-куби та кожному з них присвоюватимемо цей матеріал. У підсумку, щоразу, коли гравець переходитиме в меню, у нас буде вежа з різними скінами. Так само для цієї вежі було створено платформу, на якій вона буде розміщена. Камеру і світло так само помістили в порожній об'єкт. Для цього порожнього об'єкта розробили анімацію, яка буде переміщати камеру і світло навколо вежі, створюючи гарний проліт. Canvas містить весь інтерфейс сцени меню. Canvas потребує компонента Event System. Система подій обробляє події введення, такі як натискання кнопок, переміщення миші та введення з клавіатури. Також ще було створено два порожніх об'єкти, а саме BackgroundMusic і VolumeSettingsManager. Значення першого пояснювалося в

минулому під розділі 3.1.4. А об'єкт `VolumeSettingsManager` відповідатиме за встановлення гучності мікшерів і значень слайдерів у налаштуваннях, які було встановлено гравцем під час останнього виходу з гри. Для цього буде створено і прикріплено скрипт до цього об'єкта, який буде це встановлювати. На рисунку 3.20 зображено ієрархію створених об'єктів у сцені меню.

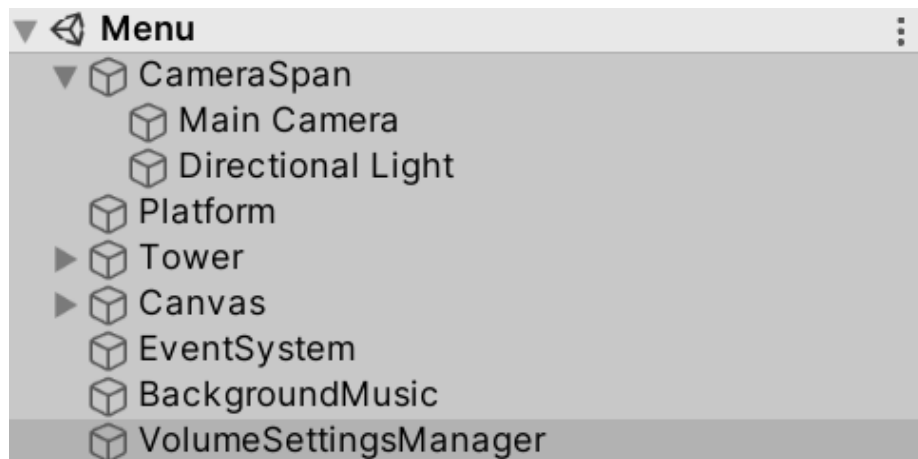


Рисунок 3.20 — Ієрархія об'єктів у сцені меню

Ігрова сцена. В основній сцені створено безліч об'єктів. Об'єкт `Rotator` містить камеру і світло. Потім на цей об'єкт буде прикріплено скрипт, за допомогою якого гравець зможе повертати камеру навколо вежі, водячи пальцем по екрану. Також було створено платформу, на якій гравець будуватиме вежу. Об'єкт `Tower` буде батьком усіх кубів, які розміщуватиме гравець. У нього вже поміщений один початковий куб, з якого починається гра і навколо якого будуть перебиратися позиції для наступного. `GhostCube` являє собою куб, який буде показувати гравцеві позицію для наступного куба. Цей куб буде постійно переміщатися навколо крайнього поставленого куба і таким чином показувати гравцю наступну позицію. Порожній об'єкт `GameController` потрібен для того, щоб потім на нього повісити основний скрипт, який керуватиме основною механікою гри. `Canvas` містить весь інтерфейс, а `Event System` обробляє події введення. `LeanTouch` і `LeanTapToSpawn` це об'єкти, які міститимуть скрипти для правильного тапа по екрану. `LeanTouch` це сторонній ресурс, який був узятий з `Unity Asset Store`, який дозволяє швидко розробляти ігри з узгодженим введенням на настільних і мобільних пристроях. Цей

легкий ресурс постачається з багатьма модульними компонентами, що дозволяє налаштувати їх відповідно до потреб проекту. LeanTouch це основний клас цього ресурсу, який дає змогу налаштувати систему введення так, як потрібно. Наприклад, ми можемо налаштувати скільки тапів потрібно для якоїсь дії або виставити чутливість введення і т.д. LeanTapToSpawn це об'єкт, що містить уже визначений скрипт, який відстежує тапи. У нашому випадку, ми будемо відстежувати тап гравця по екрану і розміщувати блок. RocketSpawner і GemSpawner це порожні об'єкти, на які будуть прикріплені скрипти, що відповідатимуть за спаун ракет і валюти відповідно. Trigger_DestroyFallingObjects і LoseTrigger_TowerFall — це 3D об'єкти, а якщо бути точніше, це площини, до яких прикріплений компонент BoxCollider, і його властивість isTrigger виставлена в true, що дає змогу вимкнути зіткнення з цим 3D об'єктом. На ці об'єкти буде прикріплено скрипти, які відповідатимуть за їх функціонал. Площина Trigger_DestroyFallingObjects буде великого розміру за довжиною і буде розміщена на певній відстані нижче за платформу. Вона слугуватиме тригером для знищення падаючих кубів, які розлітатимуться після вибуху при попаданні ракети. А площина LoseTrigger_TowerFall також буде великого розміру за довжиною і буде розміщена одразу під платформою. Вона відстежуватиме падаючу вежу і виконуватиме потрібні дії, які будуть прописані в її скрипті для обробки програшу.

На рисунку 3.21 зображена ієрархія об'єктів в основній ігровій сцені.

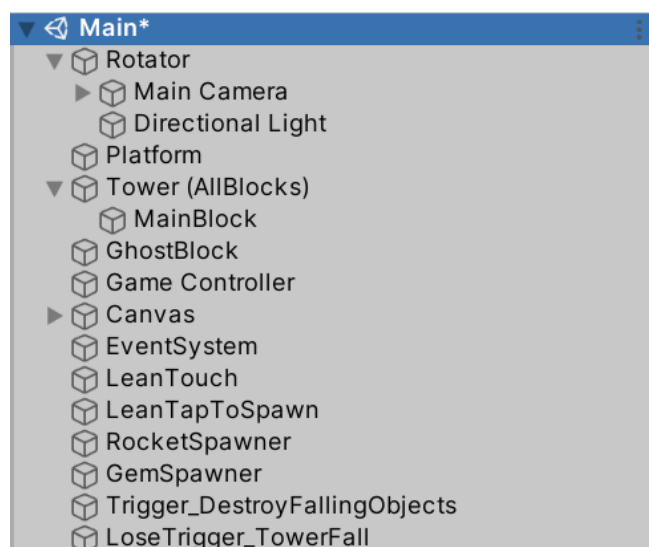


Рисунок 3.21 — Ієрархія об'єктів в основній сцені

Сцена магазину. У сцені магазину міститься небагато об'єктів. А саме камера, світло, Canvas, який містить увесь інтерфейс і функціонал магазину та EventSystem який обробляє події введення.

На рисунку 3.22 зображена ієрархія об'єктів у сцені магазину.

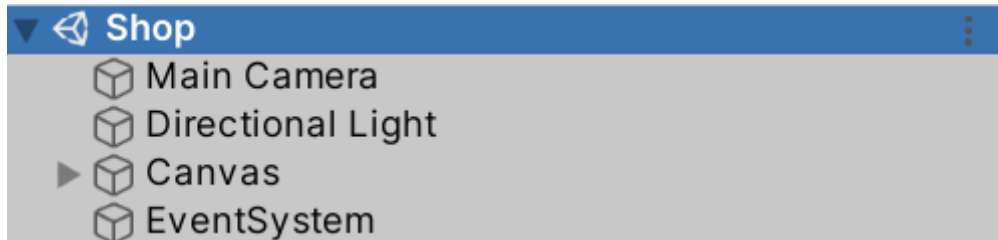


Рисунок 3.22 — Ієрархія об'єктів в сцені магазину

3.2 Розробка програмного коду гри відповідно до вибраної технології та концепції гри

У лістингу 3.1 показаний скрипт який прикріплений до об'єкта Tower у меню, в якому в методі Awake() обираємо випадковий матеріал, а потім перебираємо його дочірні об'єкти-куби та кожному з них присвоюємо цей матеріал. Тобто, щоразу, коли гравець переходить у меню, у нас буде генеруватися вежа з різними скінами.

Лістинг 3.1 — TowerRandomMaterialChanger.cs

```
using UnityEngine;

public class TowerRandomMaterialChanger : MonoBehaviour
{
    [SerializeField] private Material[] _materials;
    private Transform _thisGameObjectTransform;
    private int _randomMaterialIndex;
    private void Awake()
    {
        _thisGameObjectTransform = GetComponent<Transform>();
        _randomMaterialIndex = Random.Range(0, _materials.Length
- 1);
```

```

        for (int i = 0; i < _thisGameObjectTransform.childCount;
i++)
    {
        _thisGameObjectTransform.GetChild(i).GetComponent<MeshRenderer>().ma
terial = _materials[_randomMaterialIndex];
    }
}

```

У лістингу 3.2 показаний скрипт який прикріплений до пустого об'єкта VolumeSettingsManager в сцені меню. Цей скрипт відповідатиме за встановлення гучності мікшерів і значень слайдерів у налаштуваннях, які було встановлено гравцем під час останнього виходу з гри. Для цього в методі Start() який викликається під час старту сцени, буде використано клас PlayerPrefs, який зберігає вподобання гравця між ігровими сесіями. Він може зберігати строкові, плаваючі та цілочисельні значення в реєстрі платформи користувача. У лістингу 3.2 будуть зчитуватися дані з цього класу.

Лістинг 3.2 — VolumeSettingsManager.cs

```

using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Audio;

public class VolumeSettingsManager : MonoBehaviour
{
    private const float VolumeAtFirstLaunch = 0.50f;
    [SerializeField] private Slider SliderSettingsMusicVolume;
    [SerializeField] private Slider SliderSettingsSoundFXVolume;
    [SerializeField] private AudioMixerGroup musicMixer;
    [SerializeField] private AudioMixerGroup soundFXMixer;
    private static readonly string _firstLaunchOfAppString =
"firstLaunchOfAppEver";
    [SerializeField] public static readonly string
_musicVolumePrefs = "MusicVolumePref";
    [SerializeField] public static readonly string
_soundFXVolumePrefs = "SoundFXVolumePref";
    private int _firstLaunchOfAppInt;

    private void Start()
    {
        _firstLaunchOfAppInt =
PlayerPrefs.GetInt(_firstLaunchOfAppString);
        if (_firstLaunchOfAppInt == 0) //If first launch ever
        {
            SliderSettingsMusicVolume.value =
VolumeAtFirstLaunch;

```

```

        musicMixer.audioMixer.SetFloat("MusicVolume",
Mathf.Log10(0.50f) * 20);

        SliderSettingsSoundFXVolume.value =
VolumeAtFirstLaunch;
        musicMixer.audioMixer.SetFloat("SoundFXVolume",
Mathf.Log10(0.50f) * 20);
        PlayerPrefs.SetFloat(_musicVolumePrefs,
SliderSettingsMusicVolume.value);
        PlayerPrefs.SetFloat(_soundFXVolumePrefs,
SliderSettingsSoundFXVolume.value);
        PlayerPrefs.SetInt(_firstLaunchOfAppString, -1);
    }
    else if( _firstLaunchOfAppInt == -1) //If not first
launch
    {
        SliderSettingsMusicVolume.value =
PlayerPrefs.GetFloat(_musicVolumePrefs);
        musicMixer.audioMixer.SetFloat("MusicVolume",
Mathf.Log10(PlayerPrefs.GetFloat(_musicVolumePrefs)) * 20);

        SliderSettingsSoundFXVolume.value =
PlayerPrefs.GetFloat(_soundFXVolumePrefs);
        musicMixer.audioMixer.SetFloat("SoundFXVolume",
Mathf.Log10(PlayerPrefs.GetFloat(_soundFXVolumePrefs)) * 20);
    }
}
}

```

А у лістингу 3.3 показаний скрипт який призначений для того, щоб зберігати стан гучності у клас PlayerPrefs, який виставив гравець, щоб потім, за допомогою лістингу 3.2, зчитати ці значення та встановити таку гучність музики та звуків, яка була в крайній сесії гравця.

Лістинг 3.3 — SaveVolumeSettings.cs

```

using UnityEngine;
using UnityEngine.UI;

public class SaveVolumeSettings : MonoBehaviour
{
    [SerializeField] private Slider SliderMusicVolume;
    [SerializeField] private Slider SliderSoundFXVolume;

    public void SaveVolumeSettingsButton()
    {
        PlayerPrefs.SetFloat(VolumeSettingsManager._musicVolumePrefs,
SliderMusicVolume.value);
    }
}

```

```

PlayerPrefs.SetFloat (VolumeSettingsManager._soundFXVolumePrefs,
SliderSoundFXVolume.value);
    }
}

```

Скрипт, який показаний у лістингу 3.4, буде прикріплено до об'єкта Rotator в основній сцені, який є батьком таких об'єктів, як камера і світло. За допомогою нього гравець зможе повертати камеру навколо вежі, простим рухом пальця по екрану.

Лістинг 3.4 — FreeCameraRotatorByHand.cs

```

using UnityEngine;
using UnityEngine.EventSystems;

public class FreeCameraRotatorByHand : MonoBehaviour
{
    private Vector3 _firstPoint;
    private Vector3 _secondPoint;
    private float _xAngle = 0.0f;
    private float _xAngleTemp = 0.0f;
    [SerializeField] private float _maxRotationAngle = 180.0f;

    private void Start() {
        _xAngle = 0.0f;
        transform.rotation = Quaternion.Euler(0, _xAngle, 0.0f);
    }
    private void Update() {
        if (Input.touchCount > 0 &&
!IsPointerOverGameObjectTouch()) {
            if (Input.GetTouch(0).phase == TouchPhase.Began) {
                _firstPoint = Input.GetTouch(0).position;
                _xAngleTemp = _xAngle;
            }
            if (Input.GetTouch(0).phase == TouchPhase.Moved &&
!IsPointerOverGameObjectTouch()) {
                _secondPoint = Input.GetTouch(0).position;
                _xAngle = _xAngleTemp + (_secondPoint.x -
_firstPoint.x) * _maxRotationAngle / Screen.width;
                transform.rotation = Quaternion.Euler(0,
_xAngle, 0.0f);}}
        private bool IsPointerOverGameObjectTouch() {
            if
(EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).fingerId))

                return true;
            else
                return false;}}

```

У лістингу 3.5 показаний скрипт який прикріплений до об'єкта LoseTrigger_TowerFall в основній сцені. Він відстежуватиме падаючу вежу, а саме, коли Collider падаючої вежі входить у тригер (Collider) об'єкта LoseTrigger_TowerFall, викликається OnTriggerEnter() в якому обробляється програш.

Лістинг 3.5 — LoseTrigger_TowerFall.cs

```
using UnityEngine;

public class LoseTrigger_TowerFall : MonoBehaviour
{
    [SerializeField] private GameObject _loseMenu;
    [SerializeField] private GameObject _pauseButton;
    [SerializeField] private GameObject _rocketSpawner;
    static public bool IsLoseAlreadyHappened = false;
    private bool _executedOnce = false;

    private void Awake(){ IsLoseAlreadyHappened = false; }

    private void OnTriggerEnter(Collider other){
        if (other.gameObject.tag != "Rocket" && !_executedOnce &&
        !IsLoseAlreadyHappened){
            IsLoseAlreadyHappened = true;
            _rocketSpawner.gameObject.SetActive(false);
            _pauseButton.SetActive(false);
            _loseMenu.SetActive(true);
            _executedOnce = true;
            GetComponent().Play();
        }
    }
}
```

Наступний скрипт, який показаний у лістингу 3.6, прикріплений до об'єкта Trigger_DestroyFallingObjects в основній сцені. Він також виступатиме тригером і буде відстежувати падіння кубів при попаданні ракети у вежу, та знищувати їх. Це було зроблено для оптимізації гри, щоб куби нескінченно не падали донизу і фізичному двигуну не треба було це відстежувати.

Лістинг 3.6 — Trigger_DestroyFallingObjects.cs

```
using UnityEngine;

public class Trigger_DestroyFallingObjects : MonoBehaviour
{

```

```

private void OnTriggerEnter(Collider other)
{
    if(other.gameObject.tag != "Rocket")
        Destroy(other.gameObject);
}
}

```

Наступні два скрипти будуть пов'язані з ігровою валютою. А саме, перший, який подано у додатку А, буде прикріплено до порожнього об'єкта GemSpawner, який відповідатиме за вибір випадкової позиції та потім спауна ігрової валюти в грі. А другий, який подано у додатку Б, буде прикріплено до префабу дорогоцінного каменю, який виступає як валюта в нашій грі. Він оброблятиме всі дії, пов'язані з ним. А саме, він відстежуватиме свій час життя і буде знищуватися в разі його закінчення, а також оброблятиме його підбір гравцем і нараховуватиме потрібну кількість валюти гравцеві, а ще відповідатиме за ефекти підбору, звук і т.д.

Скрипти RocketSpawner і RocketDestroyer, які подані в додатках В і Г відповідно, відповідають за систему ракет. Перший скрипт (Додаток В) прикріплений на порожній об'єкт RocketSpawner в основній сцені. Він відповідає за підбір позиції, часу спауна і виконує спаун ракет і хвиль ракет. А другий скрипт (Додаток Г), прикріплений на префаб ракети, яку спаунить минулий скрипт, і він відповідає за опрацювання всіх дій пов'язаних із ракетою, таких як опрацювання знищення ракети тапом, потрапляння ракети у вежу тощо.

Скрипт, який показаний у лістингу 3.7 відповідає за контроль паузи в основній сцені, він прикріплений до кнопки паузи.

Лістинг 3.7 — PauseController.cs

```

using UnityEngine;

public class PauseController : MonoBehaviour
{
    [HideInInspector] public static bool IsGamePaused = false;
    [SerializeField] private GameObject _pausePopUp;
    public void PauseGame()
    {
        GetComponent().Play();
        _pausePopUp.SetActive(true);
    }
}

```

```

        Time.timeScale = 0f;
        IsGamePaused = true;
    }

    public void UnPauseGame()
    {
        GetComponent().Play();
        Time.timeScale = 1f;
        IsGamePaused = false;
    }

    private void OnApplicationFocus(bool focus)
    {
        if(!IsGamePaused && !focus)
        {
            Time.timeScale = 0f;
            IsGamePaused = true;
            _pausePopUp.SetActive(true);
        }
    }
}

```

3.3 Тестування гри

Тестування розробленої мобільної гри має вирішальне значення з кількох причин. Перш за все, воно гарантує, що гра функціонує за призначенням і забезпечує плавний і приємний користувацький досвід. Виявляючи та виправляючи будь-які помилки, збої або технічні проблеми, тестування допомагає підвищити загальну якість гри. Крім того, тестування дозволяє оцінити продуктивність гри на різних мобільних пристроях та операційних системах. Це важливо, оскільки мобільні пристрої бувають різних моделей, розмірів екрану та апаратних можливостей. Тестування допомагає забезпечити сумісність і оптимізувати роботу гри на широкому спектрі пристроїв, тим самим охоплюючи більшу аудиторію. Крім того, тестування допомагає перевірити дизайн і механіку гри. Воно дозволяє визначити рівень складності гри, збалансувати елементи ігрового процесу [53].

3.3.1 Тестування ігрового процесу та виявлення несправностей

Під час тестування ігрового процесу було помічено, що ракети вилітають доволі часто, що призводить до того, що у гравця залишається менше часу на концентрацію побудови вежі. Було вирішено трошки зменшити частоту вильоту ракет, щоб дати гравцеві більше часу на побудову вежі та збирання валюти.

Також під час тестування було виявлено несправність, пов'язану з тим, що після того, як ми програли та в нас з'явилося вікно з програшем (див. рисунок 3.9), ми все ще могли повертати камеру, рухаючи пальцем по вікну програшу. Після з'ясування і пошуку бага, було з'ясовано, що цей баг пов'язаний з тим, що в скрипті повороту камери (див. лістинг 3.4) не перевірялася умова, чи водить гравець пальцем над UI елементами (як вікна, кнопки і т.д.). Після додавання методу в цей скрипт на перевірку цієї умови, баг було успішно виправлено.

3.3.2 Тестування продуктивності гри

Тестування перформансу гри проводилося на телефоні Redmi Note 7, який має наступні характеристики:

- роздільна здатність — 1080x2340 пікселів;
- співвідношення сторін — 19.5:9;
- частота оновлення — 60 Гц;
- процесор (CPU) — Qualcomm Snapdragon 660, 2200 МГц, 8 ядер;
- графіка (GPU) — Adreno 512, 850 МГц;
- обсяг ОЗП — 4 Гб, 1866 МГц;
- версія операційної системи Android — Android 10.

Тестування проводилося за допомогою вбудованої утиліти MIUI Power Monitor. Результати тестування показані на рисунках 3.23, 3.24, 3.25. В результаті тестування можна сказати, що fps (frames per second) у меню майже стабільно тримається на 60, але бувають просідання до 57 fps. Це не значне просідання, особливо з огляду на те, що сцена меню не має ігрового навантаження, і тут він ні на що не впливає. У основній сцені fps стабільно тримається на 60 і просідання бувають тільки під час потрапляння ракети у вежу, оскільки відбувається багато обчислень фізичного двигуна, під час розпаду і розльоту блоків. У магазині

стабільні 59-60 fps. Загалом, можна сказати, що це чудовий результат, особливо з огляду на те, що апарат, на якому проводили тест, вже є застарілим.



Рисунок 3.23 — Результат тестування перформансу гри у сцені меню

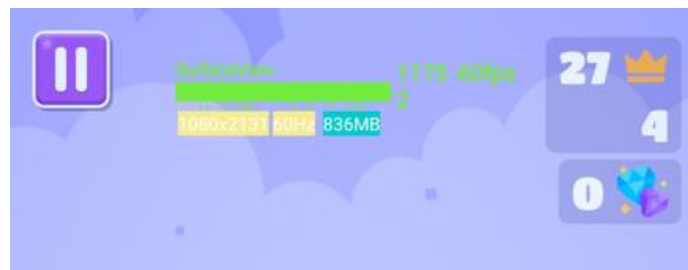


Рисунок 3.24 — Результат тестування перформансу гри у основній сцені

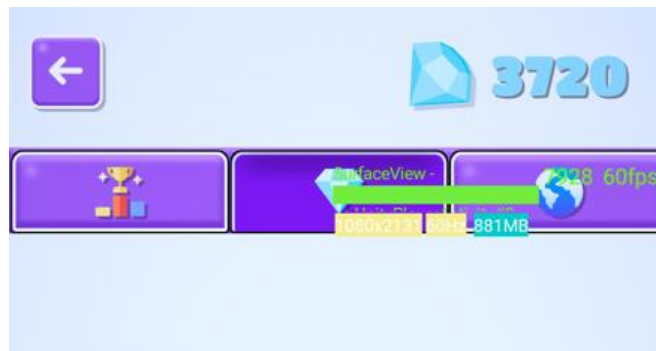


Рисунок 3.25 — Результат тестування перформансу гри у сцені магазину

ВИСНОВКИ

Мобільні ігри стають все більш популярним видом розваг і мають значний вплив на життя людини. Вони є джерелом розваг для людей різного віку. Мобільні ігри легкодоступні, і в них можна грати будь-де та будь-коли. Вони допомагають людям розслабитися та відволіктися від стресів повсякденного життя. Тому мобільні ігри є дуже важливими для людей у наш час.

Розробка мобільної гри — це багатоетапний процес, який включає різні етапи та ресурси, такі як планування та концептуалізація гри, проектування геймплею та інтерфейсу, програмування та тестування гри, створення графіки та звукових ефектів, а також підтримка та оновлення гри після її випуску. Всі ці етапи вимагають значних витрат часу, праці та грошей, тому розробка мобільної гри є складним та багатогранним процесом.

Метою дослідження була розробка мобільної аркадної гри на базі операційної системи Android.

Для досягнення мети були поставлені такі завдання:

- переглянути програмні засоби та технології розробки мобільних ігор;
- проаналізувати технічні вимоги до мобільної гри на платформі Android;
- розробити концепцію гри;
- спроектувати та розробити структуру гри;
- реалізувати гру;
- протестувати розроблену гру.

Під час дослідження програмних засобів та технологій розробки мобільних ігор були переглянуті різні підходи та інструменти, які використовуються в індустрії розробки ігор для платформи Android. Це дозволило зрозуміти сучасні тренди та найкращі практики розробки мобільних ігор. Аналіз технічних вимог до мобільної гри на платформі Android дозволив визначити основні характеристики, які має відповідати розроблена гра, включаючи підтримку певних версій операційної системи, роздільну здатність екрану, обсяг пам'яті та інші

функціональні вимоги. Розробка концепції гри була проведена з урахуванням цільової аудиторії, геймплею та унікальності гри. Були визначені: механіка гри, стиль візуального оформлення. Структура гри була спроектована, включаючи організацію сцен, взаємодію об'єктів гри та інші важливі елементи. Це дозволило створити логічну та зручну структуру для подальшого розроблення гри. За допомогою використання вибраних програмних засобів та технологій розробки мобільних ігор, була реалізована гра на базі операційної системи Android. Розроблено графічний інтерфейс, геймплейні механіки та систему управління, що відповідають визначеним технічним вимогам та концепції гри. Після реалізації гри проведено тестування, що дозволило виявити та виправити помилки, забезпечити стабільну роботу гри та задовольнити очікування користувачів. Тестування включало перевірку функціональності, продуктивності та загальної якості гри.

Мета дослідження роботи була досягнута, а завдання були успішно виконані. Результатом дослідження є розроблена мобільна аркадна гра на базі операційної системи Android, яка відповідає вимогам та концепції гри, а також пройшла тестування для забезпечення якості та задоволення користувачів.

Рекомендації для подальшого розвитку та вдосконалення гри:

- додавання ще більше різноманітних скінів у магазин;
- розробка і додавання гарних переходів між сценами, для більш гладкого і приємного переміщення гравця між сценами;
- додавання системи перків у гру, які урізноманітнять геймплей;
- додавання локалізації гри на різні мови та реалізація функції зміни мови в налаштуваннях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дані ігрового ринку. URL: <https://newzoo.com/> (дата звернення: 20.04.2023).
2. Процес розробки мобільних ігор. URL: <https://whimsygames.co/blog/game-development-process/> (дата звернення: 20.04.2023).
3. Що таке Ігровий двигун. URL: <https://www.studytonight.com/3d-game-engineering-with-unity/game-engine> (дата звернення: 20.04.2023).
4. Офіційний веб-сайт Godot Engine. URL: <https://godotengine.org/> (дата звернення: 21.04.2023).
5. Логотип ігрового двигуна Godot Engine. URL: <https://godotengine.org/press/> (дата звернення: 21.04.2023).
6. Документація Godot Engine. URL: <https://docs.godotengine.org/en/stable/> (дата звернення: 21.04.2023).
7. Reddit ком'юніті Godot Engine. URL: <https://www.reddit.com/r/godot/> (дата звернення: 21.04.2023).
8. Огляд і відгуки Godot Engine на веб-сайті capterra. URL: <https://www.capterra.com/p/234843/Godot-Engine/> (дата звернення: 22.04.2023).
9. Логотип ігрового двигуна Construct 3. URL: https://commons.wikimedia.org/wiki/File:Construct_3_Logo.svg (дата звернення: 22.04.2023).
10. Офіційний Веб-сайт Construct 3. URL: <https://www.construct.net/en> (дата звернення: 22.04.2023).
11. Огляд і відгуки Construct 3 на веб-сайті g2. URL: <https://www.g2.com/products/construct/reviews> (дата звернення: 22.04.2023).
12. Логотип ігрового двигуна Cocos2D. URL: https://www.cocos.com/_nuxt/img/logo.b924db5.png (дата звернення: 22.04.2023).
13. Офіційний веб-сайт Cocos2D. URL: <https://www.cocos.com/> (дата звернення: 22.04.2023).

14. Огляд і відгуки Cocos2D на веб-сайті g2. URL: <https://www.g2.com/products/cocos2d/reviews> (дата звернення: 22.04.2023).
15. Логотип ігрового двигуна Corona SDK. URL: <https://logos-download.com/70375-corona-sdk-logo-download.html> (дата звернення: 22.04.2023).
16. Що таке SDK. URL: <https://www.adjust.com/glossary/sdk/> (дата звернення: 22.04.2023).
17. Веб-сайт Corona SDK. URL: <https://coronalabs.com/> (дата звернення: 22.04.2023).
18. Огляд і відгуки Corona SDK на веб-сайті g2. URL: <https://www.g2.com/products/corona-sdk/reviews> (дата звернення: 22.04.2023).
19. Офіційний веб-сайт GameMaker Studio. URL: <https://www.yooyogames.com/gamemaker> (дата звернення: 23.04.2023).
20. Логотип ігрового двигуна GameMaker Studio. URL: <https://gamemaker.io/en/legal/brand> (дата звернення: 23.04.2023).
21. Огляд і відгуки GameMaker Studio на веб-сайті capterra. URL: <https://www.capterra.com/p/158594/GameMaker-Studio/reviews/> (дата звернення: 23.04.2023).
22. Логотип ігрового двигуна Unreal Engine. URL: <https://epicgames.ent.box.com/s/c2m8idcyejqvg5mjf4e2q73b7jbaghft> (дата звернення: 24.04.2023).
23. Офіційний веб-сайт Unreal Engine. URL: <https://www.unrealengine.com/> (дата звернення: 24.04.2023).
24. Форуми Unreal Engine. URL: <https://forums.unrealengine.com/> (дата звернення: 24.04.2023).
25. Aram Cookson. Unreal Engine 4 Game Development in 24 Hours, Sams Teach Yourself 1st Edition. Видавництво "Sams Publishing", 2016. 496 с.
26. Огляд і відгуки Unreal Engine на веб-сайті g2. URL: <https://www.g2.com/products/unreal-engine/reviews> (дата звернення: 24.04.2023).
27. Satheesh PV. Unreal Engine 4 Game Development Essentials. Видавництво "Packt Publishing", 2016. 266 с.

28. Логотип ігрового двигуна Unity. URL: <https://brandguide.brandfolder.com/unity/unitylogo> (дата звернення: 25.04.2023).
29. Офіційний сайт Unity. URL: <https://unity.com/> (дата звернення: 25.04.2023).
30. Joe Hocking. Unity in Action: Multiplatform Game Development in C# with Unity 5 1st Edition. Видавництво "Manning Publications", 2015. 352 с.
31. Unity Asset Store. URL: <https://assetstore.unity.com/> (дата звернення: 25.04.2023).
32. Unity Learn. URL: <https://learn.unity.com/> (дата звернення: 25.04.2023).
33. Огляд і відгуки Unity на веб-сайті capterra. URL: <https://www.capterra.com/p/158591/Unity/> (дата звернення: 25.04.2023).
34. Unity Answers. URL: <https://answers.unity.com/index.html> (дата звернення: 25.04.2023).
35. Unity Forums. URL: <https://forum.unity.com/> (дата звернення: 25.04.2023).
36. UnityScript. URL: <https://blog.unity.com/community/unityscripts-long-ride-off-into-the-sunset> (дата звернення: 28.04.2023).
37. Unity-скриптинг. URL: <https://blog.unity.com/technology/documentation-unity-scripting-languages-and-you> (дата звернення: 28.04.2023).
38. Скриптинг на C# в Unity для новачків. URL: <https://unity.com/how-to/learning-c-sharp-unity-beginners> (дата звернення: 28.04.2023).
39. Розробка вашої першої гри за допомогою Unity та C#. URL: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2014/august/unity-developing-your-first-game-with-unity-and-csharp> (дата звернення: 28.04.2023).
40. Що таке IDE. URL: <https://www.redhat.com/en/topics/middleware/what-is-ide> (дата звернення: 3.05.2023).
41. IDE які підтримує Unity. URL: <https://docs.unity3d.com/Manual/ScriptingToolsIDEs.html> (дата звернення: 3.05.2023).
42. Логотип MonoDevelop. URL: <https://www.logo.wine/logo/MonoDevelop> (дата звернення: 3.05.2023).

43. Про MonoDevelop на офіційному сайті Unity. URL: <https://docs.unity3d.com/ru/530/Manual/MonoDevelop.html> (дата звернення: 3.05.2023).

44. Логотип JetBrains Rider. URL: <https://www.jetbrains.com/company/brand/> (дата звернення: 3.05.2023).

45. Про JetBrains Rider на офіційному сайті. URL: <https://www.jetbrains.com/rider/> (дата звернення: 3.05.2023).

46. Офіційний веб-сайт Microsoft про їх IDE. URL: <https://visualstudio.microsoft.com/> (дата звернення: 3.05.2023).

47. Огляд і різниця між IDE Visual Studio Code і Visual Studio. URL: <https://www.freecodecamp.org/news/visual-studio-vs-visual-studio-code/> (дата звернення: 3.05.2023).

48. Версії Android, жива історія. URL: <https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html> (дата звернення: 4.05.2023).

49. Офіційний веб-сайт Adobe, для підбору і створення палітри кольорів. URL: <https://color.adobe.com/ru/create/color-wheel> (дата звернення: 8.05.2023).

50. Безкоштовні іконки для проектів. URL: <https://www.flaticon.com/> (дата звернення: 8.05.2023).

51. Офіційний веб-сайт Blender. URL: <https://www.blender.org/> (дата звернення: 8.05.2023).

52. Офіційний туторіал про систему частинок Unity. URL: <https://learn.unity.com/tutorial/introduction-to-particle-systems#> (дата звернення: 10.05.2023).

53. Важливість тестування в розробці мобільних ігор. URL: <https://starloopstudios.com/the-importance-of-testing-in-mobile-game-development/> (дата звернення: 12.05.2023).

ДОДАТОК А

Лістинг А.1 — Скрипт GemSpawner

```

using System.Collections;
using UnityEngine;

public class GemsSpawner : MonoBehaviour
{
    [SerializeField] private GameController _gameController;
    [SerializeField] private Transform _forDecentGemsAnimation;
    [SerializeField] private GameObject[] _currency;
    [SerializeField] private float _megaGemSpawnChance = 0.10f;
    private WaitUntil _spawnCurrencyWU;
    private GameObject _currentCurrency;
    private float _randMegaGem;
    private Vector3 _currencyCurrentPosition;

    private void Start()
    {
        _spawnCurrencyWU = new WaitUntil(IsCurrencyTaken);
        StartCoroutine(SpawnCurrencyIE());

        IEnumerator SpawnCurrencyIE()
        {
            while (!_gameController.IsNotLose)
            {
                SpawnCurrency();
                yield return _spawnCurrencyWU;
            }

            private void SpawnCurrency()
            {
                _forDecentGemsAnimation.position =
RandPositionOfSpawnCurrency();
                _currentCurrency =
Instantiate(WhatTypeOfCurrencySpawn(), _forDecentGemsAnimation);
            }

            private bool IsCurrencyTaken()
            {
                if (_currentCurrency != null)
                    return false;
                return true;
            }

            private GameObject WhatTypeOfCurrencySpawn()
            {
                if (MegaGemRandSpawn())
                    return _currency[3];
                if (_gameController.CubeCounter <= 25)

```

```

        return _currency[0];
    if (_gameController.CubeCounter <= 50)
        return _currency[1];
    return _currency[2];
}

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

private Vector3 RandPositionOfSpawnCurrency()
{
    while (true){
        _currencyCurrentPosition = new Vector3(
            Random.Range(_gameController.CurrentCube.x - 2,
            _gameController.CurrentCube.x + 2),
            Random.Range(_gameController.CurrentCube.y,
            _gameController.CurrentCube.y + 5),
            Random.Range(_gameController.CurrentCube.z - 2,
            _gameController.CurrentCube.z + 2));
        if
        (_gameController.IsPositionEmpty(_currencyCurrentPosition))
            return _currencyCurrentPosition;
    }}}

```

Лістинг А.2 — Скрипт SimpleCollectibleScript

```

using UnityEngine.UI;
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(AudioSource))]
public class SimpleCollectibleScript : MonoBehaviour
{
    [HideInInspector] public static int GemsCountForLosePopUI =
0;

    public enum CollectibleTypes { NoType, Type1, Type2, Type3,
Type4, Type5 };
    [SerializeField] private CollectibleTypes _collectibleType;
    [SerializeField] private GameObject _currencyLevitationVFX;
    private GameObject _VFX;
    [SerializeField] private AudioClip _collectSound;
    [SerializeField] private GameObject _collectEffect;
    private WaitForSeconds _gemLifeTimeWFS;
    [SerializeField] private float _gemLifeTime;
    [SerializeField] private float _rotationSpeed;
    [SerializeField] private bool _rotate = true;
    [SerializeField] private bool _VFXLevitation;
    private Transform _rotator;

```

```

private Animator _collectAnimation;
private bool _isEntered = false;

void Start ()
{
    _gemLifeTimeWFS = new WaitForSeconds(_gemLifeTime);
    _collectAnimation = GetComponent<Animator>();
    _rotator = GetComponent<Transform>();

    StartCoroutine(DestroyGemIfLifeTimeIsOver());

    if (_VFXLevitation)
    {
        _VFX = Instantiate(_currencyLevitationVFX,
        GetComponent<Transform>().position, Quaternion.identity);
        ParticleSystem.MainModule main =
        GetComponent<ParticleSystem>().main;
        main.startColor = Color.red;
    }
}

void Update()
{
    if (_rotate)
    {
        _rotator.Rotate(-Vector3.up * _rotationSpeed *
        Time.deltaTime);
        if (_VFXLevitation)
            _VFX.GetComponent<Transform>().Rotate(-
            Vector3.up * _rotationSpeed * Time.deltaTime); //NEW
    }
}

void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "Cubes" && !_isEntered)
    {
        Collect();
        _isEntered = true;
    }
}

public void Collect()
{
    if(_collectSound)
        GetComponent<AudioSource>().Play();
    if (_collectEffect)
        Instantiate(_collectEffect, transform.position,
        _collectEffect.transform.rotation);
    if (_collectAnimation)
        _collectAnimation.SetTrigger("Tr_Fade");
    if (_collectibleType == CollectibleTypes.NoType)

```

```

        Debug.Log ("Do NoType Command");
        if (_collectibleType == CollectibleTypes.Type1)
            UpdateGemsUI(500);
        if (_collectibleType == CollectibleTypes.Type2)
            UpdateGemsUI(2);
        if (_collectibleType == CollectibleTypes.Type3)
            UpdateGemsUI(3);
        if (_collectibleType == CollectibleTypes.Type5)
            UpdateGemsUI(5);
    }

    private void UpdateGemsUI(int amount)
    {
        GemsCountForLosePopUI += amount;

        GameObject.Find("Canvas").transform.Find("LosePopUp").transform
        .Find("Background").transform.Find("PopUp").transform.Find("ContentC
        ontainer").transform.Find("BackgroundGems").transform.Find("NumberGe
        ms").GetComponent<Text>().text = "<color=#6BD6FF>" +
        GemsCountForLosePopUI + "</color>";
        PlayerPrefs.SetInt("AmountOfGems",
        PlayerPrefs.GetInt("AmountOfGems") + amount);
        GameObject.Find("Game
        Controller").GetComponent<GameController>().AmountOfGemsText.text =
        GemsCountForLosePopUI.ToString();
    }

    public void DestroyGem()
    {
        if (_VFXLevitation)
            Destroy(_VFX);
        Destroy(gameObject);
    }

    IEnumerator DestroyGemIfLifeTimeIsOver()
    {
        while (true)
        {
            yield return _gemLifeTimeWFS;
            if (!_isEntered)
            {
                _isEntered = true;
                _collectAnimation.SetTrigger("Tr_Expiring");
            }
        }
    }
}

```

ДОДАТОК Б

Лістинг Б.1 — Скрипт RocketSpawner

```

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

public class RocketSpawner : MonoBehaviour
{
    [SerializeField] private GameController _gameController;
    [SerializeField] private GameObject _rocket;
    [SerializeField] private GameObject _rocketWaveWarningPopUp;
    [SerializeField] private int _xMinRocketSpawnRange;
    [SerializeField] private int _xMaxRocketSpawnRange;
    [SerializeField] private int _zMinRocketSpawnRange;
    [SerializeField] private int _zMaxRocketSpawnRange;
    [SerializeField] private int _minTimeToSpawn;
    [SerializeField] private int _maxTimeToSpawn;
    [SerializeField] private float _rocketWaveChance;
    private Transform _cameraTransform;
    private WaitForSeconds _delayBeforeRocketWaveSpawnWFS;
    private readonly float _delayBeforeRocketWaveSpawn = 3f;
    private WaitForSeconds _delayAfterRocketWaveWFS;
    private int _cubeAmountToNextRocketWave = 5;

    private void Start()
    {
        _cameraTransform = Camera.main.transform;
        _delayBeforeRocketWaveSpawnWFS = new
WaitForSeconds(_delayBeforeRocketWaveSpawn);
        _delayAfterRocketWaveWFS = new WaitForSeconds(2f);
        StartCoroutine(SpawnRocketIE());
    }
    IEnumerator SpawnRocketIE()
    {
        while (true)
        {
            if (_gameController.CubeCounter > 0)
            {
                if (_gameController.CubeCounter >
_cubeAmountToNextRocketWave && RocketWaveRandSpawn())
                {
                    _rocketWaveWarningPopUp.SetActive(true);
                    yield return _delayBeforeRocketWaveSpawnWFS;
                    SpawnRocketWave(3);
                    _cubeAmountToNextRocketWave = 5 +
_gameController.CubeCounter;
                    yield return _delayAfterRocketWaveWFS;
                }
            }
        }
    }
}

```

```

        else
        {
            SpawnRocket();
            yield return new WaitForSeconds(RandomTimeToSpawn(_minTimeToSpawn, _maxTimeToSpawn));
        }
    }
    else
        yield return null;
}

private bool RocketWaveRandSpawn()
{
    if (Random.value < _rocketWaveChance)
        return true;
    return false;
}

private void SpawnRocketWave(int rocketsAmount)
{
    List<Vector3> uniqueRocketPositions = new List<Vector3>();
    for (int x = _xMinRocketSpawnRange + (int)_gameController.LastPlacedCubePosition.x; x <= _xMaxRocketSpawnRange + (int)_gameController.LastPlacedCubePosition.x; x++)
    {
        for (int z = _zMinRocketSpawnRange + (int)_gameController.LastPlacedCubePosition.z; z <= _zMaxRocketSpawnRange + (int)_gameController.LastPlacedCubePosition.z; z++)
        {
            uniqueRocketPositions.Add(new Vector3(x, _cameraTransform.localPosition.y + 8, z));
        }
        for (int i = 0; i < rocketsAmount; i++)
        {
            int temp = Random.Range(0, uniqueRocketPositions.Count);
            Instantiate(_rocket, uniqueRocketPositions[temp], _rocket.transform.rotation);
            uniqueRocketPositions.RemoveAt(temp);
        }
    }

    private void SpawnRocket()
    {
        Instantiate(_rocket, RandomRocketPositionToSpawn(_xMinRocketSpawnRange, _xMaxRocketSpawnRange, _zMinRocketSpawnRange, _zMaxRocketSpawnRange), _rocket.transform.rotation);
    }

    private Vector3 RandomRocketPositionToSpawn(int minX, int maxX, int minZ, int maxZ)
    {

```

```

        int x = Random.Range(minX +
(int)_gameController.LastPlacedCubePosition.x,
(int)_gameController.LastPlacedCubePosition.x);
        int z = Random.Range(minZ +
(int)_gameController.LastPlacedCubePosition.z,
(int)_gameController.LastPlacedCubePosition.z);
        return new Vector3(x, _cameraTransform.localPosition.y +
8, z);
    }
    private int RandomTimeToSpawn(int min, int max)
    {
        return Random.Range(min, max);
    }
}

```

Лістинг Б.2 — Скрипт RocketDestroyer

```

using UnityEngine;
using System.Collections;
using UnityEngine.Events;

public class RocketDestroyer : MonoBehaviour
{
    [HideInInspector]
    public UnityEvent OnRocketDestroy;
    private readonly float _yRocketMaxFlightDistance = 24f;
    private Vector3 _rocketStartPosition;
    private Transform _rocketTransform;
    private Vector3 _explosionPosition;
    private bool _isRocketdestroyingSequenceStart = false;
    [SerializeField] GameObject _touchExplosionFX;
    [SerializeField] GameObject _destroyTowerExplosionFX;

    private void Start()
    {
        _rocketTransform = transform;
        _rocketStartPosition = _rocketTransform.position;
    }

    private void Update()
    {
        if (LoseTrigger_TowerFall.IsLoseAlreadyHappened)
        {
            OnRocketDestroy.Invoke();
            Instantiate(_destroyTowerExplosionFX,
gameObject.transform.position, Quaternion.identity);
            Destroy(gameObject);
        }

        if (Input.touchCount > 0)
        {
            if (Input.GetTouch(0).phase == TouchPhase.Began)
            {

```

```

Ray ray =
Camera.main.ScreenPointToRay(Input.GetTouch(0).position);
RaycastHit hit;
if (Physics.Raycast(ray, out hit, 100.0f))
{
    if (hit.collider != null &&
hit.collider.gameObject.GetInstanceID() ==
gameObject.GetInstanceID())
    {
        GameController.IsRocketDestroyed =
false;

gameObject.GetComponent<Rigidbody>().AddExplosionForce(5000f, new
Vector3(0, 0, 0), 100000);

StartCoroutine(DestroyTimer(_touchExplosionFX, 0.2f));
    }
}

if (Vector3.Distance(_rocketStartPosition,
_rocketTransform.position) > _yRocketMaxFlightDistance)
{
    OnRocketDestroy.Invoke();
    Destroy(gameObject);
}

private void OnCollisionEnter(Collision collision)
{
    if (LoseTrigger_TowerFall.IsLoseAlreadyHappened) return;
    if (collision.gameObject.tag == "Cube")
    {
        LoseTrigger_TowerFall.IsLoseAlreadyHappened = true;

Destroy(collision.GetContact(0).otherCollider.gameObject);
Instantiate(_destroyTowerExplosionFX,
gameObject.GetComponent<Transform>().position, Quaternion.identity);

Camera.main.gameObject.AddComponent<CameraShakeVFX>();
for (int i = collision.transform.childCount - 1; i >=
0; i--)
    {
        Transform child =
collision.transform.GetChild(i);
        child.gameObject.AddComponent<Rigidbody>();
        child.SetParent(null);
    }
    Collider[] colliders =
Physics.OverlapSphere(gameObject.transform.position, 50);
    foreach (Collider hit in colliders)
    {

```

```

        Rigidbody rb = hit.GetComponent<Rigidbody>();
        Vector3 pos =
gameObject.GetComponent<Transform>().position;
        pos.y += 1;

        if (rb != null)
            rb.AddExplosionForce(2000f, pos, 50f);
    }
    _explosionPosition =
gameObject.GetComponent<Transform>().position;
    _explosionPosition.y += 1;

gameObject.GetComponent<Rigidbody>().AddExplosionForce(3000f,
_explosionPosition, 50f);
    OnRocketDestroy.Invoke();
    GameObject.Find("PauseButton").SetActive(false);

GameObject.Find("Canvas").transform.Find("LosePopUp").gameObject.Set
Active(true);

GameObject.Find("RocketSpawnerParent").transform.GetChild(0).gameObj
ect.SetActive(false);
    Destroy(gameObject);
    }
    else if (collision.gameObject.tag ==
"PlatformRocketDestroyer")
    {
        LoseTrigger_TowerFall.IsLoseAlreadyHappened = true;
        Instantiate(_destroyTowerExplosionFX,
gameObject.GetComponent<Transform>().position, Quaternion.identity);

Camera.main.gameObject.AddComponent<CameraShakeVFX>();

        GameObject cubes = GameObject.Find("Tower
(AllBlocks)").gameObject;
        for (int i = cubes.transform.childCount - 1; i >= 0;
i--)
        {
            Transform child = cubes.transform.GetChild(i);
            child.gameObject.AddComponent<Rigidbody>();
            child.SetParent(null);
        }

        Collider[] colliders =
Physics.OverlapSphere(gameObject.transform.position, 50);
        foreach (Collider hit in colliders)
        {
            Rigidbody rb = hit.GetComponent<Rigidbody>();
            Vector3 pos =
gameObject.GetComponent<Transform>().position;
            pos.y += 1;

            if (rb != null)

```

```

        rb.AddExplosionForce(2000f, pos, 50f);
    }
    _explosionPosition =
gameObject.GetComponent<Transform>().position;
    _explosionPosition.y += 1;

gameObject.GetComponent<Rigidbody>().AddExplosionForce(3000f,
_explosionPosition, 50f);
    OnRocketDestroy.Invoke();
    GameObject.Find("PauseButton").SetActive(false);

GameObject.Find("Canvas").transform.Find("LosePopUp").gameObject.SetActive(true);

GameObject.Find("RocketSpawnerParent").transform.GetChild(0).gameObject.SetActive(false);
    Destroy(gameObject);
}
}

IEnumerator DestroyTimer(GameObject VFX, float
TimeBeforeDestroy = 0.2f)
{
    yield return new WaitForSeconds(TimeBeforeDestroy);
    GameController.IsRocketDestroyed = true;
    Instantiate(VFX,
gameObject.GetComponent<Transform>().position, Quaternion.identity);
    OnRocketDestroy.Invoke();
    Destroy(gameObject);
}
}

```

ДОДАТОК В

Лістинг В.1 — Скрипт ShopSystem

```

using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class ShopSystem : MonoBehaviour
{
    [SerializeField] private CubeSkinInfo[] _cubeSkins;
    [SerializeField] private GameObject _cubeSelectionFX;
    [SerializeField] private GameObject _amountOfGemsTextUI;

    [SerializeField] private AudioClip _cubeSelectionSound;
    [SerializeField] private AudioClip _pressingButtonSound;
    [HideInInspector] public CubeSkinInfo SaveEquippedSkinCube;
    private CubeSkinInfo _hitedObject;
    private Color _generalColor;
    [SerializeField] private GameObject _buyConfirmationPopUp;
    [SerializeField] private Text _infoAboutBuyingTextUI;
    private StateOfPopUp CurrentStateOfPopUp =
StateOfPopUp.None;
    [HideInInspector] public bool BlockTouchesWhenPopUpActive =
false;
    private enum StateOfPopUp
    {
        None,
        Yes,
        No
    }

    private void Awake()
    {
        _amountOfGemsTextUI.GetComponent<Text>().text =
PlayerPrefs.GetInt("AmountOfGems").ToString();
        ColorUtility.TryParseHtmlString("#BF2DDD", out
_generalColor);
    }

    private void Start()
    {
        for (int i = 0; i < _cubeSkins.Length; i++)
            if (IsUnlocked(_cubeSkins[i]))
            {
                _cubeSkins[i].Sign.text = "";
                if (PlayerPrefs.GetInt("EquippedCube") ==
_cubeSkins[i].SkinId)
                {
                    _cubeSkins[i].Sign.text = "EQUIPPED";
                }
            }
        }
    }

```

```

        _cubeSkins[i].Sign.color = _generalColor;

SetSettingsOfOutline(_cubeSkins[i].GetComponent<Outline>());

_cubeSkins[i].GetComponent<Outline>().enabled = true;

        SaveEquippedSkinCube = _cubeSkins[i];
    }
}

}

void Update()
{
    if ((Input.GetMouseButtonDown(0) &&
!BlockTouchesWhenPopUpActive) || (Input.touchCount > 0 &&
Input.GetTouch(0).phase == TouchPhase.Began &&
!BlockTouchesWhenPopUpActive))
    {
        Ray ray =
Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit, 100.0f))
        {
            if (hit.collider != null)
            {
                _hitedObject =
hit.collider.gameObject.GetComponent<CubeSkinInfo>();
                if (_hitedObject.unlockMethod ==
UnlockMethod.Record)
                {
                    if (IsUnlocked(_hitedObject) &&
CheckForClickOnAlreadySelectedCube(_hitedObject))
                    {

GetComponent<AudioSource>().PlayOneShot(_cubeSelectionSound);
                    Instantiate(_cubeSelectionFX,
_hitedObject.gameObject.GetComponent<Transform>().position,
Quaternion.identity);
                    PlayerPrefs.SetInt("EquippedCube",
_hitedObject.SkinId);

ChangeGUIofEquippedSkin(_hitedObject);
                }
            }
            else if (_hitedObject.unlockMethod ==
UnlockMethod.Gems)
            {
                if (IsUnlocked(_hitedObject) &&
CheckForClickOnAlreadySelectedCube(_hitedObject))

GetComponent<AudioSource>().PlayOneShot(_cubeSelectionSound);
            }
        }
    }
}

```

```

Instantiate(_cubeSelectionFX,
_hitedObject.gameObject.GetComponent<Transform>().position,
Quaternion.identity);

PlayerPrefs.SetInt("EquippedCube",
_hitedObject.SkinId);

ChangeGUIofEquippedSkin(_hitedObject);
    }
    else if(!IsUnlocked(_hitedObject))
    {
        if
(PlayerPrefs.GetInt("AmountOfGems") >= _hitedObject.NeedToUnlock)
        {

GetComponent<AudioSource>().PlayOneShot(_pressingButtonSound);
        _infoAboutBuyingTextUI.text =
"Buy <color=#07931F>" + _hitedObject.Name.text + "</color> cube for
<color=#6BD6FF>" + _hitedObject.NeedToUnlock + "</color> gems?";
        BlockTouchesWhenPopUpActive =
true;

SaveEquippedSkinCube.GetComponent<Outline>().enabled = false;

_buyConfirmationPopUp.SetActive(true);

StartCoroutine(WaitingForPopUpResult(_hitedObject));
        }
    }
}

private bool CheckForClickOnAlreadySelectedCube(CubeSkinInfo
Cube)
{
    if (Cube == SaveEquippedSkinCube)
        return false;
    return true;
}

IEnumerator WaitingForPopUpResult(CubeSkinInfo CubeSkin)
{
    while (CurrentStateOfPopUp == StateOfPopUp.None)
    {
        yield return null;
    }

    if (CurrentStateOfPopUp == StateOfPopUp.Yes)
    {
        PlayerPrefs.SetInt("AmountOfGems",
PlayerPrefs.GetInt("AmountOfGems") - CubeSkin.NeedToUnlock);

```

```

        _amountOfGemsTextUI.GetComponent<Text>().text =
PlayerPrefs.GetInt("AmountOfGems").ToString();
        PlayerPrefs.SetInt(CubeSkin.gameObject.name, 1);
        ChangeGUIofEquippedSkin(CubeSkin);
        PlayerPrefs.SetInt("EquippedCube", CubeSkin.SkinId);
        CurrentStateOfPopUp = StateOfPopUp.None;
        BlockTouchesWhenPopUpActive = false;
        _buyConfirmationPopUp.SetActive(false);
    }
    else if(CurrentStateOfPopUp == StateOfPopUp.No)
    {
        CurrentStateOfPopUp = StateOfPopUp.None;
        BlockTouchesWhenPopUpActive = false;
        _buyConfirmationPopUp.SetActive(false);
    }

    SaveEquippedSkinCube.GetComponent<Outline>().enabled = true;
}

public void SetStateYesOfPopUp()
{
    GetComponent<AudioSource>().PlayOneShot(_pressingButtonSound);
    CurrentStateOfPopUp = StateOfPopUp.Yes;
}

public void SetStateNoOfPopUp()
{
    GetComponent<AudioSource>().PlayOneShot(_pressingButtonSound);
    CurrentStateOfPopUp = StateOfPopUp.No;
}

private bool IsUnlocked(CubeSkinInfo cubeSkinInfo)
{
    if (cubeSkinInfo.unlockMethod == UnlockMethod.Gems)
    {
        if (PlayerPrefs.GetInt(cubeSkinInfo.name) == 1)
            return true;
    }
    else if (cubeSkinInfo.unlockMethod ==
UnlockMethod.Record)
    {
        if (PlayerPrefs.GetInt("score") >=
cubeSkinInfo.NeedToUnlock)
            return true;
    }
    return false;
}

private void ChangeGUIofEquippedSkin(CubeSkinInfo hitedCube)
{
    hitedCube.Sign.text = "EQUIPPED";
}

```

```

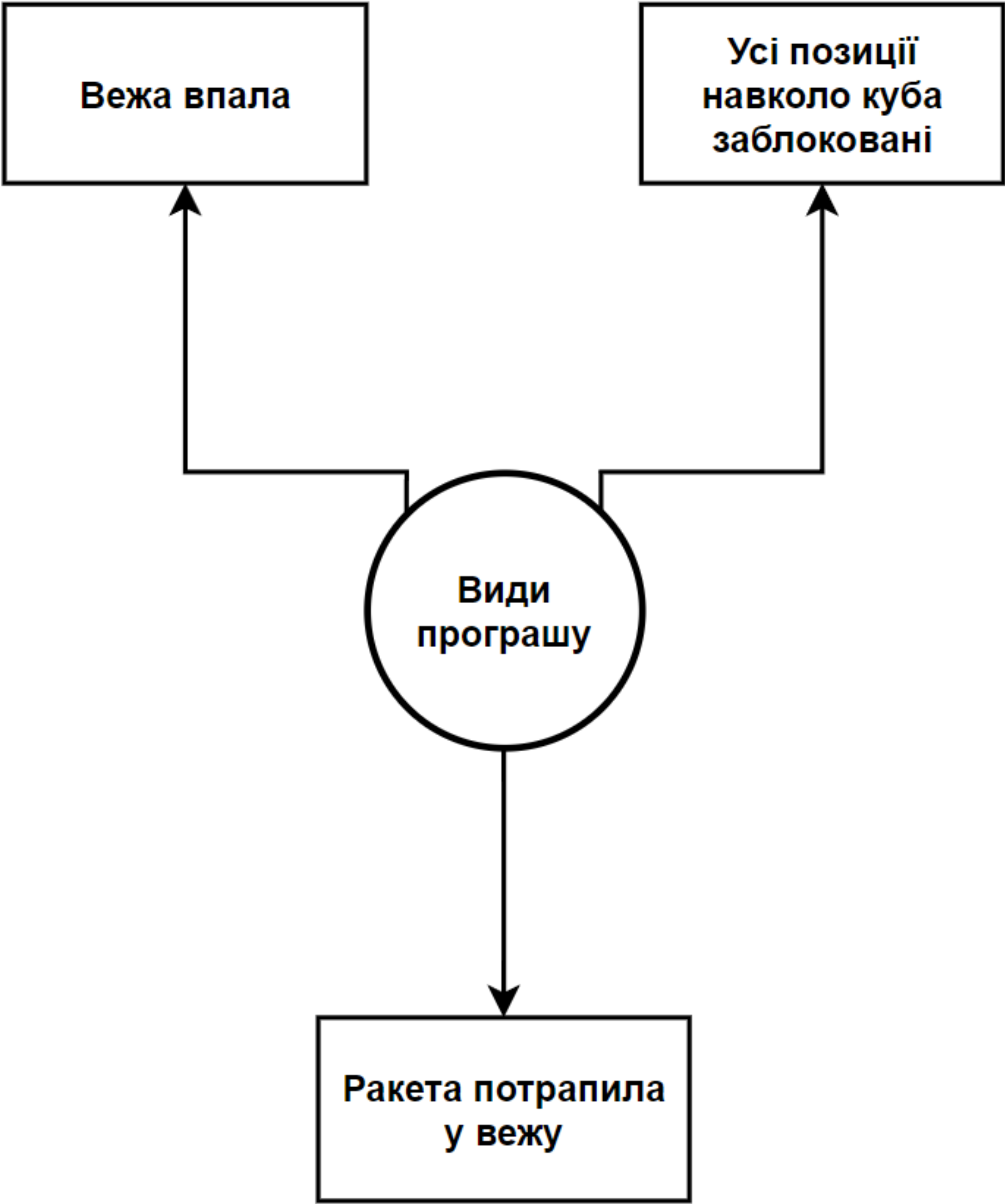
        hitedCube.Sign.color = _generalColor;

SetSettingsOfOutline(hitedCube.GetComponent<Outline>());
        hitedCube.GetComponent<Outline>().enabled = true;
        SaveEquippedSkinCube.GetComponent<Outline>().enabled =
false;
        SaveEquippedSkinCube.Sign.text = "";
        SaveEquippedSkinCube = hitedCube;
    }

    private void SetSettingsOfOutline(Outline outline)
    {
        outline.OutlineColor = _generalColor;
        outline.OutlineWidth = 9f;
        outline.OutlineMode = Outline.Mode.OutlineVisible;
    }
}

```

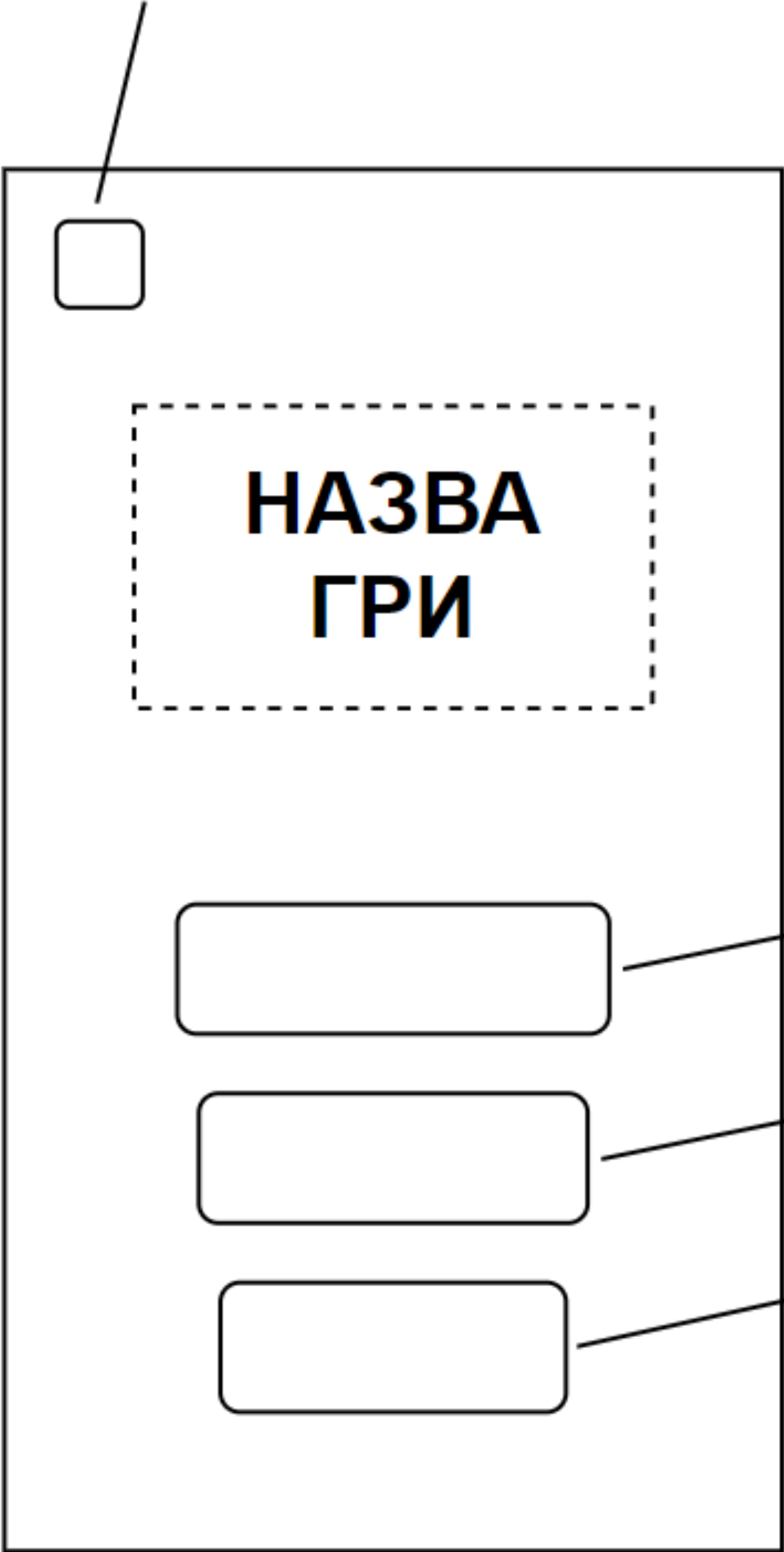
Схема видів програшу



					13.02070849.51919 ПЛ1				
					Розробка мобільної аркадної гри для ос андроїд.Схема видів програшу.				
Зам.	Лист.	№ докум.	Підп.	Дата					
Розроб.		Сушко О.С.							
Перев.		Грушко С.С.							
Т.контр.									
Н.контр.		Щербак Н.В.			НУ «Запорізька політехніка» КНТ-519				
Затв.		Кудерметов Р.К.							

Схема інтерфейсу сцени меню

Кнопка налаштувань



Кнопка "Почати гру"

Кнопка "Перейти у
Магазин"

Кнопка "Про гру"

					13.02070849.51919 ПЛ2				
					Розробка мобільної аркадної гри для ос андроїд. Схема інтерфейсу сцени меню.	Лист.	Маса	Масшт.	
Зам.	Лист.	№ докум.	Підп.	Дата					
Розроб.		Сушко О.С.							
Перев.		Грушко С.С.							
Т.контр.						Лист 1	Листів 1		
						НУ «Запорізька політехніка» КНТ-519			
Н.контр.		Щербак Н.В.							
Затв.		Кудерметов Р.К.							

Схема інтерфейсу ігрової сцени

Кнопка паузи

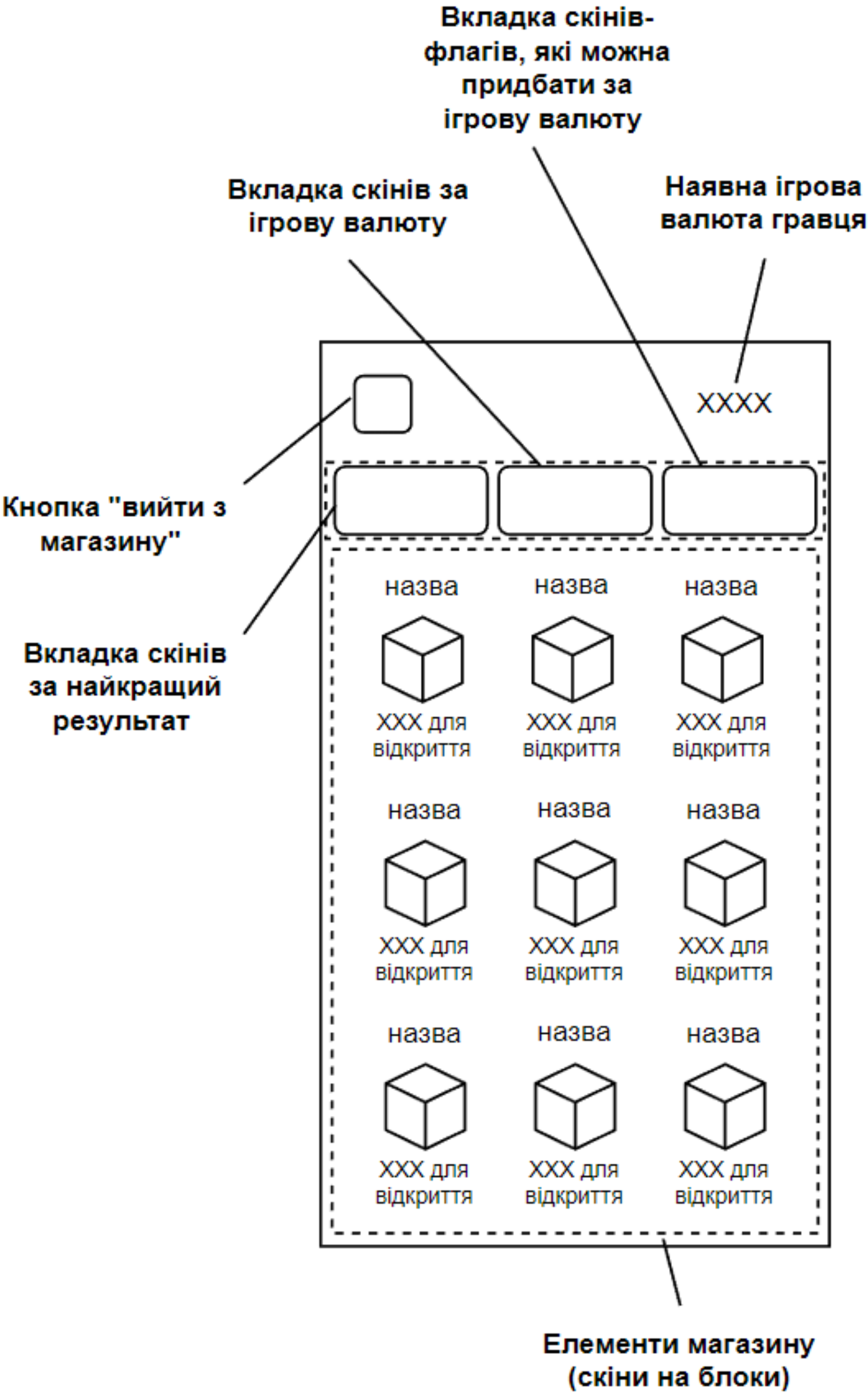
Лічильник поставлених блоків у поточній грі, та найкращий результат поставлених блоків за всі ігри

Лічильник зібраної ігрової валюти у поточній грі

Ігрове поле

					13.02070849.51919 ПЛЗ					
					Розробка мобільної аркадної гри для ос андроїд. Схема інтерфейсу ігрової сцени.			Лист.	Маса	Масшт.
Зам.	Лист.	№ докум.	Підп.	Дата						
Розроб.		Сушко О.С.								
Перев.		Грушко С.С.								
Т.контр.										
					Лист 1		Листів 1			
Н.контр.		Щербак Н.В.			НУ «Запорізька політехніка» КНТ-519					
Затв.		Кудерметов Р.К.								

Схема інтерфейсу сцени магазину



					13.02070849.51919 ПЛ4			
					Розробка мобільної аркадної гри для ос андроїд. Схема інтерфейсу сцени магазину.			
Зам.	Лист.	№ докум.	Підп.	Дата				
Розроб.		Сушко О.С.						
Перев.		Грушко С.С.						
Т.контр.								
Н.контр.		Щербак Н.В.						
Затв.		Кудерметов Р.К.						
					Лист 1	Маса	Листів 1	Масшт.
					НУ «Запорізька політехніка» КНТ-519			