МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Мобільна гра на основі 3D Unity

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма **«Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-41
_____Авагян А.Г_____
(прізвище та ініціали)

Керівник _____ Краснощок В.М. _____
(прізвище та ініціали)
_____
(науковий ступінь, звання)

Унікальність тексту – 95%

Випускна кваліфікаційна робота бакалавра допущена до захисту
Рішенням кафедри *прикладних інформаційних систем*
Протокол № від р.
зав. кафедри _____ Плескач В. Л.

Київ – 2024

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	01.11.2023	
2.	Видача завдання кваліфікаційної роботи бакалавра	15.11.2023	
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	27.11.2023	
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.12.2023	
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.12.2023	
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	31.12.2023	
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2024	
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	29.03.2024	
9.	Подання роботи у першому варіанті	**29.04.2024**	
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	02.05.2024	
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**15.05.2024**	
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2024	
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	13.06.2024	
14.	Захист кваліфікаційної роботи бакалавра	17.06.2024	

Здобувач вищої освіти _____

(підпис)

Керівник _____

(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою - англійською)	1
Зміст	1
Вступ	1
1	10
2	15
3	16
Висновки	1
Перелік використаних джерел	3
Додатки	7

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3">ДП XXXX 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td><td colspan="3"></td></tr>
<tr><td>Розробн.</td><td>Авагян А.Г.</td><td>  </td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Краснощок В.М.</td><td>  </td><td></td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td></td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав. каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ(РЕФЕРАТ)

Дипломна робота: 59 с., 38 рис., 52 джерел, 3 дод.

Метою кваліфікаційної роботи бакалавра є гра на основі застосунку розроблена засобами Unity

Для досягнення мети роботи потрібно вирішити такі **завдання**:

- описати історію ігор та провести аналіз ринку ігор;
- дослідити сучасний підхід до створення ігор;
- здійснити проектування, розроблення та впровадження мобільної гри з використанням сучасних технологій;

Об’єкт дослідження

Процеси забезпечення ігор

Предмет дослідження

Програмно-технічні засоби побудови застосунку на основі Unity.

Методи дослідження.

- Дослідження наявних програмно-технічних рішень у сфері «розробка ігор».
- Описовий метод для формування структури та завдань при розробці.

Ключові слова: мобільна гра, Unity

ANOTATION (ABSTRACT)

Thesis: 59 p., 38 fig., 52 sources, 3 appendices.

The aim of the thesis is an application-based game developed by Unity.

To achieve the goal of the work, you need to solve the following **tasks**:

- describe the history of games and analysis of the game market;
- explore the modern approach to game development;
- design, develop and implement a mobile game using modern technologies;

Object of study

Mobile game development processes

Subject of study

Software and hardware tools for building an application based on Unity.

Research methods

- Research of existing software and hardware solutions in the field of game development.
- Descriptive method for forming the structure and tasks in development.

Keywords: mobile game, Unity

ЗМІСТ

ВСТУП

Мобільні ігри дедалі глибше проникає в наше життя – якщо раніше це були лише розваги для дітей і «вбивці» часу то зараз в ігри грають не лише діти, а й дорослі, та не завжди для розваг, а все частіше для розвитку реакцій, пам'яті та для навчання.

Актуальність теми пов’язана зі стрімким розвитком технологій в мобільному сегменті та їх поширення серед населення. Зараз телефони є у всіх, від дітей до дорослих

Метою кваліфікаційної роботи бакалавра є гра на основі застосунку засобами Unity

Для досягнення мети роботи потрібно вирішити такі **завдання**:

- описати історію ігор та провести аналіз ринку ігор;
- дослідити сучасний підхід до створення ігор;
- здійснити проектування, розроблення та впровадження комп’ютерної гри з використанням сучасних технологій;

Об’єкт дослідження

Процеси забезпечення ігор

Предмет дослідження

Програмно-технічні засоби побудови застосунку на основі Unity.

Методи дослідження.

- Дослідження наявних програмно-технічних рішень у сфері «розробка ігор».
- Описовий метод для формування структури та завдань при розробці.

Практичне значення отриманих результатів полягає в тому, що розроблена гра може допомогти гравцям розвинути такі важливі навички як реакція, швидкість прийняття рішень, підвищувати настрій та відволікати від стресових ситуацій, а також допоможе у просуванні жанру раннер, що призведе до подальшого розвитку.

Структура роботи: Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, розподілених на підрозділи та висновку.

РОЗДІЛ 1
ІСТОРІЯ ІГОР ТА АНАЛІЗ РИНКУ ІГОР

1.1 Історія індустрії відеоігор

Історія індустрії відеоігор - це захоплююча подорож від скромних початків до сучасного монстра розвагової галузі. Вона почалася у 1950-х і 1960-х роках, коли перші відеоігри з'явилися на екранах. Однією з найвідоміших з них є "Spacewar!", створена в 1962 році на Массачусетському технологічному інституті.

Протягом наступних десятиліть індустрія почала швидко розвиватися. У 1970-х виходять перші аркадні ігри, такі як "Pong" від Atari (рис.1.1), яка стала справжньою сенсацією. У 1980-х настає "золотий вік" відеоігор з виходом консолей, таких як Nintendo Entertainment System (NES) і Sega Master System. Цей період також побачив народження таких легендарних ігор, як "Super Mario Bros.", "The Legend of Zelda", "Pac-Man" і "Tetris" [2].

Рисунок 1.1 – Ігрова приставка “Atari”

У 1990-2000-х роках відеоігри стають більш складними і виразними завдяки покращенню технологій. Ігрові консолі стають більш потужними, а 3D-графіка стає стандартом. З'являються такі серії, як "Final Fantasy", "Metal Gear Solid" (рис. 1.2), "Resident Evil" та "Pokémon".

Рисунок 1.2 – Гра "Metal Gear Solid"

У 2000-х роках індустрія ігор стає масштабним глобальним бізнесом з мільярдами доларів обороту. Одним із суттєвих подій цього періоду був випуск консолі PlayStation 2, яка стала найбільш успішною консоллю в історії. Також зростає популярність персональних комп'ютерів як платформи для гри.

З настанням 2010-х років ігрова індустрія продовжує еволюціювати. З'являються нові технології, такі як віртуальна реальність (VR) і доповнена реальність (AR), що відкривають нові можливості для геймдеву. Інді-ігри, розроблені невеликими командами або навіть однією людиною, також стають дедалі популярнішими завдяки доступності платформ для розробки та цифрових дистрибуційних магазинів [24].

Сьогодні індустрія відеоігор є однією з найбільш прибуткових індустрій розваг, яка постійно росте і привертає увагу мільйонів гравців по всьому світу [1].

1.2 Ринок мобільних ігор

Розглянемо більш детально ринок мобільних ігор та його складові. Основні платформи для мобільних ігор становлять значний аспект геймінгової індустрії, кожна з яких має свої унікальні особливості та переваги. iOS, як відомо, виступає під прапором Apple App Store (рис. 1.3), який славиться великим вибором ігор у

різних жанрах. Цей магазин не лише пропонує багато гарних ігор для користувачів iPhone та iPad, але й відомий своєю якістю, що сприяє розвитку високоякісних ігрових проектів з привабливою графікою та захоплюючими механіками.

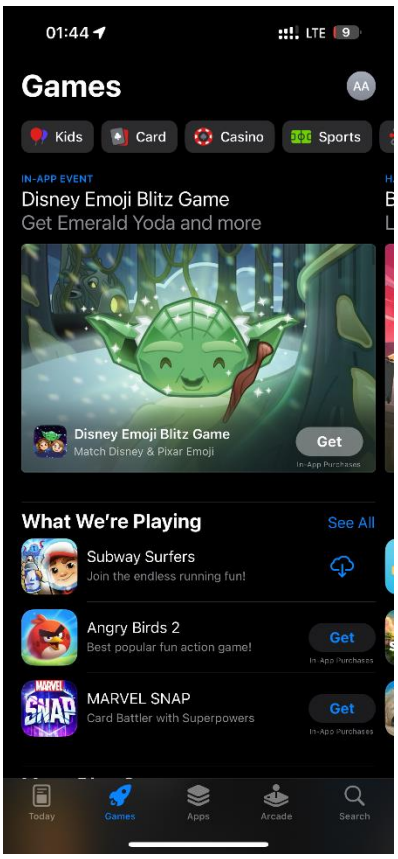

Рисунок 1.3 – Магазин застосунків Apple App Store

З іншого боку, Android, який розповсюджується через Google Play Store (рис. 1.4), володіє своїми особливостями. Цей магазин також має великий асортимент ігор, що охоплюють широкий спектр жанрів, від простих інді-проектів до амбітних ігрових творінь. Однією з головних переваг Android є його відкрита природа, яка сприяє більшій доступності для розробників, що відображається в ширшому спектрі ігрових пропозицій та більшій кількості креативних підходів до геймдеву.

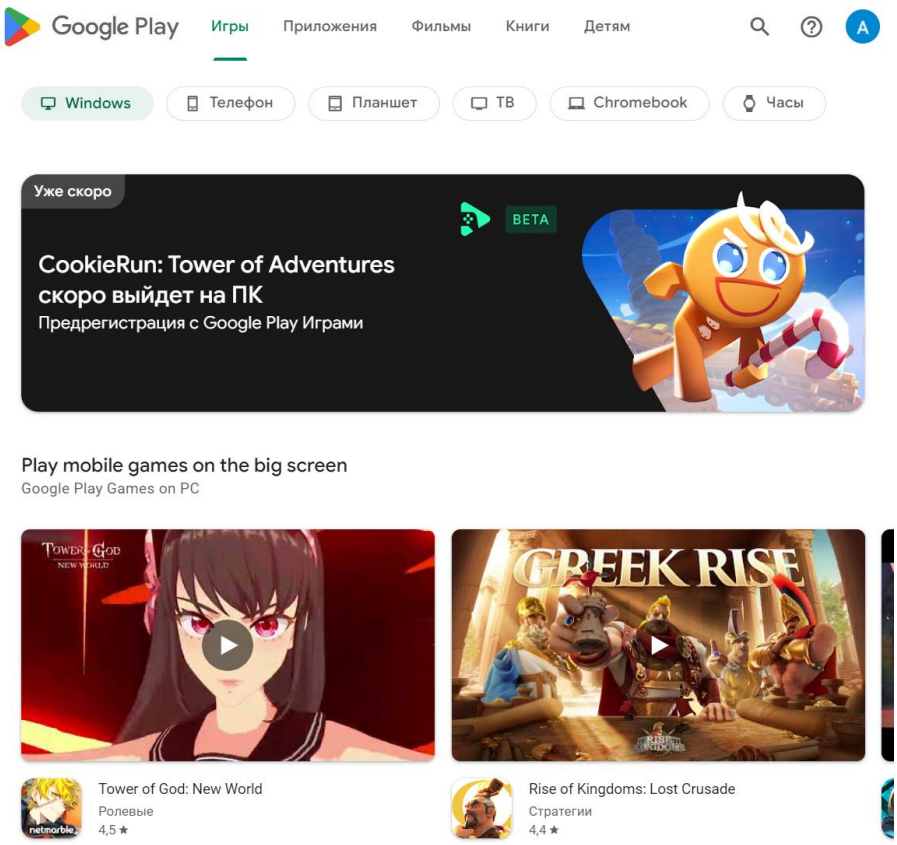

Рисунок 1.4 – Магазин застосунків Google Play Store

Крім того, існують інші, менш популярні платформи, такі як Windows Phone, які, хоч і не мають такого великого впливу на ринок, проте все ж відіграють свою роль у геймінговій екосистемі. Мобільні версії ігор, доступні на планшетах та інших пристроях, також важливі для розвитку геймінгу, забезпечуючи широкий спектр можливостей для гравців у всій своїй різноманітності [51].

Популярні жанри мобільних ігор, приклади яких ми можем побачити на головній сторінці магазинів (рис. 1.5), включають кілька ключових категорій, кожна з яких має свої унікальні особливості та привабливість для гравців. Аркади завжди були популярним жанром на мобільних пристроях завдяки своїм простим правилам і швидким геймплейним сесіям. Вони ідеально підходять для коротких перерв або чекання, дозволяючи гравцям швидко зануритися у гру і отримати задоволення без потреби витрачати багато часу на навчання або роздуми.

Головоломки є ще одним популярним жанром, які зазвичай вимагають від гравця логічного мислення та розв'язання різних завдань. Ці ігри можуть варіюватися від простих головоломок з розташуванням елементів до складних, багаторівневих головоломок, які кидають виклик навіть найдосвідченішим гравцям. Вони часто допомагають розвивати когнітивні навички та забезпечують відчуття досягнення після розв'язання кожної задачі.

Стрілялки залучають гравців можливістю керувати персонажем і стріляти по ворогах або іншим об'єктам. Цей жанр включає як класичні 2D стрілялки, де основна увага зосереджена на швидкості реакції та точності, так і сучасні тривимірні шутери, які часто пропонують більш глибокий ігровий процес та багатокористувацькі режими. Стрілялки часто мають захоплюючі сюжети та інтенсивні бойові сцени.

Рольові ігри (RPG) на мобільних пристроях зазвичай мають глибокий сюжет, розвинену систему розвитку персонажів і інтенсивний геймплей. Цей жанр охоплює як класичні японські RPG, що часто характеризуються ретельно продуманими світами та персонажами, так і сучасні ігри з відкритим світом, де гравці можуть досліджувати великі території, виконувати різноманітні завдання та взаємодіяти з іншими персонажами. RPG ігри надають гравцям можливість зануритися у фантастичні світи та відчути себе частиною епічних пригод.

Стратегії вимагають від гравця розробки стратегій та тактик для досягнення мети. Цей жанр включає як класичні стратегії в реальному часі, де гравцям потрібно швидко приймати рішення та керувати ресурсами, так і пошагові стратегії, які дозволяють більш детально планувати свої дії. Стратегічні ігри часто стимулюють аналітичне мислення та здатність до передбачення дій супротивника. Вони надають гравцям можливість керувати арміями, будувати імперії та змагатися за домінування у вигаданих світах.

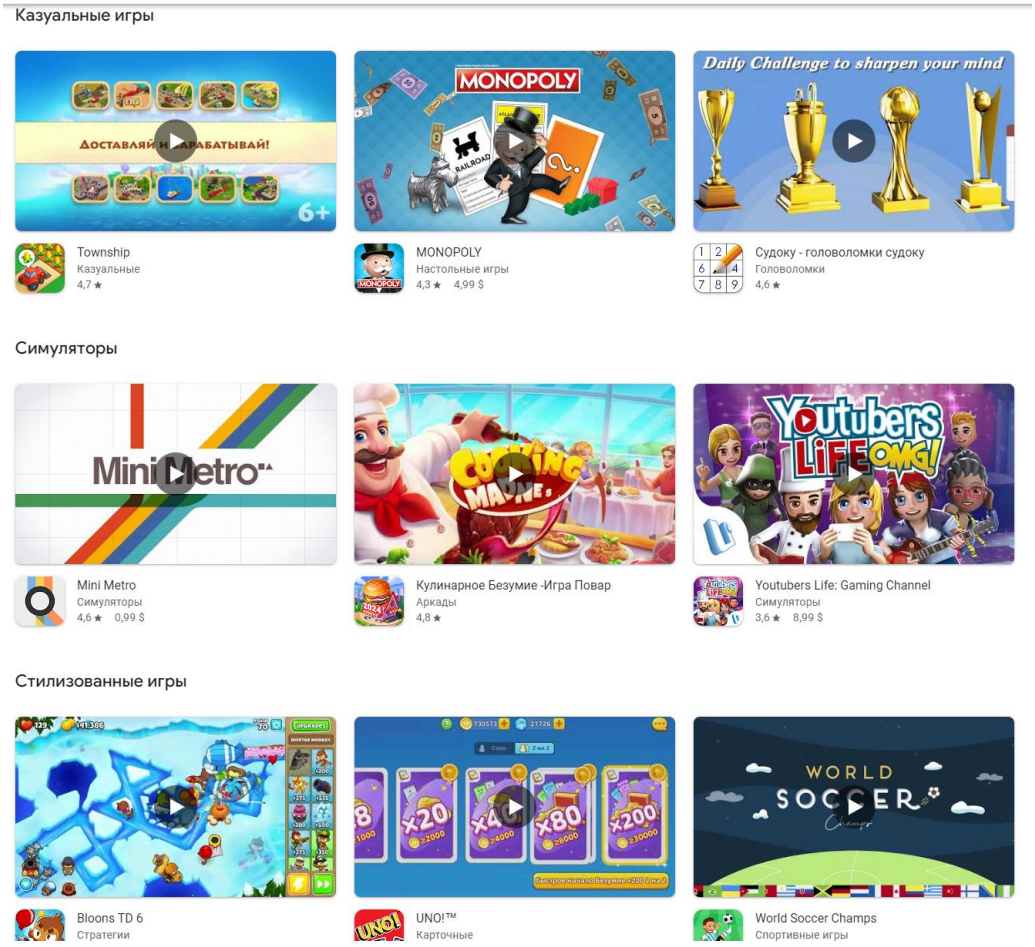

Рисунок 1.5 – Рекомендації ігор в різних категоріях від Google Play Store

Монетизація мобільних ігор здійснюється кількома основними способами. Одним із них є безкоштовні ігри з внутрішніми покупками, коли ігри пропонуються безкоштовно, але містять внутрішні покупки, що дозволяють гравцям отримувати різноманітні бонуси, вдосконалення або прискорення гри. Така модель часто стимулює гравців витрачати гроші на поліпшення свого ігрового досвіду, що робить її дуже популярною серед розробників. Внутрішні покупки можуть включати нові рівні, персонажів, предмети або косметичні покращення, що не впливають на сам ігровий процес, але підвищують задоволення від гри [25].

Іншим способом монетизації є платні ігри, які продаються за фіксовану ціну перед тим, як гравці отримають до них доступ. Ця модель часто використовується для преміум-ігор з високоякісною графікою, складним сюжетом або інноваційним геймплеєм. Платні ігри зазвичай не містять реклами

або внутрішніх покупок, що робить їх привабливими для гравців, які бажають уникнути постійних витрат або відволікаючих факторів під час гри.

Реклама є ще одним важливим методом монетизації мобільних ігор, який включає відеоролики або банери, що відображаються під час гри. Гравці можуть переглядати рекламу для отримання різних нагород, таких як додаткові життя, ігрова валюта або прискорення проходження рівнів. Розробники можуть заробляти гроші за кожний перегляд або клік по рекламі, що дозволяє їм підтримувати безкоштовний доступ до гри для широкої аудиторії.

Ще одним способом монетизації є підписка, коли деякі ігрові сервіси пропонують підписку, що надає гравцям доступ до великого каталогу ігор за фіксовану щомісячну плату. Ця модель популярна серед гравців, які бажають мати доступ до різноманітних ігор без необхідності купувати кожну з них окремо. Підписка може також включати ексклюзивний контент, ранній доступ до нових ігор або відсутність реклами, що додає додаткову цінність для користувачів.

Ринок мобільних ігор є великим і швидкозростаючим сегментом індустрії розваг. З розвитком технологій та збільшенням кількості користувачів смартфонів цей ринок буде продовжувати зростати і привертати увагу як гравців, так і розробників. Із розвитком штучного інтелекту, віртуальної реальності та інших технологій майбутнє мобільних ігор обіцяє бути ще більш захоплюючим та інноваційним.

1.3 Ігри раннери

Жанр "раннер" у світі відеоігор відіграє значну роль, привертаючи увагу гравців різного віку і рівня досвіду. Це один з найпопулярніших жанрів мобільних ігор, який зазвичай відомий своєю простотою, швидкістю та невимушеністю. В основі раннерів лежить постійний рух головного персонажу вперед, при цьому гравець відповідає за його уникнення перешкод та збирання бонусів.

Ігри жанру раннер характеризуються кількома основними властивостями. По-перше, вони пропонують гравцеві безкінечно генеровані рівні або протягом певного періоду часу. По-друге, гравцеві потрібно швидко реагувати на змінні умови та перешкоди, керуючи персонажем. Крім того, ігри раннерів зазвичай мають прості механіки керування, такі як торкання або рухи вліво/вправо. Ще однією особливістю є можливість збирати різноманітні бонуси на шляху, що допомагають підвищити рахунок або дати додаткові можливості. Хоча гра не має чіткої кінцевої мети, багато раннерів включають завдання або досягнення, які гравець може спробувати досягти.

Щодо популярності, жанр раннерів завжди має свою аудиторію. Багато гравців цінують його за можливість грати в будь-який момент часу, без потреби великої зосередженості чи інвестування в час.

Деякі з найпопулярніших ігор жанру раннер включають Temple Run, Subway Surfers (рис. 1.6), Jetpack Joyride, Canabalt і Into the Dead. У грі Temple Run гравці керують археологом, який втікає від мавп, шукаючи скарби. У Subway Surfers гравець управляє підлітком, який втікає від поліції, пробігаючи по вулицях міста та збираючи монети. У Jetpack Joyride гравці керують персонажем з реактивним ранцем, літаючи крізь лабіринти та уникаючи перешкод. Гра Canabalt є однією з перших у жанрі, де гравець просто пробігає вперед, уникаючи руйнувань будівель і інших перешкод. В Into the Dead гравець виживає в апокаліптичному світі, уникаючи зомбі на своєму шляху [45].

Рисунок 1.6 – Гра “Subway surfers”

Популярність ігор жанру раннер постійно зростає, особливо серед мобільних гравців. Цей жанр приваблює широке коло аудиторії через свою простоту, доступність та захоплюючий геймплей.

Розглянемо кілька факторів, які впливають на популярність раннерів. Мобільність є одним із ключових факторів, оскільки більшість раннерів доступні на мобільних пристроях, таких як смартфони і планшети. Це робить їх легко доступними для гравців у будь-який час і в будь-якому місці, ідеальними для коротких ігрових сесій під час перерви на роботі, у школі або в дорозі. Простота та легкість в оволодінні також сприяють їхній популярності: багато раннерів

мають дуже прості механіки керування, які може освоїти навіть початківець. Це дозволяє новачкам швидко вступити в гру, а досвідченим гравцям - швидко отримати задоволення від гри. Соціальність деяких раннерів, зокрема можливість ділитися своїми досягненнями у соціальних мережах або змагатися з друзями, стимулює здорову конкуренцію та додатково мотивує гравців. Стійка залученість забезпечується завдяки безкінечному формату гри, який може залучати гравців протягом тривалого часу без необхідності великої кількості нового контенту. Регулярні оновлення з новими рівнями, персонажами або ігровими елементами також підтримують інтерес гравців. Відмінне засвоєння основних елементів, таких як уникання перешкод та збирання бонусів, робить раннери привабливими для гравців усіх вікових груп і різного рівня досвіду.

Загалом, ігри жанру раннер користуються великою популярністю, оскільки вони пропонують простий, але захоплюючий геймплей, який може відповісти потребам будь-якого гравця. Їхня мобільність, доступність і здатність залучати гравців на тривалий час роблять їх одними з найбільш впливових ігрових жанрів у світі [38].

РОЗДІЛ 2
СУЧАСНИЙ ПІДХІД ДО СТВОРЕННЯ ІГОР

2.1 Сутність і поняття ігрового двигуна

Ігровий двигун — це складне програмне забезпечення, яке служить фундаментом для розробки відеоігор. Він дозволяє розробникам створювати ігри без необхідності програмувати кожен аспект гри з нуля, надаючи набір готових до використання компонентів та інструментів. Завдяки цьому, розробники можуть зосередитися більше на креативних та інноваційних аспектах проекту [10].

Основні компоненти ігрових двигунів включають кілька важливих елементів, серед яких графічний рендеринг, фізика, анімація, звук, штучний інтелект та скриптинг з ігровою логікою. Графічний рендеринг відіграє ключову роль у створенні візуальної частини гри, забезпечуючи візуалізацію як 3D, так і 2D графіки. Це включає рендеринг текстур, освітлення, тіней та інших візуальних ефектів, які значно підвищують реалістичність і занурення гравців у ігровий світ. Відтворення текстур і складних візуальних ефектів робить гру привабливою і цікавою для користувачів.

Фізика є ще одним важливим компонентом, який включає інтеграцію фізичних двигунів для моделювання реалістичної фізики світу гри. Це стосується таких аспектів, як гравітація, зіткнення об'єктів та інші фізичні взаємодії, які роблять ігровий процес більш правдоподібним і захоплюючим. Реалістичне моделювання фізичних властивостей дозволяє гравцям відчувати себе частиною ігрового світу.

Анімація в ігрових двигунах надає інструменти для створення та управління рухами персонажів і об'єктів. Це дозволяє їм рухатися та взаємодіяти у переконливий та природний спосіб, що значно покращує якість гри. Звук, як важлива складова ігрового досвіду, включає інтеграцію аудіо компонентів для забезпечення звукового оформлення гри. Це охоплює музику, звукові ефекти та діалоги, які допомагають створити атмосферу та занурити гравця в ігровий світ.

Штучний інтелект (ШІ) у ігрових двигунах часто включає системи для управління поведінкою неігрових персонажів (NPC). Це дозволяє їм виконувати складні дії, які виглядають переконливо і додають грі динаміки та непередбачуваності. Скриптинг та ігрова логіка є важливими для налаштування правил гри, подій та поведінки об'єктів. Ігрові двигуни надають мови скриптів або візуальні інтерфейси, що дозволяють розробникам програмувати складні ігрові сценарії, створюючи захоплюючий ігровий досвід для гравців.

Переваги використання ігрових двигунів є численними і включають ефективність розробки, спрощення роботи та багатоплатформність. Ефективність розробки забезпечується завдяки зниженню часу та витрат на розробку. Ігрові двигуни надають готові до використання компоненти та інструменти, які значно прискорюють процес створення ігор. Це дозволяє розробникам зосередитися на творчих аспектах гри, не витрачаючи багато часу на розробку базових елементів. Спрощення роботи досягається через використання візуальних редакторів та інструментів, які усувають необхідність глибоких знань у програмуванні. Розробники можуть створювати складні ігрові механіки та ефекти за допомогою інтуїтивно зрозумілих інтерфейсів, що робить процес розробки доступнішим для широкого кола людей, включаючи тих, хто не має досвіду в програмуванні. Багатоплатформність є ще однією важливою перевагою. Багато ігрових двигунів дозволяють одночасно розробляти ігри для різних платформ, таких як ПК, консолі та мобільні пристрої. Це означає, що розробники можуть створювати гру один раз і адаптувати її для різних платформ без значних додаткових зусиль. Це значно розширює аудиторію, до якої можна донести гру, і підвищує потенційний успіх проекту.

Таким чином, використання ігрових двигунів сприяє підвищенню ефективності та спрощенню процесу розробки ігор, а також забезпечує можливість розробки для різних платформ одночасно. Це робить їх важливим інструментом для сучасних розробників ігор, дозволяючи створювати високоякісні продукти швидше та з меншими витратами.

Серед популярних ігрових двигунів виділяється Unity (рис. 2.1) завдяки своїй легкості використання, багатоплатформній підтримці та великій спільноті. Unreal Engine (рис. 2.2) відомий своєю високою графічною потужністю та гнучкістю для розробки різноманітних ігор. Godot (рис. 2.3), зі своїм відкритим вихідним кодом, забезпечує доступність для розробників з будь-яким бюджетом. Він підтримує як 2D, так і 3D проекти, пропонує інтуїтивно зрозумілий інтерфейс та власну мову скриптів GDScript, яка оптимізована для ігрової логіки. Відкритий характер Godot сприяє широкій підтримці та адаптації спільноти, що дозволяє швидко впроваджувати нові технології та методики [19].

Використання ігрових двигунів дозволяє розробникам сконцентруватися на креативних аспектах гри, тим самим сприяючи інноваціям та підвищенню якості відеоігор.

Рисунок 2.1 – Ігровий рушій Unity

Рисунок 2.2 – Ігровий рушій Unreal Engine

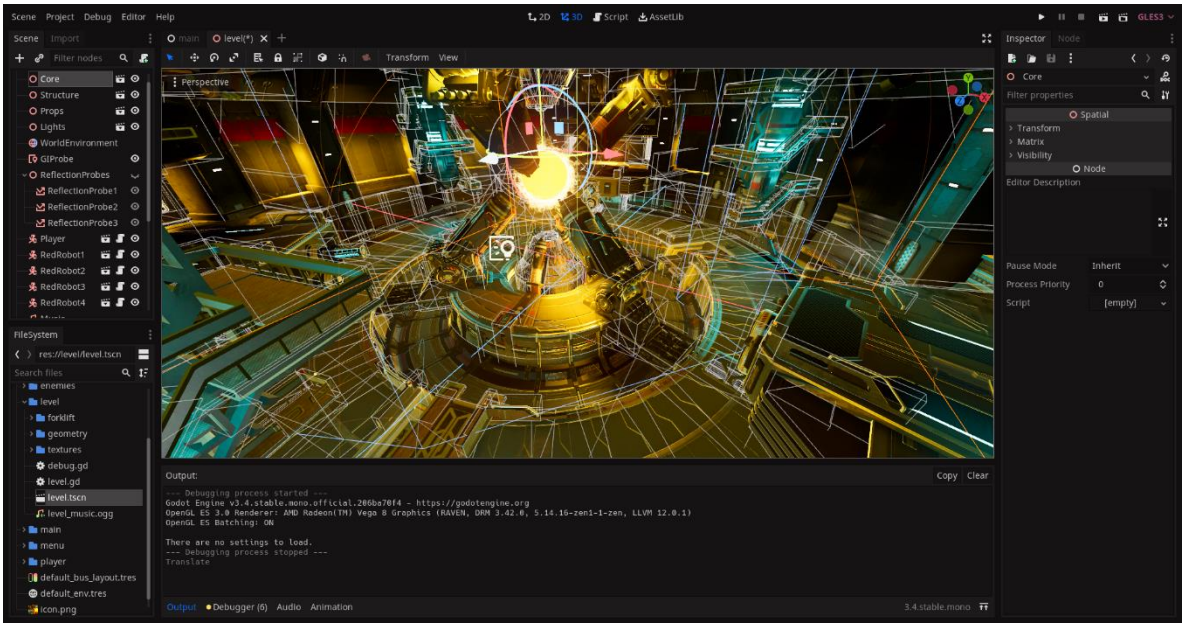

Рисунок 2.3 – Ігровий рушій Godot

2.2 Огляд можливостей рушія Unity

Unity – кросплатформний ігровий рушій та інструмент для розробки відеоігор та застосунків для різних галузей, зокрема автоіндустрії,

кінематографу та мультиплікації, архітектури, доповненої та віртуальної реальності [21].

Наразі, однією з головних переваг Unity є розробка та рендерінг у реальному часі. Ця технологія дозволяє змінювати продукт без необхідності повторної побудови сцени у режимі реального часу, майже у якості один до одного порівняно з офлайн-рендерингом та компіляцією. Unity у реальному часі опрацьовує фізику, відображає фізично коректні матеріали та світло, та багато іншого.

Що ж у випадку Unity означає ігровий рушій? Unity зкомпоновує надані розробником 3д моделі, текстури, аудіо, скрипти, анімації та 2д спрайти у один продукт та дозволяє з ними працювати у власному редакторі.

Чого не може Unity? Ми повинні надавати редактору ассети створені у зовнішніх програмах. Наприклад, для створення 3д моделей необхідно використовувати Maya, ZBrush або Blender. Для створення аудіо Adobe Audition, Logic Pro, Reaper, and Audacity. Хоча, насправді, Unity надає базові засоби для створення найпростіших 3д моделей та редагування 2д спрайтів.

Впорядковування та слідкування за всіма ассетами у вкладинці Project (рис 2.4) надає можливість проводити прев'ю ассетів, проводити з ними базові маніпуляції по переміщенню та копіюванню та створювати більшість наявних у редакторі об'єктів.

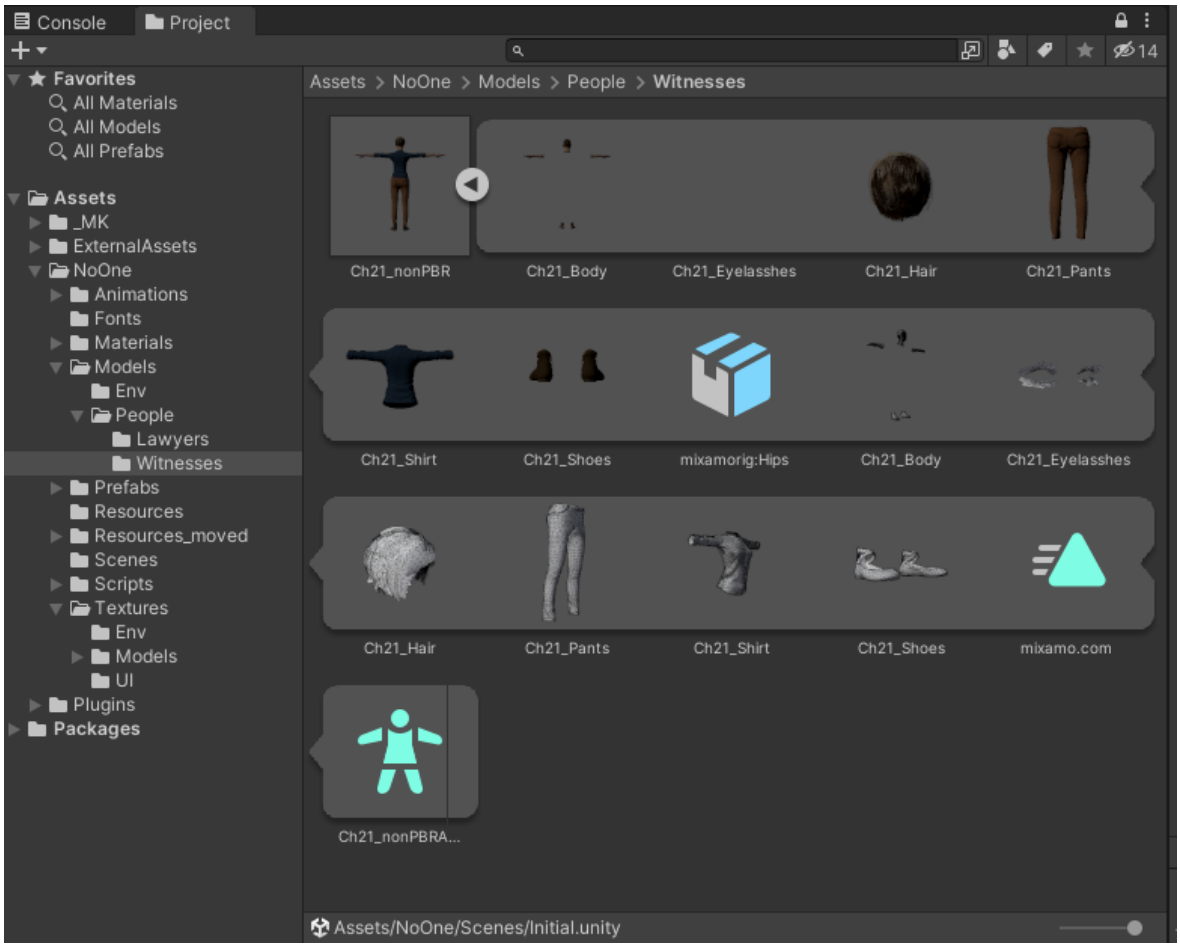

Рисунок 2.4 - Приклад вкладинки Project

Побудова сцен та керування їх об’єктами у вкладинці Hierarchy (рис. 2.5). Також маємо можливість створювати нові об’єкти. Одночасно відкритою може бути тільки одна сцена. Як видно на рисунку 1.2, редактор підтримує вкладені об’єкти, що є важливим при наслідуванні значень компонентів

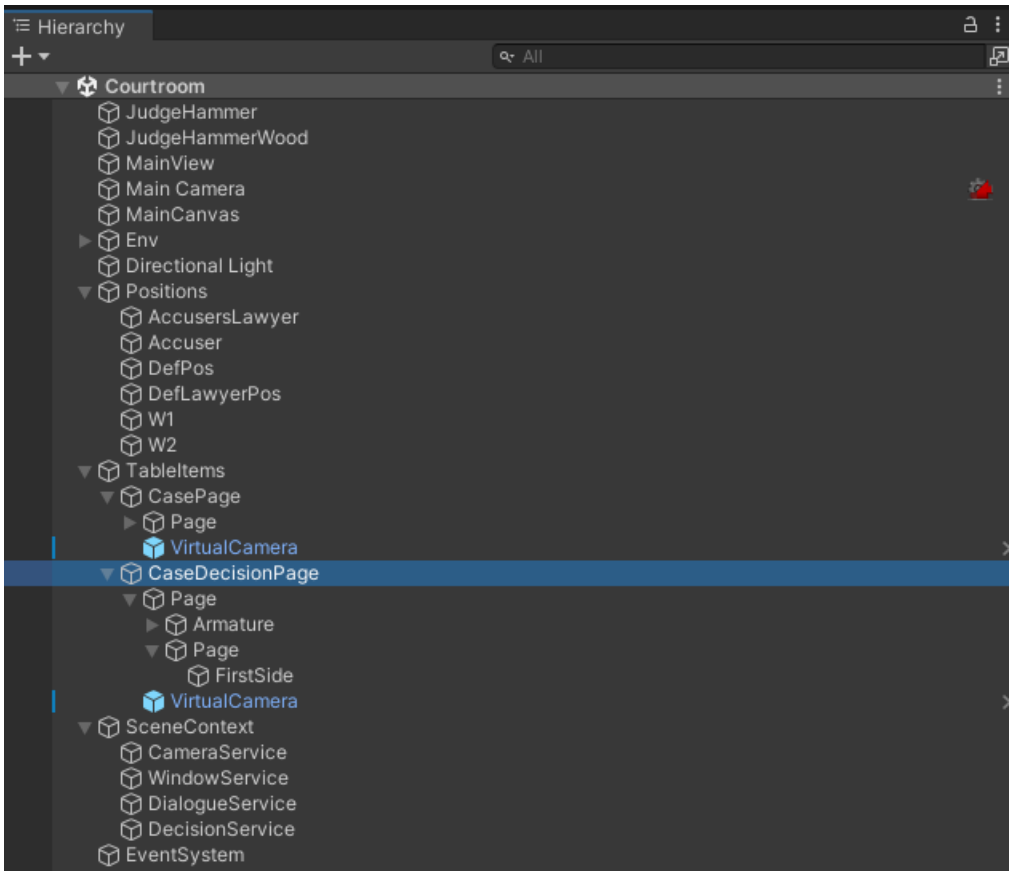

Рисунок 2.5 - Приклад вкладинки Hierarchy з об’єктами сцени

Налаштування та створення властивостей об’єктів як на сцені так і загалом у проекті у вкладинці Inspector (рис. 2.6). Наявна можливість копіювати компоненти та їх значення між об’єктами, а також редагувати ідентичні компоненти відразу у декількох виділених об’єктах. Також дозволяє змінювати властивості під час виконання програми [20].

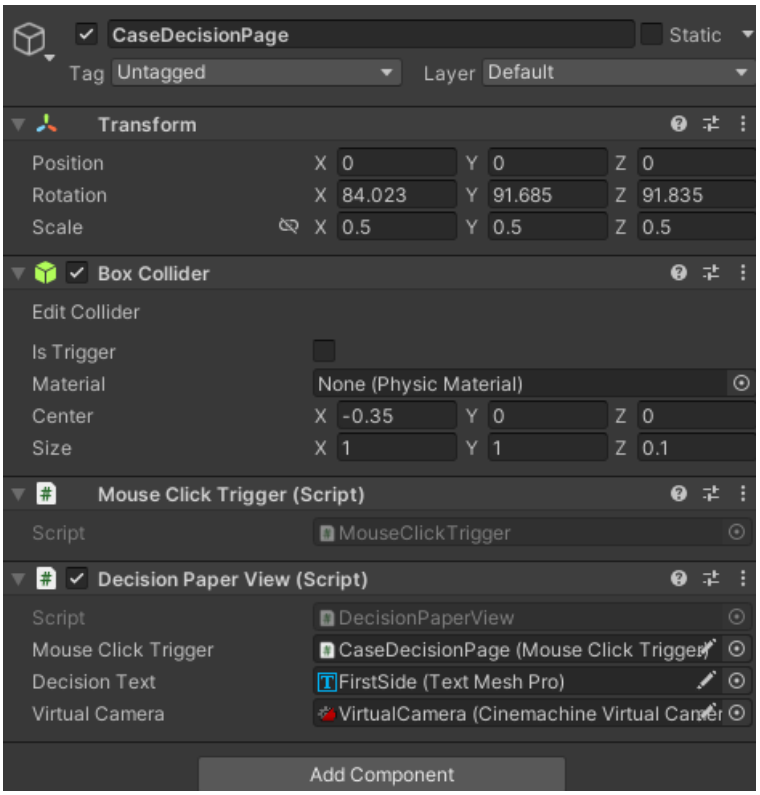

Рисунок 2.6 - Приклад вкладинки Inspector з обраним об’єктом

Розміщення об'єктів на сцені та їх візуальне редагування на вкладинці Scene (рис. 2.7). Із доступних інструментів ми маємо можливість змінювати розмір, положення та поворот обраних об'єктів. Також, наявний інструментарій для руху камери у двух та трьох вимірах з підтримкою ортогонального та ізометричного бачення. Підтримує зміни об'єктів під час виконання програми [3].

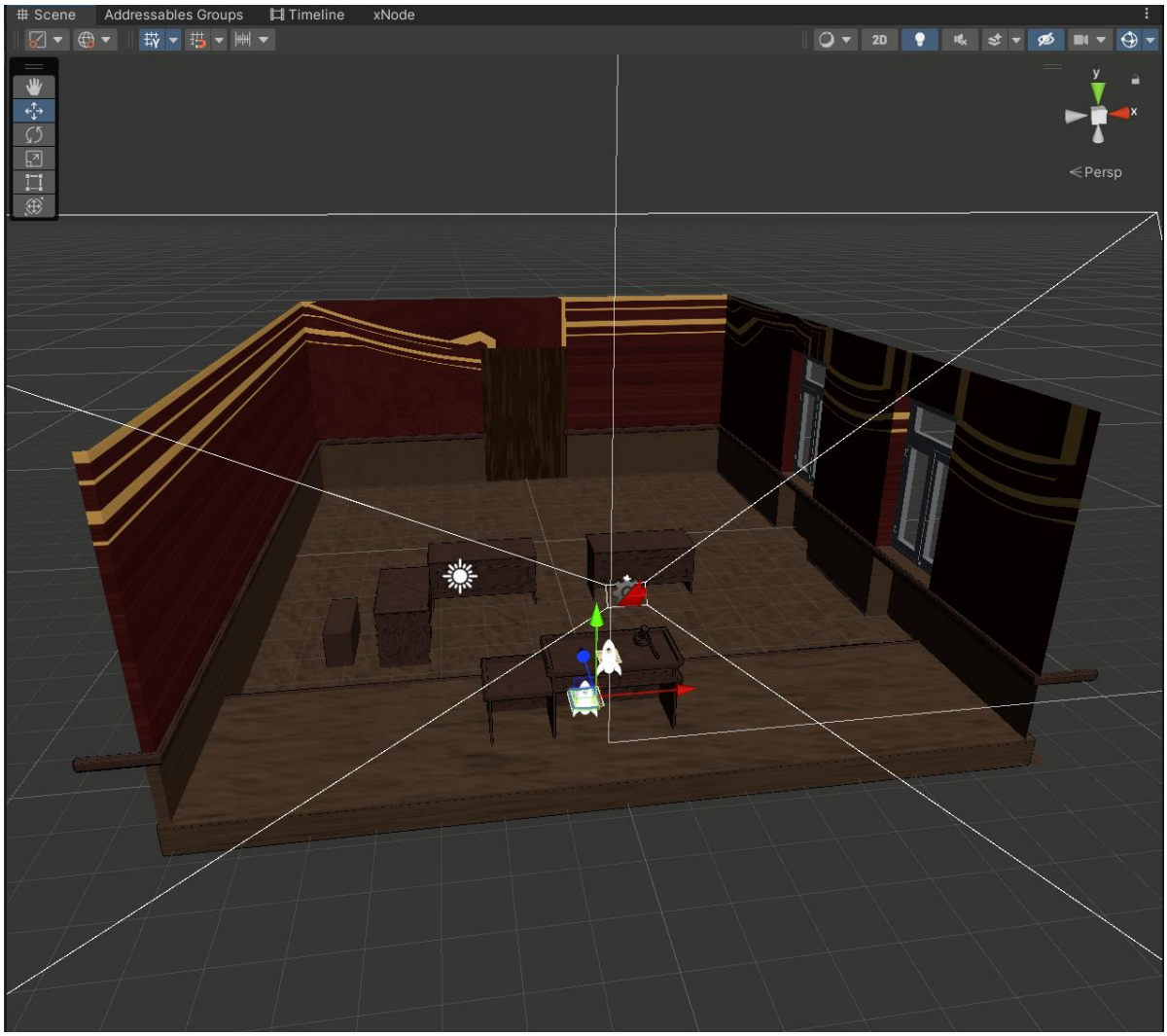

Рисунок 2.7 - Приклад вкладинки Scene з об’єктами інтерфейсу та камерою у режимі 3д

Виконання застосунку у реальному часі на вкладинці Game (рис. 2.8). Надає можливість змінювати розмір та відношення екрана. Під час виконання є можливість отримувати інформацію через консоль, що може відображатись окремим вікном або як маленька стрічка знизу екрана. Відображає зміни, виконані у інших вкладинках редактора [18].

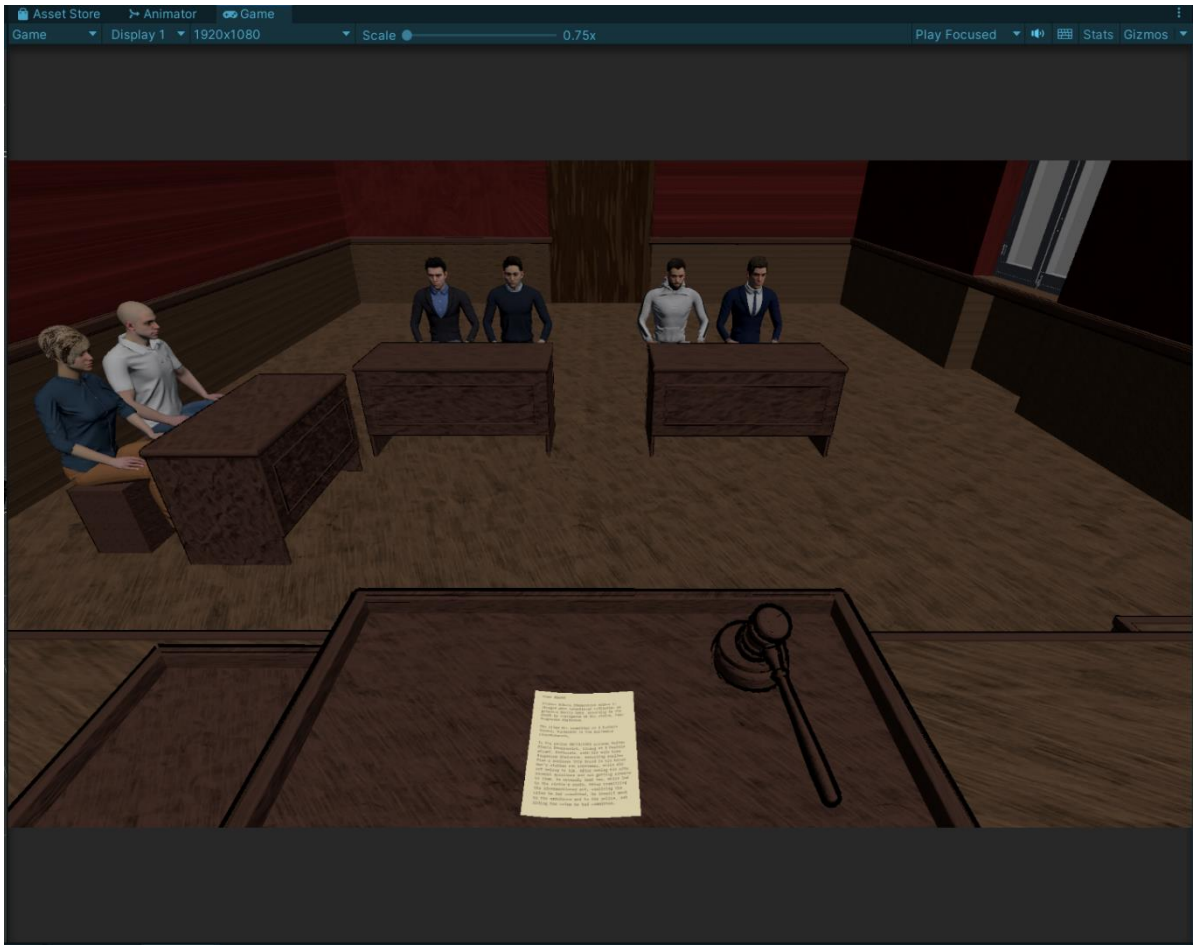

Рисунок 2.8 - Приклад вкладинки Game після запуску гри

Важливим аспектом використання ассетів є інтегрований Unity Asset Manager. Тут зберігаються придбані та імпортовані ассети розробника. Ці ассети можна отримати у Unity Asset Store (рис. 2.9) – тут знаходяться та продаються готові тематичні набори моделей, цілі системи скриптів, наприклад для взаємодії з UI, та багато іншого, що прискорює розробку та є чимось на кшталт бібліотек у програмуванні [29].

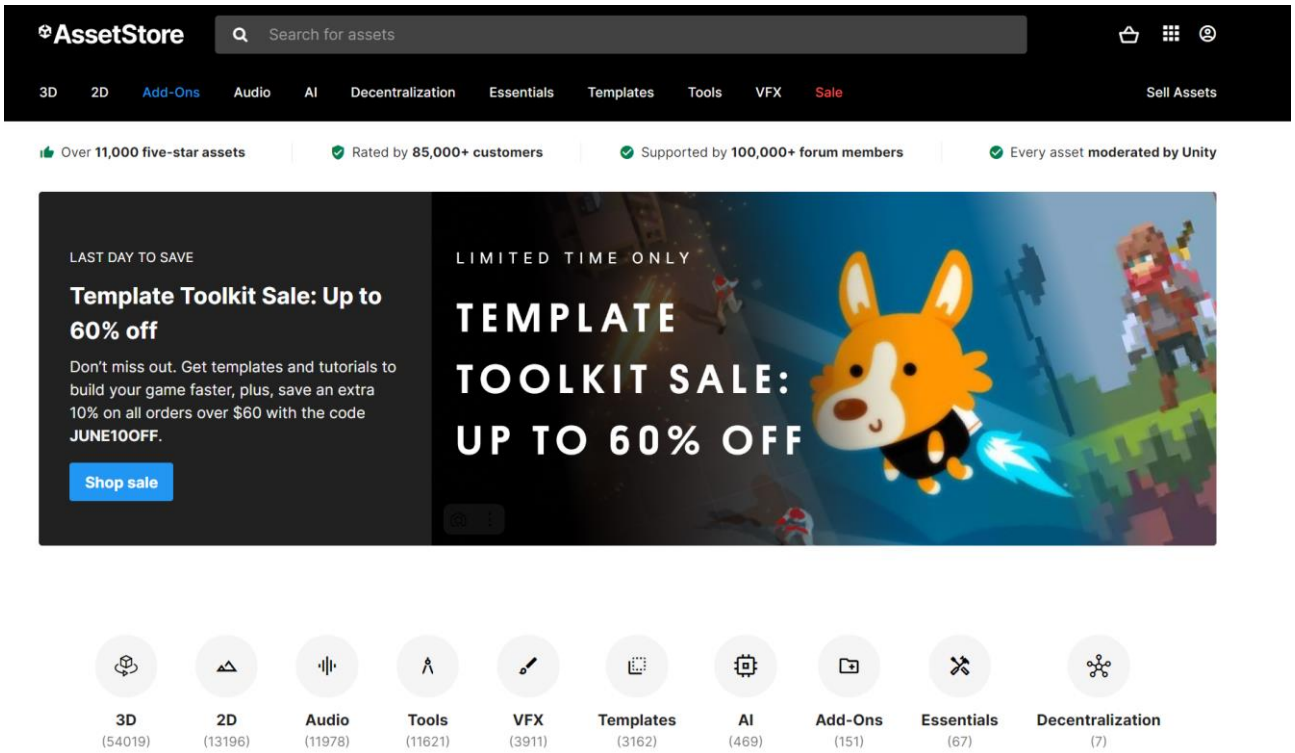

Рисунок 2.9 – магазин Unity Asset Store

Також, пліч-о-пліч з Unity Asset Manager йде Unity Package Manager (рис. 2.10) – менеджер пакетів, що дозволяє підключати та використовувати нову логіку та засоби при розробці. Наприклад, можна підключити пакет для побудови програми для WebGL та її розміщення на сайті Unity або будь-якій іншій платформі.

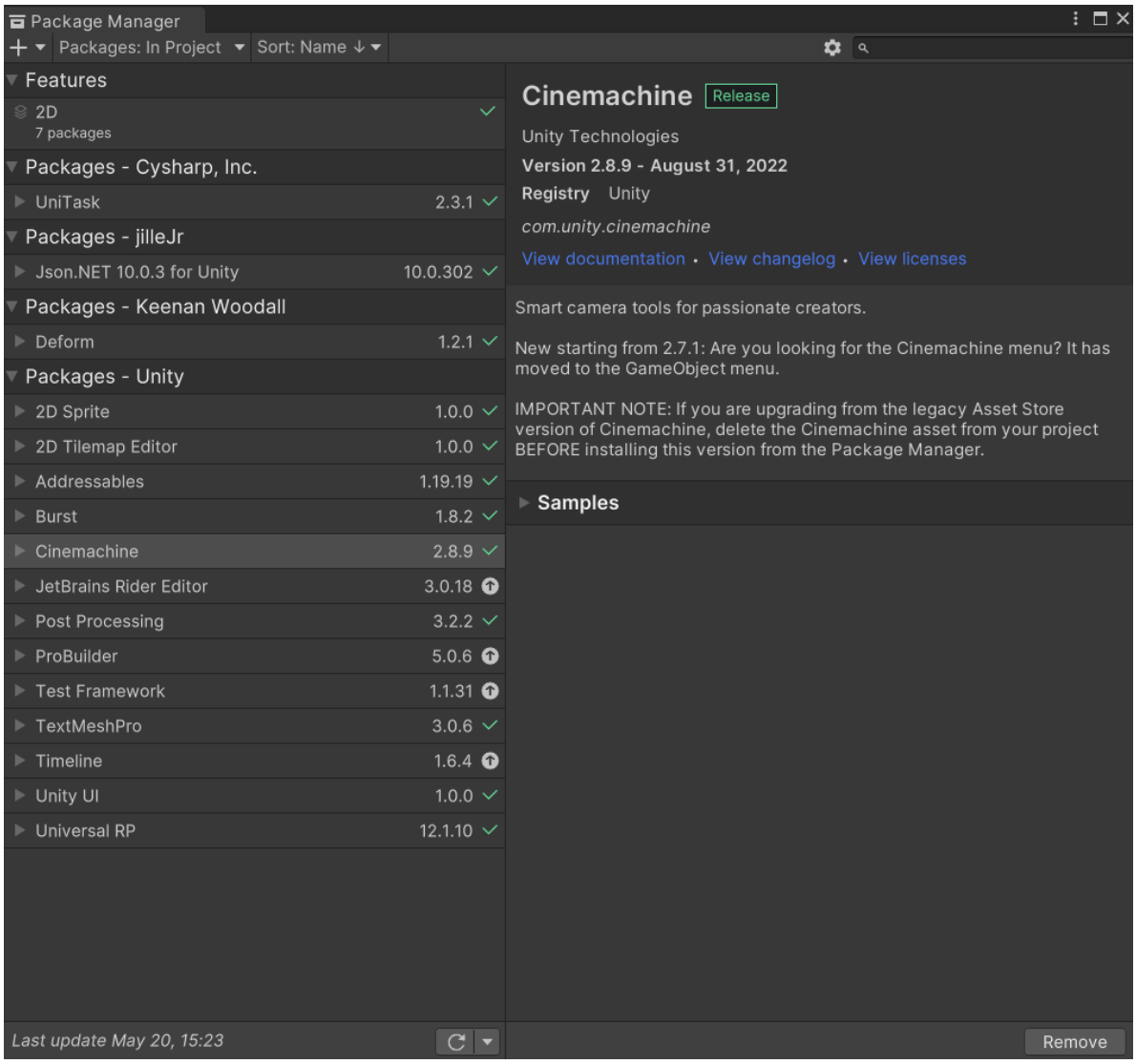

Рисунок 2.10 - Вікно Unity Package Manager (In project)

Також розробнику надані засоби для оптимізації та аналізу застосунку у вкладинці Profiler (рис. 2.11). Дозволяє ефективно відслідковувати, що саме займає найбільше часу під час виконання. Існує можливість виокремлювати блоки коду у скриптах та присвоювати їм окремі назви для зручного вимірювання [43].

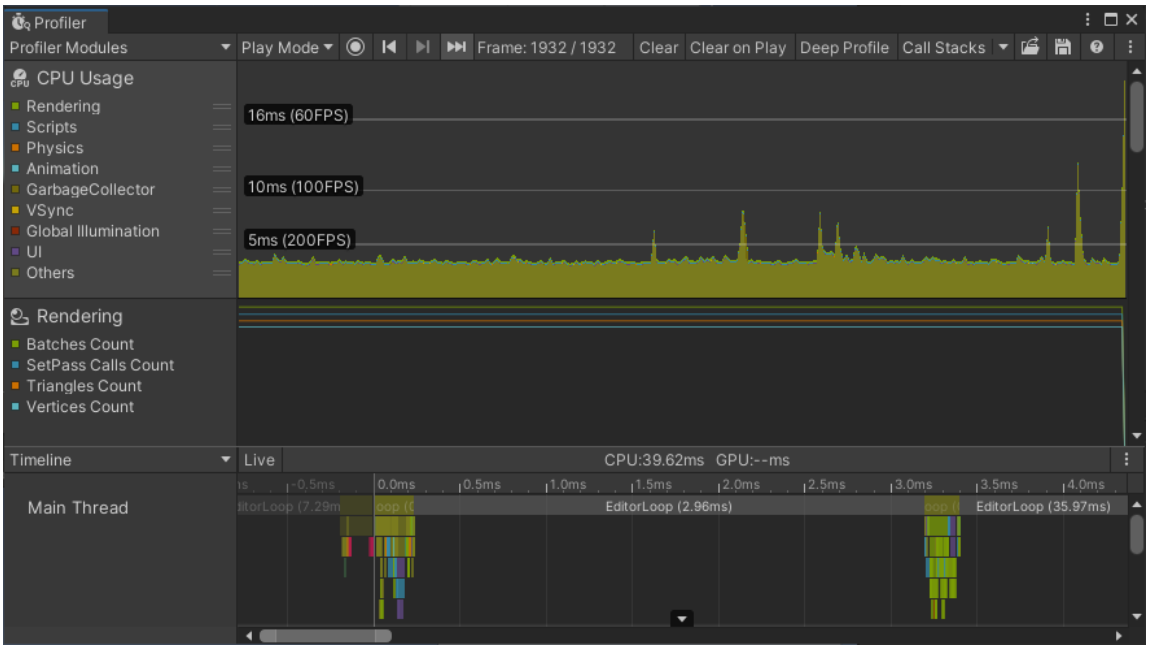

Рисунок 2.11 - Вкладинка Profiler під час виконання застосунку

Для анімації об’єктів та їх переходів між станами існує вкладинка Animator (рис. 2.12). Тут можуть відображатись параметри анімацій, умови переходу між однією анімацією та іншою, а також дерево їх зв’язків

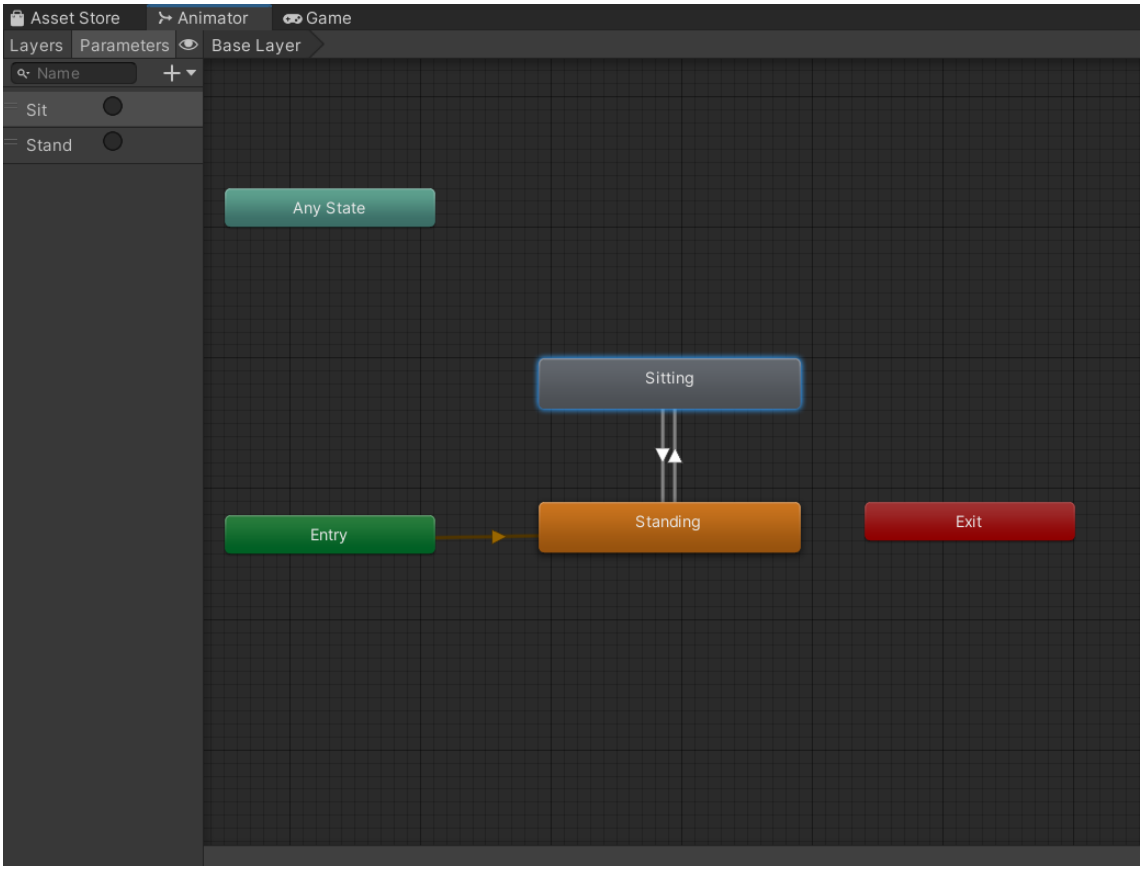

Рисунок 2.12 - Вкладинка Animator

2.3 C# як мова програмування для Unity

C# - це об'єктно-орієнтована мова програмування, що містить елементи інкапсуляції, спадкування та поліморфізму. Вона була розроблена компанією Microsoft у період з 1998 по 2001 роки під керівництвом Андерса Гейлсберга в рамках відділу розробки програмного забезпечення Microsoft .NET Framework з метою створення програм для цієї платформи та відповідала вимогам стандартів ECMA-334 і ISO/IEC 23270.

C# є мовою програмування, що спирається на синтаксис мови C, близький до мов програмування C++ і Java. Мова має статичну типізацію, підтримку поліморфізму, оператори, властивості, типи даних та методи, ітерацію, анонімні функції, LINQ, делегати, обробку винятків та коментування у форматі XML [14].

Назва "C#" (произн. Сі-шарп) відповідає музичному позначенню "діез" з музичної нотації, яке вказує на підвищення на півтону, аналогічно до «++» в назві мови "C++", що вказує на інкремент змінної на одиницю (в цьому випадку, інкремент мовного рівня). Це також відображає прогресію мов програмування від C до C++ і C#.

Мета C# - надати просту, безпечну, сучасну, об'єктно-орієнтовану, високорівневу мову програмування, яка була б легкою для вивчення та розуміння програмістами, а також забезпечити їм широкі можливості розробки програм, включаючи веб-додатки, веб-служби, програми для Microsoft Windows Forms і багато іншого.

Розробники мови C# можуть використовувати вже наявні знання і навички мов програмування C, C++ та Java для досягнення успішних результатів, не змушуючи їх відмовлятися від свого досвіду в цих мовах, а також інтегрувати різноманітні рішення програмного забезпечення, включаючи веб-технології, на базі Microsoft Windows Forms і взаємодію з інтерфейсами клієнт-сервера.

C# має дуже зручний синтаксис, який є дуже простим і зрозумілим під час вивчення. Його особливі та схожі з C, C++ і Java конструкції зробили його відомим серед програмістів, які вже знають одну з цих мов. Ті розробники, які

володіють хоча б однією з цих мов, легко починають працювати з C# і отримують швидко ефективні результати.

C# допомагає впроваджувати методи та типи, які використовуються для підвищення та управління об'єктним кодом. Усі змінні та методи включаються в головний метод Main певного класу. Класи можна успадковувати від іншого основного класу [15].

Синтаксис C# (рис. 2.13) робить його простим у порівнянні з складними конструкціями C++, а також надає потужні функції, такі як NULL (відсутність значення), як в C++, і дозволяє користуватися сильнотипізованими методами та типами, як в Java, що робить його універсальною мовою програмування для багатьох задач.

Version	Date Of Release	.NET Version	Visual Studio
C# 1.0	Jan 2002	.NET 1.0	Visual Studio .NET 2002
C# 2.0	Sep 2005	.NET 3.0	Visual Studio .NET 2008
C# 3.0	Nov 2007	.NET 3.5	Visual Studio .NET 2008
C# 4.0	April 2010	.NET 4.0	Visual Studio .NET 2010
C# 5.0	April 2013	.NET 4.5	Visual Studio .NET 2013
C# 6.0	July 2015	.NET 4.6	Visual Studio .NET 2015
C# 7.0	March 2017	.NET 4.7	Visual Studio .NET 2017
C# 8.0	Sep 2019	.NET Core 3.0	Visual Studio .NET 2019
C# 9.0	April 2020	.NET 5.0	Visual Studio .NET 2019

Рисунок 2.13 – Версії C#

РОЗДІЛ 3
ПРОЕКТУВАННЯ, РОЗРОБЛЕННЯ ТА ВПРОВАДЖЕННЯ МОБІЛЬНОЇ ГРИ З ВИКОРИСТАННЯМ СУЧАСНИХ ТЕХНОЛОГІЙ

3.1 Проектування застосунку

В проектуванні ігор на Unity існують кілька підходів, які розробники можуть використовувати в залежності від специфіки проекту, вподобань і досвіду команди.

В процесі розробки ігор на Unity існує ряд різних методологій та підходів, які розробники можуть використовувати для створення власних проектів. Кожен з цих підходів має свої унікальні особливості та застосування, що робить їх корисними в різних ситуаціях і для вирішення різних завдань у процесі розробки [5].

Один із таких підходів - Model-View-Controller (рис. 3.1), який розділяє гру на три основні компоненти: модель, представлення та контролер. Модель відповідає за зберігання даних та бізнес-логіку, представлення - за відображення інформації на екрані, а контролер - за керування взаємодією між моделлю та представленням. Цей підхід дозволяє структурувати проект та робить його більш модульним та піддаємим змінам [12].

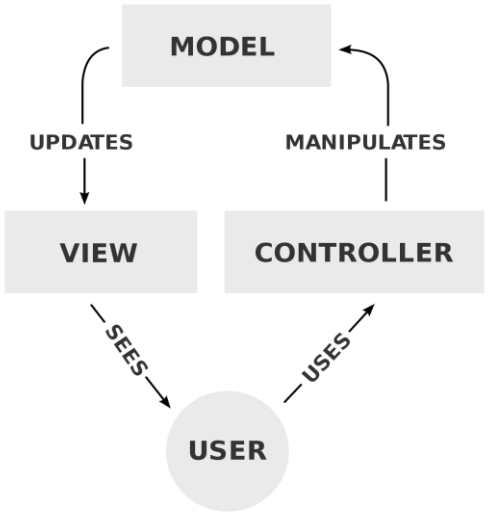

Рисунок 3.1 – Принцип роботи MVC

Ще одним підходом є Entity-Component-System (рис. 3.2), який акцентується на компонентах, що приєднуються до сутностей в грі. Кожен компонент відповідає за певний аспект поведінки або властивість об'єкта, а системи обробляють групи об'єктів з певними компонентами. Цей підхід особливо корисний у випадках, коли потрібно працювати з великими обсягами даних або забезпечити високу продуктивність гри [13].

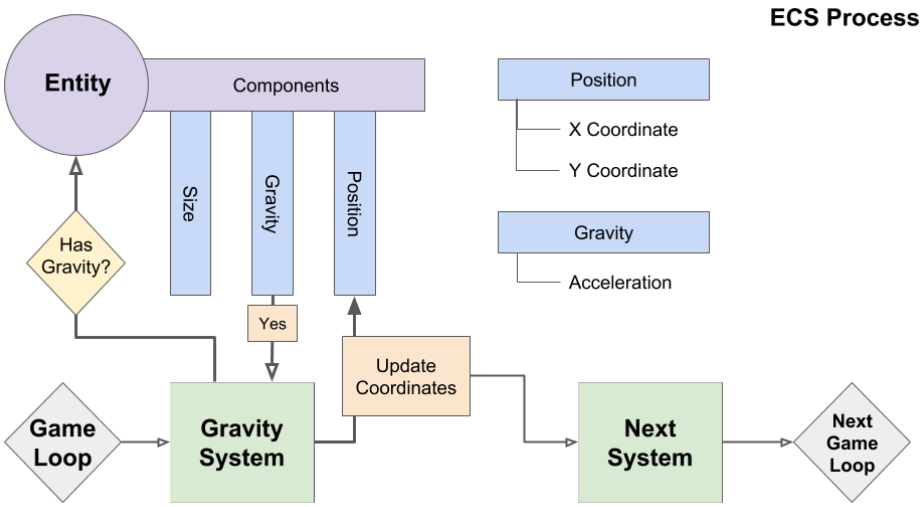

Рисунок 3.2 – Принцип роботи ECS

Також варто відзначити Component-based architecture, який, подібно до ECS, базується на компонентах, але є менш формалізованим. Замість строгого розподілу на компоненти, сутності і системи, розробники можуть використовувати компоненти як основні будівельні блоки ігрових об'єктів, комбінуючи їх для створення необхідної поведінки [11].

State-driven design (рис. 3.3) - це ще один підхід, який базується на використанні кінцевих автоматів для керування станом ігрових об'єктів. Кожен об'єкт може перебувати в певному стані, а перехід між станами відбувається в залежності від певних умов. Це дозволяє спростити керування поведінкою об'єктів та реалізувати різні варіанти взаємодії [16].

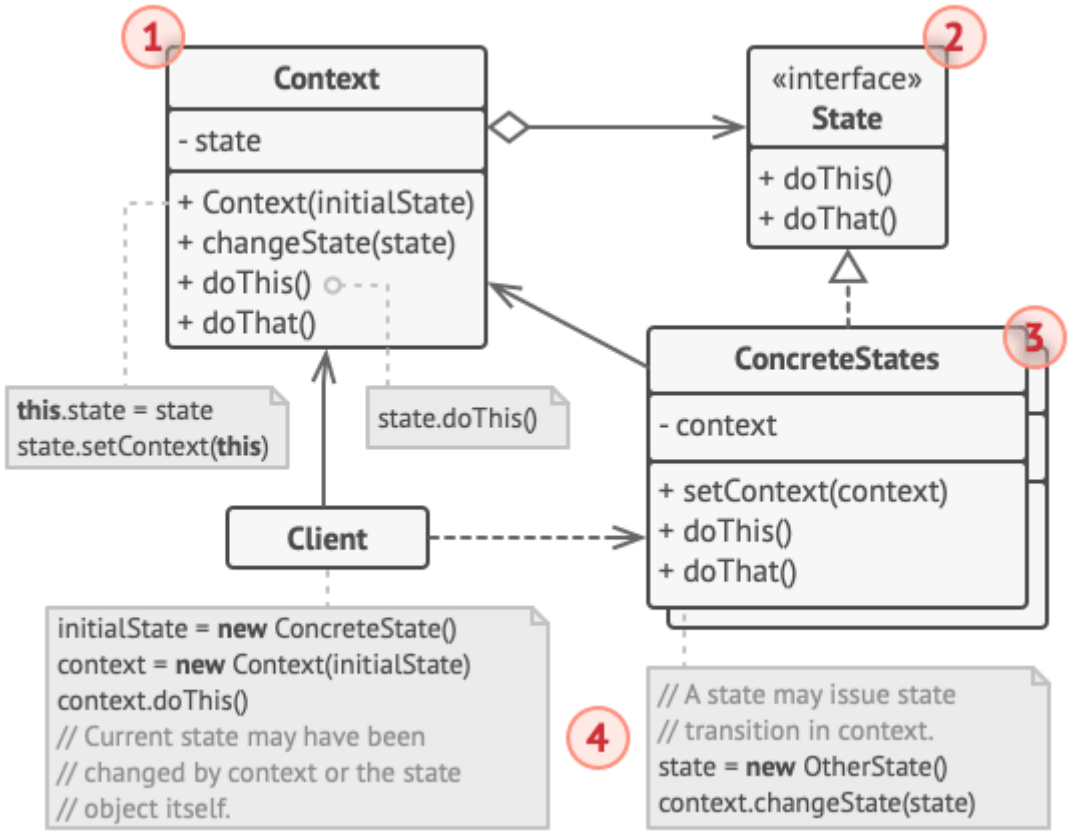

Рисунок 3.3 – Принцип роботи SDD

Нарешті, Event-driven architecture базується на обробці подій, які відбуваються в грі. Різні частини гри можуть генерувати події, на які інші частини можуть реагувати. Це дозволяє керувати взаємодіями між різними системами та об'єктами в грі, що робить цей підхід досить гнучким та піддаємим змінам у процесі розробки.

Кожен з цих підходів має свої переваги й недоліки, і вибір конкретного залежить від конкретних вимог проекту та вподобань команди розробників [9].

Для успішної реалізації гри необхідно розробити геймплей та механіку гри, візуальний стиль, звуковий дизайн, управління та інтерфейс, монетизацію, соціальні функції, а також провести тестування та отримати зворотний зв'язок від користувачів. Геймплей та механіка гри включають визначення основної механіки руху персонажа, рівнів складності, типів перешкод та способів отримання бонусів. Візуальний стиль передбачає вибір графічного стилю, анімацію персонажів та створення привабливих ілюстрацій і візуальних ефектів. Звуковий дизайн охоплює створення атмосферних звуків для підтримки

геймплею. Управління та інтерфейс включають розробку інтуїтивно зрозумілого ігрового інтерфейсу, а монетизація - вибір моделі монетизації гри. Соціальні функції передбачають включення можливостей обміну результатами з іншими гравцями та режими гри з багаторазовими гравцями. Тестування та зворотний зв'язок проводяться з метою виявлення помилок та отримання відгуків для подальшого вдосконалення гри.

Ця тема є дуже популярною серед розробників мобільних ігор, оскільки дозволяє поєднати технічні вміння, творчість та спритність у створенні захоплюючого ігрового досвіду для користувачів мобільних пристроїв.

3.2 Розроблення інтерфейсу

Створення інтерфейсу я почав з версти головного екрану (рис. 3.4). Оскільки гра в мене починається по тапу на пусте місце екрану, тому кнопки «Старг» як в інших іграх в мене нема. На головний екран я додав:

- назву-логотип гри
- кнопку налаштуваннь
- кнопку магазину
- кнопка Battlepass`у
- кількість грошей

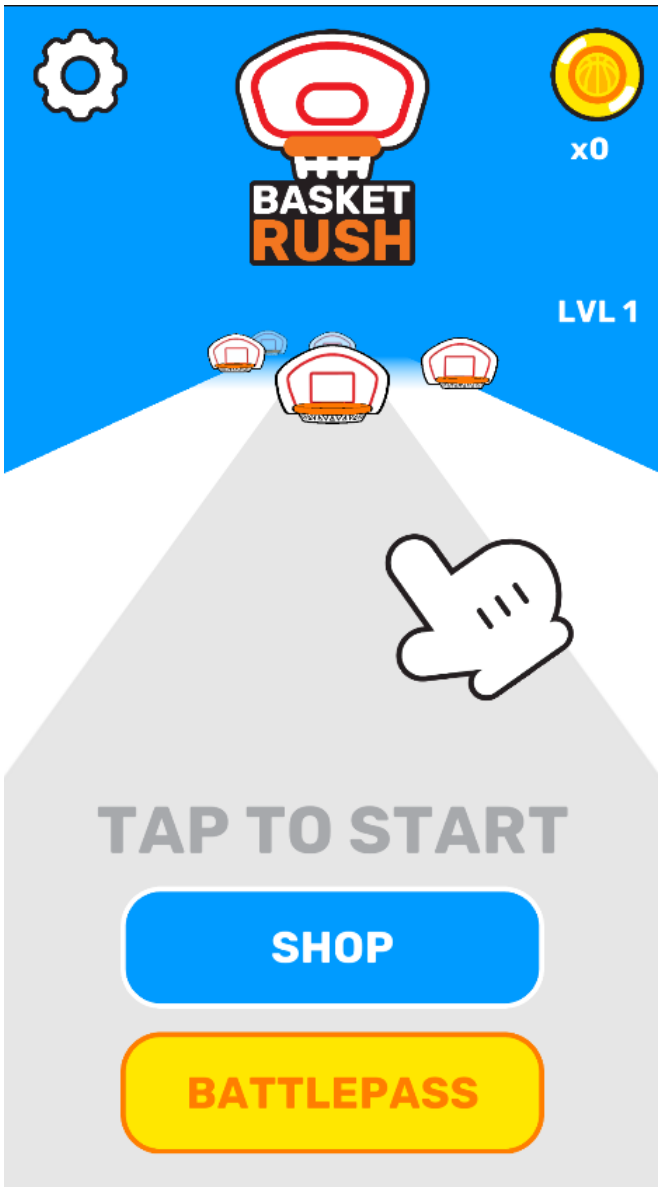

Рисунок 3.4 – Головний екран

Наступним кроком я вирішив зверстати магазин (рис. 3.5). Спочатку я додав кнопку реклами задля отримання грошей, потім зробив 2 сітки з кнопок. Перша сітка з кнопками це кнопки для купівлі скнів на мячі, а друга – для купівлі скінів на корзинки [46].

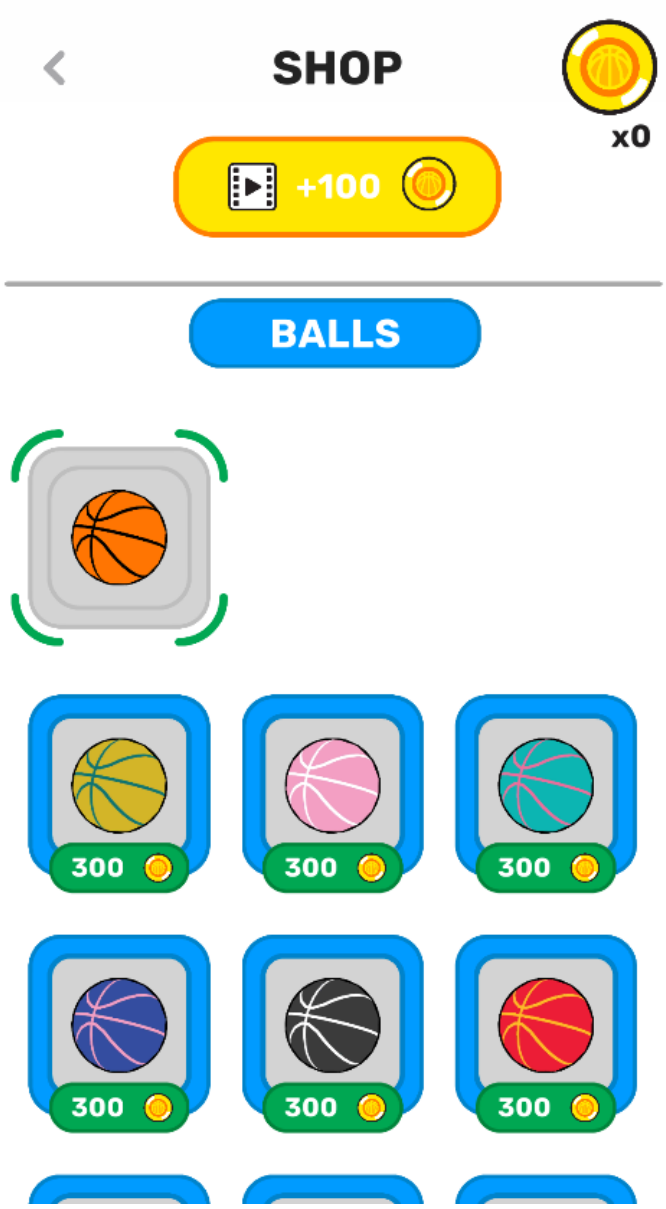

Рисунок 3.5 - Магазин

Останнім важливим віном є вікно Battlepass’у (рис. 3.6). Тут я почав з точно що добавив компонент Scroll View для того щоб можно було гортати сторінка. Заповнив сторінку кнопками у вертикальній группі та додав Content Size Fiiter для апаптування під розмір екрану.

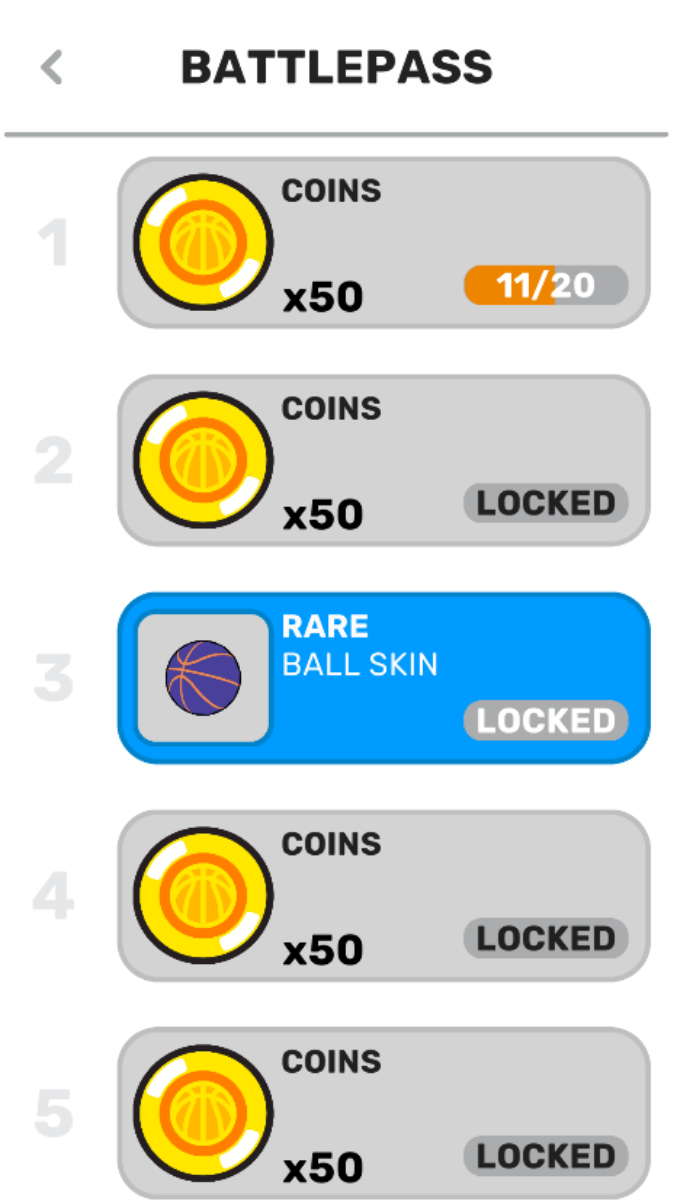

Рисунок 3.6 – Battlepass

Всі матеріали що я викроистав в верстці потрібно було десь знайти, купити або створити. Я вирішив обрати останній варіант та зробити усе самому в фотошопі використовуючі референси з інтернету. Після довгого часу роботи я отримав готові ассети для створення гри.

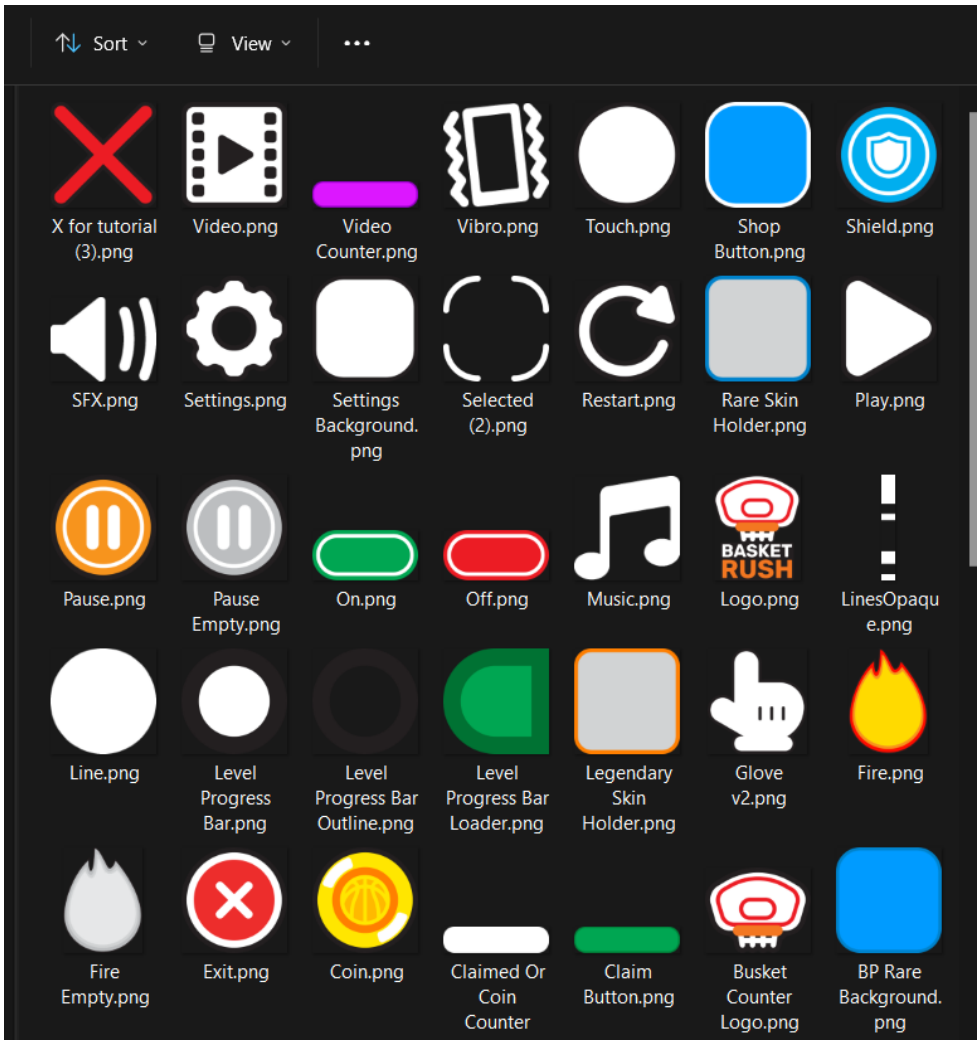

Рисунок 3.7 – Основні спрайти UI

Після створення матеріалів (рис. 3.7) для гри потрібно було їх імпортувати в гру та виставити налаштування. Це є дуже важливим етапом оскільки системі треба розуміти який тип матеріалі як спрймати. Наприклад, в нас будуть текстури (рис. 3.9) та спрайти (рис. 3.8) які треба імпортувати по різному, аби вони правильно працювали та не було багів [8].

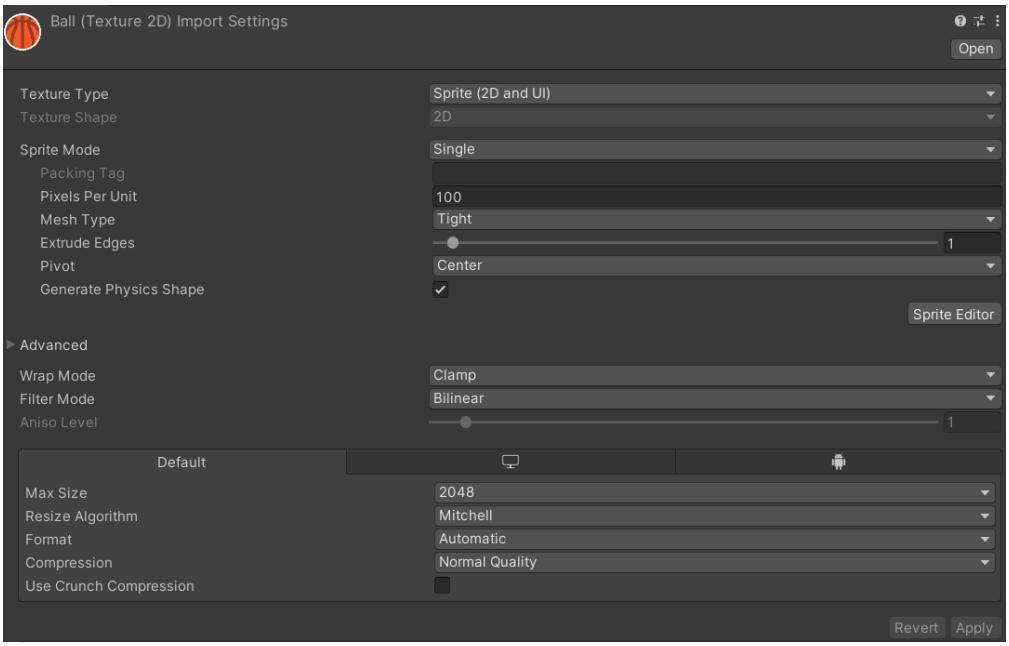

Рисунок 3.8 – Приклад імпортування спрайту

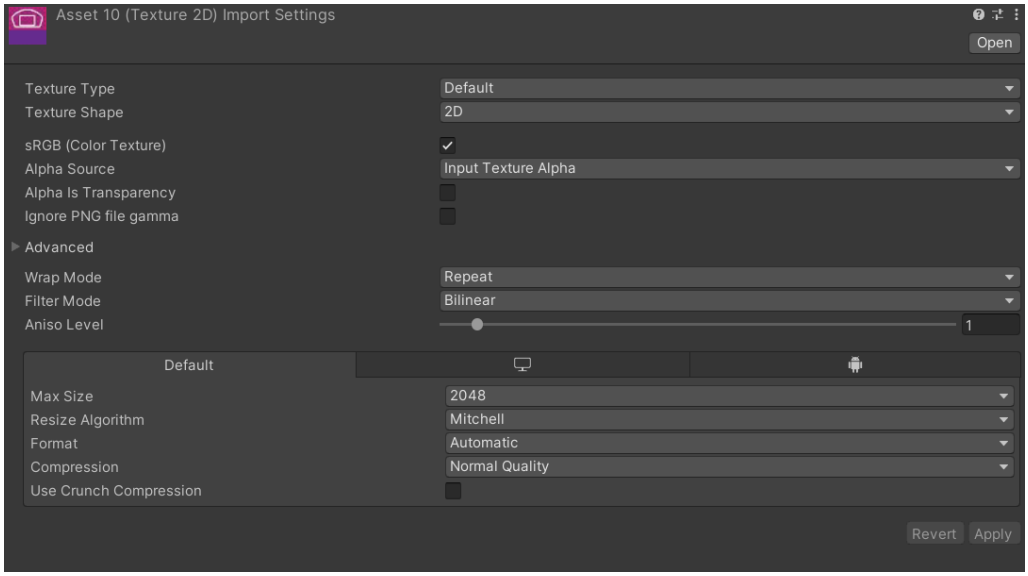

Рисунок 3.9 – Приклад імпортування текстури

Для того щоб зробити більш приємне оточення в ігрі мені не вистачили стандіртні засоби Unity, адже я хотів додати тума, аби гравець не міг бачити що його очікує. Також, мячі та корзинги виглядали дуже плоско, тому було прийнято рішення зробити обводку. Після великої кількоісті витраченого часу в пошуках рішення проблеми я прийшов до того що в мене є 2 основних способа це зробити – самописні шейдери або ассети. Оскільки я не хотів витрачати кошти на вже готовий ассети я вирішив спробувати написати повноцінний шейдер власноруч [4].

Шейдери в Unity - це програми, які використовуються для керування тим, як об'єкти виглядають при візуалізації в грі. Вони дозволяють контролювати різні аспекти візуального представлення, такі як колір, текстури, освітлення, прозорість тощо. Основні типи шейдерів, що використовуються в Unity, це вершинні та піксельні шейдери [7].

У Unity, інноваційному середовищі для розробки ігор, існують два ключові інструменти для створення шейдерів - ShaderLab і Shader Graph. Кожен з цих інструментів має свій власний неповторний характер, відкриваючи широкі можливості для розробників у написанні та візуальному створенні шейдерів.

ShaderLab (рис. 3.10) є мовою та середовищем опису для створення шейдерів в Unity. Цей інструмент дозволяє розробникам визначати основні характеристики шейдерів, такі як текстури, освітлення, альфа-змішування та багато інших параметрів. Використання ShaderLab дає можливість точно налаштувати зовнішній вигляд об'єктів у грі, створюючи реалістичні та захоплюючі візуальні ефекти [6].

Font.shader

```
Shader "GUI/Text Shader" {
    Properties {
        _MainTex ("Font Texture", 2D) = "white" {}
        _Color ("Text Color", Color) = (1,1,1,1)
    }

    SubShader {

        Tags {
            "Queue"="Transparent"
            "IgnoreProjector"="True"
            "RenderType"="Transparent"
            "PreviewType"="Plane"
        }
        Lighting Off Cull Off ZTest Always ZWrite Off
        Blend SrcAlpha OneMinusSrcAlpha

        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            struct appdata_t {
                float4 vertex : POSITION;
                fixed4 color : COLOR;
                float2 texcoord : TEXCOORD0;
            };

            struct v2f {
                float4 vertex : SV_POSITION;
                fixed4 color : COLOR;
                float2 texcoord : TEXCOORD0;
            };

            sampler2D _MainTex;
            uniform float4 _MainTex_ST;
            uniform fixed4 _Color;

            v2f vert (appdata_t v)
            {
                v2f o;
                o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
                o.color = v.color * _Color;
                o.texcoord = TRANSFORM_TEX(v.texcoord,_MainTex);
                return o;
            }

            fixed4 frag (v2f i) : SV_Target
            {
                fixed4 col = i.color;
                col.a *= tex2D(_MainTex, i.texcoord).a;
                return col;
            }
            ENDCG
```

Рисунок 3.10 – Приклад коду ShaderLab шейдеру

З іншого боку, Shader Graph (рис. 3.11) - це візуальний інструмент, що дозволяє створювати шейдери без необхідності в програмуванні. Використовуючи графічний інтерфейс, розробники можуть легко налаштовувати текстурні ефекти, освітлення, колірні змішування та інші аспекти візуального відображення, з'єднуючи вузли для створення складних ефектів. Shader Graph створений для того, щоб спростити процес розробки шейдерів, забезпечуючи шлях до творчості, що не обмежений технічними деталями.

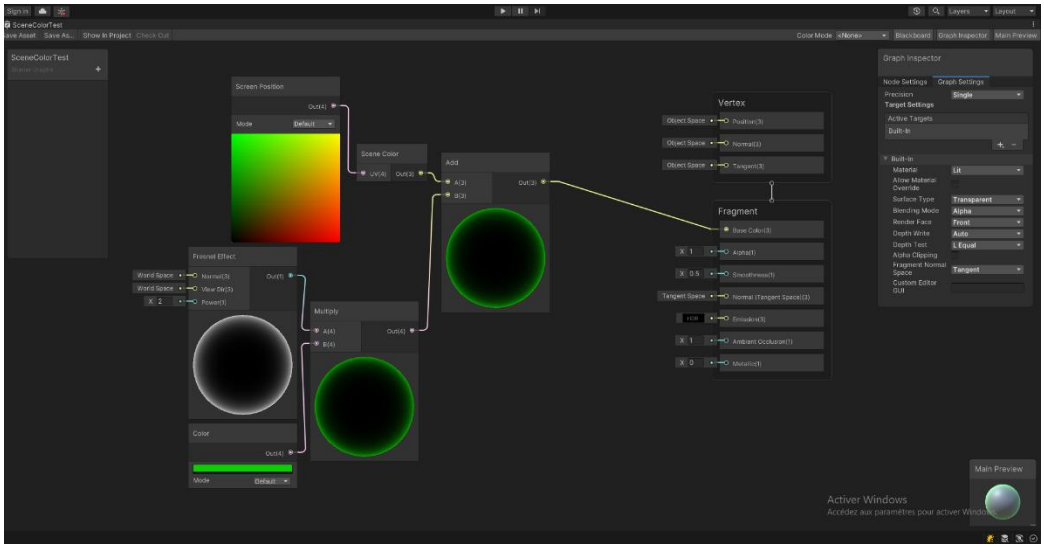

Рисунок 3.11 – Приклад ShaderGraph шейдеру

Таким чином, завдяки комбінації ShaderLab і Shader Graph, розробники мають можливість ефективно та творчо створювати шейдери для своїх ігор, надаючи гравцям неповторні візуальні враження та забезпечуючи захоплюючий іммерсивний геймплей [17].

Після зважування плюсів та мінусів кожного з варіантів я обрав ShaderLab оскільки він дає нам більше контролю.

В результаті я написав шейдер з обводкою (рис. 3.12) та шейдер туману (рис. 3.13).

Рисунок 3.12 – Приклад шейдера з обводкою

Рисунок 3.13 – Приклад шейдера з туманом

3.3 Геймплей створенної гри

Гра вбирає в себе відтворення гравця, який рухаючись по лініям має закидувати баскетбольного м'ячика до корзинки аби пройти далі. Задача гравця полягає в тому щоб швидко реагувати на зміни в ігри, своєчастно міняти лінії та закидувати мячики чітко у корзинку. Якщо гравець на набирає мінімуму по корзинкам то вона має почати рівень заново. Осікльки у грі гравець весь час біжить вперел та міняє лінії гри, то наша гра підпадає піж жанр «раннери» [37].

Гра має головне меню, яке включає в собі глобальну кнопку на весь екран (рис. 3.14) для почтку гри "Tap to Start". Кнопку переходу до магазину, до налаштуваннь та до Battlepass`y

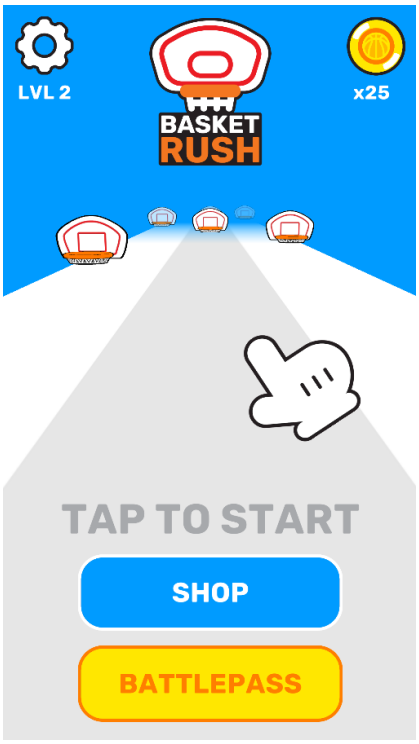

Рисунок 3.14 – Головне меню

Щоб гравець був більше залучений до гри, ми додаємо різні типи корзин (рис. 3.15) та збільшуємо рівень складності гри. Є звичайна корзинка, корзика-програш та корзинка з бустом якогось з типів.

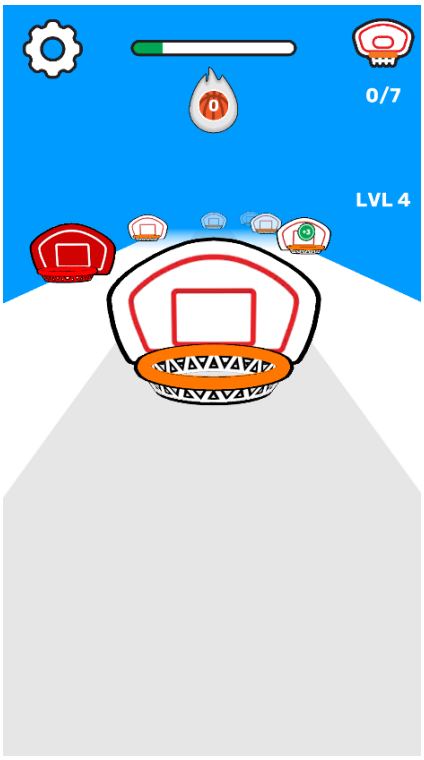

Рисунок 3.15 – Типи корзин

У випадку перемоги в рівні відображається попап (рис. 3.16) із кнопкою перезапуску рівня, переходу на наступний рівень, кнопка перегляду реклами для отримання x2 грошей, скільки кошиків ми зібрали та прогресс BattlePass`y

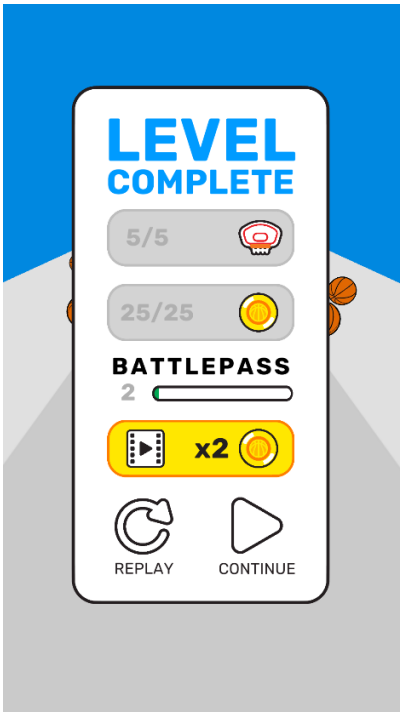

Рисунок 3.16 – Попап проходження рівня

Також додаємо BattlePass (рис. 3.17), який дає можливість гравцеві переходити рівні щоб збирати достатньо балів та отримувати різні типів нагород.

У BattlePass також є можливість отримати скіни на м'ячів та корзинки, які розширюють досвід гравця та більш глибоко занурюють його в гру.

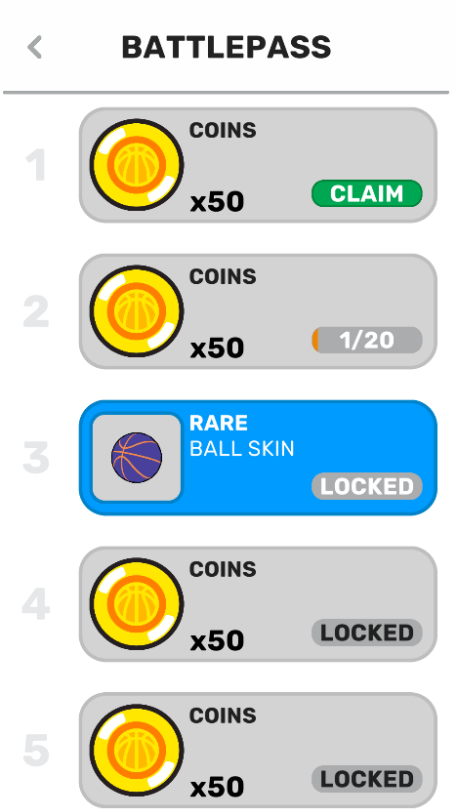

Рисунок 3.17 - Battlepass

Також додаємо розділ магазину (рис. 3.18), де гравець може купити різні типи м'ячів та корзинок, за гроші.

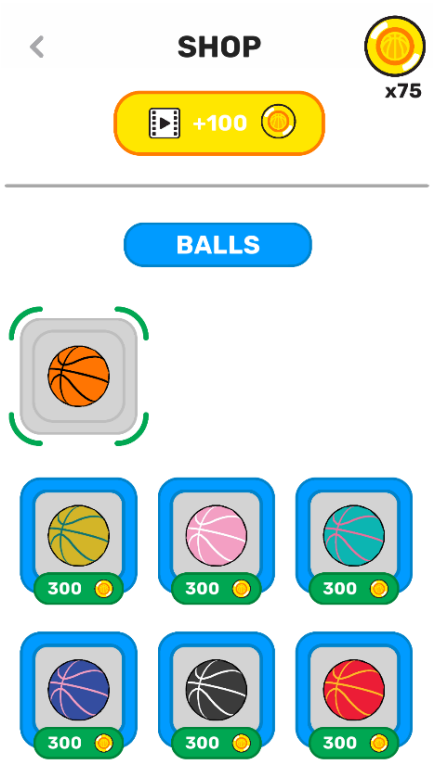

Рисунок 3.18 - Магазин

У налаштуваннях гри (рис. 3.19) ми додаємо звук та можливість вимкнути вібрацію. У меню також доступні налаштування, де можна змінити типові значення.

Рисунок 3.19 - Налаштування

ВИСНОВОК

У результаті виконання кваліфікаційної роботи бакалавра було виконано такі завдання:

- досліджено теоретичні основи технологій розробки мобільних ігор засобами Unity
- досліджено платформи та засоби розробки з використанням ігрового рушія Unity.
- вивчено наявні на ринку ігор жанру “Раннер”, що можуть створити конкуренцію;
- спроектовано та розроблено мобільну гру з використанням ігрового рушія Unity.

З дипломної роботи можна зробити висновок, що Unity є зручним рушієм для створення мобільної гри яка може створювати гарну конкуренцію на ринку.

У дипломній роботі було проведено дослідження різних рушії для разробки мобільних ігор таких як Unity, Godot, Unreal Engine. Також було проведено аналіз існуючих ігор на мобільному ринку, що дозволило виявити певні тенденції та особливості в їх розробці та функціональності. Було виявлено, що багато корустуваів обирають жанр раннерів для того щоб тренувати свої навички та реакцію. Можна зробити висновок, що мобільни ігри є важливим напрямком розвитку індустрії ігор, в особливості ігри жанру раннер.

Дослідження показали, що найбільш популярними платформами для розробки ігор є Unity. Unity було вибрано для розробки дипломної роботи через його простоту використання та доступності для початківців у галузі розробки ігор. Також, з огляду на те, що гра розроблена на рушії Unity, її можна легко оптимізувати для різних платформ та пристроїв.

Отже, дипломна робота з темою "Мобільна гра на основі 3D Unity" є актуальною та важливою в галузі розробки комп'ютерних ігор. Результатом роботи стала створена гри на рушії Unity, яка дозволяє грати в раннер тим самим тренуючі свою навички та реакцію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кров, піт і пікселі. Зворотний бік індустрії відеоігор / Шрейєр Д.: Book Chef, 2021. 338 с.
2. Game Over. Як Nintendo завоювала світ / Шефф Д.: Біле яблуко, 2014. 384 с.
3. Unity 3D. Введення в розробку ігор / Кучера В. В.: Айрис-Прес, 2019. 352 с.
4. Unity 5: від простого до складного / Голдстін Л.: Діалектика, 2018. 416 с.
5. Unity 2018 для розробників ігор. Професійне програмування / Лейдбеттер Д.: БХВ, 2019. 800 с.
6. Unity. Професійний розвиток ігор / Катлін М.: ДМК Прес, 2018. 720с.
7. Unity в дії. Мультиплатформена розробка ігор / Магоуен Д.: ДМК Прес, 2020. 440 с.
8. Програмування ігор на Unity / Лаутон К.: ДМК Прес, 2019. 336 с.
9. Unity в дії: мультиплатформена розробка ігор / Шейд М., Маклін Л.: БХВ, 2019. 824 с.
10. Unity для розробників ігор / Хоскінс Д.: Print2Print, 2017. 480 с.
11. Unity в дії. Переклад з англійської / Макфедріс Д., Меннінг Б.: Манн, Іванов і Фербер, 2019. 608 с.
12. Створення комп'ютерних ігор на Unity 3D. Детальний посібник / Манцурова А. В.: Логос, 2018. 312 с.
13. Unity 5 для початківців. Створюємо 3D-ігри / Саух С.: ДМК Прес, 2019. 416 с.
14. C# 4.0 The Complete Reference / Шилдт Г.: McGraw-Hill, 2010. 976 с.
15. CLR via C# / Ріхтер Д.: Microsoft Press, 2020. 896 с.
16. Unity в дії. Повний посібник / Джексон Е., Марк П. : БХВ, 2019. 680с.
17. Unity 5.x Cookbook / Роджерс Д., Шіро К.: БХВ, 2018. 588 с.
18. Unity в дії: мультиплатформена розробка ігор / Тер Турм Н.: БХВ-, 2019. 816 с.
19. Unity 3D і розробка ігор / Манцурова А. В.: Логос, 2017. 256 с.
20. Unity для розробників ігор / Карлсон Дж.: БХВ, 2018. 496 с.
21. Unity 3D для початківців. Створення ігор / Кучера В. В.: Діалектика, 2017. 320 с.
22. Unity 2018 Cookbook / Горрілла Б.: БХВ, 2019. 644 с.
23. Unity 5 в дії. Створення ігор та VR-проектів / Маккінніс Д.: П БХВ ітер, 2018. 520 с.
24. Створення мобільних 3D-ігор з використанням Unity: Практичний посібник: веб-сайт. URL: https://www.packtpub.com (дата звернення: 15.04.24)

25. Unity в дії: Розробка багатоплатформних ігор на C#: веб-сайт. URL: https://www.packtpub.com (дата звернення: 17.02.24)

26. Проекти віртуальної реальності на Unity: Досліджуйте світ віртуальної реальності, створюючи захоплюючі та цікаві VR-проекти з використанням Unity 3D: веб-сайт. URL: https://www.packtpub.com (дата звернення: 27.02.24)

27. Розробка мобільних ігор на Unity 2017: Створення, розгортання та монетизація ігор для Android та iOS з використанням Unity: веб-сайт. URL: https://www.packtpub.com (дата звернення: 22.03.24)

28. Кулінарна книга розробки ігор на Unity: Основи для кожного геймера: веб-сайт. URL: https://www.oreilly.com (дата звернення: 21.04.24)

29. Посібник користувача Unity: веб-сайт. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 21.04.24)

30. API для скриптів Unity: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/index.html (дата звернення: 19.04.24)

31. Магазин ресурсів Unity: веб-сайт. URL: https://assetstore.unity.com/ (дата звернення: 12.02.2024)

32. Форум спільноти Unity: веб-сайт. URL: https://forum.unity.com/ (дата звернення: 15.03.2024)

33. Спільнота Reddit про Unity3d: веб-сайт. URL: https://www.reddit.com/r/Unity3D/ (дата звернення: 09.04.2024)

34. Вступ до Unity: Створення 3D-ігри безкінечного бігу: веб-сайт. URL: https://www.pluralsight.com/courses/unity-3d-endless-runner-game (дата звернення: 07.03.2024)

35. Навчання розробці ігор на Unity 3D - Вивчайте програмування на Unity 3D: веб-сайт. URL: https://www.udemy.com/topic/unity-game-development/ (дата звернення: 25.01.2024)

36. Навчання Unity: веб-сайт. URL: https://learn.unity.com/ (дата звернення: 15.03.2024)

37. Створення безкінечної 3D-ігри з бігом в Unity: веб-сайт. URL: https://www.raywenderlich.com/unity/3d-infinite-runner-game-in-unity (дата звернення: 24.02.2024)

38. Як створити просту 3D-ігру з безкінечним бігом в Unity.: веб-сайт. URL: https://gamedevelopment.tutsplus.com/tutorials/how-to-make-a-simple-3d-endless-runner-game-in-unity--cms-27955 (дата звернення: 23.01.2024)

39. "Навчальний курс Unity 3D - Створення 3D-ігри з безкінечним бігом: веб-сайт. URL: https://academy.zenva.com/product/unity-3d-tutorial-making-a-3d-endless-runner-game/ (дата звернення: 30.04.2024)

40. Проектування та розробка 3D-ігри з безкінечним бігом за допомогою Unity: веб-сайт. URL: http://www.ijettjournal.org/archive/ijett-v50p200.pdf (дата звернення: 13.03.2024)

41. Найкращі практики розробки мобільних ігор на Unity: веб-сайт. URL: https://learn.unity.com/tutorial/best-practices-for-mobile-game-development-in-unity (дата звернення: 02.05.2024)

42. Навчання розробці ігор на Unity 3D для початківців: веб-сайт. URL: https://www.gamedev.net/tutorials/unity/ (дата звернення: 10.02.2024)

43. Оптимізація продуктивності графіки: веб-сайт. URL: https://learn.unity.com/tutorial/optimizing-graphics-performance (дата звернення: 27.04.2024)

44. Розробка мобільних ігор на Unity 3D: Від початку до кінця: веб-сайт. URL: https://www.linkedin.com/learning/unity-3d-mobile-game-development-from-start-to-finish (дата звернення: 05.02.2024)

45. Розробка мобільних ігор з використанням Unity: Створіть один раз, розгорніть будь-де: веб-сайт. URL: https://www.coursera.org/learn/mobile-game-development (дата звернення: 06.04.2024)

46. Офіційний YouTube-канал Unity: веб-сайт. URL: https://www.youtube.com/user/Unity3D (дата звернення: 21.01.2024)

47. Створення 3D-ігри з безкінечним бігом в Unity - Повний посібник: веб-сайт. URL: https://www.youtube.com/watch?v=dwcT-Dch0bA (дата звернення: 29.03.2024)

48. Як створити 3D-ігру з безкінечним бігом в Unity: веб-сайт. URL: https://www.youtube.com/watch?v=Jn_FKfP32hI (дата звернення: 14.02.2024)

49. Мобільна розробка ігор: Unity проти Native: веб-сайт. URL: https://medium.com/swlh/mobile-game-development-unity-vs-native-357c7e1f7e94 (дата звернення: 08.01.2024)

50. Розробка мобільних ігор на Unity: Ваш посібник до успіху: веб-сайт. URL: https://gameanalytics.com/blog/unity-mobile-game-development-guide.html (дата звернення: 26.04.2024)

51. Мобільна розробка ігор: Необхідний посібник: веб-сайт. URL: https://www.gamedev.net/articles/programming/general-and-gameplay-programming/mobile-game-development-the-essential-guide-r3272/ (дата звернення: 19.03.2024)

52. Остаточний посібник з мобільної розробки ігор з використанням Unity: веб-сайт. URL: https://www.codementor.io/@jcunanan05/the-ultimate-guide-to-mobile-game-development-with-unity-88b928j5x (дата звернення: 03.02.2024)

ДОДАТКИ

Додаток А

Код скрипту керування корзинками BasketManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.XR.WSA;
using Random = UnityEngine.Random;

public class BasketManager : MonoBehaviour
{
    public static void SpawnStartBaskets()
    {
        int count = GameData.Links.BasketPoolSize <
GameData.CurrentLevelBasketCount + GameData.CurrentLevelTrapBasketCount +
GameData.CurrentLevelMultipleBasketCount
            ? GameData.Links.BasketPoolSize
            : GameData.CurrentLevelBasketCount +
GameData.CurrentLevelMultipleBasketCount + GameData.CurrentLevelTrapBasketCount;

        GameData.SpawnBasketBoostInfo = new
BoostType[GameData.CurrentLevelBasketCount +
GameData.CurrentLevelTrapBasketCount +
GameData.CurrentLevelMultipleBasketCount];

        if (!GameData.IsNewGame)
        {
            GameData.SpawnBasketInfo = new
BasketType[GameData.CurrentLevelBasketCount +
GameData.CurrentLevelTrapBasketCount +
GameData.CurrentLevelMultipleBasketCount];


            int index;

            for (int i = 0; i < GameData.CurrentLevelTrapBasketCount; ++i)
            {
                index = Random.Range(0, GameData.SpawnBasketInfo.Length);
                if (GameData.SpawnBasketInfo[index] == BasketType.Casual)
                    GameData.SpawnBasketInfo[index] = BasketType.Trap;
                else
                    --i;
            }

            for (int i = 0; i < GameData.CurrentLevelMultipleBasketCount; ++i)
            {
                index = Random.Range(0, GameData.SpawnBasketInfo.Length);
                if (GameData.SpawnBasketInfo[index] == BasketType.Casual)
                    GameData.SpawnBasketInfo[index] = BasketType.Multiple;
                else
                    --i;
            }

            for (int i = 0; i < GameData.CurrentLevelBoostCount; ++i)
            {
                index = Random.Range(0, GameData.SpawnBasketInfo.Length);
                if (GameData.SpawnBasketBoostInfo[index] == BoostType.None &&
```

```
GameData.SpawnBasketInfo[index] != BasketType.Trap &&
GameData.SpawnBasketInfo[index] != BasketType.Boss)
                    GameData.SpawnBasketBoostInfo[index] = (BoostType)
Random.Range(1, 4);
                else
                    --i;
            }
        }

        for (int i = 0; i < count; ++i)
            GameData.BasketManager.Spawn(i);
    }

    private void Spawn(int index)
    {
        List<Row> availableRows = new List<Row> {Row.Left, Row.Middle,
Row.Right};
        availableRows.Remove(GameData.FarthestBasketRow);
        Row row = availableRows[Random.Range(0, 2)];

        if (GameData.IsNewGame)
        {
            switch (index)
            {
                case 0:
                    row = Row.Middle;
                    break;
                case 1:
                    row = Row.Left;
                    break;
                case 2:
                    row = Row.Middle;
                    break;
                case 3:
                    row = Row.Right;
                    break;
            }
        }

        GameData.FarthestBasketRow = row;

        GameObject basket = Instantiate(GameData.Links.BasketPrefab, new
Vector3(((int) row - 1) * GameData.Links.SpaceBetweenRows,
            GameData.Links.MainCamera.transform.position.y +
GameData.Links.BasketSpawnHeightAboveCamera,
            (GameData.FarthestBasketTransform == null ?
GameData.Links.BasketSpawnGap : GameData.FarthestBasketTransform.position.z) +
            GameData.Links.BasketSpawnGap), Quaternion.identity,
GameData.Links.MainParent.transform);

        GameData.FarthestBasketTransform = basket.transform;

        BasketController controller = basket.GetComponent<BasketController>();


        controller.BoostType =
GameData.SpawnBasketBoostInfo[GameData.SpawnedBasketCount +
GameData.SpawnedTrapBasketCount + GameData.SpawnedMultipleBasketCount];
        controller.Type = GameData.SpawnBasketInfo[GameData.SpawnedBasketCount +
GameData.SpawnedTrapBasketCount + GameData.SpawnedMultipleBasketCount];


        if (GameData.CurrentLevelBasketCount +
```

```
GameData.CurrentLevelTrapBasketCount + GameData.CurrentLevelMultipleBasketCount
==
            GameData.SpawnedBasketCount + GameData.SpawnedTrapBasketCount +
GameData.SpawnedMultipleBasketCount)
            FinishManager.Spawn();
    }

    public void ClearAll()
    {
        for (int i = 0; i < GameData.Links.MainParent.transform.childCount; ++i)
        {
            Transform child = GameData.Links.MainParent.transform.GetChild(i);

            if (child.CompareTag("Basket"))
                Destroy(child.gameObject);
        }
    }
}
```

Додаток Б

Код скрипту керування мячами BallManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography;
using UnityEngine;
using Random = UnityEngine.Random;

public class BallManager : MonoBehaviour
{
    [SerializeField] private GameObject _ballPrefab;
    [SerializeField] private float _throwForceCoefficient = 51;
    [SerializeField] private float _throwAngle = 30;
    [SerializeField] private float _playerThrowCoefficient;
    [Range(0, 1)] public float AutoaimError;

    private Queue<BallData> _ballQueue;

    private float _yzRatio = -1;

    private Vector3 _spawnPos
    {
        get
        {
            Vector3 camPos = GameData.Links.MainCamera.transform.position;

            camPos.y -= 0.5f;
            camPos.z += 1;

            return camPos;
        }
    }

    private void Start()
    {
        _ballQueue = new Queue<BallData>();

        for (int i = 0; i < GameData.Links.BallPoolSize; ++i)
        {
            BallData bd = Instantiate(_ballPrefab, Vector3.zero,
```

```
Quaternion.identity,
GameData.Links.MainParent.transform).GetComponent<BallData>();

            bd.RB.useGravity = false;
            bd.C.enabled = false;
            bd.MR.enabled = false;
            bd.OMR.enabled = false;

            _ballQueue.Enqueue(bd);
        }
    }

    public void Throw(float magnitude, Row row, bool perfect)
    {
        if (GameData.GameIsPaused)
            return;

        BallData ballData = _ballQueue.Dequeue();
        _ballQueue.Enqueue(ballData);

        if (ballData.IsThrown)
        {
            if (ballData.RespawnCoroutine != null)
            {
                GameData.Coroutines.StopCoroutine(ballData.RespawnCoroutine);

                if (ballData.ShouldDecrementAnimCount)
                {
                    ballData.ShouldDecrementAnimCount = false;
                    --GameData.CurrentAnimationsPlayingCount;
                }

                ballData.RespawnCoroutine =
GameData.Coroutines.StartCoroutine(ballData.BaD.StartDisposeTimer());
            }

            ballData.BaD.Dispose();
        }

        ballData.IsThrown = true;

        ballData.T.position = new Vector3(((int) row - 1) *
GameData.Links.SpaceBetweenRows, _spawnPos.y, _spawnPos.z);

        ballData.RB.useGravity = true;
        ballData.C.enabled = true;
        ballData.MR.enabled = true;
        ballData.OMR.enabled = true;

        RaycastHit[] baskets = GetBaskets(row);
        Array.Sort(baskets, (a, b) => (int) (a.point.z - b.point.z));

        ballData.RB.AddTorque(Random.Range(-3f, 3f), Random.Range(-3f, 3f),
Random.Range(-3f, 3f));

        if (baskets.Length == 0)
        {
            ballData.RB.AddForce(0, magnitude * _playerThrowCoefficient,
magnitude / _yzRatio * _playerThrowCoefficient);
            return;
        }

        bool threw = false;
```

```
        int i = 0;

        try
        {
            foreach (var basket in baskets)
            {
                ++i;
                BasketController basketController =
basket.transform.parent.GetComponent<BasketController>();

                Vector3 targetBasketPosition =
basketController.GetGoalTriggerPos();

                Vector3 perfectVelocity = CalculateVelocity(ballData.T.position,
                    new Vector3(targetBasketPosition.x, targetBasketPosition.y,
targetBasketPosition.z),
                    _throwAngle);

                float coefficient;

                if (targetBasketPosition.z - ballData.T.position.z < 15)
                    coefficient = _throwForceCoefficient + 6;
                else if (targetBasketPosition.z - ballData.T.position.z < 40)
                    coefficient = _throwForceCoefficient + 2;
                else
                    coefficient = _throwForceCoefficient;

                perfectVelocity *= coefficient;

                perfectVelocity.x = 0;

                if (perfect)
                {
                    if (basketController.Type != BasketType.Trap &&
basketController.Type != BasketType.Boss)
                    {
                        ballData.RB.AddForce(perfectVelocity);
                        threw = true;
                        break;
                    }

                    continue;
                }

                if (magnitude * _playerThrowCoefficient / perfectVelocity.y > 1
- AutoaimError / i && magnitude * _playerThrowCoefficient / perfectVelocity.y <
1 + AutoaimError / i)
                {
                    ballData.RB.AddForce(perfectVelocity);
                    threw = true;
                    break;
                }
            }
        }
        catch (Exception e)
        {
            ;
        }

        if (!threw)
            ballData.RB.AddForce(0, magnitude * _playerThrowCoefficient,
magnitude / (Single.IsNaN(_yzRatio) ? 1 : _yzRatio) * _playerThrowCoefficient);
    }
```

```
    private Vector3 CalculateVelocity(Vector3 source, Vector3 target, float
angle)
    {
        Vector3 direction = target - source;
        float h = direction.y;

        direction.y = 0;

        float distance = direction.magnitude;
        float a = angle * Mathf.Deg2Rad;

        direction.y = distance * Mathf.Tan(a);
        distance += h / Mathf.Tan(a);

        if (distance < 0)
            distance = 0;

        float velocity = Mathf.Sqrt(distance * Physics.gravity.magnitude /
Mathf.Sin(2 * a));
        Vector3 ret = velocity * direction.normalized;
        _yzRatio = ret.y / ret.z;

        return ret;
    }

    private RaycastHit[] GetBaskets(Row row) => Physics.RaycastAll(new
Vector3(((int) row - 1) * GameData.Links.SpaceBetweenRows, 0, 0),
Vector3.forward, Mathf.Infinity, Int32.MaxValue ^ (1 << 9));

    public void ResetAll()
    {
        Vector3 pos = new Vector3(-10, -10, -10);

        foreach (BallData bd in _ballQueue)
        {
            bd.T.position = pos;
            bd.RB.useGravity = false;
            bd.RB.velocity = Vector3.zero;
            bd.MR.enabled = false;
            bd.OMR.enabled = false;
        }
    }
}
```

Додаток В

Результат проходження антиплагіату. В результаті бачимо 5% подібності, тобто 95% унікальності. Ризик плагіату «Найвищий» через те, що на титульній сторінці шаблон, і програма в таких випадках помічає ризик як «Найвищий», цю інформацію я взяв з сайту антиплагіату - plag.com.ua