Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Ігровий застосунок в жанрі Casual на основі патерну MVC та технології Unity

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-5
Олексій МАЗУРЕНКО
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Олександр ФОМІЧОВ
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Мазуренку Олексію Едуардовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Ігровий застосунок в жанрі Casual на основі патерну MVC та технології Unity

затверджена наказом по університету від " 26 " травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1) документація мови програмування C#
2) документація рушія Unity
3) середовище розробки Unity Editor
4) інтегроване середовище розробки Visual Studio

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області
2) основні інструменти та середовище розробки
3) архітектурна та програмна реалізація проекту
4) інструкція користувача
5) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	30.05.25-31.05.25	
2	Формування переліку вимог до програми	01.06.25-02.06.25	
3	Вибір технології розробки та	03.06.25-05.06.25	
	інструментальних засобів		
4	Розробка та тестування програми	06.06.25-11.06.25	
5	Оформлення матеріалів кваліфікаційної роботи	11.06.25-15.06.25	
6	Подання кваліфікаційної роботи керівникові та	16.06.25-19.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на	19.06.25-20.06.25	
	рецензування		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Олександр ФОМІЧОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 22 рис., 0 табл., 1 дод., 20 джерел.

UNITY, CASUAL, RUNNER, ІГРОВИЙ РУШІЙ, C#, ПАТЕРН MVC.

Метою кваліфікаційної роботи є розробка гри у жанрі casual з використанням патерну MVC та на базі ігрового рушія Unity, що дозволяє легко створити структурований, масштабований та зручний у підтриманні код.

У ході виконання роботи було проаналізовано особливості жанру casual ігор, а саме його піджанру runner, вимоги до геймплею та інтерфейсу користувача. Обґрунтовано використання шаблону MVC для поділу логіки, візуалізації та управління станом гри. Реалізовано прототип гри з базовими механіками, інтерфейсом та системою обробки подій. Проведено тестування функціональності, оптимізацію продуктивності та підготовку до подальшого масштабування і розширення ігрового функціоналу. За результатами виконання було підтверджено, що обрана архітектура надає проекту зрозумілу структуру.

ABSTRACT

Bachelor’s thesis: 73 pages, 22 figures, 0 tables, 1 appendices, 20 sources.

UNITY, CASUAL, RUNNER, GAME ENGINE, C#, MVC PATTERN.

The purpose of the qualification work is to develop a game in the casual genre using the MVC pattern and based on the Unity game engine, which allows you to easily create structured, scalable and maintainable code.

During the work, the features of the casual game genre, namely the runner genre, the requirements for gameplay and user interface were analyzed. The use of the MVC pattern for separating logic, visualization and game state management was justified. A prototype of the game with basic mechanics, interface and event handling system was implemented. Functionality testing, performance optimization and preparation for further scaling and expansion of the game functionality were carried out. The results of the implementation confirmed that the chosen architecture provides the project with a clear structure.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

UI — User Interface, користувацький інтерфейс.

MVC — Model-View-Controller, архітектурний патерн.

IDE — Integrated Development Environment, інтегроване середовище розробки.

C# — C-Sharp, мова програмування, яка використовується в Unity.

Unity — Unity Game Engine, ігровий рушій, що використовується для створення застосунку.

API — Application Programming Interface, інтерфейс прикладного програмування.

ВСТУП

У сучасному світі стрімкого розвитку інформаційних технологій індустрія відеоігор стала однією з провідних у сфері цифрової економіки. Вона впливає не тільки на сферу розваг, а й на освіту, симуляційне моделювання та інші галузі, де потрібен високий рівень візуалізації та інтерактивність. Особливої популярності серед широкої аудиторії набули casual ігри, завдяки своїй орієнтованості на масового користувача, використанню простих механік, інтуїтивного управління й коротких ігрових сесій. Розробка ігор такого жанру вимагає балансування між простою і якісною реалізацією, що має певні вимоги до архітектури проекту, вибору інструментів та методів програмування.

Ігровий рушій Unity став одним із найпопулярніших середовищ розробки ігор різного рівня складності. Його функціональність, активна спільнота та велика база знань роблять цю платформу привабливим інструментом як для початківців, так і для досвідчених розробників. Однак, ефективне використання Unity потребує чіткої організації внутрішньої логіки гри. Серед різноманітних рішень щодо архітектурних підходів особливої популярності набув патерн MVC, що впроваджує зручність розробки, підтримки та масштабування коду. Використання цього патерну дозволяє уникнути змішування візуальної частини, логіки та обробки подій.

Основною метою кваліфікаційної роботи є створення структурованого прототипу з інтерактивною сценою, керованим персонажем, базовим інтерфейсом та системою перешкод.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика жанру Casual та Runner ігор

Протягом останніх років ігрова індустрія виказує тенденцію до зростання попиту на легкі у засвоєнні та легкозрозумілі проекти, саме тому casual ігри зайняли своє місце у галузі розваг, отримавши популярність серед користувачів різного віку, незалежно від їх ігрового досвіду та навичок. Особливістю таких ігор є низький поріг входу, мінімалізм у геймплейній логіці та адаптивність до різних пристроїв.

Ігри жанру casual розраховані на короткі сесії — від кількох секунд до кількох хвилин і не потребують занадто високої залученості гравця або детального вивчення правил гри. Вони можуть бути представлені головоломками, тайм-менеджмент симуляторами, clicker іграми, аркадами або runner іграми.

Жанр runner передбачає автоматичний рух персонажа вперед, де користувач реагує на перешкоди, бонуси або зміну середовища за допомогою невеликої кількості дій, зазвичай одного-двох різновидів натискання або свайпів. Найчастіше ігри жанру runner мають нескінчений режим гри, що мотивує гравців бити власні рекорди.

Також важливим фактором популярності casual ігор є їх доступність, іграм цього жанру притаманне низьке споживання ресурсів пристрою, завдяки чому вони стабільно працюють на системах із низькою продуктивністю, швидко завантажуються та не потребують постійного з’єднання з мережею. Проте сам ігровий процес залишається досить цікавим та може утримати увагу користувача на довший період, поступово підвищуючи складність і додаючи нові виклики.

Щоб краще зрозуміти характерні риси runner ігор варто проаналізувати вже існуючи рішення та визначити стандарти жанру на їх підставі[1].

Subway Surfers – можна сказати, що ця гра є найвідомішою у рамках свого жанру. У ній гравець керує персонажем, який поступово рухається вперед по залізниці, ухиляючись від поїздів та збираючи монети з бонусами. Управління реалізовано через свайпи вліво, вправо, вниз та вгору. Темп гри зростає із часом проведеним у межах однієї ігрової сесії, що підвищує рівень складності.

Рисунок 1.1 – ігровий процес Subway Surfers

Temple Run – ще один класичний представник жанру з тривимірним середовищем та виглядом від третьої особи. У ній гравець втікає від чудовиськ крізь стародавні руїни, долаючи повороти, стрибки та пастки. Механіка нагород на накопичування базується на монетах, зібраних під час усіх ігрових сесій, за ігрову валюту можна придбати нового персонажа, у якого є власні здібності.

Рисунок 1.2 – ігровий процес Temple Run

Jetpack Joyride – на відміну від попередніх ігор двовимірний runner, у якому керування персонажем відбувається завдяки тривалому натисканню на екран, від чого ігровий персонаж піднімається угору, якщо відпустити натиск, то персонаж буде летіти униз. Гра поєднує бонуси, щоденні місії та систему придбання аксесуарів для персонажу, що дозволяє підтримувати інтерес гравця протягом тривалого часу.

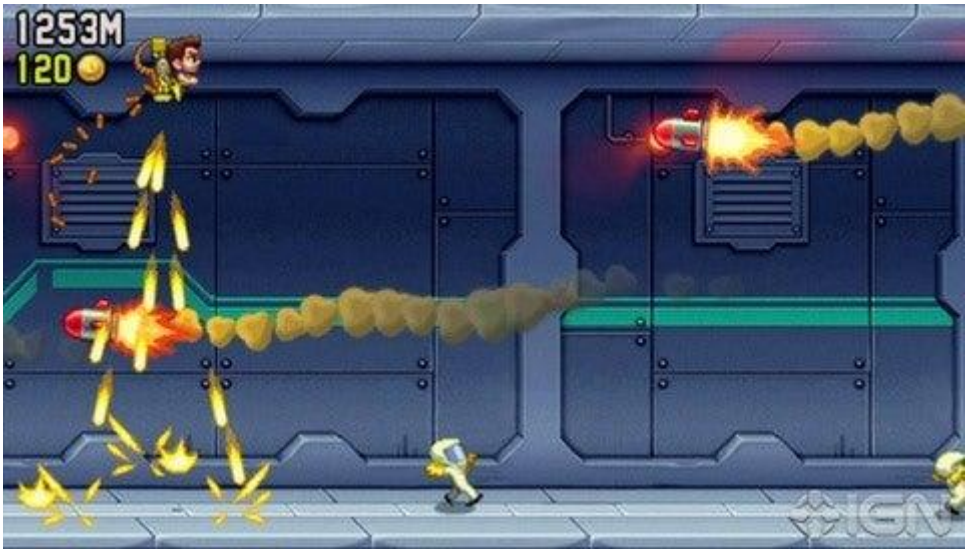

Рисунок 1.3 – ігровий процес Jetpack Joyride

Отже, на основі розглянутих представників жанру проглядаються наступні спільні риси runner ігор:

- Автоматичний рух персонажа вперед без потреби керування напрямком;
- Мінімалістичне управління, яке вкладається у кілька рухів або дій;
- Нескінченна генерація рівня, яка надає унікальності кожній сесії;
- Система балів, монет або досягнень, яка підтримує інтерес гравця;
- Гра побудована на ігрових механіках та відчутті ритму, без наявності сюжету;
- Поступове збільшення складності, а саме зростання темпу гри.

Особливість жанру runner полягає у тому, що при мінімальних правилах він здатен забезпечити цікавий ігровий процес, який зможе надовго затримати увагу гравця.

З технічного боку питання, реалізація runner-гри потребує продуманого підходу до обробки колізій, анімацій, переміщення об'єктів, створення інтерфейсу та системи очок. Все це робить ігри такого жанру дуже добрими для відпрацювання навичок розділення відповідальностей, впровадження архітектурних патернів і роботи в ігрових рушіях.

1.2 Архітектурні підходи у розробці ігор

Архітектура ігрового проекту є ключовою у забезпеченні структури, повторного використання коду, простоти обслуговування та масштабування коду. Серед найпоширеніших патернів у розробці ігор можна відокремити MVC, MVP, MVVM, MVI та концепцію clean architecture. Кожен з цих підходів має власні сильні сторони й обмеження, які мають враховуватись при обиранні структури проекту.

Патерн MVC (Model-View-Controller) вважається одним із найпростіших і найпоширеніших у розробці відеоігор[2]. Він передбачає чітке розділення на модель, подання та контролер. Основною перевагою цього патерна є відносна легкість у реалізації та швидке освоєння, особливо це

важливо для невеликих команд та індивідуальних розробників. Такий підхід дозволяє з самого початку вибудувати зрозумілу структуру коду, що особливо зручно в Unity, де кожен компонент може буте реалізовано у вигляді MonoBehaviour. Але, у великих проектах контролер часто занадто розростається и починає виконувати функції логіки, обробки подій і навіть рендеру, що суперечить принципам розподілу обов'язків, через це втрачається перевага чіткого поділу, а підтримка та розширення такого коду ускладнюється.

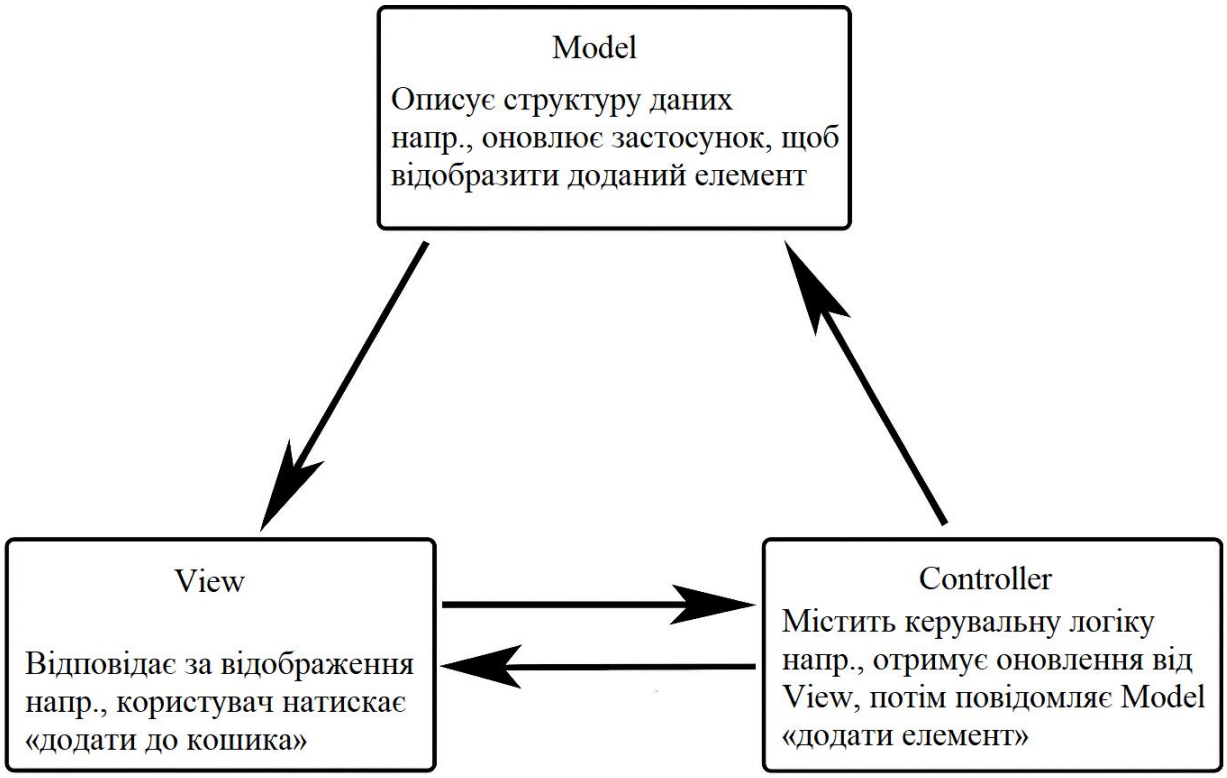

Рисунок 1.4 – архітектура MVC

Патерн MVP (Model-View-Presenter) був створений як відповідь на обмеження MVC. При такому підході логіку взаємодії між поданням та даними повністю оброблює компонент Presenter, що забезпечує більшу ізоляцію UI від логіки, тому процес тестування значно спрощується. У рушії Unity патерн MVP дозволяє винести логіку з MonoBehaviour класів, залишивши View відповідальним лише за відображення. Проте, такий підхід є

складнішим у реалізації та потребує більше часу на початкове проеткування. Для невеликих або простих проектів така архітектура може бути надмірною, а межа між Presenter і Model може виявитись нечіткою, через що ускладнюється підтримка у довгостроковій перспективі.

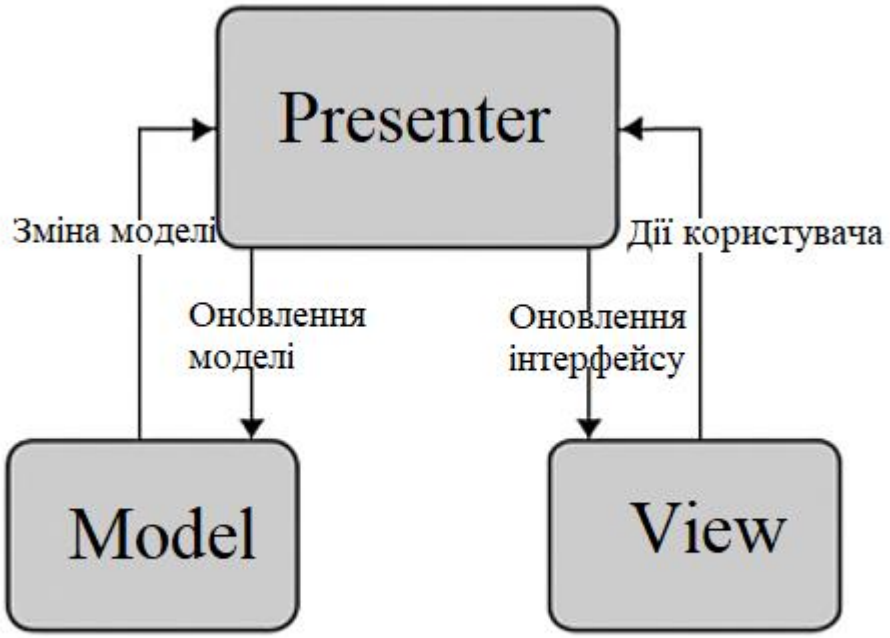

Рисунок 1.5 - Архітектура MVP

Патерн MVVM (Model-View-ViewModel) передбачає використання проміжного шару ViewModel, який відповідає за логіку відображення та підготовку даних до візуалізації. Всі зміни в інтерфейсі відбуваються через автоматичну синхронизацію з ViewModel. Цей патерн впроваджує дуже низьку зв’язність компонентів і високу тестованість. Водночас реалізація такого патерну на Unity є нетривіальною, оскільки для його впровадження потрібні додаткові бібліотеки, такі як UniRX або MVVM ToolKit. Для розробки простих ігор це стає невиправданим ускладненням процесу розробки, проте для проектів з великою кількість динамічних елементів інтерфейсу MVVM надає потужні можливості.

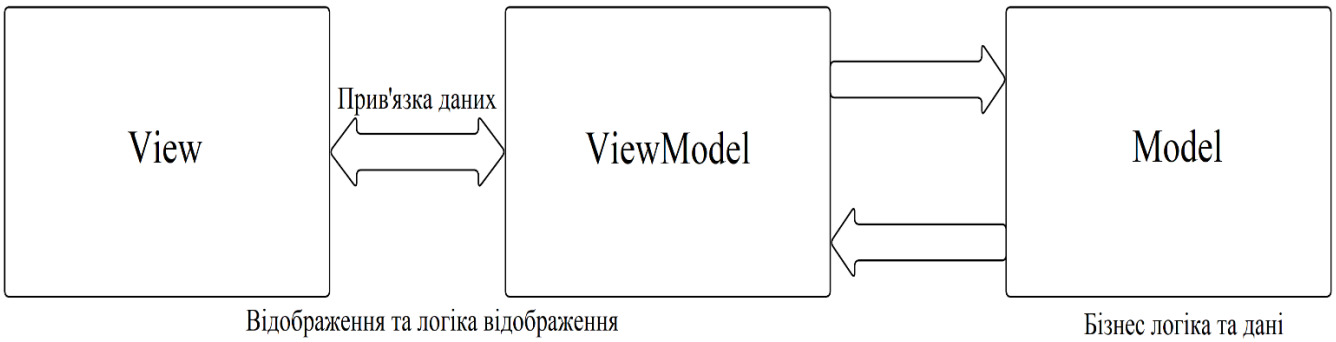

Рисунок 1.6 – архітектура MVVM

Патерн MVI (Model-View-Intent) – базується на уніфікації стану системи, кожна дія гравця розглядається як Intent, який змінює модель, а представлення оновлюється на основі нового стану. Така архітектура гарантує, що UI завжди відображає єдиний достовірний стан, що найкраще розкривається за умови великої кількості сценаріїв та подій. Реалізація патерну MVI потребує чіткого управління станами та розуміння реактивного програмування, що створює новачками та невеликим командам зайву складність, а для простих ігор такий підхід є досить формалізованим.

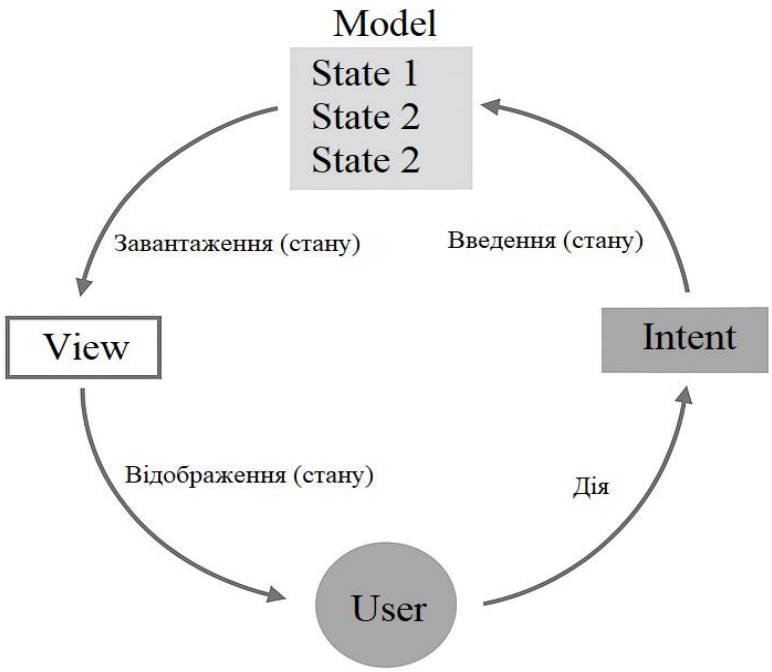

Рисунок 1.7 – Циклічна модель патерну MVI

Clean Architecture передбачає розподіл системи на незалежні шари: Domain (бізнес-логіка), Data (робота з джерелами даних), Presentation (візуалізація) та Infrastructure (інтеграція з платформою). Такий підхід дозволяє створювати проекти, що ізольовані від конкретного ігрового рушія.

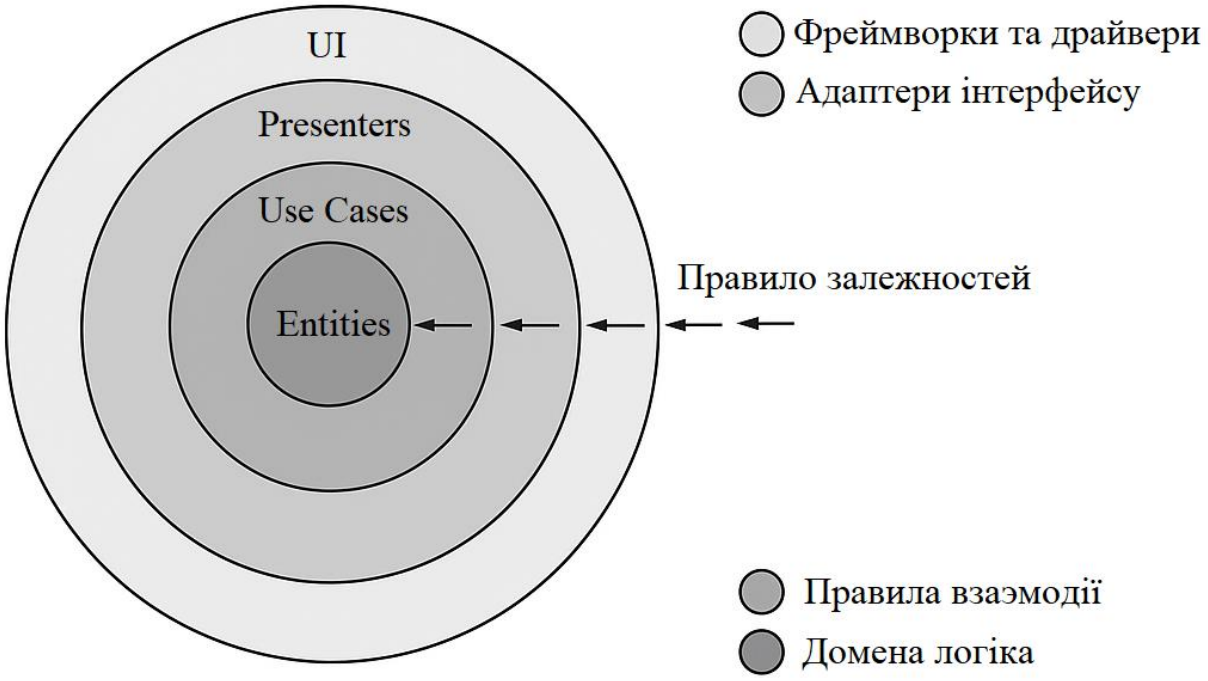

Рисунок 1.8 – структура Clean Architecture

Він забезпечує гнучкість, можливість автоматизованого тестування, масштабування коду, зміни рушія та повторного використання логіки в інших проектах. Але кількість абстракцій та складність реалізація роблять таку архітектуру недоцільною для навчальних або короткострокових проектів.

1.3 Огляд ігрових рушіїв

Ігровий рушій – це спеціальне програмне середовище, призначене для розробки інтерактивних застосунків, зокрема для мобільних та комп’ютерних ігор. За своєю суттю це набір різних спеціалізованих інструментів, бібліотек та сервісів, що забезпечують основні функціональні компоненти гри, а саме графіку, фізику, обробку вводу, рендеринг, управління ресурсами, а також мережеву взаємодію. Завдяки ігровим рушіям розробник працює не з чистим кодом, а з високорівневими конструкціями, які реалізують базову поведінку проекту.

Ключовою перевагою використання рушіїв є здатність абстрагувати низькорівневі технічні аспекти розробки і надати розробнику зручний інтерфейс для створення логіки, взаємодії та візуального оформлення. Також більшість сучасних рушіїв підтримують багатоплатформеність, завдяки чому один і той самий код може бути зібраний для різних операційних систем і пристроїв, серед яких персональні комп’ютери, мобільні телефони або консолі.

До архітектури типових ігрових рушіїв належать наступні компоненти:

- Графічний рушій – відповідає за рендеринг 2D або 3D сцен;
- Фізичний модуль – забезпечує моделювання колізій, сили тяжіння, імпульсів тощо;
- Звуковий рушій – обробляє аудіоефекти, фонову музику, просторове звучання;
- Анімаційна система – дозволяє керувати скелетною анімацією або перехідними станами;
- Система управління ресурсами – оптимізує завантаження і збереження текстур, моделей, звуків;
- Сценовий редактор – надає змогу візуального компонування;
- Інструменти скриптування – реалізують поведінку об’єктів і логіку гри.

Наразі найбільш популярними ігровими рушіями можна вважати Unity, Unreal Engine, Godot а також GameMaker Studio. Кожен із них має певну галузь застосування, особливості архітектури ті цільову аудиторію.

Unity – це кросплатформений рушій із підтримкою 2D та 3D графіки, який набув широкої популярності в інді-розробці. До ключових переваг рушія можна віднести його інтуїтивний візуальний редактор, компоненту архітектуру, гнучку систему анімації та можливість експорту проектів на понад 20 платформ[3]. Мовою програмування у середовищі є C#, яка дозволяє реалізувати як просту, так і складну логіку. Також Unity має цілу екосистему: тисячі різноманітних плагінів у Asset Store, активну спільноту, інтеграції з Visual Studio, системи профілювання та аналітики.

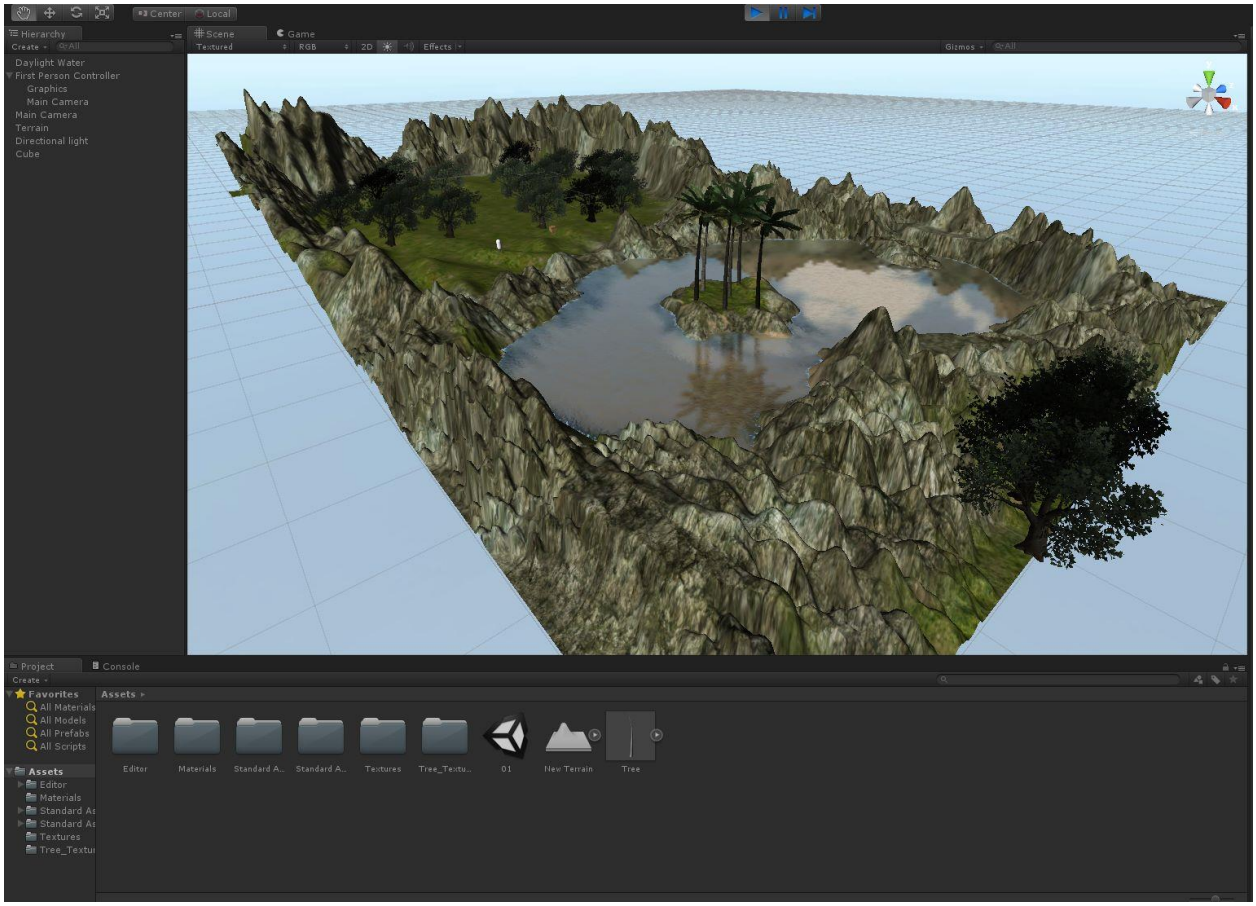

Рисунок 1.9 – робочий простір Unity

Unreal Engine – рушій, розроблений компанією Epic Games, відомий своїм високоякісним рендерингом та підтримкою фотореалістичної графіки.

Найбільш активно використовується в AAA проектах, віртуальних симуляціях та VR проектах. На відміну від Unity використовує мову C++ та має підтримку візуального скриптування через систему Blueprints, яка дозволяє створювати логіку гри без написання коду. Також середовище має передові системи освітлення, системи Lumen і Nanite, які забезпечують кінематографічну якість в реальному часі. Проте, Unreal Engine є дуже складним та має високі системні вимоги, тому не є кращим рішенням для невеликих проектів.

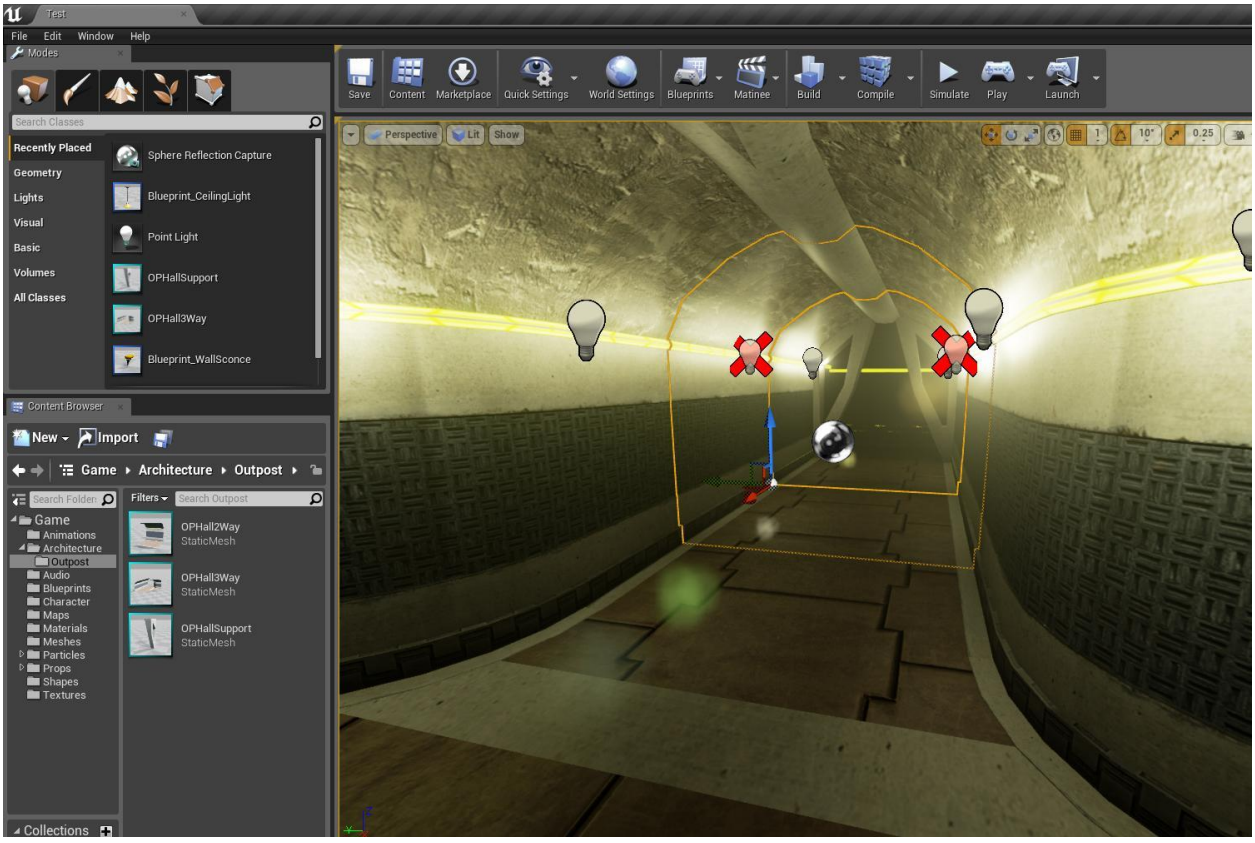

Рисунок 1.10 – робочий простір Unreal Engine

Godot – відкритий рушій з повністю безкоштовною ліцензією MIT. Серед переваг виділяють низький поріг входу, невеликий розмір (вагу), вбудований редактор сцен, а також підтримку мови GDScript – спрощеного синтаксису схожого на Python. Godot активно розвивається завдяки своїй спільноті і хоча цей рушій має обмеженні порівняно із Unity або Unreal Engine 3D-можливості, у 2D сегменті він демонструє гнучкість та продуктивність. В останніх версіях продукту реалізована підтримка C# та Vulkan API.

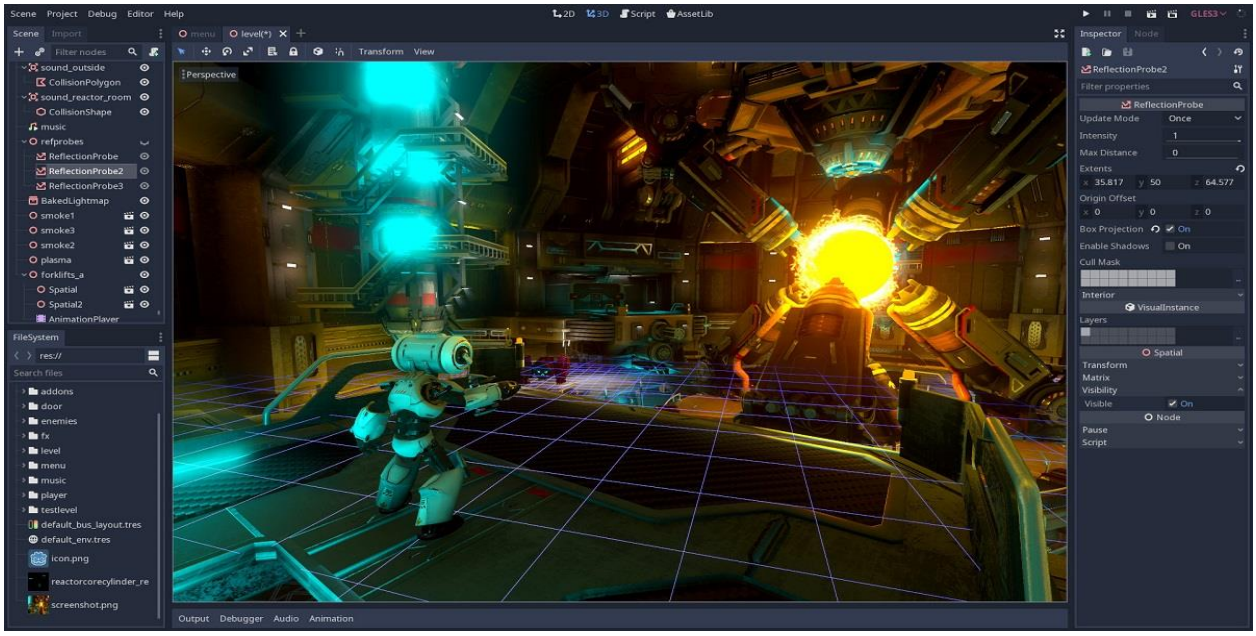

Рисунок 1.11 – робочий простір Godot

GameMaker Studio – рушій орієнтований на створення 2D ігор. Сильними сторонами є простий редактор та власна мова GML, яка дозволяє досить швидко реалізовувати ігрову логіку. Цей рушій ідеально підходить для створення аркад, головоломок, платформерів, але не має повноцінної підтримки 3D. Завдяки своїй простоті GameMaker часто використовують для навчання та створення перших ігрових проектів.

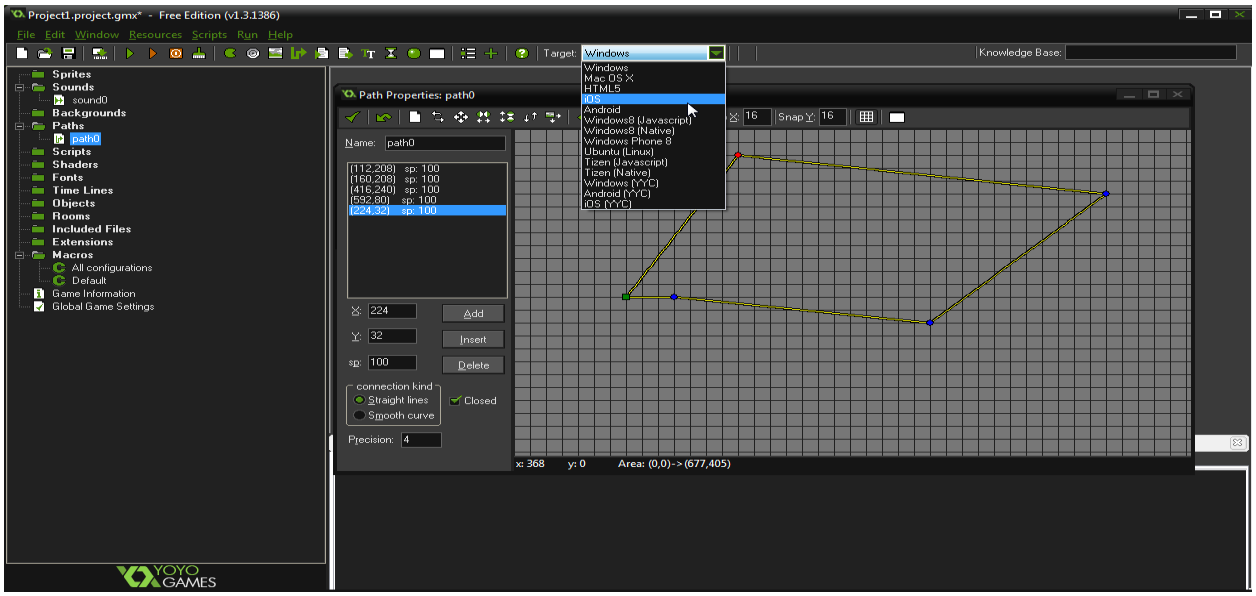

Рисунок 1.11 – робочий простір GameMaker Studio

Отже, ігрові рушії відіграють роль фреймворку, який надає розробникам всі базові механізми для створення ігрових продуктів. Вибір рушія для проекту визначається складністю, цільовою платформою, типом графіки 2D або 3D, вимогами до продуктивності та досвіду команди.

1.4 Постановка задачі

У межах кваліфікаційної роботи ставиться завдання створення ігрового прототипу у жанрі casual runner, побудованого на основі рушія Unity з реалізацією структури за архітектурним патерном MVC. Такий підхід дозволяє чітко відокремити логіку гри, візуальне представлення та керування подіями, що спрощує підтримку, налагодження та подальший розвиток проекту.

Метою роботи є розробка нескінченної runner-гри з базовим набором функціональних елементів, а саме автоматичний рух персонажа вперед, динамічна генерація бонусів та перешкод, обробка колізій, нарахування очок, відображення UI. Для досягнення поставленої мети необхідно реалізувати такі підзадачі:

- Створення інтерактивної сцени з використанням візуального редактора Unity;
- Розробка системи управління персонажем;
- Реалізація механізму генерування ігрового середовища;
- Організація обліку очок як основного показника прогресу;
- Побудування UI з елементами стартового меню, повідомлення про поразку та можливістю перезапуску гри.

Також слід враховувати технічні особливості рушія Unity, уся логіка має бути реалізована мовою C#, а організація коду відповідати принципам читабельності та розширюваності.

Очікуваний результат – завершений навчальний прототип, що

демонструє базові принципи побудови жанру runner, реалізований у сучасному ігровому середовищі із дотриманням архітектурних стандартів. Отриманий досвід може бути масштабований у майбутніх проектах і стане основою для більш складних розробок із додатковими функціональними можливостями.

2. ОСНОВНІ ІНСТРУМЕНТИ ТА СЕРЕДОВИЩЕ РОЗРОБКИ

2.1 Обґрунтування вибору рушія Unity

У процесі розробки ігор вибір рушія визначає не лише технічні можливості проекту, а й швидкість самої розробки, зручність, підтримку платформ та доступність ресурсів. Для реалізації поставленої мети було прийнято рішення використовувати рушій Unity, який є одним із найпопулярніших інструментів у сфері розробки інтерактивних додатків.

Unity – це мультиплатформений ігровий рушій, розроблений компанією Unity Technologies. Ключовою особливістю рушія є можливість створювати 2D та 3D ігри для великої кількості цільових платформ, серед яких Windows, MacOS, Android, iOS, WebGL, а також консолі[4]. Саме універсальність, відносна легкість у використанні та наявність готових модулів роблять Unity зручним вибором для розробників як комерційних проектів, так і навчальних інді-розробок.

Перш за все, рушій має компонентно-орієнтовану архітектуру, яка ідеально підходить для реалізації патерну MVC. У Unity кожен об’єкт сцени може мати довільну кількість компонентів, що дозволяє розділити логіку, візуальне представлення та контроль поведінки без зайвих залежностей. Такий підхід забезпечує гнучкість у проектуванні та легкому масштабуванні та надає можливість додавати нову функціональність або змінювати поведінку об’єктів без переписування вже існуючого коду.

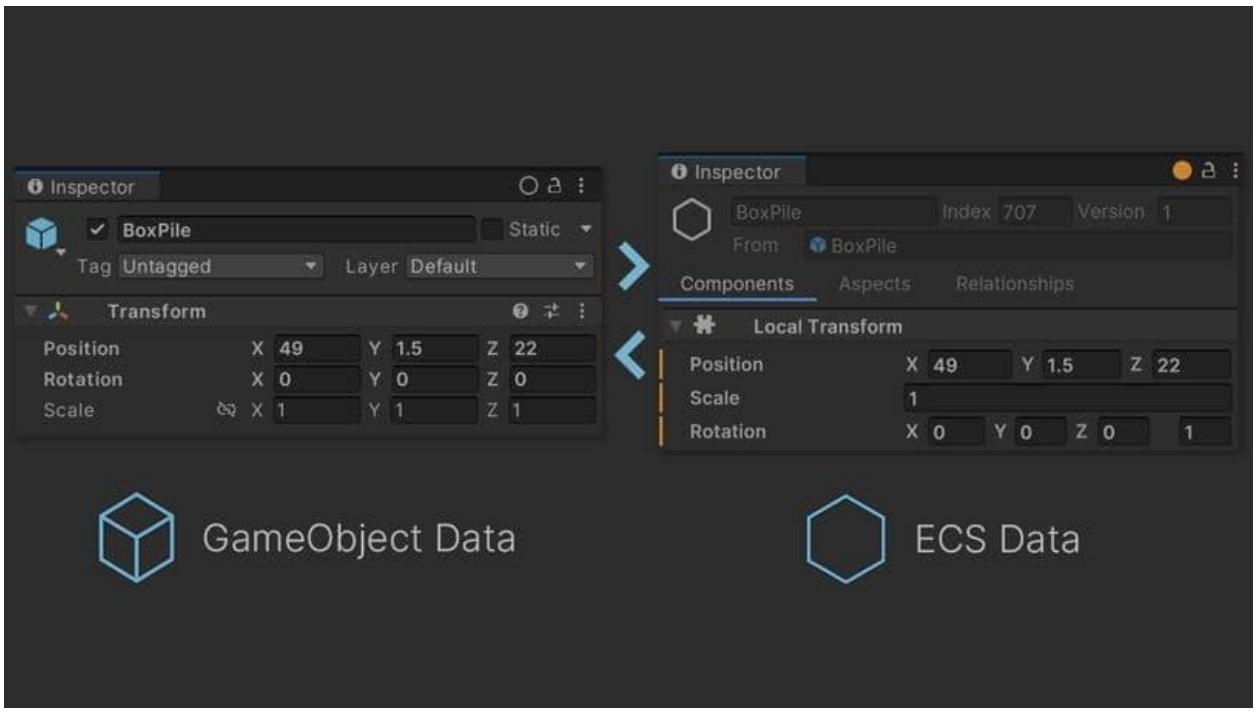

Рисунок 2.1 – принцип роботи компонентно-орієнтованої архітектури

По-друге, Unity підтримує високий рівень абстракції та візуалізації, що значно зменшує час, витрачений на створення сцени, налаштування камери, освітлення, а також UI-елементів. Усі зміни перевіряються одразу за допомогою інтерактивного редактора без необхідності повної компіляції проекту. Це значно прискорює ітерації розробки та дозволяє швидко вносити зміни та поправки в ігровий процес або інтерфейс.

Також, рушій забезпечує розвинену систему фізики, що є важливим компонентом runner ігор, де присутня обробка зіткнень, швидкості, руху. Вбудовані фізичні модулі спрощують реалізацію взаємодії об’єктів, що є ключовим для динаміки ігрового процесу у рамках жанру.

Окрім того, Unity має величезну базу матеріалів, яка постійно оновлюється, активну спільноту, відкриту документацію й офіційний магазин готових рішень AssetStore, де наявні як платні, так і безкоштовні ресурси для прискорення процесу розробки. Усі ці фактори важливі для індивідуальної роботи в умовах обмеженого часу.

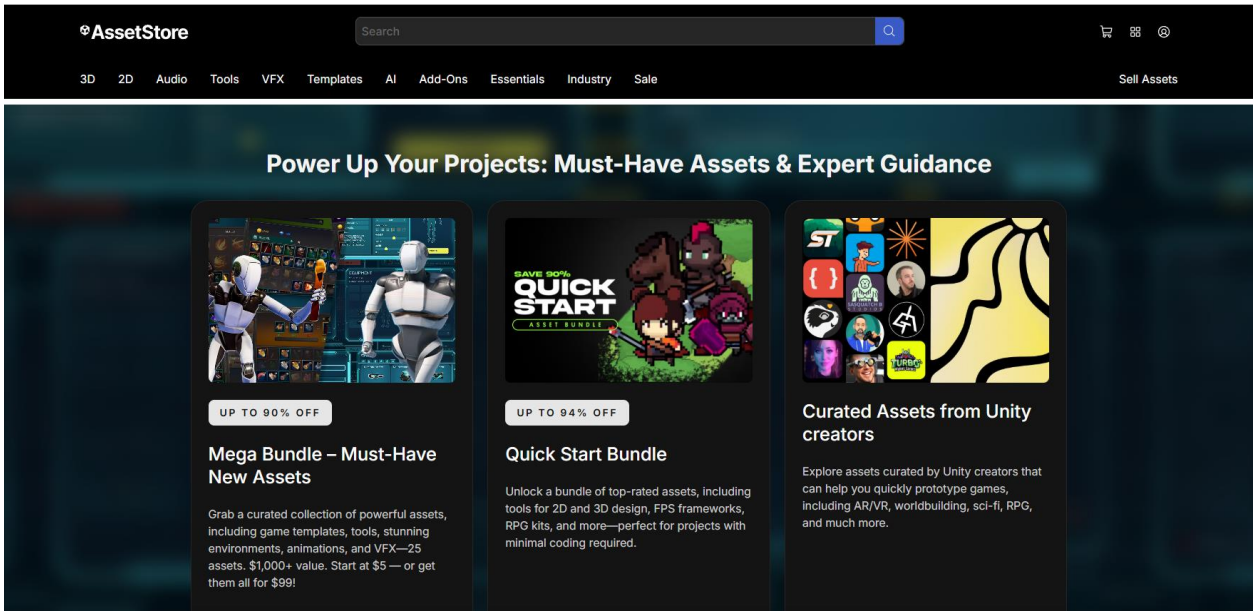

Рисунок 2.2 – головна сторінка AssetStore

Важно зазначити, що рушій підтримує мову C#, яка надає можливість використовувати сучасні об’єктно-орієнтовані підходи, працювати з подіями, делегатами та асинхронними операціями. Взаємодія коду з компонентами Unity відбувається через класи, які успадковують MonoBehaviour. MonoBehaviour є базовим класом для всіх скриптів, які можуть бути прикріплені до ігрових об’єктів. Такий підхід дозволяє скриптам отримати доступ до функціональності рушія, такої як життєвий цикл об’єкта, рендеринг, компоненти фізики тощо.

Як альтернативні рушії також розглядались Godot та Unreal Engine, проте Godot поступається в підтримці мобільних платформ та обсягах документації, а Unreal орієнтований переважно на складні проекти з великими бюджетами, тому він би ускладним реалізацію поставленої мети.

Отже, вибір Unity, як основи для створеної casual runner гри, був обумовлений доступністю, широкими можливостями, гнучкістю компонентної архітектури та зручністю у реалізації невеликих проектів.

2.2 Основи роботи з Unity: компоненти, сцени, об’єкти

Розуміння базових понять Unity є критично важливим для розробки гри, тому цей підрозділ присвячено опису основних структурних елементів рушія, такі як сцени, об’єкти, компоненти, а також принципам їх взаємодії під час розробки інтерактивного застосунку.

У середовищі Unity гра будується із так званих сцен, які є логічно відокремленими частинами, що представляють різні компоненти гри, такі як ігрові рівні, меню або інші стани додатку. Кожна сцена містить набір об’єктів, які формують середовище, ігрову логіку, візуальні елементи тощо. У розробників є можливість перемикатися між сценами або динамічно їх завантажувати під час виконання програми.

Центральним елементом кожної сцени є ігровий об’єкт, він являє собою базову сутність, яка сама по собі не має ніякої заданої поведінки чи властивостей, але слугує контейнером для компонентів. Кожен такий об’єкт може містити будь-яку кількість компонентів, що додають йому функціональності.

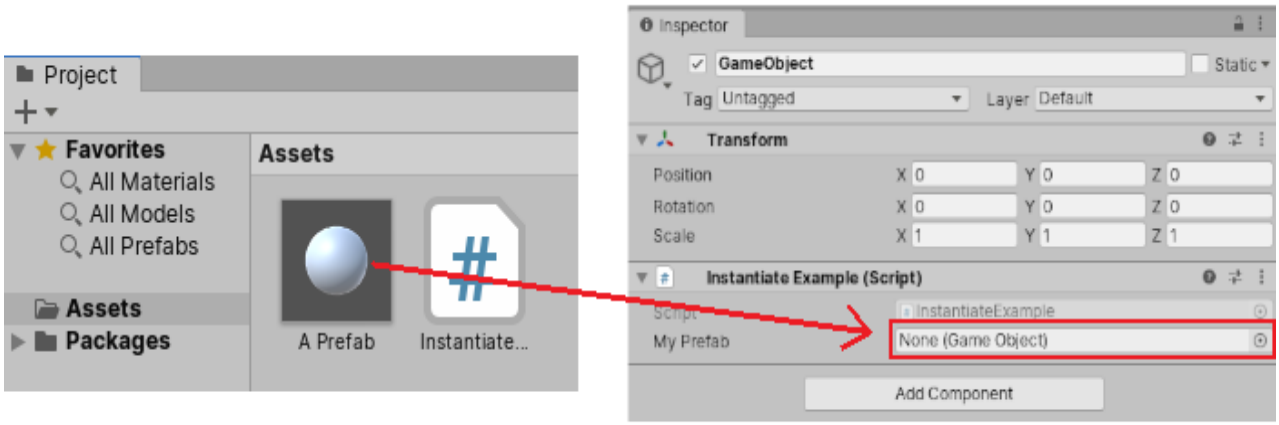

Рисунок 2.3 – базовий ігровий об’єкт

Кожен ігровий об’єкт має компонент Transform. Цей компонент визначає положення, обертання та масштаб об’єкта у тривимірному просторі

сцени. Усі компоненти, що додаються до об’єкту базуються на його Transform.

Таким чином, у Unity реалізовано компонентно-орієнтовану архітектуру, де поведінка визначається не через спадкування класів, як у класичних об’єктно орієнтованих мовах, а шляхом додавання потрібних компонентів до ігрового об’єкту. Такий підхід дозволяє гнучко керувати логікою проекту та уникати надмірної складності в ієрархії класів. Усі додані компоненти та їхні властивості можна переглядати та налаштовувати в вікні Inspector, завдяки якому забезпечується швидке прототипування та налаштування.

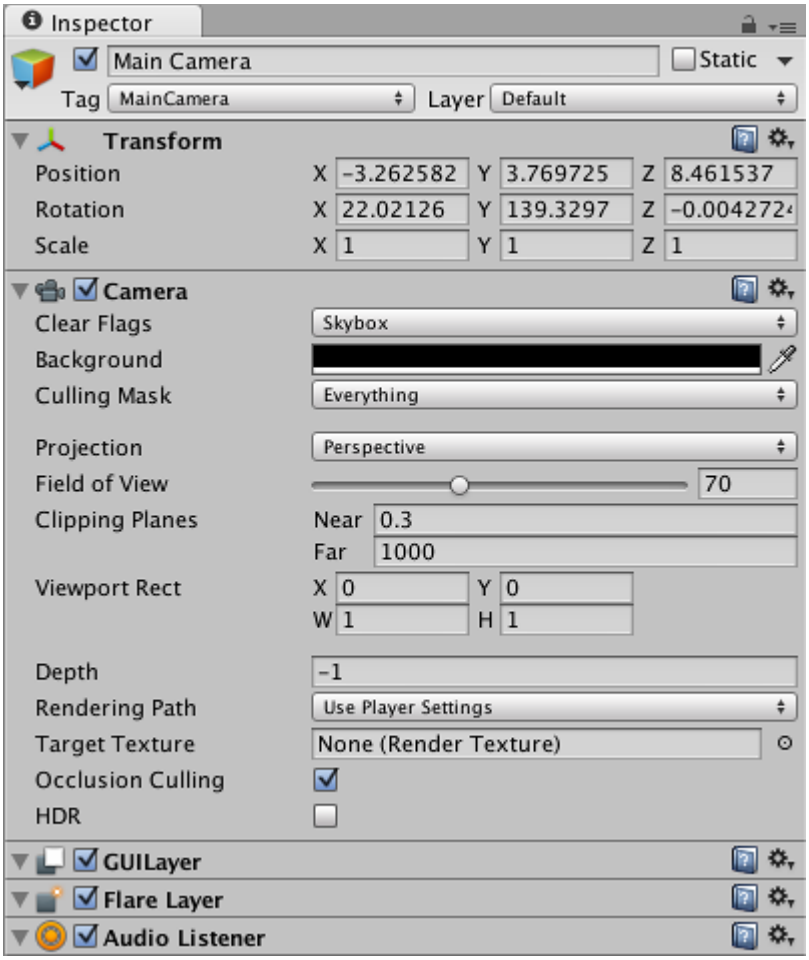

Рисунок 2.4 – вікно Inspector для ігрової камери

Також, одним із ключових компонентів рушія є скрипти, які впроваджують ігрову логіку. Всі користувацькі скрипти успадковуються від базового класу MonoBehaviour. Класи, що наслідують MonoBehaviour, можуть реалізовувати спеціальні методи життєвого циклу об’єкта, серед яких:

- Awake() – викликається один раз при завантаженні сцени або активації об’єкта, використовується для ініціалізації компонентів та встановлення посилань;
- Start() – викликається один раз перед першим кадром оновлення, якщо скрипт активований, ідеально підходить для початкової ігрової логіки;
- Update() – викликається щокадрово і використовується для динамічних змін, що не залежать від фізики;
- FixedUpdate() – викликається через фіксовані інтервали часу і використовується для роботи з фізикою та іншими розрахунками, чутливими до часу;
- OnTriggerEnter2D() – реагує на зіткнення у 2D середовищі, коли об’єкт виходить за зону тригера.

Усі ці методи інтегровані з ігровим циклом Unity, який автоматично обробляє оновлення сцен, перегляд подій, фізичні взаємодії та інші аспекти життєвого циклу об’єктів. Завдяки цьому розробники можуть витрачати час та сили на логіку гри, не переймаючись технічними деталями реалізації основного циклу.

Окрім внутрішніх компонентів, Unity надає можливість створення префабів – шаблонів об’єктів, які можна багаторазово використовувати в різних частинах сцени або в різних сценах. Префаби особливо зручні для об’єктів, що часто повторюються, наприклад, перешкоди або бонуси в runner іграх, оскільки вони дозволяють централізовано керувати їхніми властивостями та швидко вносити зміни.

Ще одним важливим аспектом є система UI, яка дозволяє створювати графічний інтерфейс за допомогою спеціальних компонентів, таких як Canvas, Text, Button, Image та інших. Інтерфейс розміщується окремо від ігрового середовища, але може динамічно реагувати на події гри — наприклад, відображати кількість зібраних балів або анімацію поразки.

Основною робочою середовищем для розробника в Unity є Unity Editor[5]. Це комплексний інтегрований інструмент, що об'єднує всі необхідні засоби для візуального конструювання, налаштування та тестування гри. Unity Editor складається з декількох ключових вікон (панелей), кожне з яких виконує свою функцію:

Scene View — дозволяє створювати та маніпулювати об'єктами у тривимірному або двовимірному просторі сцени.

Game View — відображає кінцевий вигляд гри, яким його бачить гравець, і дозволяє тестувати геймплей.

Hierarchy Window — показує список всіх ігрових об’єктів у поточній сцені в ієрархічному порядку.

Project Window — дозволяє керувати всіма ресурсами проекту, такими як скрипти, моделі, текстури, звуки, матеріали тощо.

Inspector Window — відображає властивості та налаштування обраного GameObject або компонента, дозволяючи їх редагувати безпосередньо.

Console Window — виводить системні повідомлення, помилки, попередження та власні лог-повідомлення, що є незамінним для налагодження.

Рисунок 2.5 – Приклад вигляду Unity Editor в ігровому проекті

Всі ці вікна інтегровані у єдину, гнучку робочу область, що дозволяє розробнику ефективно працювати з різними аспектами проекту, візуально розташовувати елементи, налаштовувати компоненти та миттєво перевіряти зміни. Така візуально-орієнтована середа суттєво спрощує процес розробки, особливо на початкових етапах, та підвищує продуктивність.

Таким чином, Unity надає повний набір засобів для побудови ігрової сцени, визначення її логіки, управління об’єктами та реалізації взаємодії з користувачем. Всі ці елементи у комплексі дозволяють створювати сучасні, інтерактивні та кросплатформенні проекти з високим рівнем деталізації, навіть у межах індивідуальної дипломної роботи.

2.3 Мова програмування C# у рушії Unity

Мова програмування є головним інструментом для реалізації ігрової логіки, управління подіями, роботи з даними та забезпечення інтерактивності гри. У середовищі Unity мовою програмування э C#, яка поєднує зрозумілий синтаксис, високу продуктивність і гнучкість.

Мова C# стала невід'ємною частиною екосистеми Unity завдяки низці ключових переваг, які роблять її ідеальною для розробки ігор. Насамперед, вона повноцінно підтримує об'єктно-орієнтовані принципи, такі як класи, об’єкти, успадкування, поліморфізм та інкапсуляція. Це дає змогу розробникам створювати модульну та легко розширювану архітектуру гри, де кожен ігровий елемент — будь то персонаж, ворог чи предмет — може бути представлений як об'єкт зі своїми унікальними властивостями та поведінкою. Крім того, ця мова є строго типізованою. Ця особливість допомагає виявляти багато потенційних помилок ще на етапі компіляції, що суттєво знижує ймовірність збоїв під час виконання гри та спрощує процес налагодження. Важливою перевагою також є її розвинена екосистема. Вона включає потужну стандартну бібліотеку .NET, широку підтримку від Microsoft та величезну спільноту розробників[6]. Усе це забезпечує доступ до численних

фреймворків, інструментів, детальної документації та ресурсів, необхідних для вирішення практично будь-яких завдань, що виникають у процесі розробки. Нарешті, C# ідеально інтегрована з програмним інтерфейсом Unity. Ця інтеграція дозволяє розробникам ефективно взаємодіяти з усіма елементами рушія — від трансформації об'єктів та обробки вводу до рендерингу та фізики, забезпечуючи безперебійний та інтуїтивно зрозумілий робочий процес.

У Unity C# використовується для написання скриптів, які є компонентами та прикріплюються до ігрових об’єктів. Всі такі скрипти є класами, які успадковують базовий клас MonoBehaviour, що забезпечує їхню інтеграцію у життєвий цикл Unity та доступ до спеціальних методів, які автоматично викликаються рушієм[7]. Такий механізм дозволяє зручно будувати як імперативну, так і реактивну модель поведінки об’єктів.

Важливою частиною роботи з C# у Unity є використання делегатів та подій, які забезпечують зручний та гнучкий спосіб обробки взаємодії між об'єктами[8]. Це особливо актуально в іграх, де численні об'єкти мають реагувати на зміну стану (наприклад, коли персонаж стикається з перешкодою, гравець збирає бонус або досягає певного балу), мінімізуючи жорсткі зв'язки між компонентами.

Ще одна потужна можливість — серіалізація змінних через public поля або атрибути SerializeField. Це дозволяє виводити значення змінних у редакторі Unity, що значно спрощує налаштування поведінки об’єктів, балансування ігрового процесу та швидке прототипування без необхідності щоразу компілювати код[9].

Серед конструкцій C#, що активно використовуються у розробці ігор, варто виділити:

Цикли: для ітерації по колекціях, створення безлічі перешкод, елементів рівня або управління списками гравців;

Умовні оператори: для перевірки ігрових станів, умов перемоги/поразки, вибору дії гравцем або реакції на різні типи зіткнень;

Колекції: для ефективного зберігання та управління великою кількістю ігрових об’єктів, даних інвентарю, таблиць лідерів тощо;

Асинхронне програмування: забезпечує можливість виконання тривалих операцій (наприклад, завантаження великих ресурсів, запитів до мережі, збереження/завантаження даних) без блокування основного циклу гри, що гарантує плавність та відсутність "зависань"[10].

Таким чином, мова C# не лише забезпечує технічну основу для взаємодії з рушієм Unity, але й надає всі необхідні інструменти для реалізації повноцінної, модульної, масштабованої та високопродуктивної логіки гри.

3 АРХІТЕКТУРНА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Загальна структура гри та архітектура

Архітектура гри реалізована з урахуванням принципів шаблону проектування MVC (Model–View–Controller). Даний підхід дає змогу чітко розділити внутрішню логіку програми, графічний інтерфейс користувача та обробку подій. Патерн MVC значно полегшує супровід коду, тестування, а також майбутнє масштабування проекту.

Рисунок 3.1 – UML діаграма проекту

Гра належить до жанру runner, де основна механіка полягає в безперервному русі гравця вперед із можливістю уникнення перешкод, збирання бонусів і досягнення найвищого результату. Користувач керує персонажем через свайпи, які розпізнаються як команди зміни смуги руху, стрибка або ковзання.

На високому рівні проект складається з трьох логічних підсистем:

Model — відображає стан гри, включаючи очки, монети, поточне становище персонажа та інші змінні стану.

View — реалізує візуальне відображення гри: інтерфейс користувача, рух камери, екран завершення гри тощо.

Controller — опрацьовує вхідні події, керує логікою руху персонажа, генерує перешкоди та оновлює модель відповідно до дій користувача.

Комунікація між компонентами відбувається згідно з принципами шаблону MVC: контролери зчитують події та змінюють модель, модель повідомляє подання про зміни, а подання оновлюється на основі поточного стану.

Ключову роль у взаємодії відіграє клас PlayerController, який є центральним елементом і пов’язує більшість компонентів гри. Він керує станом гравця, обробляє події зіткнення, взаємодіє з моделлю рахунку (ScoreModel) та моделю стану (GameStateModel), а також ініціює оновлення інтерфейсу через в’ю-компоненти (ScoreView, GameUIView).

Для зчитування жестів реалізований окремий контролер — SwipeInputController, який відстежує свайпи та передає відповідні сигнали PlayerController.

Усі візуальні компоненти винесено в окремі класи, що підпорядковуються виключно відображенню даних. Наприклад, ScoreView займається оновленням числових значень очок та монет у UI, CameraView слідкує за позицією гравця, а GameUIView відповідає за відображення панелі поразки або інших елементів, пов’язаних із завершенням гри.

Також важливо відзначити роль GameWorldController, який генерує та

керує елементами ігрового світу (перешкоди, платформи, бонуси). Цей клас працює на основі позиції гравця, що зберігається в моделі GameStateModel.

Таке чітке розділення обов'язків забезпечує слабке зв'язування між компонентами, підвищує тестованість і дозволяє легко змінювати або оновлювати окремі частини гри (наприклад, змінити логіку рахунку або інтерфейс користувача) без необхідності модифікації інших компонентів.

Таким чином, архітектура побудована згідно з принципами сучасного розроблення ігор, орієнтованого на масштабованість, повторне використання компонентів та легкість обслуговування. Структура гри чітко поділяється на три окремі сфери відповідальності, кожна з яких ізольована від інших через контрольовані інтерфейси взаємодії.

3.2 Model частина

Model частина відповідає за зберігання, оновлення і надання стану гри, не маючи прямого зв'язку з відображенням чи користувацьким вводом. У структурі проекту модельна частина реалізована за допомогою двох ключових класів: ScoreModel та GameStateModel.

Клас ScoreModel відповідає за облік прогресу гравця під час гри. Він зберігає поточний рахунок, представлений змінною totalScore, а також кількість зібраних монет, що фіксується в полі coins. Крім того, у моделі передбачено змінну multiplier, яка визначає коефіцієнт множення очок і може змінюватися внаслідок активації бонусів, таких як зірка. Для оновлення значень рахунку й монет реалізовані методи AddScore() та AddCoin(), які дозволяють коректно змінювати відповідні поля моделі. Для доступу до актуального стану використовуються методи GetScore() та GetCoins(), що повертають поточні значення безпосередньо іншим компонентам гри. Така реалізація забезпечує інкапсуляцію даних і дозволяє контролювати зміну ігрової статистики через чітко визначений інтерфейс.Важливо, що ScoreModel не взаємодіє з графічним інтерфейсом і не містить логіки відображення. Його

єдине завдання — підтримувати достовірний стан рахунку гравця.

Клас GameStateModel зосереджений на збереженні ігрового стану персонажа під час сесії. Він інкапсулює такі параметри, як активність бонусу безсмертя, яка визначається логічним полем isImmortal, стан ковзання персонажа, що фіксується через змінну isSliding, а також поточну смугу руху, яка задається у змінній currentLane і відображає положення гравця відносно трьох доступних смуг: лівої, центральної або правої. Ця модель дозволяє іншим компонентам гри звертатися до ключових характеристик стану гравця через публічні методи, не змінюючи їх безпосередньо, що забезпечує контрольовану зміну станів і дотримання принципів інкапсуляції. Методи ResetState(), SetImmortal(bool) та Slide() дозволяють керувати внутрішнім станом моделі в реакції на події гри. Наприклад, при активації бонусу щита контролер оновлює isImmortal, після чого цей стан використовується при обробці зіткнень.

Обидва класи — ScoreModel і GameStateModel — ізольовані від решти системи, а їхні поля доступні лише через публічні методи. Такий підхід забезпечує інкапсуляцію та дозволяє контролерам взаємодіяти з моделлю в суворо визначений спосіб. У майбутньому це спростить автоматизоване тестування, оскільки модель не залежить від Unity API або графічного інтерфейсу.

Крім того, можлива подальша розширюваність: наприклад, GameStateModel можна доповнити полем isJumping, або ж додати механізм збереження стану між сесіями гри.

3.3 View частина

Підсистема View у патерні MVC відповідає виключно за візуальне відображення даних та реагування на зміну стану моделі. У раннер-грі реалізовано кілька окремих візуальних компонентів, кожен з яких виконує свою чітко визначену функцію.

ScoreView керує відображенням числових показників — рахунку та кількості зібраних монет. Він взаємодіє з UI-компонентами Unity і надає публічні методи UpdateScore(int) та UpdateCoins(int).

Оновлення текстових полів відбувається виключно через виклики контролера, який отримує оновлені значення з ScoreModel. Таким чином, ScoreView не зберігає жодного власного стану, що відповідає принципам чистої View.

GameUIView відповідає за управління панелями інтерфейсу, які відображаються у відповідь на події. Найголовніший приклад — показ панелі поразки (losePanel). Після зіткнення гравця з перешкодою контролер викликає метод ShowLosePanel(), активуючи відповідний UI-елемент у сцені.

Важливо, що GameUIView не перевіряє стан гри самостійно — він реагує лише на команди від контролера, що забезпечує розділення обов’язків.

CameraView слідкує за персонажем, адаптуючи положення камери відповідно до зміни координат гравця. З технічної точки зору, він оновлює позицію камери у методі LateUpdate() або FixedUpdate(), зберігаючи фіксоване відносне зміщення до Transform гравця.

Камера не знає, що таке "гравець" у логічному сенсі гри, вона лише фокусується на трансформації, яку їй передано.

MainMenuView обробляє стартове меню, зчитуючи та відображаючи статистику з попередніх ігрових сесій. Дані, що виводяться у bestScoreText і coinsText, беруться безпосередньо з ScoreModel, через доступні методи або PlayerPrefs, якщо модель зберігає дані між сесіями.

Усі компоненти View не містять ігрової логіки. Вони не приймають рішення та не змінюють стан гри. Їхня єдина функція — візуалізація. Це дозволяє легко підміняти або змінювати інтерфейс без втручання в логіку, наприклад, для адаптації під мобільний інтерфейс, або для локалізації.

3.4 Controller частина

Контролери є центральною ланкою архітектури, що забезпечує зв’язок між користувацьким вводом, логікою гри та візуалізацією. У проекті реалізовано три основних контролери.

Клас PlayerController є центральним елементом управління у грі, який поєднує в собі обробку вводу, керування рухом персонажа та взаємодію з іншими компонентами системи. Він зчитує жести користувача за допомогою SwipeInputController, визначаючи напрямки свайпів, які використовуються для перемикання смуги, стрибка або ковзання. На основі введення PlayerController керує переміщенням персонажа між смугами, ініціює анімації стрибків і перекатів, а також коригує фізичні параметри, наприклад, розміри колайдера під час ковзання. Крім цього, він виконує перевірку зіткнень із перешкодами, реагує на контакти з бонусами — зокрема, активує щит або множник очок — і відповідно оновлює стан моделей. Після взаємодії з ігровими об’єктами PlayerController змінює рахунок у ScoreModel та передає оновлені дані до інтерфейсних компонентів, таких як ScoreView і GameUIView, що забезпечує актуальне відображення інформації на екрані. Таким чином, цей клас виступає координатором між вводом, логікою та візуальним представленням гри.

Він працює безпосередньо з моделями, змінюючи стан через методи типу SetImmortal() або AddScore(), і не звертається до полів моделей напряму.

SwipeInputController ізольовано відповідає за обробку введення. Він працює як адаптер: обробляє свайпи (жести пальцем або мишею), визначає напрямок руху, а потім надає контролеру PlayerController доступ до булевих прапорів (swipeLeft, swipeUp тощо).

Цей підхід дозволяє легко змінювати спосіб вводу без змін у логіці гри. Наприклад, для десктопної версії можна адаптувати цей клас під клавіатуру.

GameWorldController керує генерацією та знищенням ігрових об’єктів — плиток, бонусів, перешкод. Він орієнтується на позицію гравця (отриману з GameStateModel) для вирішення, коли необхідно спавнити нову частину рівня

або видалити застарілу. Також він може опрацьовувати випадкову генерацію об’єктів за допомогою окремих компонентів на зразок RandomSpawn.

Кожен контролер має обмежену сферу відповідальності. Всі дії, що змінюють стан гри, ініціюються саме через контролери. Таким чином досягається високий рівень розділення логіки і спрощення налагодження коду.

Контролери — єдині, хто знають про наявність інших класів. View та Model ізольовані між собою, і це є ключовим аспектом реалізації правильного MVC.

3.5 Опис роботи скриптів проекту

Ключовим елементом ігрової логіки виступає контролер персонажа — PlayerController (лістинг 1.1), який координує стан гравця, рух по доріжках, активацію бонусів та взаємодію з іншими компонентами гри. У відповідності до шаблону MVC, цей скрипт не зберігає ігровий стан безпосередньо, а взаємодіє з окремими моделями — ScoreModel (рахунок, монети) та GameStateModel (смуга, стан безсмертя, ковзання).

Рух гравця реалізовано через CharacterController, що дозволяє плавно переміщати персонажа без використання фізики Rigidbody. Гравець рухається постійно вперед, а також може виконувати три ключові дії: перестрибування перешкод (свайп вгору), ковзання (свайп вниз) та перемикання смуги (свайпи вліво/вправо). Для зчитування цих свайпів використовується окремий компонент — SwipeInputController (лістинг 1.2), який фіксує жести й перетворює їх у логічні сигнали.

Кожна дія гравця синхронізується з відповідною анімацією через Animator. Наприклад, при стрибку активується тригер isJumping, а під час ковзання — isSliding. Оскільки ковзання впливає на габарити колайдера, PlayerController тимчасово змінює розміри CapsuleCollider, після чого повертає їх у початковий стан.

У грі реалізовано кілька типів об’єктів, із якими гравець може стикатися:

монети, зірка-бонус та щит. Коли гравець входить у тригер об’єкта, відповідна логіка активується в методі OnTriggerEnter(). Наприклад, активація щита оновлює стан GameStateModel через метод SetImmortal(true), а після закінчення таймера — повертає його назад у false.

У разі зіткнення з перешкодою викликається метод OnControllerColliderHit(). Якщо персонаж знаходиться під дією щита, перешкода знищується. Інакше гра завершується, музика зупиняється, і через компонент GameUIView (лістинг 1.3) активується панель поразки.

Модель ScoreModel (лістинг 1.4) зберігає поточний рахунок і кількість зібраних монет. Вона працює в парі з UI-компонентом ScoreView, який виводить дані на екран. Зміни в моделі не впливають безпосередньо на інтерфейс — їх застосовує PlayerController, передаючи нові значення у Text поля через методи UpdateScore() та UpdateCoins().

Цей підхід дозволяє легко адаптувати інтерфейс до нових вимог (наприклад, додати систему XP або рівнів) без втручання у внутрішню логіку моделей.

Для динамічного створення середовища використовується TileGenerator (лістинг 1.5). Цей скрипт створює нові ділянки дороги перед гравцем і знищує старі позаду. Всі платформи реалізовані як префаби, що містять перешкоди, бонуси та маркери спавну. Завдяки цьому забезпечується нескінченний маршрут, який постійно оновлюється відповідно до просування гравця.

Імовірність появи об’єктів на платформі регулюється через RandomSpawn (лістинг 1.6) — компонент, що випадковим чином вирішує, чи потрібно активувати конкретний бонус чи перешкоду під час генерації. Це надає грі варіативності й дозволяє уникнути повторюваності рівнів.

Компонент CameraView (лістинг 1.7) слідкує за гравцем із фіксованим відставанням по осі Z. Це створює ефект постійного руху вперед — ключовий елемент жанру runner.

Інтерфейс користувача (UI) у грі реалізовано у вигляді окремих компонентів, кожен із яких виконує чітко визначену функцію. Компонент

GameUIView відповідає за керування панеллю програшу, яка відображається після зіткнення персонажа з перешкодою без активного захисту. Вона забезпечує виведення фінального рахунку та доступ до кнопок перезапуску чи повернення до головного меню. Компонент MainMenuView відповідає за відображення стартового екрана гри, включаючи основне меню з кнопками початку гри, переходу до налаштувань і відображення статистики. Саме через нього здійснюється запуск основної сцени гри. Окремо функціонує ScoreView, який керує виведенням числових показників — кількості очок і зібраних монет — під час активної сесії. Усі ці компоненти побудовані таким чином, щоб не містити логіки гри, а лише реагувати на команди від контролерів, забезпечуючи чітке розділення між візуальним представленням і внутрішньою логікою.

Усі ці компоненти працюють лише на основі команд від контролерів, не ініціюючи жодної ігрової логіки самостійно. Це забезпечує чисту архітектуру з мінімальним зв’язуванням між частинами.

Для керування звуком у грі використовується окремий скрипт AudioManager, який реалізовано як синглтон. Він відповідає за відтворення фонової музики та звукових ефектів, а також за збереження рівнів гучності у PlayerPrefs. Завдяки цьому гравець має змогу один раз налаштувати звук у налаштуваннях, після чого ці параметри автоматично застосовуватимуться при наступних запусках гри. Компонент працює із двома основними типами джерел звуку: фоновою музикою та ефектами. Змінювати гучність можна через відповідні методи SetMusicVolume та SetSFXVolume, до яких під’єднано повзунки в налаштуваннях.

Меню паузи реалізоване через скрипти PauseScript та ResumeScript, які відповідають за тимчасову зупинку та відновлення гри. При активації паузи зупиняється ігровий час, вимикається основний UI, і з’являється меню з кнопками. При відновленні гри UI повертається у звичайний стан, а час гри поновлюється. Такий підхід забезпечує коректну зупинку гри без втрати стану, що особливо важливо для мобільних ігор. Також у цьому меню присутня

кнопка повернення до головного меню — при її натисканні музика змінюється, а сцена перезавантажується.

Налаштування звуку обробляються окремим компонентом SettingsScript, який керує двома повзунками у сцені налаштувань. При завантаженні меню він зчитує значення гучності з AudioManager, а при зміні значення — одразу оновлює відповідну гучність у грі та зберігає її для майбутніх сесій. Це дозволяє гравцеві зручно адаптувати гучність до власних уподобань без перезапуску гри.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

Після запуску гри гравець потрапляє до головного меню, де зібрані основні елементи навігації та інформації про попередні ігрові сесії. У верхній частині екрана відображається загальна кількість монет, накопичених під час усіх проходжень. У нижній частині екрану — найкращий результат, що мотивує покращити власні досягнення. Центральна частина екрану містить кнопку, яка переносить гравця до основної гри. Меню оформлено просто й лаконічно, що дозволяє миттєво зорієнтуватись, навіть якщо користувач вперше відкриває гру, також для покращення сприйняття у меню грає легка фонова музика.

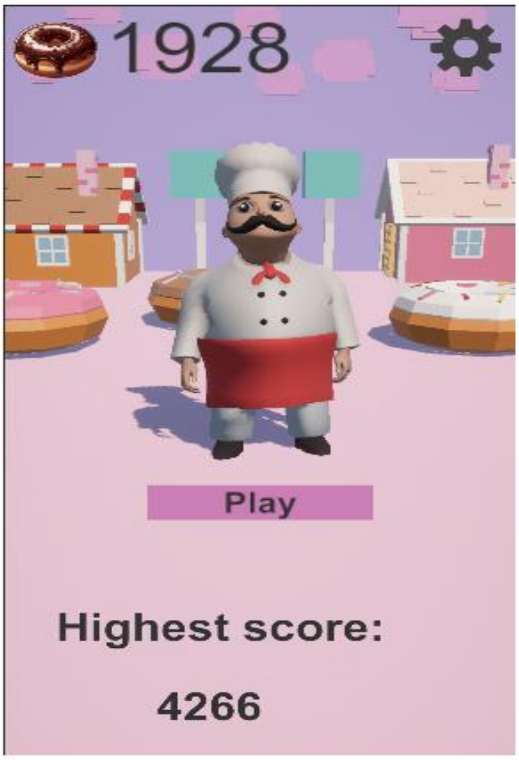

Рисунок 4.1 – Головне меню гри

Перейшовши до розділу налаштувань, гравець отримує можливість змінити гучність музики та звукових ефектів. Інтерфейс цього екрану

максимально спрощений і зосереджений лише на двох повзунках: один відповідає за фонову музику, інший — за ефекти, зокрема звуки бонусів, зіткнень або збирання монет. Налаштування зберігаються автоматично, тож гравцеві не потрібно натискати додаткові кнопки для підтвердження. У верхнбому правому куту розташована кнопка повернення, яка дозволяє вийти назад до головного меню. При цьому налаштування залишаються незмінними, а музичне оформлення плавно переходить до відповідного режиму. Завдяки цьому меню можна легко адаптувати гру до умов навколишнього середовища або індивідуальних уподобань гравця.

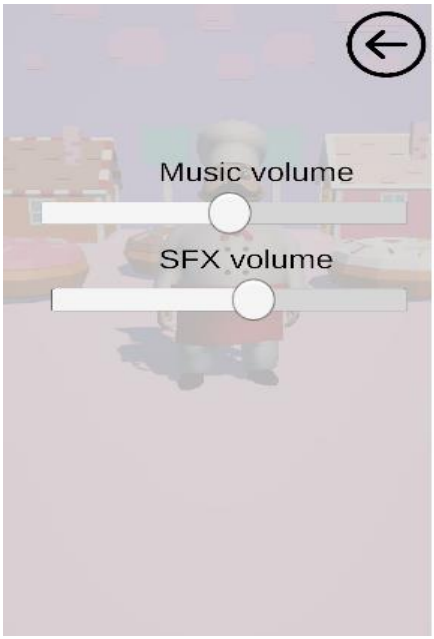

Рисунок 4.2 – Меню налаштування

Коли гравець натискає кнопку початку, він одразу потрапляє на ігрову арену. Персонаж автоматично починає рух уперед, не потребуючи жодного додаткового керування з боку користувача. Перша платформа, по якій починається рух, завжди виглядає однаково та не містить перешкод або бонусів. Це зроблено для того, щоб гравець міг адаптуватися до управління, не ризикуючи одразу програти. У цей момент можна спробувати свайпи вгору, вниз або в сторони, аби зрозуміти, як саме реагує персонаж. Камера слідкує за

героєм із фіксованої відстані, підтримуючи відчуття безперервного руху та поступового занурення в геймплей.

Рисунок 4.3 – Початок гри

Під час гри користувач часто стикається з перешкодами, які потребують швидкої реакції. Однією з базових дій персонажа є стрибок, що активується свайпом вгору. Стрибок дозволяє уникати об’єктів, які розташовані на рівні землі або трохи вище. Після свайпу персонаж підстрибує з характерною анімацією, а його колайдер автоматично адаптується до нового положення, щоб уникнути некоректного зіткнення. Під час стрибка гравець може переглянути навколишнє середовище та заздалегідь підготуватися до наступної дії. Рух виконується плавно, без ривків, що дозволяє насолоджуватися керованістю героя навіть у повітрі. Механіка стрибка розрахована на точний розрахунок моменту — надто раннє або запізніле натискання може призвести до втрати шансу пройти перешкоду.

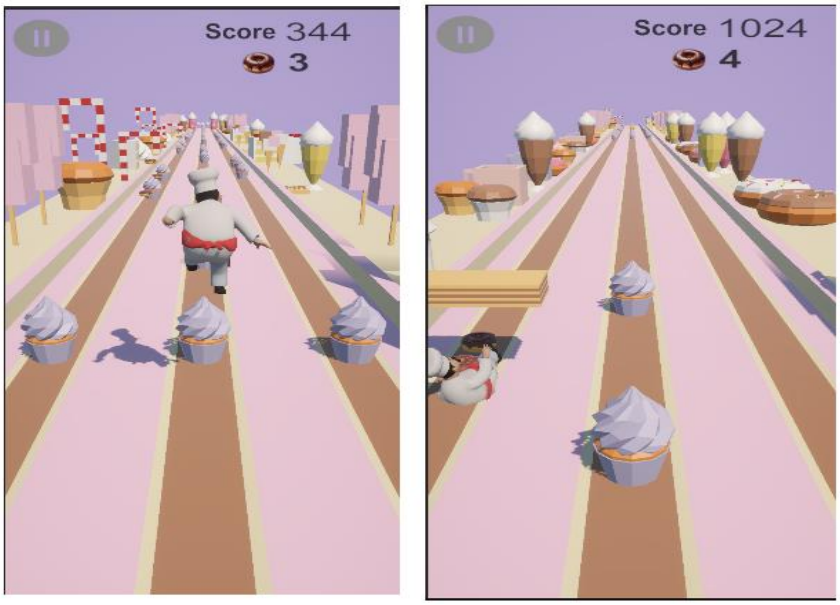

Рисунок 4.4 – Персонаж у стану стрибка та перекату

Ще однією важливою дією, яка дозволяє залишитися в грі довше, є перекат. Щоб активувати цю дію, слід свайпнути вниз. Персонаж миттєво переходить у положення ковзання, зменшуючи свою висоту та розміри, що дозволяє йому прослизати під низько розташованими об’єктами, наприклад, трубами або балками. Анімація ковзання чітко передає зміну руху, а сам процес триває близько однієї секунди. Після завершення перекату персонаж повертається у звичне положення. Цікаво, що перекат може бути використаний не лише як реакція на перешкоду — він також дає змогу перервати стрибок у повітрі. Якщо гравець під час стрибка виконує свайп вниз, герой достроково припиняє політ і плавно опускається на землю у положенні ковзання. Це відкриває додаткову тактичну можливість для досвідчених гравців і дозволяє точніше контролювати поведінку персонажа в динамічному середовищі.

Під час гри користувач у будь-який момент може призупинити гру, натиснувши кнопку паузи. Після цього з’являється спеціальне меню, яке тимчасово зупиняє весь ігровий процес. У цьому режимі персонаж зупиняється, і всі анімації та фізичні процеси заморожуються. На екрані з’являються дві кнопки: одна дозволяє продовжити гру з того моменту, де вона

була зупинена, інша повертає до головного меню. Усе зроблено таким чином, щоб не втратити прогрес і водночас мати можливість зробити паузу, якщо виникла потреба змінити умови гри, відволіктись або просто перепочити.

Рисунок 4.5 – Гра під час паузи

У разі зіткнення з перешкодою, якщо щит не активований, гра завершується, і гравець потрапляє на екран поразки. Поля із накопиченими монетами та очками за ігрову сесію зберігаются на екрані, тому їх можно спокійно переглянути. Якщо результат виявляється кращим за попередній рекорд, він автоматично зберігається. Користувачу пропонується два варіанти: почати гру заново або повернутись до головного меню. Кнопка «Restart» запускає нову сесію з тієї ж стартової сцени. Важливо, що при перезапуску оновлюється навколишнє середовище, тому кожна спроба залишається унікальною. Цей екран дозволяє швидко повернутися до гри, не втрачаючи темпу та мотивації.

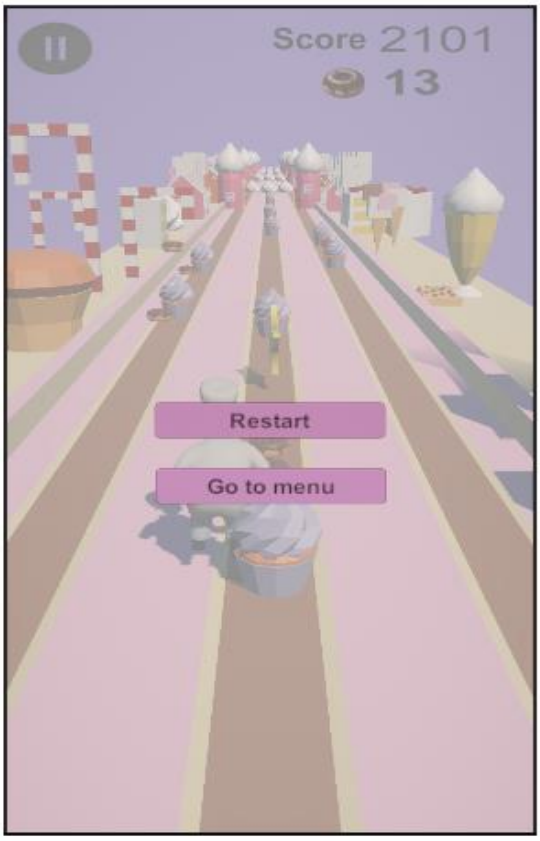

Рисунок 4.6 – Екран поразки

Під час проходження гравець може зустріти обертаючу зірку — це бонус, який активує подвоєння очок. Як тільки персонаж торкається цього об’єкта, лунає спеціальний звук, а всі подальші очки, що нараховуються за пройдену відстань або інші дії, множаться на два. Ефект триває протягом п’яти секунд, після чого рахунок повертається до звичайного режиму. Цей бонус дозволяє максимально швидко підвищити свій результат, особливо якщо активується на високих швидкостях. Завдяки цьому гравець отримує додатковий стимул ризикувати, намагаючись зібрати зірку навіть у складних ситуаціях.

Іншим корисним бонусом є щит, який візуально відрізняється характерною іконкою. Після його активації персонаж на певний час стає невразливим. Це означає, що при першому зіткненні з перешкодою вона буде знищена, а гравець продовжить рух без наслідків. Дія щита триває до 30 секунд або до моменту першого зіткнення. Після використання захист зникає, і гра

переходить у звичайний режим. Активація щита супроводжується окремим звуковим сигналом, що допомагає гравцеві зрозуміти, що захист активований. Бонус дозволяє не лише пом’якшити наслідки помилки, а й ризикнути, наприклад, для збору додаткових очок або монет у складних ділянках маршруту.

Рисунок 4.7 – вигляд бонусів щит та подвоєння очок

Уся гра побудована так, щоб бути доступною навіть для новачків. Просте управління, інтуїтивно зрозумілі жести та поступове ускладнення дозволяють швидко зануритись у геймплей без потреби в тривалому навчанні. Завдяки мінімалістичному інтерфейсу, плавній анімації та чіткій реакції на дії гравця, гра підходить як для коротких сесій у перервах, так і для тривалого намагання встановити новий рекорд. Бонуси додають динаміки та можливостей для стратегічного планування, а нескінченне середовище з випадковими перешкодами робить кожен забіг унікальним. Загалом, гра не є надмірно складною й дозволяє отримувати задоволення без стресу, поступово розвиваючи навички швидкої реакції та орієнтації в русі.

ВИСНОВКИ

У межах дипломної роботи було розроблено прототип гри в жанрі runner, який відповідає вимогам до сучасних casual-проектів: швидкий вхід у геймплей, поступове ускладнення, просте управління та адаптивна побудова середовища. Проект поєднує технічну реалізацію базової архітектури, візуального оформлення, логіки взаємодії та системи оцінювання прогресу гравця.

Під час розробки було впроваджено архітектурну модель MVC, яка забезпечила розділення відповідальностей між логікою, інтерфейсом і даними. Такий підхід дозволив підтримувати чисту структуру коду, мінімізувати залежності між компонентами та зробити проект гнучким до майбутніх змін. Створено повноцінну систему управління персонажем, що включає рух, стрибки, ковзання, анімації та логіку зіткнень із перешкодами. Усі дії гравця синхронізовані з візуальними ефектами, а колайдер персонажа автоматично адаптується до відповідного стану, що дозволяє уникнути помилок у взаємодії з об’єктами.

Середовище гри реалізовано як нескінченну динамічну генерацію плиток із випадковими перешкодами та бонусами, що дозволяє уникнути повторюваності рівня й підтримувати інтерес гравця протягом багатьох сесій. Бонуси, такі як щит або зірка, вносять різноманітність у геймплей, стимулюють ризиковану гру та дозволяють досягати вищих результатів. Водночас, зростаюча швидкість поступово ускладнює проходження, створюючи відчуття прогресу та виклику.

Окрему увагу приділено реалізації інтерфейсу користувача: лічильники очок і монет, головне меню, панель поразки, налаштування гучності, меню паузи — усі ці компоненти є невід’ємною частиною взаємодії з гравцем. Вони були реалізовані як окремі View-компоненти відповідно до архітектури, з чітким розмежуванням логіки та візуального представлення. Також були

розроблені допоміжні скрипти, зокрема для обертання бонусів, випадкового відтворення об’єктів, керування камерою, фоновим звуком та збереження прогресу. Усі ці частини системи об’єднано в єдину інтерактивну екосистему, що працює стабільно завдяки оптимізованому коду.

У процесі тестування виявлено й виправлено низку критичних помилок, зокрема баг із необмеженим стрибком, десинхронізацію між анімаціями й колайдером, некоректне повторне натискання кнопок після поразки, а також проблеми з масштабуванням інтерфейсу на різних роздільностях екрану. Завдяки цьому вдалося досягти стабільної роботи гри та забезпечити позитивний користувацький досвід.

Загалом, поставлені цілі дипломного проекту були досягнуті. Створено функціональний та масштабований каркас гри, який може бути розширено шляхом додавання нових механік, рівнів, анімацій, системи досягнень, збереження прогресу чи навіть монетизації. Отриманий прототип відповідає сучасним стандартам якості для мобільних ігор, демонструє практичне застосування знань із програмування, побудови архітектурних рішень, реалізації UI-компонентів і оптимізації під платформу Unity. Робота дала змогу закріпити теоретичні знання у реальному програмному середовищі та здобути практичний досвід створення повноцінного ігрового застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Goldstone W. Unity 3.x Game Development by Example: Beginner's Guide. Packt Publishing, 2011. — 488 p.

2. Hocking J. Unity in Action: Multiplatform Game Development in C#. Manning, 2015. — 300 p.

3. Lucero S. Mastering Game Dev with Unity: Build High-Performance Games. Independently published, 2023. — 385 p.

4. Murray J.W. C# Game Programming Cookbook for Unity 3D. CRC Press, 2014. — 504 p.

5. Sapio F. Unity UI Cookbook. Packt Publishing, 2015. — 354 p.

6. Boehm A., Murach J. Murach’s C# (8th edition). Mike Murach & Associates, 2023. — 882 p.

7. Troelsen A., Japikse P. Pro C# 10 with .NET 6. Apress, 2022. — 1342 p.

8. Stellman A., Greene J. Head First C#. O’Reilly Media, 2020. — 816 p.

9. Joshi A. Data Structures and Algorithms in C#. Packt Publishing, 2023. — 582 p.

10. Soto S. Unity Proficiency: From Beginner to Pro – A Step-by-Step Guide. Independently published, 2019. — 138 p.

11. Freeman J. C# Cookbook: Modern Recipes for Professional Developers. O’Reilly Media, 2021. — 712 p.

12. Grey B. Master the Game Coding for Programmers. Independently published, 2023. — 290 p.

13. Skeet J. C# in Depth. Manning, 2019. — 528 p.

14. Gibson J. Introduction to Game Design, Prototyping, and Development. Addison-Wesley, 2014. — 944 p.

15. Love S. C# and the .NET Type System. No Starch Press, 2023. — 424 p.

16. Freeman E., Robson E. Head First Design Patterns. – O’Reilly Media, 2021. – 694 p.

17. Rollings A., Morris D. Game Architecture and Design: Learn the Best Practices for Game Design and Programming. – New Riders Publishing, 2004. – 688 p.

18. McShaffry M., Graham S. Game Coding Complete, 4th Edition. – CRC Press, 2012. – 960 p.

19. Petri J., Huttunen M. Game Programming with Unity and C#: A Beginner's Guide – Apress, 2021. – 280 p.

20. Nystrom R. Game Programming Patterns – Genever Benning, 2014. – 354 p.