МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Навчально-науковий інститут новітніх освітніх технологій
Кафедра комп’ютерної інженерії

Ілюшин Віталій Володимирович

Програмний модуль для комп'ютерної гри на основі технології Unity / Software module for the computer game based on Unity technology

спеціальність: 123 – Комп’ютерна інженерія
освітньо-професійна програма – Комп’ютерна інженерія

Випускна кваліфікаційна робота

Виконав: студент групи КІз-41
Ілюшин Віталій Володимирович

Науковий керівник
Н. Я. Савка

2025

АНОТАЦІЯ

Ілюшин В. Програмний модуль для комп'ютерної гри на основі технології Unity – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 «Комп’ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2025.

Робота написана обсягом 40 сторінок і містить 11 рисунків, 2 таблиці, 3 додатки та 24 джерел за переліком посилань. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою цієї дипломної роботи є розробка програмного модулю для комп'ютерної гри на основі технології Unity.

Актуальність цього проєкту полягає у зростаючій важливості модульної архітектури ігор, яка підтримує швидку розробку та гнучкість. Unity, як одна з найпопулярніших платформ для розробки ігор, дозволяє інтегрувати користувацькі програмні модулі для розширення функціональності, оптимізації продуктивності та покращення ігрового процесу. Розробка програмного модуля, який відповідає найкращим практикам у програмній інженерії та ігровому дизайні, не лише відповідає поточним потребам ринку, але й сприяє академічному дослідженню компонентної розробки програмного забезпечення в ігрових контекстах. Для розв’язання поставлених задач у кваліфікаційній роботі використано сучасні підходи до розробки програм на Unity .

Ключові слова: КОМП’ЮТЕРНА ГРА, UNITY.

ANNOTATION

Ilyushin V. Software module for a computer game based on Unity technology – Manuscript.

Qualification work for the degree of Bachelor in specialty 123 "Computer Engineering", educational and professional program. Western Ukrainian National University, Ternopil, 2025.

The work is written in 40 pages and contains 19 figures, 2 tables, 3 appendices and 24 sources according to the list of references. The volume of graphic material is 2 sheets of A3 format.

The purpose of this thesis is to develop a software module for a computer game based on Unity technology.

The relevance of this project lies in the growing importance of modular game architecture, which supports rapid development and flexibility. Unity, as one of the most popular game development platforms, allows you to integrate custom software modules to extend functionality, optimize performance, and improve the gameplay. Developing a software module that meets the best practices in software engineering and game design not only meets current market needs, but also contributes to academic research on component-based software development in game contexts. To solve the tasks set in the qualification work, modern approaches to developing programs on Unity were used.

Keywords: COMPUTER GAME, UNITY.

ЗМІСТ

Вступ 8
1. Аналіз засобів розробки програм засобами Unity 10
1.1 Аналіз наявного програмно-технічного забезпечення 10
1.2Аналіз середовищ розробки 14
1.3.....Функціональні та нефункціональні вимоги 17
2. Алгоритми роботи модулів на основі Unity 20
2.1 Архітектурна та функціональна структура додатку 20
2.2..... Алгоритм роботи програмного додатку 22
2.3.....Діаграма прецедентів 25
3. Програмна реалізація ігрового модулю 29
3.1 Структура програмного модулю 29
3.2 Тестування роботи модулю 31
3.3 Інтерфейс розробки програмного модулю 34
Висновки 39
Список використаних джерел 40
Додаток А 43
Додаток Б 46

ВСТУП

В останні роки індустрія комп'ютерних ігор стала одним із секторів, що найшвидше розвиваються, у сфері інформаційних технологій, стимулюючи інновації в графічному рендерингу, штучному інтелекті та інтерактивному дизайні. Широка доступність потужних платформ розробки, таких як движок Unity, значно знизила поріг входу для розробників, дозволяючи створювати складні та візуально насичені ігри на різних платформах. Ця еволюція призвела до зростання попиту на модульні, масштабовані та зручні в обслуговуванні програмні рішення в середовищах розробки ігор.

Актуальність цього проєкту полягає у зростаючій важливості модульної архітектури ігор, яка підтримує швидку розробку та гнучкість. Unity, як одна з найпопулярніших платформ для розробки ігор, дозволяє інтегрувати користувацькі програмні модулі для розширення функціональності, оптимізації продуктивності та покращення ігрового процесу. Розробка програмного модуля, який відповідає найкращим практикам у програмній інженерії та ігровому дизайні, не лише відповідає поточним потребам ринку, але й сприяє академічному дослідженню компонентної розробки програмного забезпечення в ігрових контекстах.

Незважаючи на переваги, що пропонує Unity, розробники часто стикаються з труднощами під час створення модулів, що можна повторно використовувати та адаптувати, і які можна легко інтегрувати в різні ігрові проекти. Такі проблеми, як погана можливість повторного використання коду, відсутність модульної структури та обмежена масштабованість, можуть суттєво знижувати ефективність розробки та якість продукту. Існує потреба в добре структурованому програмному модулі, який дотримується принципів проектування, придатних для динамічних ігрових середовищ, і підтримує легку інтеграцію в проекти на базі Unity.

Мета та завдання дослідження. Метою цієї дипломної роботи є розробка програмного модулю для комп'ютерної гри на основі технології Unity.

Для досягнення мети роботи були поставлені такі завдання:

- провести аналіз сучасного стану розробки ігрових модулів, виділивши основні платформи а мови програмування;
- розробити схему структурну схему проекту;
- розробити алгоритм роботи ігрового модулю з елементами ініціалізації ігрового модулю та керування персонажем.
- провести порівняльний аналіз отриманих результатів.

Предметом цього дослідження є процес розробки та архітектура програмного модуля, призначеного для інтеграції з комп'ютерними іграми на базі Unity.

Об'єктом дослідження є програмний модуль, включаючи його структуру, функціональність та взаємодію з ігровим рушієм Unity.

Практичне значення полягає у програмній реалізації ігрового модуля на Unity.

Для розв’язання поставлених задач у кваліфікаційній роботі використано сучасні підходи до розробки програм на Unity .

У першому розділі було проаналізовано сучасні підходи до розробки ігрових модулів, виділено базові компоненти та програмні фреймворки для розробки ігор.

У другому розділі було розроблено алгоритм роботи ігрового модулю з елементами ініціалізації ігрового модулю та керування персонажем та структурну схему проекту.

У третьому розділі розроблено прикладне програмне забезпечення та представлено порівняльний аналіз.

1. АНАЛІЗ ЗАСОБІВ РОЗРОБКИ ПРОГРАМ ЗАСОБАМИ UNITY

1.1 Аналіз наявного програмно-технічного забезпечення

Unity — це кросплатформний ігровий рушій, розроблений Unity Technologies, який широко використовується для створення як двовимірних (2D), так і тривимірних (3D) ігор та симуляцій. Він надає комплексне інтегроване середовище розробки (IDE), потужний рушій рендерингу, конвеєр ресурсів та підтримку сценаріїв, переважно через мову програмування C#.

Популярність Unity зумовлена простотою використання, гнучкістю, великим сховищем ресурсів та підтримкою розгортання на різних платформах, включаючи Windows, macOS, Android, iOS, WebGL, PlayStation, Xbox та інші. Це робить його найкращим вибором як для незалежних розробників, так і для великих студій.

Важливу роль у процесі розробки відіграє середовище Unity Editor, у якому здійснюється побудова сцени, налаштування об'єктів, створення анімацій, а також тестування логіки гри в реальному часі. Написання скриптів здійснюється мовою C# у зовнішніх середовищах розробки, таких як Visual Studio або Rider, які забезпечують зручні засоби налагодження, підсвітку синтаксису та інтелектуальні підказки.

Графічні та тривимірні ресурси створюються за допомогою спеціалізованого програмного забезпечення, зокрема Blender, Autodesk Maya або 3ds Max. Ці інструменти дозволяють розробляти моделі персонажів, об'єкти середовища та анімації, які згодом імпортуються до Unity. Для створення текстур і двовимірної графіки використовуються програми на кшталт Adobe Photoshop, GIMP або Substance Painter, а для реалізації аудіоефектів часто застосовуються системи FMOD чи Wwise, які легко інтегруються з Unity та забезпечують високий рівень контролю над звуковим супроводом.

З огляду на командну роботу над ігровими проєктами, актуальним є використання систем контролю версій, таких як Git або Plastic SCM, які дозволяють кільком розробникам одночасно працювати над різними аспектами гри без конфліктів.

Також важливим компонентом екосистеми є Unity Asset Store — офіційний магазин активів, де розробники можуть придбати або безкоштовно завантажити готові моделі, скрипти, шаблони, ефекти та інші корисні елементи, що значно прискорює розробку.

Окрім програмного забезпечення, розробка ігор з використанням Unity вимагає відповідного апаратного забезпечення. Потужна робоча станція з сучасним багатоядерним процесором, достатнім обсягом оперативної пам’яті (рекомендовано не менше 16–32 ГБ) та відеокартою високого рівня (наприклад, NVIDIA RTX серії) є необхідною умовою для ефективної роботи з великими проєктами та тривимірною графікою. Для тестування проєктів розробники використовують як настільні комп’ютери, так і мобільні пристрої на базі iOS та Android. У разі розробки для платформ віртуальної чи доповненої реальності застосовуються спеціалізовані пристрої, такі як Oculus Rift, HTC Vive або HoloLens.

Рисунок 1.1 –Головне меню гри “ *Shatterline”*

Battlefield 4 – це не дуже сучасна гра але вона не відстає від сучасних ігор, також вона може похвалитись своїм сюжетом та активним геймплейом.

Також гріх не згадати можливість грати в мультиплейрі в якому відкривається 70% всієї гри. Також має зможу редагування свого персонажу та бронетехніки.

Рисунок 1.2 – Головне меню гри “Battlefield 4”

Call of Duty Modern Warfare 2 на свої часи була з дуже реалістичною графікою, але з дуже кастомною стрільбою її навіть порівнювали з такою грою як Battlefield 4, яка була набагато реалістичнішою ніж Call of Duty MW2.

До плюсів можна віднести:

- дуже цікавий сюжет;
- реалістична графіка.

До мінусів можна віднести:

- кастомна стрільба;
- не підтримується до сьогоднішнього моменту.

Рисунок 1.3 – Головне меню “Call of Duty Modern Warfare ”

Tom Clancy's The Division 2 – Гра приваблює ігровою механікою та місцевою «пісочницею», що нагадує Diablo-шутер у відкритому світі. Стрілянина реалізована за принципом рольових ігор, коли ви поступово зменшуєте смужку життя ворога. У деяких ворогів потрібно всадити кілька куль, а важкі броньовані противники можуть прийняти кілька автоматних магазинів. І залежить все це ще від рівня ворогів та характеристик вашої зброї.

Саму гру не можна назвати реалістичним шутером, але вона надихає своєю графікою, сюжетом, персонажами та багато іншим. Основними недоліками гри є сервера та баги.

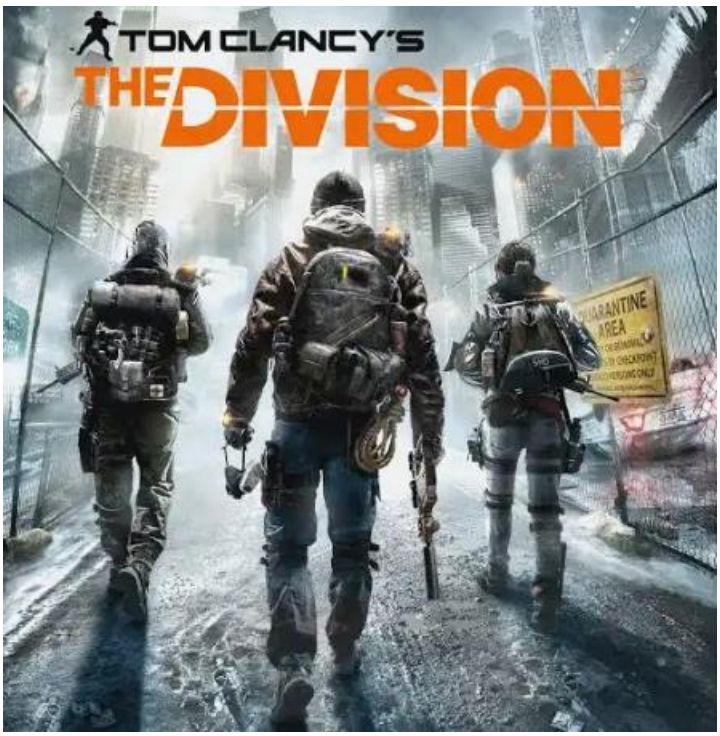

Рисунок 1.4 – Інтерфейс гри *“Tom Clancy's The Division 2”*

Згідно проведеному аналізу подібних програмних продуктів, програмне забезпечення повинно:

- підтримувати українську мову;
- споживати мінімальну кількість ресурсів комп’ютера;
- бути простим у користуванні;
- повинно мати нормальну оптимізацію.

1.2 Аналіз середовищ розробки

Середовище розробки є ключовим елементом процесу створення комп’ютерних ігор, оскільки воно визначає доступні інструменти, мову програмування, можливості тестування та кінцеву продуктивність проєкту.

Сучасний ринок ігрової індустрії пропонує розробникам широкий вибір середовищ, кожне з яких має свої переваги, недоліки та сферу застосування.

Одним із найбільш популярних середовищ для розробки ігор є Unity. Цей рушій вирізняється зручним інтерфейсом, широкими функціональними можливостями та підтримкою великої кількості платформ: від мобільних пристроїв до настільних комп'ютерів, веббраузерів, ігрових консолей і систем віртуальної реальності. Unity забезпечує гнучке середовище розробки з підтримкою компонентно-орієнтованої архітектури, що дозволяє створювати масштабовані та модульні ігрові проєкти. Основною мовою програмування в Unity є C#, що робить розробку зрозумілою для широкого кола програмістів. Unity також має великий активний ком'юніті, добре задокументований API та доступ до магазину активів (Asset Store), що дозволяє значно прискорити розробку.

Іншим відомим ігровим рушієм є Unreal Engine, розроблений компанією Epic Games. На відміну від Unity, Unreal Engine орієнтований передусім на розробку ігор із високою графічною якістю, завдяки вбудованому рушію рендерингу на базі технологій, таких як Lumen і Nanite. Unreal використовує мову C++ та візуальну систему скриптування Blueprint, що дозволяє навіть недосвідченим розробникам створювати складні логічні взаємодії без необхідності писати код. Проте Unreal Engine потребує значно потужнішого обладнання, і його використання зазвичай передбачає більш круту криву навчання.

Ще одним прикладом є Godot Engine — відкритий і безкоштовний рушій, який останніми роками активно розвивається. Godot має власну мову програмування GDScript, схожу на Python, а також підтримку C# і C++. Незважаючи на те, що Godot поки поступається Unity та Unreal у сфері 3D-графіки, він є дуже ефективним інструментом для створення 2D-ігор завдяки легкій вазі, простоті інтерфейсу та можливості швидкого прототипування.

Для розробки мобільних ігор також часто використовують такі середовища, як Cocos2d-x, Defold, або GameMaker Studio, які орієнтовані

переважно на 2D-графіку та легку вагу фінального продукту. Ці рушії популярні серед інді-розробників завдяки простому інтерфейсу, можливості створення кросплатформенних рішень і мінімальним системним вимогам.

Порівнюючи ці середовища, можна зробити висновок, що вибір рушія значною мірою залежить від типу гри, рівня досвіду команди розробників, вимог до продуктивності та графіки, а також доступного бюджету. Unity на сьогодні є універсальним рішенням, яке поєднує в собі простоту, потужність і гнучкість, що робить його одним із найкращих варіантів для створення як 2D, так і 3D-ігор. Порівняння ігрових рушії наведено у таблиці 1.1.

Таблиця 1.1 - Таблиця порівняння ігрових рушіїв

Параметр	Unity	Unreal Engine	Godot	GameMaker Studio
Мова програмування	C#	C++, Blueprint	GDScript, C#, C++	GML (власна мова)
Підтримка 2D	Висока	Середня	Висока	Висока
Підтримка 3D	Висока	Дуже висока	Середня	Обмежена
Простота у вивченні	Висока	Середня	Висока	Висока
Графічна якість	Висока	Дуже висока (AAA рівень)	Середня	Середня
Масштабованість проєктів	Висока	Дуже висока	Середня	Низька-середня

На основі проведеного аналізу можна стверджувати, що вибір середовища розробки має вирішальне значення для успішної реалізації ігрового проєкту. Серед усіх розглянутих варіантів Unity посідає провідне місце завдяки своїй універсальності, високій гнучкості, підтримці широкого

спектру платформ, потужному редактору та великій кількості готових рішень у вигляді активів і плагінів.

Unreal Engine забезпечує найвищу якість графіки, що робить його оптимальним для великих 3D-проєктів із високими вимогами до візуалу. Водночас, він потребує більшого рівня технічної підготовки та ресурсів.
Godot є чудовим вибором для інді-розробників та навчальних цілей завдяки відкритому коду, легкості у використанні та підтримці 2D-графіки. GameMaker Studio залишається хорошим варіантом для швидкого створення простих 2D-ігор, але обмежений у масштабуванні проєктів.

З огляду на мету дипломного проєкту — створення програмного модуля для комп'ютерної гри — саме Unity є найдоцільнішим вибором. Його компонентно-орієнтована архітектура, підтримка C#, багата інфраструктура інструментів і активна спільнота створюють сприятливі умови для реалізації гнучкого, повторно використовуваного та масштабованого програмного рішення.

1.3 Функціональні та нефункціональні вимоги

Процес розробки програмного модуля повинен базуватись на чітко визначених вимогах, які гарантують відповідність розробки поставленим цілям, зручність використання та технічну надійність. Умовно ці вимоги можна поділити на функціональні, які описують конкретну поведінку модуля, та нефункціональні, що визначають якість реалізації цієї поведінки.

У рамках створення ігрового проєкту важливо чітко окреслити вимоги до одного з ключових компонентів — модуля керування персонажем. Цей модуль відповідає за взаємодію між гравцем і персонажем у грі та забезпечує плавну, передбачувану і чітку реакцію на дії користувача. Його розробка вимагає врахування як функціональних, так і нефункціональних вимог.

Функціонально модуль повинен забезпечувати базові механіки пересування персонажа по ігровому середовищу. Це передбачає рух у різних напрямках з урахуванням фізики (якщо вона застосовується в грі), стрибки, присідання або біг — залежно від обраного жанру та стилю гри. Поведінка персонажа повинна змінюватися відповідно до команд, які гравець подає за допомогою клавіатури, миші, геймпада чи сенсорного інтерфейсу. Окрім основного пересування, модуль має підтримувати взаємодію з оточенням — наприклад, виявлення перешкод, обробку зіткнень, входження в тригери, а також виконання анімацій, що відповідають поточному стану персонажа (ходьба, біг, стрибок, бездіяльність тощо). Важливо, щоб розробник міг налаштовувати основні параметри руху (швидкість, сила стрибка, гравітація тощо) безпосередньо в редакторі Unity, через інспектор, без необхідності змінювати код вручну.

Крім основної поведінки, функціональні вимоги передбачають, що модуль має бути сумісним з іншими елементами гри, зокрема — камерою, UI, об’єктами сцени, а також передбачати генерацію внутрішніх подій, які можна використовувати для подальшої логіки гри. Наприклад, модуль може надсилати сигнал про зміну стану (наприклад, “персонаж приземлився” або “зачепив перешкоду”), що може бути корисним для реалізації звукових ефектів або оновлення інтерфейсу.

Нефункціональні вимоги стосуються якості реалізації цього модуля. Важливо, щоб код був побудований за принципами модульності, що дозволить у майбутньому легко змінювати або розширювати функціональність, наприклад, додати нові типи руху чи змінити спосіб взаємодії з фізичним середовищем. Продуктивність модуля має бути достатньою для забезпечення стабільної роботи гри навіть на пристроях із обмеженими ресурсами. Для цього необхідно уникати зайвих обчислень у циклі Update() та раціонально використовувати ресурси Unity.

Код повинен бути зрозумілим, структурованим та супроводжуватися коментарями — особливо в місцях, де реалізовано логіку руху, перевірку зіткнень або станів. Це полегшить підтримку проєкту та дозволить іншим розробникам або самому автору легко орієнтуватися у структурі через певний час. Також модуль має бути стійким до помилок — наприклад, він не повинен викликати винятки у випадку відсутності потрібних компонентів, некоректних налаштувань або нестандартних ситуацій на сцені.

Окрему увагу слід приділити можливості тестування: модуль має підтримувати запуск у тестовому середовищі — наприклад, на окремій сцені з базовими об’єктами, де можна перевірити коректність руху, відповідність анімацій та взаємодію з іншими елементами гри. Нарешті, масштабованість — важлива риса якісного модуля керування. Його структура має передбачати можливість інтеграції додаткових функцій, таких як автоматичне перемикання між режимами руху, керування транспортом, використання здібностей тощо.

Таким чином, модуль керування персонажем повинен бути не лише функціонально повним, а й якісно реалізованим з погляду архітектури, продуктивності, зручності супроводу та гнучкості в контексті подальшого розвитку ігрового проєкту.

2. АЛГОРИТМИ РОБОТИ МОДУЛІВ НА ОСНОВІ UNITY

2.1 Архітектурна та функціональна структура додатку

Програмне доповнення розроблено як модульний компонент, який бездоганно інтегрується в ігровий проект на базі Unity. Воно інкапсулює певні ігрові функції (наприклад, керування персонажем, систему інвентарю, механізм діалогів) та дотримується принципів компонентного проектування, розділення завдань та повторного використання.

Архітектура відповідає стандартній моделі сценаріїв MonoBehaviour від Unity у поєднанні з додатковими ScriptableObjects та Events для підвищення гнучкості та розділення.

На основі функціональних та нефункціональних вимог було розроблено структурну схему, на рисунку 2.1 відображено структурна схема.

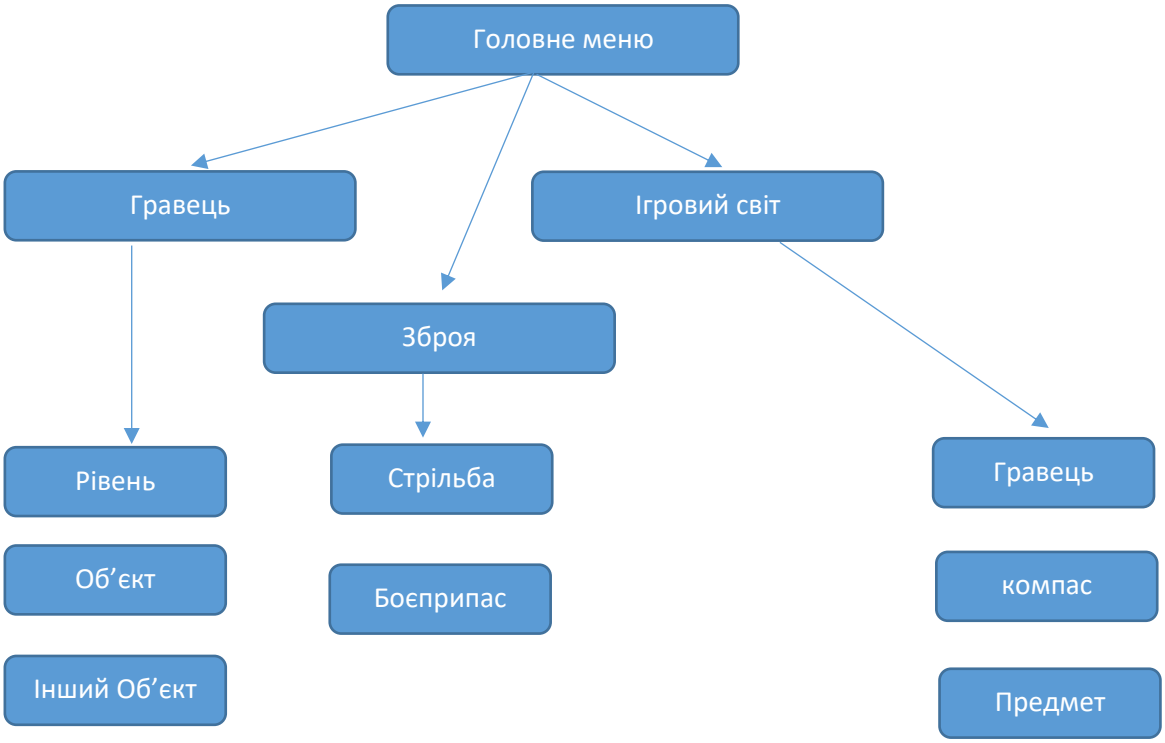

Рисунок 2.1 – Структурна схема проекту

Дана архітектура створена для підтримки:

- Інтеграції з багатьма гравцями: з додаванням хуків NetworkManager до контролера
- Керування за допомогою штучного інтелекту: шляхом заміни InputManager на AIInputController
- Налаштування: розробники можуть самостійно замінювати компоненти (наприклад, нову систему анімації, альтернативну фізику).

Запропонована архітектура програмного модуля Unity розроблена для забезпечення гнучкості, зручності обслуговування та продуктивності. Вона відповідає найкращим практикам Unity, використовуючи компонентну природу движка та підтримуючи масштабоване розширення. Розробники та дизайнери можуть працювати разом завдяки використанню ScriptableObjects та чіткому розділенню між логікою та конфігурацією.

Головний контролер модуля (наприклад, CharacterController.cs) виступає ядром модуля. Обробляє логіку, таку як рух, керування станом (простій, ходьба, стрибки) та координація між іншими компонентами (наприклад, анімація, фізика).

InputManager.cs захоплює вхідні дані гравця (клавіатура, контролер або дотик) та перетворює їх на дії. Використовує систему введення Unity (або застарілий Input.GetKey(), якщо простіше) та передає інформацію контролеру.

AnimationHandler.cs керує переходами та тригерами анімації на основі стану персонажа. Зв'язується з Unity Animator та встановлює параметри для переходів.

CollisionDetector.cs обробляє перевірки, пов'язані з фізикою, такі як виявлення землі, виявлення перешкод та тригери взаємодії. Може використовувати raycast або події колайдерів.

UIController.cs надає зворотний зв'язок користувачеві на основі стану модуля. Наприклад, відображає індикатори швидкості, здоров'я або дій, пов'язані з персонажем.

GameManager.cs надає глобальний контекст, такий як стан гри, прогрес рівня або зовнішні дані, з якими модуль може взаємодіяти.

Для покращення модульності та розділення даних конфігурації, ScriptableObjects можна використовувати для зберігання налаштувань руху персонажів (швидкість, висота стрибка), вхідної конфігурації, пресетів анімації, визначень кінцевих автоматів.

Це дозволяє дизайнерам налаштовувати параметри ігрового процесу без зміни коду.

Використання подій Unity або власної системи подій (шаблон спостерігача) допомагає роз'єднати компоненти. Наприклад: Коли персонаж стрибає, спрацьовує подія: OnJumpPerformed . Інші системи (наприклад, звук або камера) можуть підписатися на це без тісного зв'язку.

2.2 Алгоритм роботи програмного додатку

Після запуску гри Unity завантажує початкову сцену. У ній відбувається створення або активація елементів головного меню: логотип, кнопки ("Почати гру", "Налаштування", "Вихід"), можливо — фонова музика чи анімації. Відбувається підключення обробників до кнопок UI. Стан гри встановлюється як MainMenu. Весь ігровий процес очікує дії гравця.

На етапі «Очікування» система перебуває у стані очікування вибору гравця. Движок перевіряє, чи натиснута кнопка "Почати гру". Поки гравець не взаємодіє, система залишається в цьому стані. У фоні може оброблятися анімація або музика, але гравець ще не взаємодіє з геймплеєм.

Після вибору "Почати гру" меню приховується, завантажується нова сцена або активуються елементи ігрового світу. Створюються або розміщуються об'єкти: персонаж, платформи, вороги, декорації. Встановлюється початкова позиція гравця. Камера, UI-елементи (життя, рахунок) ініціалізуються.

Стан гри змінюється на GameRunning. Починається обробка геймплею. Включається обробка введення з клавіатури (або іншого пристрою). Система запускає таймер рівня, музику, вмикає візуальні ефекти.

Unity щокадрово викликає метод Update(), у якому перевіряється введення з клавіатури. Якщо натиснута клавіша вліво (Input.GetKey(KeyCode.LeftArrow) або A), змінюється позиція гравця вліво. Якщо натиснута клавіша вправо (Input.GetKey(KeyCode.RightArrow) або D), гравець рухається вправо. При цьому враховується:

- швидкість руху;
- напрямок анімації;
- перевірка зіткнень (наприклад, чи є під ногами платформа).

У відповідь на натискання клавіш:

- До позиції гравця додається відповідне зміщення.
- Вмикається відповідна анімація (Animator.SetBool("IsRunning", true)).
- Якщо клавіші не натиснуті — гравець зупиняється, грається анімація стояння.
- Якщо гравець стикається з об'єктами (стіна, ворог, межа карти) — рух блокується або змінюється відповідно до логіки гри.

Приклад псевдокоду блоку:

Початок Гри

Завантажити сцену "MainMenu"

Показати головне меню

Стан_гри ← "MainMenu"

Поки Стан_гри == "MainMenu"

Якщо користувач натиснув "Почати гру" тоді

Стан_гри ← "GameInit"

Якщо Стан_гри == "GameInit" тоді

Завантажити сцену "GameScene"

Ініціалізувати ігрове поле:

- Створити/розмістити гравця

- Розмістити перешкоди, платформи, UI

- Налаштувати камеру

Стан_гри ← "LevelStart"

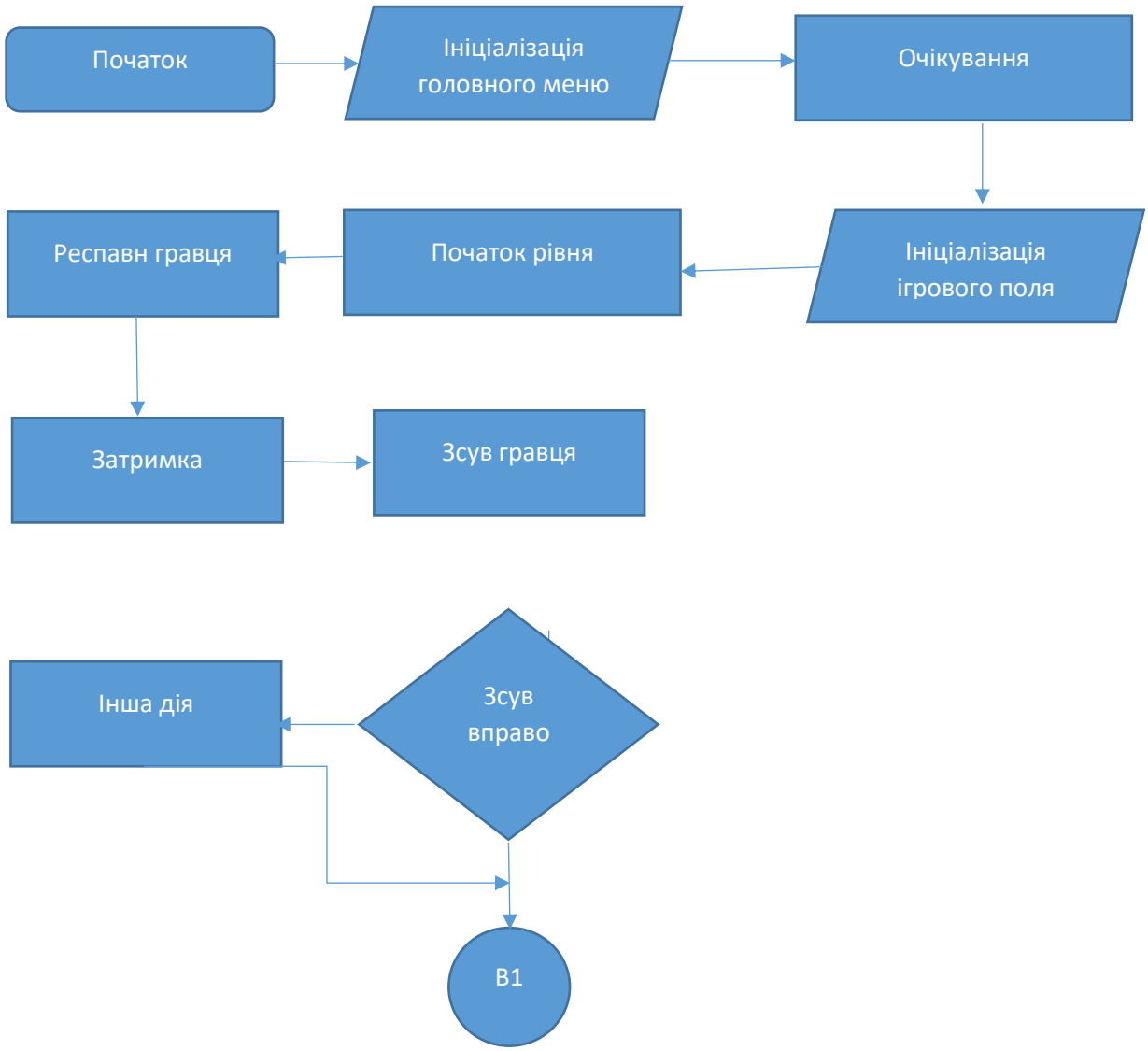

Рисунок 2.2 – Схема алгоритмів (частина 1)

Псевдокод легко адаптувати під реальну логіку в Unity з використанням MonoBehaviour, Update(), Start(), SceneManager.LoadScene().

2.3 Діаграма прецедентів

Використання UML-діаграм у процесі розробки ігрового модуля на платформі Unity є надзвичайно важливим і актуальним з точки зору забезпечення структурованого, логічного та ефективного підходу до проєктування програмної архітектури.

UML — це стандартизована мова моделювання, яка дозволяє візуалізувати структуру і поведінку системи до початку її реалізації в коді. У контексті розробки ігор, де взаємодіє велика кількість об’єктів, подій, сценаріїв та логіки, UML виступає незамінним інструментом для:

Формалізації задуму гри — UML-діаграми допомагають описати, як саме працює модуль: які об'єкти він містить, як вони взаємодіють, які методи і змінні реалізують його функціональність. Це спрощує комунікацію між членами команди.

Планування архітектури — за допомогою діаграм класів, об'єктів і компонентів можна спроєктувати ігрову архітектуру на високому рівні, що дає змогу уникнути помилок у майбутньому при розширенні гри.

Аналізу поведінки системи — діаграми станів або активностей дозволяють візуалізувати, як змінюється стан гри, персонажа чи об’єкта залежно від дій користувача або внутрішньої логіки.

Оптимізації процесу розробки — наявність моделей дозволяє швидше перейти до кодування, оскільки розробники чітко бачать структуру і взаємозв’язки. Це знижує кількість змін у коді на пізніх етапах розробки.

Уніфікації документації — UML-діаграми легко зрозумілі не лише програмістам, а й гейм-дизайнерам, аналітикам, тестувальникам. Це сприяє кращому розумінню системи на різних рівнях розробки.

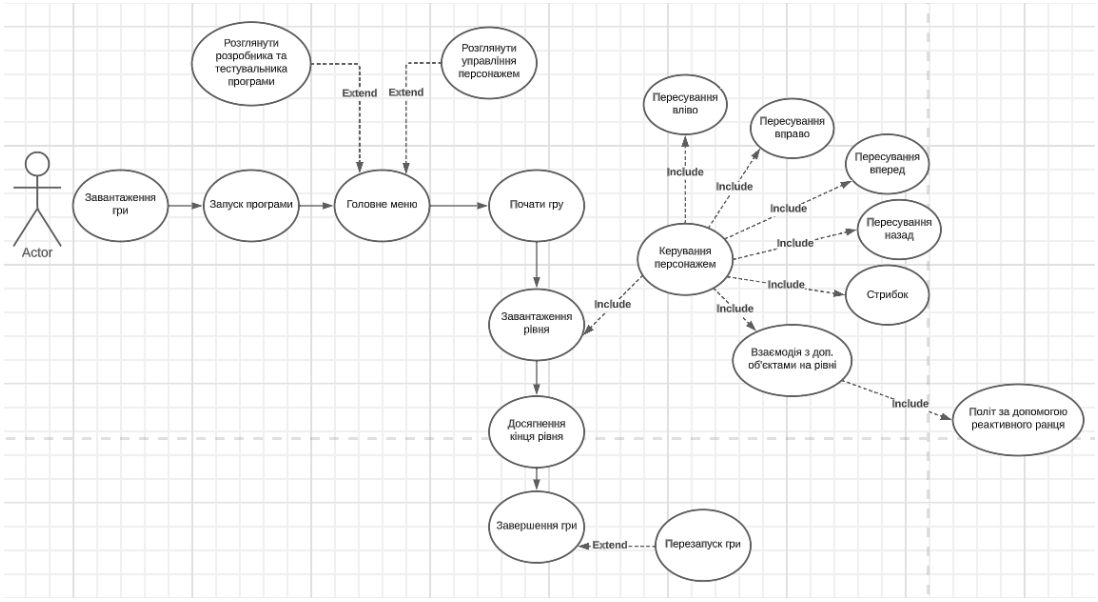

Рисунок 2.4 – Діаграма варіантів використання

На рисунку 2.5 зображено діаграму класів, яка була створена для даного програмного продукту, що розробляється.

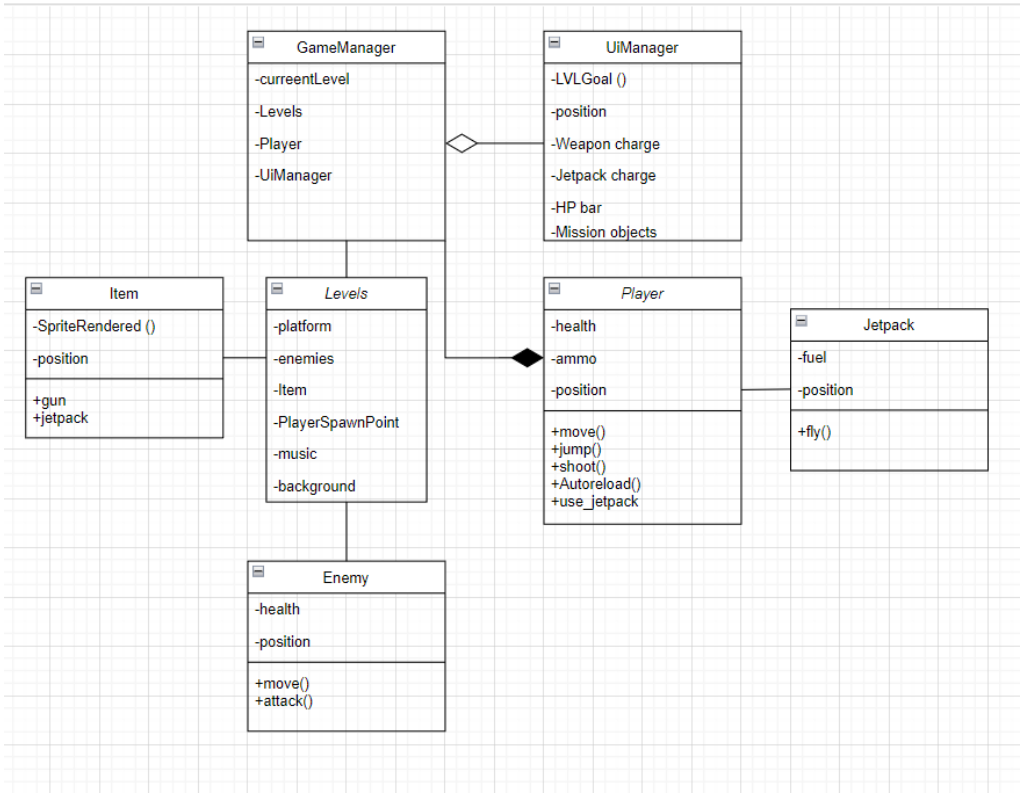

Рисунок 2.5 – Діаграма класів

GameManager - Центральний клас, який керує логікою гри.

Поля:

currentLevel — поточний рівень.

Levels — набір або посилання на всі доступні рівні.

Player — посилання на об’єкт гравця.

UiManager — керує елементами інтерфейсу.

Зв’язки:

Агрегація з UiManager — GameManager містить посилання на UI, але не володіє ним.

Композиція з Levels — рівень створюється в межах GameManager і зникає разом із ним.

UiManager відповідає за відображення інформації на екрані.

Методи:

LVLGoal() — ймовірно, оновлює або перевіряє ціль рівня.

Поля:

position — положення UI-елемента.

Weapon charge, Jetpack charge, HP bar, Mission objects — стани інтерфейсу, які відображають стан гравця чи місії.

Levels описує структуру рівня.

Поля:

platform, enemies, Item, PlayerSpawnPoint, music, background — усі основні компоненти, які формують рівень: локації, вороги, предмети, точка старту, візуальне та звукове оформлення.

Зв’язки:

Композиція з Enemy, Item, Player — всі ці елементи створюються в межах рівня і належать йому.

Player - Оосновний клас гравця.

Поля:

health, ammo, position — стандартні параметри життєздатності та боєздатності.

Методи:

move(), jump(), shoot(), Autoreload(), use_jetpack() — типові дії, які може виконати гравець.

Зв’язки:

Агрегація з Jetpack — гравець може мати реактивний ранець.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО МОДУЛЮ

3.1 Структура програмного модулю

Unity використовує компонентний підхід, де всі об’єкти гри (GameObjects) складаються з набору компонентів. Кожен компонент відповідає за певну функціональність (рендеринг, фізика, логіка тощо). Основний спосіб додати логіку у гру — це написання сценаріїв на мові C#. Скрипти прикріплюються як компоненти до GameObjects. Можна керувати поведінкою об’єктів, подіями, взаємодіями. Використовують життєві цикли методів, наприклад Start(), Update(), FixedUpdate(). Структуру директорії програмного модулю наведено на рисунку 3.1.

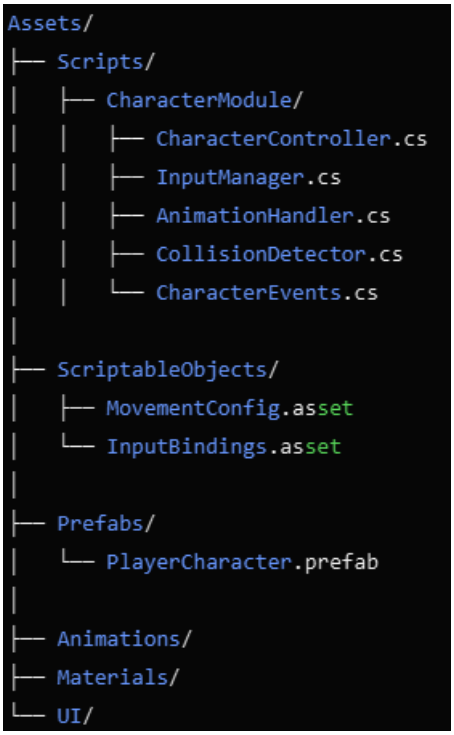

Рисунок 3.1 - Структура модулю

Assets/ - це коренева папка, яка містить усі активи проекту Unity.

Scripts/ - ця папка призначена для всіх C# скриптів, які керують логікою гри.

CharacterModule/ - це підпапка, яка містить скрипти, що стосуються безпосередньо керування персонажем.

CharacterController.cs: - скрипт відповідає за базове переміщення персонажа, його фізичну взаємодію та загальну логіку поведінки.

InputManager.cs обробляє введення від користувача (клавіатура, миша, геймпад) і перетворює його в ігрові дії.

AnimationHandler.cs - керує анімаціями персонажа, синхронізуючи їх з рухами та діями.

CollisionDetector.cs відповідає за виявлення колізій персонажа з іншими об'єктами у світі та обробку цих подій.

CharacterEvents.cs: містить визначення подій (наприклад, "персонаж стрибнув", "персонаж отримав шкоду"), до яких можуть підписуватися інші скрипти.

ScriptableObjects/ папка, містить ScriptableObjects, які є контейнерами даних, які можна зберігати в проекті та редагувати в інспекторі Unity. Це дозволяє розділити дані від коду.

MovementConfig.asset: містить налаштування руху персонажа, такі як швидкість переміщення, висота стрибка, прискорення тощо.

InputBindings.asset: Ймовірно,містить налаштування прив'язок клавіш або кнопок до ігрових дій.

Prefabs містить префаби, які є повторно використовуваними ігровими об'єктами. Зміни, внесені до префаба, автоматично застосовуються до всіх його екземплярів у сцені. PlayerCharacter.prefab - we префаб ігрового персонажа, який, ймовірно, включає в себе модель персонажа, його компоненти (наприклад, Rigidbody, Collider) та прикріплені скрипти (наприклад, CharacterController, AnimationHandler).

Animations/ призначена для анімаційних кліпів (наприклад, "біг", "стрибок", "атака") для персонажів та інших об'єктів.

Materials/ містить матеріали, які визначають, як об'єкти виглядають (колір, текстури, шейдери).

Чітке розділення за типом активів (скрипти, префаби, матеріали) та за функціоналом (CharacterModule). Скрипти, що стосуються персонажа, згруповані в окремий модуль, що полегшує їх пошук та управління.

Розділення даних від логіки - використання ScriptableObjects для конфігурації дозволяє гейм-дизайнерам легко налаштовувати параметри без втручання в код. Префаби дозволяють швидко створювати екземпляри об'єктів та забезпечувати їх послідовність.

Така структура є гнучкою і може бути легко розширена для більших ігрових проектів.

3.2 Тестування роботи модулю

Тестування програмного коду в Unity для створення ігрового модуля включає кілька основних способів, які забезпечують якість і надійність розробленого продукту. Перш за все, широко застосовується ручне тестування, коли розробник або тестувальник запускає гру у редакторі Unity або зібраному білді і перевіряє роботу функціоналу, взаємодії та поведінки об’єктів, що є швидким, але менш ефективним для великих проектів через складність відтворення помилок. Для більш системного підходу в Unity існує вбудований Unity Test Framework, який дозволяє створювати автоматизовані тести двох типів: Edit Mode Tests, що виконуються поза грою і перевіряють логіку і функції класів, а також Play Mode Tests, які запускаються у ігровому режимі для тестування взаємодії об’єктів, анімацій та UI, забезпечуючи глибоку перевірку геймплейної логіки. Додатково в процесі тестування застосовується логування через Debug.Log(), що допомагає відстежувати виконання коду та виявляти помилки або неочікувану поведінку. Для аналізу продуктивності використовується вбудований профайлер Unity, який дозволяє досліджувати завантаження CPU, GPU, використання пам’яті та час кадру, що

є важливим для оптимізації гри. Важливо також проводити інтеграційне тестування, яке перевіряє коректну взаємодію між різними компонентами модуля, а тестування інтерфейсу користувача забезпечується через Play Mode тести або сторонні інструменти, які імітують взаємодію гравця з UI. Окрім цього, значна увага приділяється тестуванню на різних платформах, оскільки Unity дозволяє збирати проекти під Windows, Android, iOS, WebGL та інші, що дає змогу врахувати особливості апаратного забезпечення, керування та продуктивності на цільових пристроях. Таким чином, поєднання ручного, автоматизованого, логування, профайлінгу та мультиплатформного тестування забезпечує комплексний контроль якості ігрового модуля в Unity.

Таблиця 3.1 – Тест кейс для перевірки відображення головного меню

Номер	*kT9z2J*/001.	
Назва	Перевірка коректності відображення головного меню	
Функція	Перевірка	
Дія	**Очікуваний результат**	**Результат тесту**: – пройдено – провалено
Передумова:		
Запустити програму	Ігровий модуль відкритий і доступний	пройдено
Кроки тесту		
Запуск програми	Програма запустилась	пройдено
Перевірка кнопок	Всі кнопки працюють	пройдено

Таблиця 3.2 – Тест кейс для перевірки запуску програми

Номер	*kT9z2J*/002.	
Назва	Перевірка коректності функції запуску програми	
Функція	Запуск програми	
Дія	**Очікуваний результат**	**Результат тесту**: – пройдено – провалено
Передумова:		
Запуск програми	Ігровий модуль відкритий і доступний	пройдено
Натиснути кнопку запуску гри	Запуск початку гри	пройдено
Кроки тесту		
Запуск програми	Програма запустилась	пройдено
Натиснути кнопку старт	Початок гри	пройдено

Таблиця 3.3 – Тест кейс для перевірки коректності ходьби персонажем

Номер	*kT9z2J*/003.	
Назва	Перевірка коректності роботи ходьби	
Функція	Ходьба головного героя	
Дія	**Очікуваний результат**	**Результат тесту**: – пройдено – провалено
Передумова:		
Запуск програми	Ігровий модуль відкритий і доступний	пройдено
Натиснути кнопку старт	Запуск початку гри	пройдено
Натиснути W/S/A/D	Всі кнопки працюють коректно	пройдено
Кроки тесту		
Запуск програми	Програма запустилась	пройдено
Натиснути кнопку старт	Початок гри	пройдено
Натиснути кнопки W/S/A/D	Всі кнопки керування коректно працюють	пройдено

Таблиця 3.4 – Тест кейс для перевірки ворожих об’єктів

Номер	*kT9z2J*/004.	
Назва	Перевірка коректності роботи ворожих об’єктів.	
Функція		
Дія	**Очікуваний результат**	**Результат тесту**: – пройдено – провалено
Передумова:		
Запуск програми	Ігровий модуль відкритий і доступний	пройдено
Натиснути кнопку старт	Запуск початку гри	пройдено
Натиснути W/S/A/D	Всі кнопки працюють коректно	пройдено
Постояти на місці	Отримати шкоду від ворожого об’єкту	пройдено
Кроки тесту		
Запуск програми	Програма запустилась	пройдено
Натиснути кнопку старт	Початок гри	пройдено
Натиснути кнопки W/S/A/D	Всі кнопки керування коректно працюють	пройдено
Постояти на місці	Отримання шкоди від ворожого об’єкту	пройдено

3.3 Інтерфейс розробки програмного модулю

Unity дозволяє розташовувати об’єкти у сцені за допомогою графічного редактора, що значно спрощує процес створення рівнів. Unity має вбудовані фізичні двигуни (PhysX), що дозволяють додавати реалістичну поведінку об’єктів (зіткнення, гравітація). Анімації можна створювати через Animator і анімаційні контролери. Для створення інтерфейсу гри (кнопки, меню, панелі) використовується система UI, заснована на Canvas, куди додаються різні UI-елементи. Сцена для розташування об’єктів ігрового модуля виглядає наступним чином (рисунок 3.2).

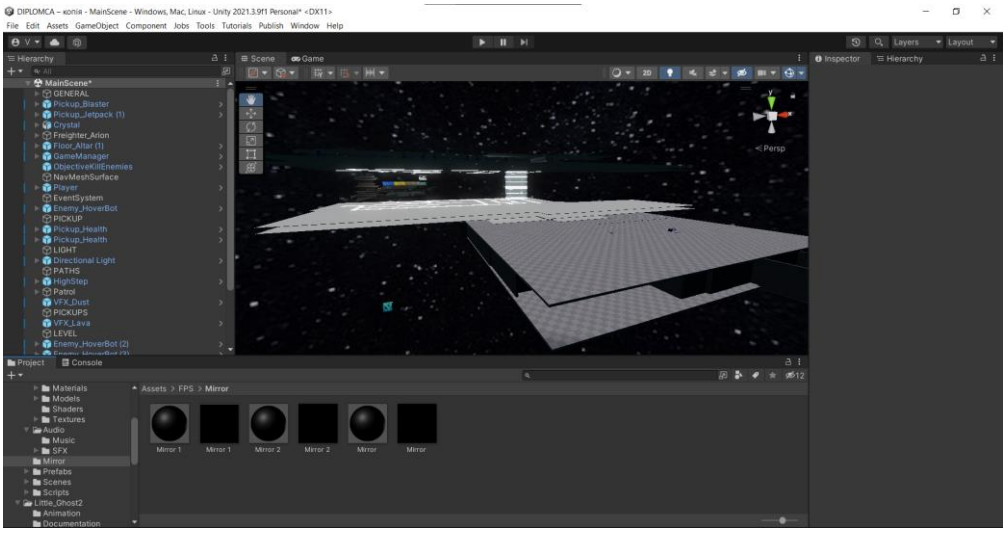

Рисунок 3.2 – Основна сцена

Для того щоб розмістити 3D об’єкт, ефекти, освітлення та аудіо, потрібно натиснути правою кнопкою миші по панелі де розміщуються всі об’єкти та вибрати об’єкт який хотіли б розмістити (рисунок 3.3).

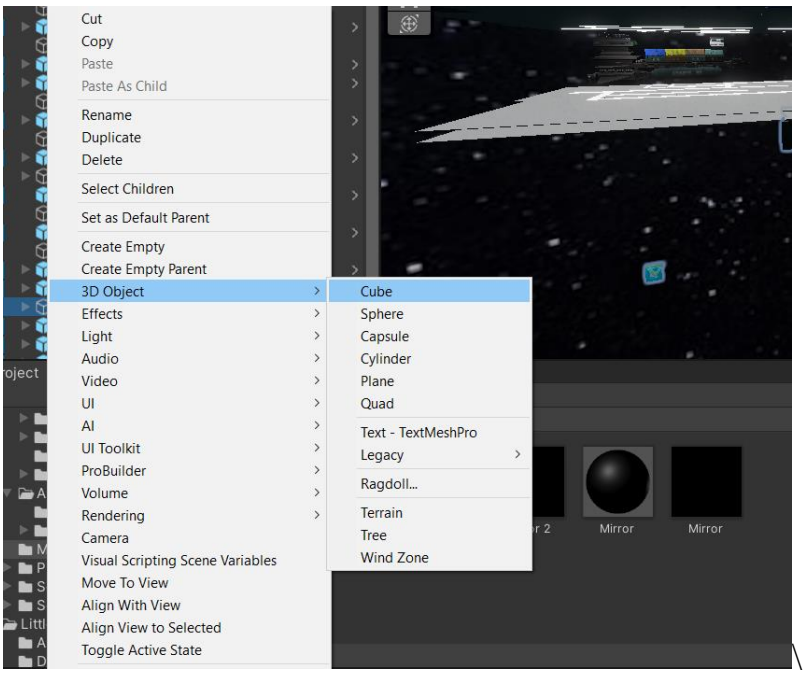

Рисунок 3.3 – Створення об’єкту

Для того щоб об’єкт працював, покажемо на прикладі дзеркала, потрібно створити *3d* об’єкт куба та розташувати його позицію по *x, y , z* (рисунок 3.3.), потім потрібно вибрати розмір та положення даного об’єкту, теж по *x, y , z* (рисунок 3.4), після чого створюємо такі компоненти як *camera, render texture* та *material* (рисунок 3.5), які потрібно поєднати між собою та розташувати на *3d* об’єкті (рисунок 3.6).

Рисунок 3.3 – Розташування та вибір розміру об’єкту

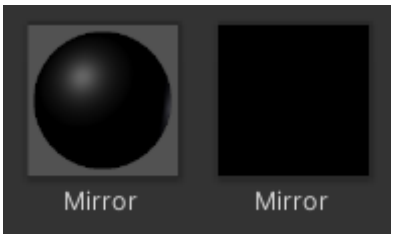

Рисунок 3.4 – Створення компонентів для дзеркала

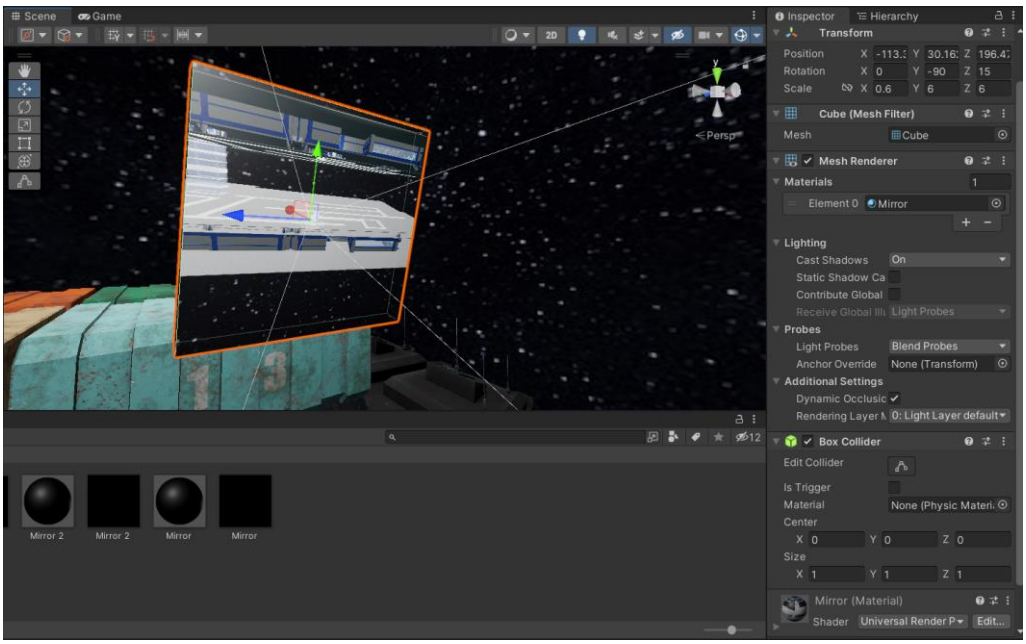

Рисунок 3.5 – Розташування всіх компонентів на *3d* об’єкт дзеркало

Unity Editor складається з декількох основних вікон:

Hierarchy - показує всі об’єкти, що знаходяться у поточній сцені, у вигляді дерева.

Scene - візуальний редактор, де розміщують і редагують об’єкти в 3D або 2D просторі.

Game - вікно для попереднього перегляду гри з камер, як її побачить гравець.

Inspector - Відображає властивості вибраного об’єкта або компонента, дозволяє їх редагувати.

Project - показує всі файли, ресурси, скрипти проєкту у вигляді файлової системи.

Console - виводить повідомлення про помилки, попередження і логування з коду.

Toolbar - містить основні інструменти для навігації по сцені, запуску гри, вибору режимів редагування.

Після запуску гри користувачеві відображається головна сторінка, яка автоматично заповнюється, а також відображаються кнопки «Відтворити» та

«Керування». На рисунку 3.6 показано інтерфейс головної сторінки, який з’являється одразу після запуску програми.

На головній сторінці користувачеві доступні дві кнопки: «Грати» та «Управління». Після натискання кнопки «Управління» з’являється вікно з інформацією про клавіші керування персонажем (рисунок 3.7), після чого гравець може натиснути кнопку «Грати» та розпочати гру..

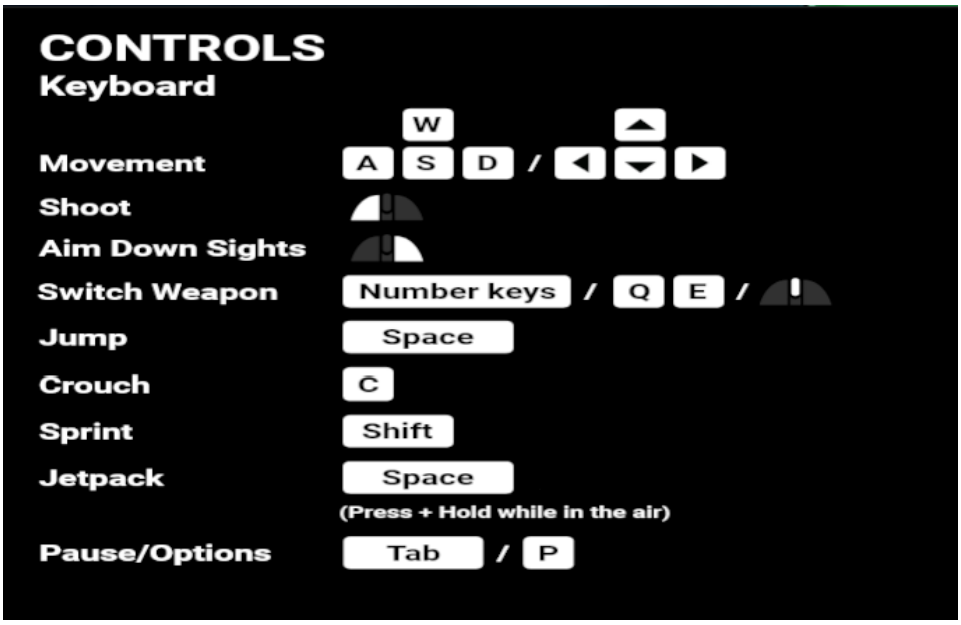

Рисунок 3.7 – Керування персонажем

Після натискання кнопки *Play* гравець з’являється на території рівня та зможе розглянути інтерфейс гри(рисунок 3.8).

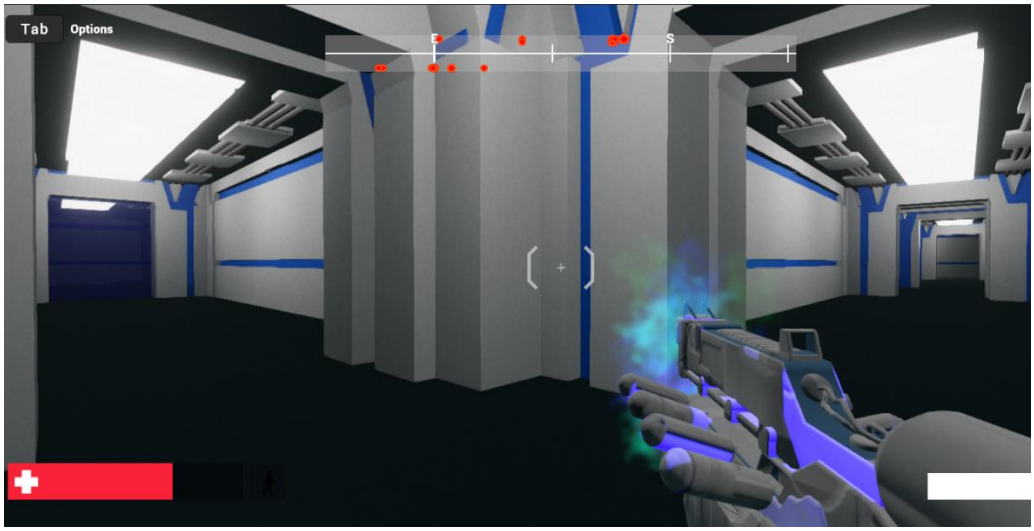

Рисунок 3.8 – Вигляд інтерфейсу гри

Будова рівня доволі таки немаленька і складається з трьох поверхів, на першому поверсі гравцю потрібно буде подолати ворожих ворогів першого поверху (рисунок 3.4) та натиснути на кнопку (рисунок 3.9), після чого відкриється прохід на другий поверх.

Рисунок 3.9 – Ворожий об’єкт першого поверху

Після проходження першого поверху гравець переходить на другий де перед ним буде перешкода у вигляді ворожого об’єкту другого поверху (рисунок 3.10).

Рисунок 3.10 – Ворожий об’єкт другого поверху

Після знищення майже всіх ворогів на карті та розв'язання всіх головоломок, гравець може або негайно знищити останнього ворога, або переглянути пройдений шлях, потім знищити останнього ворога та завершити гру.

ВИСНОВКИ

На основі аналітичного підходу проведено порівняльний аналіз програмних засобів для розробки ігрового модулю засобами фреймворку Unity, що дозволило виділити ключові компоненти модулю та технології реалізації.

На основі аналізу попередніх рішень, розроблено структурну схему проекту з основними елементами, що дозволило виділити основні компоненти ігрового модулю для подальшої реалізації.

На основі алгоритмічного підходу, розроблено алгоритм роботи ігрового модулю, що включає блоки ініціалізації ігрового модулю та керування персонажем, що дозволило розробити підстави для програмної реалізації.

На основі підходів до розробки програмного забезпечення, зокрема з використанням фреймворку Unity, програмно реалізовано модуль ігрової частини, наведено порівняльні характеристики та результати тестування, що дозволило визначити модуль як стабільний.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ілюшин . Програмний модуль для комп'ютерної гри на основі технології Unity. *II Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі» (ІКСМ весна 2025)*. (м. Тернопіль, . м. Тернопіль, С. 116.

2. Liu W., Anguelov D., Erhan D. та ін. SSD: Single Shot MultiBox Detector. *Computer Vision – ECCV 2016*. ред. Bastian Leibe, Jiri Matas, Nicu Sebe, Max Welling. Cham : Springer International Publishing, 2016. С. 21–37. DOI:10.1007/978-3-319-46448-0_2.

3. shape-predictor-68-face-landmarks · GitHub Topics · GitHub. URL: https://github.com/topics/shape-predictor-68-face-landmarks (accessed 12/05/2025).

4. William I., Ignatius Moses Setiadi D. R., Rachmawanto E. H. et al. Face Recognition using FaceNet (Survey, Performance Test, and Comparison). *2019 Fourth International Conference on Informatics and Computing (ICIC)2019 Fourth International Conference on Informatics and Computing (ICIC)*. (Semarang, Indonesia, 10.2019). Semarang, Indonesia : IEEE, 2019. DOI:10.1109/ICIC47613.2019.8985786. P. 1–6.

5. GitHub - TadasBaltrusaitis/OpenFace: OpenFace – a state-of-the art tool intended for facial landmark detection, head pose estimation, facial action unit recognition, and eye-gaze estimation. URL: https://github.com/TadasBaltrusaitis/OpenFace (accessed 12/05/2025).

6. Facial Emotion Recognition Dataset. URL: https://www.kaggle.com/datasets/tapakah68/facial-emotion-recognition (accessed 12/05/2025).

7. AffectNet Dataset | Papers With Code. URL: https://paperswithcode.com/dataset/affectnet (accessed 24/05/2025).

8. Mollahosseini A., Hasani B., Mahoor M. H. AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild. 2017. DOI:10.48550/ARXIV.1708.03985.

9. Real-world Affective Faces (RAF) Database. URL: http://www.whdeng.cn/RAF/model1.html (accessed 24/05/2025).

10. Dhall A., Goecke R., Gedeon T. та ін. Emotion recognition in the wild. *Journal on Multimodal User Interfaces*. Вип. 10, № 2. С. 95–97. DOI:10.1007/s12193-016-0213-z.

11. CheyneyComputerScience/CREMA-D. Crowd Sourced Emotional Multimodal Actors Dataset (CREMA-D). Cheyney Computer Science, 2025. URL: https://github.com/CheyneyComputerScience/CREMA-D (accessed 25/05/2025).2025.

12. DAiSEE : Dataset for Affective States in E-Environments. URL: https://people.iith.ac.in/vineethnb/resources/daisee/index.html (accessed 25/05/2025).

13. Глибовець М. М., Олецький О. В. Штучний інтелект: Підручник. К. : Вид. дім „КМ Академія”, 2002. 366 с.

14. Luger G. F. Artificial intelligence : structures and strategies for complex problem solving. 6th. Pearson Education, 2009. 779 p. Also available online, URL: http://www.uoitc.edu.iq/images/documents/informatics-institute/exam_materials/artificial%20intelligence%20structures%20and%20strategies%20for%20%20complex%20problem%20solving.pdf

15. Garvey C. Artificial Intelligence and Japan’s Fifth Generation. *Pacific Historical Review*. Vol. 88, Issue 4. P. 619–658.

16. Kavanagh, S., Luxton-Reilly, A., Wuensche, B., & Plimmer, B. (2017). A systematic review of Virtual Reality in education. Themes in Science and Technology Education, 10(2), 85-119.

17. Aguayo, C., & Eames, C. (2023). Using mixed reality (XR) immersive learning to enhance environmental education. The Journal of Environmental Education, 54(1), 58–71. https://doi.org/10.1080/00958964.2022.2152410

18. Abdullah M. Al-Ansi, Mohammed Jaboob, Askar Garad, Ahmed Al-Ansi, Analyzing augmented reality (AR) and virtual reality (VR) recent development in education, Social Sciences & Humanities Open, Volume 8, Issue 1, 2023, 100532, SSN 2590-2911,

19. Lo TTS, Chen Y, Lai TY and Goodman A (2024), Phygital workspace: a systematic review in developing a new typological work environment using XR technology to reduce the carbon footprint. Front. Built Environ. 10:1370423. doi: 10.3389/fbuil.2024.1370423

20. XR and Workers 'safety in High-Risk Industries: A comprehensive review. Joana Eva Dodoo a, Hosam Al-Samarraie b,* , Ahmed Ibrahim Alzahrani c, Tang Tang b

21. Mazurek, J., Kiper, P., Cieślik, B., Rutkowski, S., Mehlich, K., Turolla, A., Szczepańska-Gieracha, J. (2019). Virtual reality in medicine: a brief overview and future research directions. Human Movement, 20(3), 16-22. https://doi.org/10.5114/hm.2019.83529

22. Morimoto, T.; Kobayashi, T.; Hirata, H.; Otani, K.; Sugimoto, M.; Tsukamoto, M.; Yoshihara, T.; Ueno, M.; Mawatari, M. XR (Extended Reality: Virtual Reality, Augmented Reality, Mixed Reality) Technology in Spine Medicine: Status Quo and Quo Vadis. J. Clin. Med. 2022, 11, 470. https://doi.org/10.3390/jcm11020470

23. https://registry.khronos.org/OpenXR/specs/1.1/html/xrspec.html

24. Aziz, S., Lohr, D. J., Friedman, L., & Komogortsev, O. (2024, June). Evaluation of eye tracking signal quality for virtual reality applications: A case study in the meta quest pro. In Proceedings of the 2024 Symposium on Eye Tracking Research and Applications (pp. 1-8)