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)

PvP-гра з клієнт-серверною архітектурою

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-5
Максим МЕДВЄДЄВ
(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: ас. Владислав ХОЛЄВ
(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 Медвєдєву Максиму Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи PvP-гра з клієнт-серверною архітектурою

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

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи 1) Мережева архітектура: Клієнт-серверна; 2) Жанр гри: PvP.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз проблеми та огляд існуючих рішень;
2) вибір засобів розробки;
3) розробка алгоритму ігрового циклу;
4) розробка програмних модулів;
5) тестування застосунку;
6) висновки.

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

Слайд-презентація – 9 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	27.05.25-31.05.25	
2	Вибір засобів розробки	01.06.25-02.06.25	
3	Розробка алгоритму ігрового циклу	03.06.25-05.06.25	
4	Розробка та тестування застосунку	06.06.25-09.06.25	
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Владислав ХОЛЄВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 64 с., 19 рис., 2 дод., 11 джерел.

ІГРОВИЙ ЗАСТОСУНОК, КОМП'ЮТЕРНА МЕРЕЖА, СЕРВЕР, КЛІЄНТ, ІГРОВИЙ РУШІЙ, СЕРЕДОВИЩЕ РОЗРОБКИ.

Метою кваліфікаційної роботи є розробка PvP-гри з використанням клієнт-серверної архітектури, що дозволяє забезпечити багатокористувацький режим з належним рівнем синхронізації, безпеки та продуктивності.

У ході виконання кваліфікаційної роботи буде проведено дослідження принципів побудови клієнт-серверної архітектури у контексті багатокористувацьких ігор. Розглядатимуться підходи до синхронізації ігрових об'єктів, обробки мережевих подій та оптимізації обміну даними між клієнтами та сервером. Буде здійснено вибір інструментів та технологій для реалізації гри, зокрема ігрового рушія, мережевого протоколу, формату передачі даних та підходів до обробки запитів у реальному часі.

У рамках реалізації буде створено ігровий сервер, здатний обслуговувати одночасно кількох клієнтів, та клієнтський застосунок, який забезпечує взаємодію гравця з ігровим світом та іншими учасниками. Передбачено також обробку затримок та компенсацію мережевої нестабільності. У фінальному етапі буде проведено тестування проекту, виявлення та усунення можливих помилок, а також аналіз продуктивності та масштабованості реалізованої системи. Результатом стане функціональний PvP-прототип гри з інтеграцією EOS, що демонструє коректну роботу клієнт-серверної моделі, здатної забезпечити одночасну гру декількох гравців із мінімальними затримками та високою стабільністю.

ABSTRACT

Bachelor’s thesis: 64 pages, 19 figures, 2 appendices, 11 sources.

GAME APPLICATION, COMPUTER NETWORK, SERVER, CLIENT, GAME ENGINE, DEVELOPMENT ENVIRONMENT.

The major goal of this thesis is to develop a PvP game using a client-server architecture, enabling a multiplayer mode with an appropriate level of synchronization, security, and performance.

In order to achieve this goal, the principles of building client-server architectures in the context of multiplayer games will be explored. This includes studying approaches to synchronizing game objects, processing network events, and optimizing data exchange between clients and the server. The most suitable tools and technologies will be selected for the implementation, including the game engine, network protocols, data transmission formats, and methods for handling real-time requests.

A dedicated game server will be developed to support multiple clients simultaneously, along with a client-side application that enables player interaction with the game world and with other users. The system will also include latency handling and network instability compensation mechanisms.

At the final stage, the project will undergo testing and debugging. The performance and scalability of the implemented system will be analyzed. The outcome will be a functional PvP game prototype with EOS integration, demonstrating stable and synchronized multiplayer gameplay with minimal latency.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 8
ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Класифікація ігрових застосунків 11
1.2 Аналіз існуючих розробок у жанрі PvP стратегії 12
1.2.1 XCOM 2 13
1.2.2 Dofus 14
1.2.3 Divinity: Original Sin 2 15
1.2.4 Atlas Reactor 16
1.3 Постановка вимог до гри 17
2 ВИБІР ЗАСОБІВ РОЗРОБКИ ТА ПРОЕКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ 19
2.1 Аналіз ігрових рушіїв 19
2.1.1 Unity 20
2.1.2 Godot 21
2.1.3 Unreal 23
2.2 Аналіз інтегрованих середовищ розробки 26
2.2.1 Visual Studio 26
2.2.2 Rider 27
2.3 Аналіз систем контролю версій 28
2.3.1 Subversion 28
2.3.2 Helix Core 29
2.3.3 Git 29
2.4 Алгоритм реалізації та структура ігрового циклу 30
3 РОЗРОБКА ЗАСТОСУНКУ 34
3.1 Користувацький інтерфейс 34
3.1.1 Базові елементи 34

3.1.2 Віджети 36
3.1.3 Сторінки 38
3.2 Ігрова логіка 40
3.2.1 Ігровий контролер 40
3.2.2 Ігровий персонаж 43
3.2.3 Інстанція гри 46
4 ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ 48
ВИСНОВКИ 50
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 52
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 53
ДОДАТОК Б Код уміння для пересування 59

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Сеттинг – сукупність усіх елементів, що формують місце, час і обставини, в яких розгортається дія художнього твору (книги, фільму, гри, коміксу тощо) (англ., setting)

Co-op – жанр ігор, що передбачає кооперування між живими гравцями (англ., cooperative).

DAW – цифрова робоча станція для аудіо (англ., Digital Audio Workstation)

DLC – додатковий цифровий контент, який розробники випускають для відеоігри після її початкового релізу (англ., Downloadable Content)

GAS – система ігрових умінь (англ.,Gameplay Ability System)

HUD – частина ігрового інтерфейсу, що відображується без виклику додаткових меню (англ., Head-Up Display)

IDE – інтегроване середовище розробки (англ., Integrated Development Environment)

NFT – унікальні цифрові активи, котрі представляють конкретний цифровий об’єкт (англ., Non-Fungible Token)

PvP – жанр ігор, що передбачає протистояння між живими гравцями (англ., Player versus Player)

PvPvE – жанр ігор, що передбачає протистояння гравців та ботів (англ., Player versus Player versus Enviroment)

UI – користувацький інтерфейс (англ., User Interface)

VR – віртуальна реальність (англ., Virtual Reality)

VSC – система контролю версій (англ., Version Control System)

ВСТУП

Історія відеоігор почалася ще в середині XX століття. Тоді американський фізик Вільям Гіґінботам для розваги гостей Брукхейвенської національної лабораторії створив гру «Tennis for Two». Її метою була симуляція тенісного матчу з урахування швидкості віртуального м’яча та сили імітованої гравітації. В якості пристрою виведення було використано екран осцилографа, а для обробки дій користувача – аналоговий алюмінієвий контролер. Тоді цей винахід викликав величезний інтерес серед відвідувачів, що в подальшому стало передумовою для виникнення окремої сфери цифрових розваг. Вже через тринадцять років, завдяки зусиллям Білла Піттс та Нолана Бушнелла було створено перший комерційний пристрій з відеогрою, який можна вважати предком сучасних ігрових консолей. Із кожним роком кількість гравців невпинно зростала, так само як і жанрів ігор.

У 1990-х роках почали з’являтися псевдотривимірні ігри. Однією із найвідоміших стала Doom, яка продалася рекордними тиражами. Однак із збільшенням конкуренції та зростаючими очікуваннями аудиторії процес створення відеоігор значно ускладнився, а витрати на розробку наблизились до рівня кінематографічних проектів. Це спричинило переосмислення підходів до розробки: стало очевидним, що постійне створення однотипних технологій «з нуля» є економічно недоцільним. Почали з’являтися компанії, що продавали ліцензію на використання готових інструментів для розробки ігор. Це дозволило робити ігри дешевше і відкрило дорогу на ринок незалежним студіям з невеликими бюджетами.

У наш час ігрова індустрія характеризується надзвичайним різноманіттям: від мобільних казуальних до складних VR-симуляторів, від великомасштабних проектів із багатомільйонними до експериментальних інді-проектів, створених одним розробником. Сучасні тенденції включають активне впровадження штучного інтелекту для генерації контенту, розвиток

хмарного геймінгу, інтеграцію блокчейн-технологій та NFT, а також зростаючу популярність ігор-сервісів, котрі об’єднують гравців із всього світу в одному віртуальному світі.

Головною причиною збільшення попиту на онлайн-ігри став розвиток мережевих технологій. Завдяки великій кількості активної аудиторії у межах таких проектів сформувалися окремі жанри, зокрема PvP, PvPvE та Co-op

Загальним принципом роботи таких ігрових застосунків є локальна симуляція дій, котрі виконує інший гравець, і яка відбувається шляхом постійного обміну даними між пристроями мережею. Розробка таких продуктів вимагає набагато більше ресурсів, тому навіть у наш час велика частина ігрових застосунків не має такого функціоналу. Технічна складність багатокористувацьких ігор полягає не лише в організації мережевої взаємодії, але й у забезпеченні синхронізації стану світу, контролі та перевірці даних з клієнтів, оптимізації трафіку та мінімізації впливу мережевих затримок на користувацький досвід. Крім того, такі проекти потребують потужної серверної інфраструктури, систем моніторингу, резервного копіювання та цілодобової технічної підтримки, що значно збільшує експлуатаційні витрати порівняно з однокористувацькими іграми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Класифікація ігрових застосунків

Ключовою відмінністю відеоігор від будь-якого іншого виду мистецтва є надзвичайне різноманіття жанрів та їх поєднань. Адже кіно чи література класифікуються за стилем оповідання, мистецтво – зображуваною епохою, а ігри не тільки містять усе це, а ще й додають інтерактивність, чим роблять цей вид творчості вітриною усього культурного нащаддя людства.

Саме через таку велику різноманітність віртуальних світів, важко визначати їх приналежність до певної категорії, проте використовуючи рекурсивний підхід при класифікації, можна визначити набір жанрів для кожної гри. Головною ознакою кожного ігрового застосунку є режим взаємодії користувача із нею. Так ігри поділяються на:

- Одиночні ігри орієнтовані на одного користувача, який взаємодіє виключно з внутрішньою логікою гри – заздалегідь визначеними сценаріями, штучним інтелектом та оточенням. Основна увага приділяється сюжету, геймплейним викликам та атмосфері;

- Багатокористувацькі ігри дозволяють одночасну участь декількох гравців у спільному ігровому просторі. Вони в свою чергу поділяються на кооперативні, де гравці об’єднуються для вирішення задачі, та змагальні де гравці змагаються між собою.

Наступною важливо характеристикою, за якою відбувається поділ ігор, є тип ігрових механік – інструментів, котрі надаються гравцеві для виконання завдань. За ними виділяються такі жанри:

- Шутер – жанр, який передбачає знищення ворогів шляхом використання різної зброї. Виділяється динамічним геймплеєм, різноманітністю арсеналу та ефектами. Поділ у цьому жанрі відбувається за розташуванням поля зору гравця, а саме від першої чи третьої особи;

- Стратегія – ігри, в яких гравець управляє певними ресурсами за для перемоги шляхом тактичного планування. В залежності від наявності порядку у діях гравців виділять покрокові стратегії та реальному часі;

- Головоломка – жанр, в яких гравець має вирішити логічні або просторові завдання. Зазвичай потребують використання логічного мислення;

- Симулятор – ігри, що імітують реальні або умовні процеси;

- Рольова гра – ігри, в яких гравець бере на себе роль персонажа з розвитком характеристик, виконанням завдань та дослідженням світу;

- Хорор – це жанр, що націлений на викликання у гравця емоцій страху, тривоги та нервозності. Основна мета полягає не лише в перемозі над супротивниками, а у переживанні психологічного або фізичного жаху;

- Файтинг – це жанр, у якому гравець бере участь у дуелях між двома або більше персонажами, що зазвичай використовують обмежений набір прийомів. Глибина геймплею створюється завдяки їх поєднанню, що утворює ігрові стилі.

Останнім ключовим аспектом при класифікації відеоігор є платформа кінцевого споживача: ПК, ігрова консоль чи мобільні телефони. Сучасним трендом стало покриття усіх можливих ігрових девайсів, за для збільшення пізнаваності франшизи та збільшення прибутків. Використовуючи таку підхід можна точно визначити існуючі розробки та цільову аудиторію. Для мого проекту це багатокористувацькі змагальні покрокові стратегії із елементами рольової гри на персональні комп’ютери.

1.2 Аналіз існуючих розробок у жанрі PvP стратегії

Перші стратегічні ігри з’явилися ще на початку 1980-х років, коли графічні можливості комп’ютерів були обмежені. Однією з перших стала Utopia, яка вважається першим зразком жанру стратегій у реальному часі. У 1990-х роках жанр набув широкої популярності завдяки появі Dune II, яка

задала класичну структуру ігрового циклу для жанру, а саме збирання ресурсів, будівництво бази та керування армією. Подальший розвиток стратегій був тісно пов'язаний із розвитком персональних комп’ютерів, зростанням обчислювальної потужності та мережевих технологій. Вони стали основою для перших кіберспортивних змагань (наприклад, турніри з StarCraft у Південній Кореї). Окрім того, ігри на кшталт Civilization продемонстрували, що представники жанру стратегій можуть поєднувати розважальну та освітню функції, навчаючи історії, економіці, дипломатії.

Серед популярних ігор, які належать до цієї категорії та досі мають базу активних гравців, можна виділити наступні проекти.

1.2.1 XCOM 2

XCOM 2 – покрокова тактична відеогра з рольовими елементами, розроблена Firaxis Games і видана 2K Ga. Основна частина ігрового процесу поділяється на два рівні: глобальне управління, де гравець розвиває базу, досліджує нові технології, вербує солдатів та керує ресурсами, та тактичні бої, де використовуючи загін із 4-6 персонажів, котрим необхідно перемогти хвилю прибульців. Ігровий процес проекту наведений на рисунку 1.1

Рисунок 1.1 – Ігровий процес у XCOM 2

Хоч основним режимом є однокористувацька сюжетна кампанія, двоє гравців можуть змагатися між собою по мережі, керуючи загонами повстанців або прибульців, що складаються з 6 бійців. З переваг даного проекту є досить сучасна графіка та можливість участі у рейтингових іграх. Серед недоліків даного продукту можна виділити відсутність випадкових події, що провокує появу найбільш ефективних наборів персонажів та обмежену кількість гравців, що можуть брати участь у грі.

1.2.2 Dofus

Dofus – це тактична покрокова багатокористувацька рольова гра, розроблена та видана Ankama Games французьким виробником відеоігор у жовтні 2005 року. Гравці керують 2D-аватаром, що належить до одного з 19 класів персонажів їх видом камери від третьої особи. Кожен клас має унікальний набір заклинань, які інші класи не можуть отримати. На рисунку 1.2 зображено ігровий процес цього проекту.

Рисунок 1.2 – Ігровий процес у Dofus

Змагальні онлайн матчі бувають двох видів: у відкритому світі та колізей. Усі учасники отримають хід у порядку ініціативи їх персонажів та мають обмежену кількість очок дії та руху. Переможцем визначається той гравець, персонажі якого зуміють повністю ліквідувати табір супротивника. Проте у всіх видах PvP, ігри відбуваються з урахуванням рівня та відкритих умінь у кооперативних місіях, що створює високий поріг входу для нових гравців і є серйозним недоліком. Також гра надає можливість створення платного облікового запису, що пропонує повний ігровий досвід, тоді як безкоштовний значно обмежений за кількість контенту. Це поділяє ігрову спільноту, сприяє ворожнечі й неповазі у внутрішньо-ігровому чаті та на форумах обговорення. Окрім того, можлива участь лише двох команд розміром до 3 гравців, що зменшує варіативність ігрових сценаріїв. Серед переваг гри можна зазначити велику, відносно жанру, кількість активних гравців та широке різноманіття персонажів.

1.2.3 Divinity: Original Sin 2

Divinity: Original Sin II – це ізометрична (вид камери гравця зверху) одно- та багатокористувацька рольова гра з тактичними покроковими боями випущена Larian Studios 14 вересня 2017 року. Дія гри розгортається у вигаданому світі Рівелон, через століття після історії першої частини. Ігрові обмеження для балансу персонажів, гармонійно вплетені у історію, завдяки чому вирішення задач є непростим проте справедливим. Основний акцент гри сфокусований на сюжетному режимі, а мережевий функціонал планувався лише для co-op взаємодії. Проте згодом почали з’являтися користувацькі модифікації для змагальної гри із використанням ключових персонажів історії. Головною із них стала арена – ігровий режим, що розрахований до 4 гравців та пропонує сутички із заздалегідь налаштованими героями на обмежених за розміром локаціями із сюжетної кампанії. Порядок дій визначається параметром ініціативи конкретного персонажа. Окрім

обмежених ігрових ресурсів, гравець має лімітовану кількість часу для здійснення ходу. Можливе налаштування фіксованої кількості раундів. В такому випадку переможець буде визначений за сукупною кількістю здоров'я. Ігровий процес змагального режиму зображено на рисунку 1.3.

Рисунок 1.3 – Ігровий процес у Divinity Original Sin 2

1.2.4 Atlas Reactor

Atlas Reactor – це покрокова тактична гра, розроблена Trion Worlds. У Atlas Reactor гравець бере на себе роль фрілансера в мегаполісі Atlas. Гра вийшла в жовтні 2016 року. Задача у матчі полягає в тому, щоб знешкодити більше ворогів впродовж 20 ходів. Стандартним вважається битва 4 на 4, проте є можливість створення користувацьких матчів. Ігровий процес гри зображено на рисунку 1.4. У грі представлено 3 класи – це персонажі передової, вогневої потужності та підтримки. При створенні героїв розробники надихалися такими рішеннями як XCOM: Enemy Unknown, League of Legends та Warhammer 40000. Проект мав сезону модель оновлень, тобто глобальні зміни відбувалися кожний фінансовий квартал.

Рисунок 1.4 – Ігровий процес у Atlas Reactor

Серед недоліків гра має непопулярний сеттинг та наявність лише однієї ігрової мапи. Окрім того бізнес-модель проекту передбачала постійні мікротранзакції зі сторони гравців, що в подальшому стало причиною відсутності геймплейних змін та балансних патчів. Для гри такий варіант монетизації став руйнівним, тому в 2018 її сервери було вимкнено і команда розробників була розформована

1.3 Постановка вимог до гри

Задля зацікавленості потенційної аудиторії, кінцевий продукт має позбутися більшості недоліків існуючих представників жанру та додати щось нове. Перш за все, було вирішено створювати гру з використання якісних моделей та текстур з високою роздільною здатність, адже усі представники жанру використовують застарілі графічні техніки.

Щодо них змін – насамперед контент буде розповсюджуватися у повному обсязі для усіх користувачів. По-друге, при створенні матчу гравець зможе не тільки обрати клас персонажу, а й змінити його характеристики за власним бажанням. Це додасть потенціал до тактичних маневрів навіть на

етапі вибору героїв [1]. Наступним покращенням буде збільшення максимальної кількості можливих гравців. Завдяки цьому з’явиться можливість створення унікального для жанру режиму батл-рояль, де головною задачею буде пережити усіх гравців у локації що поступово зменшується.

Ключовим елементом для утримання користувача є різноманіття внутрішньо ігрового контенту. Тому має бути декілька локацій та достатня кількість персонажів. До того ж, важливим нововведенням стане система, де між раундами можна буде додавати персонажів у армію або покращувати вже наявних. Це дасть змогу підтримувати інтенсивність протистоянь та зберегти можливість перевернути хід зустрічі до самого кінця. Також буде доданий функціонал користувацьких матчів із глибоким налаштуванням сесії та ігрового режиму. Окрім того, необхідне впровадження стандартних налаштувань для додавання рейтингового режиму.

За для втілення вказаних вище вимог, необхідно обрати інструменти для розробки, що є безкоштовними, проте пропонують сучасні методи та технології. Важливим є також використання ефективних мережевих рішень, адже ігровий процес має бути задовільним навіть для людей з нестабільним або повільним з’єднанням.

2 ВИБІР ЗАСОБІВ РОЗРОБКИ ТА ПРОЕКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

Розробка програмного забезпечення, зокрема багатокористувацьких ігор, потребує ретельного підходу до вибору інструментів і технологій. Від цього вибору безпосередньо залежить ефективність розробки, масштабованість, продуктивність та якість кінцевого продукту.

У рамках даного проекту необхідно було забезпечити підтримку мережевої взаємодії, синхронізацію станів між клієнтами, зручність реалізації покрокової логіки та інтеграцію з елементами рольової системи. Також важливою вимогою було забезпечення сумісності з персональними комп’ютерами, високий рівень продуктивності та можливість додавання контенту за моделлю DLC.

У цьому розділі буде обґрунтовано вибір основних засобів розробки, зокрема ігрового рушія, мови програмування, мережевих протоколів, середовища розробки та допоміжних інструментів. Критеріями оцінки виступатимуть: функціональність, гнучкість, популярність у спільноті розробників, а також відповідність поставленим завданням проекту.

2.1 Аналіз ігрових рушіїв

Ігровий рушій є центральним компонентом при створенні будь-якої відеогри, оскільки забезпечує функціонування основних підсистем: рендерингу, фізики, анімації, обробки подій тощо [2]. Наразі на ринку представлена велика їх кількість, адже кожен має певні переваги та недоліки, котрі в основному залежать від цілей проекту та його технічної реалізації. Проте є декілька рушіїв, що надають високий рівень абстракції при проектуванні застосунку, завдяки чому розробники мають лише величезний набір інструментів і рішень, що дає можливість втілювати будь-які ідеї.

2.1.1 Unity

Unity – це кросплатформний додаток, що був створений у 2005 році з метою зробити ігрову розробку більш доступною. Спочатку був випущений ексклюзивно для Mac OS X, проте пізніше було додано підтримку Microsoft Windows і веб-браузерів.

Інтерфейс складається із різних вікон, розмір та положення яких можна редагувати на власний розсуд. Основними є оглядач сцени, ресурсів та їх ієрархії, та інспектор поточного об’єкта. Зовнішній вигляд застосунку останньої версії зображено на рисунку 2.1.

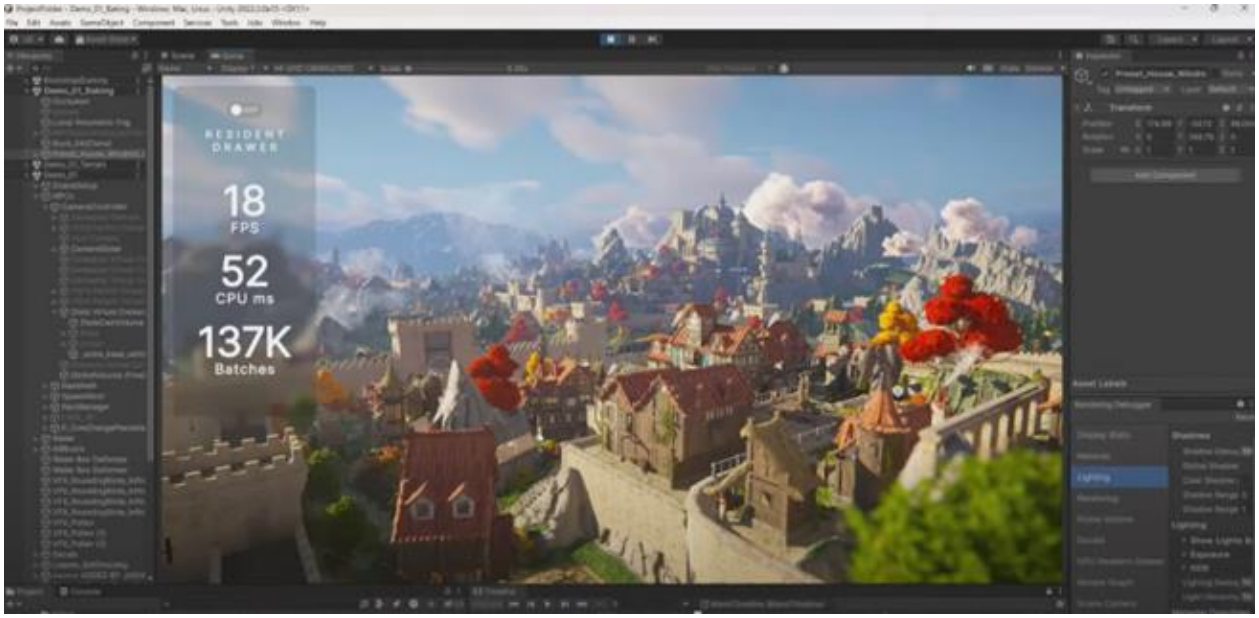


Рисунок 2.1 – Користувацький інтерфейс Unity 6

Мовою програмування, що використовується для розробки у Unity є C# – об’єктно-орієнтована мова з статичною типізацією та підтримкою узагальнених типів даних. Вона використовується для написання скриптів – код, який прикріплений до ігрових об’єктів та визначає їх поведінку. Архітектурно рушій використовує компонентно-орієнтований підхід, що дає змогу легко масштабувати проект і повторно використовувати код [3]. Окрім розширюваності сприяє система Asset Store, за допомогою якої користувачі

можуть легко інтегрувати готові програмні рішення, моделі, ефекти тощо. Щодо мережевої взаємодії, Unity містить реалізацію під назвою Unity Transport Layer (UTP), що пропонує функціонал із реплікації об’єктів, віддалених процедурних викликів та синхронізацію станів. Окрім того наявно підтримка стандартних мережевих протоколів UDP/TCP та WebSockets.

Найвідомішими проектами у жанрі стратегія, що використовували при розробці Unity є RimWorld та Cities: Skylines

Серед переваг даного рушію можна відзначити широкий спектр документації та навчальних ресурсів, що значно полегшують процес створення гри. Завдяки високій популярності, створено велику кількість готових систем та варіантів рішення більшості розповсюджених проблем. Недоліками є нижча продуктивність в порівнянні з іншими рушіями, закритість вихідного коду. Система візуального програмування є сильно обмеженою, що критично для великих команд, де не всі учасники мають навички у програмуванні (графічні спеціалісти, геймдизайнери та інші).

2.1.2 Godot

Godot – це безкоштовний ігровий рушій з відкритим кодом, що підтримує створення 2D та 3D ігор. Рушій стає все більш популярним завдяки своїй легкості, гнучкості та свободі у використанні. Його особливо цінують незалежні розробники, освітні заклади та інді-команди. Godot підтримує використання C++ або власної високорівневої скриптової мови із динамічною типізацією GDScript. Синтаксис нагадує Python, проте оптимізована вона саме під базовану на сценах архітектуру, котра і використовується у рушії [4].

Основним напрямком є саме 2D розробка, адже архітектура для цього написана окремо, а не є простим обмеженням тривимірної, як у інших рушіїв. Саме через це Godot вважається найкращим рушієм для таких проектів.

Серед найпопулярніших проектів, розроблених з використання цього рушія, у жанрі стратегія можна відзначити Lancer Tactics та Dawnfolk. Інтерфейс усього робочого простору застосунку наведено на рисунку 2.2.

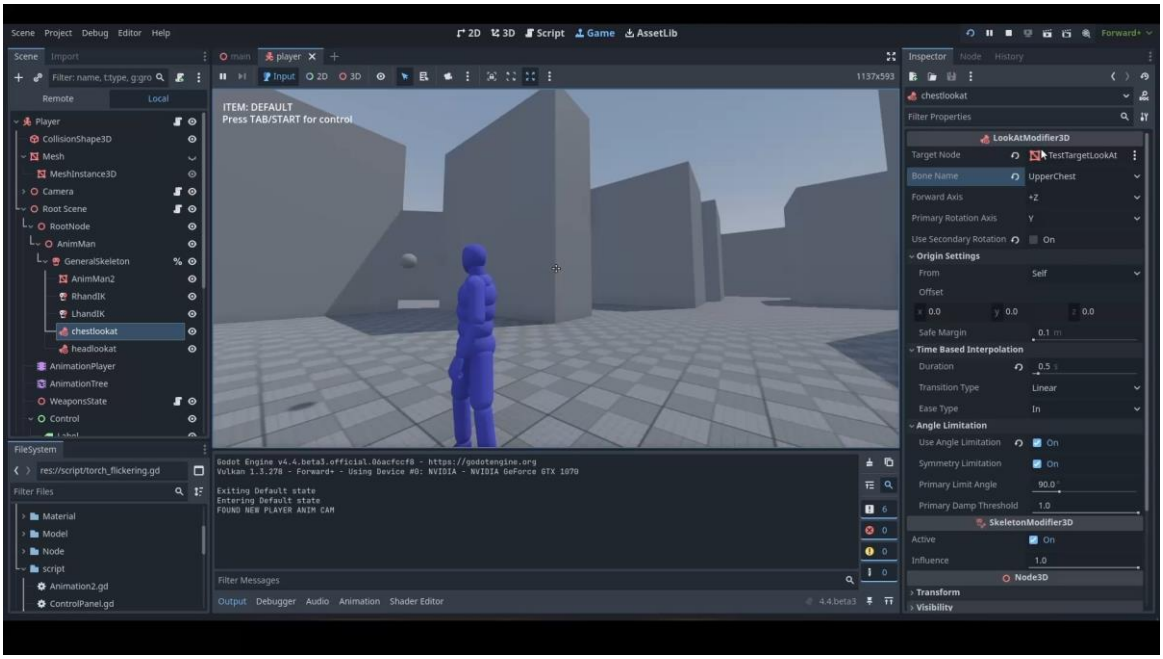


Рисунок 2.2 – Користувацький інтерфейс Godot 4.4

Графічна система побудована на основі OpenGL ES 2.0. Завдяки цьому рушій надає можливість використання таких технологій рендерингу як order-independent transparency , normal mapping та повноекранні постеффекти , такі як FXAA, bloom, DOF, HDR, гама-корекції, distance fog та інші. Для створення власних шейдерів використовується спрощена мова, що є близькою до GLSL.

Безперервними перевагами даного рушію є повна відсутність платних планів, адже розробники платять лише на добровільній основі у вигляді пожертв. Godot є чудовим вибором для двовимірних проектів, проте відповідно до визначених вимог проекту не підходить. Адже 3D-функціональність хоч і активно розвивається, але все ще дуже відстає у графіці та оптимізації в порівнянні з іншими рушіями. Також через невелику популярність кількість навчального матеріалу є малою, так само як і чисельність готових рішень.

2.1.3 Unreal

Unreal – ігровий рушій, що створений та підтримується Epic Games. Першою грою став однойменний шутер, власне для якого і створювався рушій. Відтоді він був використаний десятками компаній для створення сотень нових ігор і вважається стандартом індустрії розробки відеоігор. Зовнішній вигляд користувацького інтерфейсу вказано на рисунку 2.3.

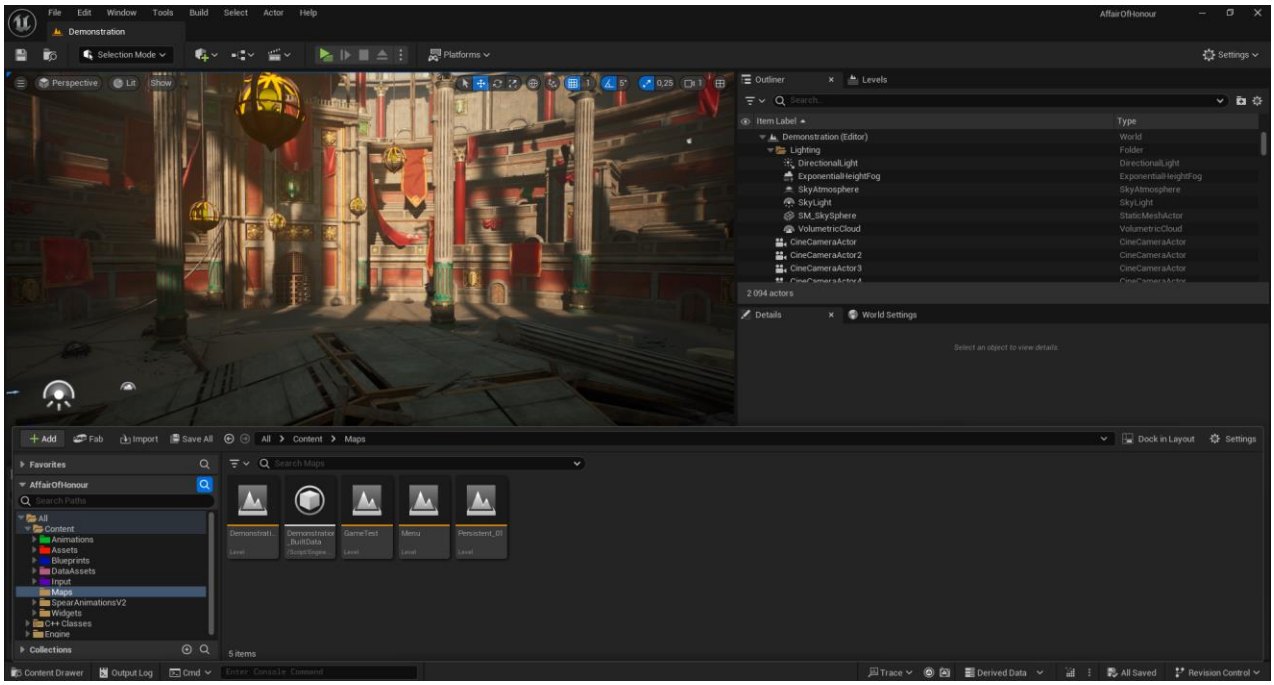


Рисунок 2.3 – Користувацький інтерфейс Unreal 5.4

Для написання коду гри або розширення функціоналу рушія використовується C++. Це високоефективна мова програмування загального призначення, котра використовується для написання потужних клієнтських та серверних програм, драйверів та відеоігор.

Окрім базового синтаксису C++ рушій має власну систему рефлексії. Цей механізм дозволяє в реальному часі отримувати інформацію про класи, їх властивості та методи під час виконання [5]. Це реалізовано шляхом особливих команд препроцесора та макросів, котрі й генерують мета-інформацію для класів. Для додавання змінної чи функції у цю систему, необхідно перед їх визначенням вказувати спеціальний код, котрий буде

повідомляти окремій утиліті Unreal Header Tool про необхідність внесення. Існують такі команди: UCLASS(), USTRUCT(), UENUM(), UPROPERTY(), UFUNCTION(). Окрім того, в кожен із цих макросів можна додати певні параметри, котрі відповідають за необхідність відображення елементу у редакторі, реплікації з сервера на клієнти та інтеграції із системою візуального програмування Blueprints.

Blueprint – це візуальна репрезентація класу, що дозволяє програмувати логіку, використовуючи вузли, які з'єднуються лініями виконання та даних [6]. Окрім того, можливе створення таких вузлів для власних функцій, структур або посилань на об’єкти. Головною умовою їх використання є наслідування від класу UObject та додавання параметру доступу у макрос. Ця система є ключовою перевагою при порівнянні із іншими ігровими рушіями, адже надає можливість людям, котрі не знайомі з програмуванням або синтаксисом C++, створювати інтерактивні, масштабовані та легко розширювані ігрові об'єкти без необхідності дублювання коду чи написання зайвої логіки вручну. Зовнішній вигляд робочого простору системи представлений на рисунку 2.4.

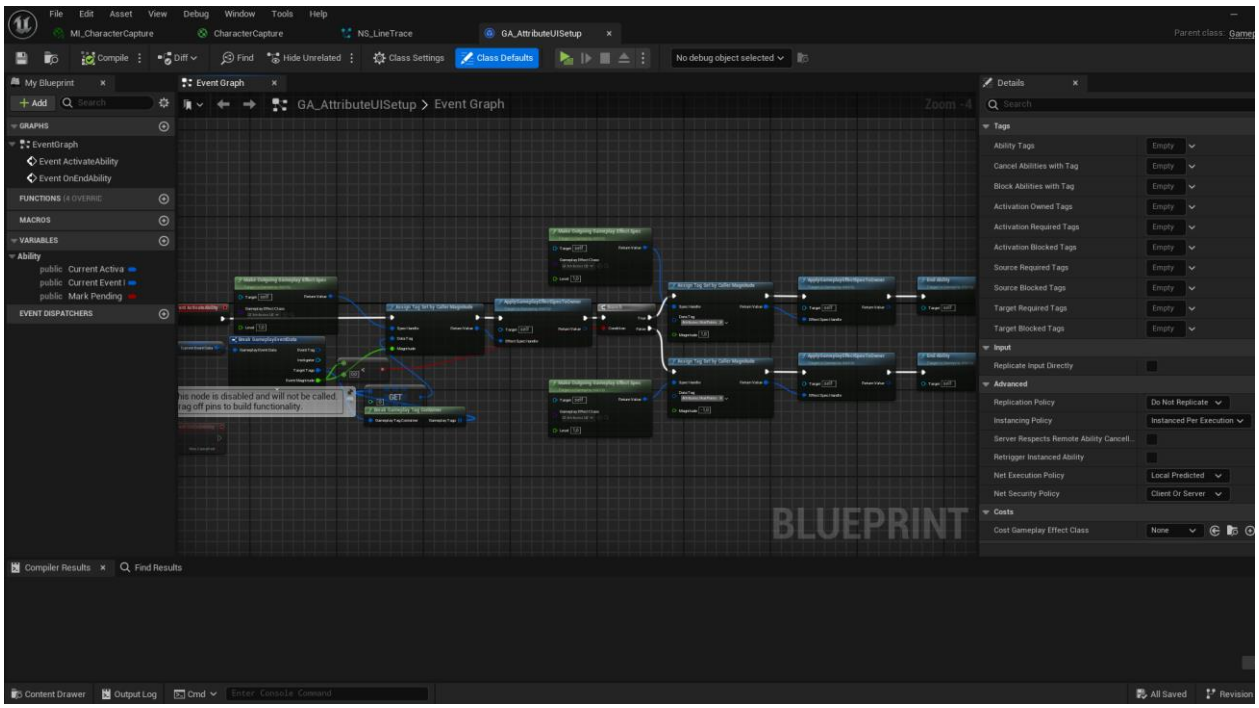


Рисунок 2.4 – Система Blueprints

Окрім цього, фундамент цієї системи використаний у інших вбудованих інструментах, таких як:

- Material Graph, що використовується для написання власних шейдерів (інструкцій для відеокарти). Для оцінки їх ефективності, розробнику показується інформація про кількість операцій перемальовування, обробки, чисельність зразків текстур та інше;

- Niagara, що замінила собою Cascade та створена для надання гнучкого та модульного підхід до створення візуальних ефектів. Завдяки ній реалізована можливість обчислення частинок на відеокарті, що в деяких випадках кратно збільшує продуктивність;

- Meta Sound, що застосовується для програмованого створення звукових ефектів на основі вузлової системи. Це інноваційний інструмент, який перетворює звук на гнучкий і повністю контрольований ресурс позбуває потреби у використані окремих DAW.

Мережевий функціонал реалізовано у вигляді клієнт-серверної архітектури [7]. Кожен об’єкт, що представлений на рівні, має мережевого власника. Їм може бути клієнт, виділений сервер або віддалений сервер. Взаємодія відбувається завдяки технології WebSockets та протоколу передачі даних UDP. Також інтегровано принцип важливості пакетів. Це допомагає розвантажити пропускну смугу мережі, бо неважливі події, наприклад візуальні, не будуть надсилати данні постійно до підтвердження прийому.

Загалом рушій дійсно надає передові рішення для розробки відеоігор. Величезна популярність та активна підтримка команди розробників сприяє появі колосальної кількості навчальних матеріалів та готових рішень. Роялті-модель в Unreal є гнучкою, адже вартість за використання залежить напряму від прибутків. Наприклад усі ігри, що мають життєвий дохід менше мільйону доларів не сплачують жодних внесків у Epic Games. Єдиним недоліком цього рушію є складність розробки, адже всі інструменти спроектовані для максимальної модульності, що збільшує кількість залежностей. Проте враховуючи усі переваги, саме його було обрано для реалізації мого проекту.

2.2 Аналіз інтегрованих середовищ розробки

Розробка сучасних програмних продуктів, зокрема відеоігор, неможлива без використання інтегрованих середовищ розробки, адже вони забезпечують зручний інтерфейс для написання, тестування, налагодження та розгортання програмного коду, а також часто включають засоби для керування проектами, автодоповнення коду, перевірку синтаксису, інтеграцію з системами контролю версій та засобами компіляції. При використанні Unreal Engine варто зазначити, що офіційно підтримується 2 застосунки, а саме Visual Studio та Rider.

2.2.1 Visual Studio

Visual Studio – IDE створене компанією Microsoft у 1997 році. Воно призначене для написання комп’ютерних програм, зокрема веб-сайтів, веб-застосунків, веб-сервісів і мобільних додатків. Visual Studio підтримує розробку з використанням платформ Microsoft, таких як Windows API, Windows Forms, Windows Presentation Foundation, Microsoft Store та Microsoft Silverlight. Середовище дозволяє створювати як керований, так і нативний код.

До складу Visual Studio входить редактор коду з підтримкою IntelliSense – компонента для автоматичного завершення коду, а також засоби для його рефакторингу. Вбудований відлагоджувач (debugger) може працювати як на рівні вихідного коду, так і на рівні машинного коду. Крім того, Visual Studio містить профайлер продуктивності коду, інструменти для створення графічного інтерфейсу користувача, веб-дизайну, проектування класів та моделювання структури баз даних.

Інтеграція з Unreal Engine реалізована наявністю плагінів для обох застосунків. Коли створюється проект в Unreal Engine з увімкненою підтримкою C++, рушій автоматично генерує проектні файли для Visual

Studio. Ці файли включають налаштування рішення, структуру проекту, підключені модулі та залежності. Після цього розробник отримує доступ до інструментів із контекстом рушія. До цього входить автодоповнення до всіх базових елементів API, відлагодження та інформацію про кількість реалізованих Blueprints. Серед недоліків необхідно відзначити: неповне покриття IntelliSense функціоналу наданого Unreal, повільна робота та необхідність вручну генерувати файли у разі змін у рушії. До переваг можна віднести наявність безкоштовного плану для індивідуальних розробників.

2.2.2 Rider

Rider – інтегроване середовище розробки для різних мов програмування (C#, VB.NET, F#, C++, JavaScript, TypeScript, HTML, та ін.) від компанії JetBrains. Перша версія JetBrains Rider з'явилася 3 серпня 2017 року й швидко здобула популярність.

Дизайн середовища орієнтовано на продуктивність праці, швидкий доступ до інструментів, стабільність, а також гнучкість у налаштуванні. Інтерфейс програми зображено на рисунку 2.5.

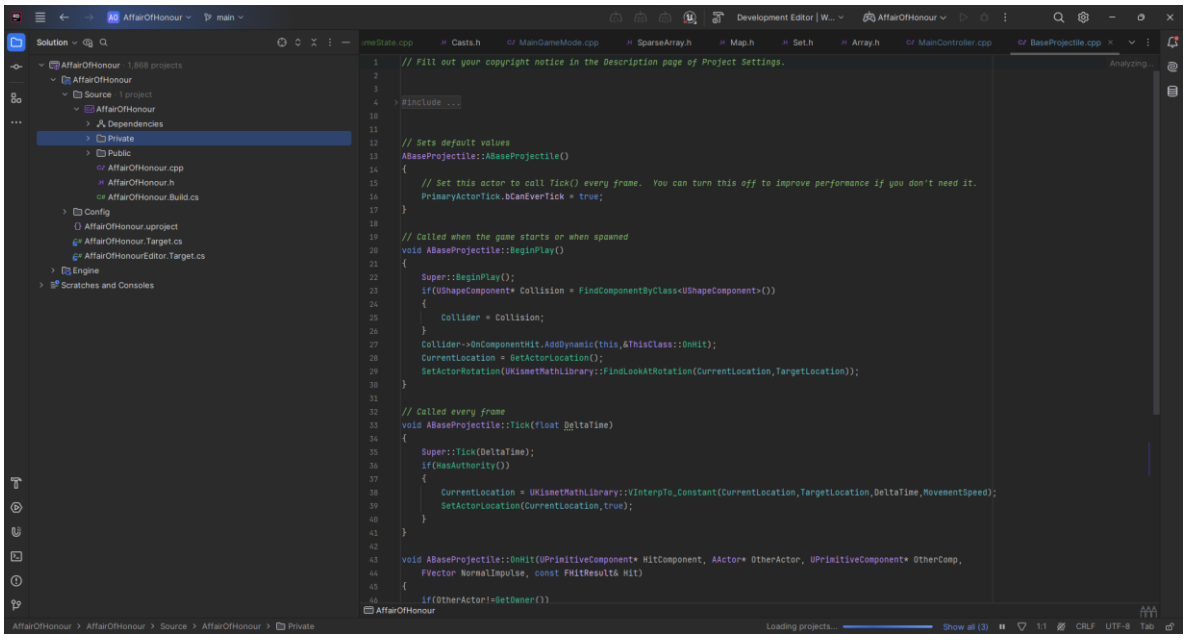


Рисунок 2.5 – Інтерфейс у Rider IDE

Rider оптимізовано для роботи з Unreal Engine [8], адже він повністю підтримує систему рефлексії, плагіни та модулі рушія. Завдяки власному механізму індексації, Rider забезпечує швидке IntelliSense, навігацію по коду, швидкий пошук символів та класів навіть у великих проектах, що є основною перевагою даного продукту над Visual Studion. Окрім того, застосунок має безкоштовний план для студентів вищих навчальних закладів із додатковим функціоналом, наприклад ReSharper – інструмент для рефакторингу, аналізу та оптимізації коду. Саме цей продукт було обрано при написанні проекту.

2.3 Аналіз систем контролю версій

У процесі розробки будь-якого програмного продукту, системи контролю версій є невід'ємною частиною робочого процесу. Вони забезпечують ефективне управління змінами в коді, ресурсах та конфігураційних файлах, дозволяючи відстежувати історію змін, співпрацювати в команді та уникати втрати важливої інформації.

Контроль версій особливо важливий при розробці ігрових застосунків, оскільки такі проекти містять велику кількість різнорідних файлів – вихідний код, 3D-моделі, текстури, звуки, скрипти, конфігурації. Зміни в одному з компонентів може впливати на весь продукт, тому важливо мати змогу повернутися до попереднього стану або вирішити конфлікти, які виникають під час командної роботи. Офіційно Unreal Engine підтримує такі VCS: Perforce, Subversion та Git.

2.3.1 Subversion

Subversion – вільна система керування версіями, що була створена Apache Software Foundation у 2000 році. Вона є централізованою, тобто дані зберігаються у єдиному сховищі. Для оптимізації процесу зберігання нових версій, використовується дельта-компресія – пошук та подальший запис

лише відмінностей, що дозволяє уникати дублювання даних. Можливе локальне та віддалене зберігання. При першому варіанті доступ отримується з використанням клієнту Subversion, а в разі віддаленого сервера: з використанням мережевого протоколу WebDAV. Основним недоліком системи є відсутність повної локальної історії, адже принцип централізованих VCS передбачає постійний зв'язок з сервером. Окрім того, процес розділення стану на гілки є менш гнучким ніж у інших інструментах.

2.3.2 Helix Core

Система контролю версій Helix Core – централізована система керування, що була компанією Perforce Software у 1995. Метою розробки продукту, було створення рішення, що зможе ефективно працювати з кодовими базами. Популярні у той час CVS або RCS були орієнтовні на невеликі текстові файли та не могли забезпечити продуктивність та надійність при роботі із величезними бінарними файлами.

У 2000 роках Perforce був активно інтегрований у сферу розробки відеоігор, де на нього переходили такі гіганти, як Electronic Arts, Ubisoft, Blizzard. Причиною стала можливість швидкої обробки файлів моделей, текстур, анімацій та інших. Наразі ця система контролю версій є стандартом для великих ігрових студії, проте через високу вартість ліцензії та обмежений безкоштовний план, майже не використовується малими командами та інді-розробниками.

2.3.3 Git

Git – це розподілена система контролю версій, створена у 2005 році Лінусом Торвальдсом, автором ядра Linux. Причиною появи продукту став конфлікт між спільнотою розробників та компанією BitKeeper, котра надавала доступ до пропрієтарної VCS за певних умов. Тоді за кілька тижнів

Торвальдсом було створено першу версію система, котра вже тоді мала ключові переваги над конкурентами, а саме наявність повного локального зберігання історії змін, її захист від підробок через контроль SHA-1 хешів.

Окрім цього Git надає швидку та зручну систему гілкування, для паралельної розробки та збереження стабільних версій. Для інтеграції змін наявні інструменти злиття та rebase. Також Git є повністю безкоштовним та з відкритим вихідним кодом. Враховуючи ці переваги саме ця система контролю версій була обрана для використання у проекті.

2.4 Алгоритм реалізації та структура ігрового циклу

Проект належить до жанру багатокористувацьких покрокових стратегій з елементами рольової гри. Основна механіка ґрунтується на змаганні між двома або більше гравцями в умовах симетричного поля бою, де кожен керує командою персонажів, що володіють унікальними характеристиками, вміннями та стилем гри. Структурно програма поділена у спосіб показаний на рисунку 2.6.

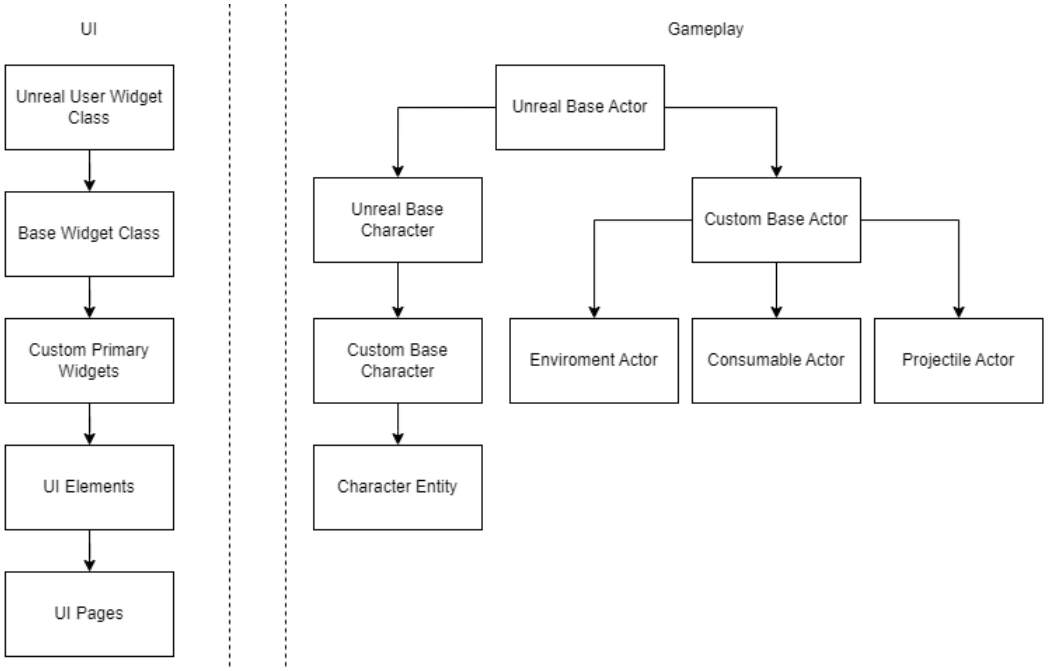
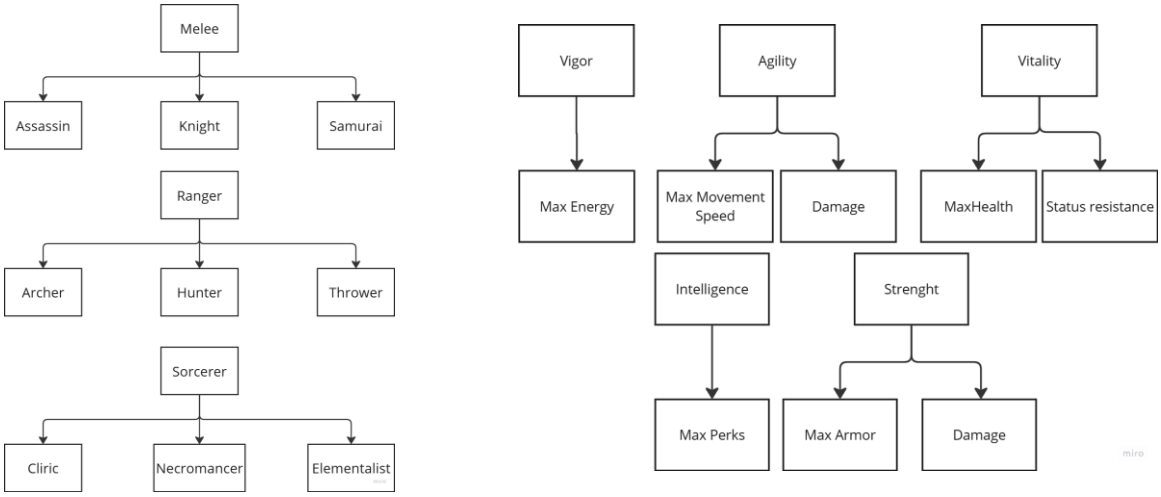


Рисунок 2.6 – Структура проекту

Персонажі мають класи та підкласи. У грі представлено 3 класи: Ranger, Melee та Sorcerer. Вони впливають на ключові атрибути героїв та перелік базових умінь. Кожен з класів має по 3 підкласи. Вони відповідають за вторинні параметри та додають спеціалізовані вміння. Загальну ієрархію зв’язків між цими елементами зображено на рисунку 2.7.



а) б)

Рисунок 2.7 – Схема зв’язків між: а) класами та підкласами; б) атрибутами

Гра починається з створення сесії до якої підключаються гравці. Після початку матчу, система визначає черговість ходів. Під час свого ходу гравець має змогу пересувати персонажів, використовувати здібності або предмети, взаємодіяти з об’єктами на карті та атакувати суперників. Кожна дія споживає певну кількість ресурсів, що обмежує обсяг активностей у межах одного ходу. Окрім того накладається часове обмеження для пришвидшення ігрового процесу.

Ключовим елементом геймплею є стратегічне планування дій, розрахунок ефективності здібностей, а також адаптація до поточної ситуації на полі бою. Переможцем вважається той, хто повністю знешкодить команду супротивника за відведену кількість раундів. Увесь ігровий процес супроводжується обміном даними між клієнтами через сервер, який відповідає за централізовану логіку, перевірку виконуваних дій, зберігання

стану гри та синхронізацію між учасниками матчу. Завдяки покроковому ігровому процесу забезпечується більша стабільність у мережевій взаємодії. Алгоритм роботи зображено на рисунку 2.8.

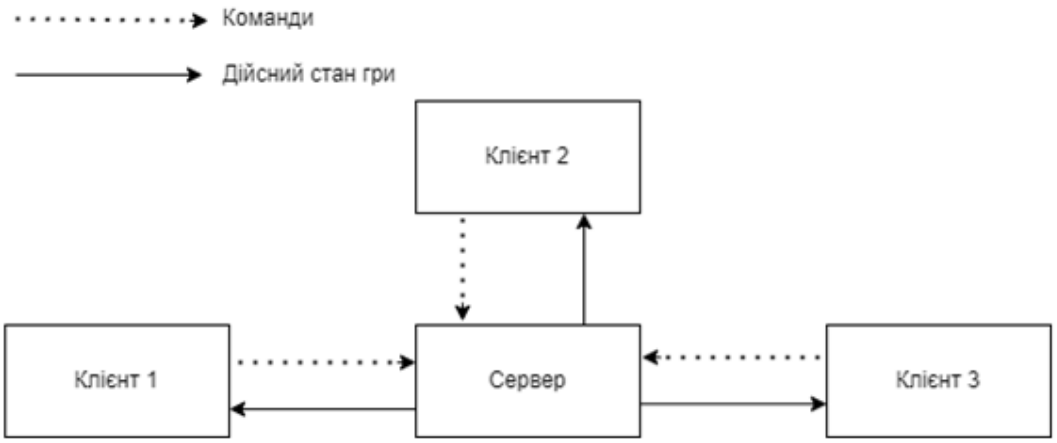


Рисунок 2.8 – Клієнт-серверна архітектура застосунку

Проблемою такого підходу є затримка керування, адже поки сервер не отримає, виконає та відправить результат команди, у гравця нічого не відбудеться. За для уникнення цього було використано систему передбачення. Спочатку створюється ключ – унікальний ідентифікатор, котрий складається з числового значення та представляє стан гри до команди. Потім надсилається запит на виконання дії на сервері та одночасно з тим починається здійснення на клієнті гравця. При отримані дозволу зі сторони сервера та отримані очікуваних даних, клієнт перевіряє їх та в разі необхідності інтерполює для плавного співставлення з коректними. Проте у разі відмови клієнт, відміняє внесені зміни та реплікує правильний стан гри із сервера. Графічний опис роботи системи зображено на рисунку 2.9.

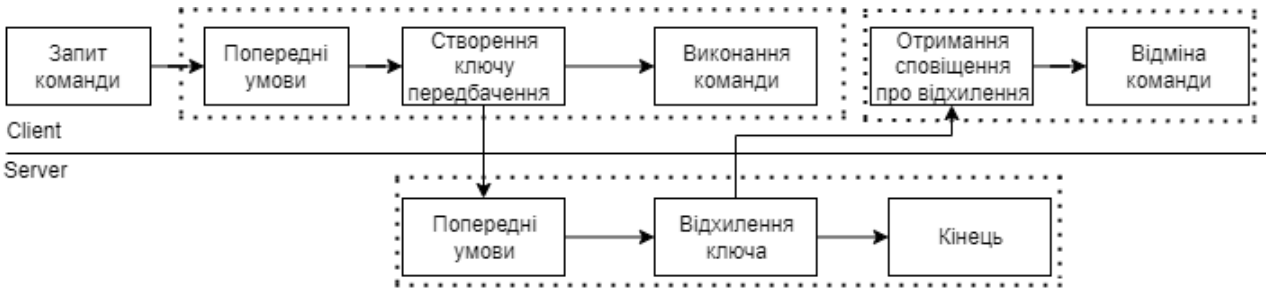


Рисунок 2.9 – Система передбачення

Для реалізації функціоналу сесійних матчів, а також для їх відстеження та зберігання результатів було використано Epic Online Services. Це набір безкоштовних хмарних інструментів і сервісів від компанії Epic Games, що надає повноцінну інфраструктуру для реалізації багатокористувацьких ігор, включаючи механізми автентифікації, збереження прогресу, пошуку матчів, голосового чату та інше. Використання сервісу вимагає реєстрації на порталі розробника та верифікації проекту. Після чого на сторінці порталу буде видано спеціальні ID, котрі треба вказати в конфігураційному файлі гри.

Використовуючи увесь описаний інструментарій та дотримуючись визначених вимог до проекту було спроектовано такий ігровий цикл:

- Гравець запускає виконуваний файл з грою та потрапляє у головне меню;
- Обирає початок онлайн гри або налаштовує її параметри;
- Потім потрапляє у меню, де є можливість створити ігрову сесію, доєднатися до існуючої або запустити рейтингову гру;
- Після приєднання до сесії, гравець очікує інших;
- Коли усі клієнти доєдналися починається гра;
- Гравець має обрати персонажів у кількості, котра залежить від налаштувань матчу. Окрім того надається можливість зміни параметрів для виконавців на розсуд учасника гри. Цей процес обмежений за часом, тому у разі відсутності вибори від гравця, герої будуть обрані випадково із базовими атрибутами;
- Після вибору починається раунд, котрий продовжується до перемоги одного з гравців;
- По завершенню раунду гравці знов починають стадію вибору або персонажу або покращення наявних. Це залежить від налаштувань матчу;
- Кроки 7 та 8 повторюються допоки не буде визначено переможця. Кількість ітерацій залежить від налаштувань матчу (максимум раундів);
- По завершенню гри, кожному з гравців показується результат та вони повертаються у головне меню.

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Користувацький інтерфейс

Сучасні ігрові додатки потребують не лише якісної реалізації ігрового процесу, а й продуманого інтерфейсу, зручного для користувача. UI відіграє важливу роль у забезпеченні зручності та інтуїтивності керування грою. Він включає головне меню, екрани налаштувань, індикатори стану гравця та інші складові, що сприяють комфортному ігровому процесу. Відповідно до описаної ієрархії графічної оболонки було відокремлено такі сутності: базові елементи інтерфейсу, віджети та сторінки. Для ефективної організації коду та чіткого розділення відповідальностей у роботі з інтерфейсом було використано архітектурний шаблон MVC (Model-View-Controller). Цей підхід дозволив чітко відокремити обов’язки. Рівень моделі відповідає за дані, наприклад статистику гравця, налаштування гри тощо. Представлення реалізоване через компоненти інтерфейсу, котрі відображають актуальну інформацію з моделі. Посередником між ними є контролер, котрий отримує дані, оброблює їх та надає для показу. Як результат, модель не знає у який спосіб значення будуть відображені, а рівень представлення не має прямого доступу до даних.

3.1.1 Базові елементи

Цей рівень є найбільш важливим, адже є батьківським для усіх подальших. До нього належать усі примітивні елементи користувацького інтерфейсу, а саме: кнопки, прапорці, лічильники та повзунки. Хоч рушій надає готові рішення для них , вони є неоптимальними через неможливість обробляти результати взаємодії з користувачем у коді та відсутність багаторазовості. Перевизначення їх базових класів дає можливість додати

необхідну функціональність та водночас використовувати їх в інструменті для побудови інтерактивного UI – UMG [9]. Він базується на фреймворку Slate, але надає більш зручний редактор на основі візуальних скриптів, що дозволяє дизайнерам та розробникам працюючи в парі створювати динамічні інтерфейси, програмісти можуть вказувати набір даних, котрі обов'язково мають бути реалізовані у кінцевому елементі.

Наприклад, для створення користувацької кнопки створюється клас UBaseButton, що успадкований від UUserWidget. Ініціалізується конструктор, що викликається під час кожного додавання частини графічного інтерфейсу на екран гравця. Наступним етапом йде створення делегатів – механізм для створення систем, керованих подіями. Вони надають можливість підписки для отримання даних або їх розповсюдження. Синтаксис оголошення наведено у лістингу 3.1.

Лістинг 3.1 – Оголошення делегатів

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnButtonClicked);
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnButtonHovered);
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnButtonUnHovered);
```

Для їх використання, необхідно визначити методи, котрі будуть видавати/одержувати інформацію. Оптимальним місцем для цього є конструктор, адже важливо якомога раніше виконати прив'язування за для уникнення втрачених даних. Окрім того, в ньому необхідно ініціалізувати зміні, котрі впливають на зовнішній вигляд. Для зберігання основних графічних параметрів рушій використовує структуру FButtonStyle. Загальний код конструктору наведено у лістингу 3.2.

Лістинг 3.2 – Конструктор кнопки

```
void UBaseButton::NativeConstruct(){
Super::NativeConstruct();
Clicked = Button->GetStyle().Pressed;
Button->OnClicked.AddDynamic(this,&HandleOnClicked);
Button->OnHovered.AddDynamic(this,&HandleOnHovered);
Button>OnUnhovered.AddDynamic(this,&HandleOnUnhovered);}
```

Простір імен Super означає те, що перед кастомною логікою буде викликано код із батьківського класу. Цей функціонал є дуже корисним, хоч і опціональним, адже допомагає уникнути постійного дублювання коду,

Після написання коду кнопки створюється дочірній blueprint клас для графічного налаштування. У ньому визначаються структура, з яких буде складатися кінцева сутність. Зовнішній вигляд кнопки та її ієрархії наведено на рисунку 3.1

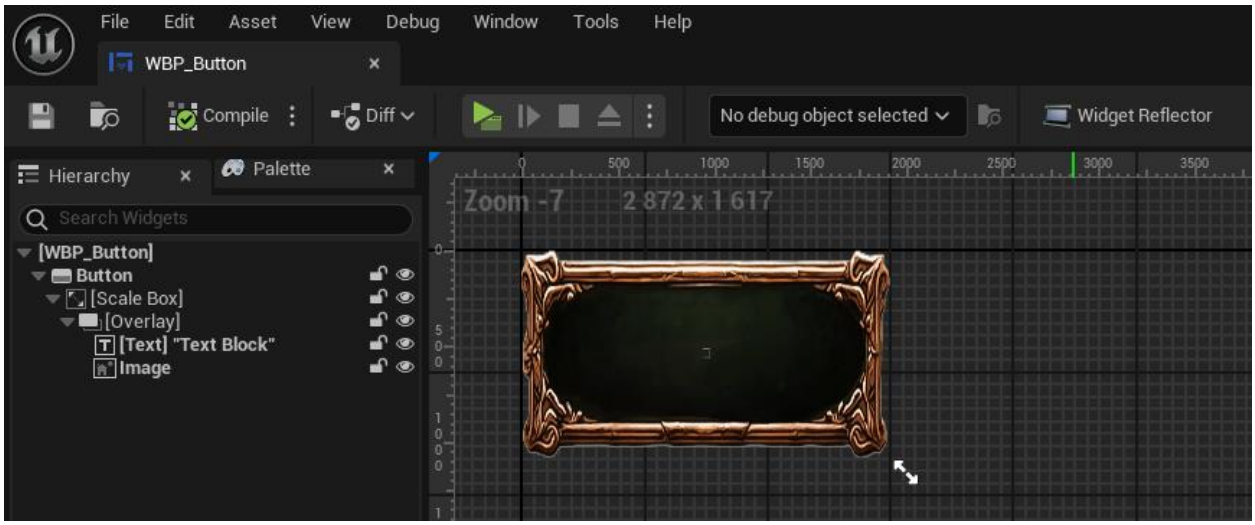


Рисунок 3.1 – Зовнішній вигляд кнопки та її ієрархія

Інші примітивні користувацькі компоненти інтерфейсу створювалися подібним чином. Надалі вони використовуються для створення складових наступного рівня – віджетів.

3.1.2 Віджети

Даний тип графічних елементів містить найбільшу кількість інформації, котра надається гравцю. Усі вони успадковані від класу UBaseWidget, котрий містить каркас для побудови зв’язку із контролером. Подальша функціональність залежить від призначення, Наприклад створено елемент для відображення здібності гравця. При створенні йому надається вказівник на єдиний об’єкт типу UAbilityWidgetController, котрий містить набір методів для визначення даних відповідно до представленого уміння.

Окрім представлення власних даних, він слугує транслятором від контролера до віджету-підказки, адже є його власником. Ініціалізація даних у графічному елементі описана у лістингу 3.3.

Лістинг 3.3 – Ініціалізація даних у віджеті уміння

```
void UAbilityWidget::InitAbilityWidget(const FAbilityInfo& Info)
{Button->SetImageFromAtlas(Info.Image);
CustomToolTipWidget=Cast<UAbilityToolTip>(CreateWidget
(GetOwningPlayer(),ToolTipWidgetClass));
CustomToolTipWidget->Description->SetText(Info.Description);
CustomToolTipWidget->Name->SetText(Info.Name);
CustomToolTipWidget->BaseAmountOfTargets-
>SetText(FText::FromString(FString::FromInt(Info.Targets)));
CustomToolTipWidget->BaseRange->SetText(FText::FromString
(FString(FString::SanitizeFloat(Info.Range/100)+"m.")));
CustomToolTipWidget->BaseAOERange->SetText(FText::FromString
(FString(FString::SanitizeFloat(Info.AOERange/100)+"m.")));
FSlateBrush Image = FSlateBrush();
Image.SetResourceObject(Info.Image);
CustomToolTipWidget->Image->SetBrush(Image);
Button->SetToolTip(CustomToolTipWidget);}
```

Заповнення цих даних відбувається із використанням спеціальних контейнерів UDataAsset. Вони надають можливість централізованого та швидкого змінення параметрів без прямого втручання у код. Приклад заповнення даних для побудови елементів інтерфесу наведено на рисунку 3.2.

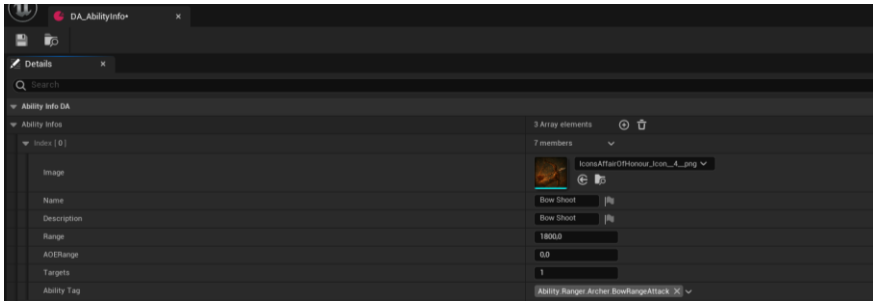


Рисунок 3.2 – Централізований підхід при побудові елементів інтерфесу

У деяких випадках складові користувацького інтерфейсу виконують велику кількість логіки. В такому випадку стає доцільною реалізація певних частин у Event Graph задля розвантаження коду. Наприклад у елементі для відображення та редагування атрибутів гравця, важкі функції пошуку та верифікації були реалізовані на C++ [10], в той час як графічне оновлення реалізовано у Blueprints. Нодова послідовність для збільшення n-ої ознаки гравця зображено на рисунку 3.3.

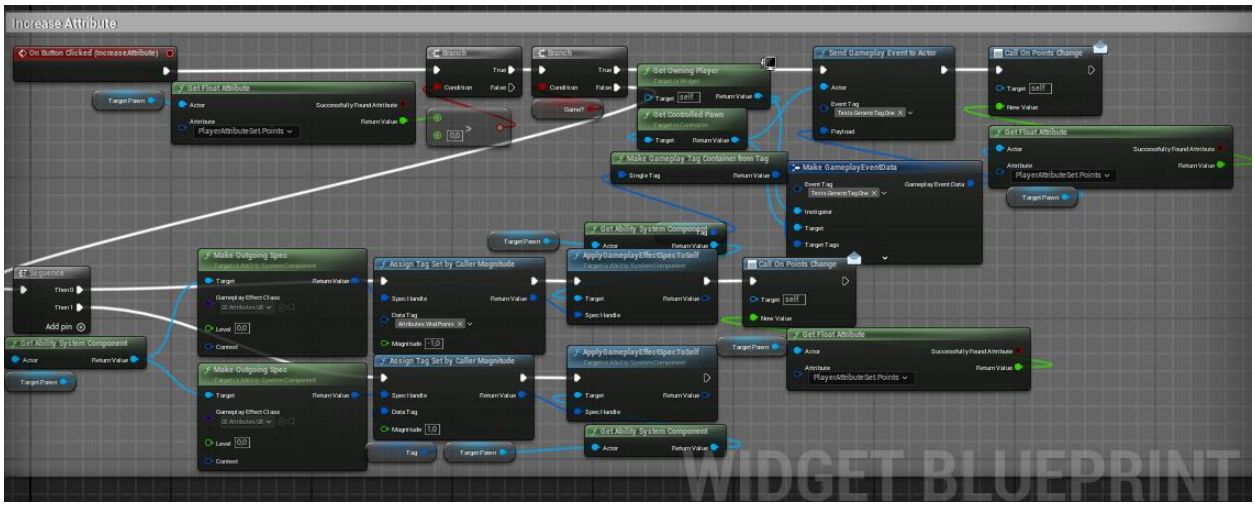


Рисунок 3.3 – Нодова послідовність для зменшення n-ої ознаки гравця

Схожий підхід було використано для інших віджетів, наприклад для вибору мапи, зображення стану всіх персонажів та створення сесій. В подальшому усі ці компоненти використовувалися для створення сторінок

3.1.3 Сторінки

Кінцевий елемент ієрархій в розробленому UI, котрий додається до екрану гравця. До нього входить HUD, головне меню, налаштування та вікно вибору персонажу. Процес додавання відбувається через виділений клас UGameplayHUD, що є власністю ігрового контролеру і надає основну інформацію з нього до сторінок. При їх побудові віджети поєднуються спеціальними складовими, серед яких переважною більшістю є панелі: canvas panel, horizontal&vertical boxes, overlay. Вони обробляють вирівнювання

елементів під час масштабування, обмежують їх розмір відповідно до співвідношення сторін та мають вбудовану підтримку різних способів вводу: клавіатура, миша, геймпад та тач. Зовнішній вигляд основних сторінок наведено на рисунках 3.4 – 3.6.

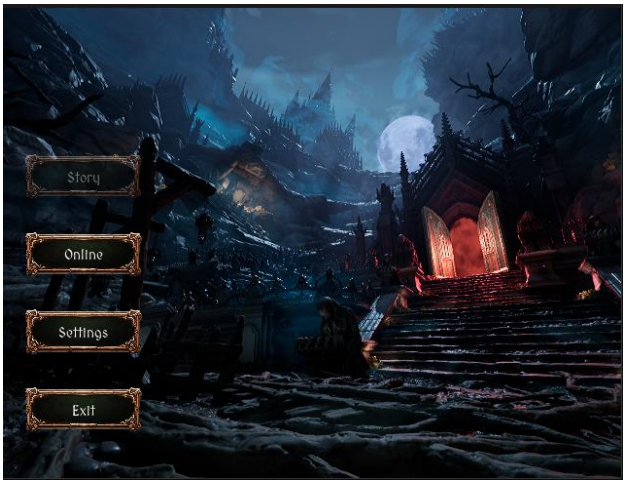


Рисунок 3.4 – Головне меню

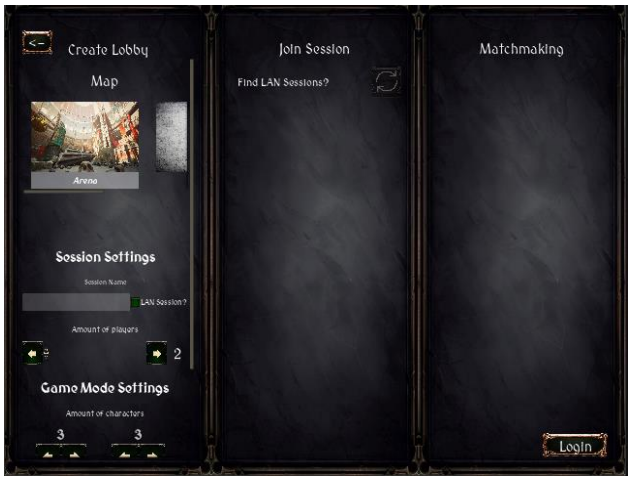


Рисунок 3.5 – Меню онлайн режимів гри

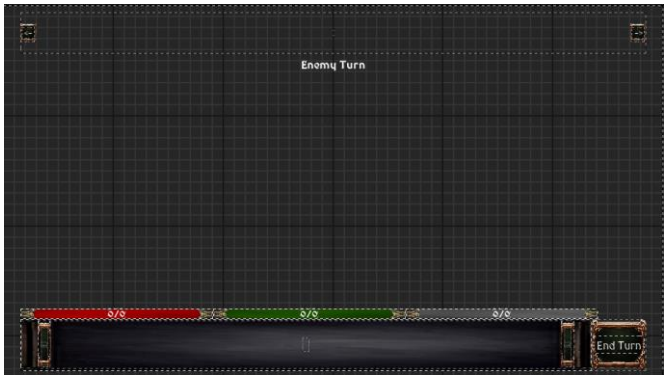


Рисунок 3.6 – HUD

3.2 Ігрова логіка

3.2.1 Ігровий контролер

APlayerController є базовим класом у рушії Unreal Engine, який забезпечує управління ігровим процесом від імені гравця. Він відіграє ключову роль у взаємодії між користувачем та ігровим світом. Основне призначення цього класу полягає в обробці введення з пристроїв користувача та за необхідності передачі команд ігровому персонажу. У проекті на його основі було створено клас AMainController. У ньому окрім перевизначених віртуальних методів додано спеціалізовані функції. Однією з таких є реакція на натискання гравцем лівої клавіші миші. Річ у тім, що сама по собі ця команда є занадто узагальненою, тому використовується MappingContext. Він дозволяє кваліфікувати умови за яких було виконано команду та надає необхідний контекст. Реалізація цієї функції наведено у лістингу 3.4.

Лістинг 3.4 – Функція обробки натискання на ліву клавішу миші

```
void AMainController::TargetSelected(const FInputActionValue&
value)
{if (Cast<IAbilitySystemInterface>(GetPawn()))
   {UAbilitySystemComponent* ASC = Cast<IAbilitySystemInterface>
   (GetPawn())->GetAbilitySystemComponent();
   FGameplayTag RequiredTag = UGameplayTagsManager::Get().
   RequestGameplayTag(FName(TEXT("Ability")));
   FGameplayTag TargetTag = UGameplayTagsManager::Get().
   RequestGameplayTag(FName(TEXT("Ability.Input.Confirm")));
   if (ASC)
    {if (ASC->HasMatchingGameplayTag(FGameplayTag(RequiredTag)))
     {const FGameplayEventData& data = FGameplayEventData();
      ASC->HandleGameplayEvent(TargetTag, &data);}
    else
    {TArray<TEnumAsByte<EObjectTypeQuery>> TypesArray;
     TypesArray.Add(ConvertToObjectType(ECC_Pawn));
     FHitResult res;
     GetHitResultUnderCursorForObjects(TypesArray, false, res);
     if (res.bBlockingHit)
      ASC->TryActivateAbilitiesByTag(
      FGameplayTagContainer(UGameplayTagsManager::Get().
      RequestGameplayTag(FName(TEXT("Movement")))), true);}
}}}}
```

Метод GetHitResultUnderCursorForObjects приймає типи об’єктів для трасування (наприклад, лише персонажі або тільки статичні об’єкти) та повертає структуру FHitResult, яка містить повну інформацію про зіткнення: координати точки дотику, нормаль поверхні, матеріал, посилання на об’єкт тощо. Функція TryActivateAbilitiesByTag намагається активувати здібність у персонажа, котрий володіє системою умінь та керується цим контролером, використовуючи ідентифікатор.

Окрім того, створений контролер відповідальний за виклик функції пов'язаних із ходами та вибором персонажів або покращень. Саме тут використовується чітке розмежування у функціоналі, адже у контексті мультиплеєру ігровий контролер існує як на стороні клієнта, так і на сервері, тому має бути описано різні варіанти функції. Код серверної реалізації початку ходу наведено у лістингу 3.5

Лістинг 3.5 – Серверна версія функції початку ходу.

```
void
AMainController::Server_StartTurn_Implementation(AAOH_BaseCharac
ter* Char)
{
ServerTimerValue =
Cast<AMainGameMode>(UGameplayStatics::GetGameMode(GetWorld()))-
>GetTurnTime();
FTimerDelegate Delegate = FTimerDelegate();
Delegate.BindUObject(this, &ThisClass::ServerTurnTimer);
GetWorld()->GetTimerManager().SetTimer(ServerTimerHandle,
Delegate, 1.0f, true);
Char->OnTurnStarted();
Client_StartTurn();
TimerSync(ServerTimerValue);
}
```

У цьому методі контролер отримує доступ до значень класу AMainGameMode, адже він існує тільки на сервері, що унеможливлює маніпуляції клієнтів із ігровими правилами. Після виконання необхідної логіки авторитарна версія надсилає RPC до гравця з наданням дозволу на виконання локальної варіації методу. У ній описується графічні особливості, такі як оновлення значення таймеру та відправка інструкцій у HUD.

Щодо стадії вибору, перед її початком кожному гравцю необхідно розрахувати порядок, за яким будуть надаватися варіанти. На це впливає обрана конфігурація при створенні ігрової сесії, а саме тип розподілення, кількість раундів та персонажів. Опис цього методу наведено у лістингу 3.6

Лістинг 3.6 – Функція розрахунку розподілення вибору

```
void AMainController::CalculateSelectionForRounds()
{
AMainGameMode* GameMode = Cast<AMainGameMode>
(UGameplayStatics::GetGameMode(GetWorld()));
int TotalRounds = GameMode->GetAmountOfRounds();
int TotalCharacters = GameMode-
>GetAmountOfCharacterForController(this);
switch (GameMode->GetSelectionType())
{
  case Instant:SelectionForRound.Add(1,FSelectionData
  (ESelectorType::Character,TotalCharacters));
  for (int i = 2; i <= TotalRounds; ++i)
  {
    SelectionForRound.Add(i,FSelectionData
    (ESelectorType::PowerUp, 1));
  }
  break;
  case Progressive:
    int BaseCharacters = TotalCharacters / TotalRounds;
    int Remainder = TotalCharacters % TotalRounds;
    for (int Round = 1; Round <= TotalRounds; Round++)
    {
       FSelectionData SelectionData;
       int CharactersInRound = BaseCharacters;
       if (Round <= Remainder)
         CharactersInRound++;
          if (CharactersInRound > 0){
            SelectionData.Type = ESelectorType::Character;
            SelectionData.Amount = CharactersInRound;}
          else{
             SelectionData.Type = ESelectorType::PowerUp;
             SelectionData.Amount = 1;}
       SelectionForRound.Add(Round, SelectionData);
    }
  break;}}
```

Ця функція отримує посилання на об’єкт класу AMainGameMode, який містить усі правила поточної гри, включаючи інформацію, необхідну для побудови послідовності вибору. Далі, в залежності від встановленого типу

селекції (Instant або Progressive), виконується розрахунок на кожен з раундів. У випадку миттєвого типу гравцеві дозволено обрати всіх персонажів у першому раунді, тоді як у наступних раундах пропонується вибір додаткових бонусів або підсилень. У разі прогресивного розподілу, загальна кількість персонажів ділиться рівномірно між усіма раундами, і залишок, якщо такий є, розподіляється по черзі на перші раунди. У кожному із варіантів, у внутрішню структуру SelectionForRound додається об'єкт типу FSelectionData, який вказує, який саме тип об'єкта (персонаж або підсилення) і в якій кількості може бути обраний на конкретному етапі гри.

3.2.2 Ігровий персонаж

ACharacter – це один з ключових класів в рушії, який використовується для створення ігрових персонажів. Його головна мета – надати основу для створення персонажів з можливістю ходити, стрибати, взаємодіяти з навколишнім середовищем, а також підтримкою анімацій та фізики. У кастомному класі AAOH_BaseCharacter було перевизначено деякий базовий функціонал та інтегровано систему здібностей (GAS), додано управління камерою та реалізовано бойовий інтерфейс

При створенні об’єкту викликається конструктор, котрий відповідає за створення та ініціалізацію всіх компонентів, що входять до складу персонажа. Серед них – компоненти камери, зброї для обох рук і система анімаційного викривлення руху. Також створюється та налаштовується ACS і його складові.

Після чого у функції для налаштування компонента введення, додаються зв’язки між командами та відповідним їм методам. Окрім того встановлюється контекст, який містить інформацію для команд.

Переміщення камери реалізовано у двовимірному просторі з автоматичною корекцією висоти. Додатково можливе постійне переміщення за героєм. Реалізація наведена у лістингу 3.7.

Лістинг 3.7 – Переміщення камери

```
void AAOH_BaseCharacter::CameraMove(const FInputActionValue&
val)
{SpringArm->AddRelativeLocation(
UKismetMathLibrary::GetForwardVector(FRotator
(0.0f, SpringArm->GetRelativeRotation().Yaw, 0.0f)) * val.Get<
FVector2D>().Y * CameraMovementSpeed);
SpringArm->AddRelativeLocation(
UKismetMathLibrary::GetRightVector(FRotator(0.0f, SpringArm-
>GetRelativeRotation().Yaw,0.0f)) * val.Get<FVector2D>().X *
CameraMovementSpeed);
TArray<FHitResult> ResMax;
TArray<FHitResult> ResMin;
TArray<AActor*> Actors;
FVector Center = SpringArm->GetComponentLocation();
UKismetSystemLibrary::CapsuleTraceMulti(this, FVector(Center.X,
Center.Y, Center.Z),FVector(Center.X, Center.Y, Center.Z - 500),
20, 20,UEngineTypes::ConvertToTraceType(ECC_Visibility), false,
Actors, EDrawDebugTrace::None, ResMax, true);
FHitResult MaxDistanceHit = FHitResult();
for (auto Element : ResMax)
{if (MaxDistanceHit.Distance < Element.Distance)
   MaxDistanceHit = Element;}
if (MaxDistanceHit.bBlockingHit)
{if (MaxDistanceHit.Distance < SpringArmMinHeightThreshold)
   SpringArm->AddRelativeLocation(
FVector(0, 0, FMath::RandRange(SpringArmMinHeightThreshold, 2 *
SpringArmMinHeightThreshold)));
if (MaxDistanceHit.Distance > SpringArmMaxHeightThreshold)
   SpringArm->AddRelativeLocation(FVector(0, 0,
   -FMath::RandRange(SpringArmMaxHeightThreshold / 2,
SpringArmMaxHeightThreshold - SpringArmMinHeightThreshold)));}
CachedSpringArmPos = SpringArm->GetComponentLocation();}
```

Адаптивна корекція для висоти реалізована шляхом оцінки відстані камери до землі відносно порогових значень. Це досягається використанням капсульного трасування. Через те, що цей компонент є локальним для кожного гравця його логіка не містить мережевих оптимізацій.

В свою чергу переміщення героя реалізовано більш комплексно. Перш за все варто зазначити, що воно відбувається з використанням уміння [11], котре отримує інформацію, в яке місце у тривимірному просторі вказує гравець. Через це клієнту необхідно надіслати ці дані до сервера. За для цього створюється спеціальне завдання UTargetDataUnderMouse, котре очікує

команди від клієнта та формує зміст структури FGameplayAbilityTargetData_SingleTargetHit, котра окрім локації містить предиктивний ключ, специфікацію вміння та іншу контекстну інформацію. Завдяки мережевій політиці вміння на локальне передбачення, на екрані гравця одразу починається переміщення. Після отримання даних сервером, він також виконує переміщення та надсилає клієнту правильну позицію. У разі задовільної якості мережевого з'єднання, розбіжність буде мінімальною та коректування буде виконано непомітно. Якщо наявні проблеми у роботі інтернету на клієнті, кінцеве місце призначення все одно буде досягнуто, проте з постійними перериваннями та телепортаціями. Задля пом'якшення цього процесу використовується експоненційна інтерполяція. Повний код уміння переміщення із пошуком шляху наведено у додатку Б.

Інші уміння глобально поділяються на ті, що завдають шкоду та ті, що застосовують ефект . Перші містять параметри шкоди, а саме значення та тип, в той час як другі опис впливу. Якщо уміння виконується на відстань, то використовується той самий підхід визначення обраної позиції клієнтом разом із додаванням спеціальної сутності для прицілювання AGameplayAbilityTargetActor.

Цей актор містить графічні елементи для відображення радіусу вміння або проекції запущенного снаряду. Окрім того він перевіряє дотримання обмежень з кількості цілей та дальності. Код перевірки надано у лістингу 3.8

Лістинг 3.8 – Підтвердження виконання умов актором прицілювання

```
void ALongRangeTargetingActor::Confirmed()
{
if ((LastCorrectHit.Location - GetActorLocation()).Length() <=
Range)
   {Targets.Push(LastCorrectHit);
   AGameplayAbilityWorldReticle* SpawnedReticle =
   GetWorld()->SpawnActor<AGameplayAbilityWorldReticle>(
   ReticleClass, LastCorrectHit.Location, FRotator());
   Reticles.Push(SpawnedReticle);
   if (Targets.Num() >= TargetsCount)
      ConfirmTargeting();}}
}
```

3.2.3 Інстанція гри

Це базова сутність у рушії, котра використовується для зберігання даних та логіки, що повинна бути доступною протягом усього життєвого циклу гри, включно з переходами між рівнями. На відміну від класів GameMode, GameState або PlayerController, які можуть бути знищені під час зміни рівня, об’єкт класу UGameInstance створюється один раз при запуску гри і зберігається до її завершення. Окрім того, він є точкою входу для створення серверу або клієнтського підключення, тому є оптимальним місцем для опису авторизації та сессій. Релізацію авторизації гравця з використання Epic Online Services наведено у лістингу 3.9.

Лістинг 3.9 – Авторизація гравця

```
void UMainGameInstance::LoginEOS()
{
IOnlineSubsystem* SubsystemPtr =
Online::GetSubsystem(GetWorld());
if (SubsystemPtr)
   {IOnlineIdentityPtr IdentityPtr =
   SubsystemPtr->GetIdentityInterface();
   if (IdentityPtr)
     {FOnlineAccountCredentials AccountCredentials;
      AccountCredentials.Id = "";
      AccountCredentials.Token = "";
      AccountCredentials.Type = AutoLoginType;
      IdentityPtr->OnLoginCompleteDelegates->AddUObject
      (this,&ThisClass::LoginEOSResult);
      IdentityPtr->Login(0,AccountCredentials);}}
}
```

На початку функції викликається метод, який надає доступ до активної реалізації підсистеми онлайн-сервісів, прив’язаної до поточного світу. Після перевірки від неї отримується інтерфейс ідентифікації користувачів та ініціалізується структура FOnlineAccountCredentials, яка використовується для передачі облікових даних до EOS. Далі до делегата OnLoginCompleteDelegates інтерфейсу ідентифікації додається метод LoginEOSResult поточного об’єкта. Цей метод буде викликано автоматично

після завершення процесу входу, незалежно від його результату. Нарешті, викликається метод Login з параметрами: LocalUserNum, що дорівнює нулю (вказує на основного гравця), та сформовані облікові дані.

Після успішної авторизації, клієнту надається можливість формувати та шукати матчі. Створення починається з отримання налаштувань ігрового режиму та сесії. До них входять: назва, мапа, кількість гравців, розмір команди для кожного клієнта та інші просунуті налаштування. Потім на основі вказаних даних створюється об’єкт конфігурації FOnlineSessionSettings , який передається у метод мережевого інтерфейсу разом із індексом користувача на ім’ям.

Метод пошуку ініціює процес виявлення доступних сесій мультиплеєрної гри, використовуючи функціональність, надану інтерфейсом IOnlineSession відповідної підсистеми онлайн-сервісів. Спочатку очищуються всі параметри пошуку за для уникнення їх дублювання у результуючому віджеті, а також встановлюється обмеження на кількість можливих результатів. Потім ініціюється асинхронний запит на пошук сесій для гравця з індексом 0. Пошукові параметри передаються як спільне посилання до відповідного об’єкта SessionSearch, котрий на кожну сутність створює UI елемент, що зображує основні параметри. Код описано у лістингу 3.10.

Лістинг 3.10 – Пошук сесії

```
void UMainGameInstance::FindOnlineSession()
{if (IOnlineSubsystem* SubsystemPtr = Online::
GetSubsystem(GetWorld()))
  {IOnlineSessionPtr Session =
SubsystemPtr->GetSessionInterface();
  if (Session)
    {SessionSearch = MakeShareable(new FOnlineSessionSearch());
     SessionSearch->QuerySettings.SearchParams.Empty();
     SessionSearch->MaxSearchResults = 100;
     Session->OnFindSessionsCompleteDelegates.AddUObject
     (this,&ThisClass::OnFindSessionCompleted);
     Session->FindSessions(0,SessionSearch.ToSharedRef());}}
}
```

4 ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

Тестування ігрового застосунку охоплювало повний цикл перевірки функціональності мережевої взаємодії, включаючи як локальну симуляцію багатокористувацького середовища, так і перевірку у реальних умовах з’єднання через Інтернет. Одним із ключових завдань тестування стало забезпечення стабільності сесій, збереження цілісності даних між клієнтом і сервером, а також коректне відображення станів гри на всіх підключених клієнтах.

На першому етапі тестування проводилося за допомогою запуску кількох копій гри на одному комп’ютері з використанням вбудованих інструментів для симуляції мережі. До них відносяться штучна затримка між пристроями, втрата певної кількості пакетів та вибір цілі для емуляції. Такий підхід дозволив перевірити логіку створення сесії, обробку запитів клієнтів, протікання ігрового процесу та відмово-стійкість написаного продукту.

На другому етапі здійснювалося тестування в умовах реальної мережі. Сервер запускався на окремому фізичному пристрої, підключеному до мережі Інтернет, тоді як клієнтські екземпляри запускались з інших пристроїв, розташованих у різних географічних локаціях. Підключення до сесії здійснювалося через використання Epic Online Services (EOS), що дозволило перевірити справжній онлайн-режим, з урахуванням мережевих затримок, авторизації гравців та з’єднання через зовнішні IP-адреси.

Тестування відбувалося на двох типах пристроїв. Серверна частина запускалася на стаціонарному персональному комп’ютері з такими технічними характеристиками: процесор AMD Ryzen 5 5500 (6 ядер, 12 потоків, 3.8–4.6 ГГц), 32 ГБ оперативної пам’яті DDR4, SSD-накопичувач типу NVMe об’ємом 512 ГБ та відеокарта AMD RADEON RX6750XT. Операційна система – Windows 11 x64. Даний ПК був підключений до стабільного дротового інтернет-з'єднання зі швидкістю 300 Мбіт/с

Клієнтський екземпляр гри запускався на ноутбуці з такими характеристиками: процесор Intel Core i7-11800H, 16 ГБ оперативної пам’яті, SSD 512 ГБ, NVIDIA GeForce RTX 3050m, Windows 10 Home x64. Підключення здійснювалося через Wi-Fi зі швидкістю до 50 Мбіт/с.

Було виявлено, що передача стану між сервером і клієнтами, включаючи переміщення, взаємодію з оточенням та обробку ігрових подій, відбувається узгоджено. Підтверджено, що у разі виходу хоста з гри сесія припиняється, а всі клієнти коректно отримують повідомлення про втрату зв’язку.

Загалом, результати тестування вказують на працездатність усіх критичних механізмів мережевої гри. Перевірка в умовах реального інтернет-з’єднання дала змогу підтвердити, що застосунок готовий до повноцінної багатокористувацької експлуатації. Надійність підключення, стабільність сесій, коректна взаємодія з EOS та відсутність критичних затримок при обміні даними свідчать про готовність гри до використання в онлайн-середовищі.

ВИСНОВКИ

У ході виконання атестаційної роботи було проведено повноцінний цикл аналізу, проектування та технічної підготовки до розробки PvP-гри з покроковим геймплеєм та клієнт-серверною архітектурою. Розгляд предметної області дозволив визначити основні жанрові особливості подібних застосунків, а також дослідити найуспішніші приклади реалізацій з подібною структурою, зокрема XCOM 2, Dofus, Divinity: Original Sin 2 та Atlas Reactor.

Проведений аналіз показав, що реалізація багатокористувацьких стратегій із мережевим PvP-компонентом вимагає застосування стабільної інфраструктури з чітким розмежуванням ролей клієнта і сервера, забезпечення синхронізації стану гри та обробки дій гравців у режимі реального часу або з невеликою затримкою.

У процесі дослідження програмних інструментів було розглянуто провідні ігрові рушії (Unity, Godot, Unreal Engine), інтегровані середовища розробки (Visual Studio, Rider), а також системи контролю версій (Git, Subversion, Perforce). Кожен із цих інструментів оцінювався з точки зору зручності для реалізації PvP-стратегії. В результаті, як найбільш оптимальний для проєкту, було обрано рушій Unreal Engine завдяки його потужному мережевому функціоналу, візуальному програмуванню (Blueprints), підтримці модульної архітектури та інтеграції з Epic Online Services.

Реалізація гри включала створення власного класу персонажа, який розширює стандартний функціонал для керування камерою, рухом та взаємодії з ігровими здібностями. У грі було впроваджено систему класів і персонажів з унікальними атрибутами та здібностями, які налаштовуються через окремі конфігураційні об’єкти. Для активації здібностей використовувалися спеціальні механізми, що дозволяють коректно

обробляти вибір цілей і передавати інформацію між клієнтом і сервером у мережевому середовищі. Мережева частина гри була зосереджена в окремому модулі, відповідальному за підключення гравців через онлайн-сервіси. Там реалізовано функції входу в систему, створення та пошуку ігрових сесій, а також приєднання до них. Під час створення сесії враховувались основні налаштування гри, включно з кількістю учасників та вибраною картою. Окремо було налаштовано обробку всіх ключових подій, пов’язаних із багатокористувацькою взаємодією.

Тестування гри проходило в два етапи: спочатку локально з емуляцією мережі для перевірки базової роботи багатокористувацьких функцій, а згодом – у реальних умовах через Інтернет. Завдяки інтеграції з Epic Online Services було успішно перевірено авторизацію користувачів, стабільність з’єднання, а також коректну взаємодію між гравцями без суттєвих відхилень, навіть за наявності затримок у мережі.

Результатом роботи став реалізований прототип PvP-гри, котри продемонстрував працездатність обраної архітектури, доцільність використання Unreal Engine для проєктів подібного типу та наявність усіх необхідних засобів для подальшої розробки повноцінного комерційного продукту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Schell, J. The Art of Game Design: A Book of Lenses. 3rd ed. Boca Raton : CRC Press, 2019. 610 p. ISBN 978-1-138-63974-9.

2. Gregory, J. Game Engine Architecture. 3rd ed. Boca Raton : CRC Press, 2018. 1024 p. ISBN 978-1-138-03545-9.

3. Unity Technologies. Unity Manual. [Електронний ресурс]. Unity Documentation. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 11.06.2025).

4. Godot Engine contributors. Godot Engine Documentation. [Електронний ресурс]. Godot Engine. URL: https://docs.godotengine.org/en/stable/ (дата звернення: 11.06.2025).

5. Petricca, T. Unreal Engine Game Development Cookbook: Over 100 Recipes to Ignite Your Creativity. 2nd ed. Birmingham : Packt Publishing, 2020. 678 p. ISBN 978-1-78953-961-2.

6. Romero M., Sewell B. Blueprints Visual Scripting for Unreal Engine 5. 3rd ed. Birmingham : Packt, 2022. 568 p

7. Smeding, T. Network Programming with Game Engines. Boca Raton : CRC Press, 2020. 360 p. ISBN 978-0-367-43407-3.

8. Sherif, M. Mastering Unreal Engine 5.X. Birmingham : Packt Publishing, 2016. 540 p. ISBN 978-1-78217-491-9.

9. Vasilis, L. Learning Unreal Engine Game Development. Birmingham : Packt Publishing, 2015. 290 p. ISBN 978-1-78355-461-1.

10. Stroustrup, B. The C++ Programming Language. 4th ed. Upper Saddle River : Addison-Wesley, 2013. 1366 p. ISBN 978-0-321-56384-2.

11. Epic Games. Gameplay Ability System for Unreal Engine. [Електронний ресурс]. Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/gameplay-ability-system-for-unreal-engine (дата звернення: 10.06.2025).