ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Розробка гри "Warmage" у жанрі стратегія в реальному часі з використанням алгоритмів інтелектуальної поведінки ботів»

Ступінь вищої освіти – бакалавр
Спеціальність – 122 «Комп’ютерні науки»
Освітня програма «Комп’ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА


Виконав: здобувач 4 курсу
групи КН-21
Денис КАЧАЛЕНКО
Керівник: к. військ. н. доцент кафедри
комп'ютерних наук
Володимир ТРОЦЬКО


Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп'ютерних наук
_____Сергій МІЧКІВСЬКИЙ
«____»____20___р

# ЗАВДАННЯ

# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Качаленко Денис Сергійович

| | |
|---|---|
| Тема роботи | Розробка гри "Warmage" у жанрі стратегія в реальному часі з використанням алгоритмів інтелектуальної поведінки ботів |
| Номер та дата наказу про затвердження теми | №121-7 від 24 грудня 2024 року |
| Коротка постановка завдання | Розробити гру у жанрі стратегія в реальному часі "Warmage" на Unreal Engine 5, зосередившись на ШІ, механіки геймплею та оптимізації продуктивності для збільшення FPS. |
| Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник) | 1. Epic Games. Документація Unreal Engine 5 [Електронний ресурс]. – Режим доступу: https://docs.unrealengine.com/.<br>2. Buckland, Mat. AI and Game Development. – Кембридж: MIT Press, 2005. – 512 с.<br>3. Nystrom, Robert. Game Programming Patterns. – Сан-Франциско: Genever Benning, 2014. – 354 с.<br>4. Rogers, Scott. Level Up! The Guide to Great Video Game Design. – Нью-Йорк: Wiley, 2014. – 552 с. |
| Вимоги до кваліфікаційної роботи | Кваліфікаційна робота має передбачити теоретичне, системотехнічне або експериментальне дослідження складного спеціалізованого завдання або практичної проблеми в галузі комп'ютерних наук, яке характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій. |

Дата видачі завдання 27 грудня 2024 р.

Керівник Володимир ТРОЦЬКО

Здобувач освітнього ступеня бакалавра Денис КАЧАЛЕНКО

# КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін виконання | Примітка |
|---|---|---|---|
| | **Підготовчий етап** | | |
| 1 | Вибір напряму дослідження | 02.12.2024 р. | *виконано* |
| 2 | Формування теми та призначення керівника | 16.12.2024 р. | *виконано* |
| 3 | Затвердження теми кваліфікаційної роботи | 23.12.2024 р. | *виконано* |
| 4 | Затвердження завдання на кваліфікаційну роботу | 27.12.2024 р. | *виконано* |
| | **Основний етап** | | |
| 5 | Розробка концепції кваліфікаційної роботи | 13.01.2025 р. | *виконано* |
| 6 | Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів | 20.01.2025 р. | *виконано* |
| 7 | Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи | 10.03.2025 р. | *виконано* |
| 8 | Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи | 24.03.2025 р. | *виконано* |
| 9 | Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль) | 31.03-04.04.2025 р. | *виконано* |
| 10 | Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи | 07.04.2025 р. | *виконано* |
| 11 | Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи | 14.04.2025 р. | *виконано* |
| 12 | Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи | 21.04.2025 р. | *виконано* |
| | **Завершальний етап** | | |
| 13 | Представлення рукопису для перевірки на плагіат | 28.04-04.05.2025 р. | *виконано* |
| 14 | Підготовка презентації та доповіді на передзахист | 05.05-11.05.2025 р. | *виконано* |
| 15 | Передзахист кваліфікаційної роботи | 12.05-16.05.2025 р. | *виконано* |
| 16 | Доопрацювання роботи за результатами передзахисту | 19.05-06.06.2025 р. | *виконано* |
| 17 | Експертиза роботи керівником та зовнішнім експертом | 09.06-15.06.2025 р. | *виконано* |
| 18 | Доопрацювання доповіді та презентації для захисту | 09.06-15.06.2025 р. | *виконано* |
| 19 | Захист кваліфікаційної роботи | 16.06-22.06.2025 р. | *виконано* |

Керівник Володимир ТРОЦЬКО

Здобувач освітнього ступеня бакалавра Денис КАЧАЛЕНКО

*Качаленко Д.С. Розробка гри "Warmage" у жанрі стратегія в реальному часі з використанням алгоритмів інтелектуальної поведінки ботів*

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 – Комп'ютерні науки (освітня програма – Комп'ютерні науки) СО Бакалавр. – ВНЗ .Університет економіки та права .КРОК., Навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп'ютерних наук, Київ, 2025.

У даній роботі розроблено стратегічну гру в реальному часі з використанням Unreal Engine 5, з акцентом на механіці ігрового процесу, оптимізації та елементах, керованих штучним інтелектом. Дослідження охоплює основні принципи дизайну RTS, покращення продуктивності та інтеграцію ШІ для підвищення стратегічної глибини та швидкості реагування.

Ключові слова: стратегія в реальному часі, RTS, розробка ігор, Unreal Engine 5, ігрова механіка, оптимізація, штучний інтелект.

Табл. 3. Рисунок. 15. Бібліограф. 30.

*Kachalenko D.S. Development of the "Warmage" game in the real-time strategy genre using algorithms for intelligent bot behavior*

Explanatory note of qualification work in specialty 122 - Computer Science (educational programme - Computer Science), Bachelor's degree - University of Economics and Law "KROK", Educational and Research Institute of Information and Communication Technologies, Department of Computer Science, Kyiv, 2023.

In this work, a real-time strategy game is developed using Unreal Engine 5, with a focus on gameplay mechanics, optimization, and AI-driven elements. The research covers core RTS design principles, performance improvements, and AI integration to enhance strategic depth and responsiveness.

Keywords: real-time strategy, RTS, game development, Unreal Engine 5, gameplay mechanics, optimization, artificial intelligence.

Table 3. Fig. 15. Bibliography. 30.

# ЗМІСТ

# ВСТУП

Актуальність теми: жанр RTS залишається життєво важливим в іграх, вимагаючи складної механіки, управління ресурсами та динамічної взаємодії. Unreal Engine 5 пропонує потужні інструменти для імерсивної графіки, мережевої взаємодії та геймплею зі штучним інтелектом. Однак багато RTS-ігор мають проблеми з балансуванням між складністю, чутливістю ШІ та продуктивністю. Ця робота зосереджена на розробці Warmage, RTS-гри з використанням Unreal Engine 5 [1], що поєднує просунутий ШІ з традиційними механіками для покращення занурення та ефективності. Проект спрямований на розвиток RTS шляхом вирішення ключових проблем у механіці, оптимізації та поведінці ШІ.

Мета дослідження: основною метою цього дослідження є розробка «Warmage» як повнофункціональної RTS-гри із забезпеченням плавного ігрового процесу, стратегічної глибини та ефективності роботи [5]. Проект досліджуватиме ключові аспекти розробки гри, включаючи основні механіки, взаємодії керовані штучним інтелектом, та оптимізацію системи в рамках рушія Unreal Engine 5.

Завдання дослідження: для досягнення поставленої мети будуть виконані наступні завдання:

- проаналізувати існуючі RTS-ігри для виявлення ключових принципів дизайну та технологічних тенденцій;
- розробити основні механіки ігрового процесу, включаючи управління ресурсами, контроль юнітів та бойові системи;
- впровадити елементи керовані штучним інтелектом, які підвищують стратегічну глибину та адаптивність поведінки NPC;
- оптимізувати продуктивність гри, щоб забезпечити безперебійну взаємодію в реальному часі та масштабованість.

Об'єкт дослідження: процес розробки стратегічної гри в реальному часі, включаючи механікі, ШІ та оптимізацію.

Предмет дослідження: реалізація геймплею RTS, функцій штучного інтелекту та покращення продуктивності у грі «Warmage».

Методи дослідження: у дослідженні буде використано декілька методів дослідження, зокрема:

- теоретичний аналіз ігрової механіки RTS та методів ШІ [6];
- розробка програмного забезпечення на Unreal Engine 5 з акцентом на ефективну реалізацію;
- порівняльний аналіз існуючих RTS-ігор для вдосконалення елементів геймплею;
- тестування та оцінка продуктивності для забезпечення стабільності та масштабованості.

Використання інформаційних технологій: проект буде використовувати Unreal Engine 5 як основне середовище розробки, що включає власні та сторонні інструменти для логіки ШІ, фізичного моделювання та графічного рендерингу. Додаткові інструменти профілювання будуть використовуватися для моніторингу продуктивності та оптимізації ефективності гри [8].

Практичне значення: це дослідження зробить внесок у розробку ігор, продемонструвавши структурований підхід до створення RTS-гри, який збалансовує стратегічну глибину, складність ШІ та ефективність продуктивності. Отримані результати можуть бути корисними для майбутніх розробників ігор, надаючи уявлення про оптимізацію механіки RTS та впровадження штучного інтелекту.

Структура кваліфікаційної роботи: кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку посилань (30 найменувань). Пояснювальна записка містить 3 таблиці, 15 рисунків. Загальний обсяг пояснювальної записки складає 56 сторінок, основний зміст викладено на 53 сторінках.

# РОЗДІЛ 1
# ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМНОГО ПРОДУКТУ

## 1.1 Опис предметної області

Стратегії в реальному часі (RTS) залишаються популярним жанром в ігровій індустрії, пропонуючи складний стратегічний геймплей, що вимагає управління ресурсами, тактичного планування та швидкого прийняття рішень. На відміну від покрокових стратегічних ігор, події в RTS розгортаються в реальному часі, вимагаючи від гравців безперервної взаємодії, щоб контролювати юніти, управляти економікою та ефективно протистояти супротивникам. Еволюція RTS-ігор визначалася розвитком ігрових рушіїв, штучного інтелекту (ШІ) [7] та мережевих технологій, що призвело до більш захопливого та динамічного ігрового досвіду.

Цей розділ містить огляд жанру RTS, його основних механізмів та ролі штучного інтелекту в покращенні ігрового процесу. Обговорення створить основу для розробки Warmage, RTS-гри, створеної з використанням Unreal Engine 5 (UE5), а також окреслить її мету та сферу застосування в ігровому ландшафті.

*Призначення програмного продукту*

Основна мета Warmage - створити RTS нового покоління, що поєднує в собі традиційні елементи геймплею з просунутими механіками, керованими штучним інтелектом [2]. Гра спрямована на досягнення стратегічної глибини шляхом включення інтелектуальних неігрових персонажів (NPC), здатних адаптуватися до стратегій гравців у реальному часі. На відміну від багатьох традиційних RTS-ігор, які покладаються на заздалегідь задану поведінку ШІ, Warmage використовує динамічні моделі ШІ, які коригують тактику на основі дій гравця, забезпечуючи більш складний та захопливий досвід.

Проект розробляється з наступними цілями:

- покращення стратегічного ігрового процесу: впровадження контрольованих ШІ юнітів, які динамічно реагують на рішення гравця [3];
- оптимізація продуктивності: використання ефективних систем для підтримки плавного ігрового процесу навіть у масштабних битвах;
- однокористувацький та багатокористувацький режими: підтримка як сутичок, керованих ШІ, так і багатокористувацької взаємодії в онлайні;
- масштабованість і модифікованість: розробка гнучкої архітектури, яка дозволяє легко розширювати та оновлювати контент [23].

*Сфера застосування програмного продукту*

Warmage призначена для гравців, які полюбляють змагальний та стратегічний геймплей, і розрахована як на казуальних, так і на хардкорних ентузіастів RTS [11][12][13]. Гра підтримує різні режими, серед яких:

- режим сутички: користувацькі матчі, де гравці можуть змагатися проти ШІ;
- багатокористувацькі битви: онлайн-матчі, що дозволяють гравцям змагатися з реальними супротивниками;

Warmage відрізняється від існуючих RTS-ігор тим, що робить акцент на адаптивності ШІ, дозволяючи NPC розвивати свою тактику протягом усього матчу. Крім того, можливості рендерингу UE5 у реальному часі сприяють візуально приголомшливим полям битв, посилюючи занурення гравців у гру.

*Технологічна основа*

Розробка Warmage побудована на UE5 з використанням ключових технологій, оптимізованих для геймплею RTS, зокрема:

- Mass Entity System - ефективно керує тисячами низько полігональних юнітів, забезпечуючи безперебійний перебіг масштабних битв;

- State Tree + Smart Objects - спрощує ШІ юнітів, структуруючи складну поведінку без надмірних витрат на обробку даних;
- AI Crowd Simulation - забезпечує плавний рух великих груп юнітів, покращуючи пошук шляхів та уникнення зіткнень;
- World Partition - оптимізує великі мапи, передаючи потокове відео лише в необхідних ділянках, покращуючи продуктивність;
- Gameplay Tags + Gameplay Ability System (GAS) - полегшує здібності та взаємодію юнітів без жорсткого кодування механіки.

Використовуючи ці технології, Warmage прагне розширити межі традиційної механіки RTS, поєднуючи стратегічну глибину, продуктивність і захопливий ігровий процес. Теоретичну основу рішення становлять підходи, описані в класичних роботах з архітектури ігрових рушіїв [10] та проектуванні штучного інтелекту для ігор [7].

## 1.2 Аналіз потенційних конкурентних переваг програмного продукту

Жанр стратегій у реальному часі (RTS) є основою ігрової індустрії протягом десятиліть, з багатьма успішними іграми, що встановлюють відмінні механіки ігрового процесу, візуальні стилі та змагальні сцени. Warmage прагне впроваджувати інновації в цій галузі, пропонуючи свіжий погляд на геймплей RTS, використовуючи просунутий штучний інтелект, покращену масштабованість та сучасні технології. Щоб зрозуміти її конкурентні переваги, необхідно порівняти Warmage з трьома провідними RTS-іграми: Age of Empires 4, StarCraft 2 та Cossacks 3. Цей аналіз висвітлить сильні та слабкі сторони цих ігор і продемонструє, чим вирізняється Warmage.

*Порівняння з Age of Empires 4*

Age of Empires 4 (AoE4) (рис 1.1) - це історично натхненна RTS, яка фокусується на побудові імперії, управлінні ресурсами та масштабних битвах. Вона підкреслює асиметричні цивілізації, кожна з яких має унікальні здібності та механіки, і відома своєю доступністю та історичною достовірністю.

Порівняльний аналіз рішень і обмежень між Warmage та Age of Empires 4 наведено у табл. 1.1.

*Таблиця 1.1 - Порівняльний аналіз рішень і обмежень між Warmage та* Age of Empires 4

| *Аспект* | *Age of Empires 4* | *Warmage* |
|---|---|---|
| *Складність та адаптивність ШІ* | Традиційний скриптовий ШІ з фіксованими шаблонами поведінки та масштабуванням складності | Адаптивний ШІ на базі State Tree + Smart Objects, що динамічно реагує на стратегії гравця |
| *Масштаби юнітів та армій* | Підтримка великих армій, але неефективний пошук шляхів у багатолюдних боях | Система масових юнітів для ефективної обробки та переміщення тисяч низько полігональних юнітів |
| *Фентезійні елементи та система магії* | Історичний реалізм без елементів магії | Фентезійний сеттинг із магічними здібностями та системою GAS для плавної інтеграції умінь без жорсткого кодування |
| *Багатокористувацька та мережева оптимізація* | Традиційний серверний пошук матчів без спеціальної оптимізації для великомасштабних боїв | Використання Replication Graph для оптимізації мережевого трафіку та забезпечення плавності великих боїв |

*Рисунок 1.1 – Скупчення юнітів у Age of Empires 4 через обмеження пошуку шляху*

*Джерело: [11]*

*Порівняння з Starcraft 2*

Starcraft 2 (SC2) (рис 1.2) - це динамічна науково-фантастична RTS, відома своєю змагальною атмосферою та точною механікою мікро менеджменту. Баланс гри та асиметрія фракцій встановили високий стандарт ігрового процесу в RTS. Порівняльний аналіз рішень і обмежень між Warmage та Starcraft 2 наведено у табл. 1.2.

*Таблиця 1.2 - Порівняльний аналіз рішень і обмежень між Warmage та Starcraft 2*

| *Аспект* | *Starcraft 2* | *Warmage* |
|---|---|---|
| *Швидкість та стратегічна глибина* | Акцент на швидкому прийнятті рішень і мікро-інтенсивному контролі | Контрольовані ШІ-формування та дерева поведінки для зниження тиску мікро управління і фокус на макрорівень |
| *Дизайн місцевості та карт* | Невеликі карти, орієнтовані на швидкі бої | Великі карти завдяки World Partition + Data Layers, що підтримують масштабні, рухливі битви |
| *Різноманітність і кастомізація фракцій* | Три жорстко визначені фракції з фіксованими стилями гри | Динамічний склад фракцій із можливістю змінювати здібності та стратегії через дерево технологій |
| *Модернізація та підтримка спільноти* | Потужний редактор карт для створення користувацького контенту | Розширені інструменти модифікації для створення нових режимів, юнітів та механік без значних зусиль |

*Рисунок 1.2 – Висока інтенсивність мікро управління під час бою у Starcraft 2*

*Джерело: [12]*

*Порівняння з Cossacks 3*

Cossacks 3 (рис 1.3) - це історична RTS, яка спеціалізується на масштабних битвах з тисячами юнітів на екрані. Вона фокусується на економічному управлінні та стратегічній глибині війни. Порівняльний аналіз рішень і обмежень між Warmage та Cossacks 3 наведено у табл. 1.3.

*Таблиця 1.3 - Порівняльний аналіз рішень і обмежень між Warmage та Cossacks 3*

| *Аспект* | *Cossacks 3* | *Warmage* |
|---|---|---|
| *Масштаб та ефективність юнітів* | Підтримка битв з тисячами юнітів | Розширена система масових юнітів для плавних масштабних битв без втрати продуктивності |
| *Формування, керовані ШІ, та тактична глибина* | Формування юнітів із традиційною поведінкою ШІ | ШІ-симуляція натовпу для органічного та динамічного руху військ залежно від умов бою в реальному часі |
| *Візуальний стиль та оптимізація продуктивності* | Попередньо відрендерингові спрайти з обмеженою фізичною взаємодією | Мультяшний стиль із використанням HLOD для збереження продуктивності без втрати візуальної якості |
| *Бойова система та інтеграція магії* | Війна епохи пороху з історичною точністю, без магічних елементів | Фентезійні бої із застосуванням магічної облогової зброї, заклинань юнітів та глибокої механіки магії |

*Рисунок 1.3 – Хаотичний рух військ у великій битві в Cossacks 3*

*Джерело: [13]*

*Короткий огляд конкурентних переваг*

Warmage відрізняється від своїх конкурентів завдяки наступним ключовим особливостям:

- адаптивна поведінка ШІ: використання StateTree + Smart Objects для створення більш чутливого та складного ШІ-супротивника [20];
- масштабні битви з оптимізованою продуктивністю: реалізація системи масових об'єктів для управління тисячами юнітів без падіння FPS [24];
- фентезійний сеттінг з унікальною механікою: відмінність від історичних та науково-фантастичних RTS-ігор завдяки впровадженню бойових дій, наповнених магією;
- розширена багатокористувацька мережа: використання графіки реплікації для ефективних масштабних багатокористувацьких битв;
- безперебійне потокове передавання світу: використання World Partition + Data Layers для створення великих і динамічних полів битв;
- широкі можливості для модифікації та налаштування: дозволяє створювати контент, керований гравцем, і розширювати механіки ігрового процесу.

Завдяки цим інноваціям Warmage прагне переосмислити досвід RTS, поєднуючи масштабні стратегічні битви, виклики, керовані штучним інтелектом, та глибокі можливості кастомізації, пропонуючи гравцям свіжий та захопливий ігровий досвід.

### 1.3 Постановка завдання на кваліфікаційну роботу

Розробка RTS гри на кшталт Warmage вимагає структурованого підходу до вирішення проблем, визначення чітких цілей та встановлення технічних вимог. У цьому підрозділі сформульовано завдання кваліфікаційної роботи,

окреслено необхідні системні, функціональні та нефункціональні вимоги, а також описано ключові кроки для досягнення цілей проекту.

*Завдання кваліфікаційної роботи*

Завданням цієї кваліфікаційної роботи є розробка стратегічної гри в реальному часі (RTS) яка інтегрує передові механіки, керовані штучним інтелектом, та оптимізовану продуктивність для великомасштабних битв. Основна увага приділяється підвищенню стратегічної глибини, забезпеченню безперебійної багатокористувацької взаємодії та створенню захопливого ігрового досвіду за допомогою добре продуманої ігрової системи. Основними завданнями є:

- розробка RTS на основі штучного інтелекту, що динамічно адаптується до стратегій гравців;
- оптимізація для великомасштабних боїв юнітів з використанням ефективних структур даних і паралельної обробки;
- безшовна багатокористувацька інтеграція з мінімальними затримками та стабільною продуктивністю;
- масштабованість та модифікованість для майбутнього розширення та оновлення контенту;
- зручний інтерфейс та інтуїтивно зрозуміле управління, що підвищує доступність як для нових, так і для досвідчених гравців.

*Вхідні та вихідні дані*

Вхідні дані:

- команди користувача: дані, що вводяться гравцем для керування юнітами, управління ресурсами та тактичними командами;
- дані для прийняття рішень ШІ: інформація про стан гри, яку ШІ використовує для динамічного коригування стратегій;
- багатокористувацькі дані: мережеві комунікаційні пакети для взаємодії гравців у реальному часі;
- параметри ігрового світу: карта місцевості, розподіл ресурсів і деталі фракцій.

Вихідні дані:

- візуальний та звуковий зворотній зв'язок: відображення в реальному часі юнітів, споруд та елементів навколишнього середовища;
- відповіді ШІ: адаптивні рішення на основі штучного інтелекту, засновані на ігрових подіях;
- багатокористувацька синхронізація: синхронізоване оновлення стану всіх підключених гравців;
- ігрова статистика та журнали: результати матчів, показники продуктивності та журнали налагодження.

*Системні вимоги*

Для забезпечення безперебійної роботи Warmage має відповідати наступним системним вимогам:

Мінімальні системні вимоги:

- операційна система: Windows 10 (64-розрядна);
- процесор: Intel Core i5-6600K / AMD Ryzen 5 1600;
- пам'ять: 8 ГБ оперативної пам'яті;
- відеокарта: NVIDIA GTX 970 / AMD Radeon RX 580;
- сховище: 20 ГБ вільного місця.

Рекомендовані системні вимоги:

- операційна система: Windows 11 (64-розрядна);
- процесор: Intel Core i7-10700K / AMD Ryzen 7 3700X;
- пам'ять: 16 ГБ оперативної пам'яті;
- відеокарта: NVIDIA RTX 3060 / AMD Radeon RX 6700 XT;
- сховище: Рекомендується SSD на 50 ГБ.

*Функціональні вимоги*

1) управління юнітами та ресурсами:
    - команди вибору, переміщення та атаки для юнітів;
    - збір ресурсів та управління економікою;
    - будівництво та модернізація будівель;

2) система штучного інтелекту:
    - адаптивні стратегії ШІ для одиночної гри та режимів сутичок;
    - пошук шляху та уникнення зіткнень для юнітів;
3) підтримка багатокористувацької гри:
    - синхронізація стану гри між гравцями в реальному часі;
    - ефективна реплікація руху та бою юнітів;
4) користувацький інтерфейс та елементи керування:
    - інтерактивний HUD, що відображає статистику ресурсів та стан юнітів;
    - налаштовувані комбінації клавіш та інтуїтивно зрозуміле управління мишею.

*Нефункціональні вимоги*

1) продуктивність та масштабованість:
    - ефективна обробка масштабних битв з сотнями юнітів;
    - оптимізація для низько полігональної графіки та плавної зміни кадрів;
2) модифікованість і розширюваність:
    - підтримка користувацького контенту та модифікацій;
    - модульна архітектура для легкого оновлення та розширення.

*Кроки для вирішення проблеми*

Процес розробки слідує структурованій методології, щоб забезпечити успішне завершення Warmage:

1) дослідження та аналіз:
    - вивчення існуючих RTS-ігор та визначення їхніх сильних та слабких сторін;
    - аналізування очікувань гравців для створення гри, яка відповідає сучасним стандартам RTS;
2) ігровий дизайн та планування:

- розробити концепт-арт, статистики юнітів та ігрові механіки;
- визначення основних циклів ігрового процесу та взаємодії з користувачем;

3) вибір технологій та створення прототипу:
   - вибрати відповідні технології, включаючи Unreal Engine 5 та підтримуючі фреймворки;
   - створення ранніх прототипів для випробування механік та продуктивності гри;
4) фаза розробки:
   - реалізація основних механізмів, включаючи управління юнітами, штучний інтелект та управління ресурсами;
   - інтеграція мережевих функцій для підтримки багатокористувацької гри;
   - оптимізація продуктивності гри для масштабних битв;
5) тестування та налагодження:
   - провести модульне, інтеграційне та ігрове тестування;
   - виявити та виправити вузькі місця в продуктивності та проблеми з геймплеєм;
6) фіналізація та розгортання:
   - полірування ігрових ресурсів, вдосконалення поведінки ШІ та балансування ігрового процесу;
   - розгортання гри для перевірки та можливого релізу.

**Висновки до розділу 1**

Аналіз предметної області показав, що стратегічні ігри в реальному часі (RTS) продовжують залишатися важливим жанром в ігровій індустрії, що розвивається. Вивчення існуючих ігор, таких як Age of Empires IV, Starcraft II та Cossacks 3, висвітлило ключові тенденції, сильні сторони та обмеження в сучасному дизайні RTS. Ці ігри слугують еталоном для розробки нового

продукту, який має на меті вдосконалити та розширити вже існуючі механіки, водночас вирішуючи загальні проблеми, пов'язані з адаптивністю ШІ, управлінням юнітами та багатокористувацькою синхронізацією.

Аналіз конкурентів виявив кілька сфер, де Warmage може забезпечити значні переваги над своїми аналогами. Серед них - просунута система геймплею зі штучним інтелектом, високо оптимізована система масових об'єктів для ефективного ведення масштабних боїв, а також модульний підхід до ігрової механіки, що підвищує масштабованість і потенціал для модифікацій. Використовуючи екосистему Unreal Engine 5, Warmage включає в себе передові технології, такі як Mass Entity System, State Tree, Replication Graph та Gameplay Ability System (GAS), щоб забезпечити високу продуктивність, стратегічну глибину та плавну багатокористувацьку взаємодію.

Завдання для цієї кваліфікаційної роботи було чітко визначено. Метою є розробка та реалізація масштабованого фреймворку RTS на основі штучного інтелекту, який може ефективно керувати тисячами юнітів у реальному часі, зберігаючи при цьому високу продуктивність та стабільність мережі. Для досягнення цієї мети в дослідженні були визначені системні, функціональні та нефункціональні вимоги, а також структурований підхід до вирішення проблем. Технологічна основа проекту була ретельно підібрана, щоб оптимізувати як ігровий досвід, так і робочий процес розробки.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

## 2.1 Моделювання поведінки продукту

Другий розділ присвячений проектуванню архітектури гри Warmage, включаючи моделювання основних процесів, структури компонентів та взаємодії елементів системи. За допомогою сучасних методів поведінкового моделювання було спроектовано логіку управління юнітами, роботу штучного інтелекту, обробку здібностей та оптимізацію багатокористувацької взаємодії.

У розділі наведено діаграми варіантів використання, діяльності, послідовності та комунікації, які відображають ключові процеси гри. Також описано застосування технологій Unreal Engine 5 і особливості архітектурних рішень, що забезпечують масштабованість, адаптивність ШІ та стабільність ігрового процесу.

*Діаграма варіантів використання (рис 2.1)*

Діаграма варіантів використання забезпечує високорівневе представлення взаємодії між користувачами (гравцями) та системою. Вона ілюструє ключові функціональні можливості, доступні в Warmage.

До основних дійових осіб в системі відносяться:

- гравець - контролює юніти, управляє ресурсами та бере участь у боях;
- ШІ супротивник - змагається з гравцем за допомогою адаптивних стратегій;
- система підбору матчів - керує багатокористувацькими ігровими з'єднаннями, пошуком та підняттям серверів;
- ігровий рушій – займається рендерингом, керуванням геймплейними системами, обробкою вводу гравця та передачею даних мережею.

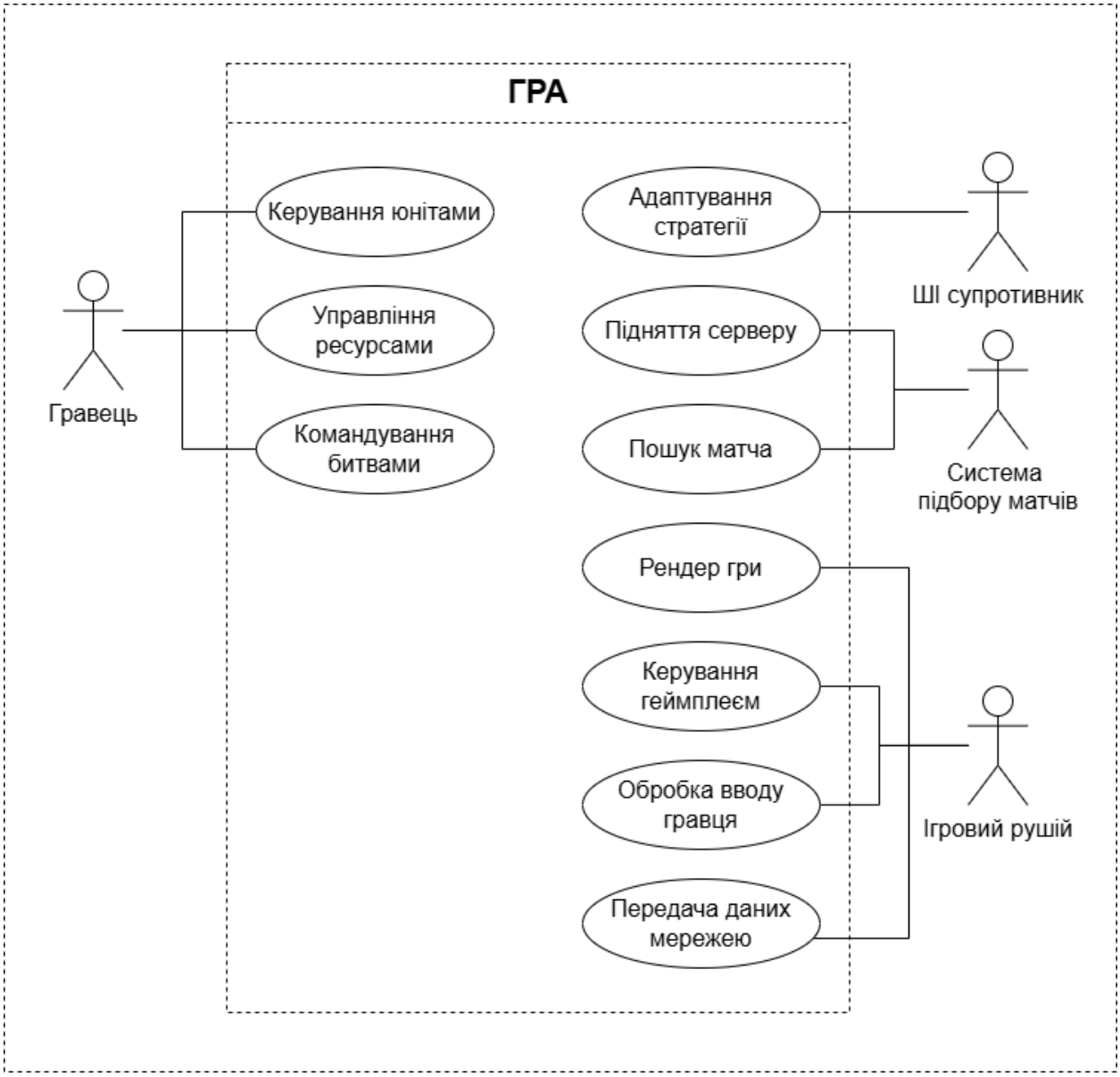



*Рисунок 2.1 – Діаграма варіантів використання (Use Case Diagram)*

*Джерело: розроблено автором*

*Діаграма активності (рис 2.2)*

Діаграма активності зображує потік дій у типовому матчі, показуючи, як змінюється стан гри на основі дій користувача та системи.

Опис:

1) гравець починає гру та обирає тип матчу;
2) якщо обрано багатокористувацький режим, система підключається до сервісу підбору матчів;

3) гра ініціалізується, завантажуються ресурси та підготовлюється ШІ;
4) гравець керує юнітами та взаємодіє з ігровим світом;
5) ШІ-суперник адаптує свою стратегію;
6) гра триває доти, доки не будуть виконані умови перемоги.

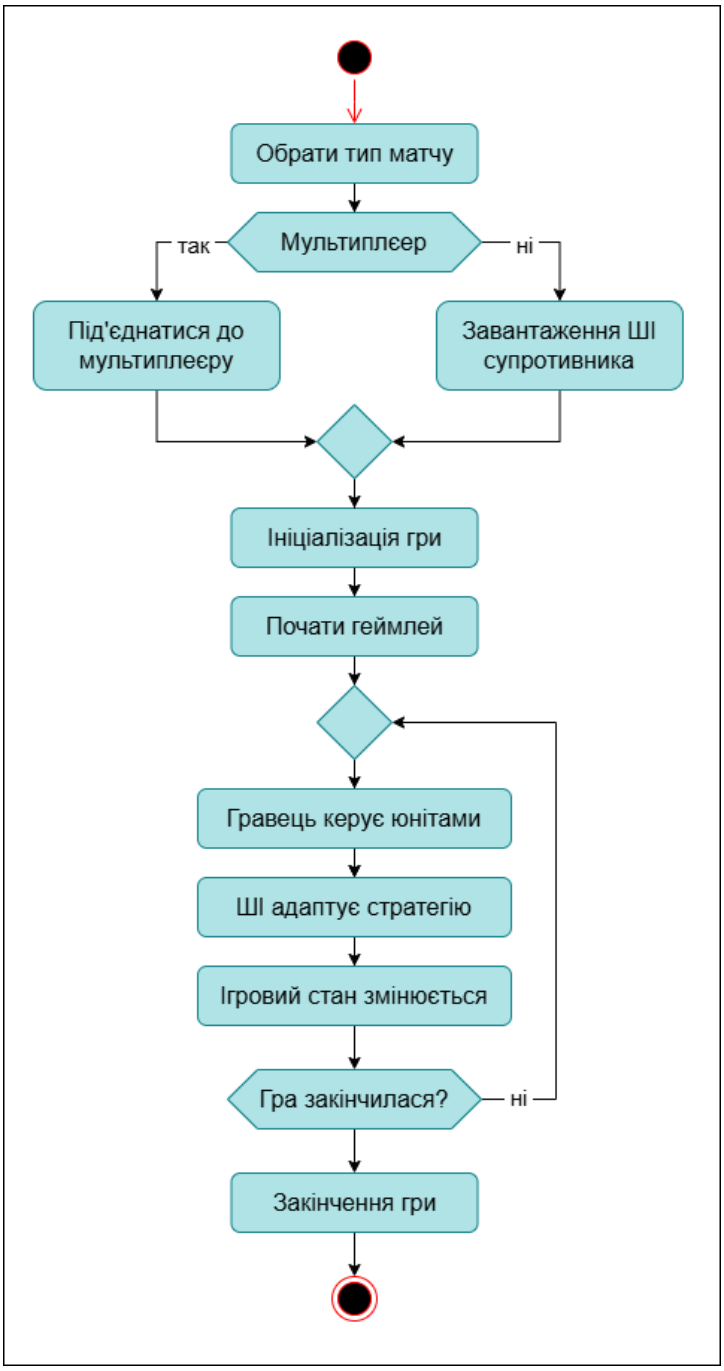



*Рисунок 2.2 – Діаграма діяльності (Activity Diagram) (блок-схема)*

*Джерело: розроблено автором*

*Діаграма послідовності (рис 2.3)*

Діаграма послідовності моделює взаємодію між гравцем, ШІ та ігровим рушієм під час ігрового процесу.

Опис:

1) гравець дає команду юніту;
2) рушій обробляє команду та оновлює стан юніта;
3) ШІ-суперник оцінює хід гравця та реагує на нього;
4) рушій оновлює всі юніти та перемальовує ігровий світ.

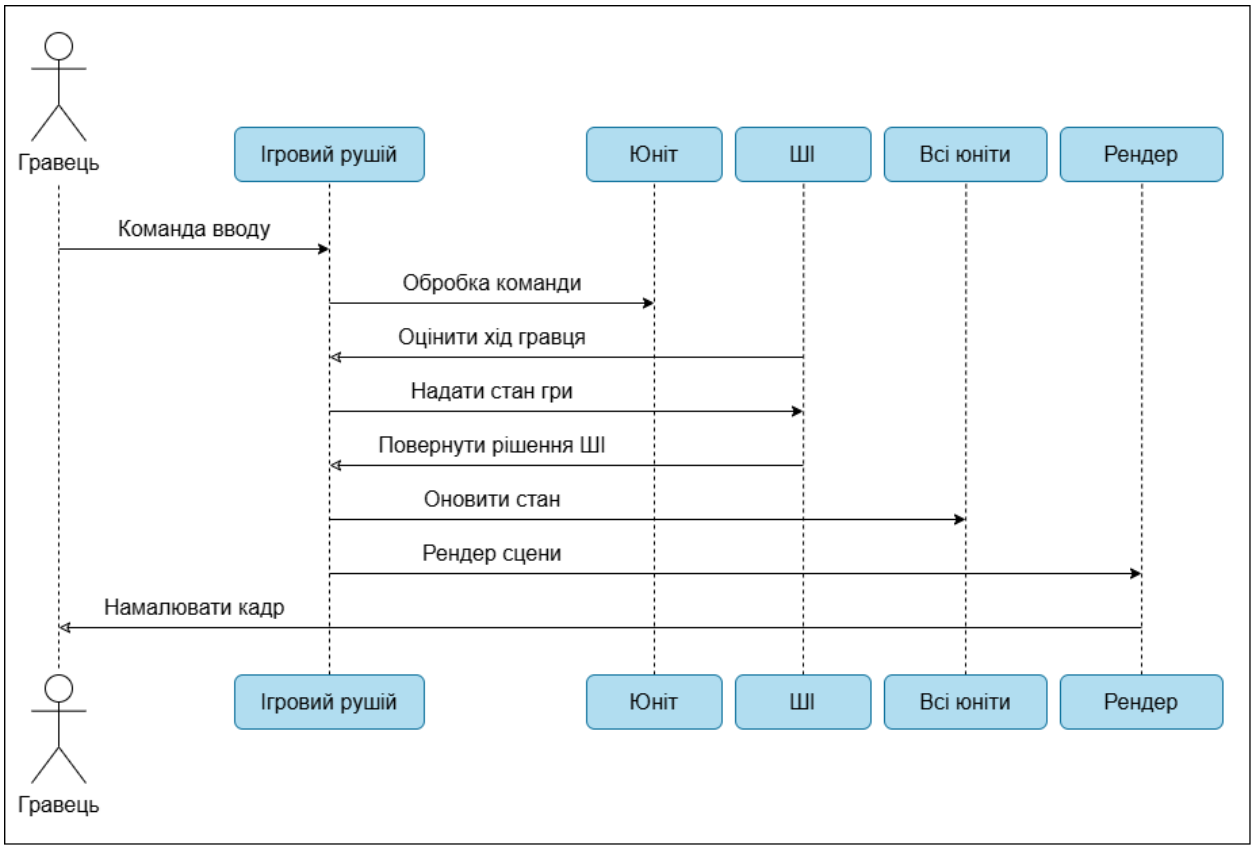



*Рисунок 2.3 – Діаграма послідовності (Sequence Diagram)*

*Джерело: розроблено автором*

Поведінкове моделювання Warmage забезпечує структурне моделювання взаємодій у цій грі. Діаграма варіантів використання висвітлює основні функціональні можливості системи. Діаграма діяльності окреслює основний потік ігрового процесу. Діаграма послідовності демонструє взаємодію в реальному часі між гравцем, ШІ та ігровим рушієм. Ці моделі

забезпечують чітку та ефективну структуру проектування Warmage, допомагаючи в розробці та оптимізації ігрових механік.

## 2.2 Моделювання структури продукту

Структурне моделювання визначає, як організована програмна система, включаючи її внутрішні компоненти, взаємозв'язки між ними та ієрархію підсистем. У контексті розробки стратегічної гри в реальному часі (RTS) таке моделювання відіграє фундаментальну роль у визначенні та документуванні підсистем, відповідальних за контроль юнітів, управління ресурсами, імітацію поведінки ШІ, обробку введених користувачем даних та підтримку багатокористувацької логіки.

У цьому проекті структурне моделювання розглядається крізь призму сучасних патернів проектування архітектури ігрових рушіїв, зокрема, з акцентом на парадигмах, орієнтованих на дані. Моделі розроблені з урахуванням масштабованості, паралелізму та модульності. Це узгоджується з принципами, закладеними у фреймворку Mass Framework для Unreal Engine 5, хоча тут не представлено жодної конкретної реалізації для рушія. Натомість, абстрактні принципи моделювання проілюстровано за допомогою формальних діаграм UML, включаючи діаграму класів, діаграму об'єктів, діаграму пакетів, діаграму компонентів та діаграму розгортання. Ці діаграми визначають архітектуру системи та забезпечують основу для реалізації, яка підтримує реагування в реальному часі та широкомасштабне моделювання.

*Діаграма класів (рис 2.4)*

Діаграма класів представляє високорівневий огляд ключових класів, що використовуються для моделювання ігрових систем, керування юнітами, симуляції та багатокористувацької взаємодії.

Ключові поняття класів:

- Entity - узагальнена одиниця в симуляції;
- PlayerController - інтерфейс для введення даних користувачем та вибору юнітів;

- AIController - обробляє автономну поведінку;
- SimulationSystem - оновлює всі активні сутності за кадр;
- CommandBuffer - збирає та обробляє видані команди;
- TagComponent - додає до сутностей ознаки ігрової логіки (аналогічно до тегів геймплею);
- StateTreeRunner - Оцінює ієрархічні дерева логіки ШІ;
- SmartObjectRegistry - Зберігає інтерактивні об'єкти світу та їхній стан;
- ResourceSystem - відстежує та змінює значення ігрових ресурсів;
- ReplicationManager - керує мережевою реплікацією та повноваженнями.

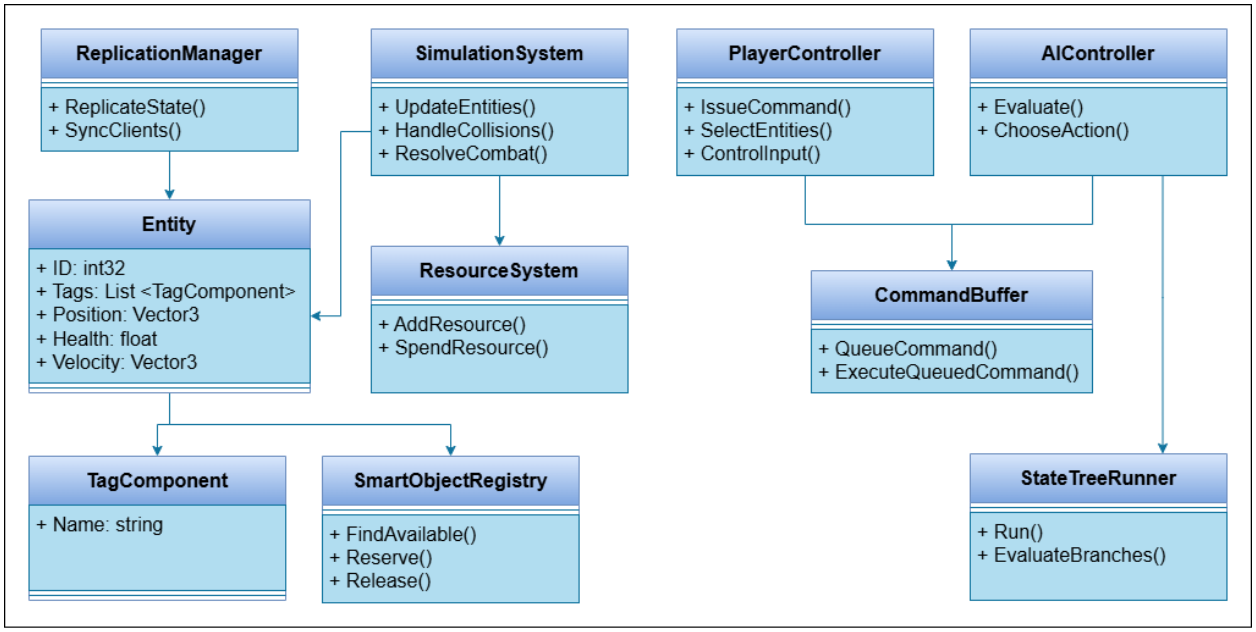



*Рисунок 2.4 – Діаграма класів (Class Diagram)*

*Джерело: розроблено автором*

*Діаграма об’єктів (рис 2.5)*

Об'єктна діаграма надає моментальний знімок стану системи під час виконання, фіксуючи конкретні екземпляри об'єктів та їхні взаємозв'язки під час активної симуляції. Вона показує, як основні компоненти гри працюють

разом у певний момент часу, показуючи, як взаємодіють гравець, ШІ, командні структури та ігрові об'єкти.

У цьому сценарії гравець і контролер ШІ віддають команди, якими керує центральний CommandBuffer. Дві ігрові сутності, солдат1 і лучник1, створюються в симуляції, кожна з яких позначена певною поведінковою характеристикою («Атакувати» і «Захищатися» відповідно) за допомогою пов'язаних об'єктів TagComponent. Логіка симуляції інкапсульована SimulationSystem, яка активно відстежує та оновлює всі запущені об'єкти, включаючи взаємодію з ResourceSystem для прийняття рішень, пов'язаних з економікою. AIController також підключений до StateTreeRunner, який визначає свої тактичні рішення на основі структурованих логічних дерев.

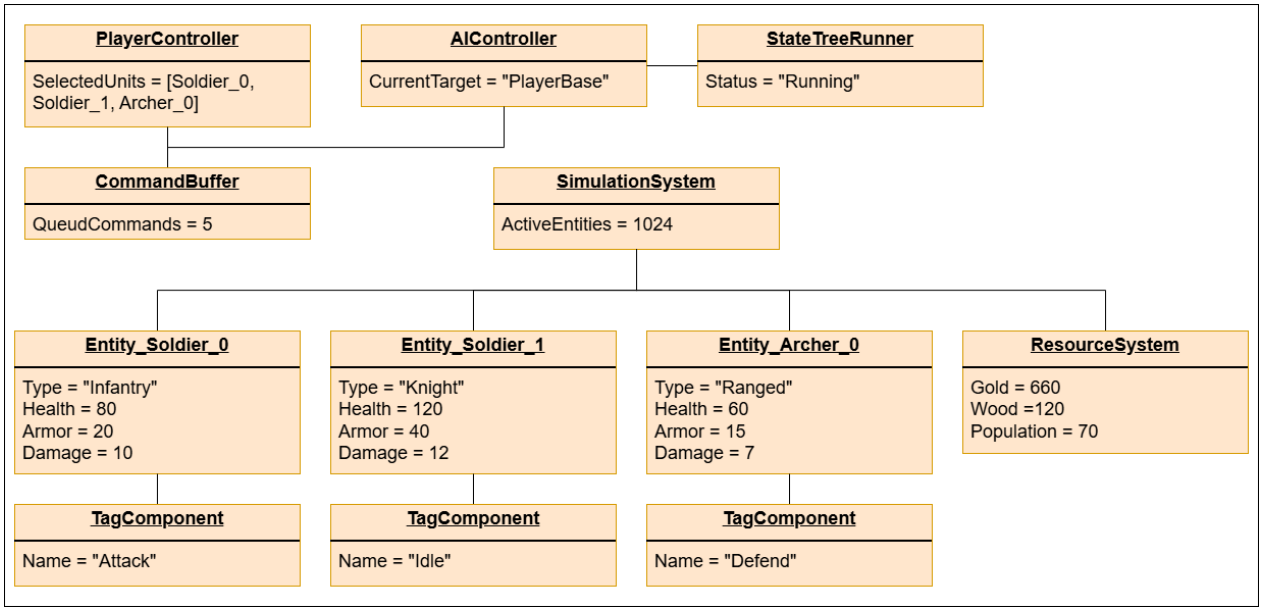



*Рисунок 2.5 – Діаграма об'єктів (Object Diagram)*

*Джерело: розроблено автором*

*Діаграма компонентів (рис 2.6)*

Діаграма компонентів відображає макроархітектуру основних модулів системи Warmage, таких як шар введення, мережевий шар, рушій симуляції та керування ресурсами. Вона демонструє залежності між компонентами та підтримку чіткої модульної структури, що полегшує масштабування,

розширення функціональності і впровадження багатопотокового виконання для оптимізації продуктивності гри.

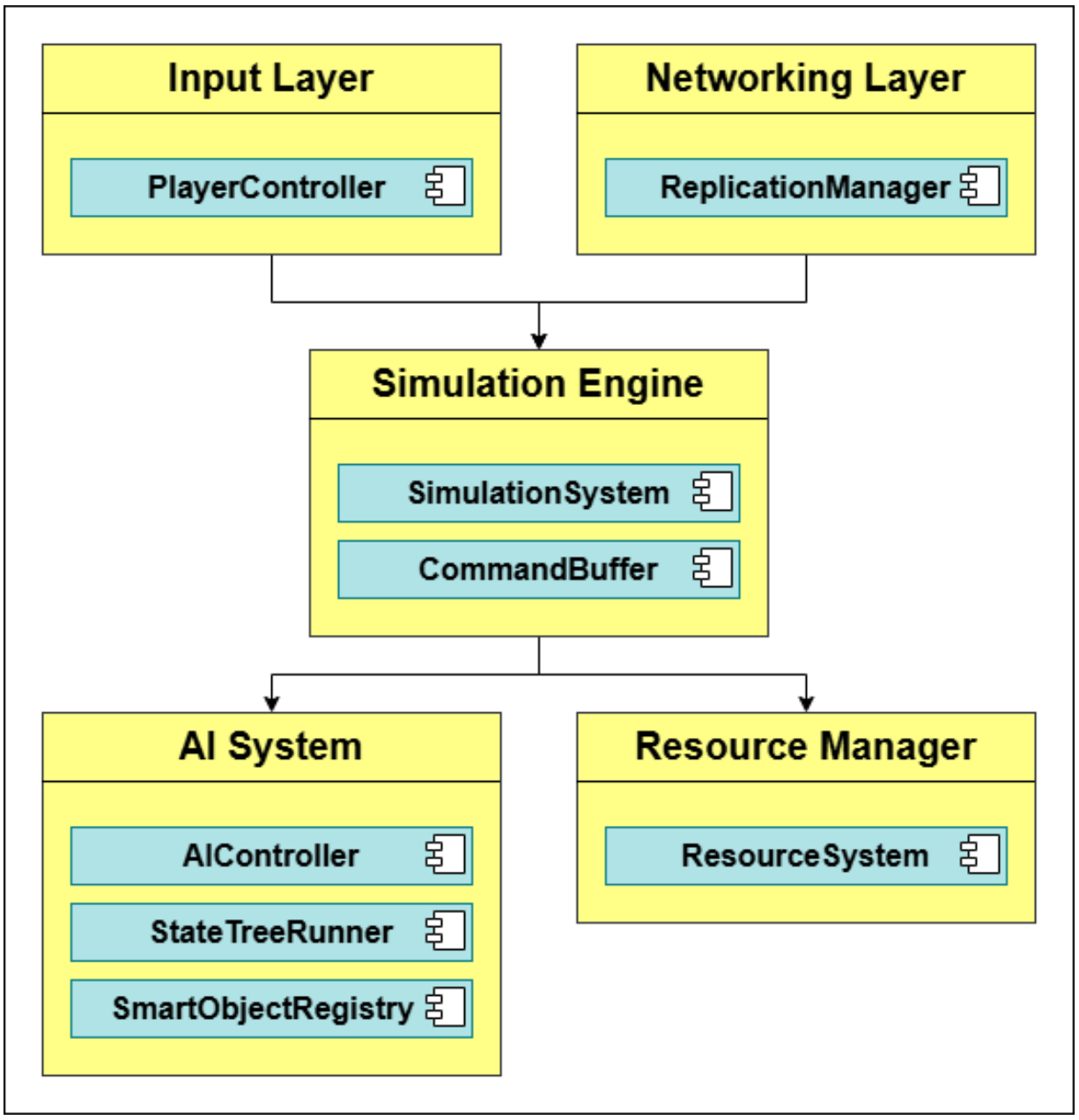



*Рисунок 2.6 – Діаграма компонентів (Component Diagram)*

*Джерело: розроблено автором*

*Діаграма розгортання (рис 2.7)*

Діаграма розгортання показує фізичний розподіл компонентів у клієнт-серверній багатокористувацькій конфігурації.

Структура продукту гри була формально змодельована за допомогою повного набору діаграм UML. Ці моделі абстраговані на основі вимог до дизайну, придатних для великомасштабної симуляції в реальному часі та моделювання поведінки ШІ. Хоча архітектурно вони натхненні технологіями, що використовуються в сучасних рушіях, таких як Unreal Engine 5, зокрема, орієнтованим на дані дизайном і логікою «сутність-компонент», вони залишаються загальними і застосовними на рівні проектування. Ці діаграми слугують планом для майбутньої реалізації та підтверджують можливість паралельного керування тисячами агентів, підтримуючи модульність, ефективне оцінювання ШІ та синхронізацію мультиплеєру у конкурентному ігровому середовищі RTS.

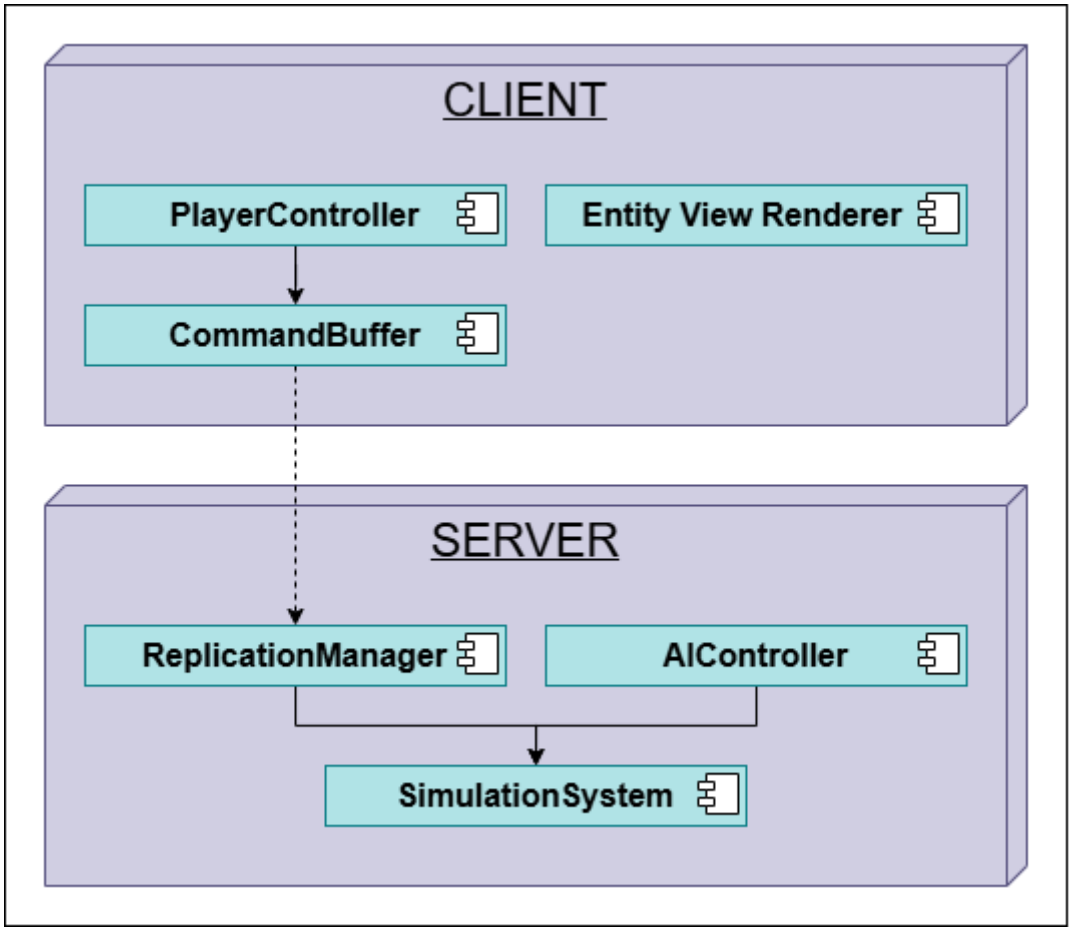



*Рисунок 2.7 – Діаграма розгортання (Deployment Diagram)*

*Джерело: розроблено автором*

## 2.3 Опис архітектури продукту

Архітектура гри Warmage розроблена з урахуванням вимог стратегічних ігор у реальному часі, де необхідно забезпечити ефективне управління великою кількістю юнітів, адаптивність штучного інтелекту, масштабованість багатокористувацьких режимів і стабільну продуктивність при високому навантаженні. Для цього в роботі використано комплексний архітектурний підхід, що включає побудову моделі класів та просторів імен, застосування моделі Kruchten 4+1 для обґрунтування ключових проектних рішень, а також використання моделі C4 для деталізації рівнів системної абстракції.

Такий підхід дозволив формалізувати архітектурне планування Warmage, визначити межі відповідальності компонентів, забезпечити підтримку паралельного виконання процесів у мережевому середовищі та підготувати систему до подальшого масштабування і розширення функціональності.

*Простори імен та структура класів*

Для забезпечення модульності та супроводжуваності архітектура продукту організована у чітко визначені простори імен, кожен з яких групує пов'язані класи та системи.

Огляд просторів імен

- ядро - низькорівневі класи діагностики рушія, що використовуються у симуляції та життєвому циклі сутностей;
- ігровий процес - специфічна для домену логіка, що включає поведінку юнітів, керування гравцем та бойові дії;
- ШІ - дерева поведінки, розумний доступ до об'єктів, контролери штучного інтелекту;
- UI - компоненти користувацького інтерфейсу для вибору, HUD і міні-мапи;
- мережа - багатокористувацька логіка, включаючи реплікацію станів і керування сеансами;

- симуляція - системи, пов'язані з оновленнями на основі кадрів, зіткненнями та обробкою команд;
- ресурси - економічні системи, включаючи генерацію та споживання ресурсів.

*Модель Крюхтена 4+1*

У розробці Warmage застосовано модель Крюхтена 4+1 View Model для формалізації архітектури та обґрунтування ключових технічних рішень. Використання цієї моделі дозволило розглянути систему з різних точок зору, що сприяло кращому структуруванню проєкту та підвищенню його масштабованості.

Сценарії:

- гравець вибирає юнітів → команда в черзі → юніт оновлюється;
- ШІ реагує на дії гравця → пошук шляху → інтелектуальне резервування об'єктів;
- багатокористувацьке приєднання → встановлення відповідності → ініціалізація стану реплікованого світу.

Застосовані погляди моделі:

- логічний погляд визначив основні сутності (гравці, юніти, ШІ) та їхню взаємодію через буфери команд, що забезпечило чітке розмежування обов'язків між системними об'єктами;
- погляд розробки дозволив побудувати модульну структуру коду за доменами простору імен, що спростило командну розробку і перевірку окремих компонентів;
- подання процесів допомогло спроектувати паралельну обробку команд, оцінку ШІ та синхронізацію стану гри між клієнтами й сервером для підвищення продуктивності в реальному часі;
- фізичне представлення забезпечило правильний розподіл навантаження між клієнтами і сервером, дозволяючи зберігати авторитетний стан ШІ на серверній стороні і підтримувати баланс у мережевих багатокористувацьких іграх;

- сценарії описали типові потоки подій у грі, включаючи вибір юнітів гравцем, адаптацію ШІ до дій гравця та ініціалізацію стану світу під час приєднання в мультиплеєрі.

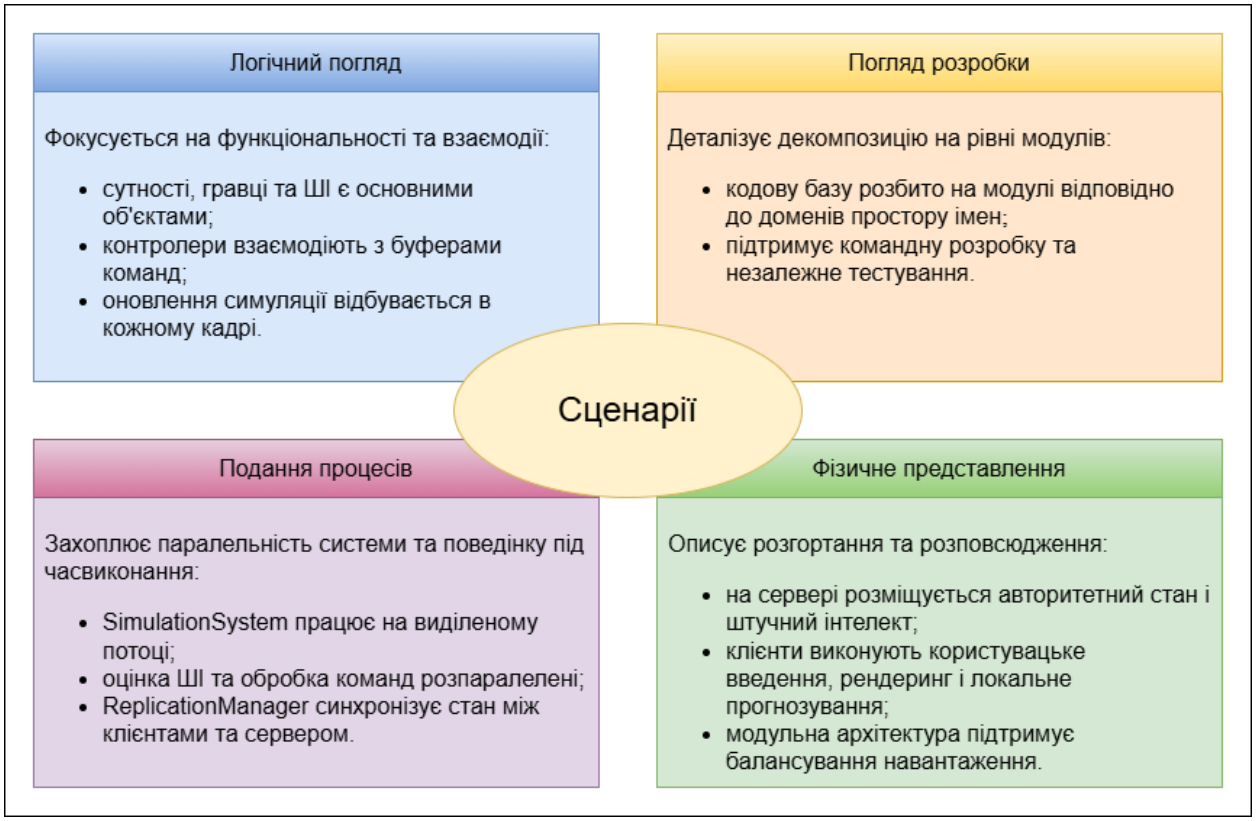



*Рисунок 2.8 – Модель Крюхтена 4+1*

*Джерело: розроблено автором*

Таким чином, модель Крюхтена 4+1 допомогла структурно організувати архітектуру Warmage і закласти основу для подальшого масштабування, оптимізації продуктивності та підтримки багатокористувацьких сценаріїв.

*Модель C4*

У процесі проектування архітектури гри Warmage було застосовано модель C4 (Context, Container, Component, Code) для структурованого опису системи на різних рівнях абстракції. Використання цієї моделі дозволило формалізувати архітектуру від загального огляду взаємодії із зовнішніми системами до детального проектування окремих компонентів і їхньої реалізації.

*Контекст (Context Diagram)*

На цьому рівні Warmage розглядається як багатокористувацька стратегічна гра в реальному часі, яка взаємодіє з наступними зовнішніми системами:

- ігрові сервери для синхронізації стану світу;
- клієнтські пристрої гравців (ПК);
- мережеве середовище для обміну ігровими даними.

Контекстна діаграма визначає кордони системи Warmage та показує основні взаємодії між гравцем, сервером і зовнішніми службами.

*Контейнери (Container Diagram)*

На рівні контейнерів архітектура Warmage складається з кількох основних частин:

- клієнтський застосунок — відповідальний за обробку введення гравця, рендеринг графіки, виконання локального прогнозування;
- серверна частина — керує авторитетним станом світу, обробкою команд, логікою ШІ та синхронізацією клієнтів;
- інтерфейси API — для зв'язку між компонентами системи та зовнішніми службами.

Поділ на контейнери забезпечує масштабованість і незалежність розробки окремих частин гри.

*Компоненти (Component Diagram)*

Кожен контейнер деталізується на рівні компонентів. Основні компоненти Warmage включають:

У клієнтському застосунку:

- Input Manager (обробка команд гравця);
- Render Manager (візуалізація сцени);
- Prediction System (локальне прогнозування стану).

На сервері:

- Simulation System (розрахунок стану світу);
- AI System (адаптивна поведінка ботів);

- Replication Manager (синхронізація стану для клієнтів).

Таке розділення дозволяє забезпечити модульність, спрощує розробку і перевірку.

*Код (Code/Classes)*

На найнижчому рівні розглядаються ключові класи і їх взаємозв'язки. Наприклад:

- APlayerController керує введенням користувача;
- ASimulationManager управляє станом ігрового світу;
- AAIController відповідає за логіку адаптивного ШІ;
- AReplicationManager забезпечує мережеву синхронізацію між клієнтами і сервером.

Організація коду в ієрархічні простори імен і розбиття на модулі дозволяє підтримувати чисту архітектуру і спрощує подальше масштабування системи.

Таким чином, застосування моделі C4 у проекті Warmage забезпечило багаторівневе архітектурне планування.

**Висновки до розділу 2**

Другий розділ кваліфікаційної роботи заклав всебічну основу для розробки масштабованої та ефективної стратегічної гри в реальному часі (RTS). Структура і поведінка Warmage були формалізовані із застосуванням сучасних методологій проектування: моделі C4, моделі Kruchten 4+1 View Model, а також стандартів моделювання UML, що включають діаграми варіантів використання, діяльності, послідовності та компонентів.

Поведінкове моделювання дозволило визначити ключові сценарії взаємодії між користувачами, системою штучного інтелекту та рушієм симуляції, що стало основою для розробки стабільної та прогнозованої внутрішньої логіки ігрового процесу. Діаграми діяльності та послідовності дали змогу проаналізувати основні етапи обробки команд гравця, реакції ШІ та мережевої синхронізації стану гри.

Структурне моделювання за допомогою діаграм класів, об'єктів, пакетів, компонентів і розгортання допомогло визначити внутрішню організацію системи, сформувати межі відповідальності модулів і описати потоки даних між підсистемами. Це особливо важливо для ефективного управління великою кількістю юнітів у реальному часі та підтримки багатокористувацьких режимів гри.

Застосування моделей C4 і Kruchten 4+1 дозволило сформувати цілісне бачення архітектури Warmage на всіх рівнях абстракції — від взаємодії із зовнішніми системами до детального проектування окремих компонентів і класів. Визначення структур просторів імен, ієрархії класів та зон відповідальності компонентів забезпечило високу модульність, супроводжуваності і можливість масштабування системи.

Таким чином, результати проектування на етапі моделювання створили надійну і добре документовану базу для подальшої реалізації, тестування і оптимізації гри Warmage.

# РОЗДІЛ 3
# РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Реалізація та конструювання програмного продукту

Розробка гри Warmage проводилася з використанням сучасного технологічного стеку, який відповідає вимогам до масштабованої, високопродуктивної стратегії в реальному часі (RTS). У цьому розділі описано обрані інструменти розробки, деталі реалізації основних систем та обґрунтування обраних технологій.

*Середовище розробки та основні технології*

Основна розробка була виконана на мові C++ з використанням можливостей Unreal Engine 5 (UE5). C++ було обрано через його пряму сумісність з Unreal Engine, ефективність роботи та контроль над низькорівневим управлінням пам'яттю, що є важливим для оптимізації великомасштабних симуляцій.

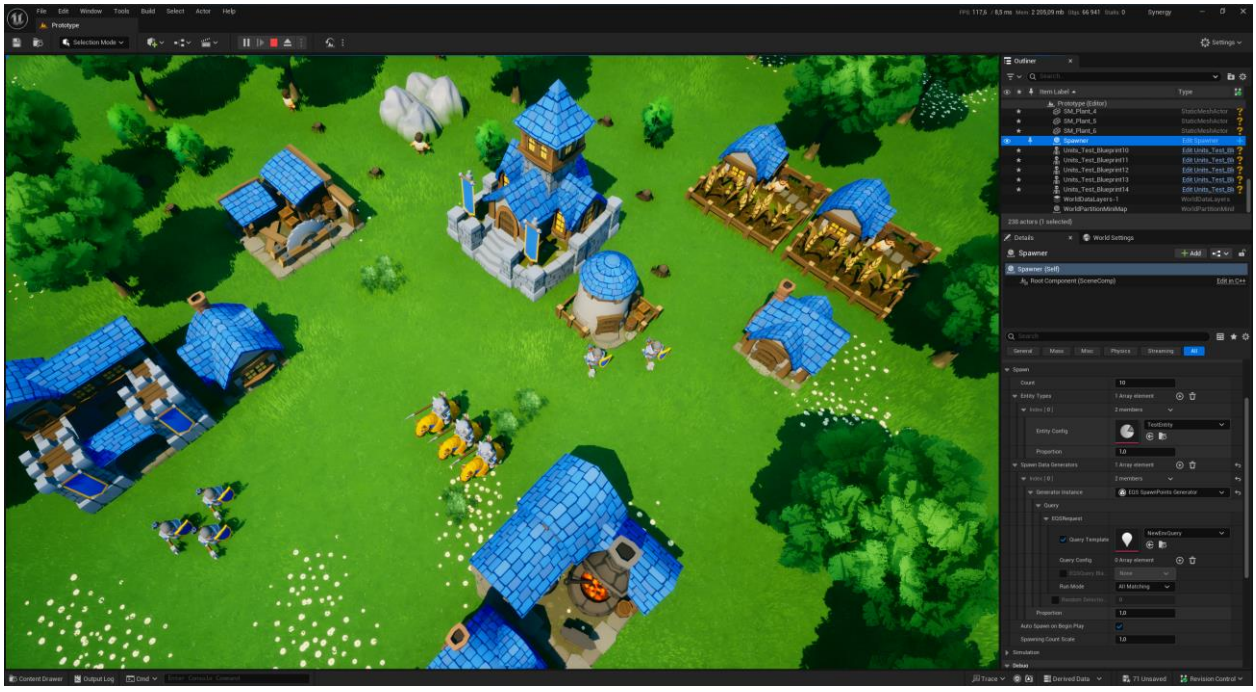


*Рисунок 3.1 – Демонстрація робочого процесу в UE5 по розробці гри*

*Джерело: розроблено автором*

Проект був розроблений для платформи Windows 10/11, забезпечуючи нативну підтримку інструментів Unreal Engine та безшовну інтеграцію з допоміжним програмним забезпеченням для розробки.

IDE JetBrains Rider, оснащена плагіном Unreal Engine, слугувала основним середовищем кодування. Його вдосконалені інструменти аналізу коду, автозавершення та рефакторингу значно підвищили продуктивність та мінімізували помилки під час розробки.

Управління проектом та комунікація були організовані з використанням Jira для відстеження завдань, Confluence для технічної документації та Discord для нотифікації про результат збірок гри та нотування протягом розробки.

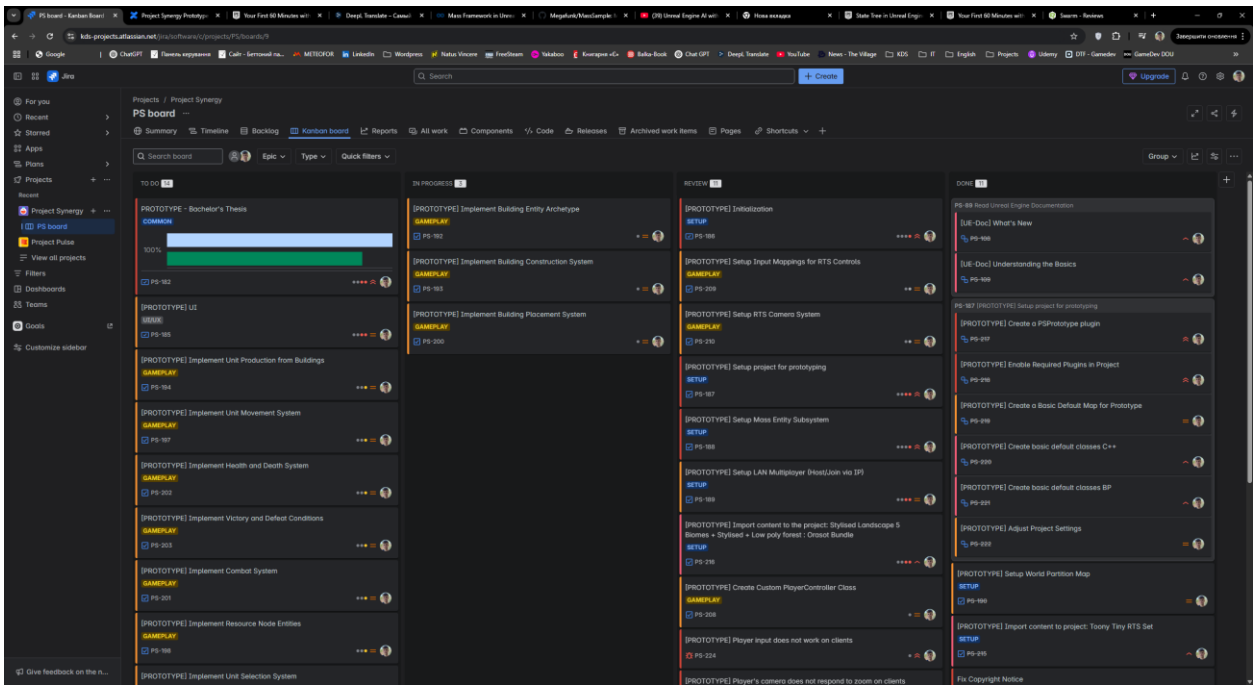



*Рисунок 3.2 – Демонстрація робочого процесу в Jira по організації задач*

*Джерело: розроблено автором*

Управління версіями та співпраця здійснювалися за допомогою Perforce Helix Core Server, повністю інтегрованого в Unreal Editor. Використання Perforce Swarm дозволило організувати робочий процес перегляду коду. Крім того, Git був використаний для управління кастомізованим вихідним кодом Unreal Engine 5 окремо від репозиторію ігрового проекту.

Автоматизація збірки та безперервна інтеграція були налаштовані за допомогою Jenkins, автоматизуючи компіляцію, приготування, пакування та випробувальні запуски. Це забезпечило регулярну перевірку збірок, підвищило стабільність проекту та спростило розгортання.

*Основна архітектура та фреймворки*

Архітектура гри значною мірою спирається на кілька сучасних систем та підсистем UE5 для досягнення бажаної масштабованості та продуктивності.

Використовувалися такі системи та підсистеми:

- Mass Entity System: всі ігрові одиниці, включаючи робітників, солдатів та об'єкти, керовані штучним інтелектом, були повністю реалізовані за допомогою фреймворку масових об'єктів (Mass Entity Framework). Для визначення специфічних даних юнітів (наприклад, здоров'я, стан руху) та логіки поведінки були створені спеціальні MassFragments та MassProcessors;
- StateTree: модульні ресурси StateTree були розроблені для підрозділів, керованих ШІ, і інкапсулювали поведінку, таку як збір ресурсів, будівництво будівель, патрулювання та бойові дії. Кожному ШІ-юніту було призначено власне дерево StateTree, щоб забезпечити гнучку, адаптивну поведінку;
- AI Mass Crowd: інтегрована в системи пересування юнітів, щоб забезпечити плавний пошук шляхів і групову навігацію без зіткнень і вузьких місць, навіть у масштабних битвах за участю сотень юнітів;
- World Partition: дозволяє динамічно завантажувати та вивантажувати фрагменти місцевості залежно від місцезнаходження гравця, забезпечуючи ефективне використання пам'яті та створення масштабних мап без погіршення продуктивності;
- Gameplay Tags та Gameplay Ability System (GAS): використовується для призначення атрибутів юнітів та

динамічного управління взаємодією. Хоча повна реалізація GAS не була пріоритетом, базові магічні здібності (наприклад, вогняна куля, заклинання зцілення) були реалізовані для перевірки системи.

*Реалізація основних систем*

Система управління юнітами: гравці можуть вибирати окремі юніти або групи юнітів, використовуючи стандартні механізми клацання та перетягування у стилі RTS. Вибрані юніти реагують на команди руху та атаки, а внутрішня черга наказів управляється за допомогою Mass.

```
80
81  void UUnitCommandSubsystem::MoveCommand(const TArray<FMassEntityHandle>& Units, const FVector& Destination) const
82  {
83      for (const FMassEntityHandle& Unit : Units)
84      {
85          if (!EntityManager.IsEntityValid(Unit))
86          {
87              continue;
88          }
89
90          auto& MoveTarget = EntityManager.GetFragmentDataChecked<FMassMoveTargetFragment>(Unit);
91          MoveTarget.Center = Destination;
92          MoveTarget.IntentAtGoal = EMassMovementAction::Move;
93      }
94  }
95
```

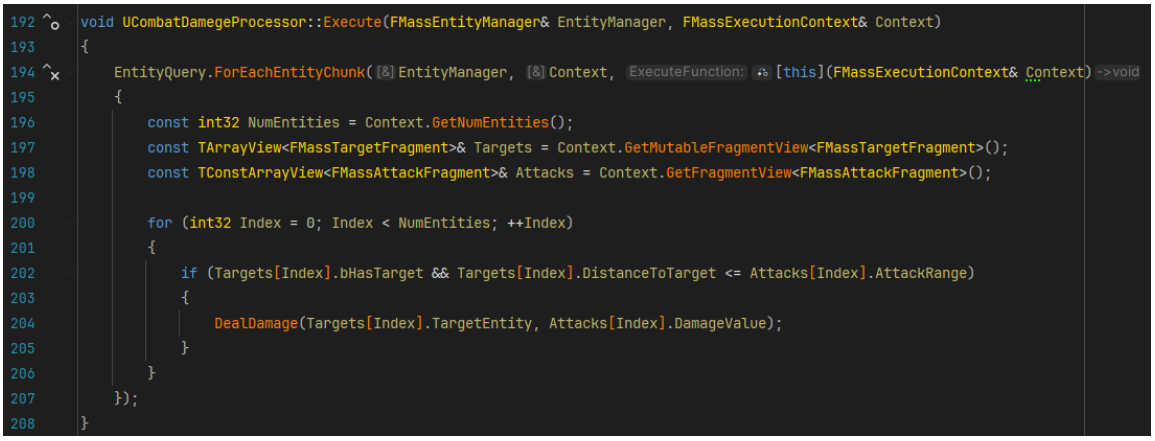
*Рисунок 3.3 – Лістинг коду: команда руху юнітів*

*Джерело: розроблено автором*

Система збору та будівництва ресурсів: робітники використовують поведінку, керовану BehaviorTree, щоб автоматично знаходити найближчий вузол ресурсів, збирати їх і доставляти до центру міста. Розміщення будівель ініціюється гравцем і узгоджується з обмеженнями на місцевості.

Виробнича система: виробничі будівлі (наприклад, казарми) управляються як масові об'єкти, здатні породжувати нові юніти після отримання виробничих замовлень.

Бойова система: юніти атакують ворожі юніти автоматично, коли знаходяться в межах досяжності, на основі параметрів прицілу та атаки, визначених у MassFragments.



```cpp
void UCombatDamegeProcessor::Execute(FMassEntityManager& EntityManager, FMassExecutionContext& Context)
{
    EntityQuery.ForEachEntityChunk([&]EntityManager, [&]Context, ExecuteFunction: [this](FMassExecutionContext& Context) ->void
    {
        const int32 NumEntities = Context.GetNumEntities();
        const TArrayView<FMassTargetFragment>& Targets = Context.GetMutableFragmentView<FMassTargetFragment>();
        const TConstArrayView<FMassAttackFragment>& Attacks = Context.GetFragmentView<FMassAttackFragment>();

        for (int32 Index = 0; Index < NumEntities; ++Index)
        {
            if (Targets[Index].bHasTarget && Targets[Index].DistanceToTarget <= Attacks[Index].AttackRange)
            {
                DealDamage(Targets[Index].TargetEntity, Attacks[Index].DamageValue);
            }
        }
    });
}
```

*Рисунок 3.4 – Лістинг коду: Mass Processor для оцінки поведінки юнітів під час атаки на основі параметрів націлювання та атаки*

*Джерело: розроблено автором*

Логіка перемоги/поразки: прості умови перевіряють, чи знищені критичні споруди гравця, або всі його юніти, викликаючи стан кінця матчу.

*Адаптивний ШІ-супротивник*

Адаптивний ШІ-супротивник у Warmage був розроблений за допомогою комбінації Mass Entities, дерев поведінки на основі StateTree та фреймворків ШІ Unreal Engine 5. Метою проектування було створення ШІ, який не просто виконує жорсткі сценарії, а динамічно реагує на мінливі умови поля бою, надаючи гравцям складного і непередбачуваного супротивника.

Система ШІ автономно виконує наступні дії під час ігрового процесу:

- збір ресурсів: робочі юніти, керовані ШІ, автоматично розставляють пріоритети і збирають доступні ресурси, адаптуючись до виснаження ресурсів шляхом переміщення до вторинних ресурсних точок;
- будівництво будівель: на основі зібраних ресурсів і стратегічних потреб ШІ вибирає і будує виробничі будівлі (наприклад, казарми) за розрахованою схемою, щоб оптимізувати оборону і виробничі можливості юнітів;

- виробництво юнітів: ШІ активно керує виробничими чергами, змінюючи склад своєї армії залежно від виявлених загроз і можливостей;
- наступальна стратегія: коли накопичується достатньо сил, ШІ ініціює атаки на базу гравця, причому час і сила атаки визначаються на основі аналізу видимих військових сил гравця і розширення бази;
- оборонна адаптація: у відповідь на агресію гравця ШІ динамічно перерозподіляє ресурси в бік оборонних споруд, або збільшення виробництва юнітів, зміщуючи фокус між захистом і нападом залежно від умов на полі бою.

Перемикання поведінки відбувається в реальному часі за допомогою модульної конфігурації дерева стану (StateTree). Кожна гілка StateTree представляє певний стратегічний стан (наприклад, розширення ресурсів, укріплення бази, підготовка наступального удару). Умови переходу між гілками оцінюються безперервно на основі метрик стану гри, таких як запаси ресурсів, кількість юнітів, близькість ворога та контроль над полем бою.

Важливо, що стратегічна поведінка ШІ не є жорстко закодованою для конкретних карт або стартових умов. Замість цього ШІ використовує оцінку даних про навколишнє середовище та ворога в реальному часі, щоб динамічно коригувати свою тактику. Це дозволяє ШІ поводитися по-різному в різних матчах навіть за однакових початкових умов, що значно підвищує реіграбельність і робить кожну зустріч зі штучним інтелектом унікальним стратегічним викликом.

*Реалізація багатокористувацької гри*

Базова багатокористувацька функціональність була реалізована за допомогою стандартних механізмів реплікації акторів Unreal Engine. Гравці можуть приймати або приєднуватися до матчів по локальній мережі через IP-адресу.

Реплікація стану виконується автоматично для юнітів, ресурсів та команд гравців, забезпечуючи синхронізацію стану гри між клієнтами та хостом.

*Процедурна генерація мапи*

Базова процедурна генерація мапи була реалізована для автоматичного розміщення стартових позицій гравців, вузлів ресурсів та особливостей місцевості. Генератор використовує випадкові шаблони для забезпечення варіативності між матчами, зберігаючи при цьому збалансовані стартові умови.

Система була розроблена для безперешкодної роботи з World Partition, розбиваючи завдання розміщення на частини на основі світових координат.

*Інтерфейс користувача (UI)*

HUD гравця:

- поточні ресурси;
- стан вибраних юнітів (здоров'я, спеціальні здібності);
- основну міні-мапу та командну панель.

Головне меню дозволяє гравцям:

- почати новий матч;
- приєднатися до існуючої сесії;
- запустити хост сервер (до якої можуть приєднатися інші гравці);
- вийти з гри.

Мінімалістичний дизайн інтерфейсу був обраний для того, щоб надати пріоритет функціональності та завершеності гри над візуальною поліровкою.

*Зберігання та управління даними*

Warmage не використовує постійну базу даних. Всі ігрові дані зберігаються в пам'яті під час виконання за допомогою структур даних та систем реплікації UE5. Прогрес гравця, стани юнітів та результати матчів існують лише в межах активної ігрової сесії.

*Інтеграція та конвеєри збірки*

За допомогою Jenkins було налаштовано конвеєри безперервної інтеграції, які виконували такі дії:

- компілювати найновіші коміти коду (compilation);
- готувати ігрові ресурси (cooking);
- пакувати ігрові збірки (packaging);
- запускати автоматизовані тести (наприклад, завантаження мапи, перевірка поведінки ШІ) (automation tests).

Артефакти збірки автоматично зберігалися на окремій машині для цілей розгортання або тестування.

Проект Warmage було реалізовано з використанням поєднання передових технологій Unreal Engine 5 та сучасних практик розробки програмного забезпечення. Результатом стала повнофункціональна стратегія в реальному часі, яка має основний геймплей RTS, адаптивну поведінку ШІ, масштабовану продуктивність завдяки системі масових об'єктів та надійну багатокористувацьку функціональність. Розроблені системи демонструють високий рівень технічного та дизайнерського виконання, повністю відповідаючи поставленим цілям проекту.

## 3.2 Тестування програмного продукту

Ефективне тестування було критично важливим етапом у розробці Warmage, спрямованим на забезпечення надійності, коректності та продуктивності всіх впроваджених систем. Цей розділ містить огляд проведеного функціонального та модульного тестування і оцінки продуктивності. Оцінювання поєднувало в собі ручне, автоматизоване та безперервну інтеграцію, щоб максимізувати покриття та перевірку.

*Функціональне тестування*

Функціональне тестування проводилося для перевірки того, що всі задекларовані ігрові функції та системи працюють відповідно до проектних специфікацій. Було використано як ручне, так і автоматизоване за допомогою Unreal Engine's Automation Framework та конвеєрів Jenkins CI.

*Методологія функціонального тестування*

Сеанси ручного оцінювання включали виконання попередньо визначених сценаріїв, які охоплювали весь цикл ігрового процесу, включаючи управління юнітами, збір ресурсів, побудову структур, бої, поведінку ШІ та управління багатокористувацькими сесіями.

Автоматизовані функціональні перевірки були інтегровані в конвеєр збірки Jenkins, щоб гарантувати, що кожна нова збірка підтримувала базову цілісність ігрового процесу без регресії.

*Контрольний список функціонального тестування*

Наступний контрольний список був створений для структурування процесу функціонального оцінювання працездатності:

1) вибір та управління юнітами:
   - вибір одного юніта за допомогою лівої кнопки миші;
   - групове виділення юнітів за допомогою перетягування поля виділення;
   - подача команд переміщення до вибраних юнітів;
   - юніти реагують на команди переміщення і досягають призначеного місця;
2) збір ресурсів:
   - робітники правильно збирають ресурси з ресурсних вузлів;
   - ресурси доставляються до центру міста;
   - лічильники ресурсів точно оновлюються в HUD;
3) система будівель:
   - гравці можуть розміщувати нові споруди на місцевості;
   - будівництво будівель прогресує з часом;
   - після завершення будівництва будівлі стають доступними для використання;
4) виробнича система:
   - будівлі (наприклад, казарми) можуть виробляти нові бойові одиниці;

- вироблені юніти з'являються у визначених точках відтворення;

5) бойова система:
   - юніти автоматично атакують ворожих юнітів і споруди в межах досяжності;
   - значення здоров'я відповідно зменшується при отриманні ушкоджень;
   - юніти та споруди знищуються, коли рівень здоров'я досягає нуля;
6) умови перемоги та поразки:
   - перемога зараховується після знищення всіх критично важливих споруд противника;
   - поразка настає, коли знищуються структури, або юніти гравця;
7) адаптивна поведінка ШІ:
   - ШІ-супротивник збирає ресурси, будує споруди, виробляє юнітів і атакує гравця;
   - ШІ адаптує тактику на основі військової сили та експансії гравця;
8) багатокористувацька функціональність:
   - успішне створення LAN-сесій;
   - успішне приєднання до LAN-сесій інших гравців;
   - синхронізація стану гри між клієнтами та сервером;
9) інтерфейс та система меню:
   - пункти головного меню працюють коректно (почати матч, приєднатися до матчу, створити сесію, вийти з гри);
   - ігровий HUD динамічно оновлює лічильники ресурсів, деталі вибору юнітів та інформацію про міні-мапу.

*Результати функціонального тестування*

Усі критичні ігрові системи пройшли функціональне тестування. Незначні неблокуючі невідповідності у користувацькому інтерфейсі були виявлені під час ранніх ітерацій, але були виправлені перед фінальним

випробуванням. Обробка багатокористувацьких сесій виявилася стабільною в умовах локальної мережі.

*Модульне тестування*

Модульне тестування було зосереджене на ізоляції та перевірці коректності окремих критично важливих систем та компонентів. Проведення випробувань проводилося за допомогою Unreal Engine's Automation Testing Framework, що дозволяє включати автоматизовані модульні перевірки в конвеєр Jenkins CI.

*Методологія модульного тестування*

Юніт-тести були розроблені для перевірки поведінки основних логічних компонентів незалежно від повного ігрового контексту. Ці перевірки допомогли виявити логічні помилки на ранніх стадіях і забезпечити стабільність компонентів під час ітерацій.

*Перевірено ключові компоненти:*

1) компонент руху:
    - перевірка того, що накази на переміщення підрозділів правильно оновлюють цілі переміщення;
    - перевірка, чи дотримуються підрозділи меж місцевості та чи уникають невірних цілей;
2) логіка збору ресурсів:
    - робітничі підрозділи збирають ресурси лише з дійсних вузлів ресурсів;
    - доставка ресурсів точно оновлює загальну кількість ресурсів гравця;
3) перевірка розміщення будівель:
    - гарантує, що будівлі можна розміщувати лише на дозволених ділянках місцевості;
    - логіка розміщення правильно обробляє близькість до інших споруд та меж мапи;
4) система черги виробництва:

- будівлі правильно стають у чергу і створюють нові юніти після отримання виробничих замовлень;
- таймери виробництва працюють з очікуваною тривалістю;

5) переходи між станами ШІ-машини:
   - агенти ШІ коректно переходять між станами збору, будівництва та атаки;
   - динамічна адаптація до дій противника (наприклад, активація режиму оборони) відбувається без перерв;
6) багатокористувацька синхронізація:
   - перевірки підтвердили, що ключові дані (позиції юнітів, кількість ресурсів, дії гравців) коректно реплікуються між сервером і клієнтами.

*Результати модульного тестування*

Усі цільові системи пройшли критерії модульного тестування. Під час процесу випробовування переходу ШІ було виявлено невелику кількість крайніх випадків (наприклад, спроба ШІ почати будівництво до того, як буде зібрано достатньо ресурсів), які згодом були вирішені шляхом уточнення умов переходу StateTree.

*Тестування продуктивності*

Тестування продуктивності було проведено для оцінки масштабованості та ефективності ігрових систем Warmage, що забезпечують безперебійний ігровий процес навіть під час масштабних битв.

*Методологія тестування продуктивності*

Включено тести продуктивності:

- моніторинг частоти кадрів (FPS) при різному ігровому навантаженні;
- стрес-випробування масштабованості Mass Entity System;
- спостереження за ефективністю потокової передачі даних World Partition під час навігації по великій мапі;

- вимірювання мережевої затримки та синхронізації в багатокористувацьких сесіях.

Оцінювання продуктивності проводилися на машині для розробки середнього класу (Windows 11, Intel Core i5 14600KF, 32 ГБ RAM, NVIDIA RTX 4070), а також на додатковому пристрої з нижчими характеристиками класу (Windows 11, Intel Core i5 8400, 16 ГБ RAM, NVIDIA GTX 1060 3GB), щоб забезпечити прийнятну продуктивність на різних рівнях апаратного забезпечення.

*Ключові показники продуктивності*

1) стабільність частоти кадрів:
   - в одиночних боях за участю 400-600 юнітів середня частота кадрів у секунду перевищувала 90;
   - у великих боях (800+ юнітів) середня частота кадрів була в межах 55-70 кадрів в секунду;
2) масштабованість системи для масових об'єктів:
   - масові процесори обробляли до 1000 активних юнітів без критичного зниження продуктивності;
   - спеціальні процесори MassProcessors динамічно оптимізували частоту кадрів залежно від ігрового контексту;
3) потокове передавання світових розділів:
   - безперебійне завантаження та вивантаження світових розділів було досягнуто без помітних затримок;
   - час потокового передавання на комірку залишався меншим за 100 мс у звичайних ігрових умовах;
4) багатокористувацька синхронізація:
   - затримка між сервером і клієнтами залишалася нижче 50 мс в умовах локальної мережі;
   - під час випробувань не було виявлено суттєвих проблем із десинхронізацією.

*Результати тестування продуктивності*

Warmage продемонстрував відмінні характеристики масштабованості та продуктивності, підтвердивши ефективність використання системи масових сутностей та світових розділів. Було зроблено незначні оптимізації для покращення ефективності симуляції натовпу, особливо в боях, де кількість юнітів перевищує 400.

Тестування підтвердило, що Warmage відповідає всім заявленим функціональним вимогам, демонструє надійність компонентів за допомогою модульного оцінювання і зберігає високу продуктивність під час великих ігрових навантажень. Поєднання ручного, автоматизованих функціональних та модульних тестів і оцінки продуктивності забезпечило комплексну перевірку якості, готовності та надійності програмного продукту.

Загалом, результати етапу тестування підтверджують, що Warmage є повністю функціональною, стабільною та масштабованою RTS-грою, здатною забезпечити захопливий та технічно якісний ігровий досвід.

### 3.3 Використання програмного продукту

Розроблений програмний продукт Warmage пропонує користувачам можливість брати участь у стратегічній грі в режимі реального часу проти адаптивного ШІ-супротивника, або в багатокористувацькому середовищі. Цей розділ містить опис основних можливостей програмного продукту та слугує базовим керівництвом користувача.

*Запуск гри*

Гра запускається за допомогою окремого виконуваного файлу, скомпільованого для платформи Windows. Після запуску програми користувачеві відкривається Головне меню, яке містить наступні опції:

- битва зі штучним інтелектом - розпочати новий однокористувацький матч проти адаптивного супротивника зі штучним інтелектом;
- приєднатися до гри - приєднатися до існуючої багатокористувацької сесії;

- стати хостом сервера – запускає сесію без ШІ ворогів до якої можуть приєднатися інші гравці;
- вийти - вийти з гри.

*Основні можливості ігрового процесу*

Керування юнітами: гравці керують юнітами за допомогою стандартної системи введення стратегії в реальному часі:

- клацання лівою кнопкою миші вибирає один юніт, або взаємодіє з UI;
- клацання лівою кнопкою миші та перетягування створює поле вибору для вибору декількох юнітів;
- клацання правою кнопкою миші дає команди руху, або атаки для вибраних юнітів.

Робітникам потрібно вручну давати команди збирати ресурси. Вибравши робітника і клацнувши правою кнопкою миші на вузол з ресурсом, робітник автоматично збиратиме ресурси і транспортуватиме їх до найближчого центру міста.

Система будівництва: гравці можуть будувати нові споруди:

- вибравши потрібний тип будівлі з доступних варіантів за допомогою UI;
- розмістити будівлю, клацнувши лівою кнопкою миші на відповідній ділянці мапи.

Будівлі дозволяють виробляти нові юніти та розширюють стратегічні можливості під час матчу.

Управління ресурсами: зібрані ресурси накопичуються і відображаються у верхній частині HUD. Ресурси необхідні для будівництва будівель і виробництва юнітів.

Бойова система: бойові одиниці автоматично атакують ворожих юнітів та споруди, як тільки вони опиняються в зоні досяжності. Смужки здоров'я над юнітами та спорудами вказують на залишок їхньої міцності.

Умови перемоги та поразки: матч виграється шляхом знищення всіх критично важливих ресурсів противника, зокрема, його міських центрів та робітничих підрозділів. Гра закінчується екраном перемоги, або поразки, як тільки одна зі сторін виконає ці умови.

*Багатокористувацький ігровий процес*

Гравці можуть брати участь у багатокористувацьких матчах через локальну мережу, виконавши такі дії:

- гравець-хост починає нову ігрову онлайн сесію;
- інші гравці вибирають «Приєднатися» до гри в головному меню;

Ігровий процес відбувається подібно до одиночної гри, де кожен гравець керує своєю базою, економікою та армією в режимі реального часу.

Багатокористувацькі сесії синхронізують позиції юнітів, кількість ресурсів та ігрові події, щоб забезпечити узгодженість ігрового процесу для всіх учасників.

*Огляд інтерфейсу користувача*

Ігровий HUD надає гравцеві ключову інформацію:

- панель ресурсів - відображає поточну кількість зібраних ресурсів;
- інформація про вибір юнітів - показує деталі здоров'я та статусу вибраних юнітів;
- командна панель - забезпечує швидкий доступ до команд пересування та будівництва;
- міні мапа - розташована в кутку екрану, міні мапа пропонує огляд дослідженої місцевості, позицій юнітів і ключових споруд в реальному часі.

Інтерфейс користувача розроблений таким чином, щоб залишатися мінімалістичним, але функціональним, забезпечуючи гравцям негайний доступ до всієї необхідної інформації без надмірного візуального безладу.

Warmage надає гравцям доступну та захопливу стратегію в реальному часі. Незалежно від того, чи б'єтеся ви з адаптивним ШІ, чи змагаєтеся з іншими гравцями через локальну мережу, користувачі взаємодіють з

інтуїтивно зрозумілим управлінням, будують великі бази, керують економічними ресурсами, командують арміями і прагнуть до стратегічної перемоги на динамічно створюваних полях битв.

**Висновки до розділу 3**

У третьому розділі кваліфікаційної роботи описано практичну реалізацію проекту Warmage - масштабованої та високопродуктивної гри в жанрі стратегії в реальному часі (RTS). Під час розробки були використані сучасні технології та інструменти, включаючи Unreal Engine 5, C++, Mass Entity System, StateTree, World Partition та Gameplay Ability System. Етап реалізації охоплював усі основні ігрові системи, такі як управління юнітами, управління ресурсами, будівництво, бойова механіка, адаптивна поведінка штучного інтелекту та базова багатокористувацька функціональність.

Тестування, що поєднувало ручне ігрове, автоматизоване функціональне та модульне, а також оцінку продуктивності, підтвердило коректність, стабільність та масштабованість розроблених систем. Перевірка функціональності підтвердила, що всі задекларовані функції працюють за призначенням. Юніт-тестування забезпечило надійність основних ігрових компонентів, а випробовування продуктивності продемонструвало високу масштабованість під час виконання масштабних ігрових сценаріїв.

В результаті розроблений програмний продукт повністю відповідає заявленим вимогам та цілям, забезпечуючи стабільну, функціональну та захоплюючу гру в RTS. Успішне завершення етапів реалізації та випробувань створює міцний фундамент для практичних результатів кваліфікаційної роботи.

# ВИСНОВКИ

В результаті досліджень, проведених в рамках даної кваліфікаційної роботи, було спроектовано, реалізовано та протестовано повноцінну стратегічну гру в реальному часі Warmage. Проект відповідає поставленим цілям, демонструючи масштабовану, ефективну та адаптивну RTS систему з використанням сучасних технологій Unreal Engine 5.

Основні наукові та практичні результати роботи полягають у наступному:

- розроблено модульну та масштабовану архітектуру гри з використанням фреймворків Mass Entity System та StateTree, що дозволяє ефективно керувати тисячами сутностей та створювати динамічні моделі поведінки ШІ;
- успішно інтегровані механізми адаптивного штучного інтелекту, що дозволило ШІ-супротивникам гнучко реагувати на дії гравців та ігрові події;
- багатокористувацька функціональність проекту була підтверджена успішною синхронізацією ігрових станів через LAN-з'єднання, що підтвердило готовність мережевої моделі до змагального ігрового процесу в реальному часі;
- випробовування продуктивності продемонструвало здатність системи зберігати стабільність і масштабованість при значних ігрових навантаженнях, підтвердивши правильність архітектурних рішень, прийнятих під час розробки.

На основі отриманих результатів пропонуються наступні практичні рекомендації та перспективи застосування:

- архітектура проекту Warmage може бути використана як основа для розширення до повномасштабної розробки RTS ігор з мінімальними структурними змінами;

- реалізовані фреймворки та патерни проектування можуть бути застосовані в освітніх цілях у сферах розробки масштабованих ігрових систем, адаптивного проектування ШІ та управління великомасштабними симуляціями;
- подальший розвиток проекту може включати інтеграцію більш досконалих методів планування ШІ, вдосконалену багатокористувацьку систему підбору гравців та ширші набори ігрових функцій.

Таким чином, завдання, окреслені на початку роботи, були повністю виконані, а отримані результати мають практичне значення для розробки сучасних, масштабованих стратегічних ігор у реальному часі.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. Epic Games. Unreal Engine 5 documentation. URL: https://docs.unrealengine.com/
2. Buckland M. AI and Game Development. – Cambridge: MIT Press, 2005. – 512 с.
3. Nystrom R. Game Programming Patterns. – San Francisco: Genever Benning, 2014. – 354 с.
4. Rogers S. Level Up! The Guide to Great Video Game Design. – New York: Wiley, 2014. – 552 с.
5. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston: Addison-Wesley, 1995. – 395 с.
6. Smed J., Hakonen H. Algorithms and Networking for Computer Games. – New York: Wiley, 2006. – 280 с.
7. Millington I., Funge J. Artificial Intelligence for Games. – Boca Raton: CRC Press, 2009. – 872 с.
8. Lengyel E. Mathematics for 3D Game Programming and Computer Graphics. – Boston: Cengage Learning, 2011. – 640 с.
9. Rollings A., Morris D. Game Architecture and Design. – Indianapolis: New Riders, 2003. – 816 с.
10. Gregory J. Game Engine Architecture. – Boca Raton: CRC Press, 2018. – 1240 с.
11. Steam. Age of Empires IV: Anniversary Edition. URL: https://store.steampowered.com/app/1466860/Age_of_Empires_IV_Anniversary_Edition/
12. Blizzard Entertainment. StarCraft II Official Website. URL: https://starcraft2.blizzard.com/en-us/
13. Steam. Cossacks 3 – Official Game Page. URL: https://store.steampowered.com/app/333420/Cossacks_3/

14. Catmull E., Wallace A. The Pixar Touch: The Making of a Company. – New York: Vintage, 2009. – 368 с.
15. Bishop M., Woolley D. Essential Mathematics for Games and Interactive Applications. – Cambridge: A K Peters, 2008. – 768 с.
16. Llopis N. Game Programming Gems. – Hingham: Charles River Media, 2000. – 704 с.
17. Schell J. The Art of Game Design: A Book of Lenses. – Boca Raton: CRC Press, 2019. – 600 с.
18. Microsoft. C++ Standard Library documentation. URL: https://learn.microsoft.com/en-us/cpp/standard-library/
19. Wikipedia. Entity Component System. URL: https://uk.wikipedia.org/wiki/Entity_component_system
20. Unreal Engine Community. AI and Mass Entity System Discussion. URL: https://forums.unrealengine.com/
21. Structurizr. C4 Model for Visualising Software Architecture. URL: https://c4model.com/
22. Kruchten P. Architectural Blueprints — The “4+1” View Model of Software Architecture // IEEE Software. – 1995. – Т. 12(6). – С. 42–50.
23. Gamma E., Beck K. Contributing to Eclipse: Principles, Patterns, and Plug-Ins. – Boston: Addison-Wesley, 2003. – 320 с.
24. Zhang H., Chien A. Parallelism and Scalability in Game Engines // Journal of Systems Architecture. – 2018. – Т. 89. – С. 43–55.
25. Koster R. A Theory of Fun for Game Design. – Scottsdale: Paraglyph, 2013. – 272 с.
26. Epic Games. Your First 60 Minutes with Mass. URL: https://dev.epicgames.com/community/learning/tutorials/JXMl/unreal-engine-your-first-60-minutes-with-mass
27. Megafunk. Mass Sample Project on GitHub. URL: https://github.com/Megafunk/MassSample?tab=readme-ov-file#mass-pm

28. Vrealmatic. MASS in Unreal Engine: Overview and Tutorials. URL: https://vrealmatic.com/unreal-engine/mass#mass
29. Epic Games. StateTree in Unreal Engine Documentation. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/state-tree-in-unreal-engine
30. Epic Games. Your First 60 Minutes with StateTree Tutorial. URL: https://dev.epicgames.com/community/learning/tutorials/lwnR/unreal-engine-your-first-60-minutes-with-statetree