

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри

_____ Едуард ЖАРИКОВ
(підпис) (ім'я прізвище)

“ ” _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного забезпечення
інформаційних систем»**

спеціальності «121 Інженерія програмного забезпечення»

на тему: Кросплатформна багатокористувацька сесійна реалізація
настільної гри «Монополія» на Unity

Виконав студент IV курсу, групи ІП-13
(шифр групи)

Бабіч Денис Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доц. каф. ІПП, к.е.н. Родіонов П. Ю. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст. викл. Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доц. каф. ІКС, к.т.н. Апенько Н. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____
(підпис)

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення
інформаційних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРИКОВ
(підпис) (ім'я прізвище)

“ ____ ” _____ 2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Бабічу Денису Володимировичу

(прізвище, ім'я, по батькові)

1. Тема проєкту Кроссплатформна багатокористувацька сесійна реалізація настільної гри «Монополія» на Unity

керівник проєкту Родіонов Павло Юрійович, доц. каф. ІІІ, к.е.н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «17» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки

1) Передпроєктне обстеження предметної області: аналіз предметної області, дослідження наявних рішень, моделювання бізнес-процесів, формулювання задачі проєкту.

2) Розроблення вимог до програмного забезпечення: визначення сценаріїв використання, формулювання функціональних і нефункціональних вимог.

3) Конструювання та розроблення програмного забезпечення: розроблення архітектури, обґрунтування вибору інструментів, створення програмного забезпечення, аналіз безпеки даних.

4) Аналіз якості та тестування: оцінка якості програмного продукту, опис методів тестування, представлення контрольного прикладу.

5) Розгортання та супровід програмного забезпечення: опис процесу розгортання системи, організація її подальшого супроводу.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна компонентів програмного забезпечення

3) Креслення вигляду екранних форм

4) Діаграма розгортання

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» березня 2025 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв'язання задачі	13.03.2025	
3	Постановка та формалізація задачі	27.03.2025	
4	Розробка інформаційного забезпечення	07.04.2025	
5	Алгоритмізація задачі	16.04.2025	
6	Обґрунтування вибору використаних технічних засобів	21.04.2025	
7	Розробка програмного забезпечення	13.05.2025	
8	Налагодження програми	15.05.2025	
9	Виконання графічних документів	22.05.2025	
10	Оформлення пояснювальної записки	26.05.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент

(підпис)

Денис БАБІЧ

(ініціали, прізвище)

Керівник

(підпис)

Павло РОДІОНОВ

(ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п'яти розділів, містить 38 таблиць, 30 рисунків та 34 джерела – загалом 76 сторінок.

Дипломний проєкт присвячений розробці кросплатформеної багатокористувацької мобільної гри, яка є цифровою адаптацією настільної гри «Монополія» з підтримкою до 5 гравців, реалізованої на ігровому рушії Unity.

Мета розробки це забезпечення автономної та стабільної роботи кросплатформної мобільної багатокористувацької гри, що є цифровою адаптацією настільної гри «Монополія», шляхом використання централізованої однорангової мережевої топології, яка мінімізує залежність від виділених серверів та забезпечує ефективну кросплатформену взаємодію між гравцями.

Об'єкт дослідження: сфера мобільних ігор та багатокористувацьких ігрових систем.

Предмет дослідження: розробка архітектури та програмної реалізації кросплатформеної багатокористувацької гри, включаючи аналіз мережевих топологій, рейтингових алгоритмів та інтеграцію з ігровими сервісами Unity.

У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення, змодельовано бізнес-процеси за допомогою BPMN-діаграм та сформульовано завдання проєкту.

Другий розділ присвячений розробленню функціональних та нефункціональних вимог, аналізу системних вимог та економічних показників.

Третій розділ охоплює конструювання програмного забезпечення, обґрунтування вибору мікросервісної архітектури, інтеграцію з зовнішніми сервісами та аналіз безпеки даних.

Четвертий розділ присвячений аналізу якості програмного продукту за метриками коду та тестуванню ключових функцій.

П'ятий розділ описує процес автоматичного розгортання та подальшого супроводу програмного забезпечення.

КЛЮЧОВІ СЛОВА: ГРА, МОБІЛЬНА ГРА, БАГАТОКОРИСТУВАЦЬКА ГРА, МІКРОСЕРВІСИ, UNITY, NETCODE, API, TLS, P2P, RPC, UDP, DTLS.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 38 tables, 30 figures, and 34 sources – in total 76 pages.

The purpose of the diploma project is the development of a cross-platform multiplayer mobile game, which is a digital adaptation of the board game "Monopoly" supporting up to 5 players, implemented using the Unity game engine.

The goal of the development is to ensure autonomous and stable operation of a cross-platform multiplayer mobile game, a digital adaptation of the board game "Monopoly," by utilizing a centralized peer-to-peer network topology that minimizes dependency on dedicated servers and ensures efficient cross-platform interaction between players.

Research object: the field of mobile games and multiplayer gaming systems.

Research subject: the development of the architecture and software implementation of a cross-platform multiplayer game, including the analysis of network topologies, rating algorithms, and integration with Unity gaming services.

The first chapter analyzes the subject area, reviews existing solutions, models business processes using BPMN diagrams, and formulates the project tasks.

The second chapter is dedicated to developing functional and non-functional requirements, analyzing system requirements, and evaluating economic indicators.

The third chapter covers software design, justifies the choice of a microservices architecture, integration with external services, and analysis of data security.

The fourth chapter is devoted to analyzing the quality of the software product based on code metrics and testing key functions.

The fifth chapter describes the process of automated deployment and ongoing maintenance of the software.

KEYWORDS: GAME, MOBILE GAME, MULTIPLAYER GAME, MICROSERVICES, UNITY, NETCODE, API, TLS, P2P, RPC, UDP, DTLS.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**Кросплатформна багатокористувацька сесійна реалізація настільної гри
«Монополія» на Unity
Технічне завдання
КПІ.ІП-1303.045490.01.91**

“ПОГОДЖЕНО”

Керівник проекту:

_____ Павло РОДІОНОВ

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Денис БАБІЧ

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ.....	3
2 ПІДСТАВА ДЛЯ РОЗРОБКИ.....	4
3 ПРИЗНАЧЕННЯ РОЗРОБКИ.....	5
4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	6
4.1 Вимоги до функціональних характеристик.....	6
4.1.1 Користувачького інтерфейсу.....	6
4.1.2 Для користувача:.....	10
4.1.3 Для адміністратора системи:.....	11
4.1.4 Додаткові вимоги:.....	11
4.2 Вимоги до надійності.....	11
4.3 Умови експлуатації.....	12
4.3.1 Вид обслуговування.....	12
4.3.2 Обслуговуючий персонал.....	12
4.4 Вимоги до складу і параметрів технічних засобів.....	12
4.5 Вимоги до інформаційної та програмної сумісності.....	12
4.5.1 Вимоги до вхідних даних.....	13
4.5.2 Вимоги до вихідних даних.....	13
4.5.3 Вимоги до мови розробки.....	13
4.5.4 Вимоги до середовища розробки.....	13
4.5.5 Вимоги до представленню вихідних кодів.....	13
4.6 Вимоги до маркування та пакування.....	13
4.7 Вимоги до транспортування та зберігання.....	13
4.8 Спеціальні вимоги.....	13
5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ.....	14
5.1 Попередній склад програмної документації.....	14
5.2 Спеціальні вимоги до програмної документації.....	14
6 СТАДІЇ І ЕТАПИ РОЗРОБКИ.....	15
7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ.....	16

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: кросплатформна багатокористувацька сесійна реалізація настільної гри «Монополія» на Unity.

Галузь застосування:

Наведене технічне завдання поширюється на розробку кросплатформеної багатокористувацької сесійної мобільної гри «Монополія», котра використовується для створення цифрової реалізації відомої настільної гри з підтримкою багатокористувацького режиму та призначена для користувачів мобільних пристроїв на платформах Android та iOS, які бажають грати разом із іншими гравцями в режимі реального часу незалежно від їх місцезнаходження.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки кросплатформної багатокористувацької сесійної реалізації настільної гри «Монополія» на Unity є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для створення кросплатформної мобільної гри, яка є цифровою адаптацією настільної гри «Монополія» з багатокористувацьким режимом, що забезпечує зручну взаємодію гравців на iOS та Android через однорангову мережеву топологію без постійної залежності від виділених серверів та сторонніх сервісів.

Метою розробки є забезпечення автономної та стабільної роботи кросплатформної мобільної багатокористувацької гри, що є цифровою адаптацією настільної гри «Монополія», шляхом використання централізованої однорангової мережевої топології, яка мінімізує залежність від серверної інфраструктури та сторонніх сервісів і забезпечує зручну взаємодію між гравцями незалежно від їхньої платформи та географічного розташування.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

- меню вибору публічного чи приватного лобі (рисунок 4.1);

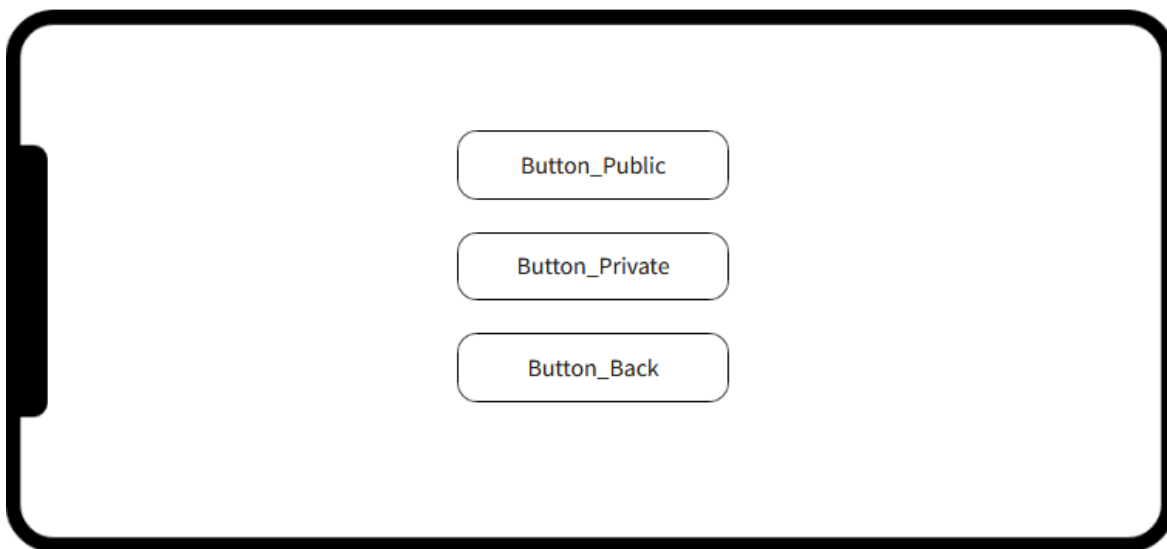


Рисунок 4.1 – Меню вибору типу лобі

- меню вибору способу підключення до приватного лобі у вигляді хоста, або клієнта (рисунок 4.2);

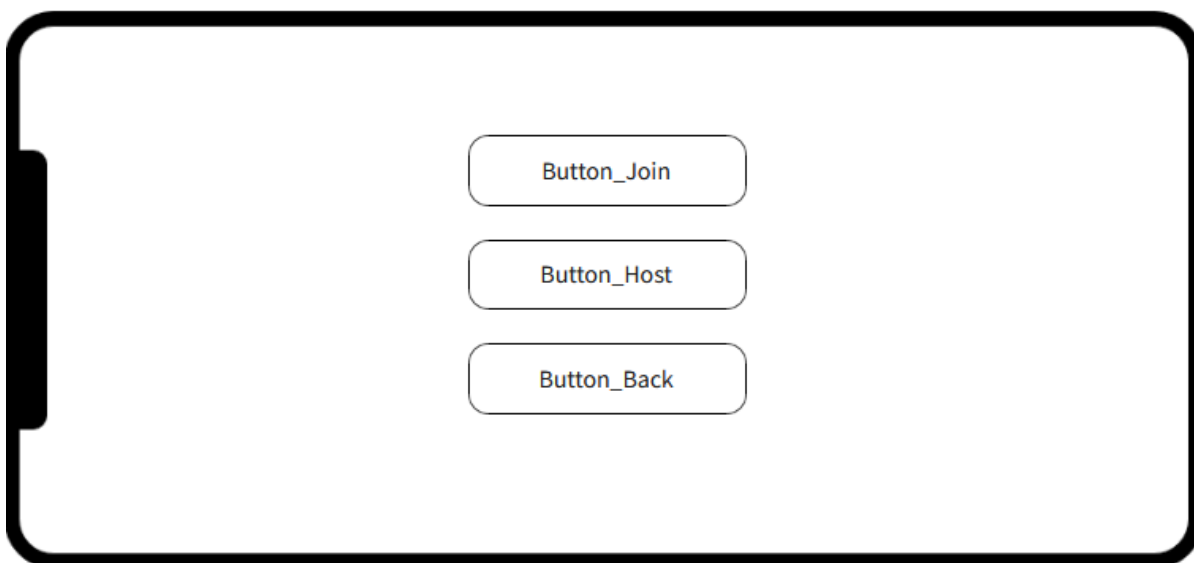


Рисунок 4.2 – Меню вибору типу підключення до приватного лобі

- меню введення коду ігрового лобі для підключення до приватної ігрової сесії (рисунок 4.3);

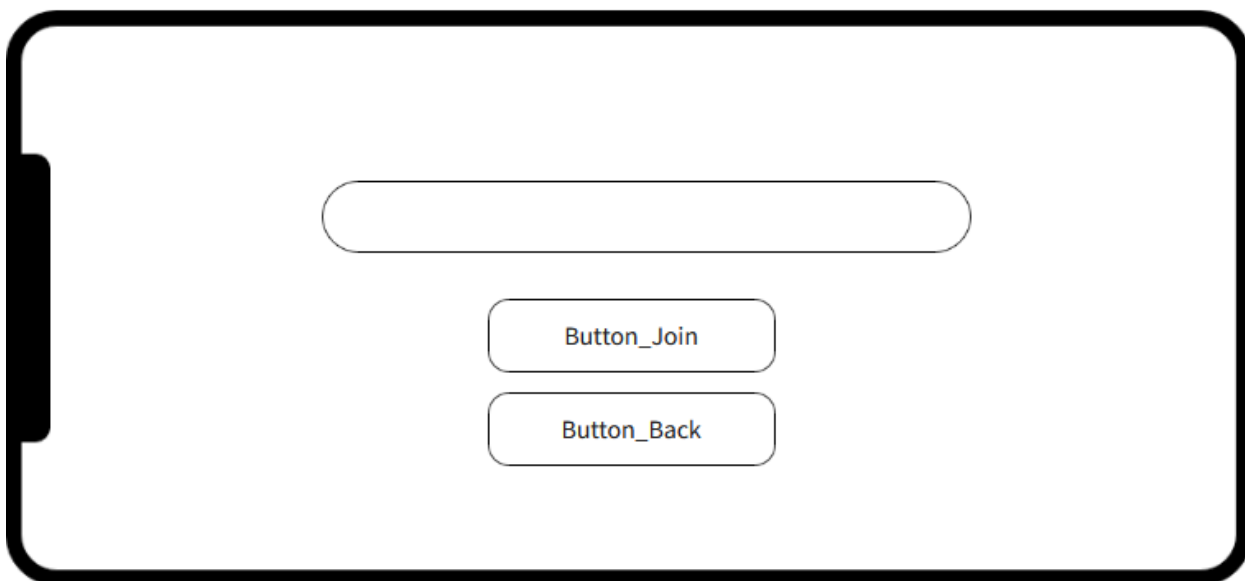


Рисунок 4.3 – Введення коду приватної ігрової сесії

- меню вибору способу підключення до публічного лобі у вигляді хоста, або пошуку лобі у випадку підключення клієнтом (рисунок 4.4);

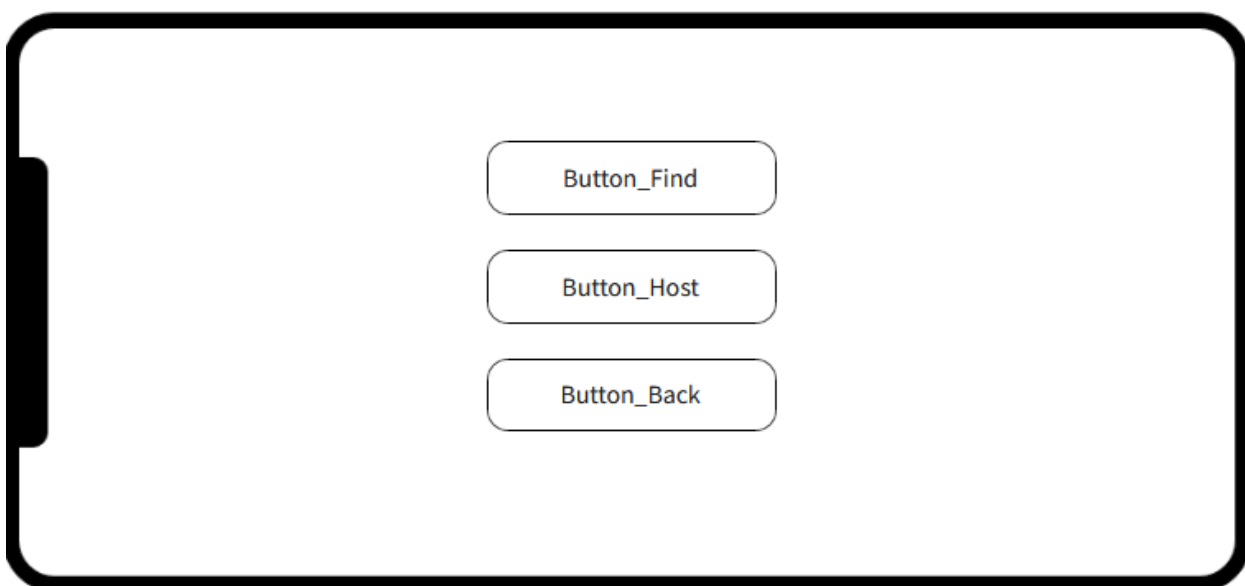


Рисунок 4.4 – Меню вибору типу підключення до публічного лобі

- стан графічного інтерфейсу у після успішного підключення до ігрового лобі, де можна побачити інших гравців (рисунок 4.5);

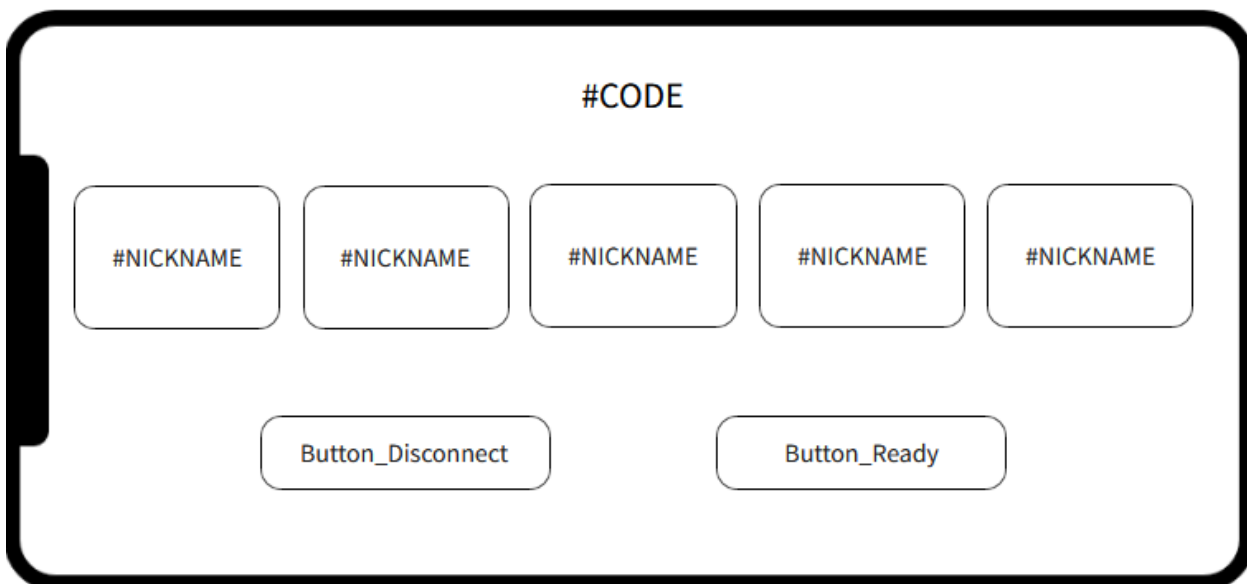


Рисунок 4.5 – Меню ігрового лобі

– стан графічного інтерфейсу у моменті отримання права ходу під час гри (рисунок 4.6);

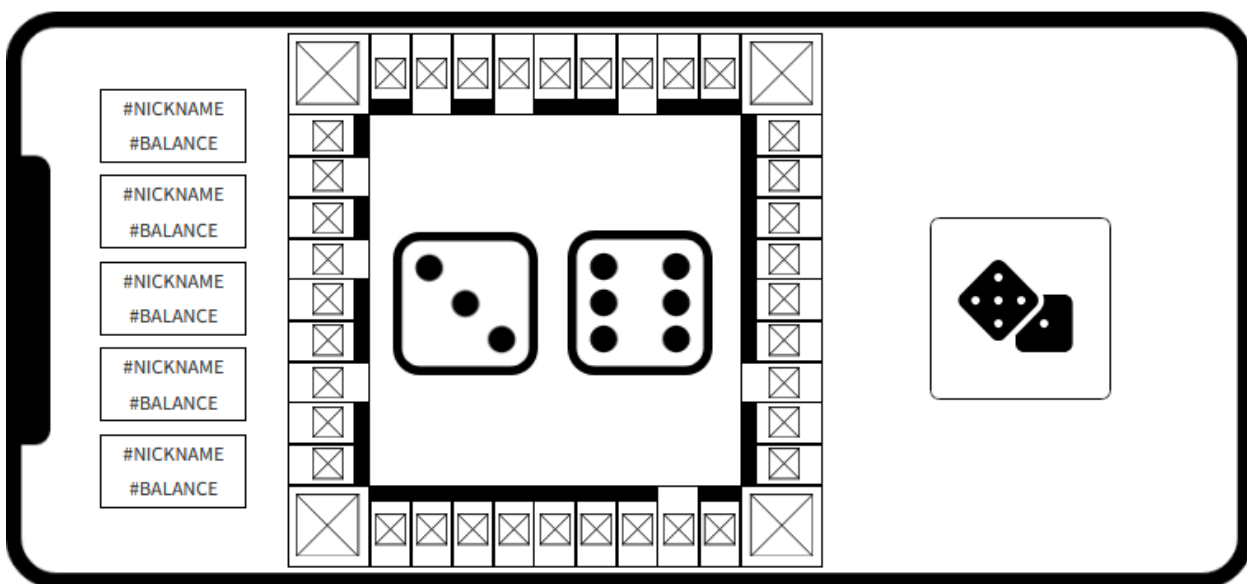


Рисунок 4.6 – Меню під час виконання ходу гравцем

– стан графічного інтерфейсу у після виконання ходу та приземлення на поле власності, яке ще нікому не належить. У такому випадку користувачу надається можливість вибору придбати конкретне поле, або відмовитися та завершити свій хід (рисунок 4.7);

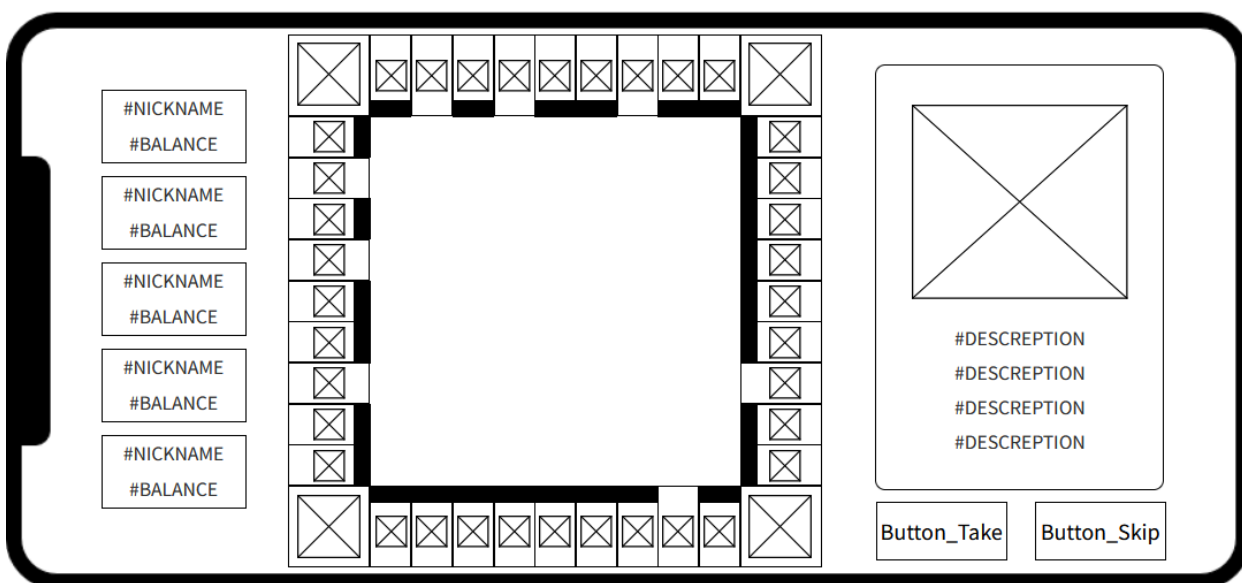


Рисунок 4.7 – Меню вибору дії над полем власності

– меню графічного інтерфейсу у після виконання ходу та приземлення на поле шансу, або податку. У такому випадку, користувачу виводиться діалогове меню з описом події, яка відбувається та кнопкою, яка завершує хід. (рисунок 4.8);

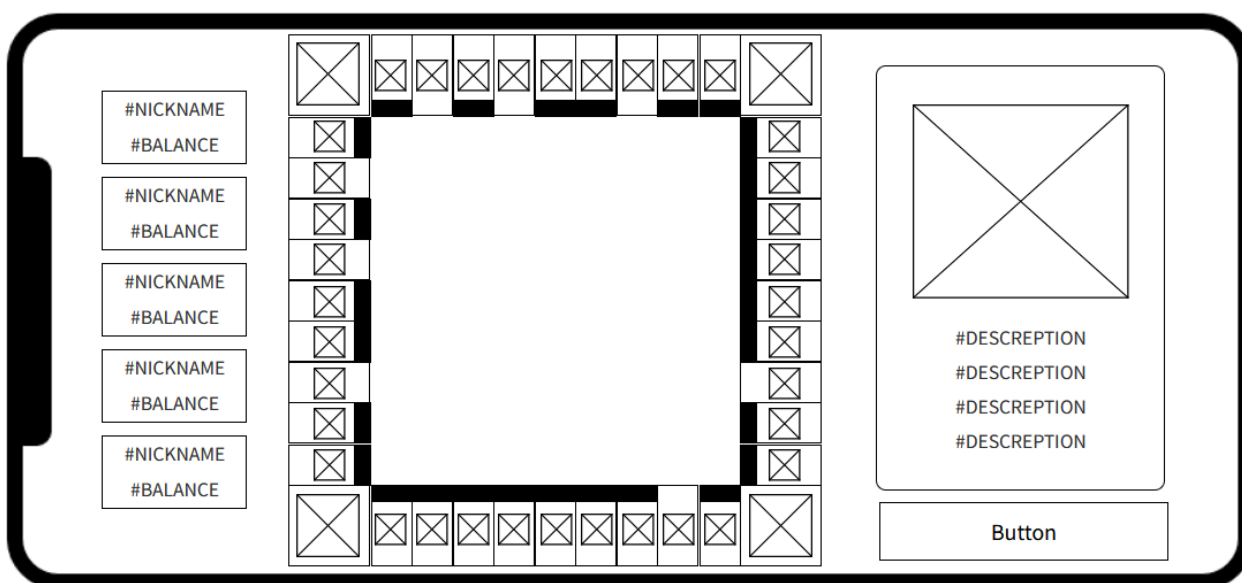


Рисунок 4.8 – Меню після приземлення на полу шанс, або податок

– меню графічного інтерфейсу у моменті, коли користувач приймає рішення про ініціалізацію процесу обміну з іншим гравцем та обирає відповідну кількість балансу й поля для обміну (рисунок 4.9);

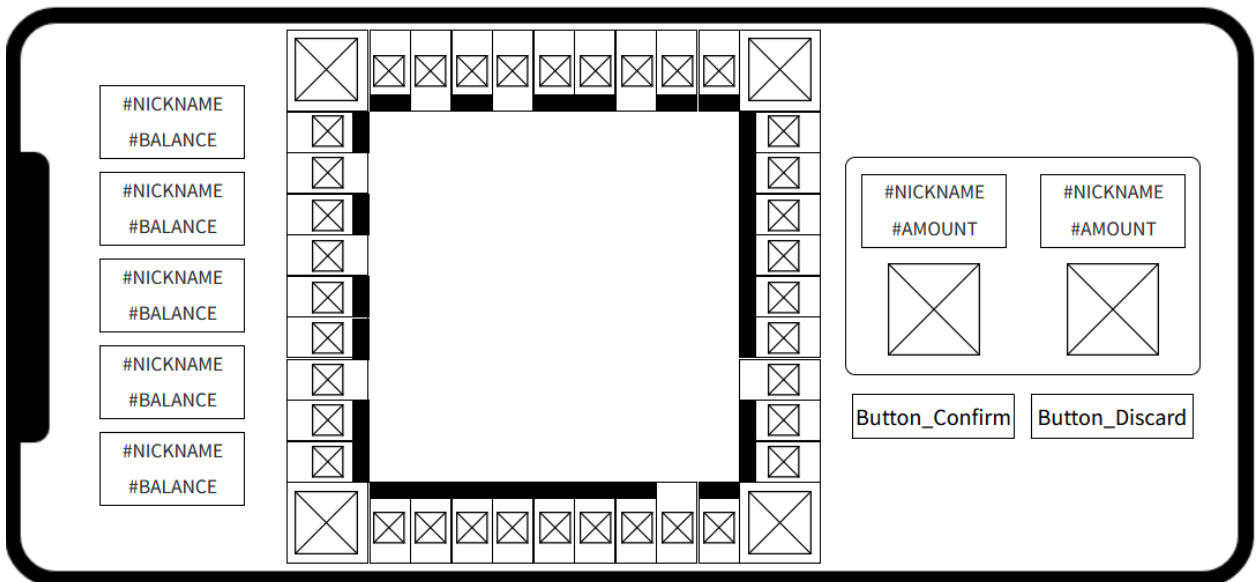


Рисунок 4.9 – Меню проведення обміну

– меню графічного інтерфейсу у моменті, коли користувач отримав пропозицію обміну від іншого гравця (рисунок 4.10).

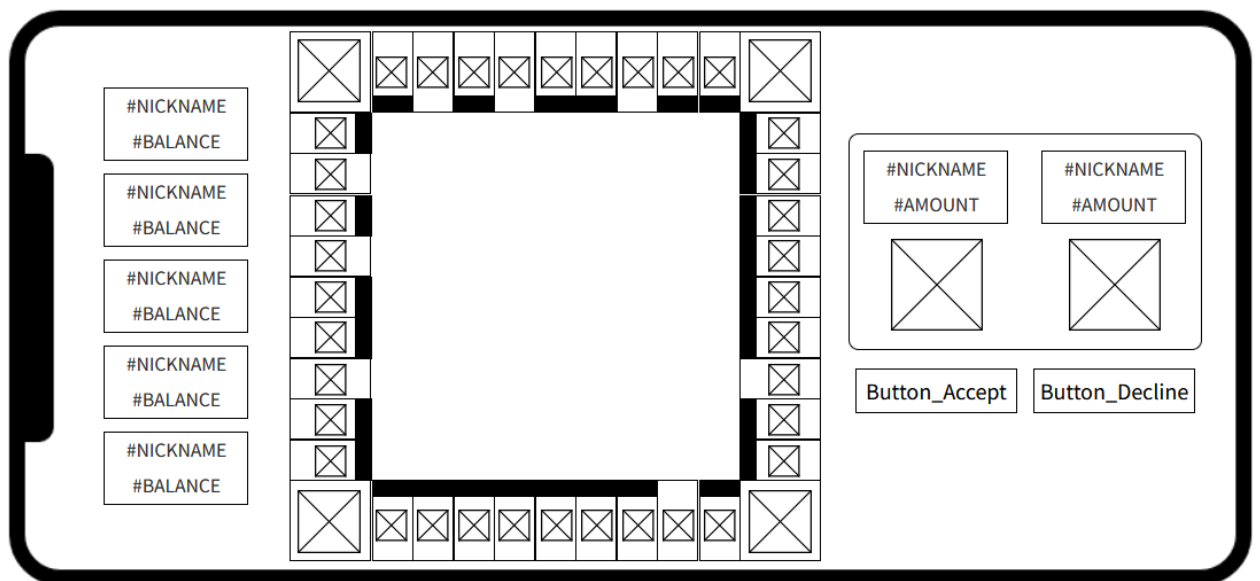


Рисунок 4.10 – Меню отриманої пропозиції обміну

4.1.2 Для користувача:

- перегляд профілю користувача у головному меню з відображенням інформації про обліковий запис, включаючи нікнейм та рейтинг;
- зміна імені користувача через редагування відповідного поля у профілі з оновленням даних локально та віддалено;

- пошук рейтингового матчу з автоматичним підключенням до доступного публічного лобі за наявності відповідного матчу;
- створення ігрової кімнати для публічного, або приватного лобі з підключенням користувача як хоста;
- підключення до приватного лобі шляхом введення коду сесії, наданого хостом лобі;
- кидання кубика для виконання ходу з переміщенням ігровим полем на відповідну кількість кроків;
- придбання вільного поля власності за наявності достатньої кількості ігрової валюти після приземлення на нього;
- покращення рівня монополії на полях, що належать гравцю, за умови володіння повною монополією та наявності достатньої кількості валюти;
- закладання власного поля для отримання компенсації через відповідне меню взаємодії;
- ініціювання обміну полями та/або ігровою валютою з іншим гравцем через вибір відповідних параметрів у панелі;
- перегляд, прийняття або відхилення пропозиції обміну, отриманої від іншого гравця, з оновленням ігрового стану;

4.1.3 Для адміністратора системи:

Вимоги до адміністратора системи не висуваються.

4.1.4 Додаткові вимоги:

- інтеграція локалізації інтерфейсу для кількох мов.

4.2 Вимоги до надійності

Для забезпечення надійності мережевої взаємодії використовується шифрування для захисту даних від несанкціонованого доступу. Серверна

перевірка дій користувача запобігає маніпуляціям і гарантує цілісність ігрового стану під час багатокористувацьких сесій.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Спеціальні умови до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно коректно функціонувати на мобільних пристроях під керуванням операційних систем Android та iOS.

Мінімальна конфігурація технічних засобів:

- процесор Snapdragon 450 1.8 ГГц, або Apple A10 Fusion 1.8 ГГц;
- об'єм ОЗП 2 ГБ;
- об'єм вбудованої пам'яті 1 ГБ вільного простору;
- підключення до мережі Інтернет зі швидкістю від 10 Мбіт/с.

Рекомендована конфігурація технічних засобів:

- процесор Snapdragon 675 2.4 ГГц, або Apple A12 Bionic 2.4 ГГц;
- об'єм ОЗП 4 ГБ;
- об'єм вбудованої пам'яті 2 ГБ вільного простору;
- підключення до мережі Інтернет зі швидкістю від 50 Мбіт/с.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем iOS (версія 14.0 і вище) та Android (версія 9.0 і вище).

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C#.

4.5.4 Вимоги до середовища розробки

Розробку виконано на ігровому рушії Unity Editor 6.0 з використанням середовища Microsoft Visual Studio Code 1.99.2.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код розробленої програми повинен бути розміщений у вигляді завершеного проєкту в репозиторії на платформі GitHub. Основні фрагменти коду, що реалізують функціональну частину, мають бути подані у документі «Текст програми».

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги до програмного продукту не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текст програми;
- програма та методика тестування;
- керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема структурна компонентів програмного забезпечення;
- схема розгортання програмного забезпечення та його модулів;
- креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	15.03	
2.	Розробка технічного завдання	24.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	12.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	25.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	14.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	15.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	17.05	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	23.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

**Пояснювальна записка
до дипломного проєкту**

на тему: **Кросплатформна багатокористувацька сесійна реалізація
настільної гри «Монополія» на Unity**

КПІ.ІП-1303.045490.02.81

ЗМІСТ

ВСТУП.....	5
1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.....	6
1.1 Постановка завдання дипломного проєктування.....	6
1.2 Аналіз предметної області.....	6
1.3 Аналіз існуючих рішень.....	9
1.3.1 Аналіз відомих програмних продуктів.....	9
1.3.2 Аналіз відомих алгоритмічних та технічних рішень.....	12
1.4 Аналіз та моделювання бізнес-процесів.....	14
Висновки до розділу.....	16
2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	18
2.1 Варіанти використання програмного забезпечення.....	18
2.2 Розроблення функціональних вимог.....	26
2.3 Розроблення нефункціональних вимог.....	31
2.4 Аналіз системних вимог.....	32
2.5 Аналіз економічних показників програмного забезпечення.....	32
2.6 Постановка завдання на розробку програмного забезпечення.....	35
Висновки до розділу.....	36
3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	38
3.1 Архітектура програмного забезпечення.....	38
3.2 Архітектурні рішення та обґрунтування вибору засобів розробки.....	40
3.3 Конструювання програмного забезпечення.....	46
3.4 Аналіз безпеки даних.....	50
Висновки до розділу.....	51
4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....	53
4.1 Аналіз якості ПЗ.....	53
4.2 Опис процесів тестування.....	54
4.3 Опис контрольного прикладу.....	62
Висновки до розділу.....	68

	3
5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	70
5.1 Розгортання програмного забезпечення.....	70
5.2 Супровід програмного забезпечення.....	72
Висновки до розділу.....	72
ВИСНОВКИ.....	73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	74
ДОДАТКИ.....	78
Додаток А – Звіт подібності.....	78

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	– Integrated Development Environment – інтегроване середовище розробки.
RPC	– Remote Procedure Call – віддалений виклик процедури.
UDP	– User Datagram Protocol – протокол дейтаграми користувача.
TCP	– Transmission Control Protocol – протокол управління передачею
TLS	– Transport Layer Security – протокол безпеки транспортного рівня.
DTLS	– Datagram Transport Layer Security – дейтаграми транспортного рівня безпеки.
OSI	– Open System Interconnection model – середовище зв'язку відкритих систем.
P2P	– Peer-to-Peer – однорангова мережа.
API	– Application Programming Interface – прикладний програмний Інтерфейс.

ВСТУП

У сучасному інформаційному світі ринок відеоігор з року в рік приваблює все більше нових гравців, а відповідно, й інвестицій. Однією з основних причин розвитку цього сегменту бізнесу є зростання загального рівня життя людства, а отже, й доступності нових технологій та пристроїв, зокрема, смартфонів, для користувачів з усього світу. Завдяки цьому, розробники отримують можливість створювати контент, що здатний охопити велику аудиторію та задовольнити запити та потреби найбільш вимогливих до платформи користувачів.

Монополія – одна з найпопулярніших настільних ігор, яка завоювала сотні мільйонів шанувальників з усього світу. Актуальність створення мобільної гри з багатокористувацьким режимом, яка є реалізацією монополії, полягає у відповідності сучасним світовим тенденціям, оскільки це рішення дозволяє гравцям з різних куточків світу грати разом, незалежно від їх місця розташування. Мобільна монополія з багатокористувацьким режимом має ряд переваг перед традиційною настільною грою, серед яких основними є простота та доступність в будь-який час і в будь-якому місці через відсутність необхідності фізичної присутності в одному приміщенні та наявності спеціальних фізичних ігрових елементів.

У даному дипломному проєкті буде розглянуто створення мобільної версії монополії з багатокористувацьким режимом, проведений аналіз існуючих рішень, та обґрунтована необхідність розробки нової мобільної монополії, яка відповідає вимогам сучасності та враховує актуальні тенденції в області розробки відеоігор.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Завдяки широкому розповсюдженню смартфонів, планшетів і доступу до Інтернету, відеоігри стали невід’ємною частиною дозвілля сотнів мільйонів користувачів у всьому світі. Особливої популярності набули мобільні ігри, що дозволяють грати в будь-який зручний момент, не вимагаючи присутності гравців в одному приміщенні та додаткового обладнання, яке необхідне для гри.

Однією з найвідоміших класичних настільних ігор є «Монополія» – стратегічна економічна гра, яка об’єднує елементи фінансового планування, торгівлі та управління ресурсами. Незважаючи на свою популярність та наявність певної кількості цифрових реалізацій, більшість з них не відповідають класичній реалізації гри, а натомість впроваджують зайвий функціонал та ігрові механіки, які надають нечесну перевагу гравцям в залежності від їх платоспроможності та сам доступ до цих ігор повністю залежить від доступності відповідних веб-ресурсів та сервісів компанії-розробника, що часто є незручним та неприйнятним через періоди недоступності у зв’язку з технічним обслуговуванням та іншими обставинами, які не залежать від користувача.

Всі вище перераховані причини створюють потребу у створенні сучасної, зручної та доступної мобільної кросплатформеної гри, яка є класичною реалізацією відомої настільної гри «Монополія», що дозволить гравцям взаємодіяти в реальному часі в незалежності від зовнішніх обставин, які не залежать від них.

1.2 Аналіз предметної області

Ігрова індустрія – це напрямок інформаційних технологій, який займається розробкою, виданням та розповсюдженням відеоігор. Вона є однією з найбільш динамічних та прибуткових напрямів ведення бізнесу. Сфера розробки відеоігор є однією з найбільших галузей розважальної індустрії, її

загальні річні прибутки перевершують доходи таких сегментів ринку, як фільми та музика. Далеко не в останню чергу такі високі показники забезпеченні розмаїттям напрямків цієї галузі, серед яких окремо можна визначити: сама розробка відеоігор під різні платформи, видавництво ігор, кіберспорт та багато інших. Такий різноманітний спектр напрямків свідчить про величезний вплив цієї індустрії на сучасну глобальну культуру [1].

Історія розробки відеоігор розпочинається з середини 20-го століття та визначається важливими трансформаціями, які охоплюють усі періоди її існування – від початку створення цієї галузі й до сучасності. Починаючи з перших відеоігор, сфера швидко розвивалася, зазнаючи вагомих змін на кожному етапі свого розвитку [2].

Перші експерименти зі створення відеоігор відбувалися завдяки окремим ентузіастам та науково-дослідницьким лабораторіям при університетах у 1950-1960-х роках, які відзначилися створенням гри під назвою «Spacewar!», яка була реалізована в 1962 році та вважається однією з перших повноцінних на той час відеоігор, яка і визначила початок цієї індустрії [3].

У 1970-80-х роках з'явилися перші аркадні ігрові автомати, а разом із ними були створені перші ігри для нових платформ, такі як «Pong» від компанії «Atari» та інші, що визначили новий етап в історії відеоігор. Втім, були всі шанси втрати цю галузь, оскільки у 1983 році світ сколихнула криза, що спричинила банкрутство багатьох компаній та надзвичайне падіння прибутків, що було пов'язане з перенасиченістю ринку іграми, переважно дуже посередньої якості. Але можна сказати, що компанія «Nintendo» вийшла з неї переможцем, бо саме вона, приблизно через десятиліття, у 1980-1990-х розробила перший виключно ігровий пристрій під назвою «Nintendo Entertainment System», який набув небаченої до цього популярності серед користувачів та ознаменував початок активного розповсюдження нових виключно ігрових пристроїв – консолей. Цей важливий етап не лише вплинув на подальший розвиток ігрових пристроїв, але й визначив стандарти ігрової

індустрії завдяки всесвітньовідомим класичним відеоіграм, які стали культовими і сформували основи сучасного ігрової культури [4].

З появою персональних комп'ютерів в 1980-90-х роках розробка ігор стала більш доступною для самостійних незалежних розробників, сприяючи зростанню їх кількості та поширенню ігор на різних платформах. Цей процес також сприяв і збільшенню кількості потенційних гравців, через появу нового масового багатозадачного пристрою. Також у 1990-2000-х роках ігрова індустрія пережила перехід до 3D-графіки, що зробило ігри більш реалістичними та вимогливими до компонентів пристроїв. Окремо варто відзначити появу доступного інтернету, який дав можливість легше розповсюджувати свої розробки та завдяки якому з'явилися перспективи для розвитку багатокористувацьких онлайн-проектів, де група друзів, чи велика спільнота не пов'язаних між собою людей, могли разом проводити час [2].

Новим витком розвитку стала поява на початку нового тисячоліття ще однієї платформи – смартфонів. Її виникнення також призвело до мобілізації ресурсів багатьох студій та незалежних розробників з метою займання нової ніші, оскільки вкладання сил та коштів в неї дає можливість охопити сотні мільйонів пристроїв та їхніх користувачів у будь-який момент часу через її портативність, а вже згаданий раніше розвиток Інтернету лише полегшив отримання доступу до широкої бібліотеки цифрових розваг [1].

Сучасні відеоігри різняться від своїх попередників, настільних ігор, але неможливо заперечувати наявність спільних ідей, концептів, ігрованих особливостей, тощо. Як мінімум, більшість сучасних виключно ігрових жанрів мають попередників саме серед настільних ігор, наприклад, прабатьком всіх рольових відеоігор є настільна рольова гра «Dungeons & Dragons», а покрокові стратегічні ігри своє існування завдячують шахам, «Монополії» та іншим. Фактично, можна стверджувати, що самі відеоігри є еволюційним явищем, розвитком класичних розваг у новій, відповідно до часу форми [1].

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації кросплатформеної багатокористувацької сесійної реалізації настільної гри «Монополія». Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

Розглядаючи переваги та недоліки конкурентних продуктів можна визначити ключові аспекти, які можуть найбільше впливати на прийом ефективних рішень у сучасній сфері кросплатформених відеоігор.

«Монополія» – дуже відома настільна гра і тому має декілька створених програмних рішень на багатьох платформах, на мобільних пристроях найбільш популярні – це «Monopoly» від «Marmalade Game Studio» та «Monopoly GO» від «Scopely» [5, 6].

«Monopoly» – це мобільна версія класичної настільної гри, розроблена студією «Marmalade Game Studio», гра вийшла 2019 року на iOS та Android, скріншот ігрового процесу наведено на рисунку 1.1. [5].



Рисунок 1.1 – «Monopoly» від «Marmalade Game Studio» [5]

Переваги «Monopoly» від «Marmalade Game Studio»:

- виконана у 3Д, має чудову графіку та анімацію, які роблять гру більш привабливою та захопливою [5].

Недоліки «Monopoly» від «Marmalade Game Studio»:

- можливість користуватися застосунком повністю залежить від доступності ігрових сервісів [5];

- наявність руйнуючих ігролад покупок у внутрішньому магазині, які можуть надати нечесну перевагу гравцям над своїми суперниками [5].

«Monopoly GO» – це мобільна версія класичної настільної гри, розроблена студією «Scopely», гра вийшла 2023 року на iOS та Android, скріншот ігрового процесу наведено на рисунку 1.2. [6].



Рисунок 1.2 – Ігровий процес Monopoly GO [6]

Переваги «Monopoly GO» від «Scopely»:

- можливість власних налаштувань правил гри [6];
- виконана у 3Д, має чудову графіку та анімацію, які роблять гру більш привабливою та захопливою [6].

Недоліки «Monopoly GO» від «Scopely»:

- можливість користуватися застосунком повністю залежить від доступності ігрових сервісів [6];
- гра пропонує внутрішні покупки, які можуть надати нечесну перевагу гравцям над своїми суперниками [6];
- ігровий інтерфейс є занадто громіздким та заплутаним [6].

Для порівняння дипломного проєкту з відповідними аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Критерій порівняння	Дипломополія	Monopoly GO	Monopoly	Пояснення
Спосіб розповсюдження	Відкритий вихідний код	Безкоштовна базова версія, покупки всередині гри	Платна базова версія, покупки всередині гри	Спосіб отримання копії клієнтами
Доступні платформи	iOS, Android	iOS, Android	iOS, Android	Пристрої, де доступна гра
Підтримка кросплатформи	+	+	+	Сумісність платформ
Продуктивність	Гра виконана у 2Д-стилі, немає ефектів	Гра виконана у 3Д, багаті ефекти	Гра виконана у 3Д, багаті ефекти	Швидкодія та вимоги до ресурсів

Продовження таблиці 1.1

Критерій порівняння	Дипломополія	Monopoly GO	Monopoly	Пояснення
Мережева топологія	Клієнт - Хост	Клієнт - Сервер	Клієнт - Сервер	Схема обміну даними
Доступність	Ігрові сесії хостяться через однорангову архітектуру	Вимагають постійної доступності сервісів та серверів	Вимагають постійної доступності сервісів та серверів	Залежність від зовнішніх серверів та сервісів

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Під час процесу розробки багатокористувацьких ігор завжди виникає потреба у визначенні мережевої топології та доцільності інтеграції сторонніх сервісів для реалізації взаємодії між клієнтами [7]. Оскільки метою розробки дипломного проєкту є мінімізація залежності кінцевих користувачів від доступності віддалених сервісів та ресурсів, то вибрані рішення мають відповідати поставленому завданню по забезпеченню автономності додатку.

Основними варіантами мережевих топологій є клієнт-серверна та однорангова. Кожен підхід має свої переваги та недоліки, але у контексті забезпечення максимальної автономності, клієнт-серверна архітектура не може забезпечити необхідний рівень доступності у зв'язку з залежністю від керованого флоту серверів, який може знаходитися на технічному обслуговуванні, або бути причиною високої затримки через невідале розташування до користувача [7]. На противагу, клієнт-серверному рішенню, однорангові мережі не вимагають наявності виділеного серверу, а працюють на основі комунікації рівноправних вузлів у мережі, що дозволяє забезпечити найвищий можливий рівень доступності додатку [8].

У багатокористувацьких іграх часто виникає потреба в об'єктивному та динамічному механізмі оцінювання рівня гри гравців у зв'язку з необхідністю забезпечення справедливого підбору гравців з метою створення умов для комфортної та чесної гри. Одним з можливих вирішень цього завдання є впровадження рейтингових алгоритмів, які дозволяють коректно оцінювати навички гравця в залежності від певних параметрів [9].

Першим відомим рейтинговим алгоритмом є система Ело, запропонована Арпадом Ело у середині XX століття. Ця система набула широкого застосування у шахах, а згодом – і в комп'ютерних іграх [9]. У класичному вигляді, Ело працює за принципом коригування рейтингу гравця після кожного матчу в залежності від очікуваного та фактичного результату. Таким чином, якщо гравець перемагає суперника з вищим рейтингом, то його власний рейтинг зростає більше, ніж тоді, коли той же гравець перемагає слабшого за себе опонента. Запропонована модель є доволі ефективною для поєдинкових матчів, але не працює в контексті багатокористувацьких ігор, де присутні різні команди, або коли результат не є бінарним значенням [10].

Для вирішення обмежень класичної системи ELO, компанією Microsoft була розроблена вдосконалена рейтингова модель – TrueSkill, яка з 2007 року активно використовується у власних ігрових розробках [10]. На відміну від Ело, TrueSkill базується на байєсівському статистичному підході, де рейтинг кожного гравця описується не одним числом, а двома параметрами, а саме середнім очікуваним рівнем навичок, який відображає очікувану силу гравця та стандартним відхиленням, яке вказує на рівень невизначеності та мінливості результатів. Такий підхід дозволяє точніше прогнозувати результати завдяки впровадженню більш чутливих параметрів та кращій адаптації до нових гравців, про яких ще не зібрано достатньо даних [10].

Ще однією перевагою TrueSkill є підтримка командних матчів та ігрових сесій з багатьма гравцями, де підрахунок рейтингу здійснюється з урахуванням усіх учасників, що робить алгоритм TrueSkill більш універсальним для сучасних рішень в контексті застосування ігрової індустрії [10].

1.4 Аналіз та моделювання бізнес-процесів

Для моделювання та аналізу програмного забезпечення було використано BPMN. Business Process Model and Notation – це стандарт для моделювання бізнес-процесів, який надає уніфікований мовний засіб для спільного розуміння, аналізу та оптимізації бізнес-процесів [11].

Для опису бізнес-процесів програмного забезпечення використовується BPMN-діаграми, які зображені на рисунках 1.3 – 1.5.

На рисунку 1.3 зображена діаграма підключення хоста та клієнта до лобі в залежності від обраної користувачем ролі.

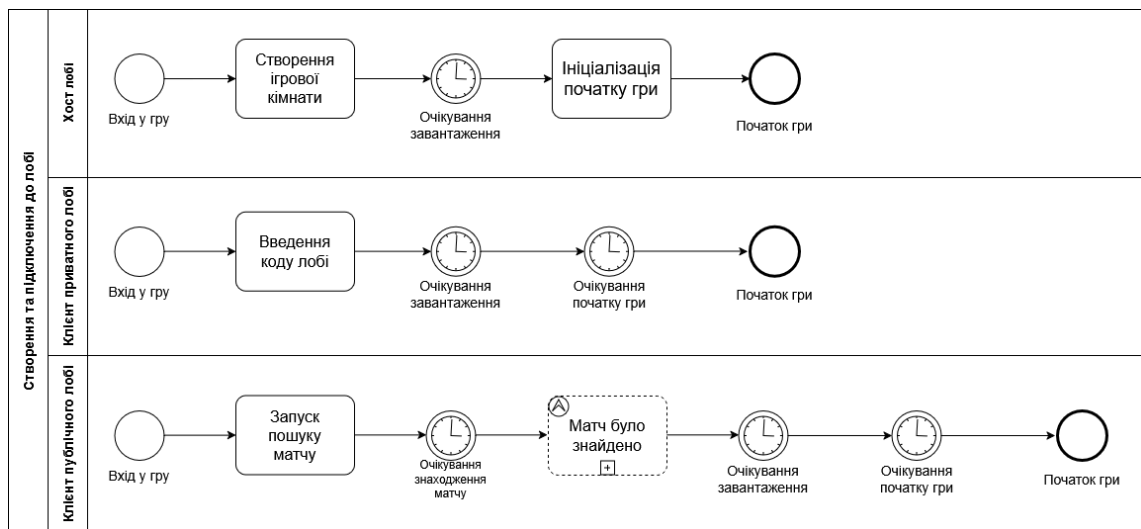


Рисунок 1.3 – BPMN-модель гравців обох типів під час під'єднання до лобі

Опис послідовності під час під'єднання до ігрової сесії:

- користувач може виконувати роль хоста приватного ігрового лобі, у такому випадку, він має можливість створення приватної ігрової сесії;
- користувач-клієнт може приєднатися до публічного ігрового лобі, запускаючи пошук матчу;
- користувач-клієнт має можливість долучитися до приватного ігрового лобі, ввівши код сесії, який може бути наданий хостом лобі.

Основна логіка виконання ходу гравцем та стандартний ігровий цикл наведено на рисунку 1.4.

Опис послідовності дій під час виконання ходу:

- гравець очікує початку свого ходу;

- гравець має можливість виконати дії над придбаною власністю;
- гравець має можливість почати процес обміну з іншим гравцем;
- коли гравець кидає кубик, то відбувається переміщення полем;
- очікування переміщення полем;
- якщо гравець приземлився на поле шанс, то виконує необхідні дії;
- якщо гравець приземлився на спеціальне поле, то виконує дії;
- якщо гравець приземлився на поле власності, яке належить іншому гравцю, то сплачує оренду;
- якщо гравець приземлився на поле власності, яке нікому не належить, то гравець має можливість придбати цю власність.

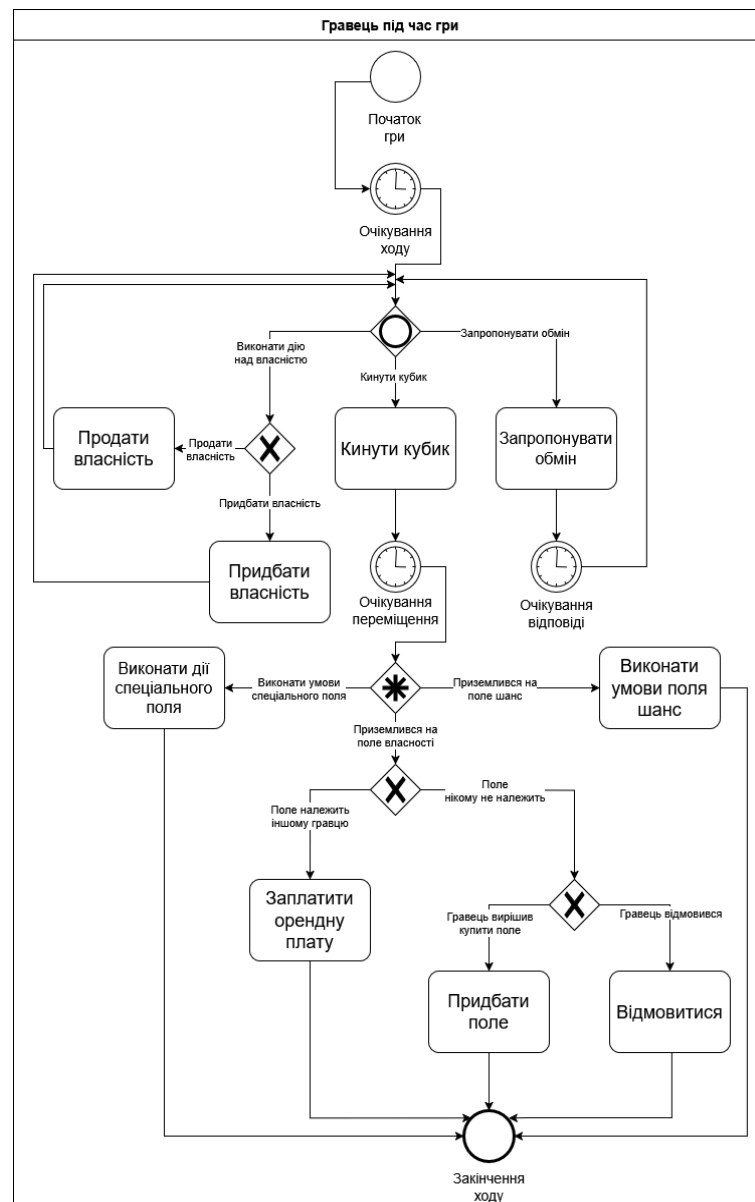


Рисунок 1.4 – BPMN-модель для гравця під час виконання ходу

На рисунку 1.5 зображена діаграма виконання обміну між користувачами під час ігрового процесу.

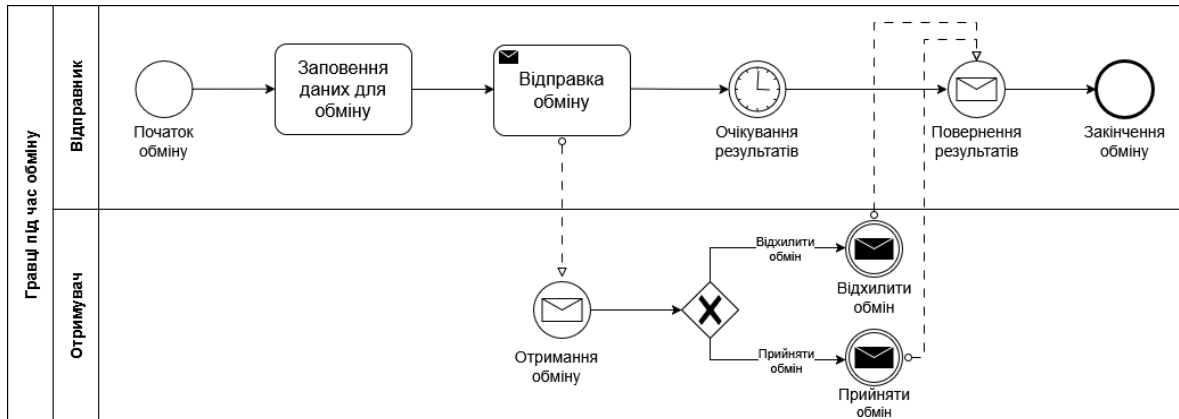


Рисунок 1.5 – BPMN-модель для гравця під виконання обміну

Опис послідовності під час виконання обміну:

- заповнення необхідних даних для обміну;
- відправка обміну;
- прийняття, або відмова від пропозиції обміну;
- повернення керування над поточним ходом до відправника.

Висновки до розділу

У даному розділі було визначено мету дипломного проєкту, яка полягає у забезпеченні автономної та стабільної роботи кросплатформеної мобільної багатокористувацької гри, що є цифровою адаптацією класичної настільної гри «Монополія». Досягнення поставленої мети виконується шляхом використання однорангової централізованої мережевої топології, яка дозволяє мінімізувати використання сторонніх сервісів та ресурсів і забезпечує зручну взаємодію гравців незалежно від їх платформи та місцезнаходження.

Актуальність розробки обґрунтовано популярністю мобільних ігор, які набули широкого розповсюдження через зручність та портативність платформи та доступу до інтернету.

На етапі аналізу існуючих рішень, таких як «Monopoly» від Marmalade Game Studio та «Monopoly GO» від Scopely, було виявлено їх переваги та недоліки. Позитивними аспектами готових рішень є якісна графіка та вищий

рівень захисту цілісності ігрової сесії, а й слабкими сторонами є наявність внутрішньо ігрових покупок, які порушують чесність гри, та повна залежність від серверів та зовнішніх сервісів, що може призвести до тривалої недоступності навіть базової версії гри.

Для реалізації проєкту було проаналізовано алгоритмічні та технічні рішення, зокрема мережеву топологію та системи оцінювання рейтингу гравців. Мережевою архітектурою була обрана однорангова, яка позбавляє необхідності наявності виділеного серверу, а рейтинговим алгоритмом було обрано TrueSkill через універсальність та підтримку багатокористувацьких ігрових сесій.

Також, за допомогою BPMN-діаграм було змодельовано основні бізнес-процеси, які включають створення та підключення до ігрової сесії, логіку виконання ходу та процес обміну між гравцями.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Розроблений мобільний застосунок є реалізацією настільної гри «Монополія» з багатокористувацьким режимом до 5 гравців, який забезпечує підвищену автономність та стабільність роботи завдяки зниженню залежності від серверної інфраструктури та сторонніх сервісів.

Серед функціональних задач, гравцям надані наступні можливості:

- створювати приватні та публічні лобі та бути хостом ігрових сесій;
- підключення до публічних лобі у ролі клієнта шляхом пошуку лобі;
- приєднуватися до приватних лобі за кодом лобі у ролі клієнта;
- кидання кубика та пересування полем на задану кількість кроків;
- можливість купівлі власності, яка нікому ще не належить;
- покращення наявної власності та збільшення ціни за оренду у випадку повної зібраної монополії;
- механіка закладання поля та повернення вартості його купівлі на баланс гравця. У випадку наявності розбудов на цьому полі, надається можливість продавати покращення поля й повернути баланс, який був витрачений на його розбудову;
- можливість проводити обміни з іншими гравцями у тому випадку, якщо умови обміну є дійсними та успішно пройшли валідацію;
- можливість завчасно здатися означає, що гравець лишається всіх своїх полів, статків і залишається у ролі спостерігача;
- зміна рейтингу гравців по закінченню рейтингового матчу в залежності від перемоги, або програшу конкретного користувача.

Всі вище перераховані функціональні задачі можна побачити на діаграмі використання, яка наведена у кресленні 1 графічного матеріалу.

В таблицях 2.1 – 2.14 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-1

Use case name	Перегляд профілю користувача
Use case ID	UC-01
Goals	Переглянути профіль користувача
Actors	Користувач
Trigger	Користувач натискає на кнопку перегляду профілю
Pre-conditions	Користувач знаходиться у головному меню
Flow of Events	Користувач натискає на кнопку перегляду профілю й переглядає інформацію власного облікового запису
Extension	-
Post-Condition	Користувачу був показаний власний профіль

Таблиця 2.2 – Варіант використання UC-2

Use case name	Зміна імені користувача
Use case ID	UC-02
Goals	Зміна імені гравця
Actors	Користувач
Trigger	Користувач натискає на поле редагування нікнейму у профілі
Pre-conditions	Користувач знаходиться у головному меню на вкладці профіль
Flow of Events	Користувач вводить своє ім'я у відповідне поле для вводу імені й повертається до попередньої вкладки меню, нікнейм користувача успішно оновлений
Extension	У випадку введення некоректного значення, показується діалогове вікно з відповідним повідомленням, що описує причину некоректно введеного імені
Post-Condition	Ім'я користувача було успішно змінено як локально, так і віддалено

Таблиця 2.3 – Варіант використання UC-3

Use case name	Пошук рейтингового матчу
Use case ID	UC-03
Goals	Знайти рейтинговий матч
Actors	Користувач
Trigger	Користувач натиснув на кнопку пошуку рейтингового матчу
Pre-conditions	Користувач знаходиться на відповідній вкладці публічного лобі у головному меню
Flow of Events	Користувач натискає на кнопку пошуку матчу, демонструється вікно з повідомленням про очікування пошуку гри, у випадку успішного знаходження гри, гравець завантажується до лобі
Extension	Повідомлення про помилку у випадку відсутності жодного відкритого публічного лобі, куди може приєднатися користувач
Post-Condition	Знайдено рейтинговий матч, користувач очікує завантаження у відповідне лобі

Таблиця 2.4 – Варіант використання UC-4

Use case name	Створення ігрової кімнати
Use case ID	UC-04
Goals	Створити ігрове лобі
Actors	Користувач
Trigger	Користувач натиснув на кнопку створення гри
Pre-conditions	Користувач знаходиться на відповідній вкладці створення публічного, або приватного лобі
Flow of Events	Користувач натискає на кнопку створення лобі, демонструється вікно з повідомленням про очікування завантаження й відбувається успішне підключення до ігрової кімнати
Extension	Повідомлення про помилку у випадку невдалого створення лобі
Post-Condition	Створено нове ігрове лобі у якому знаходиться користувач

Таблиця 2.5 – Варіант використання UC-5

Use case name	Підключення до приватної ігрової кімнати
Use case ID	UC-05
Goals	Підключитися до приватного лобі
Actors	Користувач
Trigger	Користувач натиснув на кнопку підключитися як клієнт та ввів відповідний правильний код ігрової сесії
Pre-conditions	Користувач знаходиться у вкладці підключення до лобі
Flow of Events	Користувач вводить код лобі та натискає на кнопку підключення, очікує завантаження й завантажується до лобі
Extension	У випадку невдалого підключення показується діалогове вікно з повідомленням про помилку
Post-Condition	Гравець успішно завантажився до лобі

Таблиця 2.6 – Варіант використання UC-6

Use case name	Ініціалізація початку гри
Use case ID	UC-06
Goals	Почати ігрову сесію
Actors	Користувач
Trigger	Користувач-хост, натискає на кнопку початку гри
Pre-conditions	Гравець підключений до лобі та кількість гравців у лобі дорівнює мінімальній кількості гравців для початку гри
Flow of Events	Користувач, який створив лобі, натискає на кнопку початку гри, виводиться діалогове вікно про очікування ініціалізації гри та відбувається завантаження гравців до самої ігрової сцени
Extension	У випадку невдалого початку гри відбувається повернення до лобі та показується діалогове вікно з відповідною причиною
Post-Condition	Гравці з лобі знаходяться на головній ігровій сцені

Таблиця 2.7 – Варіант використання UC-7

Use case name	Кидання кубика
Use case ID	UC-07
Goals	Кинути ігрові кубики, які показують кількість кроків, на яку має переміститися гравець за поточний хід
Actors	Користувач
Trigger	Користувач натиснув на кнопку кидання кубика
Pre-conditions	Наступила черга гравця виконувати свій хід
Flow of Events	Користувач натискає на кнопку кидання кубика і відбувається переміщення ігровим полем на необхідну кількість кроків
Extension	-
Post-Condition	Гравець перемістився ігровим полем

Таблиця 2.8 – Варіант використання UC-8

Use case name	Придбання власності
Use case ID	UC-08
Goals	Придбати нову власність
Actors	Користувач
Trigger	Гравець приземлився на поле, яке нікому не належить
Pre-conditions	Поле власності є вільним та не належить жодному іншому гравцю
Flow of Events	Користувач може придбати, або відмовитися від придбання власності натискаючи на відповідні кнопки на панелі вибору взаємодії з ігровим полем
Extension	У випадку недостатньої кількості ігрової валюти для придбання поля, виводиться діалогове вікно з відповідним повідомленням
Post-Condition	Гравець придбав поле, або відмовився від нього, завершив свій хід і право наступного ходу передається наступному гравцю

Таблиця 2.9 – Варіант використання UC-9

Use case name	Розбудова монополії
Use case ID	UC-09
Goals	Побудувати покращення ігрового поля
Actors	Користувач
Trigger	Гравець натискає на відповідне поле, яке бажає покращити
Pre-conditions	У гравця зібрана повна монополія, до якої належить це поле
Flow of Events	Користувач натискає на кнопку розбудови монополії, збільшується її рівень і відповідне значення значення рівня демонструється за допомогою кількості зірок на самому ігровому полі
Extension	У випадку недостатнього балансу для придбання покращення поля, виводиться діалогове вікно з відповідним повідомленням
Post-Condition	Гравець покращив поле

Таблиця 2.10 – Варіант використання UC-10

Use case name	Продаж власності
Use case ID	UC-10
Goals	Закласти власність, щоб повернути суму за її придбання
Actors	Користувач
Trigger	Користувач натискає на поле, яке йому належить й у відкритому вікні натискає на кнопку продажу власності
Pre-conditions	Ігрове поле є придбаним і гравець є її власником
Flow of Events	Користувач натискає на поле, яке бажає продати і натискає на кнопку продажу у впливаючому вікні та отримав компенсацію
Extension	У випадку, якщо поле вже є закладеним і його неможливо продати, користувачу демонструється відповідне повідомлення
Post-Condition	Гравець заклав поле та отримав за це грошову компенсацію

Таблиця 2.11 – Варіант використання UC-11

Use case name	Здійснення обміну
Use case ID	UC-11
Goals	Зробити обмін між гравцями
Actors	Користувач
Trigger	Користувач натискає на панель відповідного гравця, з яким бажає здійснити обмін
Pre-conditions	Користувач отримав право ходу
Flow of Events	Користувач натискає на панель відповідного гравця, з яким прагне здійснити обмін, обирає поля, якими бажає обмінятись, кількість ігрової валюти, яка братиме участь у процесі обміну й натискає на кнопку пропозиції обміну, пропозиція обміну була відправлена отримувачу
Extension	У випадку некоректних умов обміну з'являється діалогове вікно з відповідним текстом повідомленням про помилку
Post-Condition	Гравець успішно відправив пропозицію обміну іншому гравцю

Таблиця 2.12 – Варіант використання UC-12

Use case name	Прийняття обміну
Use case ID	UC-12
Goals	Прийняти запропонований обмін
Actors	Користувач
Trigger	Користувач натискає на кнопку прийняти обмін
Pre-conditions	Гравець отримав пропозицію обміну від іншого гравця
Flow of Events	Користувач ознайомлюється з пропозицією обміну і приймає обмін, натискаючи відповідну кнопку
Extension	-
Post-Condition	Обмін було здійснено успішно, поля та статки гравців оновилися, контроль над грою повернувся до поточного гравця

Таблиця 2.13 – Варіант використання UC-13

Use case name	Відхилення обміну
Use case ID	UC-13
Goals	Відхилити запропонований обмін
Actors	Користувач
Trigger	Користувач натискає на кнопку відхилити обмін
Pre-conditions	Гравець отримав пропозицію обміну від іншого гравця
Flow of Events	Користувач ознайомлюється з пропозицією обміну і приймає рішення про відхилення обміну й натискає на кнопку відхилити пропозицію обміну
Extension	-
Post-Condition	Обмін відхилено, контроль повернувся до поточного гравця

Таблиця 2.14 – Варіант використання UC-14

Use case name	Ініціалізація добровільної здачі
Use case ID	UC-14
Goals	Добровільно здатися та достроково завершити гру
Actors	Користувач
Trigger	Користувач натискає на власну бокову ігрову панель
Pre-conditions	-
Flow of Events	Користувач натискає на власну панель й у спливаючому діалоговому вікні підтверджує своє бажання достроково завершити гру і в залежності від обраного варіанту, гравець продовжує гру, або добровільно її завершує
Extension	-
Post-Condition	Гравець успішно здався, позбувся усієї власності і більше не бере участь у ігровому процесі, але може спостерігати за грою у ролі глядача

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на функціональні задачі, де кожна має свої відповідні задачі й підзадачі.

У таблиці 2.15 наведено загальну модель вимог, а в таблиці 2.16 наведено опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити у таблиці 2.17.

Таблиця 2.15 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Перегляд профілю користувача	REQ-01	Низький	Низький
1.1	Можливість перегляду рейтингу	REQ-02	Низький	Низький
1.2	Можливість зміни нікнейму	REQ-03	Високий	Низький
2	Пошук рейтингового матчу	REQ-04	Високий	Середній
2.1	Пошук відбувається в залежності від значення рейтингу гравця	REQ-05	Високий	Середній
3	Створення ігрового лобі	REQ-06	Високий	Середній
3.1	Підключення до сервісу-проксі хостом	REQ-07	Високий	Середній
4	Підключення ігрового лобі	REQ-08	Середній	Середній
4.1	Підключення до сервісу-проксі	REQ-09	Середній	Середній
5	Ініціалізація початку гри	REQ-10	Високий	Низький
5.1	Перевірка мінімальної кількості гравців	REQ-11	Високий	Низький

Продовження таблиці 2.15

№	Назва	ID Вимоги	Пріоритети	Ризики
6	Кидання кубика	REQ-12	Високий	Низький
6.1	Виконання пересування фішки гравця на задану кількість кроків	REQ-13	Високий	Низький
7	Виконання умов полю шанс	REQ-14	Середній	Низький
8	Придбання власності	REQ-15	Середній	Низький
8.1	Валідація балансу	REQ-16	Середній	Низький
9	Розбудова монополії	REQ-17	Середній	Низький
9.1	Тип монополії залежить від поля	REQ-18	Середній	Низький
10	Продаж власності	REQ-19	Середній	Низький
10.1	Зарахування вартості проданого поля на баланс	REQ-20	Середній	Низький
11	Здійснення обміну	REQ-21	Низький	Низький
11.1	Перевірка коректності умов	REQ-22	Низький	Низький
12	Прийняття обміну	REQ-23	Низький	Низький
13	Відхилення обміну	REQ-24	Низький	Низький
14	Реалізація добровільної здачі	REQ-25	Низький	Низький
14.1	Ліквідація всіх активів гравця	REQ-26	Низький	Низький
15	Реалізація системи рейтингу	REQ-27	Середній	Середній
15.1	Оновлення значення рейтингу	REQ-28	Середній	Середній

Таблиця 2.16 – Перелік функціональних вимог

Назва	Опис
FR-1	Перегляд профілю користувача Користувач має можливість переглянути свій профіль, де відображається рейтинг та нікнейм, який можна змінити.
FR-2	Пошук рейтингового матчу Можливість запуску пошуку рейтингового матчу, який відбувається з урахуванням рейтингу гравця, що дозволяє підібрати супротивників з подібним рівнем гри.
FR-3	Створення ігрового лобі Можливість створення власного лобі, якщо гравець прагне бути хостом ігрової сесії і відповідати за початок і завершення гри.
FR-4	Підключення до ігрового лобі Користувач має мати можливість підключення до ігрового лобі, у випадку, якщо користувач вводить правильний код приватного лобі.
FR-5	Ініціалізація початку гри Користувач-хост має можливість запустити ігрову сесію за умови наявності мінімальної кількості гравців, при цьому закривається лобі й забороняється іншим гравцям приєднатися до нього.
FR-6	Кидання кубика Коли настає черга ходу гравця, то у цього користувача є можливість кинути кубик, щоб виконати переміщення ігровим полем на відповідну кількість кроків.
FR-7	Виконання умов поля шанс Застосування випадкової події з завчасно визначеного набору на полі.
FR-8	Придбання власності У випадку, коли гравець кинув кубик і перемістився на поле власності, яке нікому не належить, то користувачу надається можливість придбати це поле.

Продовження таблиці 2.16

Назва	Опис
FR-9	Розбудова монополії Коли гравець зібрав повну монополію, то користувачу відкривається можливість покращувати її поля, підвищуючи ціну за оренду перебування інших гравців на цих полях.
FR-10	Продаж власності Користувачу доступна можливість продажі покращень на полі зібраної монополії, або тимчасове закладання придбаної власності у випадку відсутності розбудов.
FR-11	Здійснення обміну У гравця є можливість запропонувати обмін іншому гравцеві й обмінятися полями власності та балансом.
FR-12	Прийняття обміну Гравець, якому надійшла пропозиція обміну, має можливість її прийняти. У такому випадку, оновлюються значення балансу і відповідні поля змінюють свого власника.
FR-13	Відхилення обміну Гравець, якому надійшла пропозиція обміну, має можливість її відхилити. У такому випадку, обмін не відбувається, керування портвається до поточного гравця.
FR-14	Гравець має можливість добровільно здатися під час гри У користувача є можливість здатися, завдяки чому він вибуває з гри й продовжує знаходитися у грі у ролі спостерігача.
FR-15	Реалізація системи рейтингу Реалізована систему рейтингу, яка дозволяє відслідковувати успішність гравців на основі результатів матчів. Після завершення кожного рейтингового матчу, рейтинг гравця оновлюється у відповідності до його результату та рейтингу супротивників

2.3 Розроблення нефункціональних вимог

Нефункціональні вимоги – це характеристики системи, або програмного забезпечення, які не відповідають за конкретні функціональні задачі, але визначають додаткові аспекти та вимоги, які встановлюють загальні стандарти та обмеження, яким має відповідати розробка [12].

Відповіді нефункціональні вимоги наведені у таблиці 2.18.

Таблиця 2.18 – Нефункціональні вимоги

Номер	Категорія	Опис
NFR-1	Продуктивність	Гра повинна забезпечувати нормальний рівень швидкодії та оптимізації та споживати якомога менше заряду батареї пристрою.
NFR-2	Надійність	У випадку виникнення критичних ситуацій, помилок, гравцю мають надаватися зрозумілі та інформативні повідомлення.
NFR-3	Сумісність	Гра повинна працювати на різних мобільних пристроях, щоб забезпечити доступність для широкого кола гравців.
NFR-4	Безпека	Гра повинна гарантувати безпеку взаємодії гравців та захист персональних даних користувачів завдяки застосуванню зашифрованих централізованих шляхів обміну.
NFR-5	Зручність інтерфейсу	Гра повинна мати адаптивний, зручний та інтуїтивно зрозумілий для використання користувацький графічний інтерфейс.

2.4 Аналіз системних вимог

Мобільний ігровий застосунок призначений для роботи на сучасних версіях операційних систем Android 9.0 (Pie) і вище та iOS 14.0 і вище. Для забезпечення стабільної роботи гри рекомендується не менше 3 ГБ оперативної пам'яті й процесор не гірше Qualcomm Snapdragon 625 на пристроях Android та 2 ГБ оперативної пам'яті з мінімально рекомендованим процесором Apple A10 Fusion на пристроях iOS [14, 15].

Поглиблений аналіз системних вимог до розробки мобільної кросплатформеної гри, продемонстрував необхідність реалізації коректної адаптації графічного користувацького інтерфейсу для смартфонів з різним співвідношенням сторін та роздільною здатністю екранів [16].

У зв'язку з дизайнерським рішеннями про розміщення ігрового поля та елементів інтерфейсу, відсутня підтримка планшетів і iPad-ів через притаманні їм квадратні співвідношення сторін, які не дозволяють коректно розмістити та масштабувати необхідні графічні компоненти.

2.5 Аналіз економічних показників програмного забезпечення

Для оцінювання економічних показників була використана модель СОСОМО II, оскільки вона враховує сучасні підходи до розробки, такі як ітеративна, об'єктно-орієнтована та модульно-структурована розробка [17].

СОСОМО II дозволяє більш точно оцінювати параметри проєкту, витрачені зусилля, тривалість і кінцеву вартість розробки з урахуванням сучасних підходів та технологій. Ця методологія також враховує повторне використання компонентів та модулів, необхідність масштабованості що відрізняє його від СОСОМО, яка орієнтована на старі процеси та підходи до розробки програмних продуктів [17].

Процес розрахунку показників для моделі СОСОМО II починається з оцінки значущості факторів, відповідні значення та обґрунтування яких наведено у таблицях 2.19 – 2.20.

Таблиця 2.19 – Оцінка значущості факторів масштабу

Назва атрибуту <i>CFj</i>	Рейтинг	Пояснення
PREC Precedentedness	1.17	Високий, бо проєкт базується на відомих іграх і технологіях
FLEX Development Flexibility	1.10	Високий, оскільки використання мікросервісів забезпечує гнучкість у розвитку та масштабуванні проєкту
RESL Architecture / Risk Resolution	1.05	Високий, бо мікросервісна архітектура дозволяє ефективно управляти ризиками та спрощує масштабованість системи
TEAM Team Cohesion	1.10	Відсутність команди усуває потенційна проблеми з координацією під час розробки проєкту
PMAT Process Maturity	1.10	Високий, оскільки застосовуються відпрацьовані процеси розробки, CI/CD-пайплайни та управління

Таблиця 2.20 – Значення множника трудомісткості

Назва атрибуту <i>EMi</i>	Рейтинг	Пояснення
PERS (Personnel Capability)	1.05	Нижчий, через обмежені навички і досвід у сферах розробки
PREX (Personnel Experience)	1.0	Середній, бо досвід розробки обмежений і працює одна людина

Продовження таблиці 2.20

Назва атрибуту EM_i	Рейтинг	Пояснення
RCPX (Product Reliability and Complexity)	1.15	Високий, через складність багатокористувацької системи з мікросервісною архітектурою
RUSE (Developed for Reusability)	1.10	Високий, через компоненти створені для повторного використання
PDIF (Platform Difficulty)	1.20	Високий, через кросплатформеність та наявність мікросервісів
FCIL (Facilities)	1.05	Середній, оскільки доступні базові інструменти та бібліотеки
SCED (Required Development Schedule)	1.15	Високий, через складність проекту і жорстких термінів розробки

Формули оцінки трудомісткості проекту у людях на місяць наведені у формулах 2.1 – 2.4, де $B = 0.91$, $A = 2.94$, $C = 3.67$, $D = 0.28$ [17].

$$EAF = \prod_{K=1}^7 EM_{ik} = 1.863 \quad (2.1)$$

$$E = B + 0.01 \cdot \sum_{j=1}^5 SF_j = 0.9652 \quad (2.2)$$

$$PM = EAF \cdot A \cdot (SIZE)^E = 230.3 \text{ (людино – місяці)} \quad (2.3)$$

$$TM = SCED \cdot C \cdot (PM_{NS})^{D+0.2 \cdot (E-B)} = 16.9 \text{ (місяців)} \quad (2.4)$$

Розрахунки за моделлю СОСОМО II для мобільного застосунку «Монополія» виглядають обґрунтованими, оскільки враховують сучасні підходи до розробки, мікросервісну архітектуру та кросплатформенність. Отримані значення відповідають складності розробки подібних ігор та є реалістичними для проєкту, який розробляється однією особою з обмеженим досвідом [17].

2.6 Постановка завдання на розробку програмного забезпечення

Основна мета цього проєкту полягає у забезпеченні максимальної автономності та стабільної взаємодії гравців у мобільній кросплатформеній грі «Монополія» з підтримкою багатокористувацького режиму до п'яти учасників. Цифрова адаптація настільної гри реалізується через використання однорангової централізованої мережевої топології, що мінімізує залежність від серверної інфраструктури та сторонніх сервісів, забезпечуючи зручне підключення незалежно від платформи та місця перебування гравців.

Під час процесу розробки були вирішені та реалізовані відповідні поставлені функціональні задачі. Гравці мають можливість створювати власні лобі та бути хостом ігрової сесії, або приєднуватися до ігрових кімнат у ролі користувача-клієнта. У самій грі реалізована механіка ходу гравця шляхом кидання кубика та пересування ігровим полем на відповідну кількість кроків. Також була реалізована взаємодія з ігровими полями, їх купівля, продаж та можливість покращити поле, якщо гравець має зібрану монополію. Окрім того, було реалізовано систему проведення обмінів з іншими гравцями. Додатково, впроваджено систему рейтингу, де значення рейтингу гравця оновлюється за результатами зіграних матчів.

Під час розробки мобільної гри основна увага була спрямована на забезпечення безпеки даних користувачів, сумісності між різними мобільними пристроями з метою забезпечення доступності для потенційно більшої аудиторії та допустимого рівня продуктивності, який сприяє мінімізації витрати заряду батареї пристрою.

Висновки до розділу

У даному розділі сформульовано вимоги до розробки кроссплатформеної мобільної гри «Монополія» з багатокористувацьким режимом до 5 гравців, що забезпечує автономність і стабільність завдяки використанню однорангової мережевій топології, яка мінімізує залежність від серверної інфраструктури.

Було визначено ключові функціональні можливості, зокрема створення та підключення до публічних і приватних ігрових лобі, кидання кубика та пересування ігрової фішки гравця полем, взаємодію з полями завдяки наданню можливості купівлі, продажу, або покращення власності. Також передбачено ігрову механіку обміну між гравцями, добровільну дострокову здачу з втратою активів і функцію перегляду профілю, де відображається значення рейтингу, яке свідчить про успішність гравця на основі результатів попередніх матчів.

Також було розроблено нефункціональні вимоги, які спрямовані на продуктивність гри з акцентом на оптимізацію споживання заряду батареї та сумісність із широким спектром смартфонів через адаптацію графічного користувацького інтерфейсу з метою охоплення широкого кола користувачів.

Під час аналізу системних вимог було виявлено необхідність адаптації графічного інтерфейсу для екранів смартфонів різних роздільних здатностей та співвідношення сторін.

Економічні параметри проєкту було оцінено за допомогою моделі СОСОМО II, яка враховує сучасні підходи до розробки, такі як ітеративність і повторне використання коду, що дозволило визначити трудомісткість і тривалість створення гри на основі стандартизованих параметрів оцінювання.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Архітектура програмного забезпечення системи відіграє ключову роль у забезпеченні масштабованості, надійності, продуктивності та зручності супроводу продукту [18]. Основними варіантами архітектури для мобільної кросплатформеної багатокористувацької гри є монолітна, мікросервісна, де кожна має свої переваги та недоліки [19, 20].

Монолітна архітектура характеризується тим, що всі компоненти програмного забезпечення об'єднані в єдину суцільну систему. При такому підході, основна бізнес-логіка, доступ до бази даних, графічний інтерфейс користувача та інші модулі тісно пов'язані між собою та розгортаються як єдине ціле [19]. Перевагами такого рішення є простота розробки на ранніх етапах проекту, оскільки легше відстежувати залежності, налагоджувати систему, забезпечуючи її цілісність та простоту. Окрім того, розгортання монолітної системи, як правило, є просто задачею, оскільки потрібно завантажити та встановити лише саме застосунок [19]. Однак, монолітна архітектура має недоліки, які стають помітними під час процесу масштабування додатку та які пов'язані зі зростанням обсягу функціоналу та складності системи. Таким чином, зміна одного компонента часто вимагає рекомпіляції та повторного розгортання всієї системи, що уповільнює процес ітерації розробки. Крім того, монолітна архітектура ускладнює використання різних технологій та фреймворків для різних частин системи, через те, що всі компоненти використовують однаковий технологічний стек, що пов'язано зі складністю інтеграції різних середовищ виконання [19].

Мікросервісна архітектура передбачає розбиття системи на незалежні сервіси, де кожен реалізує свій конкретний функціонал та взаємодіє з іншими через визначені API-інтерфейси. Кожен мікросервіс може розроблятися, розгортатися та масштабуватися незалежно від інших, що забезпечує гнучкість

й адаптивність системи під вимоги у рамках всього проєкту. Масштабування мікросервісів є більш ефективним, оскільки можна масштабувати лише ті сервіси, які насправді цього вимагають через підвищене навантаження, або специфіку наявності вузьких місць у архітектурі [20]. Крім того, мікросервісна архітектура сприяє підвищенню надійності системи, оскільки збій одного сервісу часто не впливає на роботу інших. Також, ключовою перевагою мікросервісів є можливість використання різних технологій для різних сервісів, що дозволяє командам обирати найбільш вдалі інструменти для вирішення конкретних завдань. Проте, мікросервісна архітектура також має свої недоліки, які пов'язані зі складністю управління розподіленою системою та взаємодію сервісів на основі мережевої комунікації, через що виникають додаткові підзадачі обробки мережових запитів та необхідність координації їх роботи, яка вимагає впровадження рішень з оркестрації та моніторингу [20].

Для дипломного проєкту було вирішено використати мікросервісну архітектуру, оскільки вона допомагає забезпечити гнучкість та масштабованість, які необхідні для розробки кросплатформеної мобільної багатокористувацької гри [20]. З метою виконання поставлених задач, було реалізовано інтеграцію з трьома сторонніми сервісів, а саме: Unity Authentication Service, Unity Lobby Service та Unity Relay Service. Завдяки яким було реалізовано процес ініціалізації ігрової сесії, яка починається зі звертання до Unity Authentication Service для проходження аутентифікації, отримуючи унікальний токен сесії, який необхідний для подальшої взаємодії з грою [21]. Unity Lobby Service було використано для створення та підтримки віртуальних лобі, де гравці знаходяться перед початком матчу [22]. Саме безпечне та легке підключення клієнтів реалізовано за допомогою Unity Relay Service, що використовується як зворотний проксі, який забезпечує мережеве з'єднання клієнтів та спрощує роботу користувачу-хосту, який не має займатися переадресацією портів та налаштуванням брандмауера [21, 22, 23].

Відповідна діаграма архітектури у вигляді діаграми C4 другого рівня наведена у кресленні 2 графічного матеріалу.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

Оскільки у створюваному програмному забезпеченні наявний багатокористувацький режим, то виникає необхідність вирішення типу підтримуваної мережевої топології. Мережеві архітектури у контексті багатокористувацьких ігор, описують процес комунікації завдяки якому можна реалізувати синхронізацію та коректне оновлення стану локальних клієнтів. Серед можливих варіантів реалізації можна виокремити клієнт-серверну архітектуру, однорангова централізована та однорангова децентралізована [8].

Клієнт-серверна модель з виділеним сервером – це тип мережі, у якій присутній централізований сервер, який надсилає та приймає дані від кінцевих користувачів. Дана топологія забезпечує ефективний обмін даними та захист від зловмисного втручання з боку клієнтів завдяки механізму централізованого управління ресурсами та станом [24].

Відповідну схему-представлення клієнт-серверної топології можна побачити на рисунку 3.1.

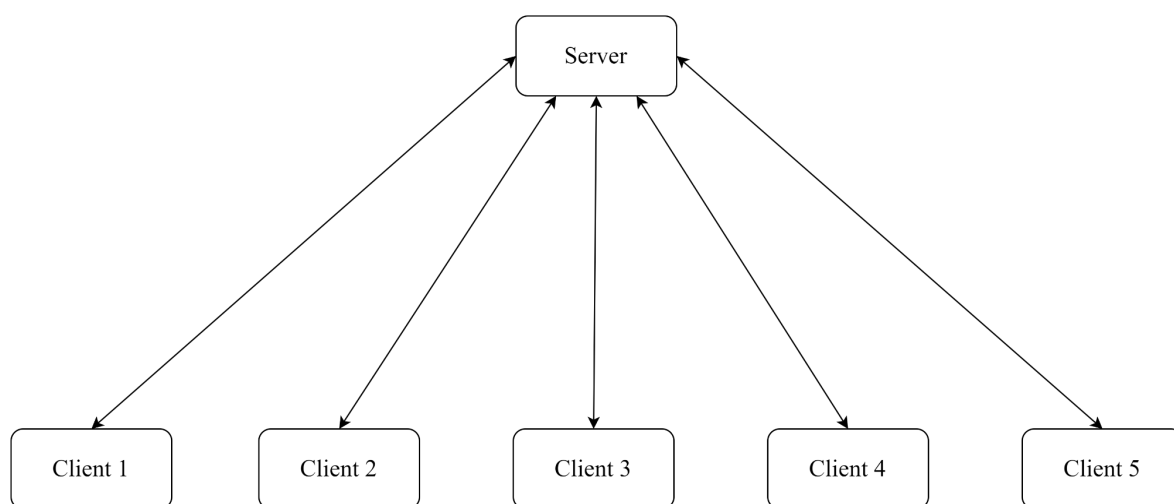


Рисунок 3.1 – Схема клієнт-серверної архітектури

Переваги клієнт-серверної архітектури:

- централізоване управління станом гри, що полегшує синхронізацію даних між гравцями та забезпечує захист від зловмисного втручання з боку кінцевих клієнтів [24];

- сервер може виконувати обробку важких, або конфіденційних розрахунків, що зменшує обсяг роботи, яка могла б проводитися локально на клієнті [8];

Недоліки клієнт-серверної архітектури:

- велика фізична відстань між сервером і гравцями може призводити до високої затримки [24];

- підтримка флоту виділених серверів вимагає значних фінансових витрат. В окремих випадках, великий обсяг трафіку та додаткові вимоги у контексті обчислювальних можливостей і безпеки передбачають підвищені вимоги до цільової інфраструктури [24];

- у випадку збільшення кількості гравців, або зростання потреби в обчислювальних ресурсах, виникають проблеми з масштабованістю наявних рішень. Додавання й інтеграція нових виділених потужностей вимагає обережного й чіткого планування, розуміння процесів і завжди потребує додаткового фінансування [7].

Однорангова децентралізована – варіант однорангової архітектури, в основі якої знаходиться мережа рівноправних вузлів [24]. Ця топологія формується таким чином, що напряду з'єднує всіх гравців безпосередньо між собою, що виключає необхідність у наявності централізованого сервера, або виділеного хоста. На відміну від клієнт-серверного підходу, де сервер приймає остаточне рішення про стан гри та виконує синхронізацію клієнтів, у одноранговій децентралізованій, кожен гравець відповідає за синхронізацію, оновлення стану гри та може вільно комунікувати з іншими клієнтами. Пряма взаємодія клієнтів може спростувати процес розробки застосунків з невеликою кількістю гравців у ігровій сесії, але повністю виключає валідацію та централізовану синхронізацію [7].

Відповідну схему-представлення однорангової децентралізованої топології можна побачити на рисунку 3.2.

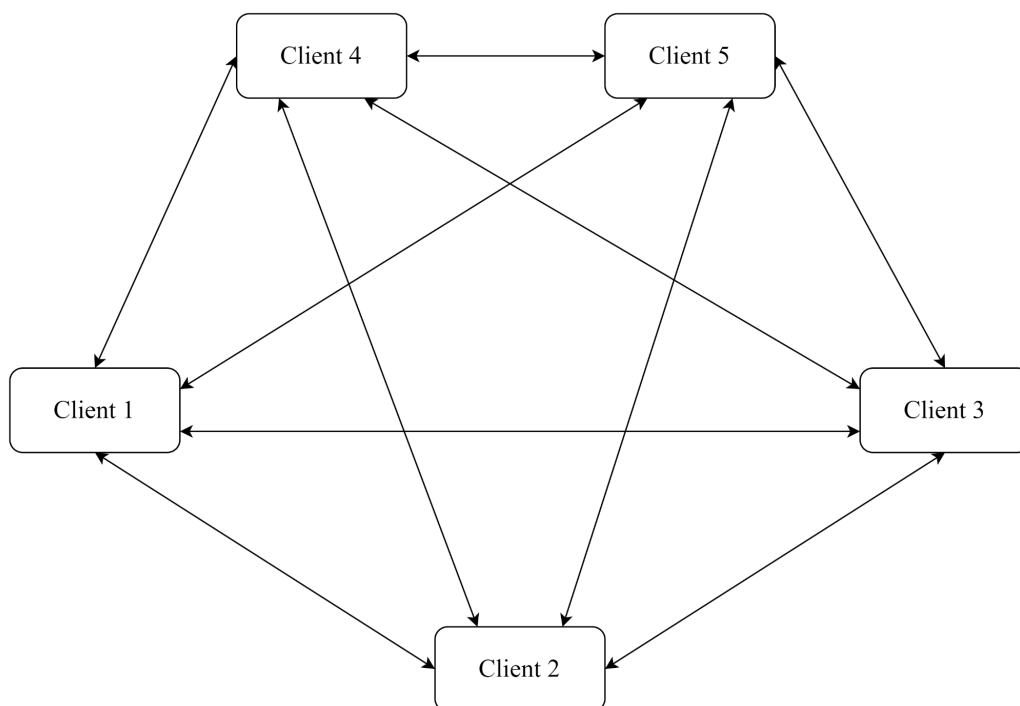


Рисунок 3.2 – Схема однорангової децентралізованої мережі

Переваги децентралізованої однорангової архітектури:

- відсутність виділеного сервера виключає залежність від централізованого вузла системи, що забезпечує більшу стійкість до відмов у випадку втрати з'єднання з головним вузлом [8];
- рівномірне розподілення обчислювальних задач обробки може знизити навантаження, оскільки кожен вузол може служити джерелом даних та їх обробником [24];

Недоліки децентралізованої однорангової архітектури:

- складність масштабування через необхідність постійної синхронізації поточного стану гри між усіма учасниками мережі, що вимагає складної й ефективної координації та синхронізації між вузлами [8];
- підвищена вразливість до шахрайських дій, оскільки вся інформація про всі підключені клієнти відома іншим клієнтами, що спрощує виконання протиправних дій у самій грі та поза нею [7];
- можлива втрата даних через відсутність централізованої точки зберігання та обробки, що ускладнює процес відновлення стану [8].

Централізована однорангова – це варіант однорангової архітектури, в якій забезпечується певний рівень централізованої координації за рахунок наявності головного вузла, який є таким самим клієнтом як й інші, але бере на себе додаткову роль хоста ігрової сесії і відповідає за її запуск та завершення [24].

Відповідну схему-представлення однорангової централізованої топології можна побачити на рисунку 3.3.

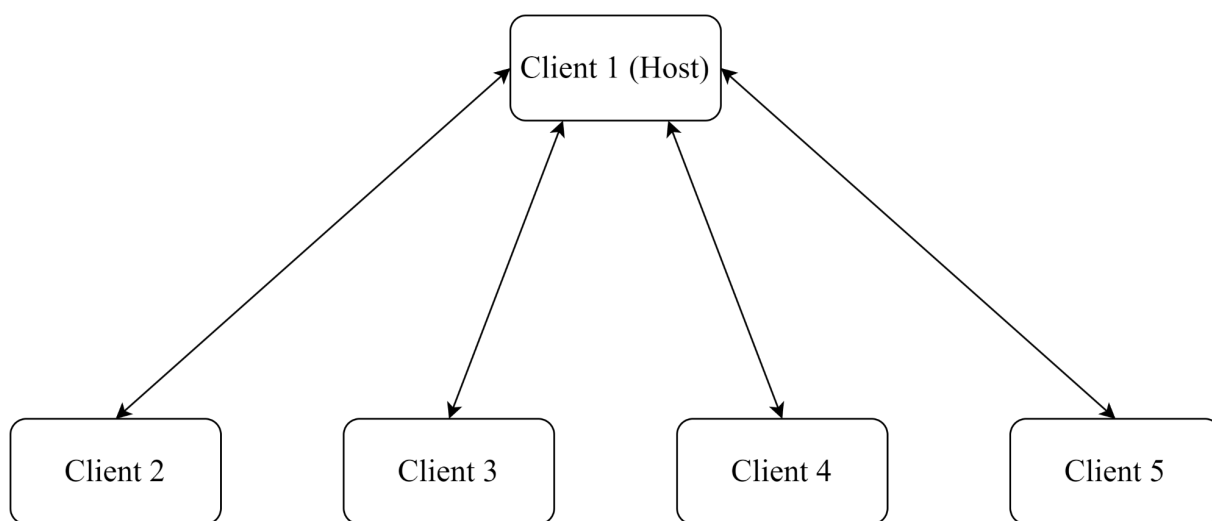


Рисунок 3.3 – Схема однорангової централізованої мережі

Переваги централізована однорангової архітектури:

- простота управління станом гри у порівнянні з децентралізованою одноранговою мережею через наявність головного вузла – хоста [8];
- централізоване управління спрощує впровадження механізмів захисту від протиправних дій з боку клієнтів, оскільки звичайні клієнти не мають всієї повноти інформації про стан гри та інших клієнтів у мережі [8, 24];
- дана архітектура може забезпечити ефективний розподіл обчислювальних ресурсів, оскільки складні обчислення можуть відбуватися лише на хості, через що знімає навантаження зі звичайних клієнтів [24].

Недоліки централізованої однорангової архітектури:

- якщо центральний вузол виявився нестійким, або вразливим, то це може призвести до збоїв у роботі всієї системи. Таким чином, якщо

центральный вузол перестає працювати, то це спричиняє збій у роботі всієї ігрової сесії, що вимагає аварійного закриття підключення [7, 8];

- масштабованість проходить легше від однорангової децентралізованої мережі, проте на певному етапі наявність єдиного центрального вузла, який паралельно виконує роль клієнта може стати обмежуючим фактором через потенційну складність запитів від інших клієнтів, виконуваних обчислень та завдань, які покладаються на центральний вузол [8].

Таким чином, централізована однорангова мережа, незважаючи на певні недоліки, є оптимальним рішенням для цільової розробки, оскільки вона здатна забезпечити стабільну синхронізацію стану гри та комунікацію між клієнтами, виключає необхідність у підтримці виділених серверів та може забезпечити прийнятний рівень механізмів безпеки у порівнянні з децентралізованою одноранговою мережею [7, 8, 24].

Для створення кросплатформеної мобільної гри, яка буде підтримувати багатокористувацький режим необхідно підібрати відповідний фреймворк, який забезпечує наступні вимоги:

- можливість кросплатформеної розробки;
- інструменти для створення однорангової централізованої мережі.

Враховуючи наведені вимоги, вибір був зроблений на користь одного з актуальних ігрових рушіїв – Unity. Ключовою перевагою цього рушія є порівняно легка кросплатформенна розробка, яка виключає необхідність внесення суттєвих змін та додаткової оптимізації, завдяки використанню високого рівня абстракцій над апаратним забезпеченням [25]. Ще однією сильною стороною є наявність широкого вибору офіційних і сторонніх бібліотек та модулів з відкритими вихідним кодом для створення багатокористувацьких ігор. До того ж, Unity має велику кількість інструментів, які пришвидшують стандартних ігрових додатків та рішень, що особливо актуально для незалежних розробників [25].

Під Unity розроблено декілька бібліотек, основною задачею яких є надання можливостей для розробки мережевого коду, проте найбільш актуальними є Mirror та Netcode for Game Objects [26, 27].

Mirror – це бібліотека-нащадок UNet, має відкритий вихідний код для створення багатокористувацьких ігрових рішень [26].

Переваги бібліотеки Mirror:

- високий рівень абстракції від безпосереднього управління мережевими ресурсами на низькому рівні [26];
- наявна підтримка клієнт-серверної та однорангової централізованої топологій, що дає можливість створювати ігри з різною топологією [26].

Недоліки бібліотеки Mirror:

- відсутність підтримки проксі [26];
- відсутня можливість створювати ігрові лобі [26].

Netcode for GameObjects – це сучасна стандартна бібліотека для створення багатокористувацьких ігор та додатків на Unity від Unity Technologies [28].

Переваги бібліотеки Netcode for Game Objects:

- офіційне рішення для багатокористувацьких ігор від Unity, що спрощує розробку через наявність документації [27];
- високий рівень абстракції від безпосереднього процесу відправки та отримання даних по мережі на низькому рівні [27];
- наявність вбудованих компонентів для роботи з мережевими підключеннями, обміном даними та іншими аспектами неткоду [27];
- підтримка різних мережевих архітектур, включаючи однорангову централізовану, однорангову децентралізовану та клієнт-сервер [27];
- полегшена можливість масштабування ігрового процесу для великої кількості гравців [27].

Недоліки бібліотеки Netcode for Game Objects:

- потреба у розумінні архітектури рушія та його компонентів [27];
- бібліотека перебуває в активній розробці, тому деякий функціонал може активно змінюватися та еволюціонувати від версії до версії [27].

Бібліотекою для написання мережевого коду було обрано Netcode for GameObjects, що обґрунтовується широким функціонал бібліотеки, підтримкою різних мережевих топологій та спрощеним масштабування [27].

3.3 Конструювання програмного забезпечення

Для розробки мобільного ігрового багатокористувацького застосунку були реалізовані наступні типи, кожен з яких виконує свою конкретну функцію.

Список основних типів та їх призначення можна побачити у таблиці 2.1.

Таблиця 3.1 – Опис класів

Назва	Опис
SessionManager	Відповідає за підключення гравців до спільної ігрової сесії, менеджмент ресурсів мережі
LobbyManager	Менеджер ігрового лобі, який відповідає за оновлення стану лобі, інформації про гравців у ньому, запуск ігрової сесії
GameManager	Ігровий менеджер, який відповідає за синхронізацію стану гри
Group	Клас для зберігання інформації про монополії та полів, які належать кожній монополії
Board	Клас для управління ігровим полем та надає методи для отримання картки шанс та інших дій
Tile	Базовий клас поля на ігровому полі
DefaultPropertyTile	Клас-нащадок поля стандартної власності
GamblingPropertyTile	Клас-нащадок особливого поля власності

Продовження таблиці 3.1

Назва	Опис
SpecialPropertyTile	Клас-нащадок поля власності транспорту
ChanceTile	Клас-нащадок поля шанс та податку
SpecialTile	Клас-нащадок кутового поля дошки
PawnController	Базовий клас контролера фішки, з відповідними методами для взаємодії з грою
PlayerPawnController	Клас-нащадок контролера фішки, який є ігровим контролером кожного гравця у сесії
BotPawnController	Клас-нащадок контролера фішки, який є ігровим контролером кожного бота у сесії
CardInformationView	Клас-представлення інформативної панелі
MessageBoxView	Клас-представлення панелі спливаючого вікна
TileManagementView	Клас-представлення панелі поля монополії
TileProposalView	Клас-представлення панелі купівлі поля
CardPaymentView	Клас-представлення панелі сплати податку
TilePaymentView	Клас-представлення панелі сплати оренди
PawnTableView	Клас-представлення панелі гравця у грі
PlayerLobbyView	Клас-представлення панелі гравця у лобі
TradeSenderView	Клас-представлення панелі відправки обміну
TradeReceiverView	Клас-представлення панелі отримання обміну

Продовження таблиці 3.1

Назва	Опис
UIManager	Клас, який відповідає за взаємодію з графічним користувацьким інтерфейсом на кожній сцені
ITileStraregyFactory	Клас, який є реалізацією патерну фабрика та надає контекстуальні стратегії для кожного поля
ICardStraregyFactory	Клас, який є реалізацією патерну фабрика та дає контекстуальні стратегії для кожного картки
PropertyTilePlayerStrategy	Клас-реалізація патерну стратегії поля для гравця
PropertyTileBotStrategy	Клас-реалізація патерну стратегії поля для бота

Для створення мережевого коду використовувалися спеціалізовані структури даних і методи, які адаптовані для потреб багатокористувацьких застосунків, де об'єм пам'яті, яка передається та швидкодія є ключовими аспектами. Зокрема, для ефективного обміну даними між клієнтом і сервером застосовувалися механізми серіалізації та десеріалізації певних типів даних, які потім передаються мережею [7, 8]. Завдяки процесу маршалізації вдалося перетворювати структури даних у компактний бінарний формат, який призначений для швидкої передачі мережею, що сприяло зменшенню обсягу даних і підвищенню швидкості їхньої передачі [8].

Найважливішим механізмом взаємодії між клієнтом і хостом є віддалені виклики процедур (RPC). RPC дозволяють викликати методи на віддаленому клієнті так, ніби вони виконуються локально, що значно спрощує задачу синхронізації даних та оновлення загального стану гри у реальному часі. Цей механізм абстрагує деталі мережевих протоколів, що дозволяє зосередитися реалізації самої ігрової логіки, а не на низькорівневому управлінні мережевими даними та управлінням підключеннями [7].

Особливістю створення багатокористувацьких ігор із централізованою одноранговою мережею є неможливість прямого виклику RPC між звичайними клієнтами з міркувань безпеки. Така пряма комунікація дозволила б одному звичайному клієнту напряму впливати на стан гри іншого без синхронізації через хоста [8]. Тому оновлення стану гри відбувається через виклик звичайного методу, який виконує всі необхідні дії локально й відправляє запит на віддалене оновлення стану гри для решти користувачів через хоста. Клієнт-хост, у свою чергу, розсилає оновлення всім підключеним клієнтам, забезпечуючи синхронізацію ігрового процесу [8].

У процесі розробки ігрових додатків, патерни проектування суттєво спрощують процес розробки, забезпечуючи структурованість, читабельність, модульність і зручність підтримки коду. Для реалізації гри було використано Entity-Component-System (ECS), основа ідея якого полягає у тому, що об'єкти в грі є не монолітними класами з фіксованим набором методів та змінних, а формуються як набори окремих компонентів, де кожен компонент відповідає за певну функціональну задачу. Такий підхід дозволяє легко модифікувати об'єкти, додаючи, або видаляючи лише певні компоненти, що робить ECS ідеальним для створення складних ігрових сцен із великою кількістю об'єктів [33].

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.2.

Таблиця 3.2 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Microsoft Visual Studio Code 1.99.2	Головний редактор коду розробленого програмного забезпечення, має безкоштовну ліцензію та широку підтримку плагінів
2	Unity Editor 6.0	Ігровий рушій, який має необхідний для безпосереднього створення гри

Продовження таблиці 3.2

№ п/п	Назва утиліти	Опис застосування
3	Gimp 3.0	Редактор растрової графіки, який використовувався для створення іконок, логотипів полів

3.4 Аналіз безпеки даних

Оскільки розроблюваний застосунок є багатокористувацькою грою, то він висуває особливі вимоги до безпеки та аналізу даних. Захист конфіденційності, запобігання несанкціонованому доступу та забезпечення стабільної роботи в режимі реального часу є важливими для забезпечення справедливого ігрового процесу та захисту від потенційних загроз [7].

Встановлення прямих підключень між гравцями та відкриття портів у багатокористувацьких іграх створюють значні ризики для безпеки, оскільки підходи можуть призвести до вразливостей несанкціонованого доступу, або перехоплення даних [24]. Відсутність належного захисту даних у реальному часі загрожує конфіденційності та стабільності гри, що може порушити ігровий процес та довіру гравців [8].

Unity Relay Service використовує зворотний проксі для безпечного з'єднання в ігрових іграх, уникаючи прямого підключення між гравцями [23]. На відміну від прямого підключення, яке вимагає відкриття портів і розкриває IP-адреси, зворотний проксі приховує їх, централізовано керує трафіком і забезпечує шифрування через DTLS, яке побудоване на основі TLS та адаптоване для роботи з UDP, забезпечує шифрування, цілісність та верифікацію даних [29]. DTLS використовує асиметричну криптографію для встановлення сеансу комунікації та симетричне шифрування для захисту пакетів даних, запобігаючи їх перехопленню чи модифікації [29]. DTLS інкапсулює UDP-пакети, додаючи криптографічний шар, що гарантує

конфіденційність і безпеку передачі в реальному часі, зберігаючи низьку затримку, яка необхідна для забезпечення стабільного ігрового процесу [7, 29].

Протокол UDP є одним із ключових протоколів транспортного рівня в мережевій моделі OSI. На відміну від TCP, UDP не забезпечує гарантованої доставки даних, не гарантує їх порядок і не використовує механізми верифікації фактів прийому пакетів, що дозволяє UDP працювати швидше, але з можливими втратами пакетів, або порушенням їх послідовності отримання [29]. Завдяки відсутності встановлення стійкого з'єднання та мінімальній додатковій з верифікації отримання пакетів, UDP ідеально підходить для багатокористувацьких ігор, де пріоритетом є швидкість передачі, а не абсолютна надійність [7].

Використання віддалених викликів процедур (RPC) може створювати потенційні ризики, якщо один із клієнтів має модифіковану версію гри. Якщо такий клієнт надсилає шкідливі, або сфальсифіковані RPC-виклики, то це може призвести до руйнування чесного ігрового процесу через потенційні маніпуляції над ігровими подіями, що порушує справедливість гри для інших гравців [8]. Для забезпечення безпеки, вся важлива логіка була перенесена на обробку хостом, а також була впроваджена серверна валідація RPC-викликів, яка перевіряє їх на відповідність поточному ігровому стану [8].

Висновки до розділу

У даному розділі було здійснено детальний аналіз архітектурних рішень для розробки кросплатформеної мобільної багатокористувацької гри, що є цифровою адаптацією настільної гри «Монополія». Основна увага приділялася вибору оптимальної архітектури програмного забезпечення та мережевої топології, які забезпечують масштабовість, надійність і безпеку гри.

Після проведення порівняння монолітної та мікросервісної архітектур було обрано мікросервісну через її гнучкість та можливість незалежного масштабування окремих компонентів.

Інтеграція з сервісами Unity Authentication Service, Unity Lobby Service та Unity Relay Service дозволила реалізувати безпечну аунтефікацію, створення ігрових лобі та стабільне мережеве з'єднання через зворотний проксі, що мінімізує ризики, пов'язані з прямими підключеннями.

Для мережевої архітектури було обрано централізовану однорангову топологію, яка забезпечує ефективну синхронізацію стану гри та управління ігровими сесіями без необхідності використання виділених серверів, що обґрунтовано можливістю спрощення розробки, зниженням витрат на інфраструктуру та достатнім рівнем захисту від несанкціонованих дій.

Використання ігрового рушія Unity та бібліотеки Netcode for GameObjects дозволяє забезпечити кросплатформенну розробку та підтримку різних мережевих топологій, що спростило реалізацію багатокористувацького режиму.

Використання Unity Relay Service із шифруванням DTLS дозволило захистити передачу даних та уникнути розкриття IP-адрес гравців, забезпечивши стабільність ігрового процесу в реальному часі.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Для проведення аналізу якості програмного забезпечення будуть використані наступні метрики програмного забезпечення:

- індекс підтримки – це метрика, яка вказує на те, наскільки легко підтримувати та вносити зміни у програмний код та оцінюється у межах від 0 до 100, де 100 свідчить про найкращий рівень підтримки [30];

- цикломатична складність – це метрика, яка відображає складність реалізації логіки програмного коду, а саме кількість можливих маршрутів виконання через використання логічних умов, циклів та інших синтаксичних конструкцій. Оцінка починається від 1 та не має верхньої межі, проте, чим нижче значення цикломатичної складності, тим легше підтримувати програмний код [30];

- зв'язність класів – це метрика, яка вимірює ступінь залежності одного типу від інших типів у програмного коді та проявляється через використання методів, полів, успадкування та делегати. Чим вище значення оцінки, тим складніше підтримувати відповідний клас, оскільки даний тип має дуже багато залежностей [30];

- глибина наслідування – це метрика, яка свідчить про кількість рівнів успадкування в ієрархії наслідування кожного типу в об'єктно-орієнтованому середовищі виконання. Чим вище значення оцінки, тим важче розбиратися у структурі класів, їх нащадках, членах та загальній архітектурі рішення [30].

Для отримання значень оцінок якості програмного коду була використана вбудована функція середовища розробки Visual Studio 2022 [30].

Відповідні результати оцінювання можна побачити на рисунку 4.1.

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling
Monopoly.Client.Shared.Utilities.Loaders	72	72	8	1
SceneLoaderUtility	72	72	8	1
Monopoly.Client.Shared.Patterns.Pools	90	90	12	5
ObjectPool<TComponent>	81	81	10	5
IPool<TObject>	100	100	2	0
Monopoly.Client.Shared.Patterns.Observables	100	100	4	0
ISubject<TArgument>	100	100	3	0
IObserver<TArgument>	100	100	1	0
Monopoly.Client.Shared.Patterns.Factories	100	100	1	0
ISimpleFactory<TInput, TOutput>	100	100	1	0
Monopoly.Client.Shared.Helpers	88	88	2	1
LateStartHelper	88	88	2	1
Monopoly.Client.Scriptable.Objects.Tokens	89	89	4	3
TokenScriptableObject	89	89	4	3
Monopoly.Client.Scriptable.Objects.Effects	89	89	3	3
EffectScriptableObject	89	89	2	3
EffectFlags	89	89	1	1
Monopoly.Client.Scriptable.Objects.Cards.Concrete	95	95	8	4
TeleportationForwardCard	100	100	1	4
TeleportationBackwardsCard	100	100	1	4
SlipCard	100	100	1	4
RewardCard	86	86	2	4
PenaltyCard	86	86	2	4
JailCard	100	100	1	4
Monopoly.Client.Scriptable.Objects.Cards.Common	100	100	2	3
CardScriptableObject	100	100	2	3

Рисунок 4.1 – Результати оцінювання програмного коду просту

Розглянувши результати оцінювання, можна помітити, що індекс підтримуваності не виходить за сталі межі норми, що свідчить про достатній рівень підтримки продукту [30].

Варто зауважити, що хоча відповідні метрики надають об'єктивні статистичні оцінки якості коду, проте вони часто не враховують специфіку самого проєкту, фреймворків та мови програмування. У контексті поточної розробки, високий рівень ієрархії наслідування класів пояснюється архітектурою рушія Unity, а високе значення залежності класів так само впливає зі специфіки розробки ігрових програмних продуктів, де такі рішення є прийнятними та необхідними [31].

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування». Для тестування буде виконане мануальне тестування, яке буде перевіряти правильність виконання всіх функціональних вимог. Варто зазначити, що оскільки розробкою є ігровий мобільний додаток, то створення та впровадження автоматичних тестів різного рівня є бажаним, але часто неможливими завданням саме у сфері розробки відеоігор [32].

У таблицях 4.1 – 4.14 наведено кейси мануального тестування програмного забезпечення.

Таблиця 4.1 – Тест 1.1 – Зміна імені користувача

Початковий стан системи	Користувач знаходиться у головному меню
Вхідні дані	Текстове значення імені користувача
Опис проведення тесту	У відповідне поле вводиться коректне ім'я, яке має складатися хоча б з 1 символу, який не є пустим символом, або символом пробілу і максимальна довжина складає 16 символів
Очікуваний результат	Зміна імені проходить успішно, значення має бути оновлено як локально, так і віддалено
Фактичний результат	Зміна імені пройшла успішно, користувач успішно змінив своє ім'я локально та віддалено

Таблиця 4.2 – Тест 1.2 – Пошук рейтингового матчу

Початковий стан системи	Користувач знаходиться на вкладці публічного лобі в головному меню
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає на кнопку пошуку матчу, система відображає вікно з повідомленням про очікування пошуку гри, і в разі успішного пошуку користувач завантажується до лобі
Очікуваний результат	Користувач успішно завантажується до рейтингового лобі У разі відсутності лобі відображається помилка
Фактичний результат	Користувач успішно завантажився до рейтингового лобі

Таблиця 4.3 – Тест 1.3 – Створення ігрової кімнати

Початковий стан системи	Користувач знаходиться на вкладці створення публічного або приватного лобі
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає на кнопку створення лобі, система відображає вікно з повідомленням про очікування завантаження, і в разі успіху користувач підключається до створеної ігрової кімнати
Очікуваний результат	Ігрове лобі створено, користувач успішно підключений. У разі невдачі відображається повідомлення про помилку
Фактичний результат	Ігрове лобі створено, користувач успішно підключений

Таблиця 4.4 – Тест 1.4 – Підключення до приватної ігрової кімнати

Початковий стан системи	Користувач знаходиться на вкладці підключення до лобі
Вхідні дані	Код приватного лобі
Опис проведення тесту	Користувач вводить коректний код лобі, натискає на кнопку підключення, очікує завантаження та підключається до лобі
Очікуваний результат	Користувач успішно завантажується до приватного лобі У разі некоректного коду відображається помилка
Фактичний результат	Користувач успішно завантажився до приватного лобі

Таблиця 4.5 – Тест 1.5 – Ініціалізація початку гри

Початковий стан системи	Користувач-хост підключений до лобі, кількість гравців відповідає мінімальній для початку гри
Вхідні дані	Мінімальна можлива кількість гравців для початку гри
Опис проведення тесту	Користувач-хост натискає на кнопку початку гри, система відображає діалогове вікно про очікування ініціалізації, після чого гравці завантажуються до ігрової сцени
Очікуваний результат	Гра успішно ініціалізована, гравці знаходяться на ігровій сцені. У разі невдачі відображається текст помилки
Фактичний результат	Гра успішно ініціалізована, гравці завантажені на ігрову сцену

Таблиця 4.6 – Тест 1.6 – Кидання кубика

Початковий стан системи	Наступила черга гравця виконувати хід
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає на кнопку кидання кубика, система відображає результат кидка, після чого гравець переміщається на відповідну кількість кроків ігровим полем
Очікуваний результат	Гравець перемістився на правильну кількість кроків відповідно до результату кидка
Фактичний результат	Гравець успішно перемістився ігровим полем на коректну кількість полів

Таблиця 4.7 – Тест 1.7 – Придбання власності

Початковий стан системи	Гравець приземлився на вільне поле власності, яке нікому не належить
Вхідні дані	Баланс для купівля поточного поля власності
Опис проведення тесту	Користувач приземляється на вільне поле, система відображає меню купівлі поля, користувач вирішив придбати за умови наявності достатньої кількості валюти
Очікуваний результат	Поле придбано, хід завершено. У разі недостатньої кількості валюти відображається текст помилки
Фактичний результат	Поле успішно придбано, хід завершено

Таблиця 4.8 – Тест 1.8 – Розбудова монополії

Початковий стан системи	Гравець володіє повною монополією, до якої належить поле
Вхідні дані	Наявність зібраної монополії та відповідний баланс, який дозволяє придбати покращення
Опис проведення тесту	Користувач натискає на поле, яке бажає покращити, вибирає кнопку розбудови монополії, система оновлює рівень монополії та відображає його за допомогою зірок на ігровому полі
Очікуваний результат	Рівень монополії підвищено, відображено на полі. У разі недостатньої валюти відображається тест помилки
Фактичний результат	Рівень монополії успішно підвищено, зірки відображені на полі

Таблиця 4.9 – Тест 1.9 – Продаж власності

Початковий стан системи	Гравець є власником поля, яке не закладене
Вхідні дані	Наявність відповідного поля у власності
Опис проведення тесту	Користувач натискає на своє поле, у спливаючому вікні вибирає кнопку продажу, після чого система зараховує компенсацію
Очікуваний результат	Поле закладено, гравець отримав компенсацію. У разі, якщо поле вже закладене, відображається текст помилки
Фактичний результат	Поле успішно закладено, компенсація була отримана

Таблиця 4.10 – Тест 1.10 – Здійснення обміну

Початковий стан системи	Користувач має право ходу
Вхідні дані	Вибрані поля для обміну та кількість залученого балансу
Опис проведення тесту	Користувач натискає на панель іншого гравця, обирає поля та суму валюти для обміну, натискає кнопку пропозиції обміну, після чого пропозиція відправляється отримувачу
Очікуваний результат	Пропозиція обміну успішно відправлена. У разі некоректних умов відображається текст помилки
Фактичний результат	Пропозиція обміну успішно відправлена

Таблиця 4.11 – Тест 1.11 – Прийняття обміну

Початковий стан системи	Гравець отримав пропозицію обміну
Вхідні дані	Вхідна пропозиція обміну
Опис проведення тесту	Користувач переглядає пропозицію обміну, натискає на кнопку прийняття обміну, після чого система оновлює поля та статки гравців
Очікуваний результат	Обмін успішно здійснено, поля та статки оновлено
Фактичний результат	Обмін успішно здійснено, поля та статки оновлено

Таблиця 4.12 – Тест 1.12 – Відхилення обміну

Початковий стан системи	Гравець отримав пропозицію обміну
Вхідні дані	Вхідна пропозиція обміну
Опис проведення тесту	Користувач переглядає пропозицію обміну, натискає на кнопку відхилення, після чого система повертає контроль до поточного гравця.
Очікуваний результат	Обмін відхилено, контроль повернуто до поточного гравця
Фактичний результат	Обмін успішно відхилено, контроль повернуто

Таблиця 4.13 – Тест 1.13 – Добровільна задача

Початковий стан системи	Користувач бере участь у грі
Вхідні дані	Відсутні
Опис проведення тесту	Натискання на власну панель та підтвердження задачі
Очікуваний результат	Користувач натискає на власну бокову панель, у спливаючому діалоговому вікні підтверджує бажання здатися, після чого система завершує його участь у грі
Фактичний результат	Гравець успішно здався, позбувся власності та перейшов у режим глядача

Таблиця 4.14 – Тест 1.14 – Оновлення значення рейтингу гравця

Початковий стан системи	Користувач завершив рейтинговий матч і знаходиться в головному меню
Вхідні дані	Виграш, або програш у матчі
Опис проведення тесту	Користувач завершує рейтинговий матч, система обробляє результати гри, оновлює значення рейтингу гравця на основі його результатів і відображає оновлене значення рейтингу в профілі користувача
Очікуваний результат	Значення рейтингу гравця оновлено коректно, оновлення відображено локально та віддалено
Фактичний результат	Значення рейтингу гравця успішно оновлено, відображено в профілі локально та віддалено

4.3 Опис контрольного прикладу

Для демонстрації вирішених функціональних задач буде наведений опис контрольного прикладу, який складається зі скріншотів стану програмного застосунку на різних етапах його виконання, що дозволяє продемонструвати коректність отриманих результатів.

Демонстрацію роботи застосунку можна побачити на рисунках 4.2 – 4.18.

Початковий стан застосунку після процесу ініціалізації підключення до ігрових сервісів гри наведено на рисунку 4.2.



Рисунок 4.2 – Початковий стан гри після запуску

Користувач має можливість вибору типу лобі, до якого бажає приєднатись, відповідний стан інтерфейсу наведено на рисунку 4.3.

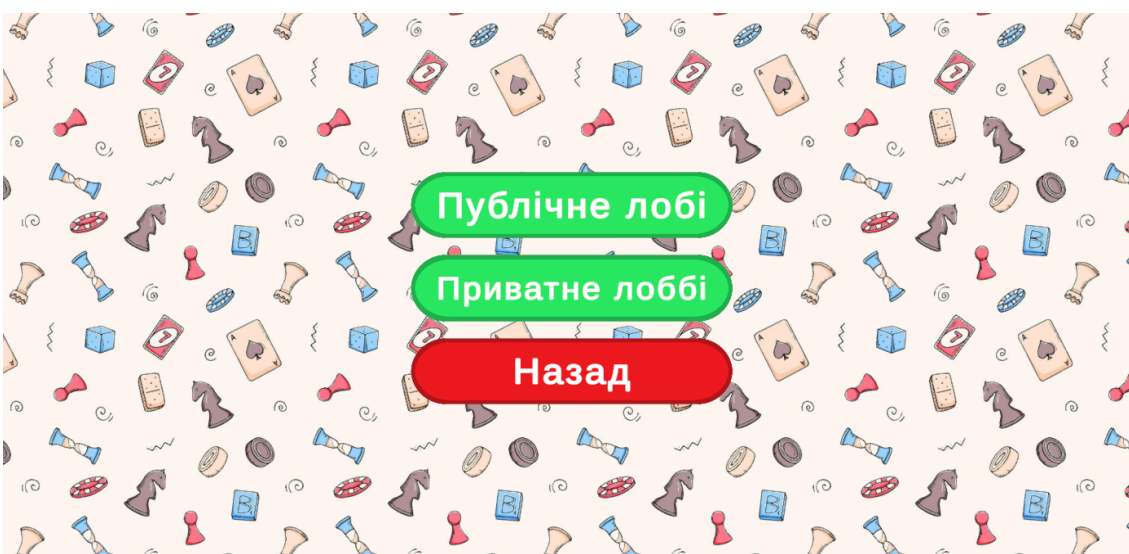


Рисунок 4.3 – Етап вибору типу лобі

Користувач може обрати свою бажань роль й бути хостом ігрової сесії, або звичайним клієнтом, відповідний стан інтерфейсу наведено на рисунку 4.4.



Рисунок 4.4 – Вибір варіанту підключення до приватного лобі

У випадку, якщо користувач обрав приватне лобі й роль хоста, то після процесу завантаження, гравець опиняється у приватній ігровій сесії, де наведено код підключення, яким можна поділитися з іншими гравцями. Відповідний стан лобі наведено на рисунку 4.5.

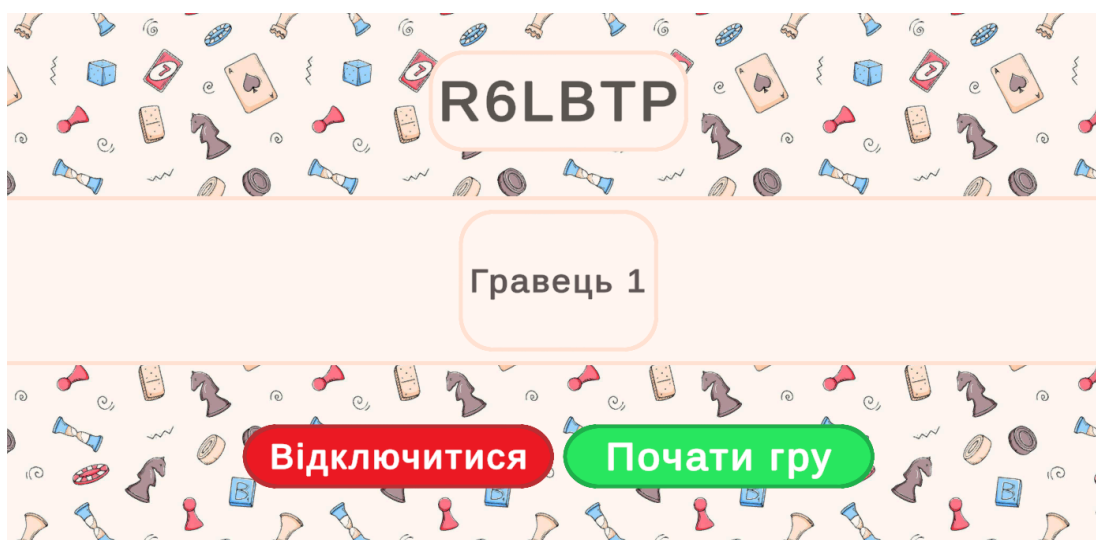


Рисунок 4.5 – Хост знаходиться у приватному лобі

Користувач-клієнт може під'єднатися до приватного лобі за наданим кодом лобі від хоста, відповідний стан інтерфейсу наведено на рисунку 4.6 та оновлене лобі зображено на рисунку 4.7.



Рисунок 4.6 – Введений код лобі у ролі звичайного клієнта



Рисунок 4.7 – Обидва гравці знаходяться у лобі

Стан гри після отримання права ходу гравцем наведено на рисунку 4.8.



Рисунок 4.8 – Гра на етапі отримання права ходу локальним гравцем

Демонстрацію безпосереднього процесу гри наведено на рисунках 4.9 – 4.11, де наведено придбання поля власності (рисунок 4.9), проведення обміну (рисунки 4.10 – 4.11).

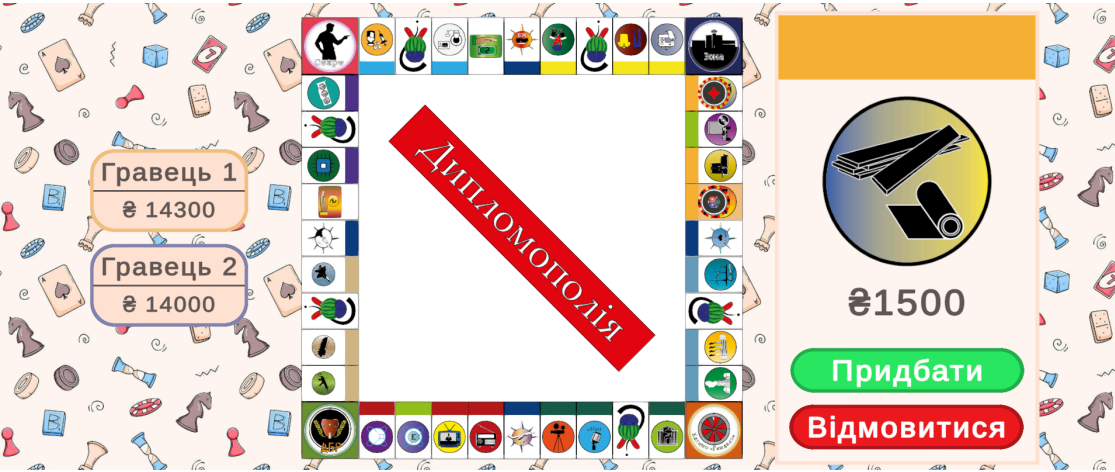


Рисунок 4.9 – Демонстрація можливості придбання поля



Рисунок 4.10 – Демонстрація можливості відправки обміну

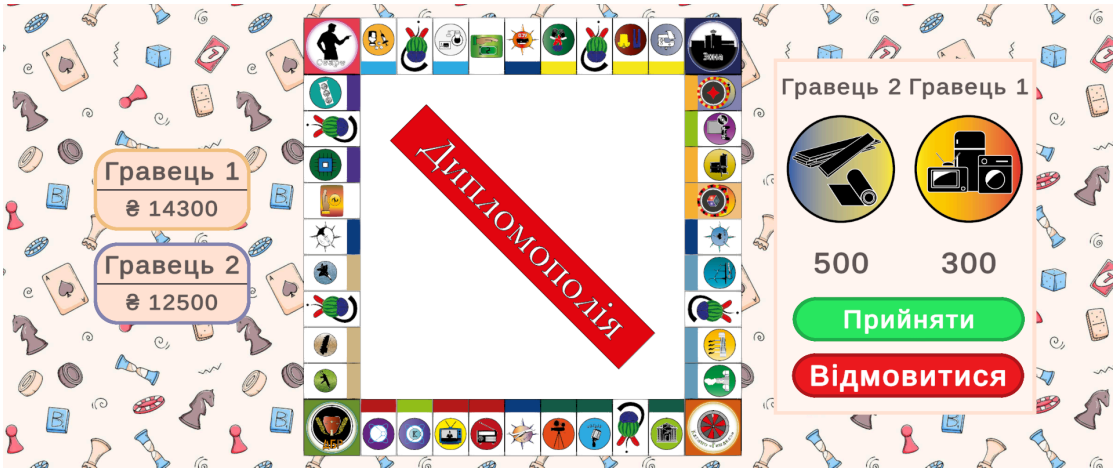


Рисунок 4.11 – Демонстрація можливості отримання обміну

Користувач має право добровільної здачі, відповідне діалогове вікно наведено на рисунку 4.12.

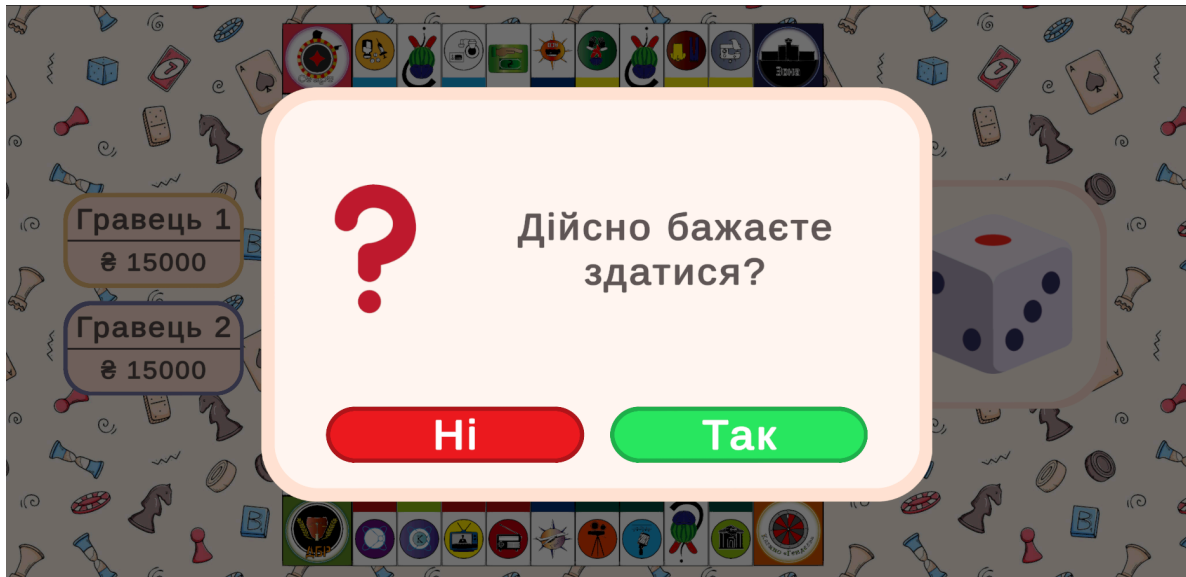


Рисунок 4.12 – Можливість добровільної здачі

Коли у матчі залишається 1 гравець, оскільки решта користувачів здалися, або залишили гру, то виводиться діалогове вікно з повідомленням про перемогу гравця (рисунок 4.13).

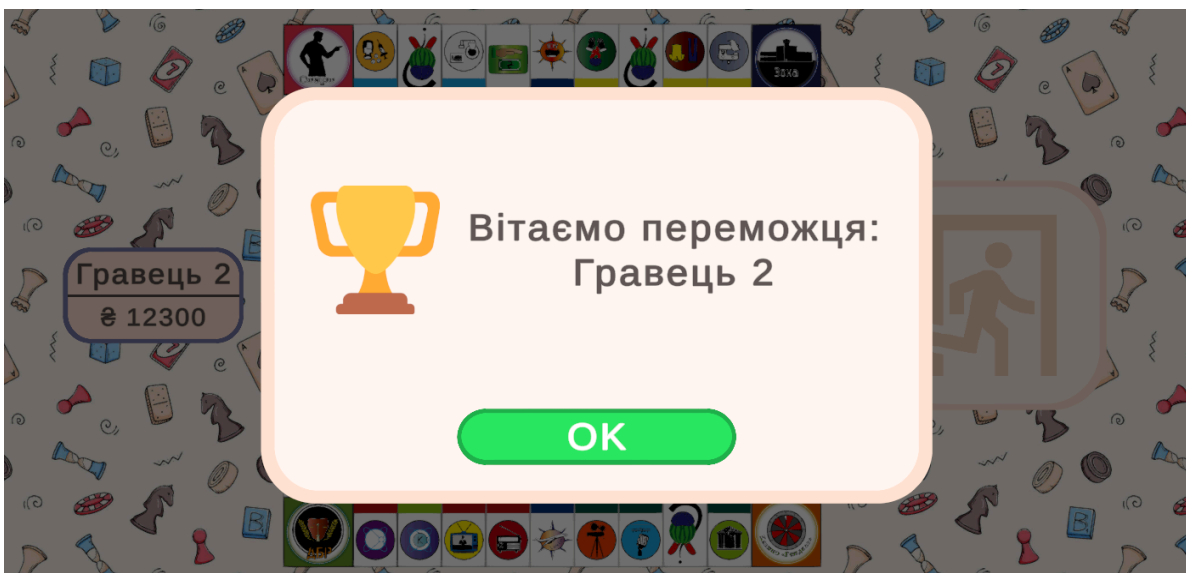


Рисунок 4.13 – Демонстрація вікна-оголошення переможця

У випадку, коли користувач-клієнт обирає публічне лобі, то йому надається можливість запустити пошук доступної публічної гри, демонстрація процесу пошуку наведено на рисунку 4.14.

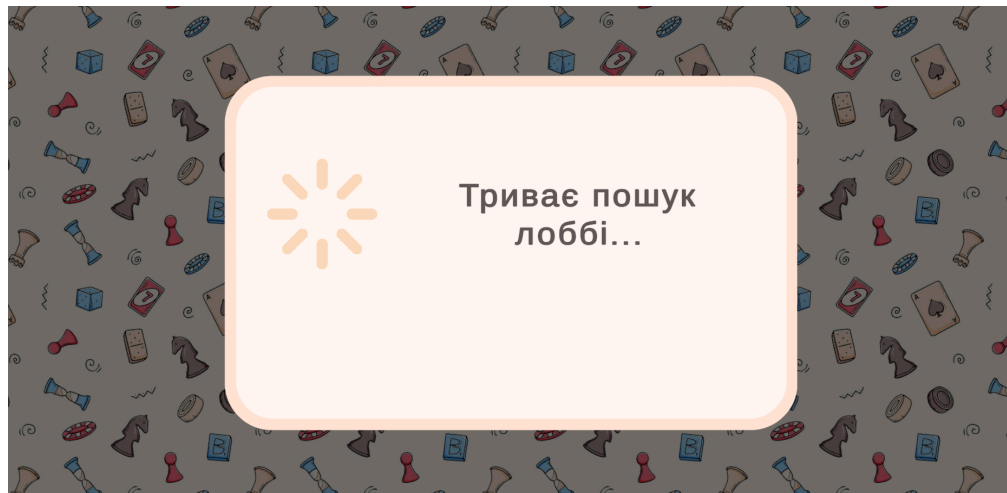


Рисунок 4.14 – Процес пошуку відкритого лобі у вигляді звичайного клієнту

Демонстрацію безпосереднього процесу гри наведено на рисунках 4.15 – 4.17, де наведено приклад сплати за оренду перебування на полі (рисунок 4.15), взаємодію з полем шанс (рисунок 4.16) й покращення власності (рисунок 4.17).



Рисунок 4.15 – Демонстрація меню сплати оренди



Рисунок 4.16 – Приклад взаємодії з полем шанс



Рисунок 4.17 – Меню покращення власності

Користувач має можливість перегляду власного ігрового профілю, де гравець має можливість зміни нікнейму та перегляду власної кількості рейтингу (рисунок 4.18).



Рисунок 4.18 – Перегляд профілю користувача у головному меню

Висновки до розділу

Було здійснено аналіз якості програмного забезпечення кросплатформеної мобільної багатокористувацької гри, яка є цифровою адаптацією настільної гри «Монополія». Для оцінки якості коду використано метрики, зокрема індекс підтримки, цикломатична складність, зв'язність класів та глибина наслідування.

Результати, які були отримані за допомогою інструментів середовища розробки Visual Studio 2022, продемонстрували достатній рівень підтримки коду, хоча деякі окремі підвищені значення метрик обумовлені архітектурними особливостями ігрового рушія Unity, що є типовим для ігрових проєктів.

Процес тестування проводився шляхом мануального виконання тестових кейсів, які охопили всі ключові функціональні вимоги гри, включаючи створення ігрових лобі, взаємодію гравців, ігрові механіки та оновлення профілю користувача. Усі тести були пройдені успішно, що підтверджує стабільність, надійність і відповідність програмного продукту заявленим функціональним вимогам.

Продемонстрований контрольний приклад показав на прикладі коректну роботу застосунку на всіх основних етапах життєвого циклу, починаючи від запуску гри до завершення ігрових сеансів.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Процес розгортання може відбуватися повністю у редакторі рушія Unity. Для цього можна перейти у відповідні вкладки налаштування профілів компіляції проекту на різних платформах [33]. Відкрите меню конфігурації профілів у відповідності до цільової платформи наведено на рисунку 5.1.

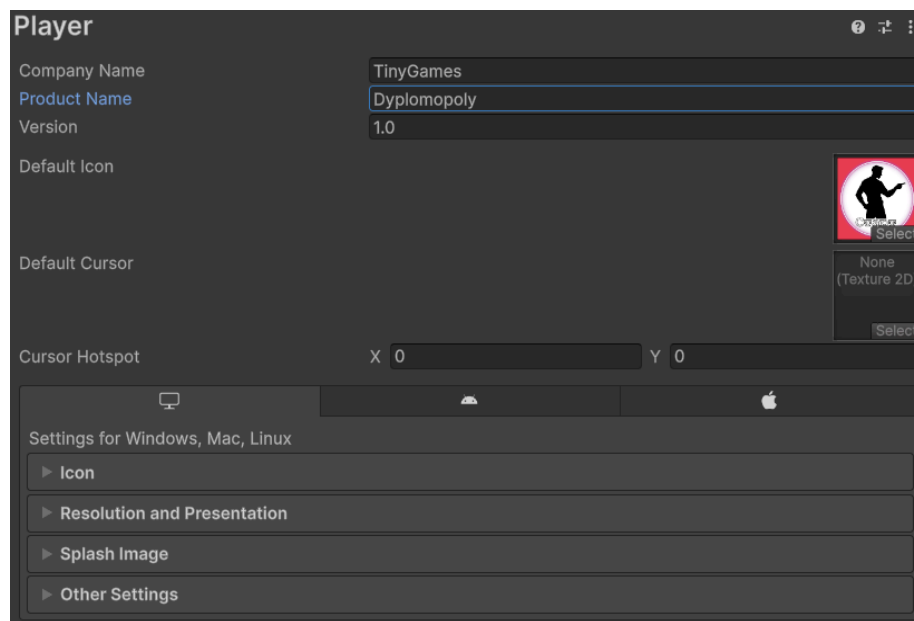


Рисунок 5.1 – Вибір додаткових параметрів цільової платформи

Процес конфігурації автоматичного розгортання додаткових модулів, необхідних для синхронізації роботи профіля користувача та обчислення змін рейтингу наведено на рисунку 5.2 [33].

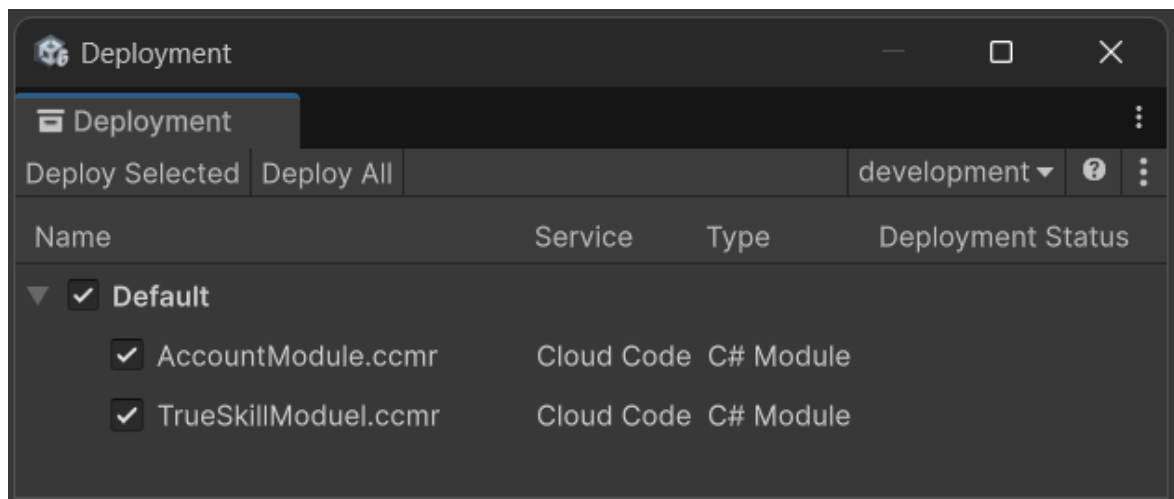


Рисунок 5.2 – Конфігурація процесу розгортання модулів

Безпосередньо сам процес компіляції під обрану платформу можна здійснити у відповідних розділах опцій редактору Unity [33]. Відповідне меню для створення збірки для iOS наведено на рисунку 5.3 а конфігурацію під Android наведено на рисунку 5.4.

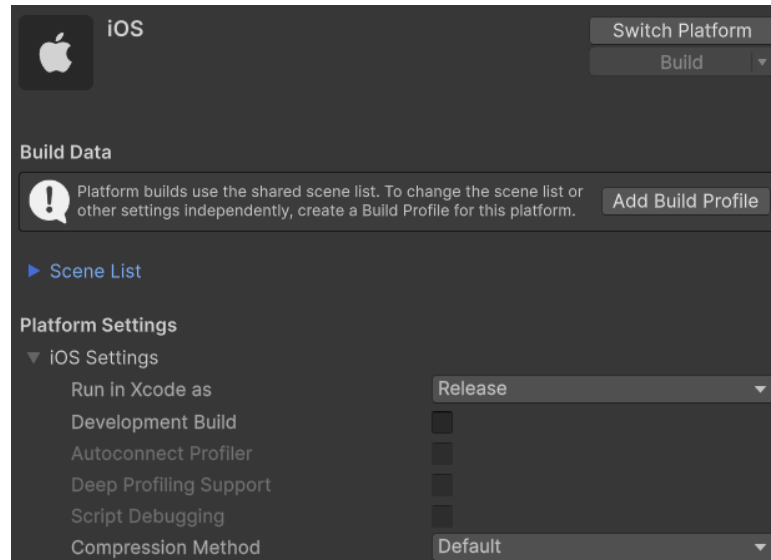


Рисунок 5.3 – Конфігурація збірки під iOS [33]

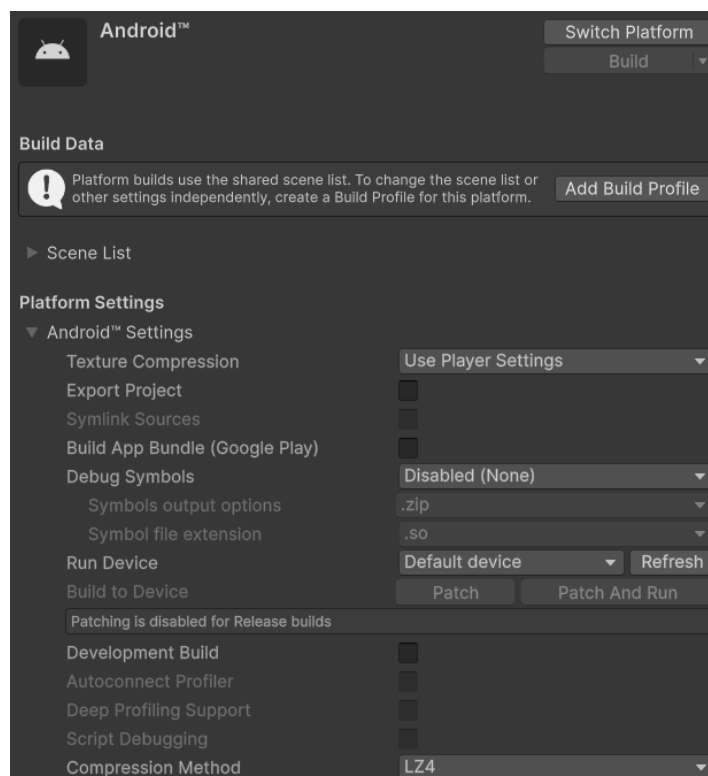


Рисунок 5.4 – Вікно ініціалізації компіляції збірка під Android [33]

Повноцінна діаграма розгортання програмного забезпечення наведена у кресленні 4 графічного матеріалу.

5.2 Супровід програмного забезпечення

Для автоматизації процесу створення нових версій Unity-проєкту для платформ Android та iOS, а також інтеграції модулів Cloud Code, було розроблено CI/CD-пайплайн на основі сервісу GitHub Actions [34].

Процес активується автоматично після додавання до основної гілки репозиторію main коміту з тегом версії. Пайплайн виконує послідовні етапи підготовки середовища з відповідною версією Unity, Android SDK та Xcode, активацію ліцензії Unity, збірку проєкту у вигляді APK-файлу для Android, створення Xcode-проєкту для iOS, пакування та передачу модулів Cloud Code до Unity Gaming Services, архівацію отриманих артефактів та їх збереження у GitHub Releases [34].

Подібний підхід забезпечує ефективну автоматизацію створення релізів, що дозволяє швидко отримувати готові до тестування або публікації версії гри.

Висновки до розділу

Процес розгортання кросплатформеної мобільної гри на базі Unity для Android та iOS виконано шляхом налаштування компіляції в редакторі Unity. Це забезпечило зручну підготовку збірок і інтеграцію модулів для роботи з даними користувачів.

Для автоматизації релізів створено CI/CD-пайплайн на основі GitHub Actions, який активується при оновленні репозиторію. Пайплайн виконує підготовку середовища, компіляцію, інтеграцію з Unity Gaming Services і збереження артефактів, що прискорює створення стабільних версій додатку.

ВИСНОВКИ

Під час розробки кросплатформеної мобільної багатокористувацької гри, яка є адаптацією настільної гри «Монополія» до 5 гравців, було реалізовано програмне забезпечення, яке відповідає всім зазначеним вимогам, оскільки є мінімально залежним від доступності виділених серверів, зовнішніх ресурсів та сервісів.

Для досягнення результату було використано мікросервісну архітектуру, яку було обрано через її гнучкість і можливість незалежного масштабування компонентів, що є перевагою у порівнянні з монолітним підходом. Ігровий рушій Unity разом із бібліотекою Netcode for Game Objects забезпечив можливість кросплатформеної розробки та підтримки впровадження безпечного зашифрованого багатокористувацького режиму. У якості мережевої топології була використана централізована однорангова модель, яка є ефективною для задач централізованої синхронізації клієнтів у мережі без необхідності підключення до виділених серверів, що значно знизило потенційні витрати на інфраструктуру та оркестрацію виділених флотів серверів.

За допомогою сервісу Unity Authentication було реалізовано безпечну автентифікацію клієнта та ініціалізацію ігрової сесії. Unity Lobby Service надав можливість створення та швидкого індексування всіх існуючих ігрових сесій у глобальному контексті всього додатку. Впровадження сервісу зворотнього проксі Unity Relay Service допомогло в організації безпечної зашифрованої комунікації на основі DTLS над UDP для клієнтів та унеможливило публічне поширення IP-адрес клієнтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Kerr, A. The Business and Culture of Digital Games: Gamework/Gameplay. SAGE Publications, 2006. 50-370 с. ISBN: 978-1412900478.
- 2) Donovan, T. Replay: The History of Video Games. Yellow Ant, 2010. 25-475 с. ISBN: 978-0956507204.
- 3) Graetz, J. M. The Origin of Spacewar!. Creative Computing, 1981. 10-130 с. ISBN: 978-0916688035.
- 4) Kent, S. L. The Ultimate History of Video Games: From Pong to Pokémon and Beyond. Three Rivers Press, 2001. 30-630 с. ISBN: 978-0761536437.
- 5) Monopoly, Marmalade Game Studio. [Електронний ресурс]. – Режим доступу: <https://www.marmaladegamestudio.com/games/monopoly/> (дата звернення: 10.05.2025).
- 6) Monopoly GO, Scopely. [Електронний ресурс]. – Режим доступу: <https://www.scopely.com/en/games/monopoly-go> (дата звернення: 10.05.2025).
- 7) Fiedler, G. Multiplayer Game Programming: Architecting Networked Games. Addison-Wesley Professional, 2015. 40-400 с. ISBN: 978-0134034300.
- 8) Smed, J., Hakonen, H. Algorithms and Networking for Computer Games. Wiley, 2017. 2nd ed. 20-350 с. ISBN: 978-1119259763.
- 9) Elo, A. E. The Rating of Chessplayers, Past and Present. Arco Publishing, 1978. 15-200 с. ISBN: 978-0668047210.
- 10) Herbrich, R., Minka, T., Graepel, T. TrueSkill™: A Bayesian Skill Rating System. Microsoft Research, 2007. 5-150 с. ISBN: 978-0735626423.
- 11) White, S. A., Miers, D. BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc., 2008. 30-300 с. ISBN: 978-0975528112.
- 12) Sommerville, I. Software Engineering. Pearson, 2015. 10th ed. 40-500 с. ISBN: 978-0133943030.
- 13) Horton, J. Android Game Programming by Example. Packt Publishing, 2015. 20-250 с. ISBN: 978-1785280122.

14) Unity Android Requirements, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity3d.com/6000.1/Documentation/Manual/android-requirements-and-compatibility.html> (дата звернения: 13.05.2025).

15) Unity iOS Requirements, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity3d.com/6000.1/Documentation/Manual/ios-requirements-and-compatibility.html> (дата звернения: 13.05.2025).

16) Designing UI for Multiple Resolutions, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/HOWTO-UIMultiResolution.html> (дата звернения: 13.05.2025).

17) Boehm B. W., Abts C., Brown A. W., Chulani S., Clark B. K., Horowitz E., Madachy R., Reifer D. J., Steece B. Software Cost Estimation with COCOMO II. Prentice Hall, 2000. 10-400 с. ISBN: 978-0-13-026692-7.

18) Bass L., Clements P., Kazman R. Software Architecture in Practice. Addison-Wesley, 2003. 20-450 с. ISBN: 978-0-321-15495-8.

19) Monolithic Architecture: Advantages and Disadvantages. [Электронный ресурс]. – Режим доступа: <https://medium.com/@datamify/monolithic-architecture-advantages-and-disadvantages-e71a603eec89> (дата звернения: 14.05.2025).

20) Advantages and Disadvantages of Microservices. [Электронный ресурс]. – Режим доступа: <https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservices> (дата звернения: 15.05.2025).

21) Unity Authentication, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity.com/ugs/manual/authentication/manual/overview> (дата звернения: 16.05.2025).

22) Unity Lobby, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity.com/ugs/manual/lobby/manual/unity-lobby-service> (дата звернения: 16.05.2025).

23) Unity Relay, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity.com/ugs/manual/relay/manual/introduction> (дата звернения: 16.05.2025).

24) Tanenbaum A. S., Wetherall D. J. Computer Networks. Prentice Hall, 2010. 30-900 с. ISBN: 978-0-13-212695-3.

25) Cross-Platform Considerations, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity3d.com/2023.2/Documentation/Manual/CrossPlatformConsiderations.html> (дата звернения: 14.05.2025).

26) Mirror Networking. [Электронный ресурс]. – Режим доступа: <https://github.com/MirrorNetworking/Mirror> (дата звернения: 15.05.2025).

27) Netcode for GameObjects, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs-multiplayer.unity3d.com/> (дата звернения: 16.05.2025).

28) Entities overview, Unity Technologies. [Электронный ресурс]. – Режим доступа: <https://docs.unity3d.com/Packages/com.unity.entities@1.1/manual/index.html> (дата звернения: 16.05.2025).

29) Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. Pearson, 2012. 40-850 с. ISBN: 978-0-13-285620-1.

30) How code metrics help identify risks. [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2022> (дата звернения: 17.05.2025).

31) Simplifying Game Development with Component-Based Architecture in Unity. [Электронный ресурс]. – Режим доступа:

<https://azumo.com/insights/simplifying-game-development-with-component-based-architecture-in-unity> (дата звернення: 17.05.2025).

32) The Benefits and Challenges of Test Automation in Game Development. [Електронний ресурс]. – Режим доступу: <https://dev.to/wetest/the-benefits-and-challenges-of-test-automation-in-game-development-43h3> (дата звернення: 17.05.2025).

33) Building and Running Your Project, Unity Documentation. [Електронний ресурс]. – Режим доступу: <https://docs.unity3d.com/Manual/BuildSettings.html> (дата звернення: 15.05.2025).

34) What is CI-CD? How to build CI-CD pipeline in Android using GitHub actions, Kaushal Vasava. [Електронний ресурс]. – Режим доступу: <https://medium.com/@KaushalVasava/what-is-ci-cd-how-to-build-ci-cd-pipeline-in-android-using-github-actions-60f69805657b> (дата звернення: 16.05.2025).

ДОДАТКИ

Додаток А – Звіт подібності



Дата звіту 6/5/2025
Дата редагування 6/5/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації

National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute

Заголовок

ІП-13_Бабіч_ПЗ

Автор

Науковий керівник / Експерт

ІП-13_БабічРодіонов П.Ю.

підрозділ

ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



10

Довжина фрази для коефіцієнта подібності 2

11252

Кількість слів

90177

Кількість символів

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

Кросплатформна багатокористувацька сесійна реалізація настільної гри

«Монополія» на Unity

Текст програми

КПІ.ІП-1303.045490.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Павло РОДІОНОВ

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Денис БАБІЧ

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду

<https://github.com/NgeNXQ/Monopoly>

Файл LobbyManager.cs

Реалізація системи підключення до ігрового лобі

```
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Unity.Netcode;
using Unity.Services.Lobbies;
using Unity.Services.Lobbies.Models;
using UnityEngine.SceneManagement;
using Monopoly.Unity.Services.Lobbies.Models.Extensions;
using Monopoly.Client.Shared.Utilities.Loaders;
using Monopoly.Client.Runtime.UI.Views.Concrete.Global;
using Monopoly.Client.Runtime.UI.Managers.Game;
using Monopoly.Client.Runtime.UI.Managers.Lobby;
using Monopoly.Client.Runtime.UI.Managers.Global;
using Monopoly.Client.Runtime.App;

namespace Monopoly.Client.Runtime.Core.P2P
{
    internal sealed class LobbyManager
    {
        private const float CONFIG_UPTIME = 5.0f;

        internal const int MIN_PLAYERS = 1;
        internal const int MAX_PLAYERS = 5;

        internal const string LOBBY_DATA_KEY_STATE = "state";
        internal const string LOBBY_DATA_VALUE_STATE_IDLE = "state_idle";
        internal const string LOBBY_DATA_VALUE_STATE_GAME = "state_game";
        internal const string LOBBY_DATA_VALUE_STATE_PENDING = "state_pending";
        internal const string LOBBY_DATA_VALUE_STATE_LOADING = "state_loading";
        internal const string LOBBY_DATA_VALUE_STATE_RETURNING = "state_returning";

        internal const string LOBBY_DATA_KEY_RELAY_CODE = "relay_code";
```

```

internal const string LOBBY_DATA_KEY_GAME_MODE = "game_mode";
internal const string LOBBY_DATA_VALUE_GAME_MODE_RANKED = "game_mode_ranked";
internal const string LOBBY_DATA_VALUE_GAME_MODE_UNRANKED = "game_mode_unranked";

```

```

private ILobbyEvents localLobbyEvents;
private CancellationTokenSource heartbeatTokenSource;

```

```

internal static LobbyManager Instance { get; private set; }

```

```

internal bool IsHost { get; private set; }
internal bool IsClient { get; private set; }
internal string LobbyCode { get; private set; }
internal string RelayCode { get; private set; }
internal Lobby LocalLobby { get; private set; }
internal LobbyEventCallbacks LocalLobbyEventCallbacks { get; private set; }

```

```

static LobbyManager()
{
    LobbyManager.Instance = new LobbyManager();
}

```

```

private LobbyManager()
{
    this.LocalLobbyEventCallbacks = new LobbyEventCallbacks();

    this.LocalLobbyEventCallbacks.PlayerLeft += this.OnPlayerLeft;
    this.LocalLobbyEventCallbacks.DataChanged += this.OnDataChanged;
    this.LocalLobbyEventCallbacks.PlayerJoined += this.OnPlayerJoined;
    this.LocalLobbyEventCallbacks.KickedFromLobby += this.OnKickedFromLobbyAsync;
    this.LocalLobbyEventCallbacks.PlayerDataChanged += this.OnPlayerDataChanged;

    SceneManager.activeSceneChanged += this.OnActiveSceneChanged;

    if (NetworkManager.Singleton != null)
        NetworkManager.Singleton.OnTransportFailure += this.OnTransportFailureAsync;
}

```

```

~LobbyManager()
{
    if (this.IsHost)
        this.StopHeartbeat();
}

```

```

this.LocalLobbyEventCallbacks.PlayerLeft -= this.OnPlayerLeft;
this.LocalLobbyEventCallbacks.DataChanged -= this.OnDataChanged;
this.LocalLobbyEventCallbacks.PlayerJoined -= this.OnPlayerJoined;
this.LocalLobbyEventCallbacks.KickedFromLobby -= this.OnKickedFromLobbyAsync;
this.LocalLobbyEventCallbacks.PlayerDataChanged -= this.OnPlayerDataChanged;

SceneManager.activeSceneChanged -= this.OnActiveSceneChanged;

if (NetworkManager.Singleton != null)
    NetworkManager.Singleton.OnTransportFailure -= this.OnTransportFailureAsync;
}

private async void OnActiveSceneChanged(Scene previousScene, Scene currentScene)
{
    if (this.LocalLobby == null)
        return;

    await SessionManager.Instance.LocalPlayer.SetDataRemotely(
        this.LocalLobby.Id,
        SessionManager.PLAYER_DATA_KEY_SCENE,
        currentScene.name,
        PlayerDataObject.VisibilityOptions.Member
    );
}

internal async Task HostPublicLobbyAsync(string relayCode)
{
    await this.HostLobby(relayCode, LobbyManager.LOBBY_DATA_VALUE_GAME_MODE_RANKED);
}

internal async Task HostPrivateLobbyAsync(string relayCode)
{
    await this.HostLobby(relayCode, LobbyManager.LOBBY_DATA_VALUE_GAME_MODE_UNRANKED);
}

private async Task HostLobby(string relayCode, string gameMode)
{
    this.IsHost = true;
    this.IsClient = false;

    DataObject gameModeData = new DataObject(
        DataObject.VisibilityOptions.Public,

```

```

        gameMode,
        DataObject.IndexOptions.S1
    );

    DataObject relayCodeData = new DataObject(
        DataObject.VisibilityOptions.Member,
        relayCode
    );

    DataObject lobbyStateData = new DataObject(
        DataObject.VisibilityOptions.Member,
        LobbyManager.LOBBY_DATA_VALUE_STATE_IDLE
    );

    CreateLobbyOptions lobbyOptions = new CreateLobbyOptions()
    {
        Data = new Dictionary<string, DataObject>()
        {
            { LobbyManager.LOBBY_DATA_KEY_STATE, lobbyStateData },
            { LobbyManager.LOBBY_DATA_KEY_GAME_MODE, gameModeData },
            { LobbyManager.LOBBY_DATA_KEY_RELAY_CODE, relayCodeData },
        },
        Player = SessionManager.Instance.LocalPlayer,
    };

    try
    {
        this.LocalLobby = await LobbyService.Instance.CreateLobbyAsync(
            SessionManager.Instance.LocalPlayer.Id,
            LobbyManager.MAX_PLAYERS,
            lobbyOptions
        );

        this.localLobbyEvents = await LobbyService.Instance.SubscribeToLobbyEventsAsync(
            this.LocalLobby.Id,
            this.LocalLobbyEventCallbacks
        );

        this.RelayCode = relayCode;
        this.LobbyCode = this.LocalLobby.LobbyCode;

        this.StartHeartbeat(this.LocalLobby.Id);
    }

```

```

    }
    catch
    {
        throw;
    }
}

private void StopHeartbeat()
{
    this.heartbeatTokenSource?.Cancel();
    this.heartbeatTokenSource?.Dispose();
    this.heartbeatTokenSource = null;
}

private void StartHeartbeat(string lobbyId)
{
    this.StopHeartbeat();

    this.heartbeatTokenSource = new CancellationTokenSource();

    Task.Run(() => this.RunHeartbeatLoopAsync(lobbyId, heartbeatTokenSource.Token));
}

private async Task RunHeartbeatLoopAsync(string lobbyId, CancellationToken cancellationToken)
{
    const int MILLISECONDS_PER_SECOND = 1000;

    while (!cancellationToken.IsCancellationRequested)
    {
        await LobbyService.Instance.SendHeartbeatPingAsync(lobbyId);

        await Task.Delay(
            (int)(LobbyManager.CONFIG_UPTIME * MILLISECONDS_PER_SECOND),
            cancellationToken
        );
    }
}

internal async Task JoinLobbyAsync(string joinCode)
{
    this.IsHost = false;
    this.IsClient = true;
}

```

```

JoinLobbyByCodeOptions lobbyOptions = new JoinLobbyByCodeOptions()
{
    Player = SessionManager.Instance.LocalPlayer
};

try
{
    this.LocalLobby = await LobbyService.Instance.JoinLobbyByCodeAsync(
        joinCode,
        lobbyOptions
    );

    this.localLobbyEvents = await LobbyService.Instance.SubscribeToLobbyEventsAsync(
        this.LocalLobby.Id,
        this.LocalLobbyEventCallbacks
    );

    this.LobbyCode = this.LocalLobby.LobbyCode;
    this.RelayCode = this.LocalLobby.GetData(LobbyManager.LOBBY_DATA_KEY_RELAY_CODE);
}
catch
{
    throw;
}
}

internal async Task FindLobbyAsync()
{
    this.IsHost = false;
    this.IsClient = true;

    QuickJoinLobbyOptions lobbyOptions = new QuickJoinLobbyOptions
    {
        Filter = new List<QueryFilter>
        {
            new QueryFilter(
                QueryFilter.FieldOptions.S1,
                LobbyManager.LOBBY_DATA_VALUE_GAME_MODE_RANKED,
                QueryFilter.OpOptions.EQ
            )
        },
    }
}

```

```

        Player = SessionManager.Instance.LocalPlayer,
    };

    try
    {
        this.LocalLobby = await LobbyService.Instance.QuickJoinLobbyAsync(lobbyOptions);

        this.localLobbyEvents = await LobbyService.Instance.SubscribeToLobbyEventsAsync(
            this.LocalLobby.Id,
            this.LocalLobbyEventCallbacks
        );

        this.LobbyCode = this.LocalLobby.LobbyCode;
        this.RelayCode = this.LocalLobby.GetData(LobbyManager.LOBBY_DATA_KEY_RELAY_CODE);
    }
    catch
    {
        throw;
    }
}

internal async void StartGame()
{
    if (!this.LocalLobby.ArePlayersSynchronized(SessionManager.PLAYER_DATA_KEY_SCENE,
        SceneManager.GetActiveScene().name))
        return;

    await this.LocalLobby.SetIsPrivate(true);

    await this.LocalLobby.SetData(
        LobbyManager.LOBBY_DATA_KEY_STATE,
        LobbyManager.LOBBY_DATA_VALUE_STATE_LOADING,
        DataObject.VisibilityOptions.Member
    );

    await SceneLoaderUtility.LoadNetworkSceneAsync(MonopolyApplication.Instance.SceneAssetGame,
        LoadSceneMode.Single);
}

internal async Task<bool> DoesLocalLobbyExistAsync()
{
    try

```

```

{
    QueryLobbiesOptions lobbyOptions = new QueryLobbiesOptions()
    {
        Filters = new List<QueryFilter>()
        {
            new QueryFilter(QueryFilter.FieldOptions.Name, this.LocalLobby.Name,
QueryFilter.OpOptions.EQ)
        }
    };

    QueryResponse queryResponse = await LobbyService.Instance.QueryLobbiesAsync(lobbyOptions);
}
catch
{
    return false;
}

return true;
}

internal async Task QuitLobbyAsync()
{
    await this?.localLobbyEvents?.UnsubscribeAsync();

    if (this.IsHost)
    {
        if (await this.DoesLocalLobbyExistAsync())
            await LobbyService.Instance?.DeleteLobbyAsync(this.LocalLobby.Id);
    }
    else
    {
        if (await this.DoesLocalLobbyExistAsync() &&
this.LocalLobby.Players.Contains(SessionManager.Instance.LocalPlayer))
            await LobbyService.Instance?.RemovePlayerAsync(this.LocalLobby.Id,
SessionManager.Instance.LocalPlayer.Id);
    }

    this.IsHost = false;
    this.IsClient = false;
    this.LocalLobby = null;
}

```



```

internal async Task KickFromLobbyAsync(string playerId)
{
    await LobbyService.Instance.RemovePlayerAsync(this.LocalLobby.Id, playerId);
}

private async void OnKickedFromLobbyAsync()
{
    await SessionManager.Instance?.QuitLobbyAsync();
}

private async void OnTransportFailureAsync()
{
    await this.QuitLobbyAsync();
}

private void OnPlayerLeft(List<int> leftPlayers)
{
    foreach (int playerIndex in leftPlayers)
        this.LocalLobby.Players.RemoveAt(playerIndex);
}

private void OnPlayerJoined(List<LobbyPlayerJoined> joinedPlayers)
{
    foreach (LobbyPlayerJoined newPlayer in joinedPlayers)
        this.LocalLobby.Players.Add(newPlayer.Player);
}

private void OnDataChanged(Dictionary<string, ChangedOrRemovedLobbyValue<DataObject>>
changedLobbyData)
{
    foreach (string key in changedLobbyData.Keys)
        this.LocalLobby.Data[key] = changedLobbyData[key].Value;

    switch (this.LocalLobby.Data[LobbyManager.LOBBY_DATA_KEY_STATE].Value)
    {
        case LobbyManager.LOBBY_DATA_VALUE_STATE_LOADING:
        case LobbyManager.LOBBY_DATA_VALUE_STATE_PENDING:
            UIManagerGlobal.Instance.ShowMessageBoxNone(
                UIManagerLobby.Instance?.MessagePendingGame ??
                UIManagerGame.Instance?.MessageWaitingOtherPlayers,
                MessageBoxView.Icon.Loading,
                callback: () => this.LocalLobby.ArePlayersSynchronized(

```

```

        SessionManager.PLAYER_DATA_KEY_SCENE,
        SceneManager.GetActiveScene().name
    )
);
break;
case LobbyManager.LOBBY_DATA_VALUE_STATE_RETURNING:
    UIManagerGlobal.Instance.ShowMessageBoxNone(
        UIManagerLobby.Instance?.MessageFailedToConnect ??
UIManagerGame.Instance?.MessagePlayersFailedToLoad,
        MessageBoxView.Icon.Loading,
        callback: () => this.LocalLobby.ArePlayersSynchronized(
            SessionManager.PLAYER_DATA_KEY_SCENE,
            SceneManager.GetActiveScene().name
        )
    );
break;
}
}

private void OnPlayerDataChanged(Dictionary<int, Dictionary<string,
ChangedOrRemovedLobbyValue<PlayerDataObject>>> changedPlayerData)
{
    foreach (int playerIndex in changedPlayerData.Keys)
    {
        foreach (string key in changedPlayerData[playerIndex].Keys)
        {
            this.LocalLobby.Players[playerIndex].Data[key] = changedPlayerData[playerIndex][key].Value;
        }
    }
}
}
}

```

Файл SessionManager.cs

Реалізація системи управління сесією підключення

```

using System;
using System.Net;
using System.Threading.Tasks;
using System.Collections.Generic;
using Unity.Netcode;
using Unity.Netcode.Transports.UTP;
using Unity.Services.Core;

```

```

using Unity.Services.Relay;
using Unity.Services.Relay.Models;
using Unity.Services.Lobbies;
using Unity.Services.Lobbies.Models;
using Unity.Services.CloudCode;
using Unity.Services.Authentication;
using Unity.Networking.Transport.Relay;
using UnityEngine.SceneManagement;
using Newtonsoft.Json;
using Monopoly.Unity.Services.Lobbies.Models.Extensions;
using Monopoly.Client.Shared.Utilities.Loaders;
using Monopoly.Client.Runtime.App;
using Monopoly.Client.Runtime.Core.API;
using Monopoly.Client.Runtime.Core.DTO;
using Monopoly.Backend.Gateway.Modules.Account;
using Monopoly.Backend.Gateway.Modules.TrueSkill;

namespace Monopoly.Client.Runtime.Core.P2P
{
    internal sealed class SessionManager
    {
        private const string CONFIG_CONNECTION_TYPE = "dtls";

        internal const string PLAYER_DATA_KEY_SCENE = "player_data_scene";
        internal const string PLAYER_DATA_KEY_NICKNAME = "player_data_nickname";
        internal const string PLAYER_DATA_KEY_TROPHIES = "player_data_trophies";

        internal static SessionManager Instance { get; private set; }

        internal event Action AuthenticationFailedEvent;
        internal event Action<RelayServiceException> RelayConnectionFailedEvent;
        internal event Action<LobbyServiceException> LobbyConnectionFailedEvent;

        internal Player LocalPlayer { get; private set; }
        internal AccountModuleBindings ModuleAccount { get; private set; }
        internal TrueSkillModuleBindings ModuleTrueSkill { get; private set; }

        static SessionManager()
        {
            SessionManager.Instance = new SessionManager();
        }
    }
}

```

```

private SessionManager()
{
}

internal async Task InitializeAsync()
{
    try
    {
        await UnityServices.InitializeAsync();

        await AuthenticationService.Instance.SignInAnonymouslyAsync();

        this.ModuleAccount = new AccountModuleBindings(CloudCodeService.Instance);
        this.ModuleTrueSkill = new TrueSkillModuleBindings(CloudCodeService.Instance);
    }
    catch
    {
        this.AuthenticationFailedEvent?.Invoke();
    }

    await this.InitializeLocalPlayer();

    await SceneLoaderUtility.LoadDefaultSceneAsync(MonopolyApplication.Instance.SceneAssetMenu,
LoadSceneMode.Single);
}

private async Task InitializeLocalPlayer()
{
    string accountGetResponse = await this.ModuleAccount.GetAccount();

    ApiResponse<AccountPayload> parsedAccountResponse =
JsonConvert.DeserializeObject<ApiResponse<AccountPayload>>(accountGetResponse);

    if (parsedAccountResponse.Code == HttpStatusCode.NotFound)
    {
        accountGetResponse = await this.ModuleAccount.PostAccount("<blank>", "0");

        parsedAccountResponse =
JsonConvert.DeserializeObject<ApiResponse<AccountPayload>>(accountGetResponse);
    }

    string playerTrophies = parsedAccountResponse.Payload.Trophies;
    string playerNickname = parsedAccountResponse.Payload.Nickname;

```

```

PlayerDataObject nicknamePlayerData = new PlayerDataObject(
    PlayerDataObject.VisibilityOptions.Member,
    playerNickname
);

PlayerDataObject trophiesPlayerData = new PlayerDataObject(
    PlayerDataObject.VisibilityOptions.Member,
    playerTrophies
);

PlayerDataObject scenePlayerData = new PlayerDataObject(
    PlayerDataObject.VisibilityOptions.Member,
    SceneManager.GetActiveScene().name
);

this.LocalPlayer = new Player(AuthenticationService.Instance.PlayerId)
{
    Data = new Dictionary<string, PlayerDataObject>
    {
        { SessionManager.PLAYER_DATA_KEY_SCENE, scenePlayerData },
        { SessionManager.PLAYER_DATA_KEY_NICKNAME, nicknamePlayerData },
        { SessionManager.PLAYER_DATA_KEY_TROPHIES, trophiesPlayerData },
    }
};
}

internal async Task UpdatePlayerNickname(string newNickname)
{
    this.LocalPlayer.SetDataLocally(
        SessionManager.PLAYER_DATA_KEY_NICKNAME,
        newNickname,
        PlayerDataObject.VisibilityOptions.Member
    );

    await SessionManager.Instance.ModuleAccount.PutAccountNickname(newNickname);
}

internal async Task HostPublicLobbyAsync()
{
    await HostLobbyAsync(true);
}

```

```

internal async Task HostPrivateLobbyAsync()
{
    await HostLobbyAsync(false);
}

private async Task HostLobbyAsync(bool isPublic)
{
    try
    {
        Allocation hostAllocation = await
RelayService.Instance.CreateAllocationAsync(LobbyManager.MAX_PLAYERS);
        string relayCode = await RelayService.Instance.GetJoinCodeAsync(hostAllocation.AllocationId);

        if (isPublic)
            await LobbyManager.Instance?.HostPublicLobbyAsync(relayCode);
        else
            await LobbyManager.Instance?.HostPrivateLobbyAsync(relayCode);

        RelayServerData relayServerData = AllocationUtils.ToRelayServerData(hostAllocation,
SessionManager.CONFIG_CONNECTION_TYPE);

        NetworkManager.Singleton?.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);
        NetworkManager.Singleton?.StartHost();

        await SceneLoaderUtility.LoadDefaultSceneAsync(MonopolyApplication.Instance.SceneAssetLobby,
LoadSceneMode.Single);
    }
    catch (RelayServiceException relayServiceException)
    {
        this.RelayConnectionFailedEvent?.Invoke(relayServiceException);
    }
    catch (LobbyServiceException lobbyServiceException)
    {
        this.LobbyConnectionFailedEvent?.Invoke(lobbyServiceException);
    }
}

internal async Task JoinLobbyAsync(string joinCode)
{
    try
    {
        await LobbyManager.Instance?.JoinLobbyAsync(joinCode);
    }
}

```

```

JoinAllocation clientAllocation = await
RelayService.Instance.JoinAllocationAsync(LobbyManager.Instance.RelayCode);
RelayServerData relayServerData = AllocationUtils.ToRelayServerData(clientAllocation,
SessionManager.CONFIG_CONNECTION_TYPE);

    NetworkManager.Singleton?.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);
    NetworkManager.Singleton?.StartClient();
}
catch (RelayServiceException relayServiceException)
{
    this.RelayConnectionFailedEvent?.Invoke(relayServiceException);
}
catch (LobbyServiceException lobbyServiceException)
{
    this.LobbyConnectionFailedEvent?.Invoke(lobbyServiceException);
}
}

internal async Task FindLobbyAsync()
{
    try
    {
        await LobbyManager.Instance?.FindLobbyAsync();

JoinAllocation clientAllocation = await
RelayService.Instance.JoinAllocationAsync(LobbyManager.Instance.RelayCode);
RelayServerData relayServerData = AllocationUtils.ToRelayServerData(clientAllocation,
SessionManager.CONFIG_CONNECTION_TYPE);

    NetworkManager.Singleton?.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);
    NetworkManager.Singleton?.StartClient();
}
catch (RelayServiceException relayServiceException)
{
    this.RelayConnectionFailedEvent?.Invoke(relayServiceException);
}
catch (LobbyServiceException lobbyServiceException)
{
    this.LobbyConnectionFailedEvent?.Invoke(lobbyServiceException);
}
}

```

```

internal async Task QuitLobbyAsync()
{
    await LobbyManager.Instance?.QuitLobbyAsync();

    NetworkManager.Singleton?.Shutdown();

    await SceneLoaderUtility.LoadDefaultSceneAsync(MonopolyApplication.Instance.SceneAssetMenu,
LoadSceneMode.Single);
}
}
}

```

Файл PawnController.cs

Базовий клас контролеру для гравця та бота

```

using System.Linq;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Unity.Netcode;
using Unity.Netcode.Components;
using Monopoly.Client.Shared.Patterns.Factories;
using Monopoly.Client.Shared.Patterns.Observables;
using Monopoly.Client.Runtime.UI.Managers.Game;
using Monopoly.Client.Runtime.Game.Managers;
using Monopoly.Client.Runtime.Game.Gameplay.Tiles.Common;
using Monopoly.Client.Runtime.Game.Gameplay.Tiles.Concrete.Properties;
using Monopoly.Client.Runtime.Game.Gameplay.Boards;
using Monopoly.Client.Runtime.Game.Gameplay.Groups;
using Monopoly.Client.Runtime.Game.Controllers.Strategies.Shared;
using Monopoly.Client.Runtime.Game.Serializableables;
using Monopoly.Client.Scriptable.Objects.Cards.Common;
using Monopoly.Client.Scriptable.Objects.Effects;

namespace Monopoly.Client.Runtime.Game.Controllers.Pawns.Common
{
    internal abstract class PawnController : NetworkBehaviour, ISubject<PawnController>
    {
        [SerializeField]
        private Image imageToken;
    }
}

```



```

[SerializeField, Range(0.0f, byte.MaxValue)]
private float movementSpeed = 35.0f;

[SerializeField, Range(0.0f, byte.MaxValue)]
private protected float turnDelay = 0.25f;

private int turnsInJailCount;
private float positionPrecisionThreshold;
private List<IObserver<PawnController>> observers;

internal Tile CurrentTile { get; private set; }
internal EffectFlags Effects { get; private set; }
internal NetworkVariable<int> Balance { get; private set; }
internal List<PropertyTile> OwnedTiles { get; private set; }
internal int NetWorth => this.Balance.Value + this.OwnedTiles.Sum(tile => tile.Worth);

internal Sprite PawnToken
{
    get => this.imageToken.sprite;
    private protected set => this.imageToken.sprite = value;
}

internal Color PawnColor { get; private protected set; }
internal string Nickname { get; private protected set; }

internal ISimpleFactory<Tile, ITileStrategy> FactoryTileStrategy { get; private protected set; }
    internal ISimpleFactory<CardScriptableObject, ICardStrategy> FactoryCardStrategy { get; private protected
set; }

internal abstract bool IsBot { get; }
internal abstract bool IsPlayer { get; }

internal abstract void PerformTurn();
internal abstract void RespondToTrade(TradeCredentialsSerializable credentials);
internal abstract void HandleTradeResponse(TradeCredentialsSerializable credentials);

private void Awake()
{
    this.Balance = new NetworkVariable<int>(
        GameManager.Instance.InitialBalance,
        NetworkVariableReadPermission.Everyone,

```

```

        NetworkVariableWritePermission.Server
    );

    if ((bool)base.NetworkManager?.IsHost)
        this.Balance.Value = (int)GameManager.Instance?.InitialBalance;

    this.OwnedTiles = new List<PropertyTile>();
    this.observers = new List<IObserver<PawnController>>();
}

private void Start()
{
    this.RegisterObserver(GameManager.Instance);
    this.NotifyObservers();

    this.CurrentTile = Board.Instance.TileStart;
    this.transform.position = Board.Instance.TileStart.transform.position;
    this.positionPrecisionThreshold = base.GetComponent<NetworkTransform>().PositionThreshold;
}

public void NotifyObservers()
{
    foreach (IObserver<PawnController> observer in this.observers)
        observer.HandleSubjectUpdate(this);
}

public void RegisterObserver(IObserver<PawnController> observer)
{
    if (observer == null)
        throw new System.ArgumentNullException($" {nameof(observer)} cannot be null.");

    if (this.observers.Contains(observer))
        throw new System.ArgumentException($" {nameof(observer)} is already registered.");

    this.observers.Add(observer);
}

public void UnregisterObserver(IObserver<PawnController> observer)
{
    if (observer == null)
        throw new System.ArgumentNullException($" {nameof(observer)} cannot be null.");
}

```

```

        if (!this.observers.Contains(observer))
            throw new System.ArgumentException($"{nameof(observer)} has not been registered.");

        this.observers.Remove(observer);
    }

    internal void TransactDumpBalance(int amount)
    {
        this.TransactDumpBalanceRemotelyRpc(amount);
    }

    [Rpc(SendTo.Server, RequireOwnership = false)]
    private void TransactDumpBalanceRemotelyRpc(int amount)
    {
        this.Balance.Value -= amount;
    }

    internal void TransactTakeBalance(int amount)
    {
        this.TransactTakeBalanceRemotelyRpc(amount);
    }

    [Rpc(SendTo.Server, RequireOwnership = false)]
    private void TransactTakeBalanceRemotelyRpc(int amount)
    {
        this.Balance.Value += amount;
    }

    internal void TransactSendBalance(PawnController receiver, int amount)
    {
        this.TransactSendBalanceRemotelyRpc(base.OwnerClientId, receiver.OwnerClientId, amount);
    }

    [Rpc(SendTo.Server, RequireOwnership = false)]
    private void TransactSendBalanceRemotelyRpc(ulong senderClientId, ulong receiverClientId, int amount)
    {
        PawnController sender = GameManager.Instance.GetPawn(senderClientId);
        PawnController receiver = GameManager.Instance.GetPawn(receiverClientId);

        if (sender != null)
            sender.Balance.Value -= amount;
    }

```

```

        if (receiver != null)
            receiver.Balance.Value += amount;
    }

    private protected void RollDice()
    {
        GameManager.Instance.RollDice();
        this.Move(GameManager.Instance.TotalRollResult);
    }

    internal void CompleteTurn()
    {
        if ((this.Effects & EffectFlags.Jail) == 0)
            GameManager.Instance.SwitchPawnSoftly();
        else
            GameManager.Instance.SwitchPawnForcefully();
    }

    internal void Surrender()
    {
        IList<PropertyTile> ownedTiles = this.OwnedTiles.ToList();

        foreach (PropertyTile ownedTile in ownedTiles)
            ownedTile.ResetOwnership();

        if (GameManager.Instance.CurrentPawn == this)
            GameManager.Instance.SwitchPawnForcefully();

        GameManager.Instance.DespawnOwnedObjectsRemotelyRpc(base.OwnerClientId);
    }

    internal void Apply(EffectFlags effects)
    {
        this.Effects |= effects;
    }

    internal void Move(int steps)
    {
        base.StartCoroutine(this.MoveTokenCoroutine(steps));
    }

    private IEnumerator MoveTokenCoroutine(int steps)

```

```

{
    int currentTileIndex = Board.Instance.GetIndexOfTile(this.CurrentTile);

    while (steps != 0)
    {
        if (steps > 0)
        {
            --steps;
            currentTileIndex = ++currentTileIndex % Board.Instance.TilesCount;

        }
        else
        {
            ++steps;
            currentTileIndex = Mathf.Abs(--currentTileIndex + Board.Instance.TilesCount) %
Board.Instance.TilesCount;
        }

        this.CurrentTile = Board.Instance.GetTileByIndex(currentTileIndex);

        if (steps != 0)
            this.CurrentTile.HandleFlyover(this);

        Vector3 targetPosition = Board.Instance.GetTileByIndex(currentTileIndex).transform.position;

        yield return base.StartCoroutine(this.MoveTokenForwardCoroutine(targetPosition));
    }

    this.CurrentTile = Board.Instance.GetTileByIndex(currentTileIndex);
    this.CurrentTile.HandleLanding(this);
}

private IEnumerator MoveTokenForwardCoroutine(Vector3 targetPosition)
{
    while (Vector3.Distance(this.transform.position, targetPosition) > this.positionPrecisionThreshold)
    {
        float maxDistanceDelta = this.movementSpeed * Time.deltaTime;
        this.transform.position = Vector3.MoveTowards(this.transform.position, targetPosition,
maxDistanceDelta);
        yield return null;
    }
}

```

```

        this.transform.position = targetPosition;
    }

    internal bool HasFullMonopoly(Group group)
    {
        if (group == null)
            throw new System.NullReferenceException($"{nameof(group)} cannot be null.");

        return group.GetTilesCountOwnedBy(this) == group.TilesCount;
    }

    internal bool HasPartialMonopoly(Group group)
    {
        if (group == null)
            throw new System.NullReferenceException($"{nameof(group)} cannot be null.");

        const float MONOPOLY_OWNERSHIP_THRESHOLD = 0.5f;
        return group.GetTilesCountOwnedBy(this) >= group.TilesCount *
MONOPOLY_OWNERSHIP_THRESHOLD;
    }

    internal void SendTrade(TradeCredentialsSerializable credentials)
    {
        this.SendTradeRemotelyRpc(credentials);
    }

    [Rpc(SendTo.Server, RequireOwnership = false)]
    private void SendTradeRemotelyRpc(TradeCredentialsSerializable credentials)
    {
        if (!credentials.AreValid)
        {
            credentials.Result = TradeResult.Declined;
            this.HandleTradeResponseLocallyRpc(credentials);
            return;
        }

        PawnController sender = GameManager.Instance.GetPawn(credentials.SenderClientId);

        if (sender == null)
            return;

        PawnController receiver = GameManager.Instance.GetPawn(credentials.ReceiverClientId);

```

```

    if (receiver == null)
    {
        credentials.Result = TradeResult.Declined;
        this.HandleTradeResponseLocallyRpc(credentials);
        return;
    }

    this.ReceiveTradeLocallyRpc(credentials);
}

[Rpc(SendTo.Everyone)]
private void ReceiveTradeLocallyRpc(TradeCredentialsSerializable credentials)
{
    PawnController receiver = GameManager.Instance.GetPawn(credentials.ReceiverClientId);

    if (receiver?.OwnerClientId == NetworkManager.Singleton.LocalClientId)
        receiver.RespondToTrade(credentials);

    if (NetworkManager.Singleton.IsHost)
    {
        if (receiver == null)
        {
            credentials.Result = TradeResult.Declined;
            this.HandleTradeResponseLocallyRpc(credentials);
        }
    }
}

private protected void AcceptTrade(TradeCredentialsSerializable credentials)
{
    this.AcceptTradeRemotelyRpc(credentials);
}

[Rpc(SendTo.Server, RequireOwnership = true)]
private void AcceptTradeRemotelyRpc(TradeCredentialsSerializable credentials)
{
    credentials.Result = TradeResult.Accepted;

    PawnController sender = GameManager.Instance.GetPawn(credentials.SenderClientId);

    if (sender == null)

```

```

        return;

        PawnController receiver = GameManager.Instance.GetPawn(credentials.ReceiverClientId);

        if (receiver == null)
        {
            credentials.Result = TradeResult.Declined;
            this.HandleTradeResponseLocallyRpc(credentials);
            return;
        }

        sender.TransactSendBalance(receiver, credentials.SenderBalanceAmount);
        receiver.TransactSendBalance(sender, credentials.ReceiverBalanceAmount);

        if (credentials.SenderTileIndex != TradeCredentialsSerializable.PLACEHOLDER_TILE)
        {
            PropertyTile senderTile = Board.Instance.GetTileByIndex(credentials.SenderTileIndex) as PropertyTile;

            senderTile.ResetOwnership();
            senderTile.UpdateOwnership(receiver);
        }

        if (credentials.ReceiverTileIndex != TradeCredentialsSerializable.PLACEHOLDER_TILE)
        {
            PropertyTile receiverTile = Board.Instance.GetTileByIndex(credentials.ReceiverTileIndex) as
PropertyTile;

            receiverTile.ResetOwnership();
            receiverTile.UpdateOwnership(sender);
        }

        this.HandleTradeResponseLocallyRpc(credentials);
    }

    private protected void DeclineTrade(TradeCredentialsSerializable credentials)
    {
        this.DeclineTradeRemotelyRpc(credentials);
    }

    [Rpc(SendTo.Server, RequireOwnership = true)]
    private protected void DeclineTradeRemotelyRpc(TradeCredentialsSerializable credentials)
    {

```



```

        credentials.Result = TradeResult.Declined;
        this.HandleTradeResponseLocallyRpc(credentials);
    }

    [Rpc(SendTo.Everyone)]
    private void HandleTradeResponseLocallyRpc(TradeCredentialsSerializable credentials)
    {
        PawnController sender = GameManager.Instance.GetPawn(credentials.SenderClientId);

        if (sender?.OwnerClientId == NetworkManager.Singleton.LocalClientId)
            sender.HandleTradeResponse(credentials);
    }
}

```

Файл Board.cs

Клас для реалізації поля ігрової дошки

```

using System;
using System.Linq;
using System.Collections.Generic;
using UnityEngine;
using Monopoly.Client.Runtime.Game.Gameplay.Tiles.Common;
using Monopoly.Client.Runtime.Game.Gameplay.Tiles.Concrete.Properties;
using Monopoly.Client.Runtime.Game.Gameplay.Groups;
using Monopoly.Client.Scriptable.Objects.Cards.Common;

namespace Monopoly.Client.Runtime.Game.Gameplay.Boards
{
    internal sealed class Board : MonoBehaviour
    {
        [SerializeField]
        private Group[] monopolies;

        [SerializeField]
        private CardScriptableObject[] cardsTax;

        [SerializeField]
        private CardScriptableObject[] cardsChance;

        [field: SerializeField]
        internal Tile TileJail { get; private set; }
    }
}

```

```

[field: SerializeField]
internal Tile TileStart { get; private set; }

[field: SerializeField]
internal Tile TileParking { get; private set; }

[field: SerializeField]
internal Tile TileJailTrigger { get; private set; }

private List<Tile> tiles;

internal static Board Instance { get; private set; }

internal int TilesCount => this.tiles.Count;

private void Awake()
{
    if (Board.Instance != null)
    {
        throw new TypeInitializationException(
            nameof(Board),
            new ApplicationException($"Singleton has already been initialized.")
        );
    }

    Board.Instance = this;
}

private void Start()
{
    this.tiles = new List<Tile>();

    foreach (Transform child in this.transform)
    {
        if (child.TryGetComponent(out Tile tile))
            this.tiles.Add(tile);
    }
}

internal int GetIndexOfTile(Tile tile)
{

```

```

        if (tile == null)
            throw new NullReferenceException($"{nameof(tile)} is null.");

        return this.tiles.IndexOf(tile);
    }

    internal Tile GetTileByIndex(int index)
    {
        if (index < 0 || index >= this.tiles.Count)
            throw new IndexOutOfRangeException($"{nameof(index)} is out of range.");

        return this.tiles[index];
    }

    internal CardScriptableObject GetCardTax()
    {
        return this.cardsTax[UnityEngine.Random.Range(0, this.cardsTax.Length)];
    }

    internal CardScriptableObject GetCardChance()
    {
        return this.cardsChance[UnityEngine.Random.Range(0, this.cardsChance.Length)];
    }

    internal Group GetMonopolySet(PropertyTile tile)
    {
        if (tile == null)
            throw new ArgumentNullException($"{nameof(tile)} is null.");

        return this.monopolies.FirstOrDefault(monopoly => monopoly.Contains(tile));
    }

    internal int GetDistance(int tileIndexFrom, int tileIndexTo)
    {
        int clockwiseDistance = (tileIndexTo - tileIndexFrom + this.TilesCount) % this.TilesCount;
        int counterclockwiseDistance = (tileIndexFrom - tileIndexTo + this.TilesCount) % this.TilesCount;
        return Mathf.Min(clockwiseDistance, counterclockwiseDistance) == counterclockwiseDistance ?
        -counterclockwiseDistance : clockwiseDistance;
    }

    internal int GetDistance(Tile tileFrom, Tile tileTo)
    {

```

```

        int clockwiseDistance = (this.GetIndexOfTile(tileTo) - this.GetIndexOfTile(tileFrom) + this.TilesCount) %
this.TilesCount;
        int counterclockwiseDistance = (this.GetIndexOfTile(tileFrom) - this.GetIndexOfTile(tileTo) +
this.TilesCount) % this.TilesCount;
        return Mathf.Min(clockwiseDistance, counterclockwiseDistance) == counterclockwiseDistance ?
-counterclockwiseDistance : clockwiseDistance;
    }
}
}

```

Файл GameManager.cs

Клас ігрового менеджера, який синхронізує процес гри

```

using System.Linq;
using System.Collections;
using System.Collections.Generic;
using Unity.Netcode;
using UnityEngine;
using UnityEngine.SceneManagement;
using Monopoly.Unity.Services.Lobbies.Models.Extensions;
using Monopoly.Client.Shared.Helpers;
using Monopoly.Client.Shared.Patterns.Observables;
using Monopoly.Client.Runtime.UI.Views.Concrete.Game;
using Monopoly.Client.Runtime.Core.P2P;
using Monopoly.Client.Runtime.Game.Gameplay.Boards;
using Monopoly.Client.Runtime.Game.Controllers.Pawns.Common;
using Monopoly.Client.Scriptable.Objects.Tokens;
using Monopoly.Client.Scriptable.Objects.Effects;

namespace Monopoly.Client.Runtime.Game.Managers
{
    internal sealed class GameManager : NetworkBehaviour, IObservable<PawnTableView>,
IObservable<PawnController>
    {
        [SerializeField]
        private NetworkObject bot;

        [SerializeField]
        private NetworkObject player;

        [SerializeField]
        private NetworkObject pawnTable;
    }
}

```

```

[field: SerializeField, Range(0, byte.MaxValue)]
internal int MaxJailTurns { get; private set; } = 3;

[field: SerializeField, Range(0, byte.MaxValue)]
internal int MaxDoublesSequence { get; private set; } = 2;

[field: SerializeField, Range(0, short.MaxValue)]
internal int InitialBalance { get; private set; } = 15_000;

[field: SerializeField, Range(0, short.MaxValue)]
internal int CircleSuperBonus { get; private set; } = 3_000;

[field: SerializeField, Range(0, short.MaxValue)]
internal int CircleDefaultBonus { get; private set; } = 2_000;

[field: SerializeField]
internal TokenScriptableObject[] Tokens { get; private set; } = new
TokenScriptableObject[LobbyManager.MAX_PLAYERS];

private const ulong CLIENT_ID_HOST = 0;

internal static GameManager Instance { get; private set; }

private int firstDieValue;
private int secondDieValue;
private int currentPawnIndex;
private int rolledDoublesCount;
private List<PawnTableView> tables;
private List<PawnController> pawns;

private int nextPawnIndex
{
    get => ++this.currentPawnIndex % this.pawns.Count;
}

internal event System.Action<int, int> DiceRolledEvent;
internal event System.Action<PawnController> GameFinishedEvent;
internal PawnController CurrentPawn => this.pawns[this.currentPawnIndex];
internal int TotalRollResult => this.firstDieValue + this.secondDieValue;
internal bool HasRolledDouble => this.firstDieValue == this.secondDieValue;

```

```

private void Awake()
{
    if (GameManager.Instance != null)
    {
        throw new System.TypeInitializationException(
            nameof(GameManager),
            new System.ApplicationException($"Singleton has already been initialized.")
        );
    }

    GameManager.Instance = this;

    this.pawns = new List<PawnController>();
    this.tables = new List<PawnTableView>();
}

private void OnEnable()
{
    if (base.NetworkManager != null)
        base.NetworkManager.OnClientDisconnectCallback += this.OnClientDisconnected;
}

private void Start()
{
    if (base.NetworkManager.IsHost)
        base.StartCoroutine(this.WaitAllClientsCoroutine());
}

private void OnDisable()
{
    if (base.NetworkManager != null)
        base.NetworkManager.OnClientDisconnectCallback -= this.OnClientDisconnected;
}

public override sealed void OnDestroy()
{
    base.OnDestroy();
    base.StopAllCoroutines();
}

public void HandleSubjectUpdate(PawnController pawn)
{

```

```

    if (pawn == null)
        throw new System.ArgumentNullException($"{nameof(pawn)} cannot be null.");

    this.pawns.Add(pawn);
}

public void HandleSubjectUpdate(PawnTableView table)
{
    if (table == null)
        throw new System.ArgumentNullException($"{nameof(table)} cannot be null.");

    this.tables.Add(table);
}

private async void OnClientDisconnected(ulong disconnectedClientId)
{
    if (base.NetworkManager.IsHost)
    {
        PawnController pawn = this.pawns.Where(
            pawn => pawn.OwnerClientId == disconnectedClientId
        ).FirstOrDefault();
        pawn?.Surrender();
    }

    if (disconnectedClientId != GameManager.CLIENT_ID_HOST)
        return;

    if (await LobbyManager.Instance?.DoesLocalLobbyExistAsync())
        await LobbyManager.Instance?.QuitLobbyAsync();
}

private IEnumerator WaitAllClientsCoroutine()
{
    float elapsedTime = 0.0f;

    bool havePlayersLoaded;
    string currentSceneName = SceneManager.GetActiveScene().name;

    while (true)
    {
        elapsedTime += Time.deltaTime;
    }
}

```

```

        havePlayersLoaded = LobbyManager.Instance.LocalLobby.ArePlayersSynchronized(
            SessionManager.PLAYER_DATA_KEY_SCENE,
            currentSceneName
        );

        if (havePlayersLoaded)
            break;

        yield return null;
    }

    this.InitializeGame();
}

private void InitializeGame()
{
    this.currentPawnIndex = 0;

    for (int i = 0; i < base.NetworkManager?.ConnectedClients.Count; ++i)
    {
        ulong clientId = base.NetworkManager.ConnectedClientsIds[i];

        base.NetworkManager?.SpawnManager?.InstantiateAndSpawn(this.player, clientId, true, true);
        base.NetworkManager?.SpawnManager?.InstantiateAndSpawn(this.pawnTable, clientId, true);
    }

    string currentGameMode =
        LobbyManager.Instance.LocalLobby.GetData(LobbyManager.LOBBY_DATA_KEY_GAME_MODE);

    if (currentGameMode.Equals(LobbyManager.LOBBY_DATA_VALUE_GAME_MODE_RANKED,
        System.StringComparison.Ordinal))
    {
        for (int i = LobbyManager.Instance.LocalLobby.Players.Count; i < LobbyManager.MAX_PLAYERS; ++i)
        {
            base.NetworkManager?.SpawnManager?.InstantiateAndSpawn(this.bot, (ulong)i, true, true);
            base.NetworkManager?.SpawnManager?.InstantiateAndSpawn(this.pawnTable, (ulong)i, true);
        }
    }

    base.StartCoroutine(LateStartHelper.InitializeLatelyCoroutine(
        () => this.CurrentPawn.PerformTurn()
    ));
}

```



```

}

[Rpc(SendTo.Server, RequireOwnership = false)]
internal void DespawnOwnedObjectsRemotelyRpc(ulong clientId)
{
    PawnTableView pawnTable = this.tables.Where(
        table => table.OwnerClientId == clientId
    ).First();

    PawnController pawnController = this.pawns.Where(
        pawn => pawn.OwnerClientId == clientId
    ).First();

    pawnTable.GetComponent<NetworkObject>().Despawn(true);
    pawnController.GetComponent<NetworkObject>().Despawn(true);

    this.RemoveDespawnedObjectsLocallyRpc(clientId);
}

[Rpc(SendTo.Everyone)]
private void RemoveDespawnedObjectsLocallyRpc(ulong clientId)
{
    PawnTableView pawnTable = this.tables.Where(
        table => table.OwnerClientId == clientId
    ).First();

    PawnController pawnController = this.pawns.Where(
        pawn => pawn.OwnerClientId == clientId
    ).First();

    this.tables.Remove(pawnTable);
    this.pawns.Remove(pawnController);

    if (this.pawns.Count == 1)
        this.GameFinishedEvent?.Invoke(this.pawns.First());
}

internal PawnController GetPawn(ulong clientId)
{
    return this.pawns.FirstOrDefault(data => data.OwnerClientId == clientId);
}

```

```

internal void SwitchPawnSoftly()
{
    this.SwitchPawnSoftlyRemotelyRpc();
}

internal void SwitchPawnForcefully()
{
    this.SwitchPlayerForcefullyRemotelyRpc();
}

[Rpc(SendTo.Server, RequireOwnership = false)]
internal void SwitchPawnSoftlyRemotelyRpc()
{
    this.rolledDoublesCount = 0;
    this.SwitchPawnLocallyRpc(this.nextPawnIndex);
}

[Rpc(SendTo.Server, RequireOwnership = false)]
internal void SwitchPlayerForcefullyRemotelyRpc()
{
    this.rolledDoublesCount = 0;
    this.SwitchPawnLocallyRpc(this.nextPawnIndex);
}

[Rpc(SendTo.Everyone)]
private void SwitchPawnLocallyRpc(int pawnIndex)
{
    this.currentPawnIndex = pawnIndex;

    if (base.NetworkManager.IsHost && this.CurrentPawn.IsBot)
        this.CurrentPawn.PerformTurn();

    if (this.CurrentPawn.OwnerClientId == base.NetworkManager.LocalClientId)
        this.CurrentPawn.PerformTurn();
}

internal void RollDice()
{
    const int MIN_DIE_VALUE = 1;
    const int MAX_DIE_VALUE = 6;

    this.firstDieValue = Random.Range(MIN_DIE_VALUE, MAX_DIE_VALUE + 1);
}

```

```

        this.secondDieValue = Random.Range(MIN_DIE_VALUE, MAX_DIE_VALUE + 1);

        this.RollDiceRemotelyRpc(this.firstDieValue, this.secondDieValue);
    }

    [Rpc(SendTo.Server, RequireOwnership = false)]
    private void RollDiceRemotelyRpc(int firstDieValue, int secondDieValue)
    {
        this.RollDiceLocallyRpc(firstDieValue, secondDieValue);
    }

    [Rpc(SendTo.Everyone)]
    private void RollDiceLocallyRpc(int firstDieValue, int secondDieValue)
    {
        this.firstDieValue = firstDieValue;
        this.secondDieValue = secondDieValue;

        this.DiceRolledEvent?.Invoke(this.firstDieValue, this.secondDieValue);
    }
}

```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**Кросплатформна багатокористувацька сесійна реалізація настільної гри
«Монополія» на Unity
Програма та методика тестування
КПІ.ІП-1303.045490.04.51**

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Павло РОДІОНОВ

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Денис БАБІЧ

Київ – 2025

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАНЬ.....	3
2 МЕТА ТЕСТУВАННЯ.....	4
3 МЕТОДИ ТЕСТУВАННЯ.....	5
4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ.....	6

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є багатокористувацька кросплатформена для смартфонів гра «Монополія». Тестується коректність ігрових механік, які відповідають класичним правилам «Монополії», функціонал багатокористувацької гри в реальному часі та адаптивність графічного інтерфейсу до різних розмірів і роздільних здатностей екранів смартфонів.

Гра призначена для роботи на операційних системах Android 9.0 і вище та iOS 14.0 і вище, з рекомендованими вимогами не менше 3 ГБ оперативної пам’яті та процесор Qualcomm Snapdragon 625 для Android, а також не менше 2 ГБ оперативної пам’яті та процесор Apple A10 Fusion для iOS.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка коректності реалізації ігрових механік мобільної гри «Монополія» відповідно до класичних правил;
- перевірка надійності збереження ігрових даних, включаючи стани ігрових сесій і профілі гравців;
- перевірка сумісності застосунку з операційними системами Android 9.0 (Pie) і вище та iOS 14.0 і вище на смартфонах;
- оцінка адаптивності графічного інтерфейсу для різних розмірів і роздільних здатностей екранів смартфонів;
- перевірка стабільності роботи однорангової централізованої мережі для забезпечення безперебійної взаємодії гравців у реальному часі;
- виявлення та усунення помилок і недоліків у роботі програми.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – аналіз вихідного коду та документації на відповідність стандартам програмування, виявлення потенційних вразливостей і помилок у структурі коду;
- мануальне тестування – виконання тестових сценаріїв вручну, перевірка ігрових механік, інтерфейсу користувача та взаємодії гравців у багатокористувацькому режимі через створення тест-кейсів;
- тестування «чорної скриньки» – перевірка функціональності гри без урахування внутрішньої структури, оцінка коректності вихідних результатів за заданими вхідними даними;
- тестування «білої скриньки» – аналіз внутрішньої логіки програми, зокрема алгоритмів однорангової централізованої мережі, обробки ігрових подій і збереження даних;
- стрес-тестування – оцінка стабільності програми в умовах високого навантаження або нестабільного мережевого з'єднання для забезпечення надійності однорангової мережі.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- стороннє програмне забезпечення: емулятори Android Studio (для Android) та Xcode Simulator (для iOS) для тестування на різних конфігураціях пристроїв;
- фізичні пристрої: смартфони з операційними системами Android 13.0 (Google Pixel 9a) та iOS 14.0 (iPhone 13 mini) з метою тестування в реальних умовах.

Порядок проведення тестування буде наступним:

- динамічне тестування на відповідність функціональним вимогам, включаючи перевірку ігрових механік і багатокористувацького режиму в реальному часі;
- тестування на мобільних пристроях з різною роздільною здатністю екрана для оцінки адаптивності графічного інтерфейсу;
- тестування на мобільних пристроях з різними версіями операційних систем (Android 9.0 і вище, iOS 14.0 і вище) для забезпечення сумісності;
- тестування зручності використання для оцінки інтуїтивності інтерфейсу та комфорту взаємодії гравців;
- стрес-тестування однорангової централізованої мережі для перевірки стабільності при високому навантаженні, або нестабільному з'єднанні.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**Кросплатформна багатокористувацька сесійна реалізація настільної гри
«Монополія» на Unity
Керівництво користувача
КПІ.ІП-1303.045490.05.34**

“ПОГОДЖЕНО”

Керівник проекту:

_____ Павло РОДІОНОВ

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Денис БАБІЧ

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ.....	3
2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ.....	4
2.1 Системні вимоги для коректної роботи.....	4
2.2 Завантаження застосунку.....	4
2.3 Перевірка коректної роботи.....	4
3 ВИКОНАННЯ ПРОГРАМИ.....	5

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Dyplomopoly» – це кросплатформний мобільний застосунок, розроблений на базі ігрового рушія Unity, що реалізує цифрову версію класичної настільної гри «Монополія» з підтримкою багатокористувацького режиму. Розробка орієнтована на відтворення автентичного ігрового досвіду настільної гри «Монополія» з акцентом на доступність та мінімальну залежність від зовнішніх серверів, сервісів та ресурсів.

Функціонал застосунку:

- створення та приєднання до приватних та публічних лобі;
- підтримка багатокористувацької гри через однорангову централізовану мережеву топологію;
- реалізація ключових ігрових механік, включаючи кидання кубика, переміщення ігровим полем, придбання та покращення власності, сплату оренди, взаємодію з полями шансів та обмін між гравцями;
- забезпечення кросплатформної сумісності ігрових сесій;
- оновлення рейтингу гравців на основі їхніх ігрових результатів для справедливого підбору суперників;
- можливість перегляду профілю гравця з відображенням рейтингу.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для забезпечення стабільної та коректної роботи застосунку «Dyplomopoly» необхідно виконання наступних вимог:

- Операційна система iOS версії 12.0 або вище, Android версії 9.0 або вище;
- Стабільне підключення до мережі Інтернет для багатокористувацького режиму;
- Вільний обсяг пам'яті не менше 200 МБ для встановлення та роботи застосунку;
- Процесор двоядерний із частотою 1.5 ГГц або вище;
- Оперативна пам'ять не менше 2 ГБ;
- Графічний процесор із підтримкою OpenGL ES 3.0 або вище.

2.2 Завантаження застосунку

Застосунок «Dyplomopoly» призначений для встановлення на мобільні пристрої з операційними системами Android та iOS. Для Android користувачі можуть завантажити APK-файл із вебсайту розробника та встановити його вручну за допомогою системного інсталятора. Для iOS необхідно отримати інсталяційний пакет і виконати встановлення за допомогою системних інструментів.

2.3 Перевірка коректної роботи

Користувачу необхідно встановити програму на мобільний пристрій із операційною системою Android або iOS. Після запуску застосунку користувач отримує доступ до основного функціоналу гри, що включає можливість створення або приєднання до ігрових сесій, участь у багатокористувацькому режимі, виконуючи відповідні інтерактивні ігрові дії.

3 ВИКОНАННЯ ПРОГРАМИ

Початковий стан застосунку після процесу ініціалізації підключення до ігрових сервісів гри наведено на рисунку 3.1.



Рисунок 3.1 – Початковий стан гри після запуску

Користувач має можливість вибору типу лобі, до якого бажає приєднатись, відповідний стан інтерфейсу наведено на рисунку 3.2.

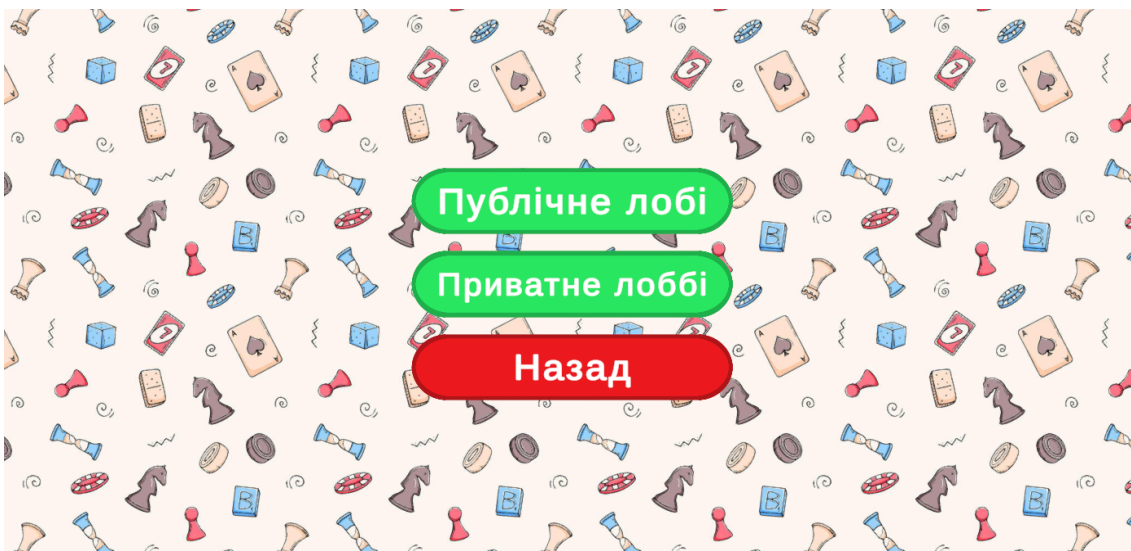


Рисунок 3.2 – Етап вибору типу лобі

Користувач може обрати свою бажану роль й бути хостом ігрової сесії, або звичайним клієнтом, відповідний стан інтерфейсу наведено на рисунку 3.3.



Рисунок 3.3 – Вибір варіанту підключення до приватного лобі

У випадку, якщо користувач обрав приватне лобі й роль хоста, то після процесу завантаження, гравець опиняється у приватній ігровій сесії, де наведено код підключення, яким можна поділитися з іншими гравцями. Відповідний стан лобі наведено на рисунку 3.4.



Рисунок 3.4 – Хост знаходиться у приватному лобі

Користувач-клієнт може під'єднатися до приватного лобі за наданим кодом лобі від хоста, відповідний стан інтерфейсу наведено на рисунку 3.5 та оновлене лобі зображено на рисунку 3.6.



Рисунок 3.5 – Введений код лобі у ролі звичайного клієнта



Рисунок 3.6 – Обидва гравці знаходяться у лобі

Стан гри після отримання права ходу гравцем наведено на рисунку 3.7.

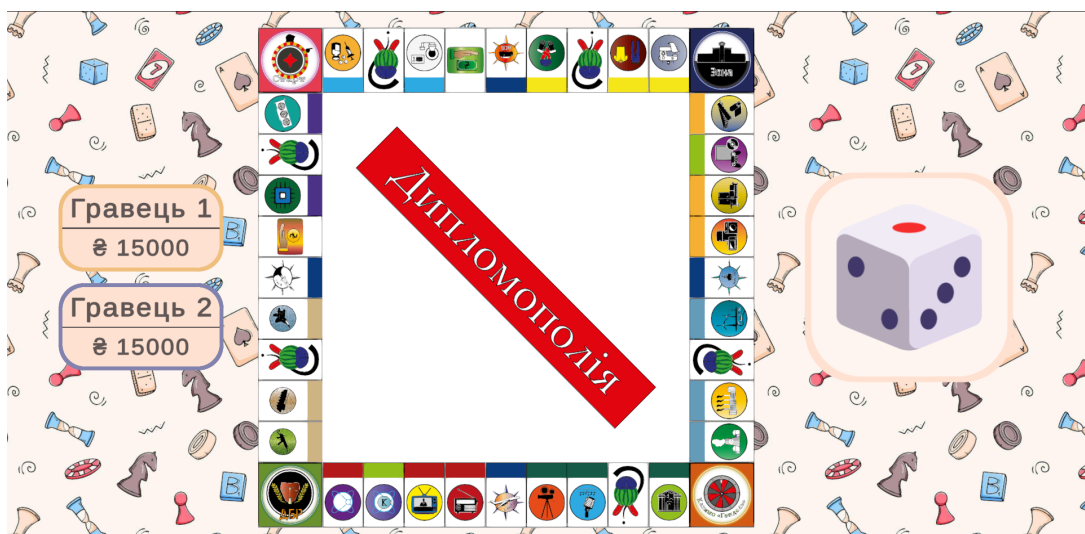


Рисунок 3.7 – Гра на етапі отримання права ходу локальним гравцем

Демонстрацію безпосереднього процесу гри наведено на рисунках 3.8 – 3.10, де наведено придбання поля власності (рисунок 3.8), проведення обміну (рисунки 3.8 – 3.10).



Рисунок 3.8 – Демонстрація можливості придбання поля



Рисунок 3.9 – Демонстрація можливості відправки обміну



Рисунок 3.10 – Демонстрація можливості отримання обміну

Користувач має право добровільної здачі, відповідне діалогове вікно наведено на рисунку 3.11.

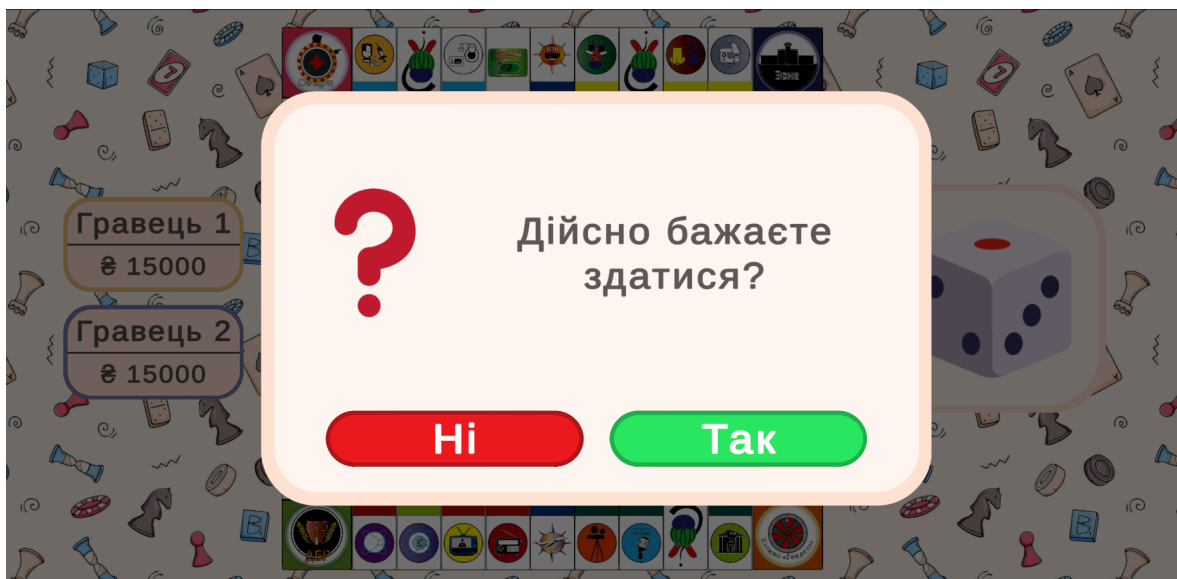


Рисунок 3.11 – Можливість добровільної здачі

Коли у матчі залишається 1 гравець, оскільки решта користувачів здалися, або залишили гру, то виводиться діалогове вікно з повідомленням про перемогу гравця (рисунок 3.12).

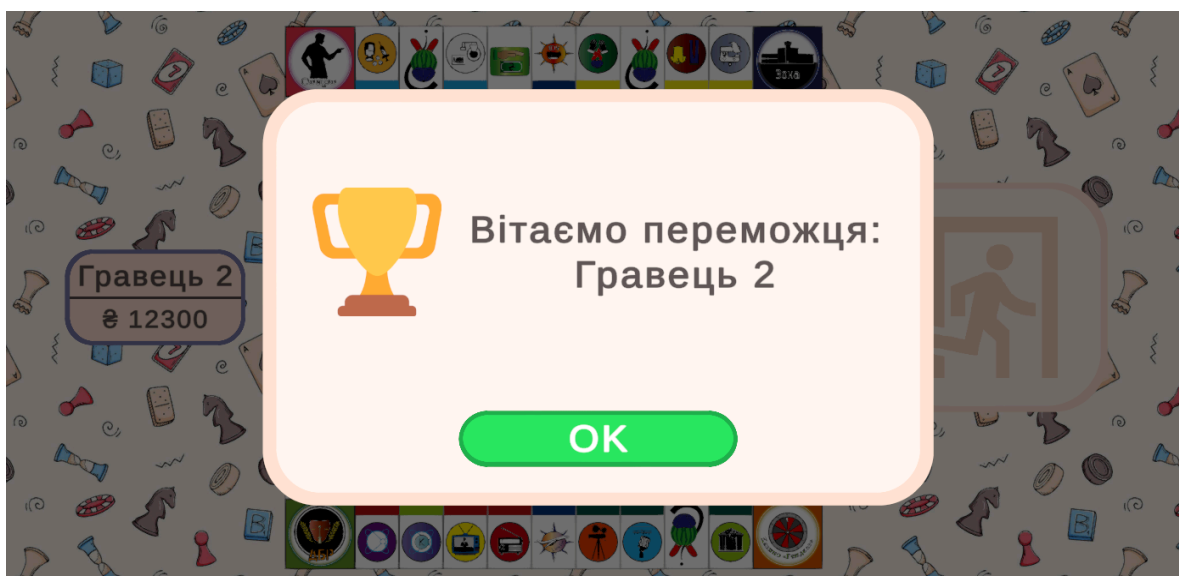


Рисунок 3.12 – Демонстрація вікна-оголошення переможця

У випадку, коли користувач-клієнт обирає публічне лобі, то йому надається можливість запустити пошук доступної публічної гри, демонстрація процесу пошуку наведено на рисунку 3.13.

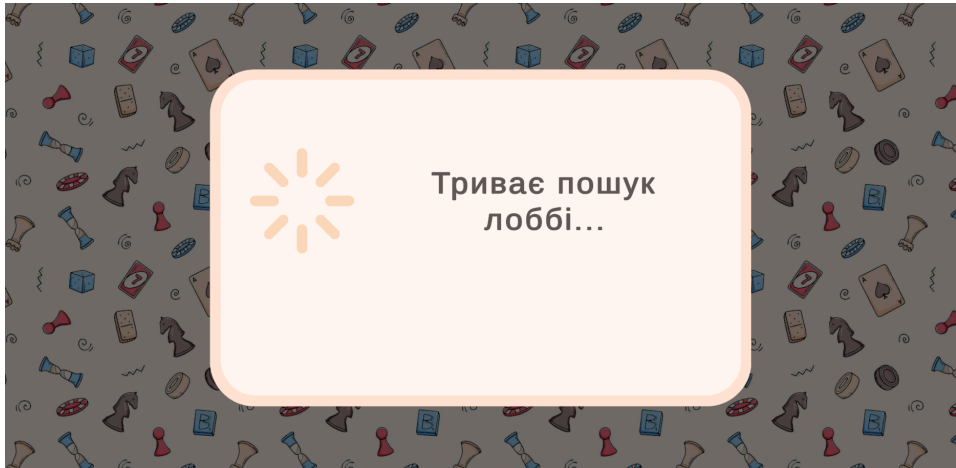


Рисунок 3.13 – Процес пошуку відкритого лобі у вигляді звичайного клієнту

Демонстрацію безпосереднього процесу гри наведено на рисунках 3.14 – 3.16, де наведено приклад сплати за оренду перебування на полі (рисунок 3.14), взаємодію з полем шанс (рисунок 3.15) й покращення власності (рисунок 3.16).

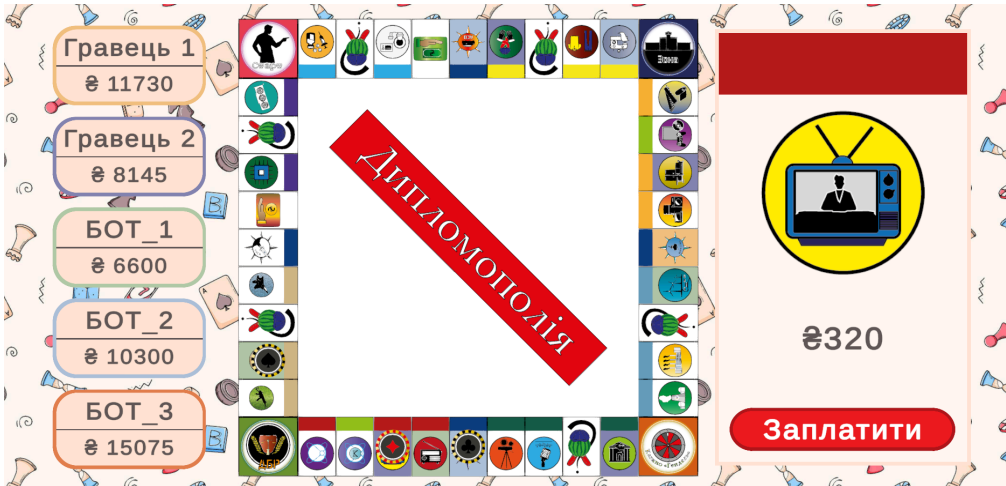


Рисунок 3.14 – Демонстрація меню сплати оренди



Рисунок 3.15 – Приклад взаємодії з полем шанс



Рисунок 3.16 – Меню покращення власності

Користувач має можливість перегляду власного ігрового профілю, де гравець має можливість зміни нікнейму та перегляду власної кількості рейтингу (рисунок 3.17).



Рисунок 3.17 – Перегляд профілю користувача у головному меню

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ” _____ 2025 р.

Кросплатформна багатокористувацька сесійна реалізація настільної гри

«Монополія» на Unity

Графічний матеріал

КПІ.ІП-1303.045490.06.99

“ПОГОДЖЕНО”

Керівник проекту:

_____ Павло РОДІОНОВ

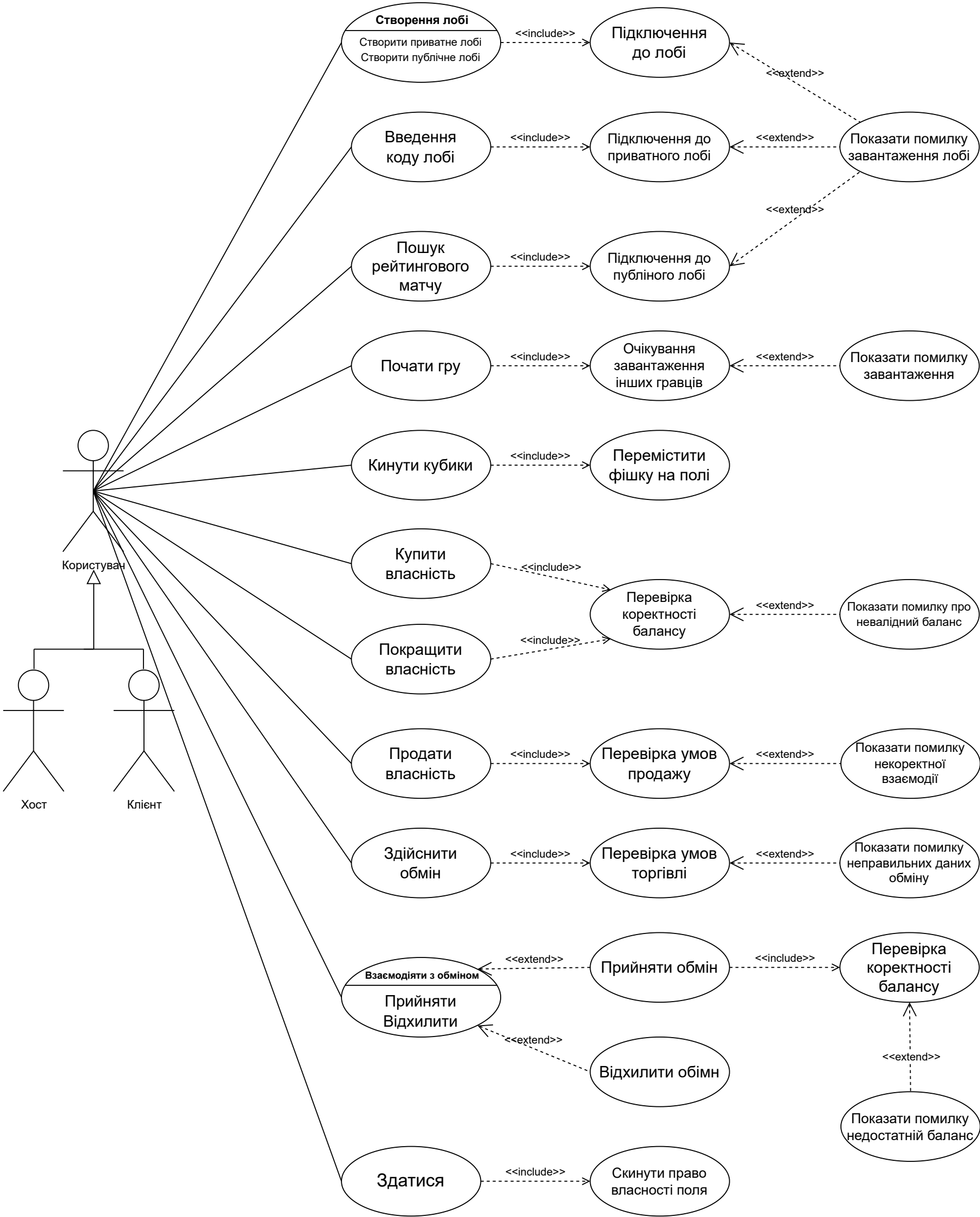
Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

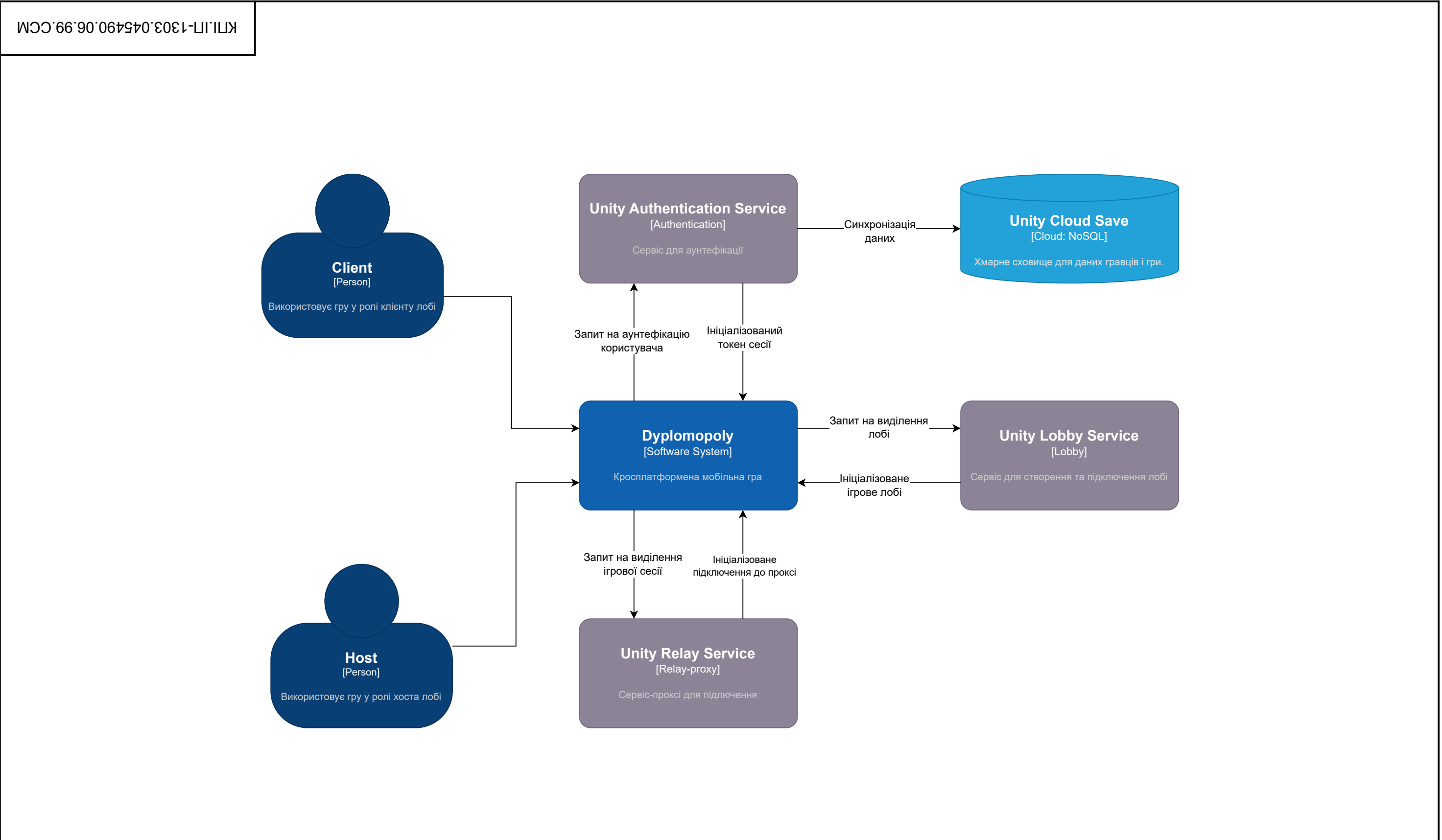
Виконавець:

_____ Денис БАБІЧ

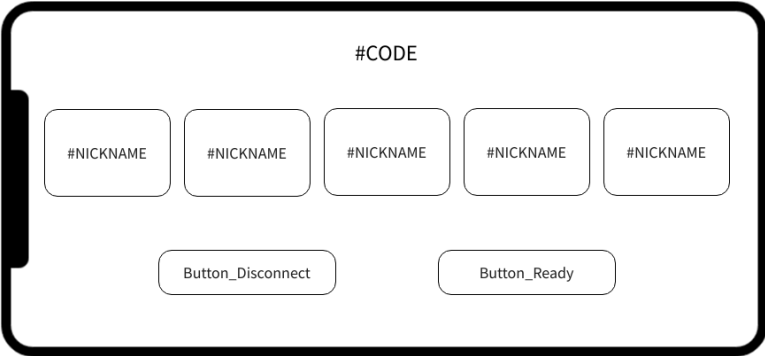
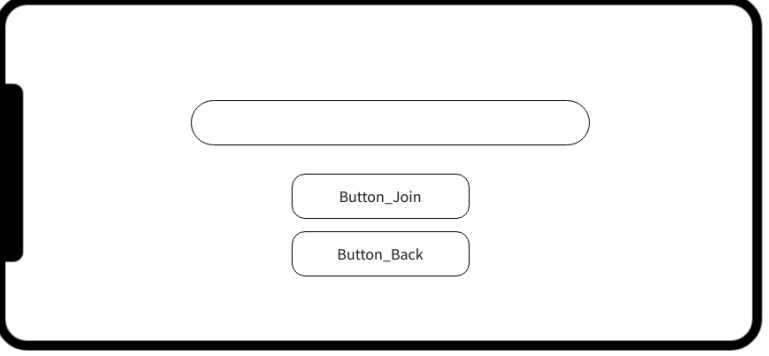
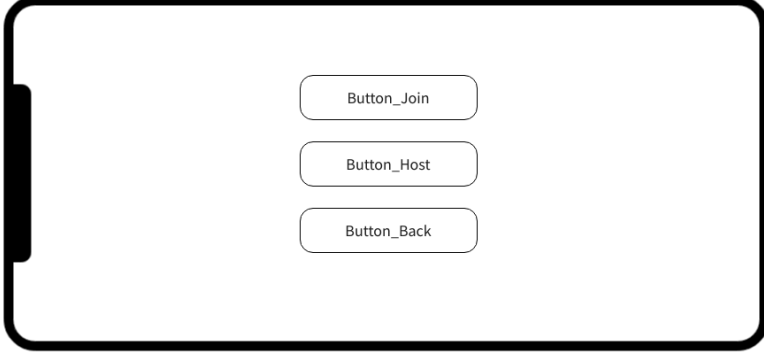
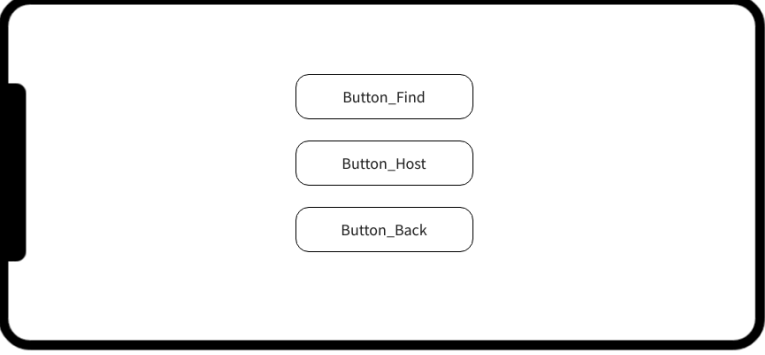
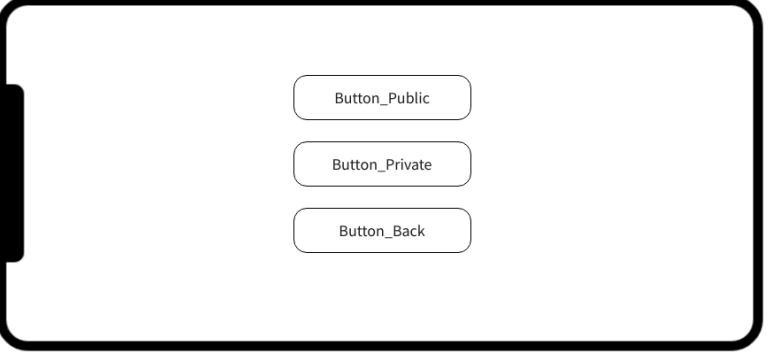
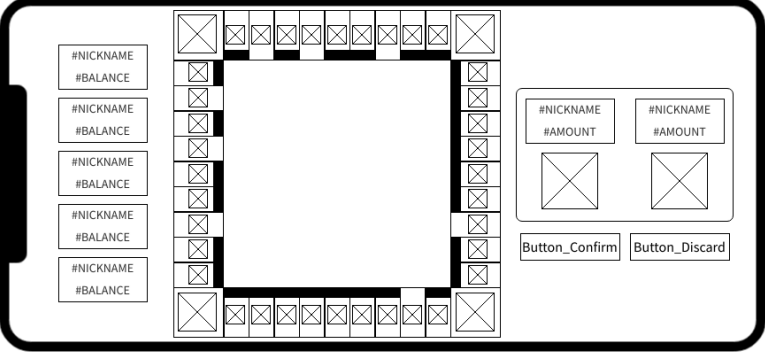
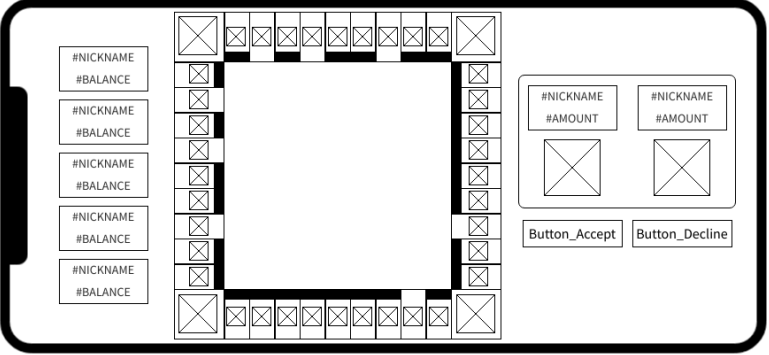
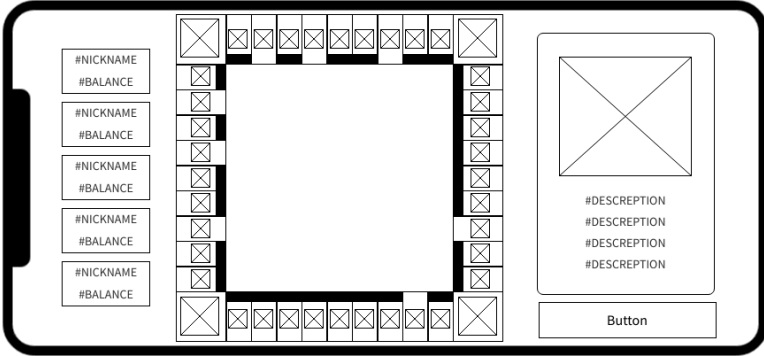
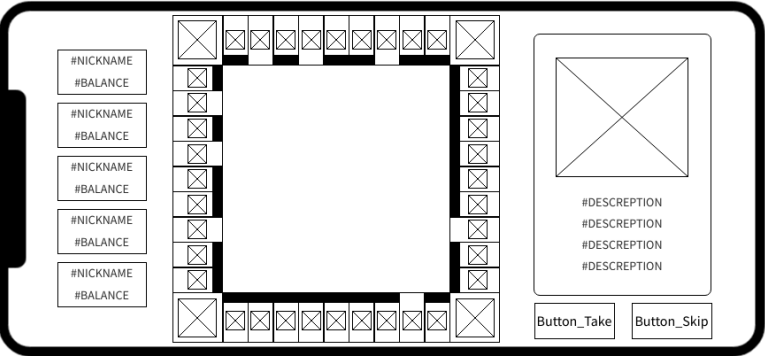
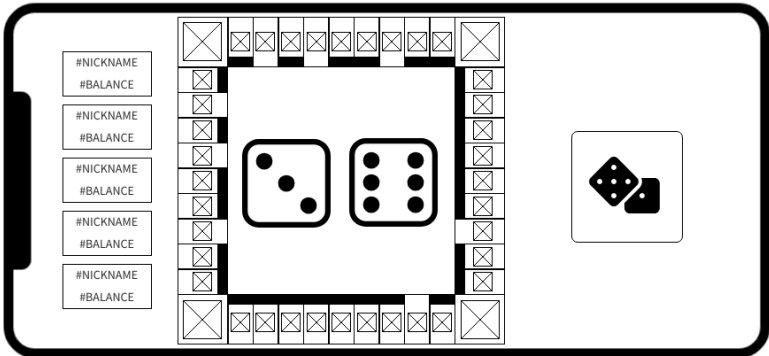
Київ – 2025



					КПІ.ІП-1303.045490.06.99.CCB											
					Схема структурна варіантів використань					Лит.		Маса	Масштаб			
										Аркуш 1		Аркушів 4				
										Кросплатформна багатокористувацька сесійна реалізація настільної гри «Монополія» на Unity						
										КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13						
Зм.	Арк.	№ докум.	Підп.	Дата												
Розробив		Бабіч Д. В.														
Перевішив		Родіонов П. Ю.														
Т. контр.																
Н. контр.		Вітковська І. І.														
Затвердив		Жаріков Е.В.														

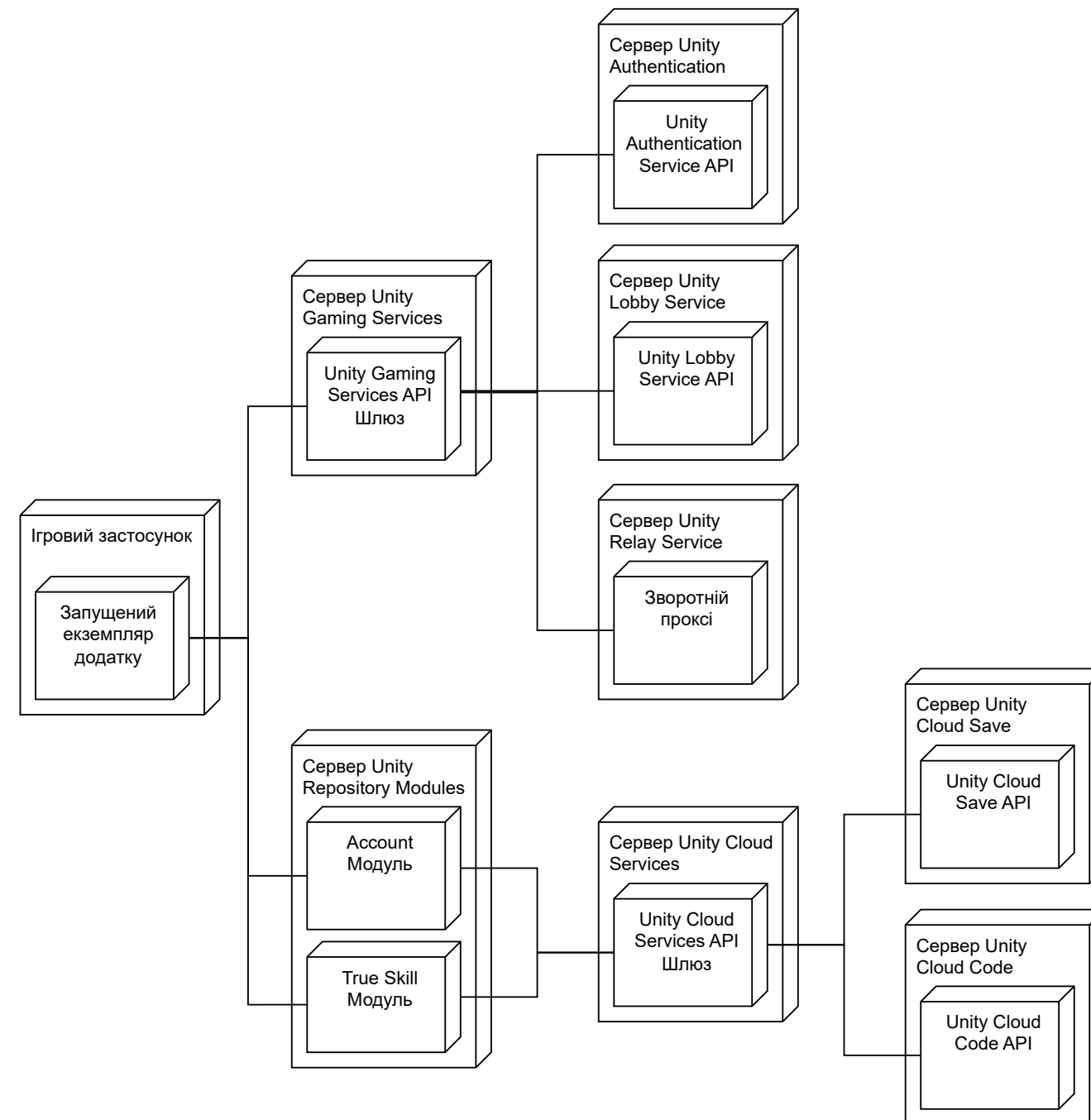


					КПІ.ІП-1303.045490.06.99.CCM							
					Схема структурна компонентів програмного забезпечення			Літера		Маса	Масштаб	
Зм.		Арк.	№ документа	Підпис				Дата				
Розробив		Бабіч Д. В.										
Перевірів		Родіонов П. Ю.										
Т. контр.												
								Аркуш 2		Аркушів 4		
Н. контр.		Вітковська І. І.			Кросплатформна багатокористувацька сесійна реалізація настільної гри «Монополія» на Unity			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13				
Затвердив		Жаріков Е.В.										



					КПІ.ІП-1303.045490.06.99.CCM						
					Креслення виду екранних форми	Літера			Маса	Масштаб	
Зм.	Арк.	№ документа	Підпис	Дата							
Розробив	Бабіч Д. В.										
Перевірів	Родіонов П. Ю.										
Т. контр.						Аркуш 3			Аркушів 4		
					Кросплатформна багатокористувацька сесійна реалізація настільної гри «Монополія» на Unity	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13					
Н. контр.		Вітковська І. І.									
Затвердив		Жаріков Е.В.									

Діаграма розгортання



Демонстраційний плакат до дипломного проекту

Кросплатформна багатокористувачська сесійна реалізація настільної гри «Монополія» на Unity

Виконав студент гр. ІІІ-13 Бабіч Д. В.

Керівник Родіонов П. Ю.