**Пояснювальна записка  
до курсової роботи**

на тему: Гра для мобільного пристрою «Монополія»

КПІ.ІП-1402.045490.02.81

Київ – 2024

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 4](#_30j0zll)

[ВСТУП 5](#_3znysh7)

[1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 6](#_tyjcwt)

[1.1 Загальні положення 6](#_1t3h5sf)

[1.2 Змістовний опис і аналіз предметної області 9](#_2s8eyo1)

[1.3 Аналіз існуючих технологій та успішних IT-проєктів 10](#_3rdcrjn)

[1.3.1 Аналіз відомих алгоритмічних та технічних рішень 10](#_lnxbz9)

[1.3.2 Аналіз допоміжних програмних засобів та засобів розробки 15](#_4f1mdlm)

[1.3.3 Аналіз відомих програмних продуктів 18](#_19c6y18)

[1.4 Аналіз вимог до програмного забезпечення 21](#_28h4qwu)

[1.4.1 Розроблення функціональних вимог 29](#_37m2jsg)

[1.4.2 Розроблення нефункціональних вимог 33](#_46r0co2)

[1.5 Постановка задачі 34](#_206ipza)

[Висновки до розділу 35](#_2zbgiuw)

[2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 36](#_3ygebqi)

[2.1 Моделювання та аналіз програмного забезпечення 36](#_sqyw64)

[2.2 Архітектура програмного забезпечення 39](#_1664s55)

[2.3 Конструювання програмного забезпечення 41](#_39kk8xu)

[2.4 Аналіз безпеки даних 46](#_48pi1tg)

[Висновки до розділу 47](#_319y80a)

[3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 48](#_40ew0vw)

[3.1 Аналіз якості ПЗ 48](#_upglbi)

[3.2 Опис процесів тестування 50](#_1tuee74)

[3.3 Опис контрольного прикладу 58](#_184mhaj)

[Висновки до розділу 67](#_4kx3h1s)

[4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 68](#_1f7o1he)

[4.1 Розгортання програмного забезпечення 68](#_2eclud0)

[4.2 Підтримка програмного забезпечення 69](#_3dhjn8m)

[Висновки до розділу 69](#_4cmhg48)

[ВИСНОВКИ 70](#_16x20ju)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 71](#_261ztfg)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| IDE | – | Integrated Development Environment – інтегроване середовище розробки. |
| --- | --- | --- |
| RPC | – | Remote Procedure Call – віддалений виклик процедури. |
| UDP | – | User Datagram Protocol – протокол дейтаграми користувача. |
| 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, а отже, й доступності нових масових пристроїв, таких як мобільні телефони для різних користувачів з усього світу. Завдяки цьому розробники отримують можливість створювати контент, що здатний охопити та задовольнити запити навіть найбільш вибагливих до платформи гравців.

Монополія – одна з найпопулярніших настільних ігор у світі, 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 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.

У даній курсовій роботі 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.

# АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Загальні положення

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 кіно та серіали. Далеко не в останню чергу такі високі показники забезпеченні розмаїттям напрямків цієї галузі, серед яких окремо можна визначити: сама розробка відеоігор під різні платформи, їх видавництво, кіберспорт та багато інших. Такий різноманітний спектр напрямків свідчить про величезний вплив цієї індустрії на сучасну глобальну культуру [2].

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 [1].

Перші експерименти зі створення відеоігор відбувалися завдяки окремим ентузіастам та науково-дослідницьким лабораторіям при університетах у 1950-1960-х роках, які відзначилася 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, 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 гральних приладів, 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 заліза. Окремо варто відзначити появу доступного інтернету, який дав можливість легше розповсюджувати свої розробки та завдяки якому з’явилися перспективи для розвитку багатокористувацьких онлайн-проєктів, 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 зайняти нову нішу. І це не дивно, оскільки вкладання сил та коштів в неї дає можливість охопити сотні мільйонів девайсів та їхніх користувачів через її головну привабливість – не потрібно купляти жодних додаткових приладів для ігрового процесу, гравець може долучатися будь-де та будь-коли до улюбленої гри, а вже згаданий раніше розвиток Інтернету лише полешлив отримання доступу до більш широкої бібліотеки електронних розваг та, як не дивно, прибутків компаніям, які отримали додаткові можливості для монетизації своїх проєктів [1].

Розробка відеогри – це складний і тривалий процес, який часто варіюється від проєкту до проєкту, але першим етапом потрібно правильно проаналізувати вимоги, які пов’язані з поточним проєктом. Під час цього аналізу необхідно визначити різноманітні аспекти, включаючи функціональні та нефункціональні аспекти гри, такі як ігролад, візуальна та звукова складові. Функціональними визначимо основні ігроладні механіки, кількість гравців тощо. Нефункціональними назвемо продуктивність самого проєкту, безпека даних, сумісність з різними платформами. До того ж, необхідно проаналізувати все з точки зору ігрового дизайну з метою забезпечення контролю рівня складності, взаємодії гравець-гра та гравець-гравець, візуальну та аудіо складову, які відповідатимуть загальному духу та вигляду продукту, оскільки все це впливатиме на ігровий досвід та подальше ставлення до гри у користувачів. З рештою, необхідне також тестування проєкту, аби кінцевий результат відповідав загальноприйнятим в індустрії нормам та забезпечив гідний рівень конкуренції [3].

Сучасні відеоігри різняться від своїх попередників, настільних ігор, але неможливо заперечувати наявність спільних ідей, концептів, ігроланих особливостей тощо. Як мінімум, більшість сучасних виключно ігрових жанрів мають попередників саме серед настільних ігор, наприклад, прабатьком всіх комп’ютерних рольових ігор є настільна рольова гра «Dungeons & Dragons», а стратегічні ігри завдячують таким представникам як шахи, «Монополія» та інші. Тобто можна стверджувати, що самі відеоігри є еволюційним явищем, розвитком класичних розваг у більш нову, відповідну до часу форму [1].

Всесвітньо відома настільна гра «Монополія» була випущена в світ у 1935 році. Вона являє собою класичну покрокову стратегію з економічним ухилом і, фактично, знайомить з основами капіталізму, а також дає можливість через гіперболізовані його прояви цікаво та захоплююче провести час. Відповідно, метою цієї гри є змусити збанкрутувати інших гравців, ставши таким чином монополістом, і досягається це купівлею, продажем та, звісно, перемовинами.

## Змістовний опис і аналіз предметної області

У світі постійного технологічного розвитку та високих вимог до якості програмного забезпечення, вивчення процесу використання знань предметної області в контексті розвитку ІТ-технологій стає надзвичайно важливим завданням.

Однією з ключових проблем у створених реалізаціях «Монополії» з багатокористувацьким режимом є перенавантажений інтерфейс, який ускладнює сприйняття гравцями гри та робить її менш зручною та зрозумілою для використання. Аналіз інтерфейсу дозволив ідентифікувати зайві компоненти та розробити план їх оптимізації, тим самим, забезпечивши більш зрозумілий та очевидний геймплей без використання зайвих непотрібних елементів інтерфейсу, кількість елементів та складність якого може перешкоджати певній частині аудиторії брати участь в ігровому процесі.

Іншою проблемою є відсутність реалізації «Монополії» у 2D, що може бути не зовсім зручним для окремих користувачів через підвищені системні вимоги у аналогах з 3D графікою та великою кількістю візуальних ефектів, які не несуть жодного сеансового навантаження, а часто відволікають від головної мети гри. До того ж, проблема системних вимог для мобільного пристрою може бути доволі вагома через потенційний виграш у зменшені енергоспоживання пристрою.

Ще одним важливим недоліком є занадто повільний темп гри, викликаний невисокими цінами на нерухомість та низькими штрафами що унеможливлює проведення довгих ігрових сеансів, що особливо актуально для мобільного пристрою через відсутність постійного джерела живлення та необхідності постійної підзарядки. Істотне завищення цін та накладання обмежень на обмін, одночасну кількість лотів під час процесу обміну полів суттєво спрощує процес стратегічного планування для гравців та забезпечує більш швидкі ігрові сеанси.

## Аналіз існуючих технологій та успішних IT-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації багатокористувацької мобільної Монополії. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

### Аналіз відомих алгоритмічних та технічних рішень

У даній розробці немає необхідності застосовувати спеціальні обчислювальні, або інші види алгоритмів.

Оскільки у створюваному програмного забезпеченні наявний багатокористувацький режим, то виникає необхідність вирішення типу підтримуваної мережевої архітектури у проєкті. Мережеві моделі для багатокористувацьких ігор описують архітектуру та дизайн, які використовуються для встановлення зв'язку між джерелом інформації та місцем призначення – наприклад, між клієнтом і сервером або між гравцями у одноранговій мережі. Серед доступних варіантів можна виокремити наступні: клієнт-серверна архітектура, однорангова централізована та однорангова децентралізована [7].

Клієнт-серверна модель з виділеним сервером – це тип мережі, у якій присутній 1 централізований сервер, який надсилає дані кінцевим користувачам, тобто клієнтами, які отримують цю інформацію, її обробляють та діють відповідно до неї. Клієнти також можуть надсилати дані на сервер. Ця модель забезпечує ефективний обмін інформацією між сервером та клієнтами, дозволяючи швидку передачу даних і забезпечуючи централізоване управління ресурсами. Важливим аспектом є можливість реалізації різних функцій на сервері, що полегшує синхронізацію та узгодженість в роботі всієї мережі. Крім того, використання цієї моделі може спрощувати моніторинг та обслуговування мережі завдяки її централізованій структурі [24].



Рисунок 1.1 – Схема клієнт-серверної архітектури

Переваги клієнт-серверної архітектури:

* Централізоване управління грою, що полегшує синхронізацію даних між гравцями та забезпечує загальну стабільність гри;
* Сервер може виконувати обробку важких розрахунків, зменшуючи обсяг обчислювального завдання на стороні клієнта;
* Ця архітектура сприяє централізованому управлінню безпекою та мінімізує можливість шахрайства.

Недоліки клієнт-серверної архітектури:

* Велика фізична відстань між сервером і гравцями може призводити до високої затримки;
* Підтримка централізованого сервера вимагає значних фінансових витрат. Великий обсяг трафіку, потреби у потужності обчислень та безпеці вимагають інфраструктури високого класу, що може бути дорогим завданням для розробників;
* При збільшенні кількості гравців можуть виникати проблеми з масштабованістю серверів. Додавання нових потужностей може призводити до зменшення продуктивності та якості обслуговування існуючих.

Вищеописаний клієнт-серверний тип мережі не є оптимальним варіантом для розробки через потенційну дорогу вартість обслуговування, а також можливу проблему довгої обробки запитів.

Децентралізований peer-to-peer, P2P (з англ. – рівний до рівного) – варіант однорангової архітектури системи, в основі якої стоїть мережа рівноправних вузлів. Існує декілька видів однорангових мереж, серед яких пірингова модель з'єднує всіх гравців між собою безпосередньо, тобто без сервера або виділеного хоста. На відміну від клієнт-серверних моделей, які вимагають виділеного сервера, за яким залишається вирішальне слово щодо загального стану гри, в P2P-моделі пристрій кожного гравця відповідає за ігрову логіку та взаємодію з іншими учасниками гри [8].

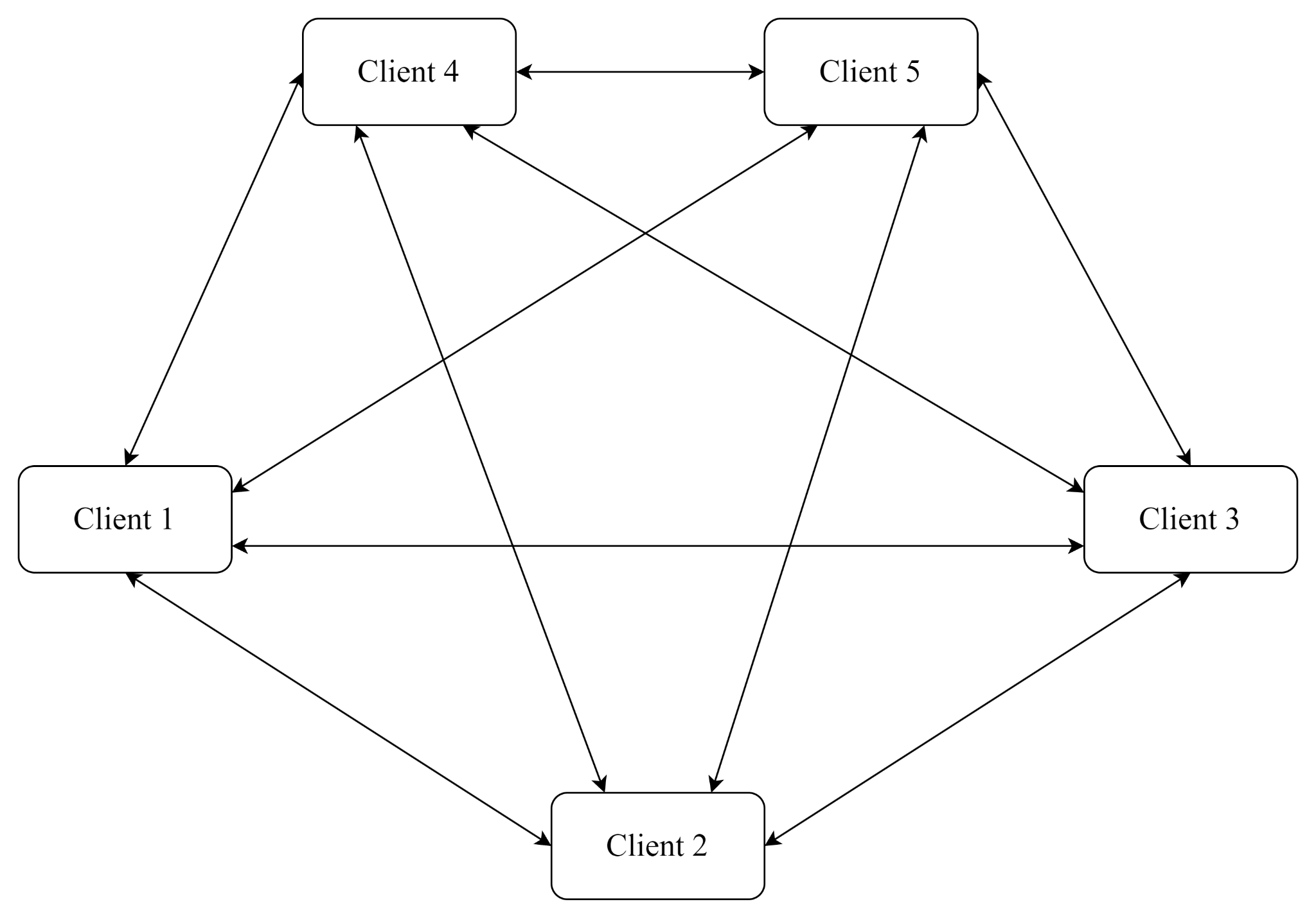


Рисунок 1.2 – Схема однорангової децентралізованої мережі

Переваги децентралізованої однорангової архітектури:

* Відсутність центрального сервера зменшує навантаження та залежність від централізованого вузла, що також забезпечує більшу стійкість до відмов;
* Рівномірне розподілення обробки може знизити затримки, оскільки кожен вузол може служити джерелом даних та обробляти їх;
* Як правило, є простішою у встановленні, оскільки не потрібно налаштовувати централізований сервер.

Недоліки децентралізованої однорангової архітектури:

* Складність масштабування через необхідність постійної синхронізації поточного стану гри між усіма її учасниками, оскільки вимагає ефективної координації та синхронізації між вузлами;
* Підвищена вразливість до протиправних дій, оскільки вся інформація про всі підключені клієнти у мережі відома іншим клієнтами, порівняно простий доступ до даних гри, що полегшує потенційне шахрайство;
* Може виникати затримка та нестабільність з'єднання через безпосередній обмін даними між вузлами;
* Можлива втрата даних через відсутність централізованої точки зберігання та обробки.

Вищеописана однорангова децентралізована архітектура мережі не є оптимальним варіантом через потенційні проблеми десинхронізації, відсутності центрального вузла, який здатен забезпечити загальну синхронізацію, а також загальну складність розробки та підвищеної складності масштабованості.

Централізована peer-to-peer, P2P (з англ. – рівний до рівного) мережа – це варіант однорангової архітектури, в якій забезпечується певний рівень централізації та координації за рахунок наявності центрального вузла, який є таким самим клієнтом-вузлом системи, як і інші гравці. Такий центральний вузол має назву host (з англ. – господар) – пристрій, який ініціює та контролює багатокористувацьку ігрову сесію. Основне визначення хоста полягає в тому, що це та сторона, яка виступає одночасно як гравець та центральний сервер [8].



Рисунок 1.3 – Схема однорангової централізованої мережі

Переваги централізована однорангової архітектури:

* Простота управління станом гри у порівнянні з децентралізованою одноранговою мережею через наявність хоста;
* Централізоване управління може полегшити впровадження безпекових заходів, які ускладнюють потенційну можливість до шахрування, а також через те, що звичайні клієнти не мають всієї повноти інформації про інших клієнтів у мережі;
* Ефективний розподіл обчислювальних ресурсів, оскільки потрібні обчислення можуть відбуватися лише на 1 клієнті, який потім може поділитися результатами своєї роботи.

Недоліки централізованої однорангової архітектури:

* Якщо центральний вузол виявився нестійким, або вразливим, це може призвести до збоїв всієї системи. Таким чином, якщо центральний вузол перестане працювати, то це викличе збій у всієї ігрової сесії;
* У випадку масштабованості цей процес проходитиме простіше на відміну від децентралізованої мережі, проте на якомусь етапі наявність єдиного центрального вузла, який ще й паралельно виконує роль клієнта може стати обмежуючим фактором виключно через складність виконуваних обчислень та завдань, які покладаються на сервер.

Таким чином, вищезгадана централізована однорангова мережа, незважаючи на певні недоліки є оптимальним рішенням для створюваної розробки, оскільки вона здатна забезпечити стабільну синхронізацію між клієнтами, завдяки відсутності необхідності підтримки виділених серверів, підвищеній безпеці у порівнянні з децентралізованою одноранговою.

### Аналіз допоміжних програмних засобів та засобів розробки

Для створення мобільної гри, яка буде підтримувати багатокористувацький режим необхідно вибрати відповідний фреймворк, який здатен забезпечити наступні вимоги:

* Простоту кросплатформної розробки;
* Наявність засобів для створення однорангової централізованої мережі;
* Присутність багатьох інструментів, які здатні допомогти у створенні відповідного програмного продукту.

Враховуючи наведені вимоги, вибір пав на один з актуальних ігрових рушіїв – Unity. Однією з ключових переваг цього рушія – це легка кросплатформенна розробка без необхідності суттєвої додаткової оптимізації у випадку створення продукту під вибрану платформу від початку. Ще однією ключовою його перевагою є наявність багатого вибору бібліотек та засобів для створення багатокористувацької гри, як офіційні, так і від сторонніх розробників з відкритим вихідним кодом. До того ж, Unity має велику кількість інструментів, які пришвидшують розробку подібних рішень, що особливо актуально для незалежних самостійних розробників [13].

Оскільки розробка програмного продукту виконуватиметься за допомогою Unity, то було обрано мову програмування пав на C#, оскільки це єдина підтримувана мова розробки для даного рушія.

Вибір інтегрованого середовище розробки пав на Visual Studio, оскільки вона має покращену інтеграцію з Unity, завдяки можливості хуку до процесу гри у реальному часі та дебагу вихідного коду прямо під час гри у неї.

Стосовно вибору бібліотеки для написання неткоду, то Unity має багато різних варіантів, як офіційні, так і від сторонніх розробників, найвідоміші рішення на даний момент: Mirror, Fish-Net, NGO (Netcode for GameObjects).

Mirror – відкрита бібліотека для створення додатків, які підтримують інтеграцію з мережевими службами та підсистемами. Ця бібліотека розроблена для спрощення створення багатокористувацьких ігор та додатків [19].

Переваги бібліотеки Mirror:

* Простий та зрозумілий API, який надає високий рівень абстракції від безпосереднього оперування та відправкою даних по мережі на низькому рівні;
* Велика активна спільнота користувачів і гарна документація;
* Наявна вбудована підтримка для роботи з серверами, що дає можливість створювати ігри з різної мережевою архітектурою.

Недоліки бібліотеки Mirror:

* Менш розвинений функціонал у порівнянні з іншими рішеннями за рахунок відсутності просунутої системи лобі;
* Відсутність проксі;
* Обмежена інтеграція з іншими наявними сервісами Unity.

Fish-Net – це відкрита бібліотека для створення багатокористувацьких ігор та додатків на Unity. Бібліотека розроблена для спрощення створення мережевих ігор, а також для надання високорівневого API, який абстрагує від низькорівневих деталей мережевого програмування [20].

Переваги бібліотеки Fish-net:

* Простий та зрозумілий API, який надає високий рівень абстракції від безпосереднього оперування та відправкою даних по мережі на низькому рівні;
* Наявність вбудованих компонентів для роботи з мережевими підключеннями, обміном даними та іншими аспектами мережевого програмування;
* Підтримка різних мережевих архітектур, включаючи P2P, клієнт-сервер та гібридні архітектури;
* Можливість масштабування ігор на велику кількість гравців.

Недоліки бібліотеки Fish-net:

* Бібліотека все ще перебуває в розробці, тому деякі функції можуть бути недоступні або мати обмежені можливості;
* Може бути складною для використання для початківців розробників.

NGO (Netcode for GameObjects) – це відкрита бібліотека для створення багатокористувацьких ігор та додатків на Unity. Бібліотека розроблена для спрощення створення мережевих ігор, а також для надання високорівневого API, який абстрагує від низькорівневих деталей мережевого програмування [15].

Переваги бібліотеки NGO:

* Офіційне рішення для багатокористувацьких ігор від Unity;
* Простий та зрозумілий API, який надає високий рівень абстракції від безпосереднього оперування та відправкою даних по мережі на низькому рівні;
* Наявність вбудованих компонентів для роботи з мережевими підключеннями, обміном даними та іншими аспектами мережевого програмування;
* Підтримка різних мережевих архітектур, включаючи P2P, клієнт-сервер та гібридні архітектури;
* Можливість масштабування ігор на велику кількість гравців;

Недоліки бібліотеки NGO:

* Бібліотека все ще перебуває в активній розробці, тому деякі функції можуть бути недоступні, мати обмежені можливості, або містити недоліки.

Бібліотекою для написання неткоду було обрано NGO (Netcode for GameObjects). Цей вибір був зроблений через широкий функціонал бібліотеки, який включає підтримку різних мережевих архітектур, масштабування на велику кількість гравців, а також через те, що NGO є офіційним рішенням від Unity, що гарантує його стабільну підтримку сумісність з боку розробників Unity [15].

### 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 мобільних 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» [12].

«Monopoly» – це мобільна версія класичної настільної гри, розроблена студією «Marmalade Game Studio», гра вийшла 2016 року на iOS та Android [12].

Переваги «Monopoly» від «Marmalade Game Studio»:

* Графіка та анімація. «Monopoly» виконана у 3Д, має чудову графіку та анімацію, які роблять гру більш привабливою та захопливою;
* Можливість власних налаштувань правил гри.

Недоліки «Monopoly» від «Marmalade Game Studio»:

* Ця гра є реалізацією класичної монополії, а отже перенесла всі недоліки з настільної гри, а саме – довгу тривалість гри, нудні описи полів шанс, тощо.
* Наявність руйнуючих ігролад покупок у внутрішньому магазині, які можуть надати нечесну перевагу гравцям над своїми суперниками.

«Monopoly GO» – це мобільна версія класичної настільної гри, розроблена студією «Scopely», гра вийшла 2022 року на iOS та Android [12].

Переваги «Monopoly» від «Scopely»:

* Виконана у 3Д, має чудову графіку та анімацію, які роблять гру більш привабливою та захопливою.

Недоліки «Monopoly GO» від «Scopely»:

* Довга тривалість гри, ігрові сесії можуть бути досить тривалими, особливо якщо грати з великою кількістю гравців.
* Гра пропонує внутрішні покупки, які можуть надати нечесну перевагу гравцям над своїми суперниками;
* У деяких моментах ігровий інтерфейс занадто громіздкий та незрозумілий.

Для порівняння курсової роботи з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

| Критерій порівняння | Курсачополія | Monopoly GO | Monopoly | Пояснення |
| --- | --- | --- | --- | --- |
| Спосіб розповсюдження | Відкритий вихідний код | Безкоштовна базова версія, внутрішньоігрові покупки | Платна базова версія, внутрішньоігрові покупки | Спосіб отримання продукту користувачам |

Продовження таблиці 1.1

| Продуктивність | Гра виконана у 2Д, без додаткових візуальних ефектів | Гра виконана у 3Д, з використанням додаткових візуальних ефектів | Гра виконана у 3Д, з використанням додаткових візуальних ефектів | Продуктивність гри на нормальному рівні дає можливість користувачам з різним рівнем пристроїв доторкнутися до продукту |
| --- | --- | --- | --- | --- |
| Зручність графічного користувацького інтерфейсу | Мінімальна кількість зайвих елементів, лише необхідний функціонал | Велика кількість вкладок з додатковим функціоналом, складність орієнтації для нових користувачів | Велика кількість вкладок з додатковим функціоналом, складність орієнтації для нових користувачів | Зручний користувацький інтерфейс дозволяє створити кращий ігровий досвід для користувачів |
| Середня тривалість гри | Швидкі ігрові сесії за рахунок змінених правил | Звичайні, довготривалі ігрові сесії | Звичайні, довготривалі ігрові сесії | Тривалість ігрових сесій є важливим аспектом, особливо, коли мова йде про мобільні пристрої |

## Аналіз вимог до програмного забезпечення

Мобільний застосунок, який розробляється, спрямований на створення реалізації мобільної гри «Монополія» з багатокористувацьким режимом до 5 гравців, забезпечуючи при цьому зручний графічний інтерфейс та покращені ігрові правила, які спрямовані на скорочення середнього часу ігрової сесії та покращення загального ігрового досвіду. Основна мета цього проєкту полягає у реалізації мобільної гри, який надасть гравцям можливість зіграти у відому настільну гру, використовуючи свій мобільний пристрій.

Серед функціональних задач гравцям надані наступні можливості:

* Створювати власні лобі та бути хостом ігрової сесії, або приєднуватися до ігрових кімнат у ролі користувача-клієнта;
* Кидання кубика та пересування ігровим полем на такі кількість кроків, яке дорівнює числу, яке було отримано внаслідок випавшого значення;
* Можливість взаємодії з ігровими полями, а саме наступні варіанти: купити власність, яка нікому ще не належить, варіант закласти поле, яке належить гравцю та на якому ще не побудовано жодних покращень, у іншому випадку – надається можливість продати покращення поля, також надається можливість покращити поле, тим самим збільшивши ціну оренди за перебування інших гравців на обраному полі;
* Можливість проводити обміни з іншими гравцями у тому випадку, якщо варіант обміну задовольняє умови валідації;
* Здатність взаємодії з обмінами, а саме прийняти, або відхилити його.
* Можливість здатися, тим самим гравець лишається всіх своїх полів, статків і більше не приймає участь у грі.

Всі вище перераховані функціональні задачі можна побачити на діаграмі використання. Діаграму використання можна побачити на рисунку 1.2.

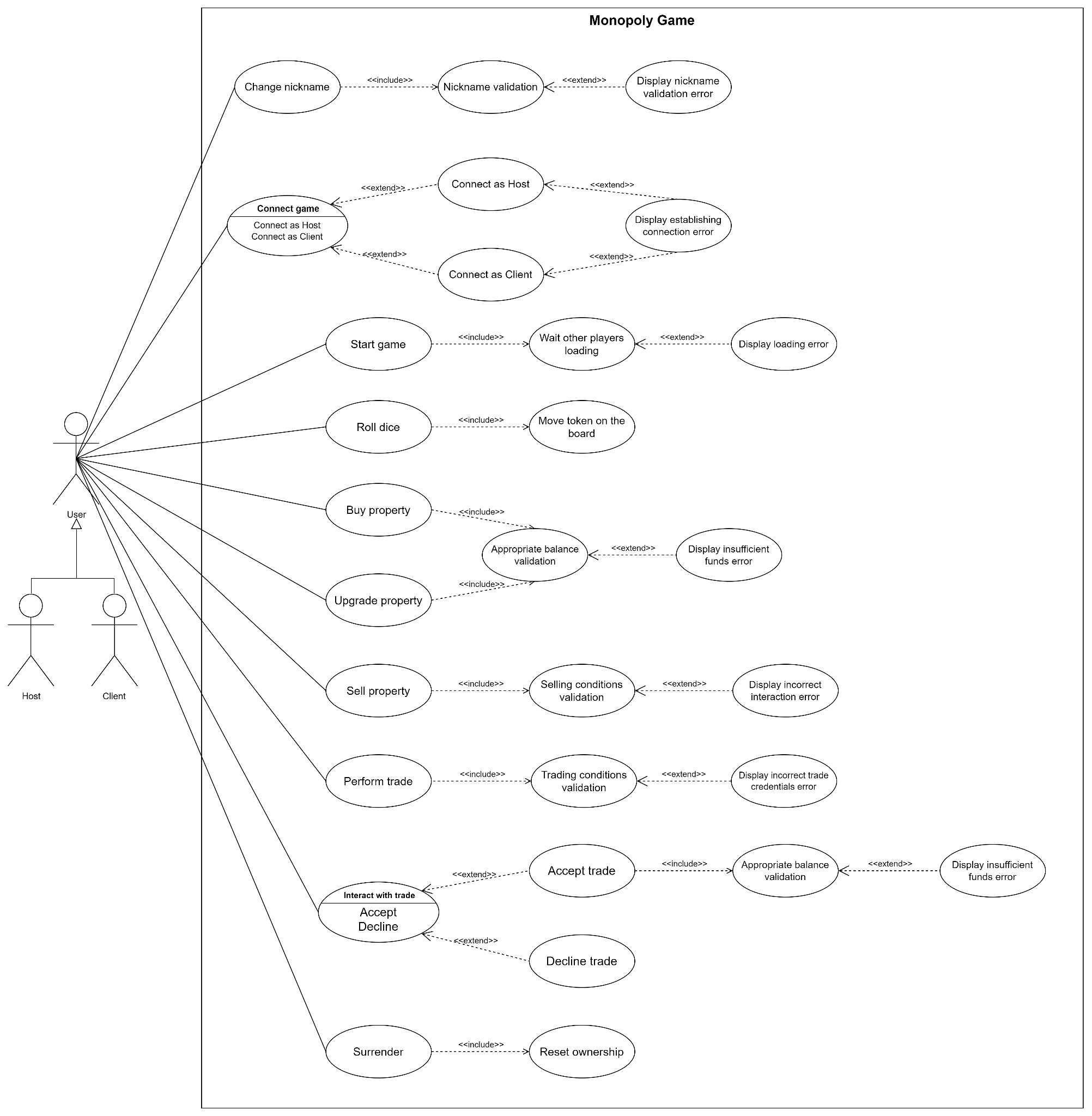


Рисунок 1.4 – Діаграма варіантів використання

В таблицях 1.2 – 1.13 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

| Use case name | Введення імені користувача |
| --- | --- |
| Use case ID | UC-01 |
| Goals | Зміна ім’я гравця |
| Actors | Користувач |
| Trigger | Користувач бажає змінити своє ім’я |
| Pre-conditions | Щойно відкрита гра, користувач знаходиться у головному меню |
| Flow of Events | Користувач вводить своє ім’я у відповідне поле для вводу нікнейму і натискає на кнопку підключення до лобі як клієнт, або створення власного лобі |
| Extension | У випадку введення некоректного значення, то показується діалогове вікно з відповідним повідомленням про некоректно введений нікнейм |
| Post-Condition | Ініціалізація початку підключення до ігрової кімнати |

Таблиця 1.3 - Варіант використання UC-2

| Use case name | Створення ігрової кімнати |
| --- | --- |
| Use case ID | UC-02 |
| Goals | Створення ігрового лобі |
| Actors | Користувач |
| Trigger | Користувач натиснув на кнопку створення гри |
| Pre-conditions | Введене ім’я користувача |
| Flow of Events | Користувач натискає на кнопку створення гри та, користувачу демонструється вікно з повідомленням про очікування |
| Extension | Повідомлення про помилку у випадку невдалого створення лоббі |
| Post-Condition | Створено нове ігрове лобі у якому знаходиться користувач |

Таблиця 1.4 - Варіант використання UC-3

| Use case name | Підключення до ігрової кімнати |
| --- | --- |
| Use case ID | UC-03 |
| Goals | Підключитися до лобі |
| Actors | Користувач |
| Trigger | Користувач натиснув на кнопку підключитися як клієнт та ввів відповідний правильний код ігрової сесії |
| Pre-conditions | Введені ім’я та код ігрової сесії |
| Flow of Events | Користувач натискає на кнопку підключення до лобі та очікує завантаження у нього |
| Extension | У випадку невдалого підключення видається повідомлення про помилку і через яку саме причину користувач не зміг під’єднатися |
| Post-Condition | Гравець успішно перемістився до лобі |

Таблиця 1.5 - Варіант використання UC-4

| Use case name | Ініціалізація початку гри |
| --- | --- |
| Use case ID | UC-04 |
| Goals | Початок гри |
| Actors | Користувач |
| Trigger | Користувач, який створив лобі 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 початку гри |
| Extension | У випадку невдалого початку гри відбувається повернення до лобі та показується діалогове вікно з відповідним повідомленням |
| Post-Condition | Гравці з лобі знаходяться безпосередньо у грі |

Таблиця 1.6 - Варіант використання UC-5

| Use case name | Кидання кубика |
| --- | --- |
| Use case ID | UC-05 |
| Goals | Кинути ігровий кубик, який вказує на кількість полів, якими може переміститися гравець |
| Actors | Користувач |
| Trigger | Користувач натиснув на кнопку кидання кубика |
| Pre-conditions | Наступила черга гравця виконувати свій |
| Flow of Events | Користувач натискає на кнопку кидання кубика і відбувається переміщення ігровим полем відповідно до отриманого числа |
| Extension | - |
| Post-Condition | Гравець перемістився ігровим полем |

Таблиця 1.7 - Варіант використання UC-6

| Use case name | Придбання власності |
| --- | --- |
| Use case ID | UC-06 |
| 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 цього, завершив свій хід і право наступного ходу успішно передається іншому гравцю |

Таблиця 1.8 - Варіант використання UC-7

| Use case name | Розбудова монополії |
| --- | --- |
| Use case ID | UC-07 |
| Goals | Побудувати покращення ігрового поля |
| Actors | Користувач |
| Trigger | Гравець натискає на відповідне поле, яке бажає покращити |
| Pre-conditions | У гравця зібрана повна монополія |
| Flow of Events | Користувач натискає на кнопку розбудови монополії, збільшується її рівень і відповідне значення демонструється за допомогою кількості зірок |
| Extension | У випадку недостатньої кількості засобів для придбання покращення поля виводиться діалогове вікно з відповідним повідомленням |
| Post-Condition | Гравець покращив поле |

Таблиця 1.9 - Варіант використання UC-8

| Use case name | Продаж власності |
| --- | --- |
| Use case ID | UC-08 |
| Goals | Закласти власність, щоб повернути суму за її придбання |
| Actors | Користувач |
| Trigger | Користувач натискає на поле, яке йому належить, у відкритому вікні натискає на відповідну кнопку продажу власності |
| Pre-conditions | Поле куплене і гравець є її власником |
| Flow of Events | Користувач натискає на відповідне поле, яке бажає продати і натискає на відповідну кнопку продажу |
| Extension | У випадку, якщо поле вже є закладеним і його неможливо продати, то користувач отримує відповідне повідомлення з текстом причини через яке невдалося продати поле |
| Post-Condition | Гравець заклав поле та отримав за це відповідну суму |

Таблиця 1.10 - Варіант використання UC-9

| Use case name | Ініціалізації здійснення обміну |
| --- | --- |
| Use case ID | UC-09 |
| Goals | Зробити обмін між гравцями |
| Actors | Користувач |
| Trigger | Користувач натискає на панель відповідного гравця, з яким прагне 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 прагне обмінятись, кількість ігрової валюти, яка братиме участь у процесу обміну і натискає на кнопку пропозиції обміну |
| Extension | У випадку неправильних умов здійснення обміну з’являється діалогове вікно з відповідним текстом повідомленням про помилку |
| Post-Condition | Гравець надіслав пропозицію обміну іншому гравцю |

Таблиця 1.11 - Варіант використання UC-10

| Use case name | Прийняття обміну |
| --- | --- |
| Use case ID | UC-10 |
| Goals | Прийняти запропонований обмін |
| Actors | Користувач |
| Trigger | Користувач натискає на кнопку прийняти обмін |
| Pre-conditions | Гравець отримав обмін 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 повернувся до поточного гравця |

Таблиця 1.12 - Варіант використання UC-11

| Use case name | Відхилення обміну |
| --- | --- |
| Use case ID | UC-11 |
| Goals | Відхилити запропонований обмін |
| Actors | Користувач |
| Trigger | Користувач натискає на кнопку відхилити обмін |
| Pre-conditions | Гравець отримав обмін від іншого гравця |
| Flow of Events | Користувач ознайомлюється з пропозицією обміну і приймає рішення про відхилення обміну і натискає на кнопку відхилити пропозицію обміну |
| Extension | - |
| Post-Condition | Обмін було відхилено, контроль повернувся до поточного гравця |

Таблиця 1.13 - Варіант використання UC-12

| Use case name | Ініціалізація процесу добровільної здачі |
| --- | --- |
| Use case ID | UC-12 |
| Goals | Добровільно здатися та завершити гру |
| Actors | Користувач |
| Trigger | Користувач натискає на власну панель і йому демонструється діалогове вікно з пропозицією здатися |
| Pre-conditions | - |
| Flow of Events | Користувач натискає на власну панель і йому у спливаючому діалоговому вікні підтверджує свій намір достроково залишити 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 може брати участь у процесі гри |

### Розроблення функціональних вимог

Програмне забезпечення розділене на функціональні задачі. Кожна окремо взята функціональна задача має свій певний набір функцій. У таблиці 1.14 наведено загальну модель вимог, а в таблицях 1.15 – 1.25 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити у таблиці 1.26.

Таблиця 1.14 – Загальна модель вимог

| ID | Призначення | Пріоритет |
| --- | --- | --- |
| FR-01 | Введення імені користувача | Високий |
| FR-02 | Створення ігрової кімнати | Високий |
| FR-03 | Підключення до ігрової кімнати | Високий |
| FR-04 | Ініціалізація початку гри | Високий |
| FR-05 | Кидання кубика | Середній |
| FR-06 | Придбання власності | Середній |
| FR-07 | Розбудова монополії | Середній |
| FR-08 | Продаж власності | Середній |
| FR-09 | Ініціалізації здійснення обміну | Низький |
| FR-10 | Прийняття обміну | Низький |
| FR-11 | Відхилення обміну | Низький |
| FR-12 | Ініціалізація процесу добровільної здачі | Низький |

Таблиця 1.15 – Функціональна вимога FR-1

| Назва | Зміна імені |
| --- | --- |
| Опис | Система повинна надавати можливість користувачу змінювати власне ім’я і зберігати його після кожної зміни і відновлювати під час нового запуску гри |

Таблиця 1.16 – Функціональна вимога FR-2

| Назва | Створення власної ігрової кімнати |
| --- | --- |
| Опис | Гра має мати можливість створення власного лобі, якщо гравець прагне бути хостом ігрової сесії і відповідати за початок і завершення гри |

Таблиця 1.17 – Функціональна вимога FR-3

| Назва | Підключення до створеної ігрової кімнати |
| --- | --- |
| Опис | Гра має мати можливість підключення до ігрового лобі, у випадку, якщо користувач вводить відповідний правильний код ігрової сесії |

Таблиця 1.18 – Функціональна вимога FR-4

| Назва | Початок гри |
| --- | --- |
| Опис | Користувач-хост повинен мати можливість запустити гру, тим самим закриваючи лобі не дозволяючи іншим гравцям, які навіть знають правильний код під’єднатись до нього. |

Таблиця 1.19 – Функціональна вимога FR-5

| Назва | Кидання кубика |
| --- | --- |
| Опис | Коли наступає черга відповідного гравця, то у цього користувача має бути можливість кинути кубик, щоб виконати переміщення ігровим полем, без кинутого кубика хід гравця не може бути зарахований, окрім певних виключних випадків. |

Таблиця 1.20 – Функціональна вимога FR-6

| Назва | Купівля власності |
| --- | --- |
| Опис | У випадку, коли гравець кинув кубик і перемістив на поле власності, яка нікому не належить, користувачу має надаватися змога купити цю власність. |

Таблиця 1.21 – Функціональна вимога FR-7

| Назва | Покращення власності монополії |
| --- | --- |
| Опис | Коли гравець зібрав повноцінну монополію, то користувачу має відкритися можливість покращувати її поля, тим самим підвищуючи ціну за оренду перебування на ній інших гравців. |

Таблиця 1.22 – Функціональна вимога FR-8

| Назва | Продаж, або закладення власності |
| --- | --- |
| Опис | Користувачу має бути доступна опція продажі покращень на полі зібраної монополії, або тимчасове закладання придбаної власності. |

Таблиця 1.23 – Функціональна вимога FR-9

| Назва | Здійснення обміну |
| --- | --- |
| Опис | Гравцю має бути надана можливість запропонувати обмін іншому гравцеві, у випадку якщо запропонована пропозиція обміну відповідає всім правилам та вимогам. |

Таблиця 1.24 – Функціональна вимога FR-10

| Назва | Можливість взаємодіяти з обміном |
| --- | --- |
| Опис | Гравцю, якому запропонували обмін має бути надана можливість прийняти його умови, або відхилити пропозицію з відповідними наслідками. |

Таблиця 1.25 – Функціональна вимога FR-11

| Назва | Можливість достроково здатися та покинути гру |
| --- | --- |
| Опис | У користувача має бути можливість здатися, тим самим він вибуває з гри, лишається всіх свої накопичених статків та зібраних власностей |

Таблиця трасування вимог використовується для аналізу та управління проєктами, основне її призначення спрямоване на відстеження залежностей між різними елементами цього проєкту. Її головна мета полягає у забезпеченні зв'язку між вихідними вимогами до системи і конкретними елементами, які реалізують ці вимоги. У випадку розробки програмного забезпечення, така таблиця зазвичай включає список вимог або функціональних можливостей, а також інші аспекти, наприклад, сценарії використання, або тестові випробування, тощо.

Таблиця 1.26 – Матриця трасування вимог

| UC  FR | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | X |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  | X |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  | X |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  | X |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  | X |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  | X |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  | X |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  | X |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  | X |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  | X | X |  |
| 11 |  |  |  |  |  |  |  |  |  |  |  | X |

### Розроблення нефункціональних вимог

Нефункціональні вимоги – це характеристики системи чи програмного забезпечення, які не стосуються конкретної функціональності, але визначають якісні аспекти їхньої роботи та характеристики. Ці можливості визначають "якість" системи та включають такі аспекти, як продуктивність, надійність, безпека та інші.

Таблиця 1.27 – Нефункціональні вимоги

| Номер | Категорія | Опис |
| --- | --- | --- |
| NFR-1 | Продуктивність | Гра повинна забезпечувати нормальний рівень швидкодії та оптимізації, 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 шляхом використання зашифрованих шляхів їх передачі. |
| NFR-5 | Зручність інтерфейсу | Гра повинна мати інтуїтивно зрозумілий та зручний для використання користувальницький інтерфейс. |

## Постановка задачі

Мобільний застосунок спрямований на створення реалізації мобільної гри «Монополія» з багатокористувацьким режимом до 5 гравців, забезпечуючи при цьому зручний графічний інтерфейс та покращені ігрові правила, які спрямовані на скорочення середнього часу ігрової сесії та покращення загального ігрового досвіду. Основна мета цього проєкту полягає у реалізації мобільної гри, який надасть гравцям можливість зіграти у відому настільну гру, використовуючи свій мобільний пристрій.

Під час процесу розробки будуть вирішені та реалізовані наступні функціональні задачі: можливість створювати власні лобі та бути хостом ігрової сесії, або приєднуватися до ігрових кімнат у ролі користувача-клієнта; кидання кубика та пересування ігровим полем на такі кількість кроків, яке дорівнює числу, яке було отримано внаслідок випавшого значення; можливість взаємодії з ігровими полями, а саме наступні варіанти: купити власність, яка нікому ще не належить, варіант закласти поле, яке належить гравцю та на якому ще не побудовано жодних покращень, у іншому випадку – надається можливість продати покращення поля, також надається можливість покращити поле, тим самим збільшивши ціну оренди за перебування інших гравців на обраному полі;

можливість проводити обміни з іншими гравцями у тому випадку, якщо варіант обміну задовольняє умови валідації; здатність взаємодії з обмінами, а саме прийняти, або відхилити його; можливість здатися, тим самим гравець лишається всіх своїх полів, статків і більше не приймає участь у грі.

Під час розробки мобільної гри основна увага буде спрямована на забезпечення високої продуктивності та ефективної оптимізації, мінімізації впливу гри на заряд батареї пристрою. Одночасно особлива увага приділяється сумісності гри з різними мобільними пристроями, забезпечуючи доступність для різноманітної аудиторії. Захист взаємодії гравців та конфіденційності їхніх даних буде досягнуто за допомогою шифрованих каналів передачі даних.

## Висновки до розділу

В даному розділі курсової роботи було виконано комплексний аналіз предметної області, що включав в себе загальний огляд предметної області, визначення основних термінів і напрямків розробок. Додатково був проведений змістовний опис і аналіз поточного стану використання знань предметної області в сфері інформаційних технологій, з фокусом на реалізацію в існуючих сучасних програмних рішеннях.

Детально були розглянуті існуючі технології та успішні проєкти в даній області, зокрема алгоритмічне забезпечення, технічні рішення, допоміжні програмні засоби, засоби розробки та готові конкуренті програмні рішення. Проведено аналіз відомих алгоритмів та технічних рішень, вибір оптимальних шляхів для розробки. Також проведений огляд допоміжних програмних засобів та засобів розробки, їх порівняльний аналіз для вибору найбільш підходящих інструментів для поточної розробки.

Додатково, було проведено порівняння розробленого програмного продукту з відомими аналогами, яке було представлене у вигляді таблиці, яка містить функціональні та нефункціональні особливості кожного продукту для більш глибокого та детального порівняння.

Також було виконано аналіз вимог до програмного забезпечення, включаючи опис головного функціоналу, діаграму варіантів використання та детальний опис варіантів використання. На основі цього визначено головні завдання та вимоги програмного забезпечення. Далі розроблені функціональні та нефункціональні вимоги, включаючи модель вимог, їх опис та матрицю трасування. Додатково було сформовано саме завдання курсової роботи, визначені головні задачі та підзадачі на шляху до виконання завдання.

# МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Моделювання та аналіз програмного забезпечення

Для моделювання та аналізу програмного забезпечення було використано BPMN (Business Process Model and Notation). BPMN – це стандарт для моделювання бізнес-процесів, який надає уніфікований мовний засіб для спільного розуміння, аналізу та оптимізації бізнес-процесів. Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 2.1 та 2.2).



Рисунок 2.1 – BPMN-модель для гравців обох типів під час під’єднання до ігрової сесії та початку гри

Опис послідовності під час під’єднання до ігрової сесії:

* Користувач вводить ім’я у відповідне поле;
* Якщо користувач планує виконувати роль хоста ігрової сесії, то натискає відповідну кнопку;
* Якщо користувач планує приєднатися як клієнт, то вводить код ігрової сесії;
* Очікування завантаження у лобі;
* Очікування інших гравців та початку гри.

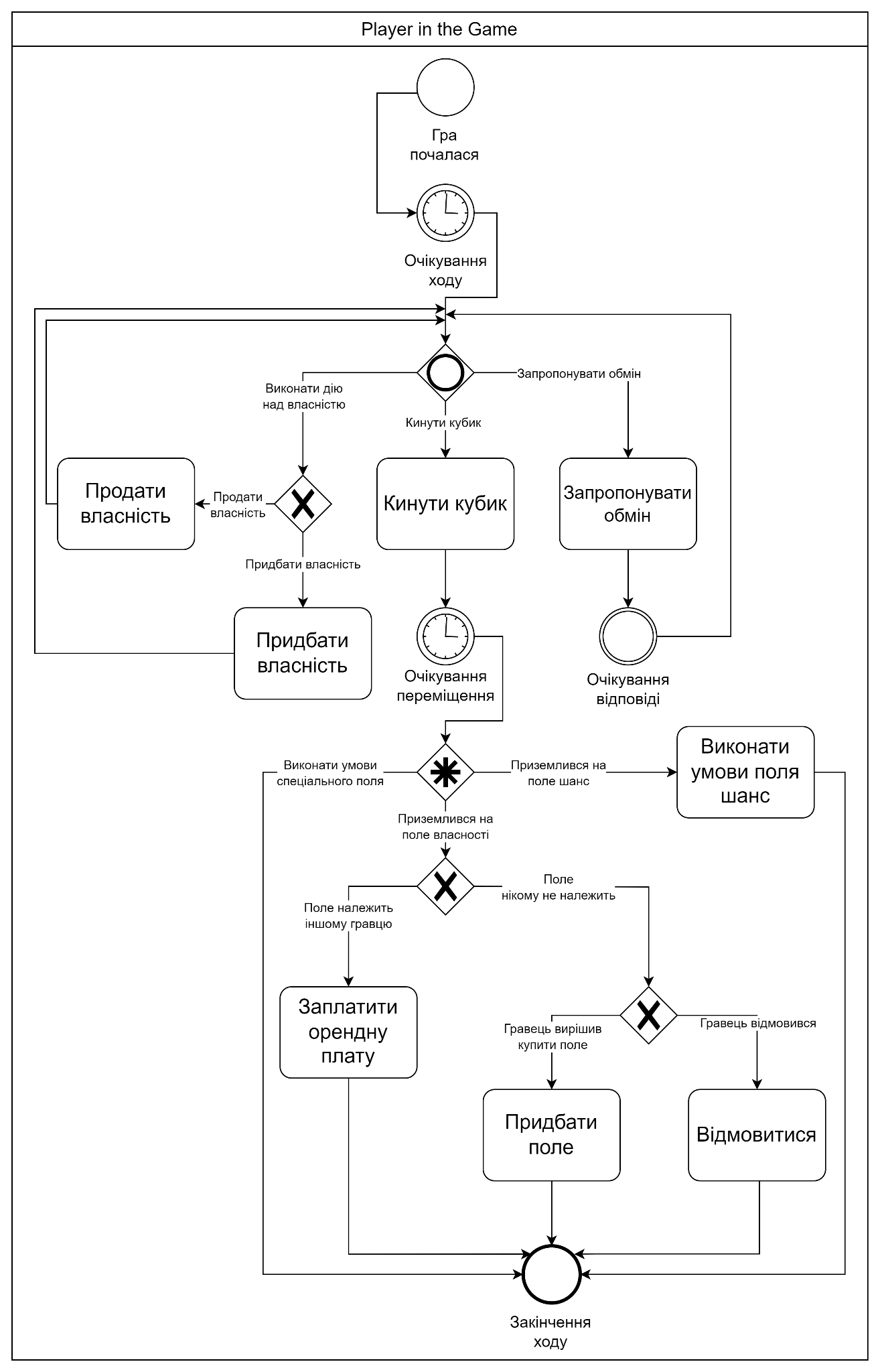


Рисунок 2.2 – BPMN-модель для гравця під час виконання ходу

Опис послідовності під час виконання ходу:

* Гравець очікує початку свого ходу;
* Якщо гравець обирає виконати дію над власністю, то виконує необхідну дію;
* Якщо гравець прагне почати процес обміну, то виконує необхідну дію;
* Якщо гравець кидає кубик, то відбувається переміщення ігровим полем;
* Очікування переміщення полем;
* Якщо гравець приземлився на поле шанс, то виконує необхідні дії;
* Якщо гравець приземлився на особливе поле, то виконує необхідні дії;
* Якщо гравець приземлився на поле власності, яка належить іншому гравцю, то сплачує оренду;
* Якщо гравець приземлився на поле власності, яке нікому не належить, то пропонується придбати цю власність.



Рисунок 2.3 – BPMN-модель для гравця під виконання обміну

Опис послідовності під час виконання обміну:

* Заповнення даних для обміну;
* Відправка обміну;
* Прийняття, або відмові від обміну від іншого гравця;
* Повернення керування до відправника.

## Архітектура програмного забезпечення

Під час розробки відеогри використання патернів проектування стає ключовим етапом для забезпечення ефективності, читабельності та легкості керування кодом і внесенням змін у нього. Для розробки використовувалися наступні патерни: Entity-Component-System (ECS) та Model-View-Controller (MVC).

Entity-Component-System (ECS) – у випадку використання цього патерну об'єкти в грі складаються з компонентів, а системи обробляють і взаємодіють з цими компонентами. Кожен модуль відповідає за конкретну функціональність, і об'єкти можуть легко модифікуватися та розширюватися за допомогою додавання або видалення компонентів [6].

Model-View-Controller (MVC) – відповідає за відокремлення компонентів гри, таких як логіка гри, графічний інтерфейс та дані гри, забезпечуючи модульність та гнучкість архітектури. У випадку розроблюваної мобільної реалізації «Монополії» модель включає в себе дані та ігрову логіку. Це може включати інформацію про стан гри (баланс гравців, поточна власність, обробка подій) та правила, за якими гра функціонує. Модель визначає, як гравець може рухатися, купувати власність, отримувати прибуток тощо. Представлення відповідає за те, як гра відображається користувачеві на екрані мобільного пристрою. Це може включати графічний інтерфейс та інші графічні аспекти відображення гри. Контролер відповідає за обробку введення користувача та взаємодію з моделлю. На мобільних пристроях це може включати елементи керування, такі як доторкання, ввод від користувача у текстові поля, тощо [5].

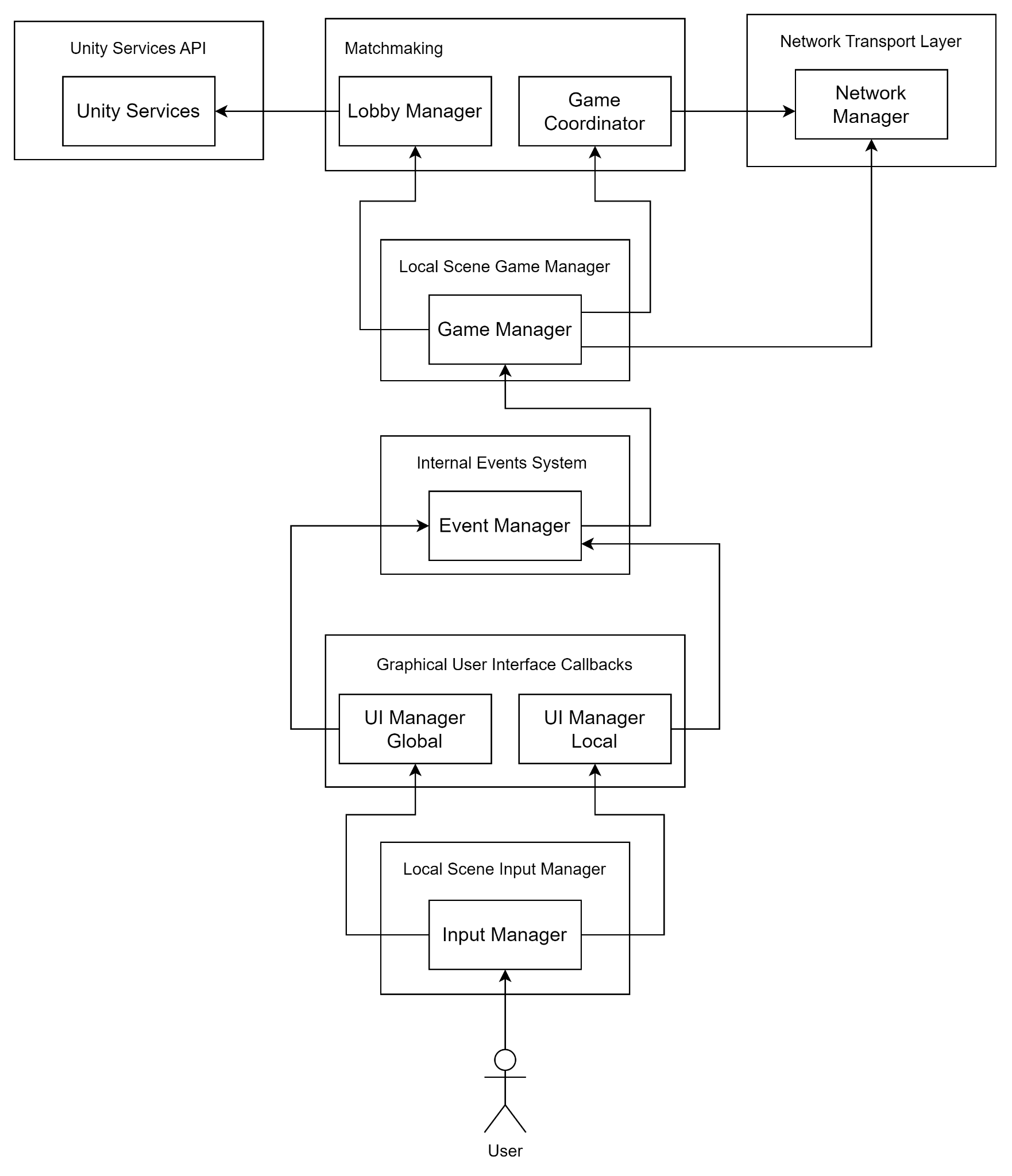


Рисунок 2.4 – Загальна діаграма архітектури програмного забезпечення

## Конструювання програмного забезпечення

Розробка мобільного ігрового застосунку була досягнута за рахунок створення наступних класів, кожен з яких виконує свою функцію.

Таблиця 2.1 – Опис класів

| Назва | Опис |
| --- | --- |
| GameCoordinator | Ігровий координатор, який відповідає за підключення гравців до спільної ігрової сесії, менеджмент ресурсів мережі, оновлення інформації про кожного гравця, тощо. |
| LobbyManager | Менеджер ігрового лобі, який відповідає за оновлення стану лобі, інформації про гравців у ньому, початок гри, тощо. |
| GameManager | Ігровий менеджер, який відповідає за синхронізацію стану гри, безпосередню ігрову логіку, розсилку повідомлень, оновлення значень. |
| MonopolyBoard | Допоміжний клас для роботи з полями монополії. |
| MonopolyNode | Клас-представлення поля монополії з відповідною інформацією про ціну за оренду, значення поточного рівня, тощо. |
| MonopolyPlayer | Клас-представлення гравця у монополію, з відповідними методами для взаємодії з ігровим світом. |
| MonopolyPlayerVisuals | Клас-представлення візуальної складової гравців. |
| MonopolySet | Клас для зберігання інформації про монополії. |

Продовження таблиці 2.1

| ChanceNodeSOEditor | Редактор інформації про поле шанс з монополії. |
| --- | --- |
| MonopolyNodeEditor | Редактор інформації про поле монополії. |
| ChanceNodeSO | Клас-шаблон для полів шанс. |
| ObjectPoolMessageBoxes | Клас-реалізація патерну Object Pool для спливаючих вікон. |
| ObjectPoolPlayersLobby | Клас-реалізація патерну Object Pool для представлення гравців у лобі. |
| PanelInfoUI | Клас-представлення інформативної панелі. |
| PanelMessageBoxUI | Клас-представлення панелі спливаючого вікна. |
| PanelMonopolyNodeUI | Клас-представлення панелі спливаючого вікна. |
| PanelOfferUI | Клас-представлення панелі пропозиції купівлі поля. |
| PanelPaymentChanceUI | Клас-представлення панелі сплати штрафу. |
| PanelPaymentPropertyUI | Клас-представлення панелі сплати податку. |
| PanelPlayerGameUI | Клас-представлення панелі гравця у грі. |
| PanelPlayerLobbyUI | Клас-представлення панелі гравця у лобі. |
| PanelReceiveTradeUI | Клас-представлення панелі отриманого обміну. |
| PanelTradeOfferUI | Клас-представлення панелі створення обміну. |
| UIManagerBootstrap | Клас, який відповідає за взаємодію з графічним користувацьким інтерфейсом на сцені завантаження необхідних компонентів гри. |

Продовження таблиці 2.1

| UIManagerGameLobby | Клас, який відповідає за взаємодію з графічним користувацьким інтерфейсом на сцені ігрового лобі. |
| --- | --- |
| UIManagerGlobal | Клас, який відповідає за взаємодію з графічним користувацьким інтерфейсом, який може бути викликаний на будь-якій сцені. |
| UIManagerMainMenu | Клас, який відповідає за взаємодію з графічним користувацьким інтерфейсом на сцені головного меню. |
| UIManagerMonopolyGame | Клас, який відповідає за взаємодію з графічним користувацьким інтерфейсом на самої гри. |
| UINodeTouchHandler | Клас, який відповідає за обробку вводу користувача. |

Оскільки у розробленому ігровому застосунку наявний багатокористувацький режим, то існує необхідність розробки відповідних сутностей для роботи з підключенням до ігрової сесії та загальному менеджменту ігрового процесу одночасно для кількох гравців. Саме тому було прийнято рішення про розробку ігрового координатора.

Ігровий координатор – сутність, яка відповідає за організацію ігрового процесу та за взаємодію між різними складовими гри у багатокористувацький ігровій сесії, де на нього покладена роль пошуку матчів, під’єднання гравців до спільної ігрової сесії, менеджментом мережевих та інших ресурсів, тощо [16].

Ще одним важливим компонентом розробленого програмного забезпечення, на яке необхідно звернути увагу – LobbyManager, який виконує роль координатора завантаженого ігрового лобі. Його роль полягала у взаємодії з API Unity-сервісів для оновлення поточного стану лобі, кількості гравців, інформації про них [14].

Через специфіку розробленого програмного забезпечення під час розробки використовувалися спеціалізовані для створення неткоду структури даних та методи. Специфіка написання неткоду передбачала використання механізмів серіалізації та десеріалізації даних для ефективного обміну інформацією між клієнтом та сервером. Система використовувала протоколи серіалізації, protobuffs (protocol buffers), щоб перетворювати структури даних у бінарний формат, який може бути легко переданий через мережу. Це було важливо для забезпечення ефективного обміну даними та зменшення обсягу пакетів з інформацією, які передаються. Додатково варто відзначити фактичну неможливість передачі всіх типів даних, які зберігаються на купі пам’яті, тому дозволена лише серіалізація та передача примітивних типів даних [18].

Додатково варто відзначити Remote Procedure Calls (RPC), які зіграли ключову роль у взаємодії між клієнтом і сервером. Використання RPC дозволило викликати методи на віддаленому сервері та клієнті так, ніби вони викликаються локально. Це полегшувало взаємодію між клієнтом та сервером, дозволяючи їм обмінюватися даними, оновлювати стан гри та виконувати дії в реальному часі. RPC є важливим механізмом, який спрямований на забезпечення взаємодії між програмами, що працюють на різних вузлах мережі. Це дозволяє викликати функції або процедури на віддалених серверах так, ніби вони викликаються локально. RPC виграє ключову роль у багатьох сценаріях, зокрема в розробці багатокористувацьких ігор, де взаємодія між гравцями та серверами є критично важливою. Цей механізм ґрунтується на ідеї абстрагування віддалених викликів від деталей мережевого обміну даними. Використання RPC дозволяє розробникам уникнути ручного управління мережевими протоколами та деталями взаємодії між клієнтами та серверами. Це робить процес розробки багатокористувацьких ігор більш ефективним та полегшує створення стійких до помилок та ефективних мережевих додатків [21].

Окрім того, для забезпечення ефективної взаємодії, використовувалися спеціальні структури даних, які містили параметри, необхідні для визначення та виклику відповідних функцій або методів на клієнті чи сервері. Вони служили інструментом для точного визначення, на якій конкретній машині мали виконатися відповідні дії локально з метою синхронізації стану гри [15].

Варто зазначити особливість створення багатокористувацьких ігор з одноранговим централізованим типом мережі, яка полягає у тому, що клієнти не можуть напряму викликати RPC інших клієнтів, оскільки це зроблено з метою забезпечення безпеки. Тому пайплайн синхронізації дій за допомогою неткоду має бути реалізований через локальний виклик методу оновлення стану гри і примусового виклику віддаленої процедури на сервері, який потім, у свою чергу, розсилає оновлення іншим зацікавленим клієнтам [16].

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.22.

Таблиця 2.22 – Опис утиліт

| №  п/п | Назва утиліти | Опис застосування |
| --- | --- | --- |
| 1 | Microsoft Visual Studio | Головне середовище розробки програмного забезпечення курсової роботи. |
| 2 | Unity Engine | Програмне забезпечення необхідне для безпосереднього створення гри завдяки наданню широкого набору інструментів, певних готових компонентів, наявності користувацького інтерфейсу. |
| 3 | Adobe Illustrator | Програмне забезпечення, яке використовувалося для створення растрової графіки у грі, а саме іконок, логотипів полів та деяких інших графічних елементів. |

## Аналіз безпеки даних

Relay Service від Unity – це служба, яка забезпечує з'єднання між гравцями в багатокористувацьких іграх. Relay використовує протокол UDP для передачі даних між гравцями, а також забезпечує шифрування даних для безпеки. Протокол UDP (User Datagram Protocol) – це протокол передачі даних, який використовується для передачі даних в реальному часі. UDP не гарантує доставку даних, але він забезпечує низьку затримку і низький додаткове навантаження.

UDP – це один із протоколів транспортного рівня в мережевій моделі OSI та мережевих протоколів. Протокол UDP є не надійним, тобто він не забезпечує гарантії доставки та порядку при передачі даних. Основна відмінність між UDP та іншим популярним протоколом транспортного рівня, таким як TCP (Transmission Control Protocol), полягає в тому, що UDP не встановлює стійке з'єднання перед передачею даних і не використовує підтвердження прийому. Це робить його швидшим, але менш надійним, особливо в умовах, коли частина даних може бути втрачена або доставлена не в потрібному порядку. Протокол UDP широко використовується для передачі даних у реальному часі, таких як передача відео та аудіо, ігри та інші додатки, де більший акцент робиться на швидкості передачі даних, ніж на гарантії доставки [9].

Інформація, яка передається через Relay, шифрується протоколом DTLS для безпеки. Протокол DTLS (Datagram Transport Layer Security) – це протокол шифрування, який використовується для захисту даних, які передаються через UDP [10]. DTLS забезпечує шифрування даних, щоб захистити їх від перехоплення і розшифровки. DTLS використовується в Relay від Unity для захисту даних, які передаються між гравцями, це допомагає забезпечити безпеку і конфіденційність для гравців [11].

## Висновки до розділу

У цьому розділі був представлений обширний аналіз ключових аспектів розробки ігрового застосунку, основною особливістю якого є багатокористувацький режим. Були детально розглянуті ключові аспекти процесу розробки, враховуючи бізнес-процеси, архітектурні патерни, алгоритми та безпеку програмного забезпечення.

Перше, що слід відзначити, це створення опису бізнес-процесів розробки, представлені за допомогою BPMN-діаграм, що дозволяє ефективно визначити та оптимізувати різноманітні етапи розробки, забезпечуючи високий рівень продуктивності та якості.

Відомо, що ефективна, стабільна та масштабована архітектура грає важливу роль у забезпеченні стабільного та високопродуктивного продукту. Під час розробки було вирішено використовувати архітектурний паттерн Entity-Component-System (ECS) для створення гнучкої системи, де об'єкти гри складаються з компонентів, що полегшує модифікацію та розширення функціональності. Також був впроваджений Model-View-Controller (MVC) для відокремлення компонентів гри, що дозволяє досягти модульності та гнучкості в архітектурі.

Також важливо відзначити конструювання програмного забезпечення, на якому були описані головні виклики, проблеми та особливості, які виникають під час створення гри з багатокористувацьким режимом. Важливим рішенням було впровадження ігрового координатора, що дозволило ефективно керувати ігровими сесіями. Ігровий координатор відповідає за організацію ігрового процесу та взаємодію між гравцями, забезпечуючи швидке підключення та менеджмент ресурсів. У розділі було також розглянуто неткод, який включає в себе використання серіалізації, protobuffs для ефективного обміну даними через мережу, а також RPC, які виявилися ключовим механізмом для забезпечення взаємодії між клієнтом і сервером. Використання RPC дозволило здійснювати виклики методів на віддаленому сервері та клієнті так, ніби вони викликаються локально, що важливо для реалізації взаємодії у реального часі.

# АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Аналіз якості ПЗ

Аналіз якості ПЗ було проведено з допомогою веб-сервісу Codacy, який автоматично генерує звіт якості коду за різними метриками. Перевагами цього сервісу є можливість обирати які самі файли з кодом необхідно включити в загальну оцінку. Ця перевага є особливо актуально для комплексних ігрових застосунків через велику кількість авто згенерованих файлів з кодом, які не потребують оцінки [13].

Метрики, за якими проводитиметься оцінка якості коду розробки:

* Issues – ця метрика вказує на наявність помилок, або інших проблем у коді. Проблеми можуть бути різного характеру, такі як логічні помилки, некоректне використання змінних, тощо. Оцінка проблем дозволяє визначити, наскільки ефективно функціонал коду виконується і які можливі проблеми можуть виникнути під час його виконання.
* Complexity – визначається різними факторами, такими як кількість вузлів у контрольному потоці, різноманітність використаних алгоритмів, глибина вкладених структур, тощо. Вища складність може призвести до складнішого розуміння коду, важкості виправлення помилок і втрати продуктивності при роботі над ним. Зазвичай висока складність вказує на потенційні проблеми, такі як неефективність і важкість супроводу.
* Duplication – вказує на наявність однакового чи подібного коду у різних частинах програми. Це може бути показником недостатньої абстракції і наявності повторного використання коду. Виявлення та усунення дублювання може поліпшити читабельність, обслуговуваність і зменшити ризик виникнення помилок, оскільки будь-які зміни потрібно буде внести лише в одному місці.



Рисунок 3.1 – Загальна оцінка вихідного коду проєкту

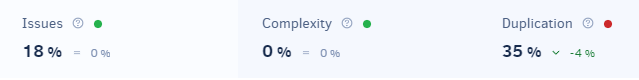


Рисунок 3.2 – Оцінка якості коду проєкту за метриками



Рисунок 3.3 – Кількість наявних проблем у коді

Щодо отриманих результатів важливо зазначити, що оцінка якості коду часто є суб'єктивною. Розробники можуть мають різні погляди на те, що вважається гарними підходами до використання певних синтаксичних конструкцій. Оцінки можуть сильно варіюватися в залежності від досвіду, стилю програмування та особистих уподобань. Також якість коду може залежати від контексту і вимог проекту. Те, що вважається гарним рішенням для одного проекту, може бути несхожим на іншому, залежно від специфіки завдань та потреб користувачів. Цей аспект є доволі актуальним для розробки ігор загалом, а особливо для багатокористувацьких проєктів, оскільки у них може використовуватися багато дублюючого коду синхронізації у неткоді, або код для створення редакторів певних компонент. Додатково варто зазначити, що зовнішні фактори, такі як обмежені ресурси, терміни проекту, стратегічні цілі компанії можуть впливати на рішення розробників та якість їхнього коду [18].

## Опис процесів тестування

Для тестування будуть використані функціональні та мануальні тести, які будуть перевіряти правильність виконання всіх функціональних вимог. Варто зазначити, що оскільки розроблюваним продуктом є ігровий мобільний додаток, то створення та впровадження автоматичних тестів різного рівня є бажаним, але часто неможливими завданням саме у сфері розробки відеоігор [23].

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.1 – 3.30.

Таблиця 3.1 – Тест 1.1

| Тест | Зміна імені користувача |
| --- | --- |
| Модуль | Головне меню |
| Номер тесту | 1.1 |
| 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 10 символів. |
| Очікуваний результат | Зміна імені проходить успішно, користувач спокійно може приєднатися до ігрового лобі. |

Продовження таблиці 3.2

| Фактичний результат | Зміна імені проходить успішно, користувач успішно змінив своє ім’я та може перейти до створення власного лобі, або підключення до існуючого |
| --- | --- |

Таблиця 3.2 – Тест 1.2

| Тест | Створення ігрової кімнати |
| --- | --- |
| Модуль | Головне меню |
| Номер тесту | 1.2 |
| Початковий стан системи | Гравець знаходиться у головному меню та ввів своє ім’я |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на кнопку створення лобі та очікує завантаження ігрової кімнати. |
| Очікуваний результат | Користувач успішно завантажився в ігрове лобі. |
| Фактичний результат | Після очікування завантаження гравець опинився в ігровому лобі. |

Таблиця 3.3 – Тест 1.3

| Тест | Підключення до ігрової кімнати |
| --- | --- |
| Модуль | Головне меню |
| Номер тесту | 1.3 |
| Початковий стан системи | Гравець знаходиться у головному меню та ввів своє ім’я |
| Вхідні дані | Код ігрової сесії |

Продовження таблиці 3.3

| Опис проведення тесту | Користувач натискає на кнопку підключення до лобі та очікує завантаження ігрової кімнати. |
| --- | --- |
| Очікуваний результат | Користувач успішно завантажився в ігрове лобі. |
| Фактичний результат | Після очікування завантаження гравець опинився в ігровому лобі. |

Таблиця 3.4 – Тест 1.4

| Тест | Початок матчу |
| --- | --- |
| Модуль | Ігрове лобі |
| Номер тесту | 1.4 |
| Початковий стан системи | Гравець знаходиться в ігровому лобі та є хостом цього лобі |
| Вхідні дані | - |
| Опис проведення тесту | Користувач наступає на кнопку початку гри, очікує завантаження гри, очікує завантаження інших гравців |
| Очікуваний результат | Після очікування завантаження гравці успішно загрузилися до гри. |
| Фактичний результат | Після очікування завантаження гри, очікування завантаження інших гравців гра успішно почалася. |

Таблиця 3.5 – Тест 1.5

| Тест | Кидання ігрового кубика |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.5 |

Продовження таблиці 3.5

| Початковий стан системи | Гравець знаходиться у грі та дочекався своєї черги робити хід. |
| --- | --- |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на кнопку кидання кубика, очікує переміщення ігровим полем на отриману кількість кроків. |
| Очікуваний результат | Гравцю було продемонстровано отримане значення внаслідок кидання кубиків і фішка, яка належить цьому користувачу виконала переміщення на відповідну кількість кроків. |
| Фактичний результат | Користувач отримав візуальне відображення значення скільки випало внаслідок кидання кубика і фішка була переміщена на задану кількість кроків. |

Таблиця 3.6 – Тест 1.6

| Тест | Придбання власності |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.6 |
| Початковий стан системи | Фішка гравця приземлилася на власність, яка нікому ще не належить. |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на кнопку придбання власності, після цього, у випадку достатню кількість ігрової валюти, оновлюється візуальна складова поля, встановлюючи колір власності у колір відповідного гравця. |
| Очікуваний результат | Оновлюється візуальна складова поля, встановлюючи колір власності у колір відповідного гравця. |

Продовження таблиці 3.6

| Фактичний результат | Гравець успішно придбав власність, колір цього поля був встановлений у колір гравця. |
| --- | --- |

Таблиця 3.7 – Тест 1.7

| Тест | Розбудова монополії |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.7 |
| Початковий стан системи | Гравець має у наявності повністю зібрану хоча б 1 монополію. |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на кнопку покращення власності, після цього, у випадку достатню кількість ігрової валюти, оновлюється візуальна складова поля, збільшуючи кількість зірочок поля і оновлюючи плату за оренду цієї власності. |
| Очікуваний результат | Оновилася кількість зірок поля і збільшилася ціна за оренду даного поля. |
| Фактичний результат | Гравець успішно покращив власність, були здійснені відповідні візуальні зміни і збільшилася плата за оренду перебування інших гравців на цьому полі. |

Таблиця 3.8 – Тест 1.8

| Тест | Продаж власності |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.8 |
| Початковий стан системи | Гравець має хоча б 1 поле у власності поточного гравця. |
| Вхідні дані | - |

Продовження таблиці 3.8

| Опис проведення тесту | Користувач натискає на кнопку продажу власності, після цього оновлюється візуальна складова поля, яка позначає, що поле було закладено, або зменшується кількість зірочок і значення ціни за оренду зменшується, або встановлюється у 0. |
| --- | --- |
| Очікуваний результат | Власність з цього поля була продана, оновилася відповідна візуальна складова і було зменшена ціна за оренду перебування на ньому. |
| Фактичний результат | Гравець успішно продав власність, були здійснені відповідні візуальні зміни і зменшилася плата за оренду перебування інших гравців на цьому полі. |

Таблиця 3.9 – Тест 1.9

| Тест | Створення обміну |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.9 |
| Початковий стан системи | Гравець знаходиться у грі та дочекався своєї черги робити хід. |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на іншого гравця власності, після цього показується панель пропонування обміну, користувач здатен натиснути на поля, яке належить користувачу, з яким здійснює обмін та на свої поля своїх власностей і здатен ввести запропоновану кількість ігрової валюти. |

Продовження таблиці 3.9

| Очікуваний результат | Користувач може взаємодіяти з іншим гравцем у грі, натискати на його власності, що призводить до відкриття панелі пропозицій обміну. Після цього користувач може обрати поля, які належать йому та гравцю, з яким він хоче здійснити обмін, і вказати необхідну суму грошей в грі. |
| --- | --- |
| Фактичний результат | Користувач може взаємодіяти з іншим гравцем у грі, торкаючись його власностей, що призводить до відкриття панелі обміну. Після цього він має можливість вибрати свої власні поля та поля іншого гравця для обміну, вказати необхідну кількість ігрової валюти для запропонованого обміну. |

Таблиця 3.10 – Тест 1.10

| Тест | Прийняття пропозиції обміну |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.10 |
| Початковий стан системи | Гравець отримав пропозицію обміну від іншого гравця. |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на кнопку прийняти обмін, була списана відповідна до обміну кількість балансу, і власники полів власностей були успішно оновлені. |
| Очікуваний результат | Після підтвердження обміну, відбувається зняття відповідної кількості балансу, і власники полів власностей успішно оновлюються, хід повертається до поточного гравця. |

Продовження таблиці 3.10

| Фактичний результат | Після узгодження обміну, сума відповідного балансу сплачується, а власники земельних ділянок успішно оновлюють свою інформацію, після чого ініціатива повертається до поточного учасника гри. |
| --- | --- |

Таблиця 3.11 – Тест 1.11

| Тест | Відхилення пропозиції обміну |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.11 |
| Початковий стан системи | Гравець отримав пропозицію обміну від іншого гравця. |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на кнопку відхилити обмін. |
| Очікуваний результат | Після відхилення обміну відбувається повернення керування до поточного гравця, якому приходить відповідне повідомлення, що інший гравець відмовився від пропозиції. |
| Фактичний результат | Обмін було відхилено, контроль ходу повернувся до поточного гравця, який має виконувати хід, було продемонстровано правильне інформативне повідомлення. |

Таблиця 1.12 – Тест 1.12

| Тест | Ініціалізація процесу добровільної здачі |
| --- | --- |
| Модуль | Ігрова сесія |
| Номер тесту | 1.12 |

Продовження таблиці 1.12

| Початковий стан системи | Гравець знаходиться у грі. |
| --- | --- |
| Вхідні дані | - |
| Опис проведення тесту | Користувач натискає на власну панель і йому демонструється діалогове вікно з пропозицією здатися, на як гравець погоджується. |
| Очікуваний результат | Гравець успішно здається, користувач лишається всієї свої власності, балансу, і залишається як сторонній спостерігач. |
| Фактичний результат | Користувач успішно здався та залишився у грі як сторонній спостерігач без права ходу. |

## Опис контрольного прикладу

У попередньому розділі було протестовано усі можливі взаємодії з ігровим застосунком. Тому у поточному розділі буде описаний контрольний приклад, який може продемонструвати ці всі взаємодії під час справжньої гри.

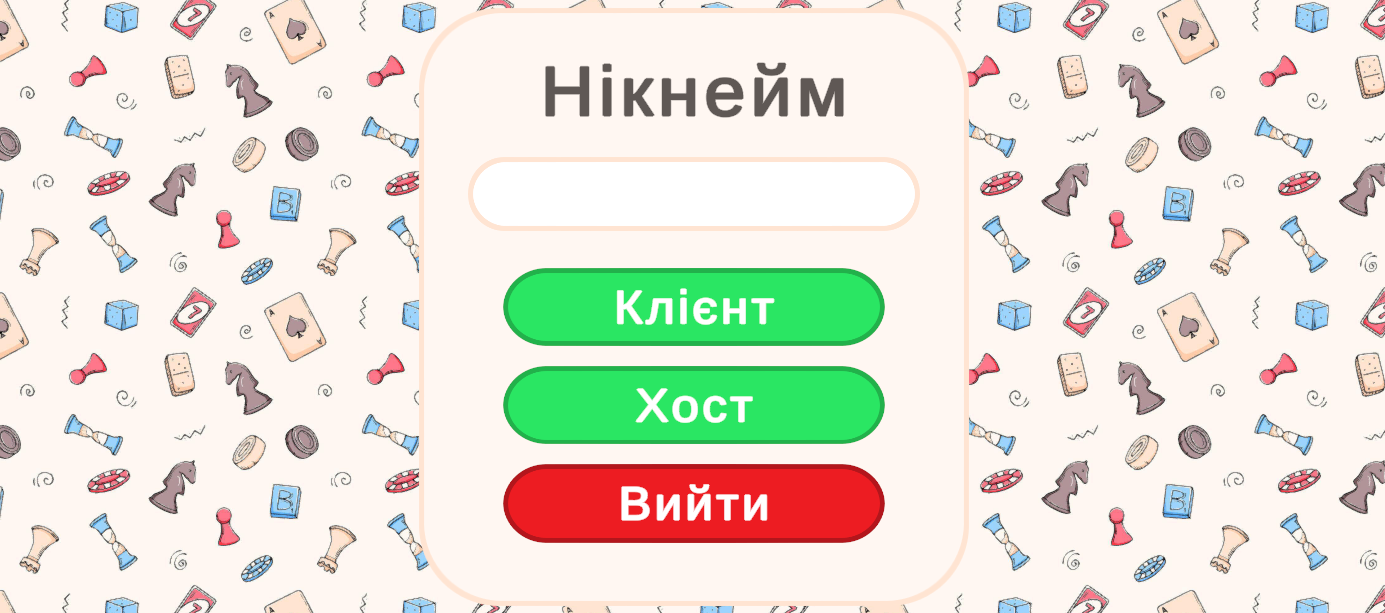


Рисунок 3.4 – Головне меню гри після відкриття

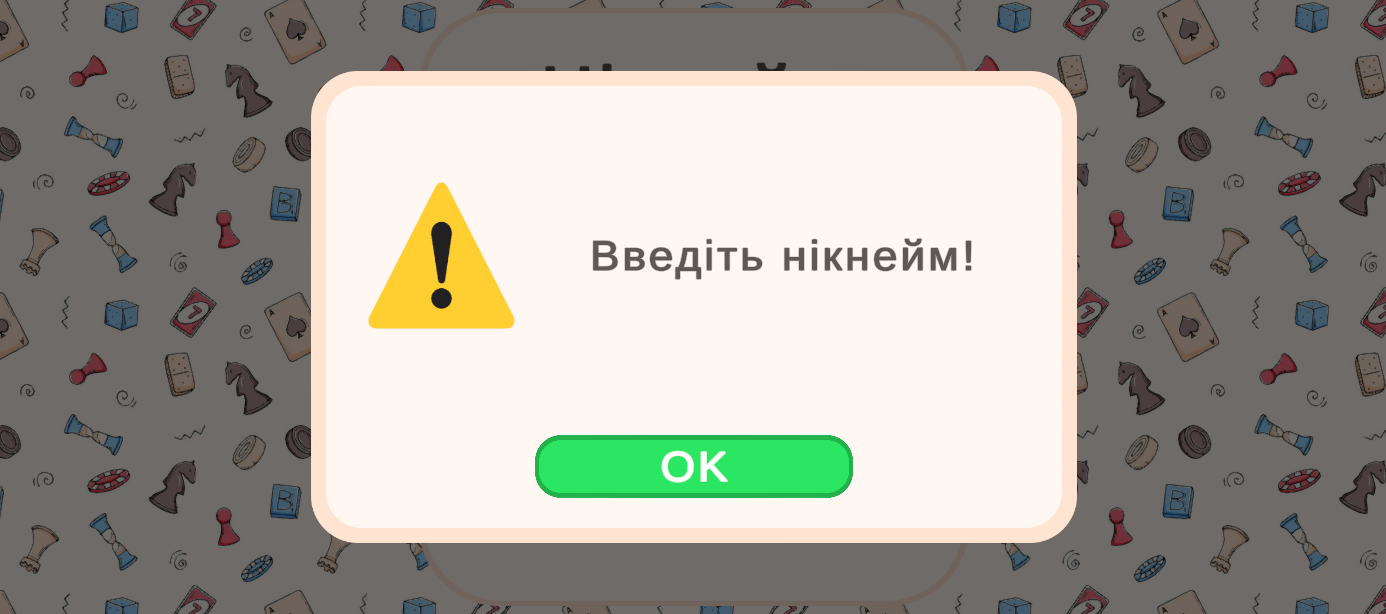


Рисунок 3.5 – Попередження у випадку введення пустого нікнейму

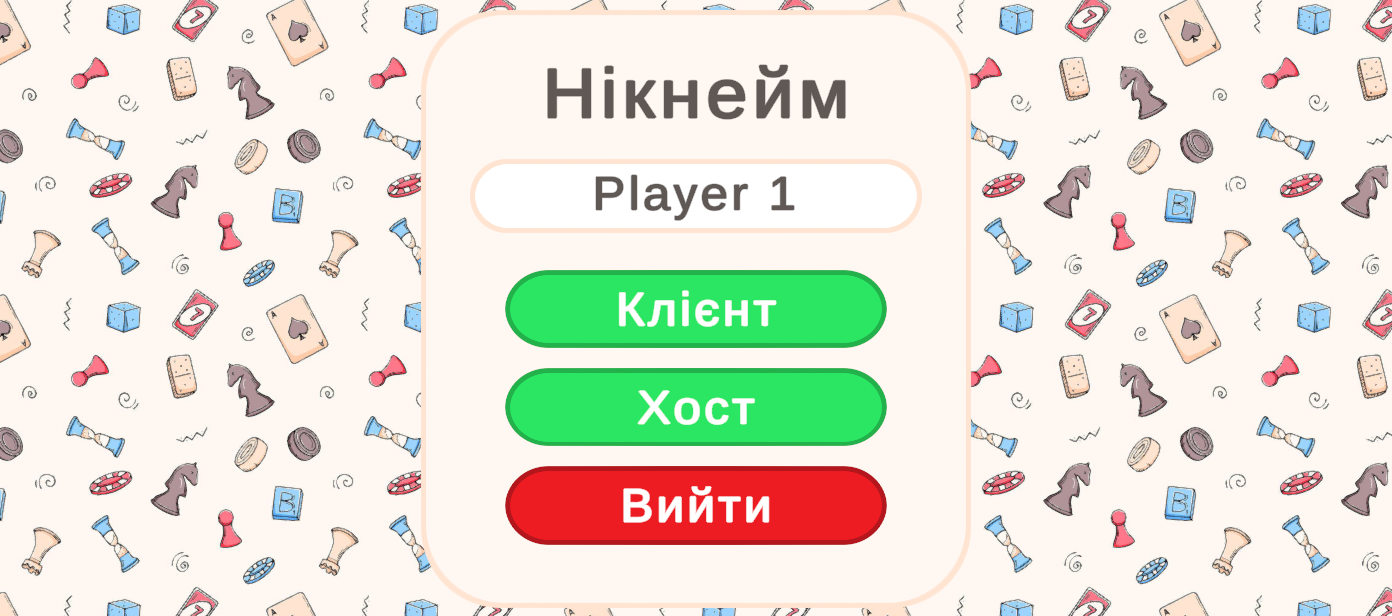


Рисунок 3.5 – Коректно введений нікнейм гравця

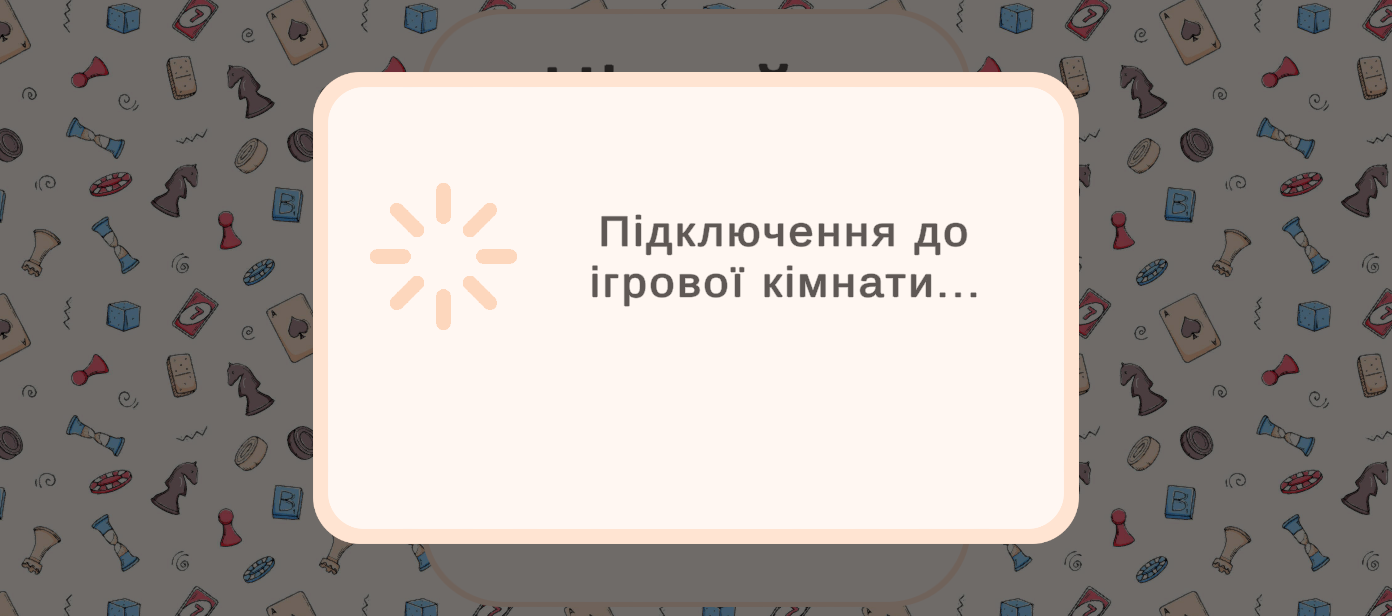


Рисунок 3.6 – Процес підключення новоствореного лобі

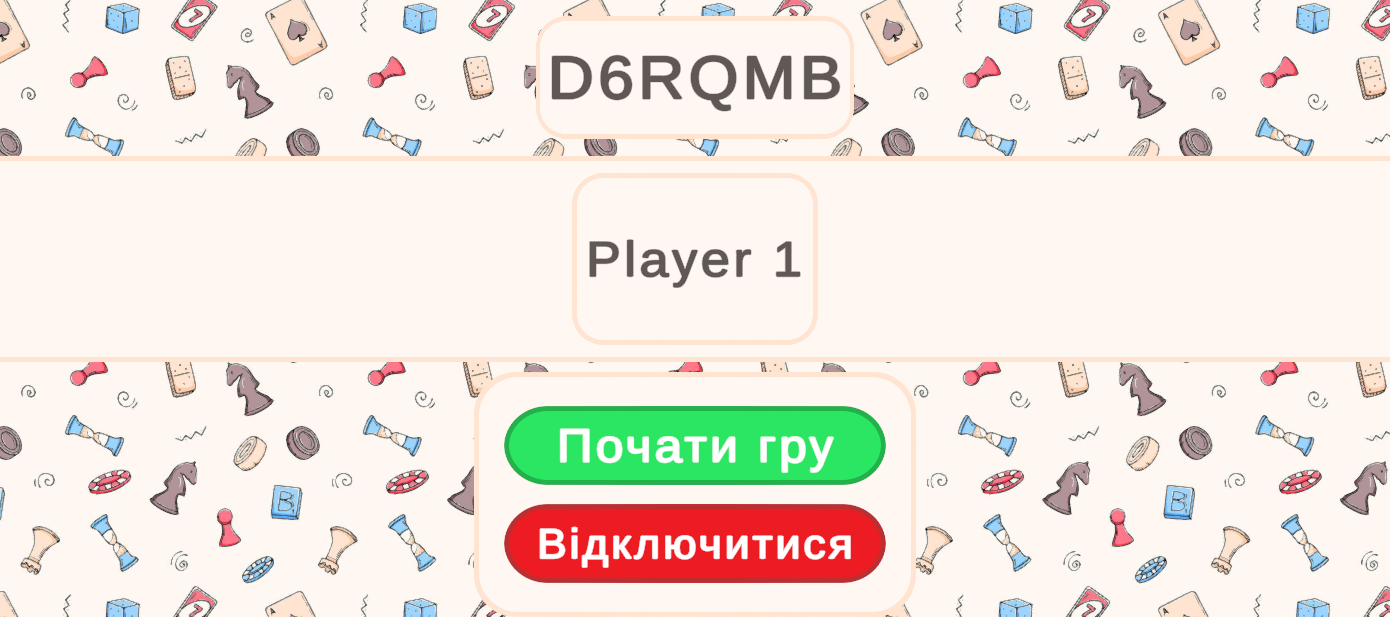


Рисунок 3.7 – Гравець, який перебуває у лобі та є хостом

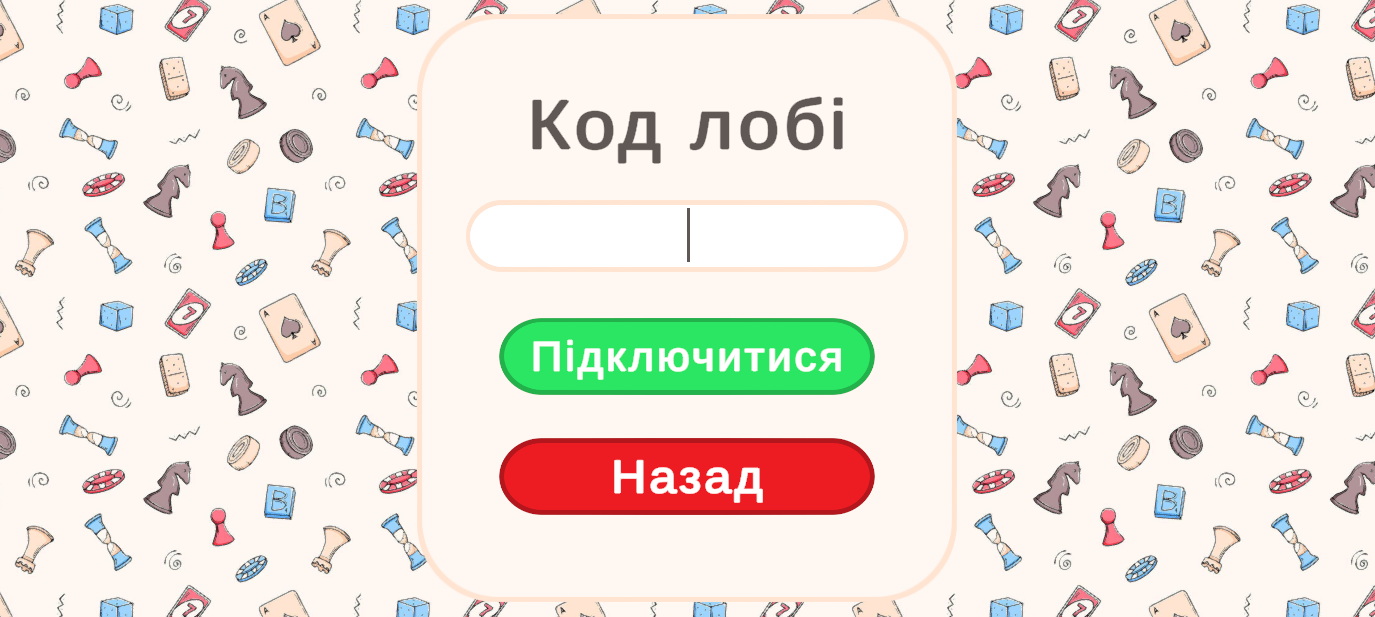


Рисунок 3.8 – Вікно введення коду лобі

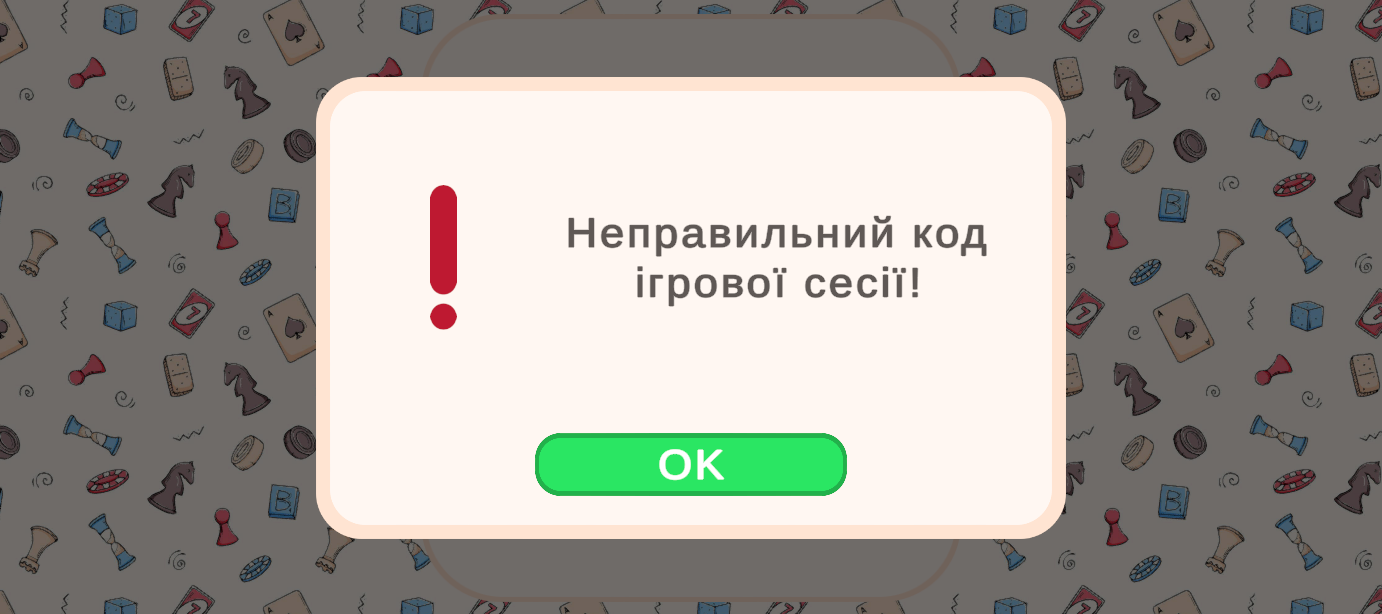


Рисунок 3.9 – Повідомлення у випадку неправильно введеного коду ігрової сесії



Рисунок 3.10 – Обидва гравці знаходяться у лобі



Рисунок 3.11 – Гра завантажилася і гравцю випала черга робити хід

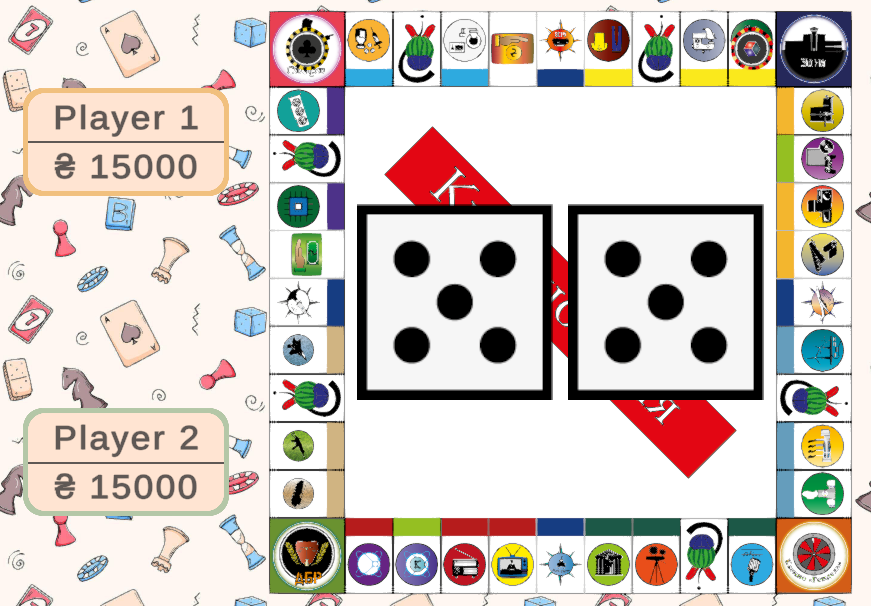


Рисунок 3.12 – Гравець виконує хід, показуються випавші значення кубиків



Рисунок 3.13 – Гравець приземлився на поле власності і йому було запропоновано придбати поле



Рисунок 3.14 – Гравець погодився з пропозицією і власність поля було успішно оновлено, також було списано відповідно суму з балансу поточного користувача



Рисунок 3.15 – Гравець запропонував здійснити обмін з іншим



Рисунок 3.16 – Іншому гравцю прийшла пропозиція обміну

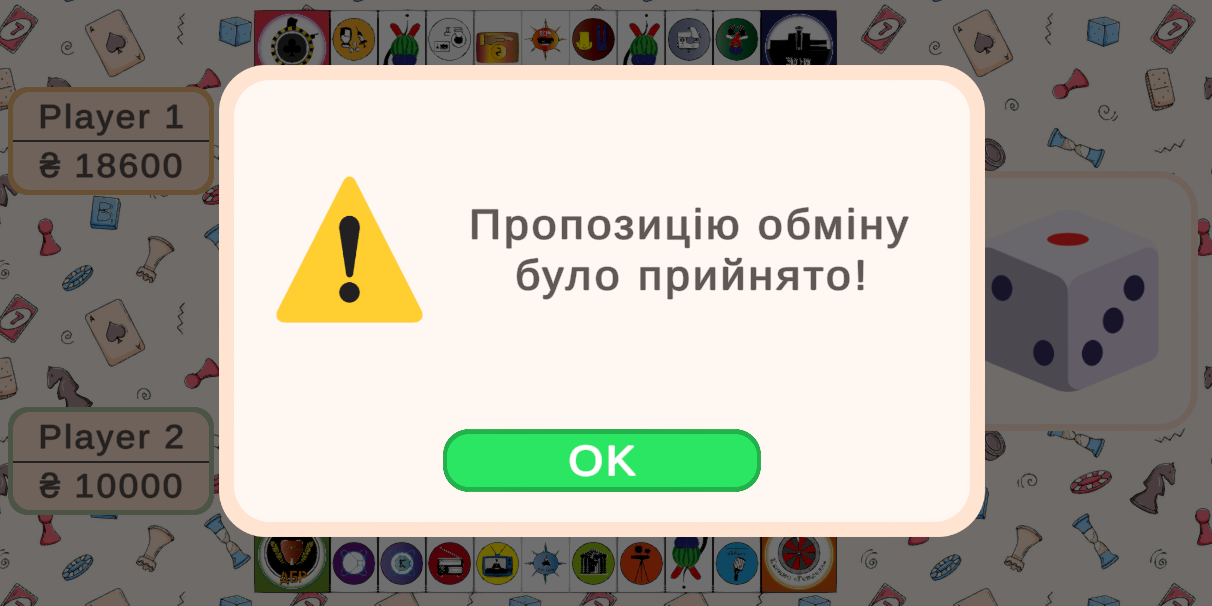


Рисунок 3.17 – Відправнику обміну прийшло повідомлення, що інший користувач успішно прийняв його пропозицію обміну



Рисунок 3.18 – Обмін було успішно здійснено, було оновлено власника поля та баланс гравців



Рисунок 3.19 – Після добровільної здачі іншого гравця поточний користувач переміг у грі

## Висновки до розділу

Аналіз якості програмного забезпечення за певними метриками дозволяє здійснити комплексне оцінювання різних аспектів продукту, що включає в себе якість коду, ефективність використання ресурсів, загальну якість архітектури, якість написаного та відповідність вимогам. Проте варто зазначити, що оцінка якості коду часто є суб'єктивною. Розробники можуть мають різні погляди на те, що вважається гарними підходами до використання певних синтаксичних конструкцій. Оцінки можуть сильно варіюватися в залежності від досвіду, стилю програмування та особистих уподобань. Також якість коду може залежати від контексту і вимог проекту. Те, що вважається гарним рішенням для одного проекту, може бути несхожим на іншому, залежно від специфіки завдань та потреб користувачів. Цей аспект є доволі актуальним для розробки ігор загалом, а особливо для багатокористувацьких проєктів, оскільки у них може використовуватися багато дублюючого коду синхронізації у неткоді, або код для створення редакторів певних компонент.

Тестування програмного забезпечення дозволяє визначити й покращити його якість, включаючи аспекти ефективності, відповідність вимогам та загальну надійність. Під час опису процесу тестування було вирішено використовувати функціональні тести, за допомогою яких відбувається перевірка головних функціональних вимог застосунку, які результати проведення цих тестів потім будуть продемонстровані під час контрольного тестування.

Контрольний приклад допомагає оцінити та підтвердити правильність роботи конкретної функціональності чи компоненту в програмному забезпеченні, забезпечуючи впевненість у його надійності та відповідності встановленим критеріям. Саме тому у розділі був наданий повний опис контрольного прикладу з усіма можливими розгалуженнями та ілюстраціями, які демонструють відповідні взаємодії, які були зазначені під час процесу тестування.

# ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Розгортання програмного забезпечення

Оскільки розробленим програмним забезпеченням є мобільна гра з відкритим вихідним кодом, яка розроблена на платформу Android та під час розробки якої було використано нові бібліотеки та рішення від Unity, тому було прийнято рішення розміщувати її на платформі GitHub. Це було зроблено для того, щоб надати можливість іншим розробникам ознайомитися з вихідним кодом, на основі якого вони зможуть створювати свої рішення. Звичайний користувач має можливість завантажити вже готовий білд за наступним посиланням: https://github.com/NgeNXQ/Monopoly/releases/tag/release.

Процес розгортання для мобільної платформи готового білду стандартний. Для розгортання потрібно виконати наступні кроки:

* Перейти за відповідним посиланням на GitHub-репозиторій у вкладку релізи.
* Завантажити останній реліз у вигляді .apk-файлу.
* Натиснути на завантажений .apk-файл.
* Погодитися на встановлення гри і дочекатися завершення цього процесу.

У випадку успішного процесу встановлення у користувача з’явиться новий застосунок з відповідною зображенням, як на рисунку 4.1.



Рисунок 4.1 – Завантажена та встановлена гра на пристрої користувача

## Підтримка програмного забезпечення

Нові версії програмного забезпечення будуть доступні для завантаження за тим же посиланням, за яким можна знайти і початковий реліз.

Окремо варто зауважити, що у випадку, якщо гравці намагатимуться зіграти з різними версіями програмного забезпечення, то як правило, гравці не зможуть приєднатися до спільної ігрової сесії, але можлива і непередбачувана поведінка.

## Висновки до розділу

У даному розділі було детально висвітлено повний опис покрокового розгортання програмного забезпечення та опис підтримки цього застосунку. Перший підрозділ включає повний опис процесу завантаження та інсталювання гри для мобільної платформи Android.

Додатково був описаний процес підтримки програмного забезпечення. Крім того, важливим аспектом є додавання води у код для забезпечення оптимального функціонування програми. Згідно з цим підходом, користувач здатен буде завантажити нову версію гри за тим же посиланням на тому ж GitHub-репозиторії, забезпечуючи безперебійну роботу та вдосконалену функціональність.

# ВИСНОВКИ

У результаті виконання курсової роботи було створено мобільний ігровий застосунок, який є реалізацією відомої настільної гри «Монополія». Розроблене програмне забезпечення було реалізовано з використанням найсучасніших технічних рішень та практик.

Під час реалізації курсової роботи було спроєктовано мобільну багатокористувацьку гру, яка використовує одноранговий децентралізований тип мережі для побудови взаємодії між користувачами. Основним інструментом для створення гри виступило середовище розробки Unity та офіційне рішення від Unity для створення багатокористувацьких ігор – Netcode For GameObject.

Ключова особливість розробленого проєкту полягає у фактичній унікальності набору технічних інструментів, які використовувалися під час розробки мобільного ігрового застосунку з багатокористувацьким режимом, а також в унікальній реалізації відомої настільної гри з особливими правилами гри, які спеціально адаптовані для більш швидких ігрових сесій, що особливо важливо для мобільних пристроїв.

Під час реалізації проєкту особлива увага приділялася не лише технічним аспектам, але й вдосконаленню геймплейного досвіду користувачів. Важливо відзначити, що створений ігровий застосунок не лише відтворює класичний варіант гри, але і пропонує унікальні правила, створені спеціально для збільшення динаміки ігрового процесу.

Після реалізації застосунку він був протестований на пристроях з різними версіями Android, з різними розмірами екранів щоб переконатися, що додаток коректно відображається на різних пристроях з різними технічними показниками.

У зв'язку з вищезазначеним, вважається доцільним продовження розвитку цього проєкту з метою удосконалення і розширення функціоналу гри, а також її адаптації для інших платформ та операційних систем. Такі кроки сприятимуть подальшому розвитку та вдосконаленню з урахуванням сучасних тенденцій та потреб користувачів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Світова історія ігор [Електронний ресурс] – Режим доступу:

https://www.interaction-design.org/literature/article/a-brief-history-of-games

1. Історія ігор всіх видів [Електронний ресурс] – Режим доступу:  
   https://www.kaloumba.com/en/history-of-games/
2. Історія сучасної ігрової індустрії [Електронний ресурс] – Режим доступу:  
   https://www.redbull.com/us-en/history-of-video-games-design
3. Історія відеоігор 20 століття [Електронний ресурс] – Режим доступу:  
   https://www.history.com/topics/inventions/history-of-video-games
4. MVC-паттерн [Електронний ресурс] – Режим доступу:  
   https://www.geeksforgeeks.org/mvc-design-pattern/
5. ECS-паттерн [Електронний ресурс] – Режим доступу:  
   https://www.guru99.com/entity-component-system.html
6. Мережева архітектура [Електронний ресурс] – Режим доступу:  
   https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-network-architecture.html
7. P2P архітектура мережі [Електронний ресурс] – Режим доступу:  
   https://www.servers.com/news/blog/differences-between-peer-to-peer-and-dedicated-game-server-hosting
8. UDP-протокол [Електронний ресурс] – Режим доступу:  
   https://www.geeksforgeeks.org/user-datagram-protocol-udp/
9. DTLS-протокол [Електронний ресурс] – Режим доступу:  
   https://datatracker.ietf.org/doc/html/rfc6347
10. DTLS загальні положення [Електронний ресурс] – Режим доступу:  
    https://www.ringcentral.com/gb/en/blog/definitions/dtls-datagram-transport-layer-security/
11. Monopoly Board Game [Електронний ресурс] – Режим доступу: https://www.britannica.com/sports/Monopoly-board-game.
12. Unity scripting API [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/ScriptReference/
13. Unity networking API [Електронний ресурс] – Режим доступу: https://docs-multiplayer.unity3d.com/
14. Unity Netcode For GameObjects [Електронний ресурс] – Режим доступу: https://docs-multiplayer.unity3d.com/netcode/current/tutorials/get-started-ngo/
15. Game architecture [Електронний ресурс] – Режим доступу:  
    https://www.studytonight.com/3d-game-engineering-with-unity/game-development-architecture
16. Game programming patterns [Електронний ресурс] – Режим доступу:  
    https://gameprogrammingpatterns.com/architecture-performance-and-games.html
17. Understanding netcode [Електронний ресурс] – Режим доступу:  
    https://www.gamereplays.org/overwatch/portals.php?show=page&name=overwatch-a-guide-to-understanding-netcode
18. Mirror Networking [Електронний ресурс] – Режим доступу:  
    https://mirror-networking.gitbook.io/docs/
19. Fish-Net Networking [Електронний ресурс] – Режим доступу:  
    https://fish-networking.gitbook.io/docs/
20. RPC [Електронний ресурс] – Режим доступу:  
    https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC
21. Unity GUI [Електронний ресурс] – Режим доступу:  
    https://blog.yarsalabs.com/gui-elements-in-unity/
22. Unity розгортання на Android [Електронний ресурс] – Режим доступу:  
    https://developer.android.com/games/engines/unity/build-in-unity
23. Client-Server архітектура [Електронний ресурс] – Режим доступу:  
    https://www.simplilearn.com/what-is-client-server-architecture-article
24. Unity обробник подій [Електронний ресурс] – Режим доступу:  
    https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/EventSystem.html