Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Ігровий застосунок "Монополія" з використанням рушія Unity

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-6
Володимир БЕЗСОНОВ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Галина МАЙСТРЕНКО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Безсонову Володимиру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий застосунок "Монополія" з використанням рушія Unity

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

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи

1. Середовище розробки Unity та Visual Studio 2022
2. Мова програмування C#
3. Операційна система Windows
4. Сервіс для роботи із зображеннями Photopea

4. Перелік питань, що потрібно опрацювати у роботі

1. Аналіз стану проблеми
2. Аналіз існуючих методів вирішення задачі
3. Аналіз та вибір програмних та апаратних засобів реалізації
4. Розробка загальної структури проектованої системи
5. Висновки

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

Слайд-презентація – 10 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз існуючих методів вирішення задачі	27.05.25 – 29.05.25	
2	Вибір програмного забезпечення та	30.05.25 – 31.05.25	
	інструментів розробки		
3	Проєктування архітектури застосунку	01.06.25 – 03.06.2025	
4	Розробка ігрової логіки	04.06.2025 – 06.06.2025	
5	Розробка графічного інтерфейсу користувача	07.06.2025 – 08.06.2025	
6	Реалізація мережевої взаємодії	09.06.2025 – 10.06.2025	
7	Тестування застосунку	11.06.2025 – 12.06.2025	
8	Подання кваліфікаційної роботи керівникам	13.06.25-14.06.25	
	для попереднього захисту		
9	Подання кваліфікаційної роботи на	15.06.25-16.06.25	
	рецензування		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Галина МАЙСТРЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 79 с., 22 рис., 1 дод., 17 джерел.

ОНЛАЙН-ГРА, МОНОПОЛІЯ, UNITY, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, МЕРЕЖЕВА ВЗАЄМОДІЯ, ООП, C#, ТЕСТУВАННЯ, ГЕЙМДИЗАЙН, АНАЛІТИКА.

Метою даної кваліфікаційної роботи є створення ігрового онлайн-застосунку "Монополія" з інтеграцією мультиплеєрного режиму.

Завдання було вирішено за допомогою використання ігрового рушія Unity, мови програмування C# та мережевих технологій NetCode.

Розроблено якісну онлайн-гру, яка надає користувачам можливість брати участь у багатокористувацьких ігрових сесіях по всьому світу, використовуючи сучасний інтерфейс та динамічну взаємодію.

Увесь код проекту було створено з урахуванням принципів SOLID, що забезпечують подальшу легку підтримку додатку та краще розуміння коду.

Програму написано мовою C# з використанням рушія Unity у середовищі програмування Visual Studio.

ABSTRACT

Explanatory note of the qualification work: 79 pages, 22 figures, 1 appendix, 17 sources.

ONLINE GAME, MONOPOLY, UNITY, USER INTERFACE, NETWORK INTERACTION, OOP, C#, TESTING, GAME DESIGN, ANALYTICS.

The purpose of this qualification work is to create an online game application "Monopoly" with the integration of a multiplayer mode.

The task was solved using the Unity game engine, the C# programming language and NetCode network technologies.

A high-quality online game was developed that provides users with the opportunity to participate in multiplayer game sessions around the world, using a modern interface and dynamic interaction.

The entire project code was created taking into account the SOLID principles, which ensure further easy support of the application and better understanding of the code.

The program is written in C# using the Unity engine in the Visual Studio programming environment.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ANDROID – операційна система і платформа для мобільних телефонів

C# – об'єктно-орієнтована мова програмування

IOS – мобільна операційна система

NETCODE – термін, який використовується для опису всієї мережевої логіки

PC – персональний комп’ютер (англ., Personal Computer)

UI – інтерфейс користувача (англ., User Interface)

UNITY – багатоплатформовий рушій для розроблення ігор та додатків

WI-FI – технологія бездротової мережі (англ., Wireless Fidelity)

ВСТУП

Ігри – це основоположний аспект людства протягом усієї історії розвитку суспільства й особистості. Вони були засобом не лише розваги, а й навчання, тобто розвитку стратегічного мислення та комунікації. Настільні ігри відкривають світ, який дозволяє зламати звичні контакти, навчитися новим навичкам, зануривши у розважальне та освітнє середовище.

Граючи в гру, людина має мету, яка залежить від типу. Як правило, коли гравець грає, він прагне виграти, відточити свої навички, кинути виклик самому собі або просто приємно провести час з друзями. Для багатьох гравців настільні ігри – це спосіб розвинути логіку та навички планування, адже більшість ігор передбачають, що гравці роблять продумані ходи, будуючи стратегії та оцінюючи певний потенційний ризик.

Настільні ігри мають багату історію. Вони були винайдені дуже давно і з тих пір зазнали багато змін. Використання ігор як джерела дозвілля та навчання набирало популярності протягом століть, від китайської гри "Го", яка сприймалася як тест на інтелект, до європейських шахів, що стали популярними в Середньовіччі. Це був початок нової ери в іграх, коли з'явилися класичні настільні ігри, такі як "Монополія", які змінили підхід до спільної гри людей, перетворивши економіку та комерційні стратегії на розвагу.

"Монополія" – одна з найкращих ігор, яка, окрім розважального фактору, є чудовим навчальним інструментом та пропонує базові передумови для економічної грамотності, розуміння того, як працює ринок і що таке конкуренція. Вперше вона була запропонована в 1930-х роках і стала однією з найпопулярніших у світі завдяки своїй універсальності та простим правилам, які дозволяють гравцям різного віку легко засвоїти ігровий формат. Мета гри – монополізувати землю, керувати фінансами та укладати розумні угоди, щоб отримати перевагу. Гравці навчаються розподіляти кошти, витрачати їх на поліпшення, такі як будинки чи готелі, а також передбачати майбутні результати своїх думок та дій. Вона дозволяє не лише розвивати економічне

мислення, а й, спробувавши себе в ролі підприємця, інвестора чи менеджера в грі, змушує гравців відчувати відповідальність за свої рішення, адже наслідки не будуть ризиком у реальному житті.

Традиційні настільні ігри переходять у цифровий формат і оживають у сучасному світі ігор за допомогою комп'ютерних технологій. Нова версія гри "Монополія" незважаючи на її класичні механіки, може бути представлена набагато різноманітніше завдяки сучасним візуальним елементам, звуку та інтерактивним функціям. Гравці тепер мають доступ до нових інструментів для дослідження економічної конкуренції та віртуального світу.

Метою даної кваліфікаційної роботи є створення ігрового онлайн застосунку "Монополія". Розробка проєкту буде відбуватися за допомогою ігрового рушія Unity та мови програмування C#.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

"Монополія" – це класична настільна гра, яка добре поєднується зі стратегією, економічною фантастикою та спілкуванням у колективі. Вона була розроблена на початку 20-го століття, а потім стала надзвичайно популярною, зробивши революцію в економічних іграх. Гра надає можливість долучитися до процесів управління фінансами, інвестування та конкуренції на змодельованому ринку, в унікальний спосіб вивчити основні принципи економічної поведінки. Вона є якісним інструментом для побудови релевантних знань та досвіду економічного середовища.

Вивчення цієї теми є актуальним не тільки через довгу історію гри, але й через її здатність адаптуватися до нових технологій та культурних умов. "Монополія" зараз існує в нових формах для будь-якої кількості цифрових платформ, пропонуючи приводи для інтерактивного навчання та стимулюючи стратегічне мислення.

Метою роботи є аналіз ігрової механіки економічної гри "Монополія", розгляд її впливу на формування економічних навичок гравця, а також розробка власної економічної версії з використанням сучасних технологій розробки, а саме платформи Unity та мови програмування C#. В ході роботи вивчаються ключові аспекти, пов'язані з реалізацією ігрових механік, дизайном інтерфейсу користувача та забезпеченням якісного ігрового досвіду.

Під час виконання кваліфікаційної роботи необхідно вирішити задачі:

- створення ігрового застосунку: розробка інтерактивного ігрового середовища, що імітує механіки класичної гри "Монополія", купівлю та продаж нерухомості, управління особистими фінансами та укладання угод між гравцями; розробка та впровадження основних ігрових механік, таких як збір орендної плати, сплата податків та взаємодія з випадковими подіями для динамічності гри;
- дизайн інтерфейсу: створення візуального дизайну застосунку, який буде зручний для користувача відповідно до естетичних стандартів та

тематичних умов гри;

- забезпечення приємного досвіду користувача: впровадження інтуїтивно зрозумілої навігації та плавних анімацій, які полегшують взаємодію гравців з грою, роблячи процес захоплення грою максимально простим та зрозумілим:

- створення основних ігрових сцен, які мають свій унікальний функціонал та надають гравцям зручність та комфорт під час гри (головне меню, лобі та ігрове поле);

- інтеграція можливостей онлайн-гри: функціонал, що дозволяє гравцям змагатися один з одним в режимі реального часу, з'єднання та взаємодія між користувачами з різних локацій;

- процес тестування та балансування гри з метою виявлення та виправлення помилок, а також балансування ігрових механік для забезпечення рівності та інтересу до гри для кожного з гравців.

1.1 Концепція економічної гри "Монополія"

Настільна гра "Монополія" була розроблена, як гра, що моделює економічні процеси, і за десятиліття здобула величезну популярність по всьому світі. З тих пір її видавали багато різних компаній, включно з Hasbro, і вона залишається актуальною і донині, в цифрову еру. Її версії тепер доступні на різних платформах, таких як мобільні пристрої, комп'ютери та ігрові приставки, що розширює базу користувачів та приваблює нове покоління гравців. Зрештою, гра "Монополія" стала символом економічних ігор на дошці, завдяки тому, що в ній представлено управління фінансами та будівництвом власності. Це гра, яка дозволяє гравцям відчути себе підприємцями, які ведуть бізнес і змагаються за вплив на вигаданому ринку. Гра є чудовим інструментом для вивчення економічної поведінки, а механіка гри базується на моделях реальних економічних процесів, таких як купівля та продаж власності, управління ресурсами, інвестиції та ринкова конкуренція.

Таким чином "Монополія" дозволяє гравцям не тільки отримати задоволення, але й глибше зрозуміти ринкові явища через інтерактивну участь.

Завдяки своїм простим, але продуманим правилам, "Монополія" є улюбленою грою серед дітей та дорослих. Попри всю свою азартність, стратегічна глибина гри є достатньою, щоб зробити її привабливою для гравців, які полюбляють серйозні розрахунки та обдумані рішення. Це дозволяє гравцям навчитися корисним навичкам через досвід прийняття рішень у фінансах, плануванню та аналітичному мисленню.

Окрім базових економічних операцій, таких як купівля та продаж майна, у "Монополії" гравці навчаються управлінню бюджетом, розрахунку ризиків та довгостроковому плануванню. Гра також змушує учасників приймати рішення щодо своїх інвестицій, балансування доходів та витрат, що є ще одним ключовим компонентом фінансової грамотності. Оскільки гра поєднує випадковість (кидання кубиків) зі стратегічним плануванням, вона заохочує гравців навчитися балансувати між ризиками та потенційними результатами своїх рішень. Це особливо актуально для розуміння стану фінансових та економічних справ, на які впливає невизначеність, коли гравці починають процес вимірювання прибутків та збитків у динамічних обставинах.

Крім того "Монополія" заохочує комунікативні навички: гравці ведуть переговори з гравцями-суперниками, щоб обмінятися майном, укласти угоди або об'єднатися з іншим гравцем для отримання взаємної вигоди. Ці домовленості та переговори моделюють ділову взаємодію, допомагаючи гравцям розвивати вміння домовлятися, переконувати інших та йти на компроміси. "Монополія" також дозволяє побачити принципи монополізації ринку. Коли гравець володіє всіма дошками одного кольору, це дає йому право стягувати вищу орендну плату з інших гравців, забезпечуючи собі постійний потік доходу. Цей сценарій підкреслює реальну економічну проблему, яку мають монополії, коли вони встановлюють ціни та обмежують конкуренцію для інших гравців на ринку. Це допомагає зрозуміти, чому монополії можуть спричиняти економічні проблеми, наприклад, перешкоджати виходу на ринок

нових конкурентів та змушувати споживачів платити більше за товар чи послугу.

Ризики, пов'язані з інвестуванням, є ще одним важливим аспектом гри. Гравці повинні вирішити, що і куди інвестувати, і за допомогою хорошої стратегії вони можуть значно збільшити свій капітал. Але нерозважливе інвестування або відсутність грошових резервів для сплати податків чи орендної плати може призвести до банкрутства. Це показує необхідність планувати наперед, а також бути обережним.

Гра також ілюструє циклічність економічних процесів: проходячи по полю, гравці повинні сплачувати податки, стикатися зі штрафами та випадковими подіями, які можуть мати як позитивне, так і негативне значення. Це нагадування про нестабільність економічної ситуації, що характеризується припливами та відпливами, в періоди, коли підприємствам і людям доводиться пристосовуватися до умов. Економіка розвивається від буму до спаду, і здатність передбачити цю динаміку, пристосуватися до неї може визначити успіх людини.

1.2 Огляд існуючих аналогів гри "Монополія"

У міру того, як людина переходить у більш цифровий світ, настільні ігри, наприклад "Монополія", реалізовуються у вигляді настільних/серверних та мобільних застосунків. Тепер людина може грати в цифрові версії в будь-який час і в будь-якому місці, в неї є можливість грати з друзями та іншими гравцями по всьому світу, а також вони мають інтерактивні елементи, які додають більше енергії та гострих відчуттів в ігровий процес. Розглянемо найпопулярніші цифрові аналоги гри "Монополія", їх функції та унікальні особливості.

Безумовно, "Monopoly" [1] є однією з найпоширеніших цифрових версій класичної настільної гри для платформ PlayStation, Xbox та PC. У грі представлено 3D-вигляд ігрового поля (рисунок 1.1) разом з анімованими

персонажами, що оживляють гру. Гравці можуть спостерігати за тим, як розвивається місто, коли вони інвестують на своїх ділянках. Завдяки 3D-графіці гра виглядає більш привабливою та цікавою. Крім того, гравець може грати онлайн, змагаючись з друзями або опонентами по всьому світу. Єдиним її недоліком є ціна, і деякі користувачі згадують про деякі технічні недоліки онлайн-режиму.

Рисунок 1.1 – Зовнішній вигляд "Monopoly"

"Монополія" від Marmalade Game Studio [2] – це класична версії гри для користувачів iOS або Android. Це та сама гра з елементами високоякісної 3D-анімації та можливістю змінювати правила та підлаштовувати гру на власний розсуд (рисунок 1.3). Користувачі мають можливість грати проти штучного інтелекту або проти інших користувачів в багатокористувацькому режимі. Однією з переваг є мобільність, яка дозволяє грати в "Монополію" в дорозі. Гравець може грати як в онлайн-режимі, де він грає один на один проти своїх друзів або проти будь-кого іншого по всьому світу. Але в ігровому режимі особливості гри можуть обмежувати досвід ваших безкоштовних користувачів. Також, тривала ігрова сесія на мобільних пристроях може споживати багато заряду акумулятора.

Рисунок 1.2 – Вибір тематики гри в реалізації від Marmalade Game Studio

"Ренто" [3] – відомий безкоштовний цифровий аналог гри "Монополія" для Android, iOS та ПК. Вона має 4 режими гри: онлайн, одиночний, передача пристрою та режим Wi-Fi. В онлайн-режимі гравець може змагатися з гравцями по всьому світу або грати з друзями. Одиночна гра – це гра з роботами (зі штучним інтелектом), які грають так само добре, як і людина. Спільна гра дозволяє насолоджуватися грою з сім'єю на одному пристрої. Режим Wi-Fi означає, що гравець може грати в режим офлайн на кількох пристроях, підключених до домашнього маршрутизатора.

"Rento" (рисунок 1.3) також має свої особливості, такі як різні типи ігрових полів та додаткові параметри налаштування правил. З іншого боку, гра безкоштовна, а це означає, що в неї, ймовірно, гратиме набагато більше людей, а її здатність змінювати правила гри дає гравцям більше можливостей для налаштування гри на свій смак. З іншого боку, деякі розширені опції повинні бути винагородженні додатковими монетами, наприклад, після перегляду відеореклами, тому привабливість гри знижується принаймні для "неоплачуваних" гравців.

Рисунок 1.3 – Зовнішній вигляд гри "Rento"

"Монополія Мільйонер" [4] – ще одна мобільна версія, призначена для відносно швидкої гри. Гра фокусується на сумі (мільйоні), вона схожа на класичний ігровий процес, але дозволяє закінчити гру набагато швидше, ніж оригінал. Інтуїтивно зрозумілий дизайн застосунку вітає гравців, які шукають короткочасних розваг. З іншого боку, через спрощення правил гра трохи втрачає класичну спрямованість "Монополії", яку можливо, цінують традиціоналісти.

Розглянуті версії гри "Монополія" орієнтовані на різні типи гравців: прості адаптації класичних ігор з відстежуваними правилами або більш складні економічні симулятори. Кожна з них має свої плюси та мінуси та пропонує гравцям різні рівні складності, інтерактивності та різноманітності ігрового процесу: повні набори функцій, чудова графіка, безліч ігрових режимів та можливість грати онлайн. Але вони зазвичай мають платну підписку або більш складну механіку, для освоєння якої може знадобитися більше часу. "Ренто" пропонує дешевшу альтернативу, яка робить акцент на швидкому темпі гри та можливості грати з друзями онлайн. Тим не менш, деякі з цих ігор мають мікротранзакції або рекламу, що може негативно вплинути на досвід користувача.

Рисунок 1.4 – Зовнішній вигляд гри "Monopoly Millionaire"

Тим часом, варіант гри в кваліфікаційній роботі є безкоштовним аналогом економічної гри, який за своєю якістю кращий за інші варіанти. Він дозволяє гравцям грати в "Монополію" без необхідності платити чи переглядати нудну рекламу навіть за якісь додаткові функції, а отже, відкриває доступ до гри ширшій аудиторії. Такий доступ приваблює користувачів, які зазвичай не готові платити за цифрові версії класичних ігор, і дозволяє не стягувати плату за ігрові функції всередині гри. Крім того, проєкт у своїй реалізації є простим з точки зору інтерфейсу та легким для сприйняття, у поєднанні з можливістю грати в "Монополію" в її класичному вигляді, не надто ускладнюючи її.

1.3 Проблеми та виклики розробки гри "Монополія"

Механіка гри "Монополія" включає різноманітні дії, що імітують економічні та фінансові процеси: купівлю-продаж нерухомості, стягнення орендної плати, сплату податків та штрафів, інвестиції в інфраструктуру, що

призводить до збільшення прибутку, а також ведення переговорів та укладення угод з іншими учасниками. Усе це має бути враховано під час планування, а також під час їх використання у грі. Купівля та продаж майна, наприклад, має враховувати, чи є у гравця кошти, чи існують обмеження щодо того, хто може володіти майном, і чи застосовуються спеціальні правила. Система оренди, повинна вираховуватися залежить від кількості об’єктів одного кольору, які гравець купив. Усе це потрібно прорахувати та врахувати, щоб створити відчуття справжнього управління фінансами гравця. Все стає ще складнішим щодо угод, які передбачають обмін майном. Для цього потрібна гнучкість, дисциплінованість у виконанні правил та система взаємодії, яка дозволяє укладати угоди в зручній та зрозумілій формі. Ігрова механіка має бути не лише сумісною, щоб запобігти зіткненням, розбіжностям коду, але й водночас підтримувати дотримання правил та справедливості в грі.

Баланс є одним з найважливіших аспектів підтримки інтересу гравців – він дає кожному відчуття рівних можливостей та що всі вони можуть боротися до кінця в змагальному матчі. У "Монополії" рівновагу можна порушити через випадковість (наприклад, кидання кубика) або надто сильну позицію гравця (монополізація об’єктів). Це слід добре продумати, щоб не допустити занадто односторонньої гри. Важливо звернути пильну увагу на структуру доходів та витрат кожного залученого, включаючи такі фактори, як сума, яку платить гравець, а також витрати на штрафи та податки. Випадкові події або додаткова вартість товарів можуть, наприклад, бути розроблені так, щоб дозволити повільним гравцям наздогнати лідера. Крім того, слід розглядати додаткові поля, з яких гравці можуть отримати кошти, навіть якщо вони не займають лідируючі позиції. Також є додатковий рівень напруги та невизначеності, який можна використовувати для формування стратегій, що використовують різні сторони. Балансування, звичайно, є вимогливим процесом, який передбачає навмисне експериментування, включаючи цілеспрямоване використання різноманітних потенційних сценаріїв з метою збереження захоплюючого ігрового процесу в умовах рандомізації.

Дизайн інтерфейсу є важливим аспектом, оскільки він визначає, наскільки легко гравець зможе орієнтуватися в грі, знайти йому потрібну інформацію і те, як він взаємодіє з іншими елементами гри. Оскільки "Монополія" має багато компонентів (гроші, картки власності, картки з інформацією про події та штрафи), інтерфейс має бути максимально зручним та інтуїтивно зрозумілим, щоб гравці не перевантажувалися, маючи при цьому легкий доступ до потрібних параметрів та відповіді на них.

Будь-яка дія, яку виконує гравець у грі, від кидання кубиків до переміщення фішок, має сигналізуватися за допомогою чіткого візуального та звукового зворотного зв’язку, щоб гравець не загубився в моменті та розумів, що відбувається на екрані. Активні зони дошки, де можна виконати дію, можна позначити кольоровими маркерами і також розмістити звукові підказки, щоб гравці отримували відгук про стан дошки. Графічний елемент, повинен відповідати як класичному стилю "Монополії", так і сучасному, щоб залучити ширшу аудиторію.

Анімація переходів та руху фішок, обертання кубиків та статус картки (приховано/показано) має бути плавним та гармонійно вписуватися в інші елементи дизайну. Стильні, але витончені шрифти, яскраві кольори та динамічні ефекти створюють ідеальну атмосферу, поєднуючи старе та нове. Кожне з них створює для гравців плавний та захоплюючий досвід, дозволяючи навіть новачкам мати чітке розуміння того, як орієнтуватися в інтерфейсі.

2 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

2.1 Обґрунтування вибору платформи Unity та мови програмування C#

Однією з найпопулярніших платформ для розробки інтерактивних програм є Unity [5,6]. Її вибір базується на наявності широкого спектру вбудованих інструментів та функцій, які дозволяють створювати проєкти різної складності, найпростішими з яких є 2D-ігри, а найскладнішими – тривимірні симуляції з багатошаровою графікою. Unity дозволяє інтегрувати величезну кількість 3D- та 2D-активів, які можна використовувати для посилення візуального вмісту за допомогою детальних графічних об’єктів та світлових ефектів. Unity добре підходить для створення реалістичних та динамічних сцен завдяки своїй здатності моделювати фізичні властивості об’єктів. Елементи керування анімацією дозволяють користувачам додавати об’єкти без будь-яких зусиль, сприяючи плавним переходам та візуальним ефектам. Також значною перевагою є наявність гнучкої системи управління, завдяки якій над одним проєктом можна працювати як поодинці, так і в команді, де кожен учасник виконує свої задачі за допомогою спільного доступу та системи контролю версій. Unity має можливість розширити свою функціональність за допомогою інших модулів, плагінів та інтеграції з іншими мовами програмування.

Мова програмування C# [7-9] – це надійна об’єктно-орієнтована мова, яка дозволяє легко інтегруватись із платформою Unity. Навіть ті, хто не має попереднього досвіду, можуть ознайомитися з нею за короткий проміжок часу завдяки простому та зрозумілому синтаксису. Крім того, C# підтримує такі функції, як багатопоточність, що дозволяє розробникам оптимізовувати ігри, розподіляючи обчислення між кількома ядрами процесора. Ця мова охоплює об’єктно-орієнтовану та структуровану парадигму, що дозволяє створювати масштабовані системи зі складними зв’язками між об’єктами. В Unity є багато шаблонів та документації для роботи з C#, розробник може швидко адаптувати

мову під різні типи задач, що дозволяє створювати прототипи досить просто.

Unity – це надзвичайно універсальний інструмент, і однією з особливостей, яка робить його таким, є його кросплатформна підтримка, що не має собі рівних. Програми розроблені в Unity можна легко налаштувати для роботи на інших пристроях, можливо створювати мобільні програми, програми для настільних комп'ютерів та консольні ігри (PlayStation, Xbox, Nintendo Switch). Ця кросплатформність гарантує, що розробники можуть охопити найширшу можливу базу користувачів, мінімізуючи вартість адаптації продукту для кожної окремої платформи. Додаткова перевага полягає в тому, що Unity забезпечує спрощений спосіб експорту проєктів для інших платформ. Вона дозволяє легко адаптувати гру, яка працює на різних пристроях без значних змін коду. Портативність стає все більш важливою для ігрового ринку, коли користувачі грають на різноманітних пристроях, а здатність гри працювати незалежно від типу пристрою, не лише дає змогу більшій кількості гравців отримати доступ до продукту, але й дає додаткову свободу щодо розповсюдження.

Як новим так і досвідченим розробникам легко використовувати Unity завдяки його дружньому інтерфейсу користувача, простоті та інтуїтивності. Цій інструмент пропонує візуально редактор, який допомагає керувати сценами, об'єктами, текстурами та звуками, дозволяючи налаштовувати візуальні елементи, не потребуючи значних технічних навичок. Редактор дозволяє змінювати кожну деталь гри, від простих функцій до складної анімації чи динаміки освітлення. Unity має інтегровані інструменти для тестування в реальному часі та налагодження, що надзвичайно спрощує цикл розробки та дозволяє розробникам майже миттєво помічати зміни, коли це необхідно. Таким чином, це підвищує ефективність розробки, оскільки розробники можуть набагато швидше виявляти та виправляти помилки.

Однією з найбільших переваг використання Unity є активна спільнота користувачів та розробників. Процес навчання та розвитку зручний завдяки великий кількості відкритого матеріалу, який включає відеоуроки, форуми,

документацію та блоги. Unity має активний форум та базу знань, і кожен бажаючий зможе знайти відповіді на переважну більшість запитань, які виникають під час розробки. Крім того, Unity Asset Store містить величезну колекцію готових 3D- та 2D-моделей, шейдерів, звукових ефектів та навіть фрагментів коду, що означає, що розробник зможе швидко створити прототип та заощадити час.

2.2 Порівняння платформи Unity з існуючими аналогами

Рушій Unreal Engine [10] надає можливість створювати реалістичні зображення з високою деталізацією завдяки своєму потужному графічному механізму. Крім того, він надає розробникам власні вбудовані інструменти для корегування освітлення, створення текстур та ефектів для складної 3D-ігри із високою графікою. Фотограметричні можливості Unreal Engine, реалістична система освітлення, підтримка трасування променів (освітлення об’єктів для реалістичних ефектів світла й тіні) чудово працюють для ігор класу AAA, кінематографічного моделювання та програм віртуальної та доповненої реальності, де важливий високий рівень деталізації.

Оскільки Unreal Engine є високопродуктивним рушієм, він має набагато вищі вимоги до апаратного забезпечення. Для простих ігор, таких як "Монополія", використання такого потужного механізму є надмірним. Ці ігри зазвичай не потребують потужності такого движка, оскільки навіть середньодеталізована графіка, яку надає Unity, підходить для таких ігор. Це не тільки спрощує розробку гри, але навіть зменшує необхідне обладнання, тобто в гру можна грати і на відносно недорогих пристроях.

Unreal Engine використовує мову C++ [11], яку новачкам важче вивчити, ніж C#, який підтримує Unity. Ядро C++ вимагає від розробників глибоких знань щодо обробки пам’яті та ресурсів, це, безумовно, складніше для студентів та новачків у розробці ігор. Крім того, Unreal Engine має складнішу архітектуру та більше інструментів, орієнтованих на професійних

розробників, що може ускладнити роботу тим, хто вперше вивчає розробку ігор. Unreal Engine надає більше можливостей професіоналам із графікою, аудіо, фізикою та мережними функціями. Однак цей рушій може бути надто складним та дорогим для невеликих проєктів або проєктів з обмеженими ресурсами, які не вимагають реалістичної графіки.

Рушій Godot широко вважається зручним для розробки 2D-ігор, оскільки він містить інструменти спеціально для їх створення; інструменти анімації, підтримки фізики та редактор, створений для найкращої роботи з 2D-графікою. Він підходить для створення казуальних ігор, мобільних застосунків, 2D-ігор тощо. Хоча Godot є дещо простішим та має менше опцій, ніж Unity, він легший для початківців, оскільки їм легше розпочати створення базових ігор. GDScript [12] – це мова програмування Godot, яка розроблена для використання в системі, має синтаксис, подібний до Python, що дозволяє розробникам без офіційного досвіду кодування або тим, кому потрібно швидко створювати прототипи, брати участь у розробці ігор за допомогою движка Godot. Мова GDScript дозволяє писати код так, щоб не потрібно було турбуватися про складні речі, а інтерфейс редактора Godot простий та зосереджений на тому, що потрібно розробнику для 2D та 3D розробки; тому це особливо привабливо для тих, хто починає.

Godot справді має основні 3D-інструменти, але вони значно поступаються тим, що пропонує Unity. Рушій наразі не має передових систем освітлення та тіней, сучасних графічних ефектів або широкого меню 3D-інструментів. Якщо гра покладається на високоякісну 3D-графіку або складніший ігровий процес у 3D-середовищі, Godot може зіткнутися з обмеженнями. Тому для таких ігор, як "Монополія", які потенційно використовуватимуть базові 3D-елементи, Unity буде набагато кращим варіантом.

Що стосується навчальних матеріалів та інструментів, порівняно з Unity та Unreal Engine, Godot має менш розвинену спільноту. Недоліком Godot є те, що він все ще перебуває у стадії інтенсивної розробки, і на даний час доступно

менше ресурсів у відкритому доступі, тому початківцям може бути важче знайти відповіді на запитання чи вирішити технічні проблеми. Крім того, масштабування проєктів є складною задачею через обмежену кількість плагінів та інструментів для обробки даних. Godot є абсолютно безкоштовним, з відкритим вихідним кодом, що забезпечує менші труднощі для студентів та невеликих команд, які хочуть створювати ігри без обмежень щодо ліцензій. Це дає змогу розробникам змінювати двигун (якщо це необхідно для проєкту).

Головна перевага, чому в даній роботі було обрано рушій Unity, полягає в його балансі між потужністю, простотою використання, найсучаснішими інструментами для роботи як з 2D, так і з 3D-графікою, доступністю ресурсів та широким співтовариством. У той час як Unreal Engine надто ресурсоємкий та складний для таких проєктів, як Monopoly, Unity відносно простий для 3D, але також досить багатофункціональний порівняно з Godot.

2.3 Інструментарій для створення графічних елементів

Важливою частиною розробки дизайну є розробка елементів, таких як логотипи, ілюстрації та картки, які безпосередньо впливають на враження користувача про гру та на те, як гра взаємодіє з користувачем. Використовується ряд інструментів, щоб допомогти створити вишуканий зовнішній вигляд, зберігаючи доступність.

Прикладом безкоштовного онлайн-редактора можливо навести Photopea, який пропонує функції, подібні до Adobe Photoshop, без необхідності купувати ліцензію. Також в ході розробки проєкту будуть використовуватися такі ресурси, як Freepik, Vecteezy та Flaticon, щоб створювати векторні ілюстрації, піктограми чи картки. Ці онлайн-ресурси містять безліч готових матеріалів, завдяки яким можна економите час не на розробці, а на отриманні високоякісних компонентів для гри чи іншого проєкту.

Логотип є невід’ємною частиною ідентифікації гри чи бренду, тому його розробці слід приділяти додаткову увагу. Він приймає прості форми, кольори

та шрифти, які найкраще передають суть проєкту. За допомогою Photopea можна працювати з різними шарами, що спрощує процес об’єднання графічних компонентів (тексту, піктограм, абстрактних форм тощо). Крім того, програмне забезпечення підтримує навіть роботу з векторними зображеннями, що буде особливо корисно під час розробки чітких та масштабованих логотипів. Photopea також пропонує функції для створення унікальних логотипів, таких як пензлі, фільтри та стилі шрифтів. Вони дозволять розробити логотип, який легко впізнати та запам’ятати.

Картки потрібно розглядати як функціональні елементи гри, вони повинні не лише служити своїй меті, але й виглядати так, ніби вони належать до візуальної ідентичності гри [14-16] (інтерфейсні картки гри). Можна використовувати Photopea, щоб накласти кілька шарів та текстур, щоб створити більш складні текстури та накладення на текстурованій основі, надаючи кожній картці унікальну індивідуальність. А завдяки роботі з багатошаровими зображеннями можна створювати багатошарові елементи, додавати такі ефекти, як тіні чи відблиски світла, які створюють глибину та деталі в дизайні, покращуючи візуальне сприйняття гри, підвищуючи візуальне задоволення від неї.

Однією з найважливіших переваг Photopea є безкоштовний доступ до інструментів, які зазвичай пропонуються лише в дорогих програмах, таких як Adobe Photoshop. Оскільки цей сервіс знаходиться безпосередньо у браузері, він доступний кожному, і розробнику не потрібно встановлювати програмне забезпечення. Інструмент також підтримує файли PSD, що дозволяє імпортувати та експортувати графічні елементи з інших програм.

Крім того, є можливість зберігати проєкти у форматах PNG, JPEG, GIF та векторних для використання в будь-якому додатковому графічному програмному забезпеченні, яке розробник збирається використовувати надалі або просто в самому проєкті.

Freepik – це один із найпоширеніших сервісів для пошуку безкоштовних графічних ресурсів. Він наповнений векторними та растровими

зображеннями, ілюстраціями, іконками та фотосесіями, які можна використовувати при розробці ігор, сайтів, рекламних матеріалів тощо. Цей інструмент дозволяє швидко знаходити потрібні елементи за допомогою зручного пошуку та фільтрів. Платформа Freepik має широкий спектр векторних ілюстрацій для створення високоякісної графіки, безкоштовні та преміальні матеріали та різні шаблони графічного дизайну для створення логотипів, банерів тощо.

Vecteezy – ще один чудовий варіант пошуку ресурсів векторних зображень, він пропонує як безкоштовні так і платні матеріали. Цей ресурс орієнтований на пошук ілюстрацій, які можна використовувати для створення карток, значків, персонажів або фону для ігор. Векторні зображення дозволять створювати чіткі та масштабовані графічні елементи, простий інтерфейс також спрощує пошук матеріалів за категоріями та темами.

Vecteezy також дозволяє користувачам редагувати зображення та налаштовувати їх відповідно до вимог проєкту, а також можливість експортувати матеріали в робочі проєкти разом із різними популярними графічними програмами. Це дуже корисний ресурс для створення графічних елементів, які використовуються в дизайні інтерфейсу гри, анімаційних карток або інших візуальних елементів.

Flaticon спеціалізується на великій колекції піктограм. На сайті доступні різні варіанти піктограм, тому можливо обрати матеріали на основі бюджету проєкту. Відповідно до власного вебсайту Flaticon має один із найбільших архівів піктограм у світі: понад 5 мільйонів елементів.

Flaticon підтримує різні векторні формати (наприклад, SVG та PNG), що дозволяє змінювати розмір та колір значків без шкоди для якості. Нарешті, він має можливість створювати спеціальні набори піктограм та завантажувати їх у різних форматах. Flaticon також має гарний пошук іконок за типами, стилями та темами, що робить його незамінним інструментом для розробників ігор та мобільних додатків.

Photopea та Freepik – це потужні інструменти з відкритим вихідним

кодом із нескінченними екзистенціальними валідаторами для бізнес-випадків використання. Усі вони пропонують певні переваги: від професійних графічних редакторів до придбання дорогих ліцензій і до безлічі шаблонів, які допомагають швидко створювати різноманітні елементи.

Підводячи підсумок, можна зробити висновок, що з цим потужним набором інструментів у руках, Photopea, Freepik, Vecteezy та Flaticon вдасться створити гарні проєкти з хорошими візуальними елементами будь-якої складності. Усі вони пропонують різні переваги. Наприклад, можна використовувати професійні графічні редактори, не купуючи дорогі ліцензії, отримати доступ до великих колекцій готових матеріалів, що подвоює швидкість розробки.

Photopea дозволяє створювати графічні елементи та керувати ними, працювати з шарами та векторними зображеннями, надаючи потужні пристрої для виконання проєктів будь-якої складності. У Freepik можливо знайти мільйони векторних та растрових зображень, які дозволять створити елементи дизайну. У той час як Vecteezy зосереджується на векторних зображеннях, які можна змінювати відповідно до потреб проєкту, Flaticon пропонує величезну кількість піктограм, які є життєво важливими для створення інтерфейсів гри.

Для пошуку векторних зображень доцільно використовувати кілька сервісів, оскільки вони забезпечені різними матеріалами. Наприклад, у Freepik є багато векторних ілюстрацій, фотографій та шаблонів, яких немає у Vecteezy, що надає спеціально розроблені векторні зображення. У цьому дусі Flaticon є лідером серед бібліотек піктограм, і хоча його ресурси можна використовувати для розробки інтерфейсу, вони не є альтернативою повним векторним ілюстраціям – із Freepik або Vecteezy.

Поєднання цих інструментів може підвищити швидкість розробки, заощадивши час та зменшивши витрати, водночас забезпечуючи високоякісні добре опрацьовані візуальні елементи. Усе це разом дозволяє отримати професійний результат та ширший діапазон можливостей за невелику частку вартості, значно покращуючи якість дизайну, який потрібно створити.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

3.1 Структурна схема проєкту

Структурна схема проєкту [17] складається з наступних декількох основних частин (рисунок 3.1):

- проміжна сцена – це "буферна зона", яка необхідна для деяких функціональних маніпуляцій зі звуком;

- головне меню гри – початковий екран, з якого користувач починає взаємодію з грою. Тут він може відкрити налаштування, подивитися інформацію щодо автора, вийти з додатку та розпочати гру;

- панель "Налаштування гри" надає можливість користувачу змінювати деякі параметри гри;

- інформація щодо автора;

- вихід з застосунку;

- кнопка "Почати гру" знаходиться у головному меню гри, її основною ціллю є переміщення гравця до сцени з лобі;

- лобі – це місце, де користувач повинен самостійно обрати один з двох можливих варіантів. Перший передбачає створення нової ігрової сесії та очікування підключення інших гравців. Другий дозволяє швидко або за допомогою коду доєднатися до вже існуючої сесії;

- функціонал "Приєднатися до гри" реалізований у вигляді декількох кнопок у лобі, що дозволяють підключитися до існуючої сесії;

- вікно "Створити гру" – місто, де гравець самостійно може створити відкриту або закриту (приєднання тільки за кодом) ігрову сесії;

- місце очікування гравців – це сцена де користувачі чекають доки приєднаються усі бажаючі пограти. Тут є можливість вибору кольору, з яким гравець буде грати під час сесії;

- процес гри – це ігрова сцена, де гравці змагаються один з одним.

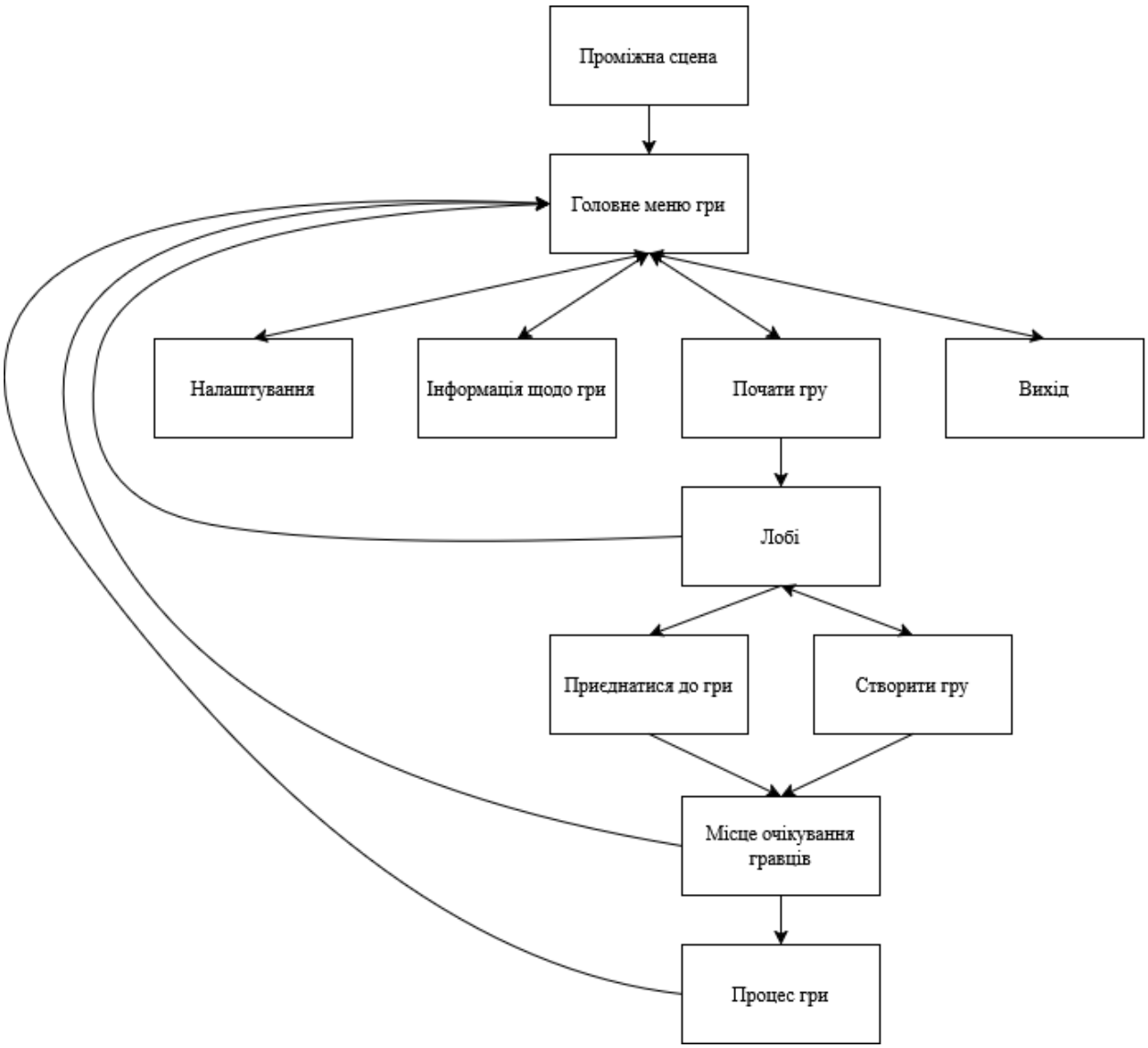

Рисунок 3.1 – Структурна схема проєкту

3.2 Основні сцени у грі

Для коректної роботи гри та для зручної її підтримки надалі, було розроблено 5 основних сцен, кожна з яких відповідає за конкретний ігровий функціонал. Сцени, у налаштуваннях проєкту (рисунок 3.2), розташовані у конкретному порядку за зростанням, за яким вони повинні бути завантажені, також кожній сцені присвоєно конкретний номер. Далі буде більш детально описано кожну сцену.

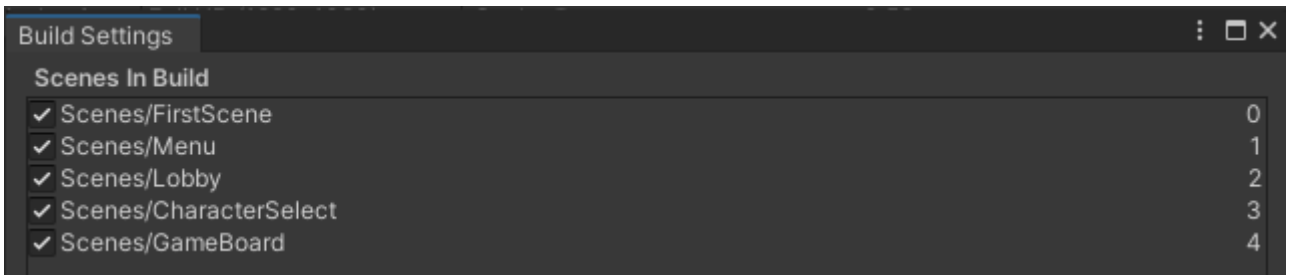

Рисунок 3.2 – Послідовність сцен у налаштуваннях проєкту

3.3 Проміжна сцена

Сцена під назвою "FirstScene" є першою, що завантажується під час запуску гри. Головною її задумкою є проміжний функціонал, який вона виконує. Сцена не містить особливих ігрових елементів на єкрані (рисунок 3.3), але в неї присутній один важливий ігровий елемент, а саме об’єкт під назвою AudioManager.

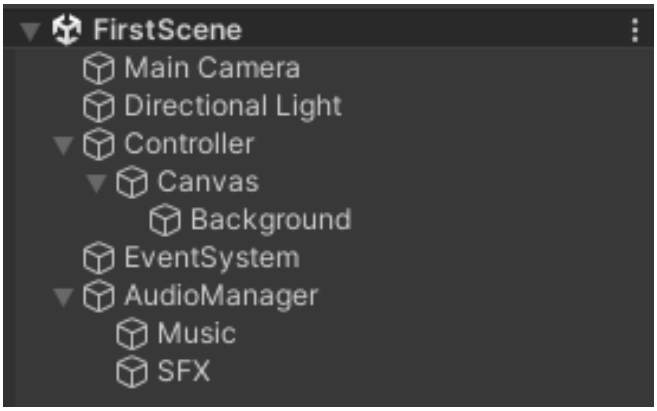

Рисунок 3.3 – Основні елементи першої сцени

AudioManager – це пустий об’єкт, який містить три компоненти: прикріплений скрипт "AudioManager", об’єкт "Music" та "SFX".

Скрипт містить посилання на два об’єкти "Music" та "SFX", а також на список звуків (включно з музикою), які будуть використовуватися під час гри. Також цей скрипт дозволяє об’єкту AudioManager вільно переміщуватися між сценами, що дозволяє користуватися ним впродовж усього терміну життя програми.

Об’єкти "Music" та "SFX" містять компонент AudioSource, який дозволяє відтворювати звук у грі. Але кожен з цих об’єктів містить окремий звуковий вихід. У грі реалізовано AudioMixer (рисунок 3.4) з декількома доріжками, кожна з яких виконує свою функцію. Доріжка "Music" зв’язана з відповідним об’єктом "Music" та є його звуковим виходом, який використовується тільки для відтворення музики. Така сама зв’язка реалізована з "SFX", але тут доріжка відтворює лише звукові ефекти.

Таким чином побудовано дві незалежні системи для відтворення звуку, якими зручно керувати під час гри. Але цього замало, тому що наразі звук лише відтворюється але його ще потрібно слухати.

Для вирішення цієї задачі у компонента MainCamera є відповідний елемент AudioListener, який дозволяє слухати аудіо у грі. Відомо, що кожна сцена містить компонент MainCamera, а об’єкт AudioManager не знищується при завантаженні нової сцени та перебуває у вільному доступі. Отже у підсумку виходить, що кожна сцена має слухача (MainCamera) та відтворювач аудіо у вигляді об’єкта AudioManager, що гарантує роботу аудіо у будь-якій сцені.

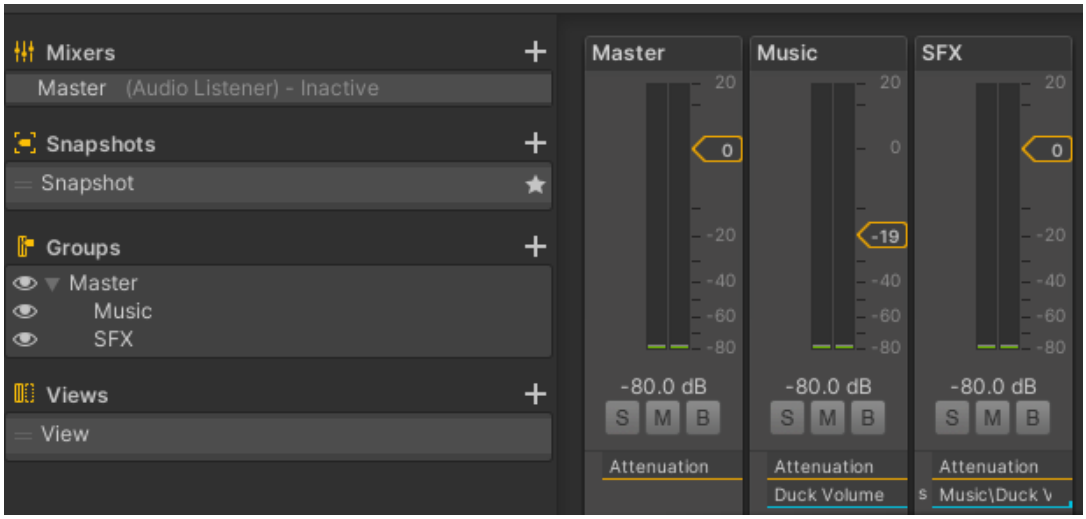

Рисунок 3.4 – Конфігурація Audio Mixer у Unity

Однієї з причин чому AudioManager було вирішено реалізувати саме у цій сцені, це тому що користувач більше ніколи до неї не повернеться. Справа

у тому, що якби цей функціонал був реалізований наприклад, у наступній сцені "Menu", то у разі повернення користувача до неї з будь-якої іншої сцени, елемент AudioManager дублювався би ще раз, і далі б з’являлися проблеми зі звуком. Тому дану проблему, було вирішено створенням проміжної сцени, яка завантажується лише один раз перед головним меню гри, і до якої користувач більше ніколи не повернеться.

3.4 Головне меню

За відображення головного меню у грі, відповідає сцена під назвою "Menu", яка містить багато цікавих елементів (рисунок 3.5), основні з яких: кнопки, логотип та зображення для заднього фону. Під час завантаження гри ця сцена додається другою, після попередньої "FirstScene". Сцена побудована у 2-вимірному просторі, адже головна її мета – це взаємодія користувача з кнопками для подальших дій.

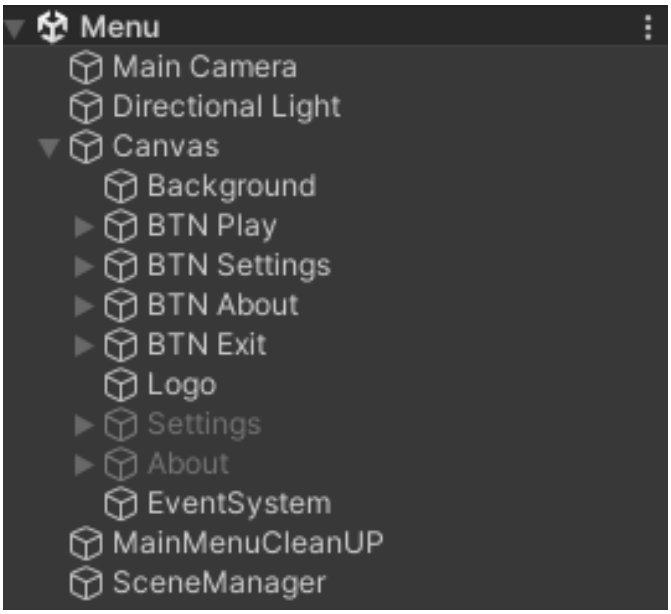

Рисунок 3.5 – Основні елементи сцени "Menu"

Перший та головний елемент цієї сцени – це камера (Main Camera), через яку користувач має змогу переглядати вміст сцени. Камера тісно пов'язана з

наступним елементом "Canvas", який містить усі необхідні UI елементи для роботи, а саме: кнопки, зображення, текстові поля, поля для введення, слайдери та багато чого іншого. Розглянемо роботу елементу "Canvas". Йому потрібно заздалегідь задати деякі параметри, наприклад, розмір (у нашому випадку це 1920*1080), тип відображення та величину "match", яка відповідає за режим масштабування під різні екрани.

Також прикріплено скрипт "Menu", який відповідає за коректну взаємодію користувача з кнопками (рисунок 3.6).

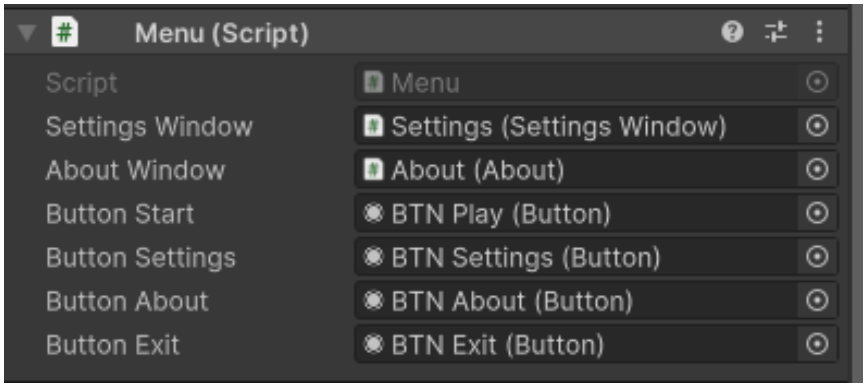

Рисунок 3.6 – Скрипт "Menu"

У Canvas загалом існує три типи відображення. Перший "Screen Space – Overlay" використовується для рендерингу напряму на екран, поверх сцени, тобто не залежить від камери. Другий режим "Screen Space – Camera" рендерить через конкретну камеру, яка встановлюється у редакторі (у цій сцені використовується саме цей тип). І третій режим "World Space", дозволяє Canvas існувати як звичайний 3D-об’єкт, тобто відображається у просторі як будь-який інший об’єкт.

3.4.1 Клас "Menu"

Скрипт "Menu" розроблено для забезпечення взаємодії між користувачем та сценою "Menu". Через нього користувач може зробити наступні дії: розпочати гру, відкрити меню налаштувань, подивитися

інформацію про автора та вийти з гри. Скрипт "Menu" має декілька основних полів, які містять посилання на кнопки та об’єкти типу "GameObject".

Поле "Settings Window" – це посилання на невелике вікно, яке відкривається під час натискання на кнопку "BTN Settings". За роботу меню налаштувань відповідає скрипт "Settings Window", який буде детально розглянутий пізніше. Поле "About Window" – це посилання на віконце, яке відкривається під час натискання на кнопку "BTN About". Під час натискання на кнопку "BTN Exit" відбувається вихід з гри, а під час натискання на кнопку "BTN Play" – відриття наступної сцени.

Якщо розглянути метод "Start" цього класу (лістинг 3.1) можна побачити використання інструменту "PlayerPrefs". Це вбудований інструмент Unity, який відповідає за збереження деяких простих типів даних між ігровими сесіями. У даному прикладі він використовується для запам’ятовування гучності музики та звукових ефектів, які користувач визначив у меню налаштувань. Це робить ігровий досвіду гравця більше приємним та комфортним.

Лістинг 3.1 – Метод "Start"

```
private void Start()
{    if (PlayerPrefs.HasKey("MusicVolumeKey") &&
PlayerPrefs.HasKey("SFXVolumeKey"))
    {        settingsWindow.LoadVolume();
    }
    else
    {
        settingsWindow.SetValueMusic();
        settingsWindow.SetValueSFX();
    }
}
```

Також у цьому класі прописано, що під час настискання на кнопку, викликається необхідний метод, який супроводжується відповідним звуковим ефектом. Це реалізовано за допомогою викликання методу "PlaySFX", у об'єкта AudioManager, з вхідним параметром типу int, який вказує на необхідний звуковий ефект для виклику у масиві звуків.

3.4.2 Клас "SettingsWindow"

Скрипт "SettingsWindow" накладений на об’єкт "Settings" та призначений для взаємодії користувача з меню налаштувань. Меню налаштувань містить декілька параметрів, які користувач може змінити, а саме: гучність музики та звукових ефектів, вхід/вихід з повноекранного режиму та роздільну здатність додатку.

Важлива деталь, що напочатку завантаження сцени "Menu" цей об'єкт не є видимим для користувача, тому у редакторі він є сірого кольору (рисунок 3.7). Об'єкт містить багато UI елементів: зображення для заднього фону вікна, кнопку вихід, два слайдери для регулювання гучності, меню з випадаючими елементами (для вибору роздільної здатності), тексти для позначення інструментів та кнопку переходу у повноекранний режим. Скрипт має посилання на майже усі ці об’єкти, для подальшої роботи з ними.

Рисунок 3.7 – Основні елементи об’єкту "Settings"

Для розуміння роботи меню з вибором роздільної здатності та кнопки переходу у повноекранний режим або навпаки потрібно зрозуміти як формується список для меню (лістинг 3.2) на початку роботи скрипта.

Спочатку створюється пустий список "options" з типом string та змінна "result". Також у класі вже присутній пустий список "SelectedResolutionList" з типом "Resolution", він потрібен для подальшого звернення та встановлення конкретного значення саме з нього. У масив "AllResolutions" поміщаються усі

роздільні здатності, які доступні на цьому пристрої. А далі за допомогою циклу foreach масив перебирається та у змінну "result" записуються у правильному вихідному форматі значення. Далі виконується перевірка, якщо у списку "options" такого елементу ще немає, то він додається, якщо навпаки, то не додається. Також значення без форматування додаються до списку "SelectedResolutionList". Коли цикл завершився готові значення списку "options" додаються як елементи до меню з випадачаючим списком.

Отже виходить два готові списки для роботи. Перший "options" містить значення роздільної здатності у правильному форматуванні, для відображення користувачу, а другий "SelectedResolutionList" використовується для подальших маніпуляцій зі зміною роздільної здатності та переходу у повноекранний режим та навпаки.

Лістинг 3.2 – Метод "InitializeListResolutions"

```
public void InitializeListResolutions()
{
    List<string> options = new List<string>();
    string result;
    AllResolutions = Screen.resolutions;
    foreach (Resolution resolution in AllResolutions)
    {
        result = resolution.height.ToString() + " x " +
resolution.width.ToString();
        if (!options.Contains(result))
        {
            options.Add(result);
            SelectedResolutionList.Add(resolution);
        }
    }
    resolutionDropDown.AddOptions(options);
}
```

Далі розглянемо роботу повноекранного режиму (лістинг 3.3). Спочатку перш за все виконується звуковий супровід, далі перевіряється чи була натиснута кнопка переходу у повноекранний режим. Якщо так, то програма запам’ятовує поточні розміри екрану і далі зі списку "SelectedResolutionList", який був заповнений під час старту, вибирає останнє значення, так як воно є

максимальним, яке доступне на цьому пристрої. Якщо ж ні, то під час входу у повноєкранний режим програма запам’ятала останні розміри, тому вона повертає розміри до попердньо записаних значень.

Лістинг 3.3 – Метод "ChangeFullScreenMode"

```
public void ChangeFullScreenMode()
{
    AudioManager.Instance.PlaySFX(1);
    bool IsFullScreen = ToggleFullScreen.isOn;
    if (IsFullScreen)
    {
        RememberLastResolutions();
Screen.SetResolution(SelectedResolutionList[SelectedResolutionLi
st.Count - 1].width,
            SelectedResolutionList[SelectedResolutionList.Count
- 1].height, IsFullScreen);
    }
    else
    {
        Screen.SetResolution(LastSelectedResolutionWidth,
LastSelectedResolutionHeight, IsFullScreen);
    }}
```

Функціонал слайдерів, які відповідають за гучність музики та звукових ефектів дуже простий. Скрипт має доступ до них, тому легко отримує значення з кожного, а далі просто конвертуючи передає їх до відповідних "Music" та "SFX" AudioMixer. Також у кінці записується значення зі слайдерів до PlayerPrefs.

3.4.3 Клас "About"

Даний скрипт прикріплений до об’єкту "About" (рисунок 3.8) та відповідає за відображення віконця з інформацією щодо автора. Як і попередній об’єкт "Setitings", цей об’єкт також напочатку завантаження сцени невидимий для користувача і з’являється лише при натисканні на відповідну кнопку.

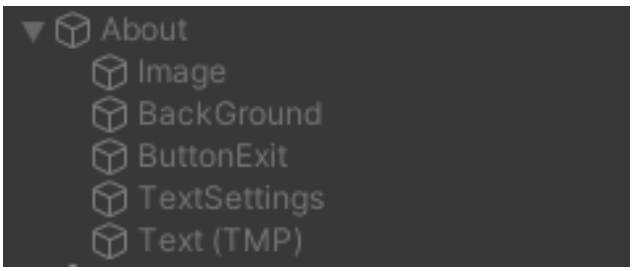

Рисунок 3.8 – Основні елементи об’єкту "About"

Загалом скрипт дуже простий, він має звичайні UI елементи (одну кнопку, декілька текстових полів та зображення) та відповідає лише за відкриття та закриття вікна.

3.4.4 Клас "SceneManager"

Для повного розуміння роботи кнопки "BTN Play" необхідно розглянути елемент у сцені "SceneManager". Це пустий 3D-об’єкт, який містить скрипт з такою ж назвою "SceneManager".

Головне завдання цього скрипта, це керування сценами та переміщення об’єкта, на який накладений скрипт, між ними. Для цього було створено пустий 3D-об’єкт та повідомлено компілятору, за допомогою метода "DontDestroyOnLoad", що при завантаженні нової сцени, цей об’єкт не потрібно знищувати, а так як цей скрипт накладний на 3D-об’єкт, він буде постійно переміщуватися від сцени до сцени.

Також у цьому класі реалізовано паттерн "Singleton", який гарантує що є тільки один екземпляр цього класу. Це надає можливість, за допомогою посилання через статичну змінну "Instance", абсолютно з будь-якого місця програми звернутися до цього класу. Це забезпечує постійну присутність та доступ до фунціоналу керування незалежно від сцени.

Клас містить на перший погляд два однакові методи "PlayScene" та "PlaySceneNetwork" (лістинг 3.4). Але ключова відмінність між ними у тому, що перший метод виконується локально на комп’ютері користувача та забезпечує його переміщення між сценами, яке не потребує мережної

взаємодії з іншими гравцями, наприклад, відкриття лобі або вихід до головного меню, тощо. Другий метод використовується для синхронізованого переміщення гравців між сценами за допомогою мережних сервісів NetCode. Наприклад, коли всі гравці підключилися та готові розпочати гру, виклається цей мережний метод для синхронізованого переміщення гравців до сцени з грою.

Лістинг 3.4 – Скрипт "SceneManager"

```
public enum Scenes
{
    Menu,
    GameBoard,
    Lobby,
    CharacterSelect
};
public class SceneManager : MonoBehaviour
{
    public static SceneManager Instance;
    private void Start()
    {
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }
    public static void PlayScene(Scenes sceneEnum)
    {
UnityEngine.SceneManagement.SceneManager.LoadScene(sceneEnum.ToS
tring());
    }
    public static void PlaySceneNetwork(Scenes sceneEnum)
    {
NetworkManager.Singleton.SceneManager.LoadScene(sceneEnum.ToStri
ng(), LoadSceneMode.Single);
    }}
```

Також варто звернути увагу на синтаксис цих методів, а саме на вхідний параметр, який вони приймають. Перед цим класом оголошено публічний перелік enum "Scenes", який містить назви усіх сцен гри, окрім самої першої "FirstScene", адже до неї гравець ніколи не повертається. І таким чином, коли потрібно викликати наприклад, метод "PlayScene", вхідним параметром до нього передається проста та наочна конструкція (Scenes.Menu), що робить використання класу та методу зручним та зрозумілим.

3.5 Сцена "Лобі"

Третьою сценою, яка завантажується у грі є "Lobby", куди користувач потрапляє натисканням кнопки "Play" у попередній сцені "Menu". Данна сцена містить багато різних об’єктів (рисунок 3.9) та призначена для вирішення задачі по направленню гравців далі. Тут користувачі самостійно обирають, що вони хочуть робити: власноруч створити сесію або приєднатися до вже існуючої.

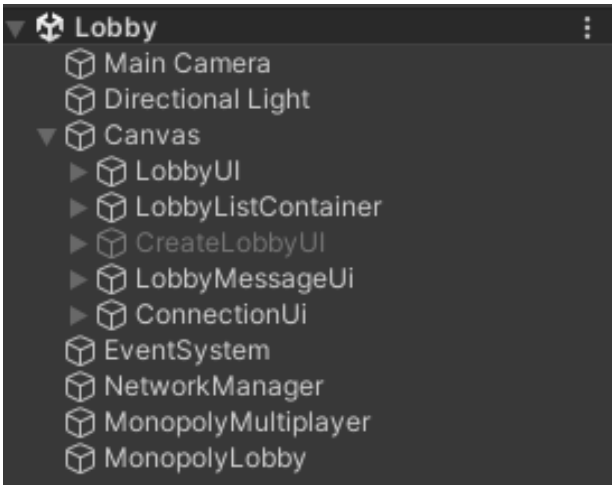

Рисунок 3.9 – Основні елементи сцени "Lobby"

Для розуміння роботи сцени з лобі, яка починає плавно торкатися теми мережної взаємодії користувачів, доцільно розглянути три критично важливі для розуміння об’єкти, а саме: "MonopolyMultiplayer", "MonopolyLobby" та "NetworkManager". Три ці об’єкти при зміні сцени на наступну, не будуть знищуватися, адже увесь функціонал мережної взаємодії знаходиться саме у них, тому вони будуть переходити від сцени до сцени.

3.5.1 Об’єкт "NetworkManager"

NetworkManager – це спеціальний компонент у Unity, який використовується для керування усіма процесами, що торкаються мережної

взаємодії. Він містить багато налаштувань та відповідає за багато процесів: обробка підключень та відключень клієнтів, синхронізація гравців та сцен але найголовніше, що він може робити – це запускати сервер та клієнт. Також є деякі особливості його використання, наприклад, при спавні гравців він повинен заздалегідь знати, як виглядає гравець, тобто мати посилання на префаб гравця.

NetworkManager має 3 режими підключення до мережі. Перший "Start Host" означає, що гравець одночасно є і сервером, і клієнтом, тобто уся обробка даних у грі, яка стосується серверної частини, буде відбуватися саме у цього гравця, і паралельно він сам може приймати участь у ній. Другий "Start Server" означає, що створюється тільки сервер без локального гравця. Цей режим використовується, коли сервер запускається окремо, наприклад, на віддаленій машині. Третій режим "Start Client" означає, що гравець під’єднується як клієнт до серверу і лише отримує від нього дані. У даному проєкті будуть використовуватися перший та третій тип підключення.

Також варто зазначити, що разом з NetworkManager використовується скрипт "UnityTransport". Це низькорівневий транспортний шар, по якому передаються дані від сервера до клієнта і навпаки. Скрипт не керує самими об’єктами, як це робить NetworkManager, він лише передає інформацію. Тобто використання цих двох компонентів разом створює гарну основу для побудови чіткої та стабільної мережної взаємодії у грі.

3.5.2 Об’єкт "MonopolyMultiplayer"

Об’єкт "MonopolyMultiplayer" містить накладений на нього скрипт "MonopolyMultiplayer", який відповідає за правильну функціональну взаємодію з NetworkManager, а саме: коректне підключення клієнтів до серверу, правильну обробку критичних ситуацій (клієнт від’єднався або сервер припинив свою роботу), створення списку підключених клієнтів та заповнення необхідних даних про клієнта, правильне від’єднання сервером

клієнта та багато інших корисних методів.

Для розуміння базових речей у цьому скрипті, необхідно розглянути для початку два простих методи "StartHost" та "StartClient".

Перший метод "StartHost" відповідає за створення гри у режимі хосту (лістинг 3.5). Спочатку створюються підписки на події, що пов'язані з клієнтом, а саме: підтвердження підключення клієнта до серверу "NetworkManager_ConnectionApprovalCallback", клієнт успішно під’єднався до серверу "NetworkManager_ServerConnectedCallback" та клієнт від’єднався від серверу "NetworkManager_Server_OnClientDisconnectCallback". А далі за допомогою NetworkManager створюється хост. Важливо зауважити, що усі ці події опрацьовуються на стороні серверу.

Лістинг 3.5 – Метод "StartHost" скрипта "MonopolyMultiplayer"

```
public void StartHost()
{
    NetworkManager.Singleton.ConnectionApprovalCallback +=
NetworkManager_ConnectionApprovalCallback;
    NetworkManager.Singleton.OnClientConnectedCallback +=
NetworkManager_ServerConnectedCallback;
    NetworkManager.Singleton.OnClientDisconnectCallback +=
NetworkManager_Server_OnClientDisconnectCallback;
    NetworkManager.Singleton.StartHost();
}
```

Головною метою події "NetworkManager_ConnectionApprovalCallback" (лістинг 3.6) є перевірка на можливість доєднання клієнта до серверу, це виконується за допомогою двох наступних перевірок. Перша перевірка спрямована на визначення початку гри. Сервер робить запит на поточно сцену, це має бути сцена під назвою "CharacterSelect" адже саме у ній гравці збираються та підтверджують свою готовність до початку гри. Тобто якщо гравці вже у іншій сцені, це каже про те, що гра вже почалася і гравцю, який хоче під’єднатися буде відмовлено. Друга перевірка спрямована на визначення кількості вже підключених клієнтів. У цьому проєкті встановлено ліміт на максимальну кількість гравців в одній грі. Цей ліміт складає 6 гравців на одну

ігрову сесію. Тобто якщо сесія вже набрала максимальну кількість гравців то клієнт, який хоче підключитися отримає відмову.

Лістинг 3.6 – Подія "NetworkManager_ConnectionApprovalCallback"

```
private void
NetworkManager_ConnectionApprovalCallback(NetworkManager.Connect
ionApprovalRequest connectionApprovalRequest,
NetworkManager.ConnectionApprovalResponse
connectionApprovalResponse)
{
    if
(UnityEngine.SceneManagement.SceneManager.GetActiveScene().name
!= Scenes.CharacterSelect.ToString())
    {
        connectionApprovalResponse.Approved = false;
        connectionApprovalResponse.Reason = "Game has already
started";
        return;
    }
    if (NetworkManager.Singleton.ConnectedClientsList.Count >=
MaxPlayersNumber)
    {
        connectionApprovalResponse.Approved = false;
        connectionApprovalResponse.Reason = "Game is full";
        return;
    }
    connectionApprovalResponse.Approved = true;
}
```

Ця подія дуже корисна, адже запобігає винекненню непередбачуваних помилок та непорозумінь у майбутньому.

Наступна подія "NetworkManager_ServerConnectedCallback" (лістинг 3.7) спрацьовує при успішному підключенню клієнта до сесії. Подія потрібна для запису деяких значень клієнта до списку "playerDataNetworkList".

Важливо розуміти, що список "playerDataNetworkList" – це мережний список, який доступний як клієнтам (для читання), так і серверу (для запису значень). Тип, який використовується у цьому списку, структура "PlayerData", яка зберігає інформацію про гравця та через додавання певних інтерфейсів (INetworkSerializable) має можливість підтримувати мережну синхронізацію.

Лістинг 3.7 – Подія "NetworkManager_ServerConnectedCallback"

```
private void NetworkManager_ServerConnectedCallback(ulong obj)
{
    playerDataNetworkList.Add(new PlayerData
    {
        clientId = obj,
        colorId = GetFirstUnusedColorId(),
    });
    SetPlayerNameServerRpc(GetPlayerName());

SetPlayerIdServerRpc(AuthenticationService.Instance.PlayerId);
}
```

Справа у тому, щоб користуватися мережними списками, потрібно щоб вони мали значимий тип даних (наприклад, структуру), а не посилання (наприклад, клас). Це потрібно для того, щоб Netcode міг правильно серіалізувати значення, тобто перетворити об’єкт у дані і навпаки, які далі можно зберегти або передати по мережі.

Для вирішення цієї задачі була створена структура "PlayerData" (лістинг 3.8), яка використовується як тип даних, і містить наступні поля: ідентифікатор клієнта, значення кольору, яке гравець попердньо обрав, ім’я гравця, ідентифікатор клієнта (для сервісів Unity) кількість грошей та показник банкрутсва гравця.

Може виникнути питання, навіщо було реалізовано два ідентифікатора клієнта і відповідь буде проста. Перший числовий ідентифікатор (clientId) використовується локально мережним сервісом Netcode у межах однієї сесії, для позначення гравців, зазвичай значення для клієнта починаються від 1, адже 0 завжди належить хосту або серверу. Другий – це строковий унікальний ідентифікатор користувача (playerId), який встановлює сама Unity і використовує його для хмарних сервісів. Цей ідентифікатор вже буде однаковий у гравця на різних сесіях, тому що він прив’язаний саме до його аккаунта.

Також важливо розуміти значення інтерфейсу IEquatable<T>, який потрібен для порівння нової та старої версії об’єкту й вирішення про потребу сповістити клієнтів про оновлення. Без цього інтерфейсу Unity не може

дізнатися, чи дійсно було змінено поле і таким чином не надішле оновлення по мережі.

Лістинг 3.8 – Структура "PlayerData"

```
public struct PlayerData : IEquatable<PlayerData>,
INetworkSerializable
{
    public ulong clientId;
    public int colorId;
    public FixedString64Bytes playerName;
    public FixedString64Bytes playerId;
    public int playerMoney;
    public bool playerBankrupt;
    public bool Equals(PlayerData other)
    {
        return
            clientId == other.clientId &&
            colorId == other.colorId &&
            playerId == other.playerId &&
            playerMoney == other.playerMoney &&
            playerBankrupt == other.playerBankrupt &&
            playerName == other.playerName;
    }
    public void NetworkSerialize<T>(BufferSerializer<T>
serializer) where T : IReaderWriter
    {
        serializer.SerializeValue(ref clientId);
        serializer.SerializeValue(ref colorId);
        serializer.SerializeValue(ref playerName);
        serializer.SerializeValue(ref playerId);
        serializer.SerializeValue(ref playerMoney);
        serializer.SerializeValue(ref playerBankrupt);
    }
}
```

Головною метою події "NetworkManager_ServerConnectedCallback" є встановлення нових значень до "PlayerData", а саме кольору та ідентифікатору клієнта (Netcode) і подальше занесення цієї структури до списку "playerDataNetworkList". За встановлення значення ім’я гравця відповідає подія "SetPlayerNameServerRpc" (лістинг 3.9).

Перше на що слід звернути увагу, операція по присвоєнню імені гравця виконується тільки на сервері. Доступ на запис до мережного списку "playerDataNetworkList" має тільки сервер (але це можна змінити). Спочатку

за допомогою методу "GetPlayerDataIndexFromClientId" отримується значення індексу клієнта у списку, далі відбувається присвоєння структури, яка взята зі списку по значенню, до нової структури типу "PlayerData" і вже у ній змінюється поле "playerName". Останнім єтапом є заміна старої структури під цим індексом у списку на нову. Саме така процедура присвоєння працюю з декількох причин. По-перше, список містить структури, тобто значимі типи даних і змінюючи їх здійснюється зміна їх копії. По-друге, неможливо напряму у списку змінити конкретне поле, тому що мережні списки такі зміни не відслідковують. Через це потрібно повністю змінювати об’єкт і потім наново його присвоювати у список, щоб ці зміни коректно синхронізувались по мережі. Метод "SetPlayerIdServerRpc" працюю по такій самій схемі, тільки змінюється поле "playerId".

Лістинг 3.9 – Метод "SetPlayerNameServerRpc"

```
[ServerRpc(RequireOwnership = false)]
private void SetPlayerNameServerRpc(string playerName,
ServerRpcParams serverRpcParams = default)
{
    int playerDataIndex =
GetPlayerDataIndexFromClientId(serverRpcParams.Receive.SenderCli
entId);
    PlayerData playerData =
playerDataNetworkList[playerDataIndex];
    playerData.playerName = playerName;
    playerDataNetworkList[playerDataIndex] = playerData;
}
```

Остання подія, на яку ми підписуємося під час методу "StartHost" – це "NetworkManager_Server_OnClientDisconnectCallback" (лістинг 3.10).

Лістинг 3.10 – Подія "OnClientDisconnectCallback"

```
private void
NetworkManager_Server_OnClientDisconnectCallback(ulong clientId)
{    for (int i = 0; i < playerDataNetworkList.Count; i++)
    {        PlayerData playerData = playerDataNetworkList[i];
        if (playerData.clientId == clientId)
        {            playerDataNetworkList.RemoveAt(i);        }
    }}
```

Головна задача події вище, коли гравець з якоїсь причини покидає сесію, необхідно щоб він був видалений зі списку "playerDataNetworkList" для подальшої коректної роботи сесії. Логіка роботи методу проста, за допомогою циклу перебирається весь масив і при співпадіння переданого "clientId" зі значенням з списку, такий об'єкт видаляється.

Другий метод "StartClient" використовується для ініціалізації під’єднання клієнта до серверу або хосту (лістинг 3.11). Метод має один виклик події типу "EventHandler", яка оголошена на початку скрипта "MultiplayerMonopoly". Вона необхідна для сповіщення скрипта "ConnectionUI". Підключення гравця до сесії займає деякий час, тому для візуального відображення цього процесу існує об'єкт "ConnectionUI" з відповідним скриптом. Коли гравець намагається підключитися, на екрані він може бачити надпис "Connecting…" та трохи затемнений екран. Це робить процес очікування гравця більш зрозумілим та чітким для нього. Також у метод "StartClient" додано підписку двох методів. Перший спрацьовує, коли гравець від’єднується від серверу, другий навпаки, при приєднанні. Також у кінці викликається метод, який ініціює підключення клієнта через "NetworkManager".

Лістинг 3.11 – Метод "StartClient"

```
public void StartClient()
{
    OnTryingToJoinGame?.Invoke(this, EventArgs.Empty);
    NetworkManager.Singleton.OnClientDisconnectCallback +=
NetworkManager_Client_OnClientDisconnectCallback;
    NetworkManager.Singleton.OnClientConnectedCallback +=
NetworkManager_OnClientConnectedCallback;
    NetworkManager.Singleton.StartClient();
}
```

Метод "OnClientDisconnectCallback" (лістинг 3.12), який спрацьовує під час від’єднання клієнту, має на меті сповіщення скрипту "ConnectionUI" та об’єкту "LobbyMessageUI". Логіка реагування скрипту "ConnectionUI" необхідна для приховування вікна з надписом "Connecting…".

Скрипт "LobbyMessageUI" накладений на об’єкт "LobbyMessageUI", який вмикається, коли гравець не зміг доєднатися до сесіїї, з якоїсь причини. Візуально це відображається у повідомленні, де пояснюється причину, та кнопку повернення назад.

Лістинг 3.12 – Метод "OnClientDisconnectCallback"

```
private void
NetworkManager_Client_OnClientDisconnectCallback(ulong clientId)
{    OnFailedToJoinGame?.Invoke(this, EventArgs.Empty);    }
```

Наступний метод "OnClientConnectedCallback" (лістинг 3.13) виконує схожі дії, які вже були описані раніше, а саме: запис імені гравц’я до списку та присвоєння гравцю унікального номеру.

Лістинг 3.13 – Метод "OnClientConnectedCallback"

```
private void NetworkManager_OnClientConnectedCallback(ulong
clientId)
{
    SetPlayerNameServerRpc(GetPlayerName());
  SetPlayerIdServerRpc(AuthenticationService.Instance.PlayerId);
}
```

Загалом цього достатньо для розуміння необхідності класу "MonopolyMultiplayer". Він містить ще багато цікавих та корисних методів але для їх розуміння потрібно розглядати ці методи спільно з об’єкати, які їх використовують.

3.5.3 Об’єкт "MonopolyLobby"

Об’єкт "MonopolyLobby" містить скрипт з відповідною назвою. Головною метою скрипта є реалізація функціоналу кнопок, а також деяких інших особливостей лобі.

Для початку потрібно розглянути оголошені зміні класу "MonopolyLobby" (лістинг 3.14). Змінна "Instance" дозволяє з будь-якого місця

звернутися до методів цього класу. Цей паттерн має назву сінглтон. Далі оголошено низка подій "OnCreateLobbyStarted", "OnCreateLobbyFailed", "OnCreateLobbyFailed" та інші, які тісно пов’зані зі скриптом "LobbyMessageUI". Кожна з цих подій, відповідає за конкретний випадок з гравцем та відповідне повідомлення, яке буде виводитися гравцю у тій чи іншій ситуації.

Окремо потрібно виділити подію "OnLobbyListChanged", типом делегату якого є міні клас "OnLobbyChangedEventArgs", який містить одне поле, а саме список під назвою "lobbyList". Цей список містить посилання на усі створені гравцями лобі. Задумка використання події з користувацким типом наступна, коли список змінюється, наприклад, якийсь гравець створив нове лобі, потрібно щоб лобі гравця оновилося та відобразило нове, щойно створене лобі. Для цього необхідно створити новий об’єкт (посилання на щойно створено лобі) через який гравці зможуть доєднатися до нього. Це реалізується за допомогою скрипта "LobbyUI".

Лістинг 3.14 – Структура класу "MonopolyLobby"

```
public static MonopolyLobby Instance { get; private set; }
public event EventHandler OnCreateLobbyStarted;
public event EventHandler OnCreateLobbyFailed;
public event EventHandler OnJoinStarted;
public event EventHandler OnQuickJoinFailed;
public event EventHandler OnJoinFailed;
public event EventHandler <OnLobbyChangedEventArgs>
OnLobbyListChanged;
public class OnLobbyChangedEventArgs : EventArgs
{     public List<Lobby> lobbyList;     }
private Lobby joinedLobby;
private int MaxPlayersNumber = 6;
private float heartbeatTimer;
private float listLobbiesTimer;
```

Далі оголошено змінну "joinedLobby", типом якої є внутрішній клас Unity "Lobby". Зміна призначена для зберігання лобі, до якого гравець приєднався (через швидке підключення або за допомогою коду), або яке він створив. Змінна "MaxPlayersNumber" зберігає максимальну кількість клієнтів,

які можуть доєднатися до однієї сесії..

На початку роботи цього класу, в методі "Awake", виконується ініціалізація мережних сервісів Netcode (лістинг 3.15). Метод "InitializeAuthentication" – це асинхронний метод, який виконується на фоні та не блокує основний потік виконання. На початку перевіряється чи були ініціалізовані мережні сервіси (для запобіганню дублювання), далі створюється об’єкт параметрів ініціалізації та встановлюється новий профіль користувача, з випадковим ім’ям користувача. Після цього ініціалізуються UnityServices, з попередньо створеними параметрами та виконується анонімна авторизація, яка дуже зручна для простого входу, адже користувачу не потрібно вказувати логін та пароль, Unity автоматично привласнить унікальний Id користувачу. Цей базовий але важливий для роботи крок потрібен для використання наступних сервісів Unity: Lobby та Relay.

Лістинг 3.15 – Метод "InitializeAuthentication"

```
private async void InitializeAuthentication()
{
    if (UnityServices.State != 
ServicesInitializationState.Initialized)
    {
        InitializationOptions options = new 
InitializationOptions();

options.SetProfile(UnityEngine.Random.Range(0,1000).ToString());
        await UnityServices.InitializeAsync(options);
        await 
AuthenticationService.Instance.SignInAnonymouslyAsync();
    }
}
```

Lobby – це сервіс Unity, який дозволяє збирати гравців в одному місці, перед тим як розпочати гру. Цей сервіс потрібно окремо підключати до проєкту адже за замовчуванням його немає. Сервіс дуже сильно полегшує розробку будь-яких мультиплеєрних проєктів та дозволяє розробникам зекономити час на мережевій взаємодії.

Relay – це також сервіс Unity, який довзоляє гравцям спілкуватися між

собою через сервіси Unity. Його мета створити віртуальне з’єднання між гравцями та передавати дані між ними. Тобто усі мережні дані проходять через сервіси Unity, що дозволяє уникнути проблем із мережею, не турбуватися про відкриті порти та запускати гру на будь-якому інтернет-з’єднанні. Саме без цього сервісу не можливо запустити гру на двох різних пристроях та грати разом.

Далі слід розглянути метод "Update", який виконується кожен кадр, тобто стільки разів на секунду, скільки кадрів рендерить гра. Він містить два важливих методи: "HandleLobbyHeartBeat" та "HandlePeriodicListLobbies".

Перший метод "HandleLobbyHeartBeat" (лістинг 3.16) потрібен для підтримання лобі активним. Справа у тому, що коли гравець створює лобі, він очікує під’єднання інших гравців, а це у свою чергу займає деякий час. Unity може зчитати лобі не активним та автоматично видалити його. Тому для усунення цієї проблеми, було розроблено цей метод (який виконується тільки сервером або хостом), який кожні 15 секунд надсилає сигнал до лобі та робить його активним.

Лістинг 3.16 – Метод "HandleLobbyHeartBeat"

```
private void HandleLobbyHeartBeat()
{    if (IsLobbyHost())
    {
        heartbeatTimer -= Time.deltaTime;
        if (heartbeatTimer <= 0f)
        {
            float heartbeatTimerMax = 15;
            heartbeatTimer = heartbeatTimerMax;
LobbyService.Instance.SendHeartbeatPingAsync(joinedLobby.Id);
        }
    }     }
```

Другий метод "HandlePeriodicListLobbies" (лістинг 3.17) потрібен для постійного оновлення списку створених лобі. Метод оновлює список кожні 3 секунди та відображає лише відкриті лобі. Спочатку виконується перевірка на відсутність посилання на лобі через змінну "joinedLobby", тому що якщо посилання є – значить користувач вже під’єднався до лобі. Далі перевіряється

чи користувач був авторизований через сервіси Unity. Фінальним етапом є перевірка поточної сцени, адже якщо сцена інша (не лобі), то користувач або вийшов з лобі або вже очікує сесію у лобі. Далі виконується розрахунки зі значенням часу та коли змінна "listLobbiesTimer" менше або дорівнює 0 виконується оновлення значень таймеру та оновлення списку лобі за допомогою методу "ListLobbies" (лістинг 3.18).

Лістинг 3.17 – Метод "HandlePeriodicListLobbies"

```
private void HandlePeriodicListLobbies()
{
    if (joinedLobby == null &&
        AuthenticationService.Instance.IsSignedIn &&
UnityEngine.SceneManagement.SceneManager.GetActiveScene().name
== Scenes.Lobby.ToString())
    {
        listLobbiesTimer -= Time.deltaTime;
        if (listLobbiesTimer <= 0f)
        {
            float listLobbiesTimerMax = 3f;
            listLobbiesTimer = listLobbiesTimerMax;
            ListLobbies();
        }
    }     }
```

Метод "ListLobbies" – є асинхронним методом, адже оновлення списку лобі не повинно заважати виконанню основного коду програми. Метод виконує запит до серверу через створену вибірку (кількість вільних місць у лобі повинно бути більше 0) та потім через виклик події оновлює список, який зберігає усі лобі. Увесь цей код відбувається у блоці try-catch, адже у разі невдалого запиту до серверу, потрібно коректно оброблювати помилки.

Лістинг 3.18 – Метод "ListLobbies"

```
private async void ListLobbies()
{    try
    {        QueryLobbiesOptions queryLobbiesOptions = new
QueryLobbiesOptions
        {
            Filters = new List<QueryFilter>
            {
                new
```

```
QueryFilter(QueryFilter.FieldOptions.AvailableSlots, "0",
QueryFilter.OpOptions.GT)
            }
        };
        QueryResponse queryResponse = await
LobbyService.Instance.QueryLobbiesAsync(queryLobbiesOptions);
        OnLobbyListChanged?.Invoke(this, new
OnLobbyChangedEventArgs
        {
            lobbyList = queryResponse.Results
        });
    } catch (LobbyServiceException ex)
    {
        Debug.Log(ex);
    }
}
```

Далі будуть розглянуті ще 4 головних методи цього класу, а саме: створення лобі та 3 різні версії підключення до сесії. Перед тим, як почати розбір методу, який створює лобі, важливо зазначити одну річь. Коли гравець натискає, у сцені з лобі, на кнопку створити лобі, відкривається невелике віконце, де користувач вказує назву цього лобі та його тип (відкрите чи закрите) і після цього вже відбувається процес створення за допомогою асинхронного методу "CreateLobby" (лістинг 3.19).

Даний метод приймає декілька вхідних параметрів, а саме назву та тип лобі та виконується наступним чином. Спочатку відбувається виклик події "OnCreateLobbyStarted". Як вже зазначалося раніше, на цю подію підписан скрипт "LobbyMessageUI", який відповідає за візуальне відображення та інформування користувача під час деяких подій. Далі у блоці try-catch відбувається спроба створення та присвоєння лобі до змінної "joinedLobby", за допомогою асинхронної функції "CreateLobbyAsync", параметрами якої є: назва лобі, максимальна кількість користувачів та тип лобі. За допомогою методу "StartHost" скрипта "MonopolyMultiplayer" відбувається ініціалізація гравця як хоста та підписання його на деякі події (більш детально цей метод було описано раніше). Фінальним єтапом є перенесення гравця до нової мережевої сцени. У разі невдалої спроби створення лобі викликається подія "OnCreateLobbyFailed".

Лістинг 3.19 – Метод "HandlePeriodicListLobbies"

```
public async void CreateLobby(string lobbyName, bool isPrivate)
{    OnCreateLobbyStarted?.Invoke(this, EventArgs.Empty);
    try
    {    joinedLobby = await
LobbyService.Instance.CreateLobbyAsync(lobbyName,
MaxPlayersNumber, new CreateLobbyOptions
      {   IsPrivate = isPrivate,        });
          MonopolyMultiplayer.Instance.StartHost();
          SceneManager.PlaySceneNetwork(Scenes.CharacterSelect);
    } catch (LobbyServiceException ex)
    {   Debug.Log(ex);
        OnCreateLobbyFailed?.Invoke(this, EventArgs.Empty); } }
```

3.5.4 Інтерфейс лобі

За візуальне відображення сцени лобі, відповідає об’єкт "LobbyUI". Він містить багато необхідних елементів (рисунок 3.10), а саме: фон для панелі, кнопку повернення назад до меню, текстове поле для вводу імені гравця, кнопку створення сесії та два варіанти під’єднання за допомогою кнопок.

Рисунок 3.10 – Основні елементи сцени "Lobby"

На об’єкт "LobbyUI" також накладений скрипт "LobbyUI", який містить посилання на усі необхідні UI елементи.

3.6 Сцена очікування гравців

Дана сцена призначена для очікування підключення інших гравців. Попередньо гравець обрав лобі, до якого хоче приєднатися, для того щоб

візуально відобразити підключення та перебування його у ігровій сесії і призначена ця сцена. Коли гравець під’єднується до кімнати, у ній вже завжди буде присутній один гравець – сервер. Адже, за функціональність сесії відповідає гравець-сервер, а не наприклад, віддалений сервер.

Логіка під’єднання гравця наступна, у сцена заздалегідь вже є 6 префабів граців (максимально допустима кількість підключених гравців у одній сесії). Коли гравець-сервер створює лобі та переходить до цієї сцени, префаби стають неактивними, тобто невидимими для користувачів. Надалі під час підключення гравця, наступний за порядком префаб стає активним.

Варто зазначити, що над гравцями видно їх ім’я, яке вони написали у попередній сцені. Також є додатковий функціонал у вигляді видалення з лобі гравців. Над гравцями у сцені є кнопка видалення, яку бачить лише сервер. Під час видалення гравців, відбуваються процеси з видалення зі списків цього гравця, видалення з NetworkManager та відбувається відписка гравця від події. Загалом це не дуже складний процес але без нього програма не буде коректно працювати. Також в усіх гравців є можливість вибору кольору з вибірки кольорів, яка знаходиться внизу екрана. Під час приє’днання гравця до сесії, він отримує перший незайманий колір зі списку кольорів (лістинг 3.20), який надалі може бути змінений.

Лістинг 3.20 – Метод "GetFirstUnusedColorId"

```
private int GetFirstUnusedColorId()
{    for (int i = 0; i < PlayerColorList.Count; i++)
    {        if (IsColorAvailable(i))
        {       return i;        }
    }
    return -1;}
```

Також зроблено перевірку, перед тим як гравець натискає на уподобаний колір, виконується перевірка на зайнятість цього кольору вже іншим гравцем (лістинг 3.21). Якщо колір вже зайнятий, то нічого не відбувається, якщо ні то присвоюється цьому гравцю, відповідно префаб гравця стає відповідного кольору.

Лістинг 3.21 – Метод "IsColorAvailable"

```
private bool IsColorAvailable(int colorId)
{    foreach (var playerData in playerDataNetworkList)
    {        if (playerData.colorId == colorId)
        {     return false;      }   }
    return true;}
```

Додатково зверху екрану розташовано два текстових поля. Перший, відображає назву лобі, з якою його видно у списках лобі. Другий, це код лобі, за допомогою якого гравці можуть доєднатися.

Останнім функціональним елементом сцени є кнопка "Ready", яка позначає готовність конкретного гравця до початку гри. Логіка проста, гравці не потраплять до наступної сцени, поки усі не підтвердять свою готовність. Також, тут встановлено обмеження щодо серверу, коли він перебуває у сесії один, кнопка "Ready" для натискання блокується, адже після її натиснення, його одного буде перенесено до сцени з грою, що не зовсім коректно. Кнопка починає працювати, коли лобі містить більше ніж 1 гравця, що робить її роботу більш логічною.

Варта відокремити наступний момент, коли гравці підтвердили свою готовність до початку гри, вони переходять до наступної сцени, але лобі все ще залишається видим для підключення. Тому спочатку виконується видалення лобі, а потім вже перехід до наступної сцени.

Отже, дана сцена потрібна для декількох важливих дій, а саме: очікування підключення усіх гравців, вибір кольору та підтвердження щодо готовності розпочати гру. Відсутність цієї сцени, позначилась би на погіршенні досвіду користування та більш незрозумілості для гравців.

3.7 Головна сцени гри

Данна сцена є найбільшою за функціоналом у грі. Саме тут реалізовано велика кількість ігрових механік, а саме: переміщення гравців по дошці, взаємодія з полями дошки, кидання та візуальне відображення ігрових кубиків,

візуальне відображення майна для кожного гравця, інформація по картці при натисканні на поле, повний функціонал створення угод, переміщення камери, створення навколишньої атмосфери біля дошки, відображення таблиці з усіма гравцями, створення та заповнення майнових карток, спеціальні інтерактивні поля та багато іншого.

При успішному підключенні усіх гравців до цієї сцени, відбувається декілька важливих речей. По-перше, зі сторони сервера відбувається створення необхідної кількості фішок гравців. Саме сервер повинен створювати мережні об’єкти для подальшого коректного відображення та роботи їх на усіх клієнтах. Далі сервер створює два мережні об’єкти – ігрові кубики, які на сцені користувач прямо не бачить, оскільки механізм підкидання та випадіння чисел у даному проекті реалізований не зовсім у стандартний спосіб.

Кубики розташовані в окремому місці і не змінюють свого положення протягом сесії, тобто вони просто прокручуються на місці. До них прикріплена окрема камера, яка виводить зображення на спеціальний матеріал Render Texture, який прикріплений до UI елемента на екрані. Таким чином, у верхньому лівому куту екрана користувач постійно має змогу бачити 2 кубики. Це дозволяє вільно змінювати розмір екрану та не перейматися щодо адаптування 3D-об’єктів до UI, адже поверхня, через яку відбувається трансляція кубів, самостійно правильно адаптується до нових розмірів екрану користувача.

Далі постає питання, як зрозуміти, яке число випало та коректно відобразити грань з цим числом користувачеві. Для коректної взаємодії та подальшого логіки виконання, на кожну грань куба було додано пустий об’єкт, який має фізичні властивості та потрібен для передачі сигналу. Для обертання об’єктів у Unity, потрібно використовувати фізику. Для цього є метод AddTorque, який приймає параметр напрямку поштовху у вигляді параметру з типом Vector3 та використовується для застосування крутного моменту до об'єкта з фізикою, тобто щоб обертати об'єкт навколо його осей.

Скрипт "DiceController" постійно має посилання на два куби, тому по черзі дає поштовхи кожному з кубів. Далі скрипт чекає поки швидкість кубиків буде меншою ніж 2.8f, і потім вже завдяки сигналам на гранях, визначаються дві найближчі сторони до умовної площини і виконується обчислення значень та подальше плавне підведення граней до гравця. Скрипт заздалегідь знає усі можливі комбінації та необхідні кути при яких куби будуть красиво доводитися та у кінцевому випадку гравець може чітко побачити результат випадіння кубиків.

Усі прорахунки та підведення граней відбуваються лише на стороні серверу, а клієнт лише переглядає та синхронізує обертання осей кубиків. Після того, як куби остаточно зупинилися до скрипту "GameController" передаються значення отримані під час розрахунку кубів і далі, в залежності від ситуації, відбувається подальші дії з цими даними.

Після успішного створення фішок та кубів, сервер починає створення таблиці с усіма гравцями. Логіка дуже проста, у скрипті "GameController" є лічильник, який підраховує кількість підключених гравців до сесії. Цей параметр передається до методу "PutPlayersOnTableUI" скрипта "TablePlayersUI", де відбувається створення ігрових префабів та заповнення їх відповідними даними.

Для отримання конкретних даних від гравця, наприклад його ім’я або поточну кількість грошей у нього, використовується вже описана вище структура "PlayerData", яка містить інформацію щодо кожного гравця. Доступ до цієї структури можна отримати за допомогою класу "MonopolyMultiplayer", а саме відповідних методів "GetPlayerDataFromPlayerIndex" або "GetPlayerMoney".

Після успішного створення усіх відповідних компонентів, сервер ініціалізує важливий список "PlayersBunkruptList", який необхідний для подальшого коректного відпрацювання банкротства гравців.

Фінальним етапом початку гри є надання будь-якому гравцю ходу. Це відбувається за рахунок методу "btnTurnController", який має доступ до

багатьох кнопок та відповідно до кожної фази у грі вмикає або вмикає кнопки. У грі реалізовано декілька кнопок, а саме: кнопка старту хода, кнопка завершення ходу, кнопка покупки разом з кнопкою завершення ходу. Кожна з цих кнопок спрацьовує при певних умовах, наприклад, коли поле вже куплено кимось спрацьовує кнопка завершення ходу, коли поле вільне то кнопка покупки разом з кнопкою завершення ходу. Таким чином, це дає грі більш гнучкість та вибір гравцеві, щодо його подальших дій. Після того як сервер виконав усі вищезгадані дії – починається гра.

Гра починається з натисканням одного з гравців кнопки "Roll Dices" після якої відбувається прокручування кубиків та отримання результату. Далі потрібно плавно перемістити гравця на значення цього результату. Знов таки, переміщувати мережні об'єкти може тільки сервер або власник цього об'єкта. У даному проєкті за переміщення гравців відповідальний сервер, а клієнти лише бачать синхронізацію з сервером.

Отже у скрипті "GameController" є відповідний метод "MoveCurrentPlayerServerRpc", який бере індекс поточного гравця та викликає у цього гравця відповідний метод "PlayerMove". Справа у тому, що у грі є дві групи (структура та клас), відповідальні та пов’язані якось з гравцями. Перша, це "PlayerData", яка вже була описана раніше. Друга, це клас "Player", який дається гравцеві при створенні його фішки. Головна мета цього класу – це управління фізичним об’єктом гравця. Тож коли потрібно перемістити гравця по дошці, викликаються відповідні методи з цього класу.

Для красивого переміщення гравців по ігровому полю, використовуються корутини. Корутина – це спеціальний метод, який дозволяє виконувати код поступово, а саме кожен кадр. Таким чином, можна призупиняти виконання між кадрами на певний час, не блокуючи основний потік виконання гри.

Корутина "PlayerMoveCoroutine" (лістинг 3.22) реалізована у наступний спосіб. Спочатку задаються деякі важливі параметри (наприклад, висота, на якій буде гравець, тривалість корутини тощо). Далі з отриманого результату

кубів, виконується покроковий розрахунок та переміщення гравця. Береться стартова позиція гравця, далі отримуються координати карти. Важливо зробити перевірку на наявність останньої картки, адже якщо це дійсно так, потрібно обнулити значення наступної позиції (максимальна кількість полів на дошці 40) і викликати подію, яка надасть гравцеві трохи грошей за проходження кола дошки. Далі відбуваються присвоєння значень у змінну "goTo", яка зберігає значення нових координат. Нарешті у циклі while відбується переміщення гравця кожен кадр за допомогою методу Vector3.Lerp, який дозволяє інтерполювати (плавно переходити) між двома векторами. Після успішного переміщення відбувається присвоєння нової поточної позиції гравця та сповіщення класу "BoardController" про зміну позиції гравця. Це потрібно для подальших дій пов’язаних з полями, інформацією щодо них.

Лістинг 3.22 – Корутина "PlayerMoveCoroutine"

```
public IEnumerator PlayerMoveCoroutine(int steps)
{    float playerHeight = 0.16f;
    float moveDuration = .6f;
    for (int i = 0; i < steps; i++)
    {   Vector3 startPosition = gameObject.transform.position;
        float elapsedTime = 0f;
        int nextPosition = currentPosition + 1;
        if (nextPosition == 40)
        {  playerCircleGameBoard?.Invoke(this, EventArgs.Empty);
            nextPosition = 0;        }
        Vector3 goTo =
BoardController.Instance.GetBoardPosition(nextPosition);
        if (nextPosition == 2 || nextPosition == 5 ||
nextPosition == 22 || nextPosition == 25)
        {  goTo.z = startPosition.z;        }
        if (nextPosition == 15 || nextPosition == 17 ||
nextPosition == 35 || nextPosition == 38
        {  goTo.x = startPosition.x;        }
        goTo.y = playerHeight;
        while (elapsedTime < moveDuration)
        {  gameObject.transform.position =
Vector3.Lerp(startPosition, goTo, elapsedTime/moveDuration);
            elapsedTime += Time.deltaTime;
            yield return null;        }
        gameObject.transform.position = goTo;
        currentPosition = nextPosition;
BoardController.Instance.CurrentPlayerPosition(currentPosition);
    }}
```

Таким чином, відбується покадрове переміщення гравця від карти до карти, також при потраплянні на деякі карти виконується додавання певних значень для кращого візуального розташування на дошці.

Після того як гравця було переміщено на нове поле, потрібно зрозуміти, на якому полі гравець знаходиться та зробити декілька наступних речей. Якщо гравець на звичайному полі з майном, то потрібно перевірити платіжоспроможність гравця щодо цієї клітки (у разі якщо гравець забажає купити це поле) і далі ввімкнути необхідні кнопки. Якщо гравець може придбати поле ввімкнути кнопку купити разом з закінченням ходу, якщо ні то ввімкнути лише закінчення ходу. У разі, якщо гравець знаходиться на спеціальному полі, наприклад з податками, необхідно викликати у гравця метод "PlayerPayTax", який потребує з гравця заплатити необхідну суму коштів. Загалом існує декілька спеціальних полів, а саме: поля в’язниця, поле яке відправляє до в’язниці, поле відпочинку, поле скарбів, поле податків та поле шансу. Варто підкреслити, коли гравець змінює своє положення на дошці, відбувається зміна положення камери, для кращого відображення дошки відносно гравця.

Після того як гравець зробив свій вибір щодо покупки поля або виконав дії спеціальних полів він завершує свій хід і передає хід іншому гравцеві, це відбувається за допомогою методу "NextPlayerTurnServerRpc".

Далі будуть розглянуті функціональну особливості покупки, продажу покращення та погіршення майна. Для розуміння роботи цих функцій, потрібно розглянути, де взагалі розташований функціонал для гравця.

Покупка майна доступна для гравця лише коли він перебуває на цьому полі і зараз його хід. Коли гравець купив майно, колір над майном став кольором гравця, і це є візуальним підтвердженням покупки.

Слід розглянути наступну важливу деталь, а саме перегляд інформації щодо майна. За візуальне відображення інформації пов’язаної з карткою відповідальний клас "BoardController". Коли гравець натискає на будь-яку карту виконуються операції зі створення невеликих вікон. Головне що

визначається при створенні таких вікон, чи володіє даний гравець полем, яке він хоче переглянути. Адже якщо гравець не володіє ним, потрібно заборонити йому доступ до функцій над цим полем. Перше вікно, яке створюється, це вікно, що відповідає за операції над цим полем (продаж, підвищення або зниження орендної плати). Потрібно зробити так, щоб доступ до цих операцій був лише у власника цього поля. Друге вікно надає користувачеві можливість переглянути вартість необхідну для покупки цього поля, для поліпшення (придбання будинків або готелів) або для погіршення (продаж будинків або готелів), а також вартість орендної плати в залежності від рівня поля. Також потрібно визначити поточний рівень орендної плати (якщо поле придбано) та візуально відобразити це спеціальним напівпрозорим об’єктом над ціною.

Не менш важливим є прорахунок платоспроможності гравця щодо поліпшення рівня картки. Наприклад, якщо гравець немає всієї суми для поліпшення поля, необхідно кнопку, яка відповідає за поліпшення зробити неробочою і таким чином зробити неможливим (тимчасово) підвищення орендної плати. Або наприклад, якщо наразі діє перший рівень орендної плати, тобто поле тільки придбано, кнопку зниження орендної плати зробити неможливим, адже поки ще немає будинків або готелів до продажу.

Отже, у даному класі перед створенням вікон з інформацією щодо конкретного поля, відбувається постійно перевірки на власника цього поля, на платоспроможність гравця, на поточний рівень орендної плати тощо.

Покупка, продаж, підвищення або зниження орендних плат може відбуватися лише у свій хід. Повертаючись до алгоритму покупки поля, потрібно ще згадати про одну важливу річ під назвою список майна гравця.

На екрані у правому верхньому куту, гравець має можливість постійно бачити список придбаного майна, у разі необхідності відкрити його та подивитися. За створення та оновлення цього списку візуально, відповідальний клас "UpdatePropertyTableUI", який слідкує за подією "AddPropertyFromPlayerList" класу "MonopolyMultiplayer".

Таким чином, коли гравець натискає на кнопку придбання поля, перше

що відбувається це додавання індексу цього поля до списку "playerPropertyList", який є у кожного гравця та зберігає усе майно куплене цим гравцем. Відповідно до події "AddPropertyFromPlayerList" відбувається візуальне оновлення майна на екрані. Далі у гравця викликається метод "BuyCard", який віднімає у гравця кількість грошей необхідну для покупки поля, встановлює нового власника цього поля та прибирає з картки текст, який відповідає за інформування щодо вартості цього поля. Далі відбувається заповнення над карткою поля кольором гравця та у разі, якщо відкриті вікна (інформування щодо картки) цього поля, відбувається перемальовування них, адже інформація щодо поля оновилася. Це повністю завершений цикл покупки картки гравцем.

Підвищення орендної плати поля відбувається за наступним алгоритмом. Спочатку визначається індекс поточного гравця та індекс поля, з яким гравець взаємодіє. Далі у картки визначається вартість підвищення, в залежності від поточного рівня (будинки або готелі), ціна може бути різною. Далі відбувається списання необхідної кількості грошей у гравця та встановлення нового рівня орендної плати. Так як інформація щодо картки змінилася, у гравців в яких відкрита поточна картка відбувається оновлення інформації.

Зниження орендної плати відбувається за схожим алгоритмом. Спершу отримується доступ до поточного гравця та поля, до якого він звернувся. Далі відбувається отримання ціни підвищення оренди та гравцеві повертається на рахунок половина від цієї суми. Цей невеликий але важливий для гри момент, дає гравцеві зрозуміти важливість своїх кроків та обдумання їх наперед задля уникнення небажаних ситуацій. У кінці відбувається оновлення інформації щодо картки у гравців, які наразі переглядають її.

Продаж майна відбувається наступним чином. Спочатку, як і при покупці, відбувається оновлення списку майна, а саме видалення за індексом, поля зі списку майна гравця. Далі отримується вартість цього поля при покупці і передається у метод "SellCard", викликаний у гравця. Цей метод повертає

половину від вартості поля гравцеві, встановляю власником цього поля нікого та вмикає для відображення текстове поле з вартістю цієї картки на дошці. Далі відбувається встановлення стандартного кольору поля, яке знаходиться над карткою. У кінці оновлюється інформація щодо цього поля у всіх клієнтів, які наразі переглядають цю картку.

Також при виконання усіх цих операції пов’язаних з карткою, відбувається текстове сповіщення усіх гравців у чаті за допомогою класу "ChatManager". У чаті гравці мають змогу як розмовляти один з одним так і бачити події, які відбуваються у грі.

Варто зазначити, що можливість збільшувати або зменшувати орендну плату з’являється, коли гравець купив усі міста однієї країни. Це відбувається за допомогою методу "VerifyAllCountryBelongToOnePlayer", який перевіряє це після того як гравець купив чи продав майно.

Разом зі списком майна гравця присутні ще дві важливі кнопки. Перша – це кнопка банкрутства гравця, при натискання на яку відбувається видалення гравця з черги на хід та з декількох важливих списків гравців. Але цей функціонал не передбачує видалення гравця з ігрової сесії, тобто він може залишитися для перегляду гри або за бажання вийти з цієї сесії. Друга – це кнопка для створення трейду, який передбачає обмін майном або грошима. Коли гравець натискає на цю кнопку, на екрані з’являється вікно, яке відображає список гравців (крім поточного гравця). Клієнт обирає гравця, якому хоче запропонувати угоду та далі бачить перед собою нове вікно, у якому є два слайдери, два поля для вводу та два списки майна гравців. Слайдери використовуються для зручного вибору кількості грошей, яку гравець хоче отримати або заплатити. Для більш точного вводу є спеціальне поле. Зі свого списку гравець може обрати поля, які він хоче віддати, зі списку іншого гравця, поля які він би хотів отримати. Після налаштування угоди, гравець натискає на кнопку "Create Trade" та усі гравці у поточній сесії бачать у вікні "Trades" нову угоду та мають змогу ознайомитися з нею.

Тут варто зазначити, що угоду можуть переглянути усі гравця але доступ

до неї мають тільки ті гравці, яких вона стосується. Тож гравець, якому вона адресована може прийняти або відхилити угоду. Гравець, який пропонує, має можливість видалити угоду. У разі прийняття відбувається автоматичний обмін майном та грошима відповідно до умов угоди. У разі відмови відбувається видалення угоди зі списку активних пропозицій. Таким чином, система угод розширює можливості взаємодії між гравцями та робить ігровий процес більш гнучким та динамічним.

Отже у підсумку можливо зазначити, що було розглянуто основні сцени цього проєкту та головні функціональні особливості, які стосуються як мережних взаємодій так і алгоритмів розрахунку. Було реалізовано механіку переміщення гравців між сценами, механіки купівлі/продажу полів, збільшення або зменшення вартості орендних плат, механіку створення угод та багато іншого.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Покроковий запуск проекту

Коли гравець запускає гру, він має змогу бачити перед собою головне меню (рисунок 4.1) з якого може зробити наступні кроки: розпочати грати, відкрити налаштування (рисунок 4.2), подивитися інформацію щодо автора (рисунок 4.3) або вийти з гри.

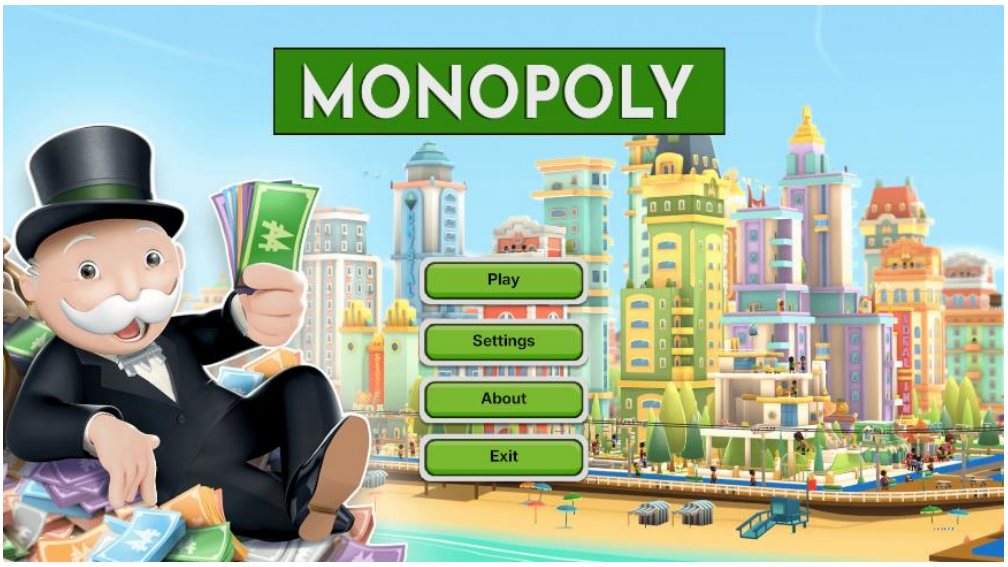

Рисунок 4.1 – Головне меню гри

Рисунок 4.2 – Вікно налаштувань гри

У меню налаштувань гри, користувач може змінити роздільну здатність екрану, налаштувати звук та музику.

Коли гравець вирішив грати, він натискає на кнопку "Play" (рисунок 4.1) та переміщується до наступної сцени під назвою "Лобі" (рисунок 4.4).

Рисунок 4.3 – Вікно з інформацією щодо автора гри

У лобі користувач має змогу самостійно створити сесію (відкриту чи закриту) або доєднатися до вже існуючої за допомогою кнопок чи відповідно кліком на лобі у списку.

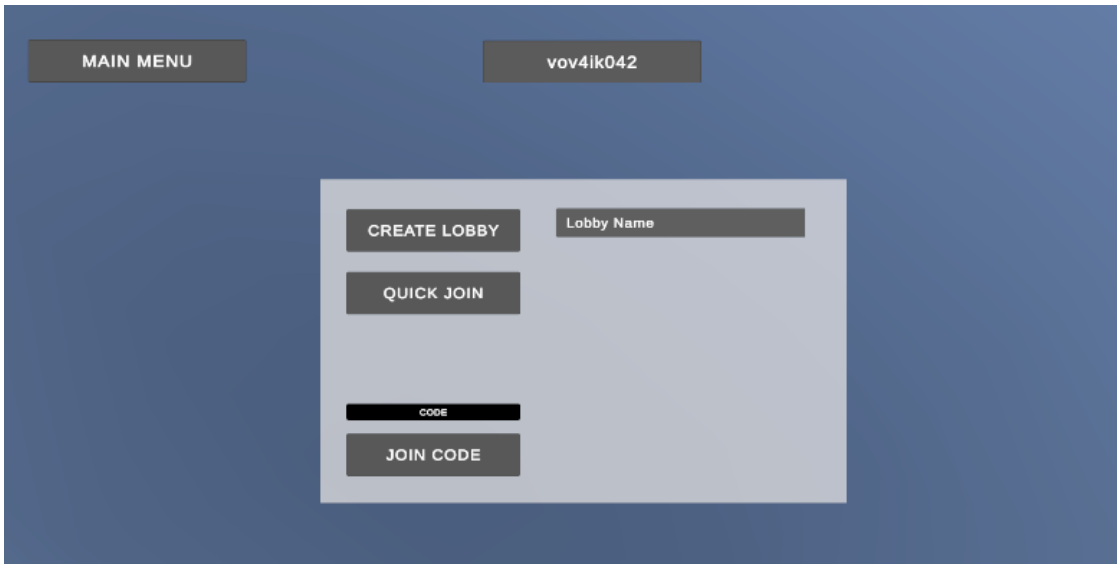

Рисунок 4.4 – Лобі

Після успішного створення або приєднання до сесії, користувач потрапляє до наступної сцени (рисунок 4.5), де очікує під’єднання інших гравців. Під час очікування користувач може обрати будь-який незайманий колір. Коли користувач готовий грати та усі гравців під’єдналися до сесії, він натискає на кнопку "Ready".

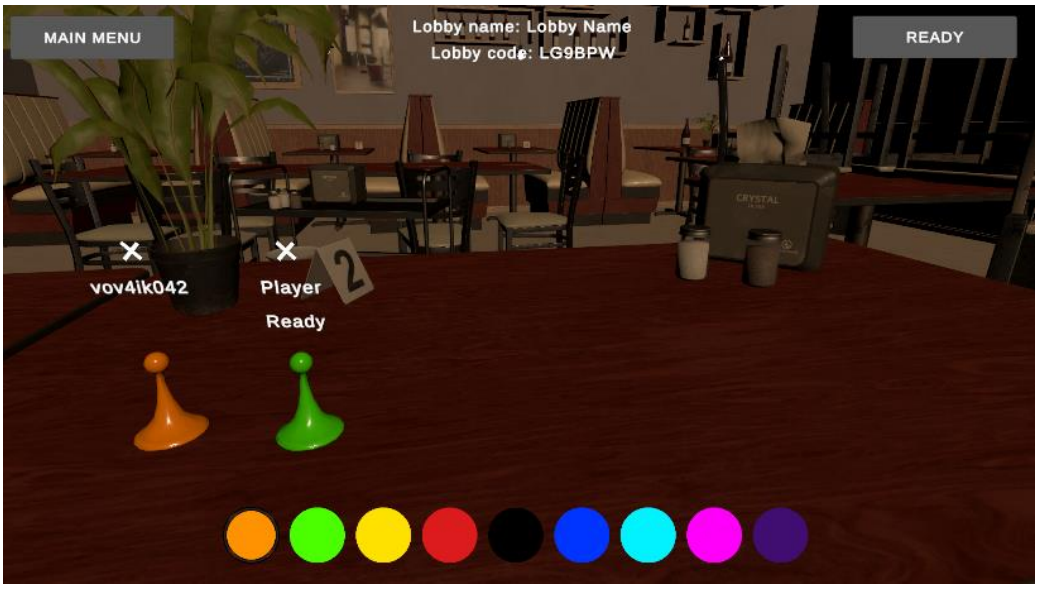

Рисунок 4.5 – Місце очікування гравців

Коли усі гравці підтвердили свою готовність розпочати гру, вони переміщуються до останньої сцени, у якій починаю грати (рисунок 4.6).

Рисунок 4.6 – Ігровий процес

Користувач має змогу спілкуватися за допомогою чату з іншими гравцями, переглядати інформацію щодо карток (рисунок 4.7), підкидати кубики, переміщуватися по полю, купляти чи продавати майно та створювати вигідні угоди (рисунок 4.8).

Рисунок 4.7 – Перегляд інформації поля у грі

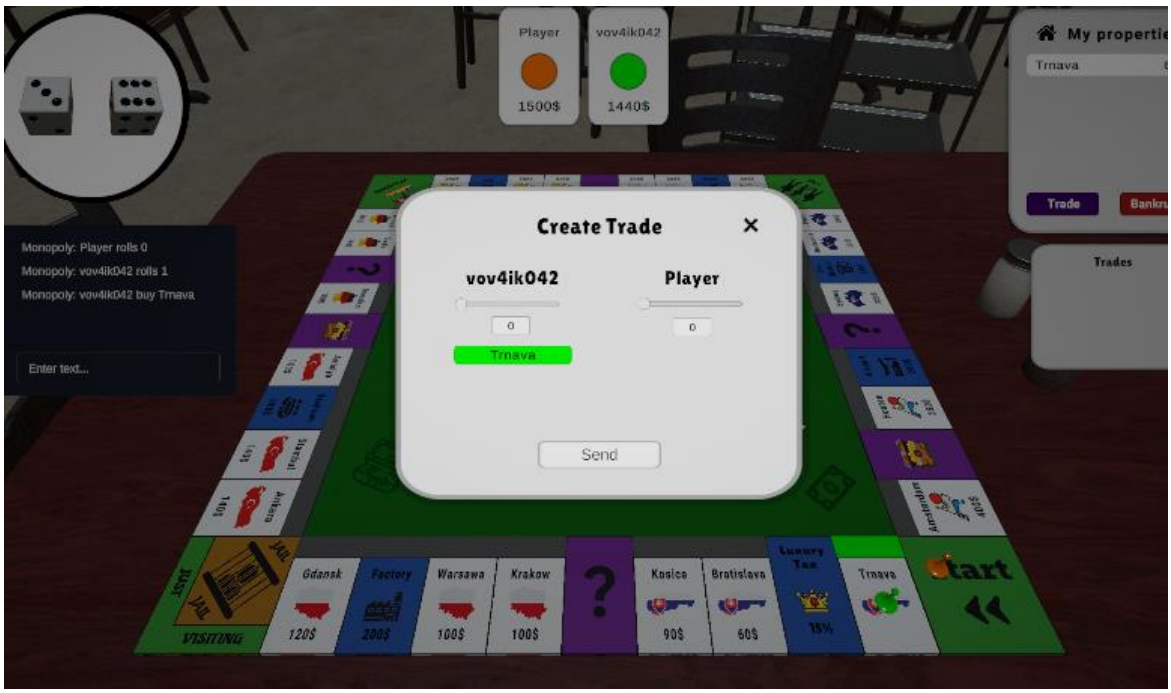

Рисунок 4.8 – Приклад створення угоди

ВИСНОВКИ

У межах даної кваліфікаційної роботи було успішно розроблено прототип багатокористувацької гри "Монополія", що поєднує класичні механіки настільної гри та сучасні технології.

У ході виконання роботи було досягнуто певних результатів, а саме: повністю створено ігрову логіку, яка відповідає класичним правилам гри "Монополія" (кидання кубиків, переміщення гравців по дошці, купівля/продаж об’єктів, сплата податків та оренди, утворення взаємовигідних угод, банкрутство, можливість спілкування між гравцями за допомогою чату, тощо), впроваджено підтримку мультиплеєрного режиму за допомогою мережних технологій, що забезпечують взаємодію між гравцями у режимі реального часу, розроблено зручний та інтуїтивно зрозумілий інтерфейс користувача, структуровано архітектуру проєкту, що дозволяє в майбутньому легко підтримувати та масштабувати проєкт.

У ході розробки проєкту багато уваги було приділено алгоритмам логіки гри та обробці нестандартних ситуацій, а саме: робота усіх сцен проєкту під впливом різних розмірів екрану користувача та адаптування відповідних UI елементів, незапланований вихід гравця під час гри, невдале під’єднання гравця до сесії (гра вже почалась або вже підключена максимальна кількість гравців), тощо. Також було багато часу виділено на тестування кінцевого результату на зручність користування програмою та зрозумілість інтерфейсу.

Отримані результати свідчать про успішну реалізацію поставлених завдань. Проєкт може слугувати базою для подальшого модифікування та вдосконалення, наприклад, додавання можливості гри з штучним інтелектом, мобільна адаптація гри, що збільшить кількість гравців або впровадження системи рейтингу гравців.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Steam Monopoly. URL: https://store.steampowered.com/app/2929170/MONOPOLY.

2. Marmalade Game Studio Monopoly. URL: https://www.marmalade gamestudio.com

3. Rento Fortune. URL: https://rento.com

4. Monopoly Millionaire. URL: https://pdalife.to/monopoly-millionaire-android-a2454.html

5. Kelly S., Kumar K., Unity Networking Fundamentals: Creating Multiplayer Games with Unity. Apress, 2021, 1st Edition. P. 292.

6. Hocking J. Unity in Action: Multiplatform Game Development in C# with Unity 5. Manning Publications, 2015, 1st Edition. P.352.

7. Lukosek G. Learning C# by Developing Games with Unity 5.x - Second Edition: Develop your first interactive 2D platformer game by learning the fundamentals of C#. Packt Publishing, 2016, 2nd Edition. P.230.

8. Patrick T. Start to Finish Visual C# 2015. Owani Press, 2016. P. 592.

9. Albahari J., Albahari B. C# 7.0 in a Nutshell: The Definitive Reference. O'Reilly Media, 2017, 1st Edition. P. 1087.

10. Troelsen A., Japikse P. Pro C# 7: With .NET and .NET Core. Apress, 8st Edition, 2017. P. 1437.

11. Doran J. Unreal Engine Game Development Cookbook, Packt Pub Ltd, 2015. P.326.

12. Meyers S. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14, 2014, 1st Edition. P. 332.

13. Marques G., Manzur A. Godot Engine Game Development in 24 Hours, Sams Teach Yourself: The Official Guide to Godot 3.0. Sams Publishing, 2018 1st Edition. P. 432.

14. Tidwell J., Valencia A., Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media, 2020, 3d Edition. P. 599.

15. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, New Riders, 2013 3d Edition. P. 216.

16. Platt D. The Joy of UX: User Experience and Interactive Design for Developers, Addison-Wesley Professional, 2016, 1st Edition. P.238.

17. Безсонов В.О., Майстренко Г.В., Філімончук Т.В. Ігровий онлайн застосунок "Монополія" з використанням рушія Unity. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.2: секція 2. Баку: ІСУ АР; Харків: НТУ «ХПІ»; Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ. 2025. с. 33.