Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка мультиплеєрної комп'ютерної гри «Tricky Squares» з використанням C# та Unity

Виконав: студент 4 курсу
групи ЗПІ-20б спеціальності
121 Інженерія програмного забезпечення
(шифр і назва напрямку підготовки, спеціальності)
Зелінський В.Р.
(Прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Коваленко О.О.
(Прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ПЗ Колесник І. С.
(Прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _____
« 10 » червня 2024 р.

м. Вінниця – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 12 » березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Зелінському Владиславу Руслановичу

1. Тема роботи – «Розробка мультиплеєрної комп'ютерної гри «Tricky Squares» з використанням C# та Unity».

Керівник роботи: Коваленко Олена Олексіївна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 року №80.

2. Срок подання студентом роботи: 3 червня 2024 року.

3. Вихідні дані до роботи: середовище розробки – Visual Studio, мова програмування – C#, движок – Unity, операційна система – Windows 7/8/10.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз сучасного стану комп'ютерних ігор у жанрі 2D та постановка задач дослідження; розробка методу, моделі та алгоритмів роботи застосунку; розробка модулів програмного застосунку; тестування програми; висновки; список використаних джерел; додатки; ілюстративна частина.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність розробки; тема; мета, об'єкт та предмет дослідження; постановка задачі;

новизна; практична цінність отриманих результатів; аналоги та їх аналіз; метод ініціалізації системи для автоматизації ігрових процесів; блок-схема загального алгоритму роботи застосунку; модель системи на прикладі діаграми варіантів використання; uml-діаграми; тестування застосунку; апробація та публікації результатів роботи; впровадження; фінальний слайд.

6. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 – 4	Коваленко О. О., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану комп'ютерних ігор у жанрі 2D та постановка задач дослідження	14.03.24 – 24.03.24	вик
2	Розробка методу, моделі та алгоритмів роботи застосунку	25.03.24 – 15.04.24	вик
3	Розробка модулів програмного застосунку	15.04.24 – 30.04.24	вик
4	Тестування програми	01.05.24 – 10.05.24	вик
5	Оформлення матеріалів до захисту БДР	11.05.24 – 30.05.24	вик

Студент (підпис) **Зелінський В. Р.** (прізвище та ініціали)

Керівник бакалаврської дипломної роботи (підпис) **Коваленко О. О.** (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 62 сторінки формату А4, на яких є 32 рисунки, 4 таблиці, список використаних джерел містить 24 найменування.

У бакалаврській дипломній роботі проведено детальний аналіз методів і засобів мультиплеєрної комп'ютерної гри «Tricky Squares» та існуючих аналогів. Описано об’єкт, предмет, завдання та методи дослідження. Сформульовано мету досліджень – покращення функціоналу 2D ігор за рахунок розробки логічної комп’ютерної гри, що дозволить в ігровому режимі тренувати стратегічні здібності користувача.

У бакалаврській роботі розроблено програмні засоби реалізації комп'ютерної гри «Tricky Squares». Розроблений застосунок, призначений для комп’ютерної системи, являє собою 2D гру, спрямовану на покращення навчальних інтерактивних програм за допомогою ігор, що допоможуть розвивати свої тактичні та стратегічні здібності.

Програмний застосунок розроблено з використанням мови C#, середовища розробки Microsoft Visual Studio 2022 та движка Unity.

У результаті виконання бакалаврської дипломної роботи отримано навчальну комп’ютерну систему, що підвищить ефективність когнітивних навичок та сприятиме розвитку логічних здібностей.

Отримані в бакалаврській дипломній роботі результати можна використати для покращення навчальних інтерактивних програм за допомогою ігор та впровадження в заклади освіти з метою розвитку когнітивних навичок, таких як концентрація, логіка, пам'ять та увага.

Ключові слова: тестування, комп’ютерна гра, мультиплеєр, гравці.

ABSTRACT

The bachelor thesis consists of 62 pages of A4 format, on which there are 32 figures, 4 tables, the list of used sources contains 24 names.

A detailed analysis of the methods and tools of the multiplayer computer game «Tricky Squares» and existing analogues was carried out in the bachelor thesis. The object, subject, tasks and research methods are described. The purpose of the research is formulated – to improve the functionality of 2D games by developing a logical computer game that will allow training the user's strategic abilities in the game mode.

In this bachelor's work, software tools for the implementation of the computer game «Tricky Squares» were developed. The developed application is intended for a computer system, is a 2D game aimed at improving educational interactive programs with the help of games that will help develop your tactical and strategic abilities.

The software application is developed using the C# language, the Microsoft Visual Studio 2022 development environment, and the Unity engine.

As a result of completing the bachelor thesis, an educational computer system was obtained, which will increase the effectiveness of cognitive skills and contribute to the development of logical abilities.

The results obtained in the bachelor's thesis can be used to improve educational interactive programs with the help of games and implementation in educational institutions, with the aim of developing cognitive skills such as concentration, logic, memory and attention.

Keywords: testing, computer game, multiplayer, players.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Що таке ігри? Граючи в ігри, ти відчуваєш, що досяг успіху, ти відчуваєш, що в грі ти досяг більшого ніж в реальному житті. Мозок людини влаштований так, що не важливо цей успіх є досягненнями в реальності чи в грі. Мозок виділятиме дози дофаміну в будь-якому разі за досягнення. Людство вже декілька сотень тисячоліть користувалось принципом – що приємно, те корисно. Бо людина при якихось досягненнях спрощувала собі життя, а що зараз, зараз не так легко отримати задоволення від навчання, якщо воно не є цікавим та наглядно демонструє де ці знання можна використати в житті. Люди грають в ігри, а не в життя, бо ігри дають відчуття успіху за короткий проміжок часу. Чому б нам не взяти за приклад шахи? Шахи популярні вже декілька тисячоліть і гравців до появи відеоігор не ставало менше, ще декілька десятків років тому весь світ слідкував за дебютами Бобі Фішера котрий переміг Бориса Спаського. У наш час десятки мільйонів слідкують за турнірами відеоігор [1].

Одні ігри перевершують інші, але не потрібно відкидати той факт, що ігри це великий клаптик нашого життя.

Людям потрібне щось на зразок тетрісу, тобто просте в освоєні, з приємним дизайном та водночас без зайвих елементів, що приноситиме задоволення від процесу, від власного швидкого успіху, від того що їхня стратегія в цій грі спрацювала. Люди хочуть отримувати задоволення від життя, то чому б не надати їм таку можливість?

Продукт має зацікавити тих, хто вже наситився від стандартних симуляторів ферм, ігор, що лише повторюють існуючі шаблони, та повсемісного грінду. Головна ціль – створити проект, який вирізняється гармонійним поєднанням усіх компонентів. Не надто складна графіка, але не архаїчна, відмінно працюючі механіки і приємний візуальний стиль. Гра не забирає багато часу, але надихає гравців повертатися до неї [2].

Наявні аналоги не надають, повністю або частково всього зазначеного функціоналу. Звичайно є більш кращі аналоги проте вони є платними, а інші мають ряд великих мінусів та недоліків. Натомість власна розробка різниться декількома перевагами: безкоштовністю, що робить її доступною для широкого кола гравців, можливістю мультиплеєру, що створює соціальну інтеракцію та сприяє формуванню спільноти гравців, привабливим ретро стилем, який викликає ностальгію та привертає увагу нових гравців, а також цікавим геймплеєм, який утримує увагу гравців і сприяє популярності гри. Тому актуальною є розробка власної комп'ютерної гри для десктопних систем.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є покращення розвитку логічного та стратегічного мислення за рахунок розробки і використання логічної комп’ютерної гри, що дозволить в ігровому режимі у багатокористувацькому форматі проводити тренування в середовищі гейміфікованої системи та поєднувати режим тренування з отриманням задоволення від використання комп’ютерної ігрової програми.

Задачі дослідження:

– провести аналіз існуючих методів і засобів розробки 2D ігор для визначення напрямків підвищення їх продуктивності;

– розробити метод ініціалізації системи для автоматизації ігрових процесів;

– розробити модель десктопного застосунку для тренування логічних здібностей, яка включатиме основний функціонал застосунку;

– програмно реалізувати функціонал застосунку;

– розробити зручний інтерфейс користувача, який буде легким у використанні;

– забезпечити підтримку української мови застосунку та доступність на різних платформах і пристроях;

– провести тестування розробленого застосунку.

Об’єкт дослідження – процес розробки багатокористувацької гри.

Предмет дослідження – методи та засоби реалізації комп’ютерної гри в мультиплеєрному режимі.

Методи дослідження. У дослідженні було використано методи:

– методи і технології модульної розробки застосунків для програмної реалізації модулів системи;

– методи UI/UX для розробки та тестування інтерфейсу застосунку;

– методи тестування для перевірки працездатності роботи програми.

Новизна отриманих результатів:

1. Подальшого розвитку набув метод ініціалізації системи, який, на відміну від існуючих, надає користувачеві можливість встановлювати значення елементів за замовчуванням шляхом відтворення збережених даних з попередньої ігрової сесії, що дозволяє автоматизувати процеси налаштування ігрової програми.

2. Подальшого розвитку набула модель комп’ютерної гри, яка, на відміну від існуючих, забезпечує реалізацію мультиплеєрного режиму з використанням функціоналу методу Awake й класу LobbyUI, що дозволяє реалізувати коректний розподіл доступу до ігрової сесії заявлених користувачів та зберігає налаштування їх ініціалізації в системі для автоматизованої підтримки даних у майбутніх ігрових сесіях.

Практична цінність отриманих результатів. Практична цінність полягає у тому, що розроблений ігровий програмний застосунок може використовуватися гравцями для розвитку свого логічного і стратегічного мислення у процесі проходження ігрової сесії. Запропонована програма сприяє розвитку когнітивних навичок, таких як концентрація, логіка, пам'ять та увага.

Програмні засоби, розроблені в роботі, можуть бути використані для впровадження їх у сферу освіти, яка потребує інтерактивних навчальних ігор.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій роботі[3], опублікованій у співавторстві, автору належать такі результати: користувацький інтерфейс, архітектура та розробка мультиплеєрної гри.

Апробація матеріалів бакалаврської дипломної роботи. Результати бакалаврської дипломної роботи доповідалися на конференції «LIII науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024)», Вінниця 2024.

Публікації. Результати дослідження були опубліковані в матеріалах LIII науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (2024) [3].

Структура та обсяг роботи. Бакалаврська дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, що містить 24 найменування, 4 додатків. Робота містить 32 ілюстрації та 4 таблиці.

1 АНАЛІЗ СУЧАСНОГО СТАНУ КОМП'ЮТЕРНИХ ІГОР У ЖАНРІ 2D ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану комп'ютерних ігор у жанрі 2D

На сьогоднішній день комп'ютерні ігри у жанрі 2D зберігають свою актуальність та привабливість для широкої аудиторії гравців. Хоча в останні роки домінуючими стали тривимірні ігри, 2D ігри відіграють значну роль в індустрії розваг та продовжують привертати увагу своєю унікальною естетикою та геймплеєм [4].

З одного боку, розвиток технологій дозволяє створювати вражаючі візуальні ефекти та деталізацію навіть у просторі 2D. Багато розробників активно використовують векторні та піксельні графічні движки для створення вражаючих образів та атмосфери в своїх проектах.

З іншого боку, 2D ігри мають деякі очевидні переваги, такі як простота розробки та доступність для широкого кола користувачів. Це дозволяє невеликим командам розробників створювати цікаві та захоплюючі ігри з меншими витратами на ресурси порівняно з тривимірними проектами [5].

Зростаюча популярність ретро-стилю та ностальгії серед гравців також сприяє популярності 2D ігор. Багато проектів пропонують унікальні графічні стилі, які натхнені класичними іграми минулих десятиліть, що дозволяє відчути смак ранніх епох в ігровій індустрії.

У цілому, комп'ютерні ігри у жанрі 2D залишаються живим та важливим аспектом геймінгової культури, привертаючи як ветеранів, які пам'ятають класичні проекти, так і нових гравців, які цінують їхню простоту, атмосферу та геймплейну механіку.

Отже, розробка програмних засобів комп'ютерної гри «Tricky Squares» не лише є важливою з точки зору особистого творчого виразу, але також має значний потенціал у сферах розваг, освіти та інновацій. Розробка також має потенціал об'єднати спільноту гравців, створити можливості для навчання та

розваги, а також принести нові технологічні рішення, впливаючи на мільйони гравців по всьому світу і відкриваючи шлях до нових творчих можливостей.

1.2 Порівняльний аналіз аналогів

Ринок комп’ютерних ігор пропонує велику кількість десктопних застосунків для різних типів користувачів. Розглянемо популярні додатки як аналоги розроблюваної мультиплеєрної комп'ютерної гри «Tricky Squares»: Tetris, Tricky Towers, Candy Crush.

Особливості гри Tetris:

«Tetris» – це ігра, що завойовала світову популярність завдяки своєму простому, але водночас захоплюючому геймплею та відмінному музичному супроводу. Основною метою гравця є складання геометричних фігур, які падають зверху донизу на ігрове поле. Гравець повинен розміщувати ці фігури так, щоб утворювалися горизонтальні ряди без прогалин. Коли такий ряд утворюється, він зникає, даючи гравцю очки, а вільні місця знизу заповнюються. Це просте завдання ускладнюється з часом, оскільки фігури починають падати швидше, і гравець повинен швидко приймати рішення та реагувати на зростаючий темп гри. Крім того, звуковий дизайн гри, який включає в себе звуки руху та падіння блоків, додає до загального досвіду гри і стимулює гравця до продовження гри [6].

Гра має простий, але визнаний усім світом візуальний стиль. Блоки мають яскраві кольори, що дозволяє гравцям легко розрізняти їх. Цей стиль сприяє швидкому відгуку та допомагає гравцям концентруватися (рисунок 1.1).

«Tetris» також славиться своєю доступністю на різних платформах, включаючи комп'ютери, консолі, мобільні телефони та інші пристрої. Це робить гру досить універсальною та доступною для широкого кола гравців.

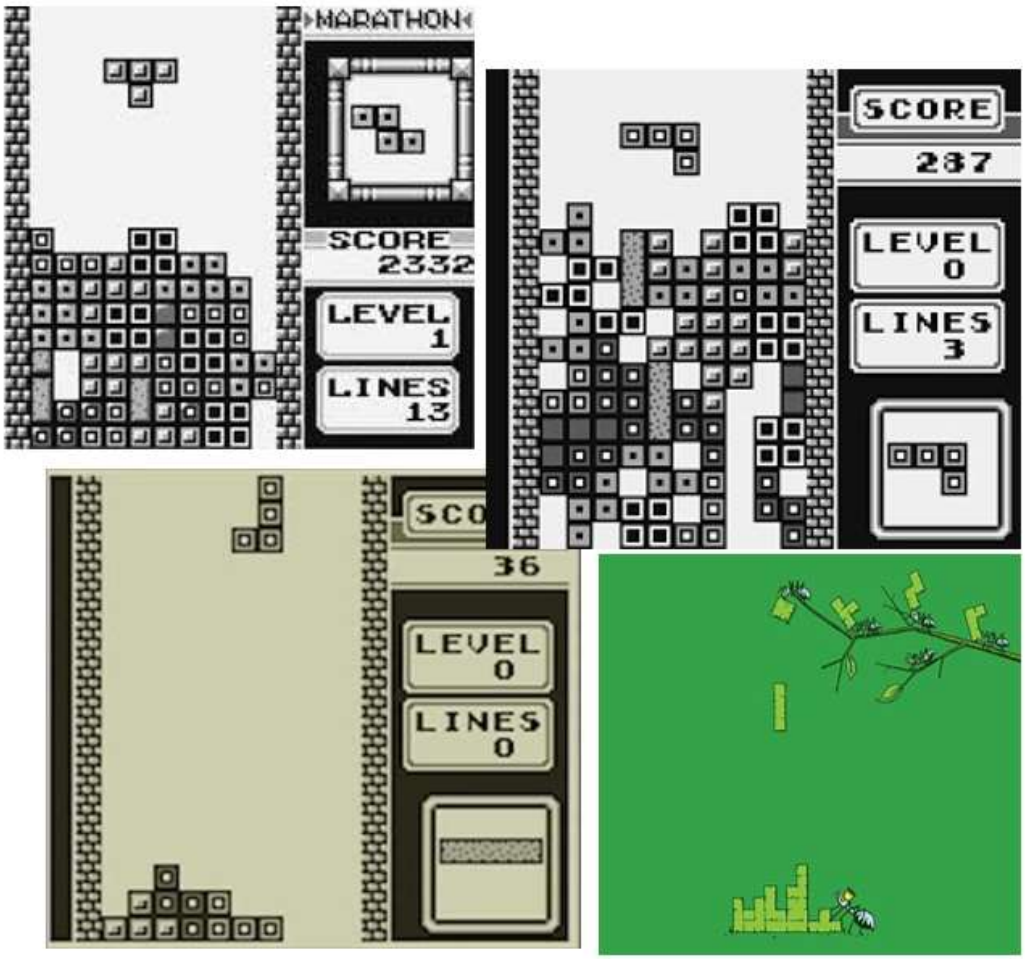

Рисунок 1.1 – Геймплей гри Tetris

Особливості гри Tricky Towers:

Данна гра не сильно відрізняється від попередньої, проте в цій грі фігури потрібно розташовувати(переплітати) так щоб вони утворювали вежу та не звалилися вниз, бо тепер на них працюють закони фізики [7].

Також у грі доступний мультиплеєр проте дії гравця не впливають прямим чином на дії суперника.

«Tricky Towers» – це весела гра-головоломка, яка поєднує елементи Tetris і фізики. Ця гра відрізняється від класичного Tetris своїм унікальним підходом до геймплею і інтеграцією різноманітних ефектів та взаємодії між блоками (рисунок 1.2).

Рисунок 1.2 – Геймплей гри Tricky Towers

У «Tricky Towers» гравці будують вежі з блоків, а не спробують заповнити ряди, як у Tetris. Але схожість полягає в тому, що блоки падають зверху вниз і гравці повинні розташовувати їх так, щоб створити стійку конструкцію. Однак, у цій грі фізика відіграє важливу роль: блоки можуть зрушуватися або впасти з вежі, якщо конструкція нестійка.

Крім того, «Tricky Towers» має різноманітні режими гри, включаючи змагання один на один або в команді, а також спеціальні завдання, які вимагають від гравців будувати вежі з обмеженим часом або змагатися за максимальну висоту вежі. Гра також має елементи випадковості, такі як спеціальні блоки або ефекти, які можуть змінювати хід гри і стратегію гравців.

Загалом, «Tricky Towers» відрізняється від «Tetris» своїм унікальним підходом до геймплею, додаванням фізичних ефектів і спеціальних режимів гри, що робить її цікавою і захоплюючою для гравців, які шукають щось нове і незвичне у жанрі головоломок.

Особливості гри Candy Crush:

«Candy Crush» надзвичайно популярна гра-головоломка жанру match-three(три в ряд). Гра має яскравий та привабливий дизайн, забарвлений кольоровими цукерками та іншими солодощами. Вона складається з різноманітних рівнів і головоломок, де гравцю потрібно співставляти різнокольорові цукерки у рядки або групи для їх знищення (рисунок 1.3).

Багато рівнів мають обмеження кількості ходів або часу, що додає елемент складності та вимагає стратегічного планування в кожному руху. Також гравці можуть отримувати бонуси та підсилення, які допомагають у пройденні рівня. Наприклад, вони можуть знищити всі цукерки одного кольору або перемішати їх на полі.

Розробники постійно додають нові рівні та функції, щоб гравці мали постійний інтерес до гри. Час від часу в грі проводяться спеціальні події або конкурси, під час яких гравці можуть виконувати бонусні завдання та отримувати унікальні винагороди [8].

Рисунок 1.3 – Геймплей гри Candy Crush

Завдяки своїй простій ігровій механіці, барвистій графіці та захоплюючій природі «Candy Crush» захопила мільйони гравців у всьому світі з моменту свого

випуску в 2012 році. У грі є різні рівні, бонуси та виклики, які залучають гравців та змушують їх повертаються за новими емоціями.

Tetris як і Tricky Towers навчає мислити швидко, але не тоді коли проти тебе грає людина. В цілому ігри такого типу зосередженні на прийнятті миттєвих рішень та тренують реакцію. Теж саме стосується й Candy Crush проте у цьому випадку тут зроблено акцент на візуальний стиль який надихає та привертає увагу нових гравців. У поєданні з цікавим геймплеєм Candy Crush також є свого роду діамантом серед інших ігор свого жанру.

Для наочної демонстрації відмінностей розглянутих додатків було зведено їх переваги і недоліки у таблицю порівняння (таблиця 1.1).

Таблиця 1.1 – Порівняльний аналіз аналогів

Критерії	Tetris	Tricky Towers	Candy Crush	Власна розробка «Tricky Squares»
Безкоштовність	1	0	1	1
Мультиплеєр	0	1	0	1
Цікавий геймплей	1	1	1	1
Ретро стиль	1	0	0	1
Кастомізація	0	1	0	0
Сумарний коефіцієнт	3	3	2	4

Згідно з порівняльним аналізом аналогів, доцільно розглянути власну розробку. Оновлений продукт буде спроможний вирішити недоліки існуючих рішень, а також забезпечити унікальний та захоплюючий досвід. Дослідження виявило потенціал для розвитку та вдосконалення продукту, що відкриває можливості для нових інновацій.

Отже, розглягувши виявленні недоліки та потенційні можливості існуючих рішень, які можна вирішити через впровадження нових функцій та покращень.

Отриманий продукт має потенціал покращити користувальницький досвід і привернути нових користувачів завдяки своїм унікальним можливостям. Додатково, це може сприяти збільшенню конкурентоспроможності на ринку через створення нового захоплюючого продукту, який відповідає потребам споживачів.

1.3 Аналіз методів розв’язання задачі

Розробка нової гри, яка поєднує найкращі аспекти розглянутих аналогів, вимагає аналізу різних методів розв’язання завдання. Враховуючи переваги і недоліки кожного продукту, можна визначити найбільш ефективний спосіб створення власного застосунку.

Хоча можна вдосконалювати ідеї, це не означає повне копіювання або використання наявних елементів гри. Це може спростити розробку, але може створити проблеми з інтеграцією в існуючі ігрові системи, що уповільнить процес і зменшить його ефективність. Більше того, такі продукти мають своїх власників, що може створити проблеми з авторськими правами. Крім того, цей підхід не сприяє значним інноваціям і не відповідає попиту користувачів. Найкращим рішенням є розробка унікального та повноцінного застосунку з нуля.

Ігровий процес застосунку представлений у вигляді ігрового поля на якому суперники намагатимуться розробити стратегію при заповненні полів прямокутниками з різними сторонами. Завдання гравця розташувати їх так щоб захопити якнайбільшу територію, не давши цього зробити супернику, гра закінчиться коли всі поля будуть заповнені або коли один гравець захопить більше ніж половину полів на ігровому полі.

Програмний продукт передбачений для десктопних систем, з можливим перенесенням на мобільні пристрої в майбутньому з метою створення кроссплею та розширення користувацької аудиторії. Основними засобами керування програмою є пристрої введення інформації, введення команд реалізовано за рахунок взаємодії з комп’ютерною мишкою та клавіатурою.

Передача даних буде відбуватись через підключення до інтернет мережі, що дозволить синхронізацію пристроїв в режимі онлайн гри. Зі сторони клієнта будуть приходити дії, які будуть оброблятись сервером та приходити у відповідь. Такий розподіл забезпечуватиме захист даних користувачів та не дозволить їм виконувати некоректні дії, які можуть привести до збою в роботі програми з мінімальним навантаженням на їх системи.

Після ретельного аналізу було вирішено створити власний ігровий продукт, який поєднує в собі найкращі аспекти ретро ігор, проте з унікальним підходом та інноваційними функціями. Це дозволить використовувати найцікавіші механіки цих ігор, але при цьому забезпечить створення унікального геймплею та враження для гравців.

1.4 Постановка задач бакалаврської дипломної роботи

Після оцінки переваг і недоліків існуючих аналогів схожих 2D ігор було встановлено низку завдань для створення власного застосунку:

- провести аналіз існуючих методів і засобів розробки 2D ігор для визначення напрямків підвищення їх продуктивності;
- розробити метод ініціалізації системи для автоматизації ігрових процесів;
- розробити модель десктопного застосунку для тренування логічних здібностей, яка включатиме основний функціонал застосунку;
- програмно реалізувати функціонал застосунку;
- розробити зручний інтерфейс користувача, який буде легким у використанні;
- забезпечити підтримку української мови застосунку та доступність на різних платформах і пристроях;
- провести тестування розробленого застосунку.

1.5 Висновки

У першому розділі було розглянуто сучасний стан комп'ютерних ігор у жанрі 2D. Після проведення порівняльного аналізу аналогів власної ігрової програми було встановлено, що ігри даного жанру користуються значною популярністю. Серед існуючих аналогів розглянуто популярні ігрові застосунки: Tetris, Tricky Towers, Candy Crush.

На основі виявлених недоліків функціонування аналогів було розглянуто можливості вдосконалення функціоналу застосунків, що підтвердило доцільність розробки власного програмного рішення. Було визначено завдання для створення власної ігрової комп’ютерної програми.

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ЗАСТОСУНКУ

2.1 Аналіз даних системи

Для забезпечення успішного функціонування застосунку потрібно дослідити та опрацювати головні аспекти та особливості гри включаючи механіки та правила гри.

У грі передбачений мультиплеєр на 4 гравця, де користувачі можуть грати одночасно по локальній мережі або онлайн. Також на вибір доступні режими гри такі як: кооперативний, змагальний чи командний.

Основними механіками є заповнення клітинок шляхом ставлення фігур, приєднання території, поворот фігур, видалення заповнених клітинок.

При підключенні до сервера дії гравців та стан ігрових полів синхронізуються. Що дає можливість грати один проти одного в реальному часі.

Інтерфейс користувача інтуїтивно зрозумілий. У грі присутня система підказок, гравець може отримати підказки при наведені на інструкцію користувача.

Також у грі наявна система рейтингу, відстеження інформації про поточний стан гри, очок та таймер у мультиплеєрному режимі, що доповнюють ігровий процес.

Ігрове поле – це матриця з клітинок, де кожна клітинка може бути захоплена. Розмір матриці зазвичай 20x20 або 36x36, проте гравці можуть вказати свою розмірність.

Елементами гри є набір 2-х кубиків вибраної гравцями розмірності, які визначають ширину та висоту фігури відповідно. Фігури можна повертати й приєднувати до своєї існуючої території. Всі фігури гравця мають обраний відповідний колір, що є унікальним.

Правила гри:

1. Гравці кидають два гральних кубика (в грі це відбувається за допомогою розробленої механіки чесного рандому).

2. Гравці створюють прямокутник зі сторонами, що згенеровані кубиками.

3. Прямокутник має бути приєднаний до існуючої території гравця.

4. Перший прямокутник гравця поміщається в обраний кут, кут суперника протилежний.

5. Якщо гравець не може поставити згенерований прямокутник через вичерпання місця, то він пропускає хід.

6. Гра рахується закінченою, коли весь простір поля заповнений.

7. Перемагає той гравець, у кого більша сумарна площа територій.

Вхідні дані є фундаментальним етапом у будь-якому проекті чи дослідженні. Вони визначають обсяг і якість інформації, доступної для подальшого аналізу та обробки. Отже, правильне формулювання та організація вхідних даних визначає успішність у вирішенні завдань.

2.2 Розробка методу ініціалізації системи для автоматизації ігрових процесів

Ключовою особливістю розроблюваної системи є надання користувачеві можливості встановлення значення елементів за замовчуванням, тобто відтворення збережених даних з попередньої ігрової сесії.

Метод Awake в класі LobbyUI використовується для ініціалізації різних компонентів та налаштувань перед початком роботи об'єкта класу.

Метод ініціалізації системи включає базовий функціонал:

1. Встановлення поточного екземпляру класу.

Встановлення поточного екземпляру класу як статичної змінну «Instance» за допомогою функції Instance = this.

2.Ініціалізація варіантів для випадаючого списку.

List<Dropdown.OptionData> options = new();

foreach (BaseModeOptionsUI block in m_modeUIs){

options.Add(new Dropdown.OptionData(block.DropdownName));}

Створюється список варіантів для випадаючого списку з об'єктів масиву «m_modeUIs».

3. Заповнення полів введення тексту із збережених налаштувань.

У поле введення тексту m_usernameField встановлюється значення імені гравця PrefsKey_LastPlayerName з глобальної змінної PlayerPrefs, якщо воно є. Далі поле введення тексту m_usernameField підписується на подію onSubmit, що спрацьовує, коли користувач закінчує введення тексту. У випадку спрацювання події в поле Username глобальної змінної NetworkManager встановлюється значення імені гравця, зберігається ім'я гравця у PlayerPrefs. І, якщо гравець онлайн, надсилає ім'я гравця іншим гравцям.

У поле введення тексту m_ipField встановлюється значення IP-адреси і порту хоста PrefsKey_LastUsedIP із глобальної змінної PlayerPrefs, якщо воно є, і значення «127.0.0.1:51473», якщо нема. Далі поле введення тексту m_ipField підписується на подію onSubmit, що спрацьовує, коли користувач закінчує введення тексту. У випадку спрацювання події значення поля m_ipField зберігається в глобальну змінну PlayerPrefs.

4. Налаштування перемикача m_isPublicLobbyToggle.

Перемикач m_isPublicLobbyToggle, що відповідає за те, чи буде лобі публічним, підписується на подію onValueChanged, яка спрацьовує, коли значення перемикача змінено. Якщо перемикач буде увімкнено, то лобі буде публічним, якщо вимкнено, то не буде публічним.

5. Налаштування випадаючого списку m_gameDropdown.

Випдаючий список m_gameDropdown очищуєтья методом ClearOptions, якщо там щось є. А потім додаються елементи з масиву options. Далі m_gameDropdown підписується на подію onValueChanged, що спрацьовує, коли вибирається значення випадаючого списку. Якщо спрацьовує, то виконується метод LoadUIByDropdown, який змінює елементи лобі залежно від значення.

6. Перевірка стану мережі та налаштування кнопок/елементів UI.

Якщо значення IsLocal(чи лобі локальне) в NetworkManager є true, то кнопка m_joinButton(кнопка для приєднання до лобі) стане неактивною.

Інакше, якщо значення NetworkManager.Client.IsConnected є true, текст кнопки m_joinButton зміниться на «Disconnect» і тоді кнопка буде відповідати за від'єднання від лобі. А також група елементів m_group і перемикач публічності лобі m_isPublicLobbyToggle стануть неактивними.

7. Підписки на події мережевого менеджера.

Поле Client публічної змінної NetworkManager підписується на події ConnectionFailed, що відповідає за випадок, коли з'єднання було не вдале, Connected, що спрацьовує, коли з'єднання було успішним і Disconnected, що спрацьовує, коли клієнт від'єднується від хоста. У випадку спрацювання подій виконуються методи Client_ConnectionFailed, Client_OnConnected і Client_OnDisconnected.

Публічна змінна ServerData підписується на подію OnClientChanges, що спрацьовує, коли в клієнті відбулися якісь зміни. В разі спрацювання події OnClientChanges буде виконано метод ServerData_OnTeamChanges.

Розроблений метод дозволяє зберігати налаштування ініціалізації користувачів у системі для автоматизованої підтримки даних в майбутніх ігрових сесіях.

2.3 Розробка моделі системи

Для кращого розуміння функціоналу та взаємодії компонентів аркадної гри «Tricky Squares» було вирішено розробити модель системи на прикладі UML діаграми варіантів використання.

На рисунку 2.1 подана UML діаграма варіантів використання для комп'ютерної гри «Tricky Squares». Згідно з цією діаграмою, першим кроком користувач запускає гру на своєму десктопному комп'ютері. Після цього користувач проходить авторизацію та має можливість обрати режим гри (одиночний або мультиплеєрний), переглянути свій профіль зі списком

досягнень та статистикою, а також налаштувати параметри гри (наприклад, звукові ефекти або роздільну здатність екрану). Після вибору режиму гри користувач має можливість розпочати гру або переглянути довідку щодо правил і управління грою. У разі вибору мультиплеєрного режиму гри, гравець може приєднатися до онлайн-матчу з іншими гравцями у лобі.

Ця діаграма відображає основні варіанти використання гри «Tricky Squares» та дозволяє краще зрозуміти її функціонал для десктопних систем.

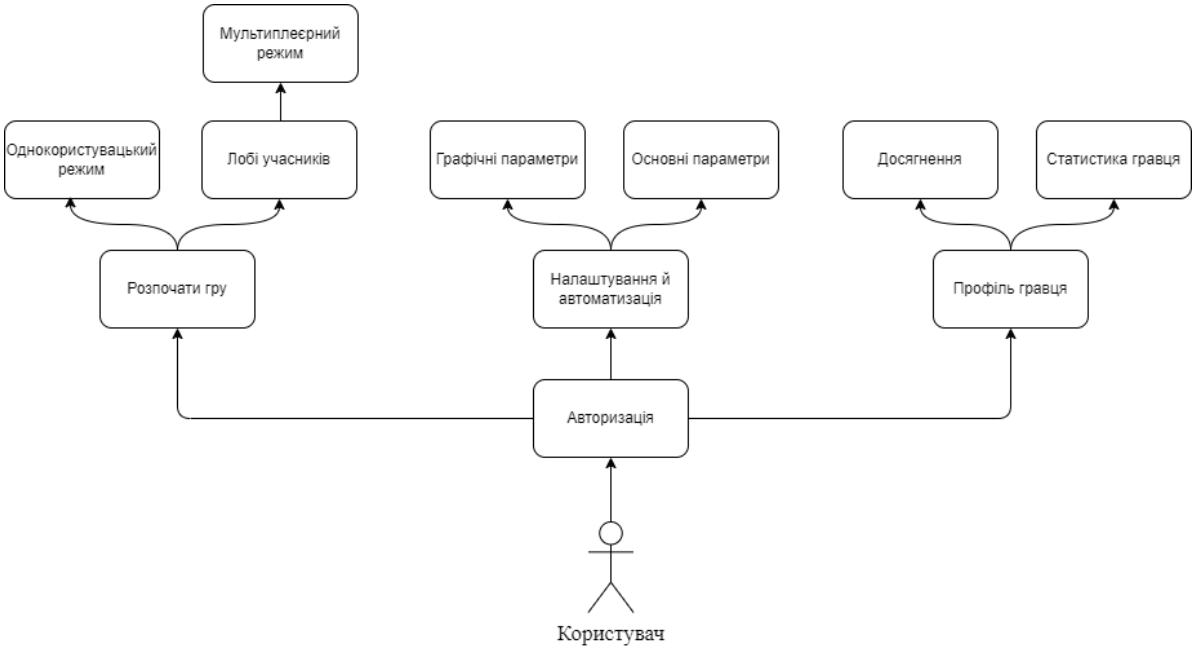

Рисунок 2.1 – Модель системи у вигляді діаграми варіантів використання

Діаграма варіантів використання – це інструмент аналізу, який допомагає визначити можливі шляхи використання продукту, послуги або системи з точки зору кінцевого користувача. Ця діаграма корисна для розробників, щоб зрозуміти потреби та очікування користувачів, а також для здійснення стратегічних рішень щодо подальшого розвитку продукту [9].

Розглянемо варіанти використання гри «Tricky Squares».

При вході в меню гри користувачі можуть авторизуватись, використовуючи свій логін та пароль для входу до додатку та доступу до особистого кабінету, або зайти у режимі гостя.

Після авторизації відкривається можливість зайти в налаштування, розпочати гру, чи зайти в особистий профіль, де можна переглянути інформацію про нього, його досягнення, статистику та інші дані.

Зайшовши в налаштування гри, користувач обирає основні параметри гри, такі як: мова, гучність, звукові ефекти, наявність підказок під час проходження, якість графіки. Або ж графічні параметри, де можна налаштовувати роздільну здатність, рівень деталізації, обсяг текстур.

Якщо користувач обрав варіант розпочати гру, то його перекидає у лобі учасників, де користувачі створюють або приєднуються до інших користувачів, де вони також можуть взаємодіяти з іншими перед початком гри.

Приєднавшись до мультиплеєрної гри, користувач взаємодіє з іншими учасниками в залежності від вибраного режиму через локальну мережу або онлайн. Гравці можуть вибрати однокористувацький режим, де вони грають самі проти штучного інтелекту, без участі інших гравців або інтернет-з'єднання.

Кожен гравець має особистий профіль, де можна переглянути інформацію про нього, його досягнення, статистику та інші дані.

При відкритті профілю гравці мають змогу переглянути статистику своїх звершень, таку як кількість перемог, програних ігор, час гри, а також досягнення за виконання певних завдань у грі.

2.4 Розробка алгоритмів роботи застосунку

Розробка загального алгоритму роботи системи гри «Tricky Squares» передбачає розробку різних компонентів, які були зображені та описані за допомогою діаграми послідовності.

Основні елементи діаграми послідовності включають:

1. Об'єкти – суб'єкти або елементи системи, які беруть участь у взаємодії. Кожен об'єкт подається у вигляді прямокутника з назвою.

2. Лінії життя – вертикальні лінії, які презентують часовий проміжок, під час якого об'єкт існує або активний. Лінія життя зазвичай пов'язана з об'єктом та позначає його «життєвий цикл».

3. Послідовність повідомлень – стрілки, які показують передачу повідомлень між об'єктами. Вони позначають порядок викликів або взаємодії між об'єктами.

4. Повідомлення – спосіб показу взаємодії між об'єктами, який включає ім'я повідомлення та, за необхідності, параметри.

5. Операції – це дії, які виконуються об'єктами в рамках взаємодії. Вони можуть бути позначені навпроти лінії життя та підписані назвою операції.

Діаграми послідовності часто використовуються для моделювання взаємодії в системах програмного забезпечення, таких як процеси бізнес-логіки, веб-застосунки та інші системи, де важлива послідовність подій та взаємодія об'єктів. Вони допомагають розібратися у складних сценаріях взаємодії та виявити потенційні проблеми або покращення у дизайні системи [10].

На рисунку 2.2 зображена діаграма послідовності.

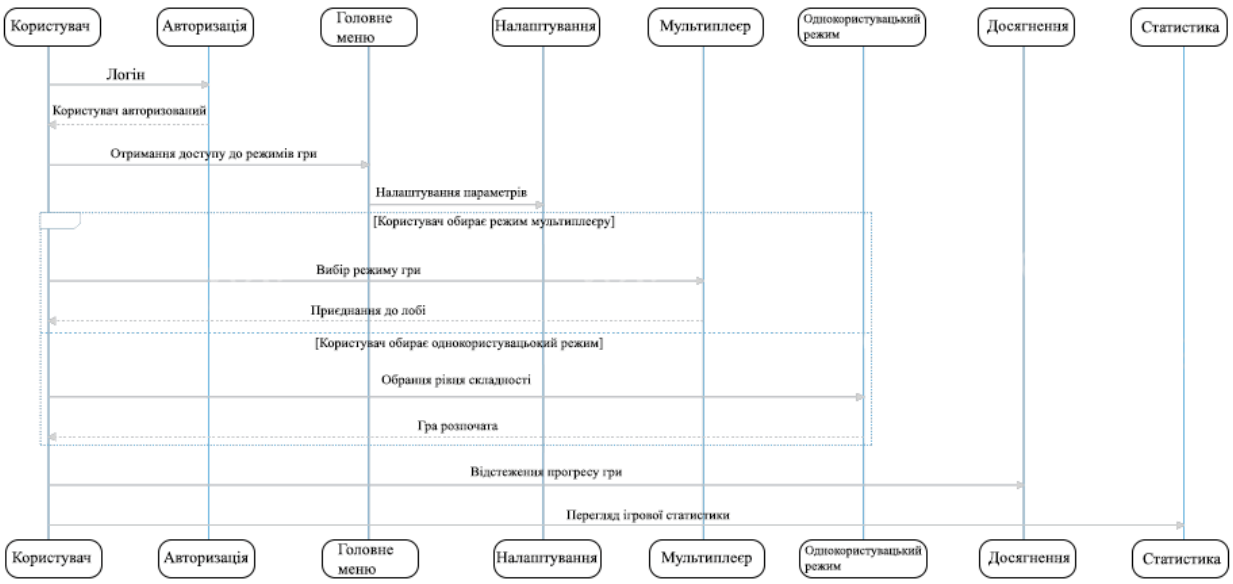

Рисунок 2.2 – Загальний алгоритм роботи системи зображений у вигляді діаграми послідовності

Діаграма послідовності є одним з видів UML-діаграм, який використовується для візуалізації послідовності взаємодії між об'єктами у системі впродовж певного часу. Це важливий інструмент для моделювання та аналізу взаємодії між компонентами системи та послідовності подій в рамках цієї взаємодії [11].

Діаграма послідовності для комп'ютерної гри «Tricky Squares» детально описує послідовність взаємодій між різними об'єктами та компонентами гри під час її виконання.

Опис діаграми послідовності:

1. Користувач заходить у гру здійснюючи авторизацію.

2. Перед користувачем відкривається можливість вибору режимів гри та входу в налаштування задля зміни параметрів.

3. Якщо користувач обрав вхід в налаштування, то йому відкривається меню параметрів, де він може встановити потрібні властивості після чого вийти з меню.

4. Користувач обирає режим гри у разі вибору мультиплеєрного режиму користувач приєднується у лобі гравців.

5. Якщо ж користувач обрав однокористувацький режим, то йому відкривається можливість вибору рівня складності.

6. Користувач розпочинає гру, відбувається взаємодія між об’єктами та гравцями.

7. Якщо гравець досяг перемоги або поразки, гра ініціює відображення результатів, щоб показати гравцеві результати його гри.

8. Під час гри збирається статистика яку гравець зможе переглянути після завершення гри.

Наведена на рисунку 2.2 діаграма послідовності допомагає розібратися у взаємодії між гравцем та самою грою під час різних етапів геймплею.

Завдяки діаграмі, наведеній на рисунку 2.2, можна визначити межі проєкту та зрозуміти, як працює система, не маючи додаткових знань у проєктуванні системної архітектури чи програмуванні засобів.

На рисунку 2.3 зображений алгоритм роботи методу ініціалізації системи для автоматизації ігрових процесів.

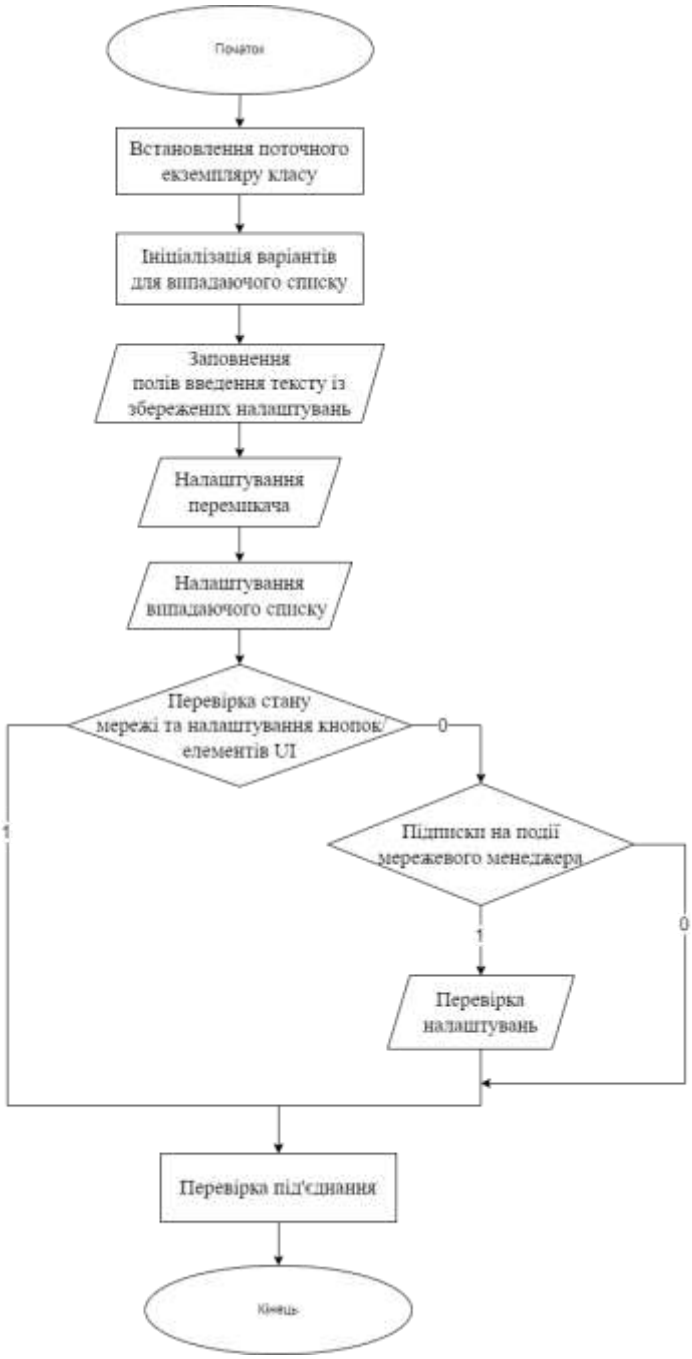

Рисунок 2.3 – Алгоритм роботи методу ініціалізації системи для автоматизації ігрових процесів

Метод надає користувачеві можливість встановлення значення елементів за замовчуванням, тобто відтворення збережених даних з попередньої ігрової сесії.

На рисунку 2.4 зображений алгоритм роботи мультиплеєра.

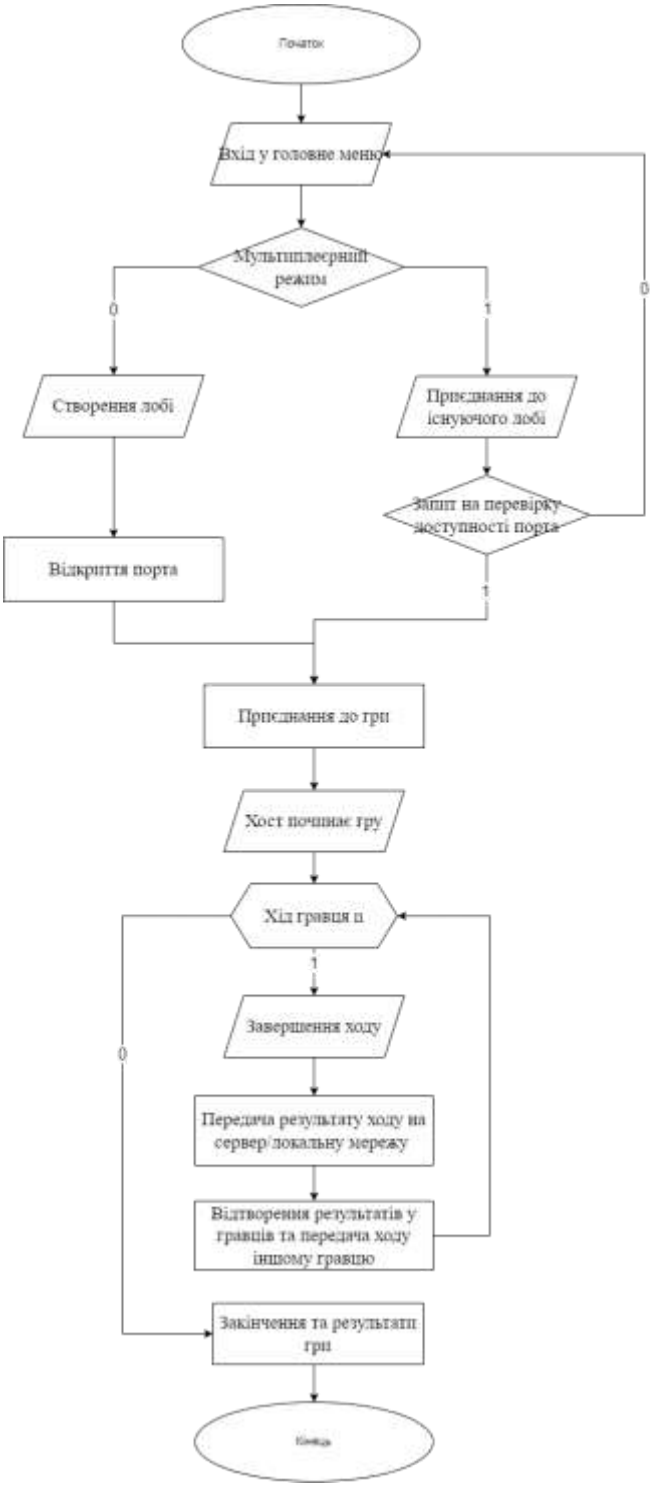

Рисунок 2.4 – Алгоритм роботи мультиплеєра

Мультиплеєр, що розрахований на 4 гравця, де користувачі можуть грати одночасно по локальній мережі або онлайн.

Користувачі розпочинають гру з приєднання до лобі інших гравців або створюють свою кімнату до якої можуть приєднатись інші.

Якщо користувач обрав публічне лобі (хост), то відкривається порт для інших гравців. У свою чергу коли гравець під’єднується до іншого лобі, то у нього спрацьовує запит на перевірку доступних місць у лобі. І якщо місця є, то гравця під’єднує до кімнати, якщо ж ні, то його викидає у головне меню де він може спробувати під’єднатись до тієї самої чи іншої кімнати або ж створити власну.

Після приєднання користувач який створював лобі тобто є хостом, розпочинає гру, розмірність карти, гранність гральних кубиків та інші параметри генеруються за налаштуваннями.

Гравці ходять по черзі, і заповнюють кількість клітинок, яка відповідає комбінації кубиків, що випали користувачу. Після кожного ходу інформація про результат надсилається на сервер з якого надходить гравцям або ж по локальній мережі передається на пристрої гравців. Після цієї операції, хід передається наступному гравцеві. Коли у гравця закінчуються ходи тобто можливість захоплювати територію, він пропускає хід. Гра закінчується коли всі клітинки поля заповнені, тобто коли жоден з гравців не може зробити хід.

2.5 Висновки

У другому розділі було здійснено аналіз даних системи. Описано розробку метода ініціалізації системи. Розроблена загальна модель системи, яка подана UML діаграмою варіантів використання.

На основі проведеного аналізу було розроблено діаграму послідовностей роботи системи.

3. РОЗРОБКА МОДУЛІВ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Вибір мови програмування важливий, оскільки він визначає ефективність розробки, продуктивність, масштабованість проекту та здатність до інтеграції з існуючими технологіями. Кожна мова має свої унікальні характеристики, які роблять її більш або менш підходящою для конкретного завдання, і правильний вибір допомагає забезпечити успішну реалізацію проекту.

C# – це мова програмування, яка посідає визначне місце у світі розробки програмного забезпечення. Вона привертає увагу розробників з усього світу завдяки своїй потужності, універсальності та широкому спектру застосування. Це одна з основних мов програмування, яка використовується в середовищі Unity. Вона має простий синтаксис, що полегшує розробку, а також вона є потужною інструментальною мовою з багатьма корисними функціями, такими як керування пам'яттю, обробка винятків, створення потоків тощо [12].

Використання цієї мови дозволяє реалізацію основних елементів гри на движку Unity, який підтримує всі актуальні ігрові платформи та є достатньо легким в освоєнні для нових розробників. Він містить у собі елемент візуального редагування інтерактивних об’єктів, що дозволяє наочно побачити й відредагувати поведінку ігрових елементів, їх позиціонування в самому середовищі, що значно спрощує роботу з ними [13].

Перевага мови програмування C# в тому, що більшість написаних програм будуть працювати на різних операційних системах, на відмінну інших мов програмування. Саме через цю та інші не менш вагомі переваги, що наведені в таблиці, мову програмування C# було обрано для реалізації поставленого завдання (таблиця 3.1).

Таблиця 3.1 – Порівняльна таблиця мов програмування

Мова Програмування	C#	Java	C++	JavaScript	Python
Об'єктно-орієнтовна	1	1	1	1	1
Багатопоточність	1	1	1	0	0
Прибирач сміття	1	1	0	1	1
Кросплатформність	1	1	1	1	1
Узагальнене програмування	1	0	1	1	0
Сумарний коефіцієнт	5	4	4	4	3

Об’єктно-орієнтовна мова програмування – мова, побудована на принципах об'єктно-орієнтованого програмування. В основі концепції об'єктно-орієнтованого програмування лежить поняття об'єкта – певної сутності, яка об'єднує в собі поля (дані) і методи (виконувані об'єктом дії).

Багатопоточність це можливість використовувати декілька потоків компіляції на багатопроцесорних системах для прискорення процесу компіляції програм.

Прибирач сміття це автоматичний процес, який дозволяє звільнити непотрібну пам'ять, яку використовує програма, включаючи оперативну пам'ять.

Кросплатформність це властивість програмного забезпечення працювати на різних операційних або апаратних платформах. Технології кросплатформності дозволяють ефективно зменшити витрати на розробку та адаптацію програм.

Узагальнене програмування це парадигма програмування, яка полягає в описі алгоритмів і даних, які можуть бути застосовані до різних типів даних, змінюючи лише сам опис, і підтримується різними мовами програмування.

При подальшій розробці можливе використання й інших мов. Наприклад, для реалізації серверної частини може бути обрана мова PHP, яка є однією з

найкращих для передачі та обробки даних між клієнтами. Також можливе використання інших специфічних для кожної платформи бібліотек, що дозволить продукту бути максимально гнучким та ефективним.

Використання движка Unity має безліч переваг, особливо для розробки ігор та інтерактивних додатків. Ось кілька ключових переваг:

Кросплатформеність. Unity підтримує багато платформ, включаючи Windows, MacOS, Android, iOS, консолі PlayStation, Xbox, Nintendo Switch та інші. Це дозволяє розробникам легко створювати проекти, які працюють на різних пристроях без значних змін у коді.

Широкі можливості: Unity має потужну інтегровану середу розробки (IDE), яка включає в себе інструменти для редагування графіки, анімації, фізики, звуку та багато іншого. Це дозволяє розробникам працювати над усіма аспектами свого проекту в одній програмі.

Спільнота та ресурси. Unity має велику та активну спільноту розробників. Існує безліч онлайн-ресурсів, таких як документація, форуми, туторіали та відеоуроки, які допомагають початківцям і досвідченим розробникам розвиватися та вирішувати проблеми.

Швидкість розробки. Unity пропонує широкий набір готових компонентів і ресурсів, які дозволяють розробникам прискорити процес розробки. Також наявність магазину активів (Asset Store) дозволяє швидко знаходити й використовувати готові рішення для різних задач.

Можливості для розширення. Unity має розширювану архітектуру, що дозволяє розробникам створювати власні плагіни та розширення для вирішення конкретних завдань чи оптимізації робочого процесу.

Усі ці переваги роблять Unity одним з найпопулярніших та потужних інструментів для розробки ігор та інтерактивних додатків [14].

Вибір середовища розробки має велике значення, оскільки від нього залежить продуктивність, зручність та ефективність вашої роботи. Правильно обране середовище забезпечить вам необхідні інструменти, зручний інтерфейс

та підтримку для потрібних технологій, що сприятиме якісному розвитку вашого проекту та загальній задоволеності від процесу програмування.

Visual Studio містить більшість необхідних бібліотек та компонентів для створення проєктів на вибраному движку, надає зручний та гнучкий інтерфейс, підтримує Windows – основну систему, на якій відбувається розробка продукту, а також C#, останню версію, на якій й створено код. Крім того, дане середовище надає змогу архівації коду для подальшого перенесення для запуску чи тестування на іншу систему з усіма використаними компонентами та модулями. Це означає, що кінцевому користувачу не потрібно встановлювати середовище для запуску та коректної роботи програми на його операційній системі [15].

Отже, для розробки використано мову програмування C# та платформу Visual Studio, які забезпечили ефективну інтеграцію з движком Unity та зручний розробницький інтерфейс. Також обговорено можливості подальшої розробки, включаючи використання інших мов програмування та бібліотек для забезпечення максимальної гнучкості та ефективності програмного продукту.

3.2 Розробка структури та функцій системи

Для реалізації комп'ютерної гри «Tricky Squares» потрібно описати основні функції, які забезпечать повноцінне функціонування геймплейю та іммерсію гравців.

Реалізація функціоналу аркадних ігор включає в себе низку ключових елементів, спрямованих на створення захопливого і веселого геймплею. Це включає в себе реалізацію різноманітних рівнів зі зростаючою складністю, можливість збільшення швидкості та складності гри з часом, систему бонусів та винагород для гравців, а також механіки управління, які роблять гру інтуїтивно зрозумілою і цікавою. Крім того, ефективна реалізація передбачає наявність звукового супроводу, що підкреслює динаміку подій в грі та створює належну атмосферу, яка заохочує гравця продовжувати грати і досліджувати світ гри [16].

Функціонал системи є критично важливим етапом в розробці будь-якого програмного продукту чи інформаційної системи. Цей аспект визначає основні можливості та функції системи, які будуть доступні користувачам. Він служить як важлива основа для розробки та тестування програмного забезпечення.

Опис функціоналу повинен бути ясним, конкретним і повним. Він має включати усі основні можливості, які система надає користувачам, від основних операцій до додаткових функцій. Крім того, важливо відзначити обмеження та вимоги до системи, які можуть впливати на її функціональність. Опис функціоналу також відіграє важливу роль у визначенні обсягу робіт, розробці тестових планів і визначенні вартості проекту. Чіткий та повний опис функціоналу дозволяє розробникам розуміти, як система повинна працювати. Без належного опису функціоналу ризик неповного розуміння потреб користувачів і невідповідності їх очікуванням значно зростає, що може призвести до невдачі проекту. Таким чином, важливість опису функціоналу системи полягає у забезпеченні ясності, узгодженості та успішності проекту в цілому [17].

Функціонал програми забезпечує:

- можливість провести час, незалежно від місця та часу;
- можливість налаштовувати параметри гри;
- можливість грати в режимі офлайн проти штучного інтелекту;
- можливість грати в мультиплеєрі з підключенням до серверу чи локальної мережі.

Розглянемо додатковий функціонал ігрової програми:

1. Авторизація користувачів через SQLite. Система забезпечує безпечний та зручний процес входу користувачів за допомогою сервісу SQLite, що гарантує надійність та захист особистих даних.

2. Збереження прогресу гравця у вигляді історії ігрових сесій та їх результатів.

Інтеграція сервісу SQLite дозволяє ефективно зберігати дані про прогрес гри користувачів, їх досягнення та статистику, що дозволяє користувачам повертатися до гри з будь-якого пристрою та відстежувати свій прогрес.

3. Перегляд та аналіз статистики гри. Система автоматично збирає дані про активність користувачів у грі та надає можливість оглянути власні успіхи та поразки для оцінки ефективності.

Модулі в розробці програмного забезпечення відіграють важливу роль розділяючи функціональність програми на логічні блоки для кращої організації та підтримки. Вони сприяють зручній організації коду, покращують його читабельність та зрозумілість, спрощують внесення змін та розширення функціоналу [18].

Модулі програми включають:

- графічний інтерфейс (GUI), який відповідає за візуальне відображення гри для користувачів, включаючи розміщення елементів інтерфейсу, графічний дизайн та анімацію;
- API (Application Programming Interface) – забезпечує зв'язок між фронтендом та бекендом, включаючи обробку запитів від користувачів та передачу даних між фронтендом і базою даних;
- базу даних (DB) – модуль, який відповідає за інтеграцію з сервісом SQLite, забезпечує передачу та збереження даних користувачів у SQLite, а також взаємодію з аутентифікацією, відправку повідомлень та керування користувачами;
- Business-logic – модуль, який містить основну логіку гри, алгоритми обробки даних та взаємодію з користувачем, відповідає за виконання основних функцій гри, розподіл завдань та обробку дій користувачів.

Важливим аспектом у розробці застосунку є UML діаграми. Найдоцільніше показати структури моделі в UML можна за допомогою діаграми класів зображеної на рисунку 3.1.

Структурна діаграма класів використовується для подання статичної структури моделі системи. Під час проєктування системи визначають ряд класів і групують їх на діаграмі класів, яка допомагає визначити статичні зв'язки між ними. При детальному моделюванні класи концептуального проєкту часто розбиваються на підкласи [19].

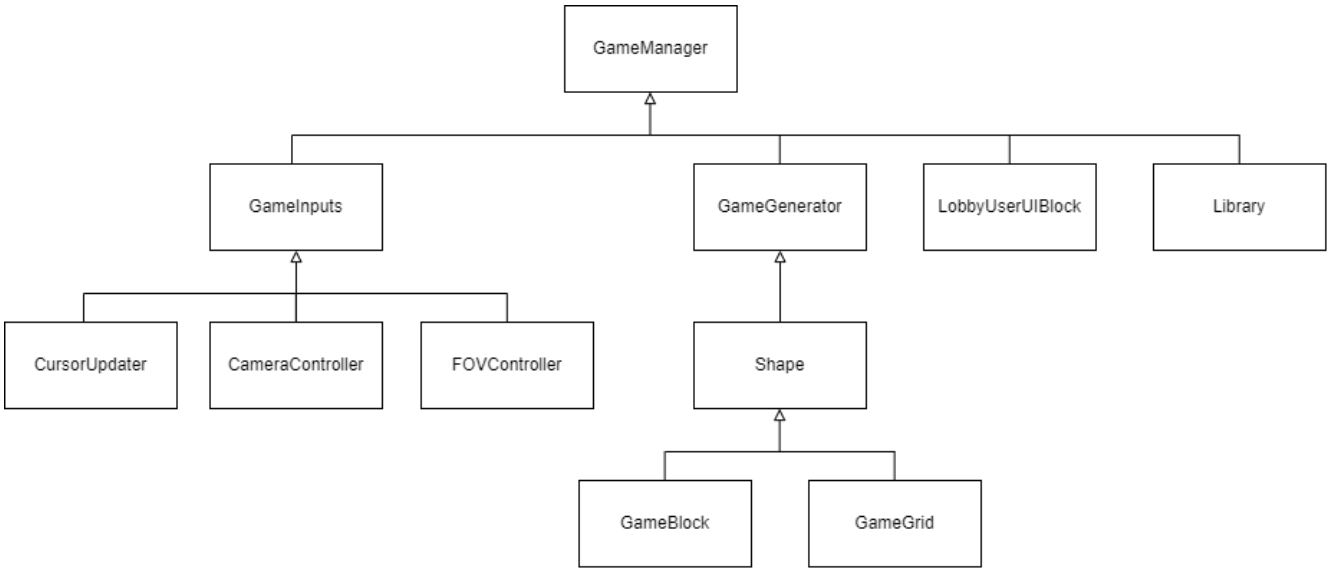

Рисунок 3.1 – Структурна діаграма класів

Структурна діаграма класів без опису атрибутів та особливостей зв’язків також є структурною схемою проекту. Діаграма класів описує модель предметної області, у ній присутні тільки класи прикладних об'єктів, використовується для управління процесом мислення, розуміння та потребує концептуальної цілісності. Також діаграма застосовується при проектуванні інформаційних систем і призначена для подальшої розробки моделей реалізації; головна її вимога це зрозумілість.

Загальний аналіз функціональності системи дозволяє зрозуміти її можливості та обмеження. Розглянутий функціонал програмного продукту охоплює широкий спектр можливостей для користувачів, включаючи режими гри та збереження прогресу. Подальше вдосконалення функцій системи може сприяти покращенню якості та роботи продукту.

3.3 Розробка модуля графічного інтерфейсу

Модуль графічного інтерфейсу (GUI) – це програмний компонент, призначений для створення і управління графічним інтерфейсом користувача у програмах. GUI надає користувачеві зручний спосіб взаємодії з програмою через візуальні елементи, такі як вікна, кнопки, поля введення тощо [20].

У програмуванні модуль GUI зазвичай містить набір класів і функцій, які дозволяють створювати візуальні елементи та обробляти події користувача, такі як натискання кнопок або введення тексту.

GUI дозволяє користувачам легко взаємодіяти з програмами, що робить їх більш доступними та зрозумілими для широкого кола користувачів. Відповідно, використання модулів GUI є важливою частиною розробки багатьох програм, особливо тих, які мають велику кількість користувачів або призначені для непрофесійних користувачів.

На рисунку 3.2 зображено клас LobbyUI.

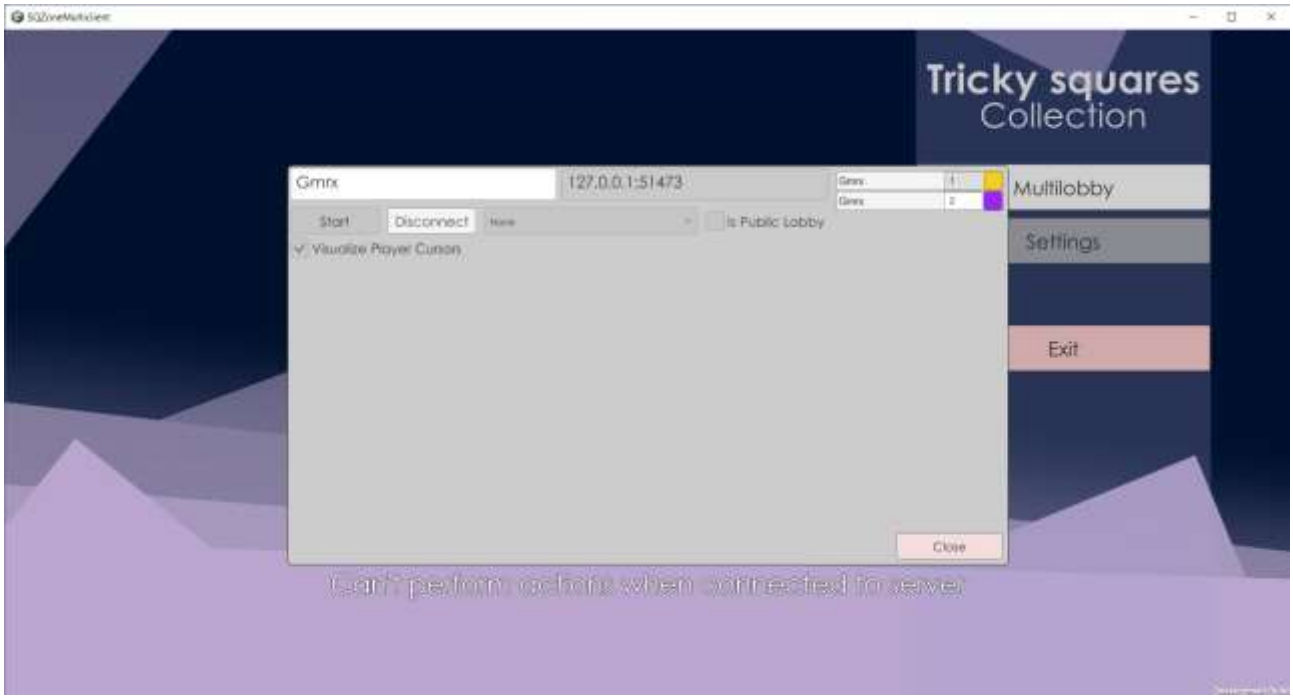

Рисунок 3.2 – Клас LobbyUI

Клас LobbyUI є ключовим компонентом управління інтерфейсом в програмі. Він відповідає за відображення та взаємодію з елементами, які користувач бачить у лобі гри. Основна мета використання цього класу – забезпечити зручний та ефективний спосіб навігації для користувача та подання необхідної інформації про доступні опції та функціонал.

Метод Awake викликається під час першого запуску скрипту і задає потрібні параметри графічному інтерфейсу за замовчанням. Задає текст для текстових полів, вказує останній використаний IP опонента, задає пункти випадаючих списків та налаштовує мережеві параметри (рисунок 3.3).

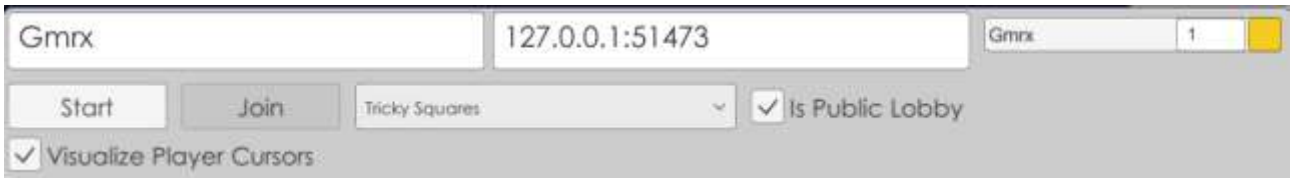

Рисунок 3.3 – Метод Awake

Метод LoadUIByDropdown змінює інтерфейс після вибору значення випадаючого списку (рисунок 3.4).

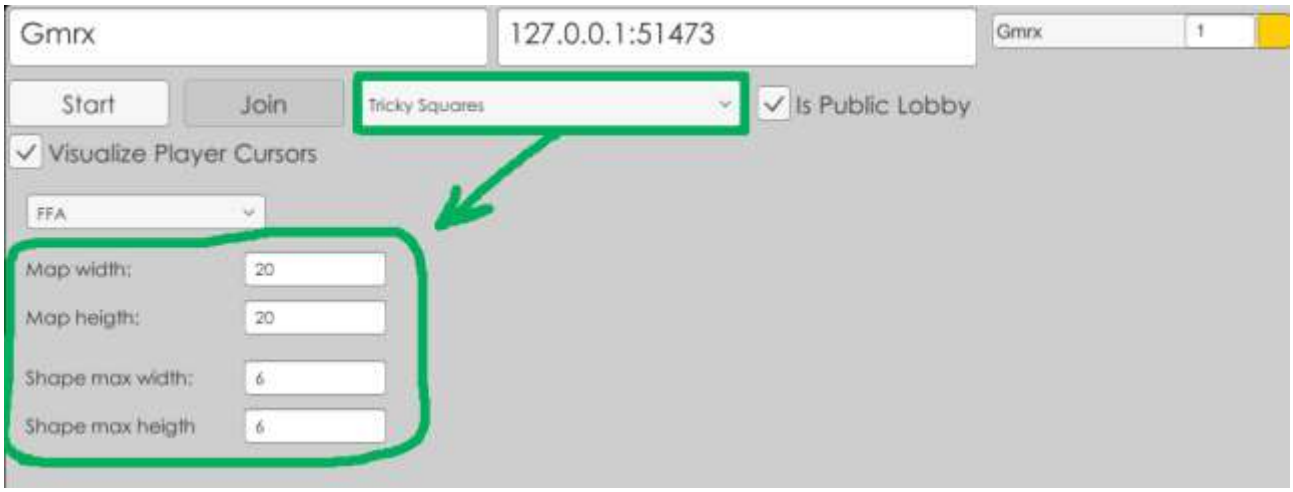

Рисунок 3.4 – Метод LoadUIByDropdown

Метод HostClicked відповідає за створення хоста у програмі, коли користувач натискає на відповідну кнопку у інтерфейсі. Цей метод включає в себе низку кроків, які необхідні для ініціалізації та налаштування нового хоста для гри. Метод перевіряє наявність доступних ресурсів та вільних слотів для нового хоста. Якщо такі ресурси є, він ініціює створення нового хоста, встановлюючи всі необхідні параметри, такі як налаштування гри, кількість гравців та інші опції.

Після цього метод HostClicked може ініціювати низку додаткових дій, таких як створення унікального ідентифікатора для хоста, реєстрація його в системі, оновлення інтерфейсу для відображення нового хоста та повідомлення інших користувачів про його наявність. У цілому, метод HostClicked відіграє ключову роль у процесі створення нового хоста для гри, забезпечуючи його належну ініціалізацію та налаштування перед початком гри (рисунок 3.5).

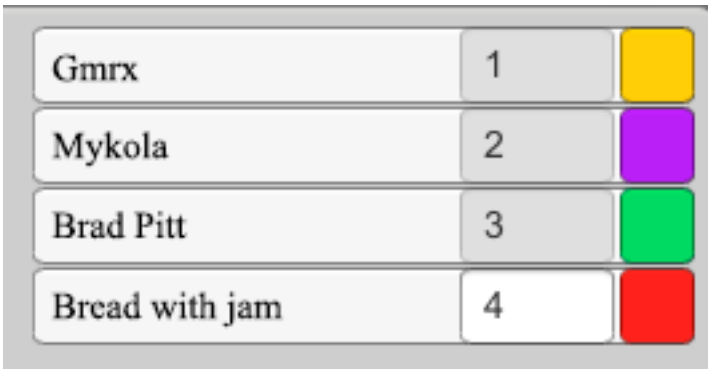

Рисунок 3.5 – Метод HostClicked

Метод JoinClicked відповідає за процес приєднання користувача до вже існуючого хоста у мережі, коли гравець натискає на відповідну кнопку в інтерфейсі. Цей метод включає в себе кілька кроків, які необхідно виконати для успішного приєднання до хоста. Він перевіряє доступність обраного хоста та його поточний статус. Якщо хост доступний і готовий приймати нових гравців, метод продовжує процес приєднання.

Після цього метод може виконати ряд додаткових дій, таких як запит на інформацію про гру від хоста, отримання необхідних налаштувань та параметрів гри, а також оновлення інтерфейсу для відображення актуальної інформації про гру та інших гравців, які вже приєдналися. Метод JoinClicked реалізує приєднання користувача до гри, забезпечуючи взаємодію з вже існуючим хостом та підготовку до спільної гри з іншими гравцями (рисунок 3.6).

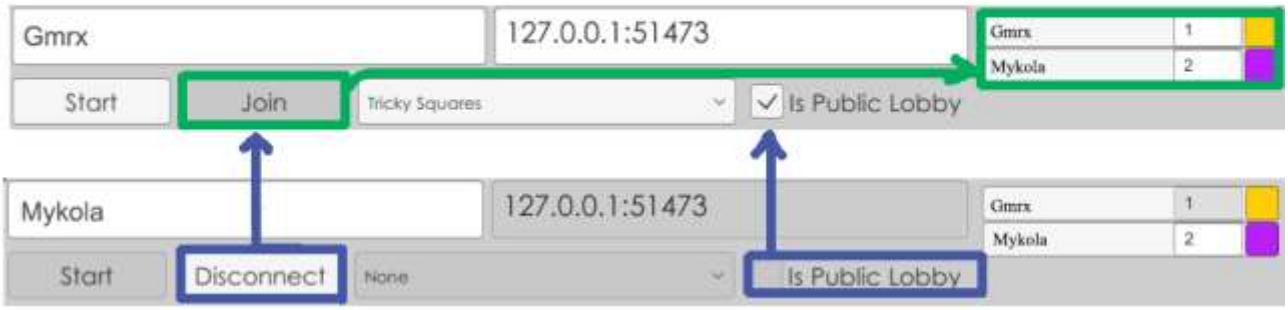

Рисунок 3.6 – Метод JoinClicked

Метод Client_OnConnected налаштовує параметри, коли клієнт приєднується до хоста. Метод Client_OnDisconnected налаштовує параметри, коли клієнт від’єднується від хоста. Метод Client_ConnectionFailed налаштовує параметри, коли клієнт не зміг під'єднатися.

Метод UpdateUsernames і метод UpdateUsername оновлюють інформацію про імена гравців для певних елементів інтерфейсу лобі.

Проведено детальну роботу з проєктування та імплементації інтерфейсу користувача для взаємодії з програмним продуктом. Процес розробки модуля графічного інтерфейсу передбачав розробку дизайну інтерфейсу та його реалізацію. А саме створення компонентів, стилізацію елементів і встановлення їх відповідно до дизайну. Отриманий результат готовий до подальшої інтеграції з загальною структурою програмного продукту.

3.4 Розробка модуля логіки застосунку

Business-logic або модуль логіки це частина програмного забезпечення, яка відповідає за логічне функціонування програми або системи. В залежності від конкретного застосування, цей модуль може включати в себе різні функції, такі як обробка введених даних, прийняття рішень на основі цих даних, керування потоком програми та забезпечення взаємодії з іншими модулями чи компонентами системи. У практичному програмуванні цей термін може мати різні інтерпретації залежно від контексту та конкретної технології.

Клас GameGrid є центральним елементом управління головною логікою гри і забезпечує ефективну реалізацію правил та обмежень, необхідних для правильного захоплення території у грі. Він відповідає за управління ігровим полем, на якому відбувається вся дія гри (рисунок 3.7).

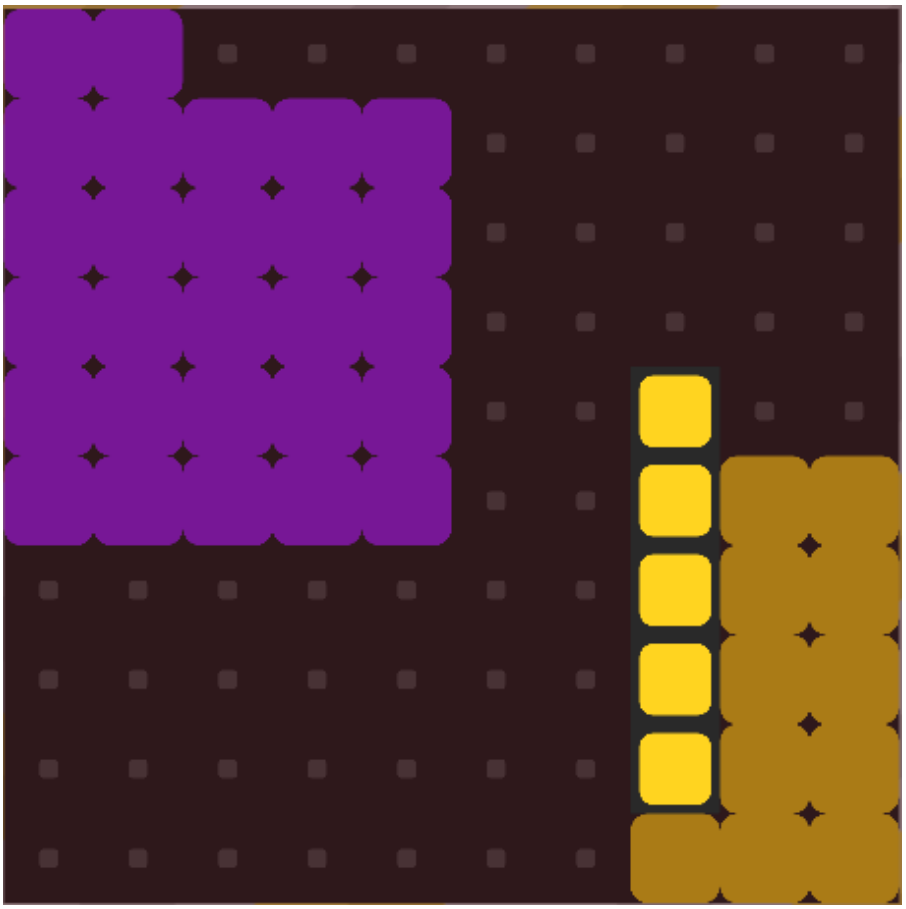

Рисунок 3.7 – Клас GameGrid

Метод ValidateFillZone перевіряє чи зайняті клітинки у вибраних межах. Метод отримує на вхід вибрані межі в якій користувач планує розмістити свою територію. Він перевіряє, чи всі клітинки в цій зоні вже зайняті іншими об'єктами чи діями. Це допомагає уникнути ситуацій, коли об'єкти перекриваються або займають одну і ту ж позицію.

Він повертає значення true, якщо всі клітинки у межах вільні і доступні для використання, і false, якщо хоча б одна клітина вже зайнята.

У підсумку, метод ValidateFillZone забезпечує коректність дій гравців та уникнення конфліктів у розміщенні об'єктів на ігровому полі. На рисунку 3.8 зображена ситуація коли гравець 1 хоче розмістити фігуру на територію яка вже зайнята гравцем, проте не може цього зробити оскільки метод ValidateFillZone взаємодіє з методом ValidateContacts та методом IsCellOccupied.

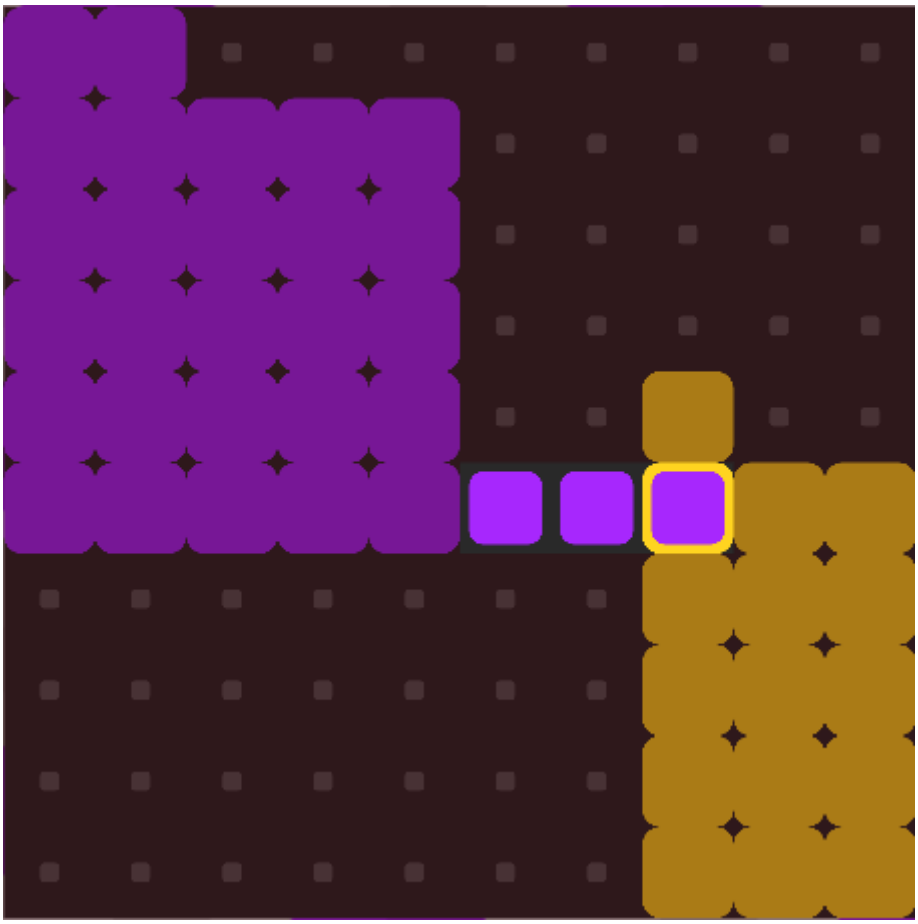

Рисунок 3.8 – Метод ValidateFillZone

Метод ValidateContacts перевіряє правильність зіткнення країв фігур на ігровому полі. Це ключовий етап, який допомагає уникнути ситуацій конфлікту між об'єктами та забезпечує коректність геймплею.

Метод отримує на вхід інформацію про краї фігур, які можуть зіткнутися під час ходу гравця. Він аналізує цю інформацію, перевіряючи, чи відбулося зіткнення країв, та визначає, чи є воно правильним з точки зору правил гри.

А метод IsCellOccupied перевіряє, чи є певна клітинка на ігровому полі вже зайнятою територією.

Метод Discard дозволяє гравцям скасувати виділення території у випадках, коли це необхідно або вони передумали щодо свого рішення. Коли гравець натискає кнопку відміни спрацьовує метод Discard і ініціює процес скасування виділення території, яку він раніше обрав. Цей процес включає в себе зняття заповнення клітинки та зняття підсвічування з клітинок, які раніше були виділені гравцем.

На рисунку 3.9 зображено метод SendShapeSubmit.

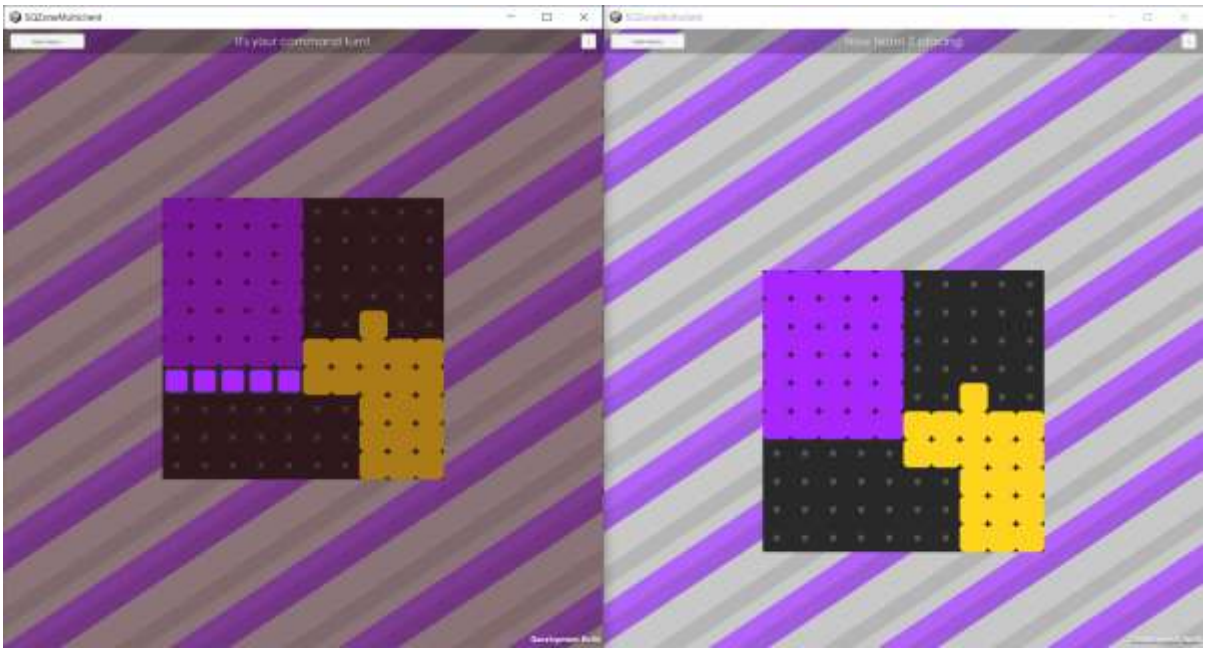

Рисунок 3.9 – Метод SendShapeSubmit

Метод SendShapeSubmit надсилає супротивнику інформацію про територію, яку вибрав гравець, щоб той мав змогу бачити, яка конкретна зона була вибрана. Коли гравець закінчує вибір території, метод надсилає дані про виділену зону іншому гравцеві. Це включає в себе передачу інформації про координати та форму виділеної зони. На рисунку 3.9 зображено початковий етап методу, сам процес появи нових фігур у користувачів здійснюється за допомогою анімації яку потрібно подавати у вигляді багатьох фреймів.

Було розроблено та впроваджено необхідні алгоритми для забезпечення функціональності програмного застосунку. Ці алгоритми мають оптимальну швидкодію та надійність. Розроблений модуль логіки успішно інтегрується з іншими компонентами програмного застосунку, що забезпечує цілісність та працездатність системи.

3.5 Розробка структури та інтерфейсу застосунку

Розробка структури інтерфейсу користувача (UI) є важливим етапом у створенні програмного забезпечення. Вона передбачає планування і проектування розташування елементів інтерфейсу, що забезпечують зручність використання програми. Структура UI включає компоненти, такі як навігаційні панелі, кнопки, форми, текстові поля, таблиці та графічні елементи. Основною метою є забезпечення легкого доступу до функціоналу додатку та покращення користувацького досвіду.

На початку розробки структури UI важливо провести дослідження цільової аудиторії для визначення її потреб та уподобань. Після цього створюються прототипи та макети, які демонструють розташування та взаємодію елементів. Важливо дотримуватися принципів дизайну, таких як консистентність, простота та доступність. Крім того, слід забезпечити адаптивність інтерфейсу для різних пристроїв і розмірів екранів. Регулярне тестування та зворотній зв'язок від користувачів допомагають вдосконалювати структуру та функціональність інтерфейсу, забезпечуючи високий рівень задоволеності користувачів.

Складемо списки екранів застосунку:

1 Головне меню.

2 Лобі гри.

3 Геймплей.

На рисунку 3.10 наведено схему користувацького інтерфейсу головного меню.

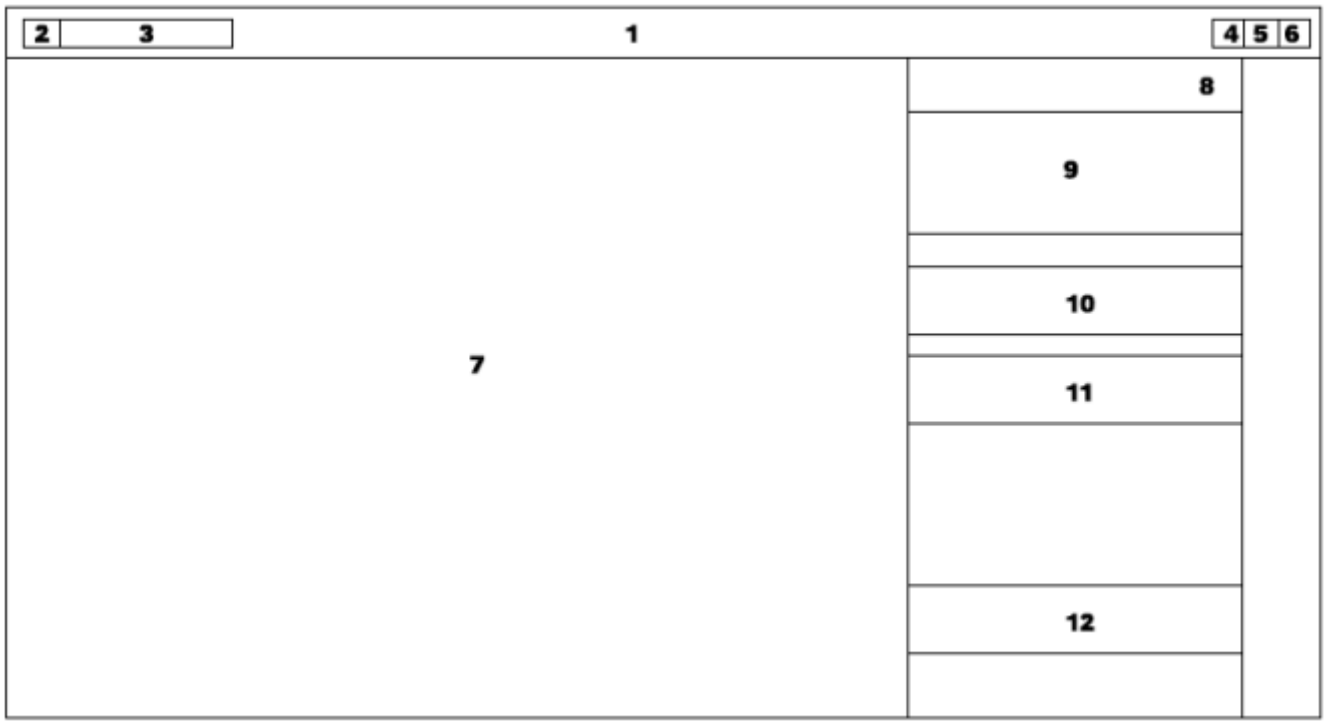

Рисунок 3.10 – Схема користувацького інтерфейсу головного меню

Назвемо елементи зі схеми, зображеної на рисунку 3.10:

1. Панель заголовку вікна.
2. Логотип гри.
3. Назва гри.
4. Згорнути вікно.
5. Згорнути у вікно.
6. Закрити вікно.
7. Головний екран(Фон).

8. Панель меню.
9. Назва гри.
10. Лобі.
11. Налаштування.
12. Вихід.

На рисунку 3.11 наведено схему користувацького інтерфейсу лобі гри.

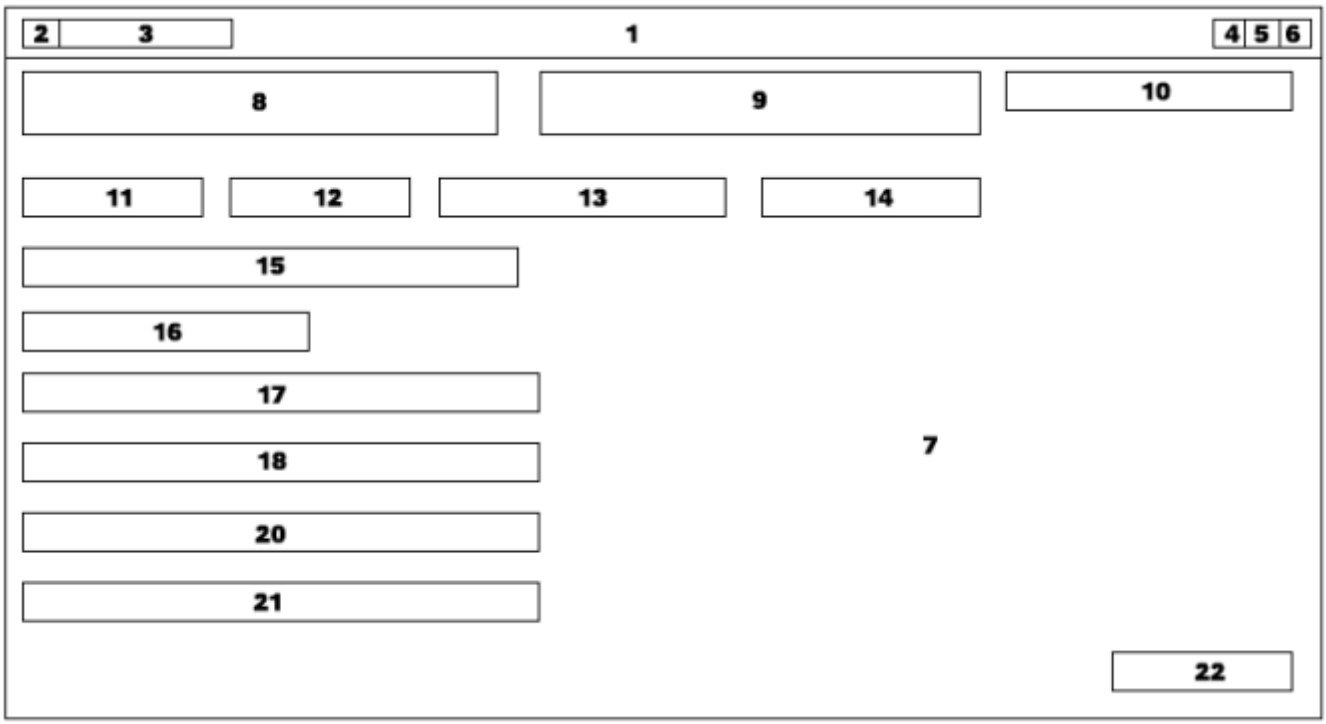

Рисунок 3.11 – Схема користувацького інтерфейсу лобі гри

Назвемо елементи зі схеми, зображеної на рисунку 3.11:

1. Панель заголовку вікна.
2. Логотип гри.
3. Назва гри.
4. Згорнути вікно.
5. Згорнути у вікно.
6. Закрити вікно.
7. Вікно меню.

8. Ім’я користувача.
9. IP користувача.
10. Список гравців, що приєднались.
11. Кнопка «Розпочати гру».
12. Кнопка «Приєднатись».
13. Вибір режимів гри.
14. Можливість стати хостом.
15. Візуалізація курсору.
16. Вибір розширених режимів гри.
17. Гранність першого грального кубика.
18. Гранність другого грального кубика.
19. Розмірність поля в клітинках по ширині.
20. Розмірність поля в клітинках по висоті.

На рисунку 3.12 наведено схему користувацького інтерфейсу геймплею.

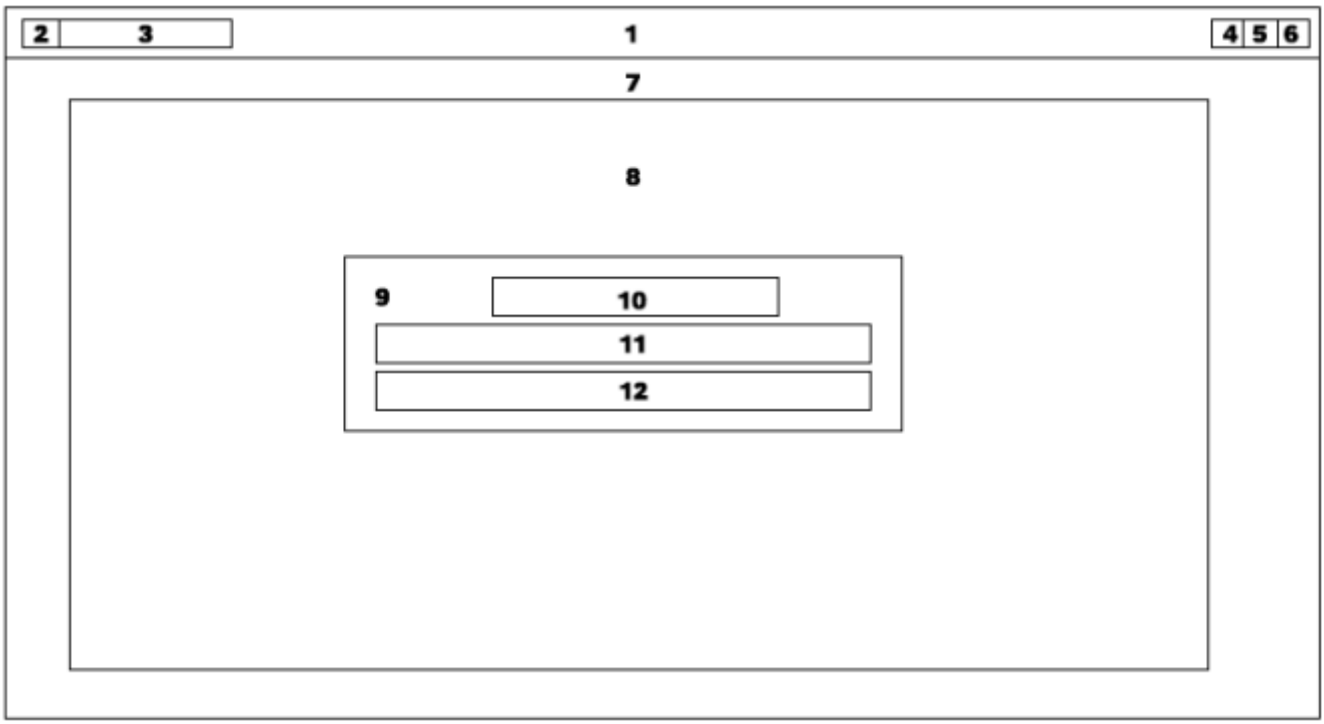

Рисунок 3.12 – Схема користувацького інтерфейсу геймплею

Назвемо елементи зі схеми, зображеної на рисунку 3.12:

1. Панель заголовку вікна.
2. Логотип гри.
3. Назва гри.
4. Згорнути вікно.
5. Згорнути у вікно.
6. Закрити вікно.
7. Фон гри.
8. Дошка гри на якій відбувається ігровий процес.
9. Вікно закінчення гри.
10. Команди переможців.
11. Статистика гравців.
12. Кнопка повернення в меню.

3.6 Розробка дизайну інтерфейсу застосунку

Інтерфейс комп'ютерної гри є важливою складовою та відіграє величезну роль у загальному враженні користувача від гри та його взаємодії з нею [21]. Ось кілька ключових аспектів, що підкреслюють важливість інтерфейсу для комп'ютерної гри:

Перші враження. Інтерфейс гри є першим, з чим користувач стикається при запуску гри. Він має вразити гравця, зацікавити його і зробити гру привабливою для подальшого використання.

Легкість використання. Чим простіший та зрозуміліший інтерфейс, тим легше для користувача орієнтуватися та взаємодіяти з грою. Занадто складний або заплутаний інтерфейс може відлякувати користувача.

Іммерсивність. Інтерфейс повинен допомагати зануритися гравцю в гру, створюючи відчуття присутності в ігровому світі. Це може бути досягнуто за допомогою атмосферного дизайну, звукових ефектів та анімацій.

Функціональність. Інтерфейс повинен надавати всі необхідні функції та можливості для гри, такі як доступ до налаштувань, інвентаря, карті гри тощо. Він повинен бути ефективним і зручним для користувача.

Адаптабельність. Інтерфейс має бути адаптований до різних типів пристроїв та роздільних здатностей екранів. Він повинен бути придатний як для мобільних, так і для настільних пристроїв.

Взаємодія з геймплеєм. Інтерфейс може значно впливати на геймплей, допомагаючи гравцеві керувати своїми діями в грі. Від вдало розробленого інтерфейсу залежить якість взаємодії гравця з ігровим світом.

При запуску гри вперше, що зустріне користувач, – це екран завантаження з движком гри (рисунок 3.13) та її логотипом (рисунок 3.14). Його основна мета – показати користувачеві, що гра запускається, та дати гравцеві візуальні асоціації з ігровим брендом та движком.

Рисунок 3.13 – Екран завантаження(движок)

Рисунок 3.14 – Екран завантаження(логотип)

Онлайн-система має можливість реєстрації користувачів, для чого був створений відповідний екран. Після реєстрації користувач автоматично переходить на екран авторизації, де він вводить свої дані для входу в додаток. Після успішної авторизації користувач може обирати режими гри та переглядати свої досягнення та нагороди.

На головному екрані гравець бачить меню, яке складається з блоків різних кольорів. У верхній частині екрану розташована назва гри нижче розміщені кнопки для доступу до різних опцій гри, таких як налаштування, початок гри або вихід на робочий стіл. Також на екрані буде відображена інформація про поточний рахунок гравця. Головний екран створений з метою привабити користувача яскравим дизайном, зручним інтерфейсом та можливістю швидкого початку гри (рисунок 3.15)

Рисунок 3.15 – Головний екран

На головному екрані гри також розміщені елементи дизайну, що підкреслюють тематику гри, відповідно до аркадного стилю гри «Tricky Squares». Це включає в себе графічні елементи, анімацію та фонове зображення, що створюють атмосферу гри і привертають увагу гравця.

На рисунку 3.16 зображено лобі гри.

Рисунок 3.16 – Лобі гри

Лобі гри «Tricky Squares» для десктопних систем – це вихідне місце, де гравець зустрічається з інтерфейсом гри та має доступ до різноманітних опцій. Тут він може здійснити налаштування перед початком гри, вибрати режим або рівень складності, а також знайти додаткову інформацію.

Вибір режиму гри – гравець може обрати режим гри, наприклад, однокористувацька гра проти комп'ютера або мультиплеєрний режим для гри з іншими гравцями онлайн.

Вибір лобі – гравець може приєднатись до кімнати з гравцями або створити свою зі своїми правилами та налаштуваннями.

Налаштування гри – гравець може відредагувати різні налаштування гри, такі як розмір ігрового поля, гранність кубика, тощо.

Статистика та досягнення – у верхній частині екрану гравцеві будуть доступні коротка статистика, більш детальна інформація знаходитиметься у профілі гравця.

Графічні елементи – лобі буде стилізоване графічними елементами, що відображають тематику гри «Tricky Squares», такими як кольорові квадрати, анімації або фонові зображення.

Ці елементи дозволяють гравцю зручно налаштувати гру перед початком, а також поглибити іммерсивність гри, створюючи атмосферу та асоціації з її тематикою.

Мультиплеєрна гра передбачає прямокутне поле (зазвичай квадратне) на якому може змагатись від 2 до 4 гравців. Ціль гри захопити всі нейтральні поля швидше ніж це зробить супротивник, головною стратегією може також бути заважання супернику та його команді з метою захоплення якнайбільшої кількості території (рисунок 3.17).

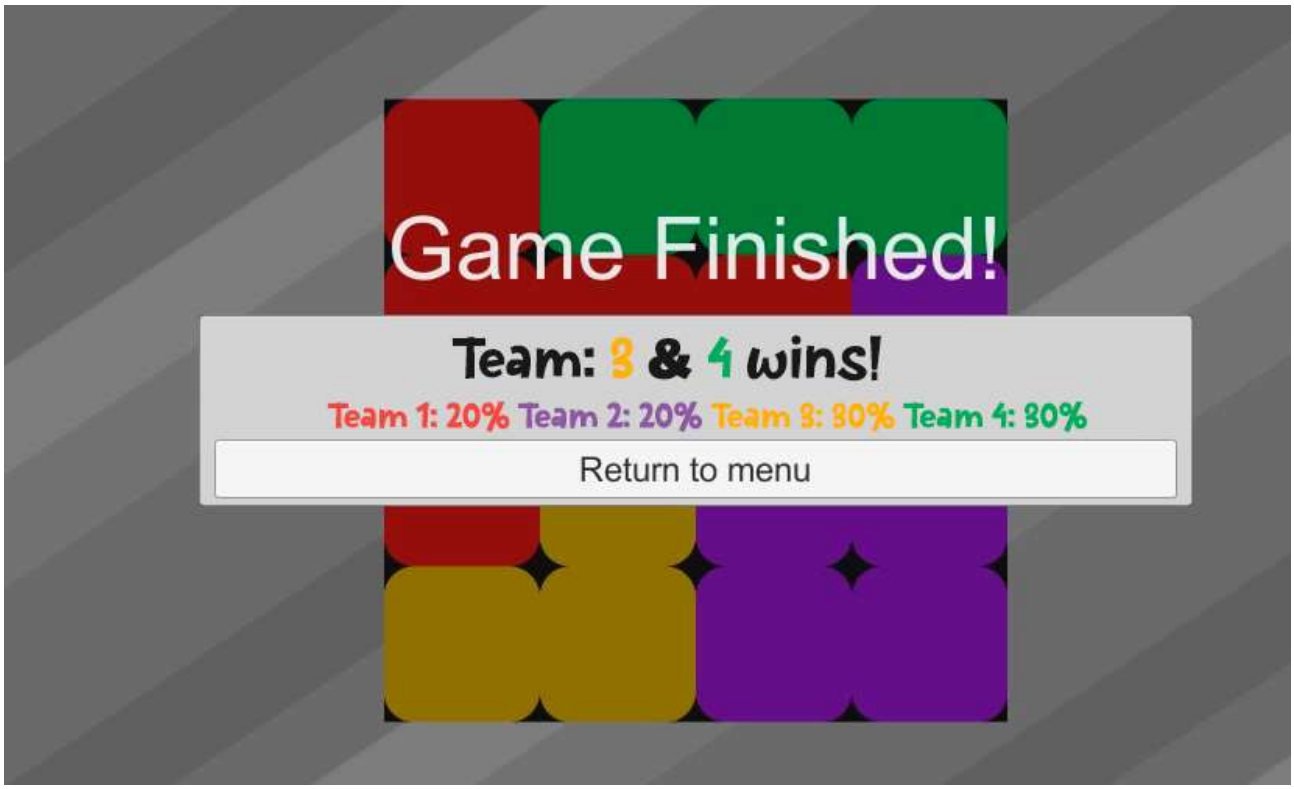

Рисунок 3.17 – Мультиплеєрна гра

Мультиплеєрний режим може включати різні варіанти гри, такі як змагання один на один або командні битви, залежно від вподобань гравців.

Розробка графічного інтерфейсу комп'ютерної гри була проведена у середовищі розробки Unity з використанням мови програмування C#.

3.7 Висновки

У третьому розділі було здійснено аналіз обґрунтування вибору засобів для реалізації програмного забезпечення, описано розробку програмних модулів, їх перелік, а також функціонал та розробку системи. Були наведені структура, функції та інтерфейс застосунку. Проведено аналіз інтерфейсу та геймплею застосунку. А також детально описано аспекти головних модулів системи, а саме графічного інтерфейсу (GUI) та логіки застосунку. Розроблені програмні модулі відповідають поставленим вимогам та специфікаціям Вони успішно інтегруються між собою та утворюють функціонально повний продукт.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування комп'ютерної гри – це невід'ємна частина процесу її розробки та випуску. Під час тестування перевіряють різні аспекти гри, включаючи її функціональність, стабільність, відповідність вимогам та загальне враження від геймплею. Ведеться активний пошук помилок, які можуть вплинути на коректну роботу гри [22].

На головному екрані, зображеному на рисунку 3.15 в третьому розділі, користувача зустрічає меню вибору яке складається з опцій Settings – налаштування, Multilobby – початок гри та Exit – вихід на робочий стіл.

При підключені до мультиплеєрної гри з головного меню гравець потрапляє в лобі, де він може запустити гру як хост або ж приєднатися до створеної гри (рисунок 4.1).

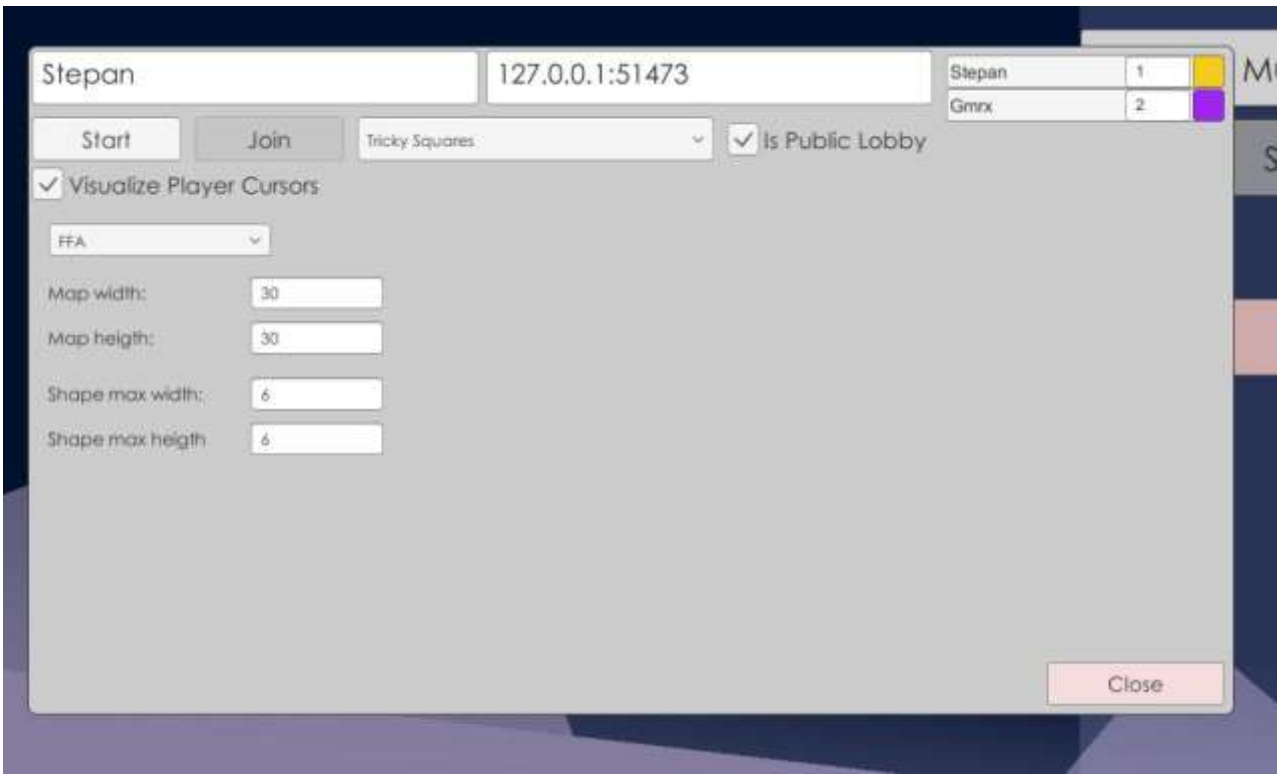

Рисунок 4.1 – Лобі користувача, який створює гру

Виступаючи хостом тобто створюючи гру, користувач має змогу встановлювати налаштування, такі як розмір поля гри та гранність грального кубика, а також обирати режими гри.

На рисунку 4.2 зображено лобі користувача, який приєднався.

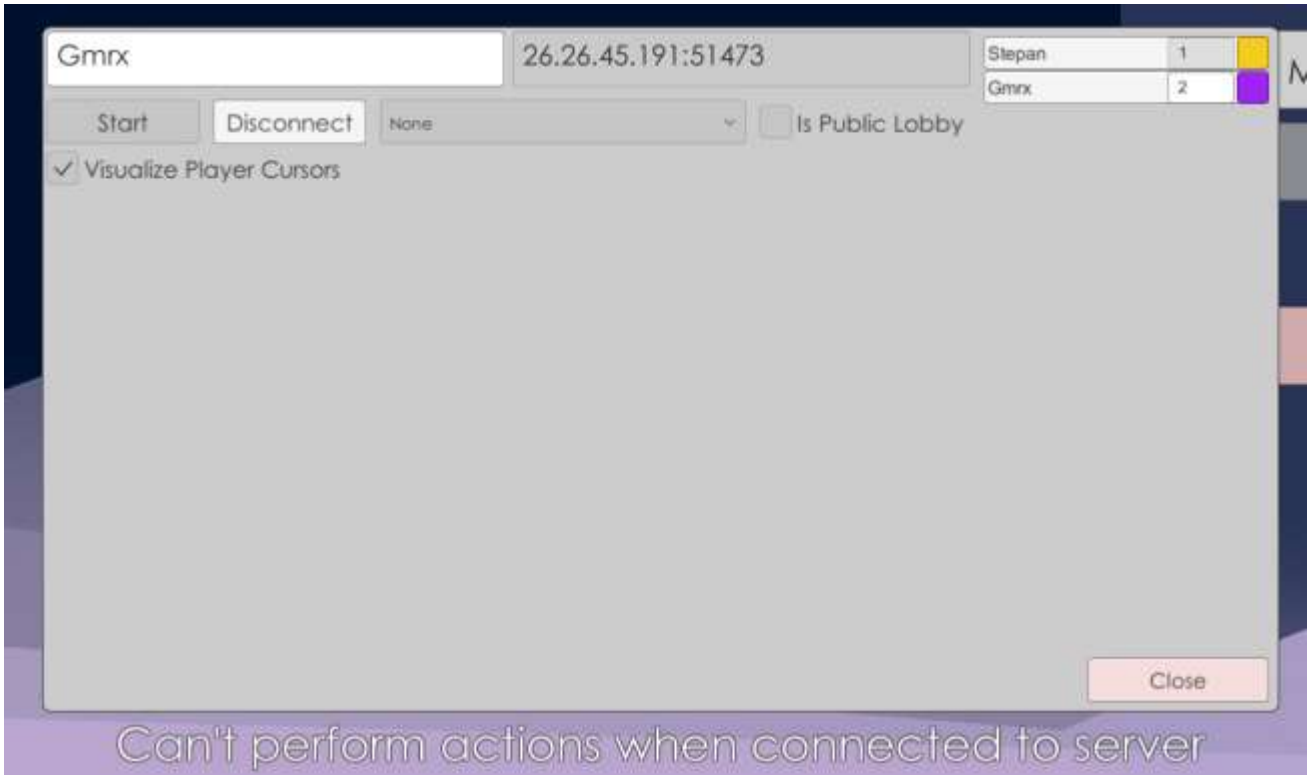

Рисунок 4.2 – Лобі користувача, який приєднався

Користувач, який приєднується до гри, повинен ввести IP творця та встановлений ним порт через двокрапку після чого він зможе приєднатись у лобі. В лобі користувач, що приєднався не має можливості редагувати налаштування гри, але він може в будь-який момент відключитись при нагоді та може змінювати власну інформацію таку як нікнейм, назву команди (за замовчуванням встановлено порядковий номер) та колір команди.

Гра починається з того, що гравець обирає кут, в якому він починатиме гру. Розпочати гру можна лише з кута, користувач може виділити доступну область хоч по центру дошки, але поставити фігуру він зможе лише в куті за правилами гри (рисунок 4.3).

Рисунок 4.3 – Перший хід гравця

Гравцеві 1 випали кубики з гранями 1 та 3 відповідно, тобто він може поставити фігуру розміром 1:3 або 3:1 при обертанні у вибраному куті. Після цього хід перейде до суперника.

На рисунку 4.4 зображено перший хід супротивника.

Рисунок 4.4 – Перший хід супротивника

Супротивник також може обрати будь-який кут із трьох, що залишились, в даному випадку гравець обрав протилежний кут. Для усіх гравців діють однакові правила, тому фігури не можна ставити де завгодно, а лише у кутах.

Гравцеві 2 натомість випали кубики з гранями 1 та 2 відповідно тобто він може поставити фігуру розміром 1:2 або 2:1 при обертанні в обраному куті.

У наступних ходах гравці приєднують фігури до своїх територій, тобто кожна наступна фігура приєднується до наявної території. Приєднання фігури до території гравця наочно зображено на рисунку 4.5.

Рисунок 4.5 – Приєднання фігур до своєї території

Гра продовжується до тих пір, поки гравці не заповнять усю територію. Стратегічно гра триває до того моменту, поки один із гравців не заповнив більшість території, навіть просто огородивши її фігурами, оскільки в подальшому він зможе заповнити прогалини без втручання противника (рисунок 4.6).

Рисунок 4.6 – Ігровий процес

При завершені гри гравцям видається повідомлення де вказаний переможець та відсотковим співвідношенням захопленої території суперників (рисунок 4.7).

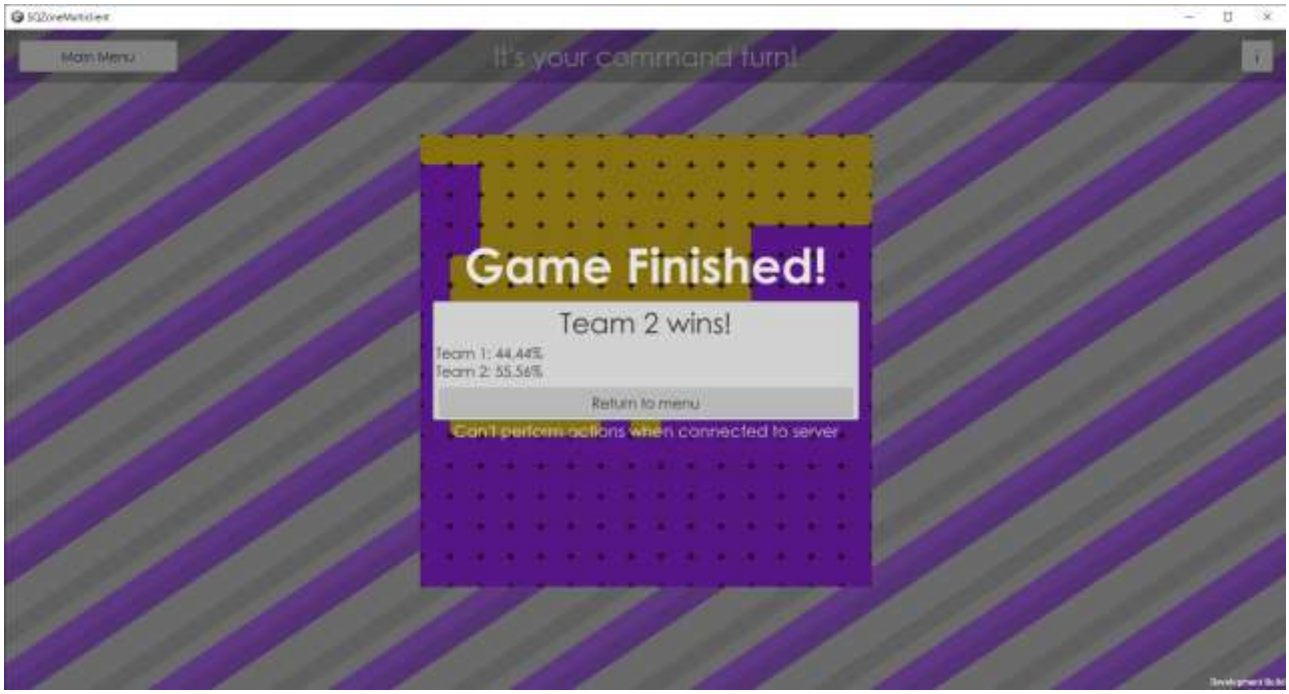

Рисунок 4.7 – Завершення гри

Під час тестування застосунку, були здійснені ретельні перевірки його функціональності та якості. Виявлені проблеми, які потребують уваги та виправлення були усунені. Проведені додаткові тести та внесені необхідні покращення для забезпечення стабільності та надійності.

4.2 Розробка інструкції користувача

Інструкція користувача – це документ, який містить інформацію про те, як користуватися певним пристроєм, програмою, сервісом або послугою. Її мета – надати користувачеві всю необхідну інформацію та крок за кроком вказівки для успішного використання продукту [23]. Інформація про мінімальні та рекомендовані вимоги комп’ютера наведена в таблицях 4.1 та 4.2 відповідно.

Таблиця 4.1 – Мінімальні вимоги:

Процесор	Intel Core i3-2100 або AMD Athlon 200GE
Оперативна пам'ять	2 ГБ
Пропускна здатність мережі	15-20 Мбіт/с
Місце на жорсткому диску	3 ГБ
Операційна система	Windows 7/8/10 (64-біт) або Ubuntu 22.04 LTS

Таблиця 4.2 – Рекомендовані вимоги:

Процесор	Intel Core i3-10100F або AMD Ryzen 3 3100
Оперативна пам'ять	4 ГБ
Пропускна здатність мережі	50 Мбіт/с
Місце на жорсткому диску	5 ГБ
Операційна система	Windows 10 (64-біт), або Ubuntu 22.04 LTS

Перевірка сумісності програми з різними версіями операційної системи є основним критерієм успішного тестування.

Клавіатура має різні конфігурації клавіш, залежно від моделі та виробника. Проте, основні елементи клавіатури зазвичай мають стандартний розмір та розташування.

Для застосунку було обрано комп’ютерну мишку як основний засіб введення та клавіатуру як допоміжний. Гравець здійснює приближення та наведення на об’єкти за допомогою комп’ютерної мишки. Гравцеві слід затиснути ліву кнопку миші та провести для виділення території. Або натиснути на праву кнопку миші для відміни вибору.

Клавіатура використовується для повороту фігур, заповнення фігури, підтвердження ходу та пропуску ходу. А також для переміщення камери (рисунок 4.8).

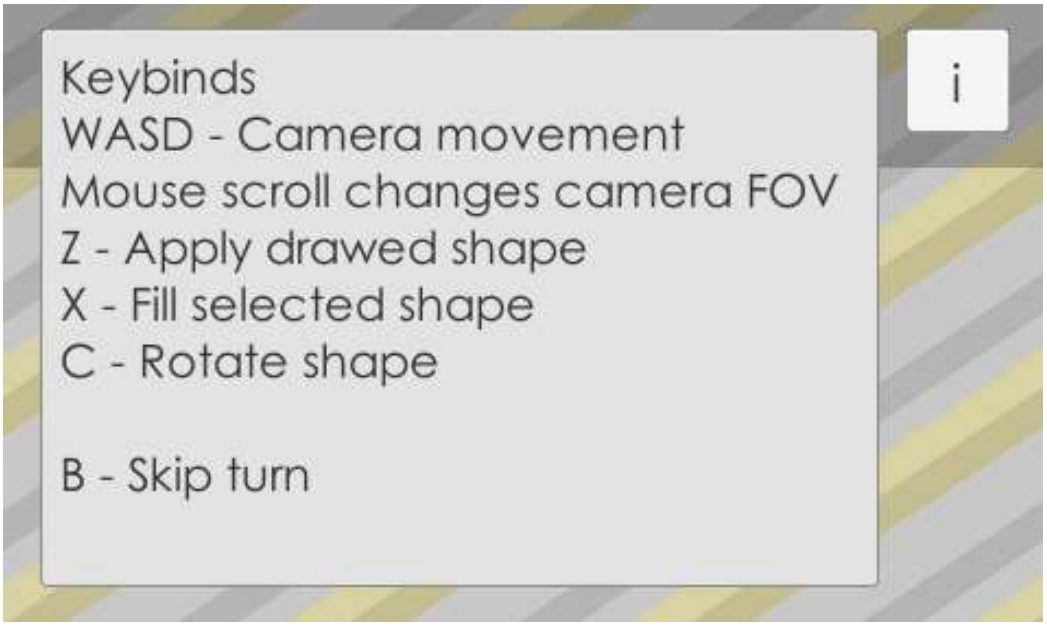

Рисунок 4.8 – Конфігурація клавіш

За переміщення камери відповідають клавіші WASD.

Підтвердження ходу гравця, тобто підтвердження постановки фігури, здійснюється за допомогою клавіші «Z».

Заповнення фігури за допомогою клавіші «X». Якщо фігура велика за розміром, то достатньо виділити кути фігури і натиснути вказану клавішу, це

дозволяє зберегти час гравців, не витрачаючи його на заповнення кожної клітинки.

Поворот фігури здійснюється за допомогою клавіші «C».

Також гравець має можливість пропустити хід за допомогою клавіші «B».

Було ретельно розглянуто процес розробки інструкції користувача та характеристики системних вимог. Під час розробки інструкції враховано ясність, доступність та послідовність інформації, щоб зробити процес оволодіння продуктом якнайбільш простим і зручним для користувача.

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів комп'ютерної гри «Tricky Squares». Було протестовано механіки й основні функції модулів застосунку. Описана робота гри та наведені скріншоти її роботи. Розроблена інструкція користувача з вказаними системними вимогами, а також описана взаємодія з програмою.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмні модулі реалізації комп'ютерної гри «Tricky Squares» для покращення навчальних інтерактивних програм за допомогою ігор в закладах освіти для розвитку когнітивних навичок, таких як концентрація, логіка, пам'ять та увага.

Проведено порівняльний аналіз сучасного стану комп'ютерних ігор у жанрі 2D, розглянуто існуючі аналоги, такі як: Tetris, Tricky Towers, Candy Crush. Визначено їхні переваги та недоліки в порівнянні з розроблюваною системою, встановлено, що розроблюваний програмний засіб переважає у функціональності розглянуті аналоги та підтверджено доцільність розробки власного програмного рішення.

Базуючись на результатах аналізу та порівняння, було поставлено основні задачі бакалаврської дипломної роботи та обрано методи розробки системи. Проведено варіантний аналіз та обґрунтування вибору засобів реалізації програмного продукту. В якості мови програмування було обрано мову програмування C#. В якості середовища програмної розробки було використано Microsoft Visual Studio 2022. В якості движка програмної розробки обрано Unity.

Відповідно до поставлених задач було зроблено наступне:

- проведено аналіз існуючих методів і засобів розробки 2D ігор для визначення напрямків підвищення їх продуктивності;
- розроблено метод ініціалізації системи для автоматизації ігрових процесів;
- розроблено модель десктопного застосунку для тренування логічних здібностей, яка включає основний функціонал застосунку;
- програмно реалізовано функціонал застосунку;
- розроблено зручний інтерфейс користувача, який є легким у використанні;

- забезпечено підтримку української мови застосунку та доступність на різних платформах і пристроях;
- проведено тестування розробленого застосунку.

Також було розроблено схеми загального алгоритму роботи комп’ютерної гри та модель системи з використанням UML діаграм.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

Бакалаврську дипломну роботу оформлено відповідно до методичних вказівок [24].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Eric L.B. Games People Play: The Psychology of Human Relationships. – New York: Grove Press, 1964 – 256с. (дата звернення 01.03.2022).

2. Jason S. Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made. – New York: Harper Paperbacks, 2013 – 353с. (дата звернення 01.03.2022).

3. Зелінський В.Р., Войтко В.В., Денисюк А.В., Гаврилюк О.В., БарчукН.Є. Розробка мультиплеєрної комп'ютерної гри «Tricky Squares». Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20438/16968 (дата звернення 01.04.2022).

4. CHRISTIAN L. Gramática visual. – Barcelona: Gustavo Gili, 2015 – 56с. (дата звернення 09.03.2022).

5. 2D computer graphics URL: https://graphics.fandom.com/wiki/2D_computer_graphics (дата звернення 14.03.2022).

6. Tetris URL: https://en.wikipedia.org/wiki/Tetris (дата звернення 14.03.2022).

7. Tricky Towers URL: https://trickytowers.com/ (дата звернення 14.03.2022).

8. Candy Crush URL: https://en.wikipedia.org/wiki/Candy_Crush_Saga (дата звернення 14.03.2022).

9. Use case diagram URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення 06.04.2022).

10. Sequence diagram URL: https://en.wikipedia.org/wiki/Sequence_diagram (дата звернення 06.04.2022).

11. Sequence diagram URL: https://www.lucidchart.com/pages/uml-sequence-diagram (дата звернення 06.04.2022).

12. C# URL: https://en.wikipedia.org/wiki/C_Sharp_(programming_language) (дата звернення 26.03.2022).

13. Unity and C# URL: https://unity.com/how-to/beginner-game-coding-resources (дата звернення 28.03.2022).

14. Unity URL: https://en.wikipedia.org/wiki/Unity_(game_engine) (дата звернення 28.03.2022).

15. Visual Studio URL: https://en.wikipedia.org/wiki/Visual_Studio (дата звернення 29.03.2022).

16. Game mehanics URL: https://en.wikipedia.org/wiki/Game_mechanics (дата звернення 09.04.2022).

17. Functional of game URL: https://it-osvita.diia.gov.ua/task/item/9a2c5a67-ef75-4b16-8846-918bdbf8a338 (дата звернення 09.04.2022).

18. Module in game development URL: https://www.kent.ac.uk/courses/modules/module/DIGM6450 (дата звернення 09.04.2022).

19. Class diagram URL: https://en.wikipedia.org/wiki/Class_diagram (дата звернення 12.04.2022).

20. Graphical user interface URL: https://blog.hubspot.com/website/what-is-gui (дата звернення 12.04.2022).

21. User interface URL: https://retrostylegames.com/blog/ui-game-meaning/ (дата звернення 03.04.2022).

22. Computer game testing URL: https://pinglestudio.com/blog/game-testing/what-is-the-game-testing-process (дата звернення 14.04.2022).

23. User manual URL: https://document360.com/blog/creating-a-user-manual/ (дата звернення 14.04.2022).

24. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

64

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«12» березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу
«Розробка мультиплеєрної комп'ютерної гри «Tricky Squares» з використанням C# та Unity»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н., доц. Коваленко О. О.
«12» березня 2024 р.

Виконав:
студент гр. ЗПІ-20б Зелінський В. Р.
«12» березня 2024 р.

Вінниця – 2024

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мультиплеєрної комп'ютерної гри «Tricky Squares» з використанням C# та Unity».

Галузь застосування – навчальні інтернет ресурси.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №17 від 20 березня 2024 р.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є покращення розвитку логічного та стратегічного мислення за рахунок розробки і використання логічної комп’ютерної гри, що дозволить в ігровому режимі у багатокористувацькому форматі проводити тренування в середовищі гейміфікованої системи та поєднувати режим тренування з отриманням задоволення від використання комп’ютерної ігрової програми.

Призначення роботи – розробка програмного продукту для покращення навчальних інтерактивних програм за допомогою ігор.

4. Вихідні дані для проведення НДР

1.1. Eric L.B. Games People Play: The Psychology of Human Relationships. – New York: Grove Press, 1964 – 256с.

2. Jason S. Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made. – New York: Harper Paperbacks, 2013 – 353с.

3. CHRISTIAN L. Gramática visual. – Barcelona: Gustavo Gili, 2015 – 56с.

5. Технічні вимоги

Комп’ютер з 64-розрядним процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 2 ГБ, розмір жорсткого диску 128 ГБ. Операційна система

Windows 7/8/10.

6. Конструктивні вимоги.

Додаток має відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської дипломної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз сучасного стану комп'ютерних ігор у жанрі 2D та постановка задач дослідження	14.03.24 – 24.03.24
2	Розробка методу, моделі та алгоритмів роботи застосунку	25.03.24 –15.04.24
3	Розробка модулів програмного застосунку	15.04.24 – 30.04.24
4	Тестування програми	01.05.24 – 10.05.24
5	Оформлення матеріалів до захисту БДР	11.05.24 – 30.05.24

10 Порядок контролю та прийняття.

Усі етапи бакалаврської дипломної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

67

Додаток Б – Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

Назва роботи: «Розробка мультиплеєрної комп'ютерної гри «Tricky Squares» з використанням C# та Unity»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Коваленко О.О.

Оригінальність	97,6%
Схожість	2,4%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку: _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи: _____ Зелінський В.Р

Керівник роботи: _____ Коваленко О.О.

Додаток В – Лістинг програми

CameraController

```
using Assets.Scripts.Game;
using Games.SQZone;
using Main;
using System.Runtime.CompilerServices;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] protected SQZoneCursorInputs m_gameInputs;
    [SerializeField] protected Camera m_camera;
    [SerializeField] protected FOVController m_fovController;
    [SerializeField] protected float m_moveSpeed = 1f;
    [SerializeField] protected float m_velocityRezistance = 0.6f;
    [Space(4f)]
    [SerializeField] protected Transform m_leftBottomMapCorner;
    [SerializeField] protected Transform m_rightTopMapCorner;

    public bool IsReceiveInputs { get; set; } = true;
    public Vector3 Velocity { get; private set; }


    private const float VelocityMinimumValue = 0.0001f;
    private void Update()
    {
        if (!IsReceiveInputs) return;
        if (Input.mouseScrollDelta.y != 0)
        {
```

```
            m_fovController.SetFOV(m_fovController.TargetFOV -
Input.mouseScrollDelta.y);
        }

        Vector2 input = HandleInput();
        if (input == Vector2.zero)
        {
            if (Velocity.x == 0f && Velocity.y == 0f) return;

            Velocity -= Velocity * (m_velocityRezistance * Time.deltaTime);
            if (Mathf.Abs(Velocity.x) < VelocityMinimumValue && Mathf.Abs(Velocity.y) <
VelocityMinimumValue)
                Velocity = Vector2.zero;
        }
        else
        {
            float localMovingSpeed = m_camera.orthographicSize * m_moveSpeed *
Time.deltaTime;
            Velocity = input * localMovingSpeed;
        }
        m_camera.transform.localPosition = ChangeCameraPosition();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        Vector3 ChangeCameraPosition()
        {
            return new Vector3(
                x: Mathf.Clamp(m_camera.transform.localPosition.x + Velocity.x,
m_leftBottomMapCorner.localPosition.x, m_rightTopMapCorner.localPosition.x),
                y: Mathf.Clamp(m_camera.transform.localPosition.y + Velocity.y,
m_leftBottomMapCorner.localPosition.y, m_rightTopMapCorner.localPosition.y),
                z: m_camera.transform.localPosition.z);
        }
    }
```

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private Vector2 HandleInput()
            => new(x: (Input.GetKey(Keybinds.Global.MoveLeft) ? -1 : 0) +
(Input.GetKey(Keybinds.Global.MoveRight) ? 1 : 0),
                   y: (Input.GetKey(Keybinds.Global.MoveDown) ? -1 : 0) +
(Input.GetKey(Keybinds.Global.MoveUp) ? 1 : 0));

    #if UNITY_EDITOR
        private void Reset()
        {
            if (TryGetComponent(out FOVController controller))
            {
                m_fovController = controller;
            }
            else m_fovController = gameObject.AddComponent<FOVController>();
            m_fovController.Camera = m_camera;
        }
        private void OnDrawGizmosSelected()
        {
            if (m_rightTopMapCorner is null || m_leftBottomMapCorner is null ||
m_leftBottomMapCorner.parent is null) return;

            Gizmos.color = new(0.4f, 0.4f, 1f, 0.4f);
            Gizmos.matrix = Matrix4x4.TRS(m_leftBottomMapCorner.parent.position,
m_leftBottomMapCorner.parent.rotation, m_leftBottomMapCorner.parent.lossyScale);
            Gizmos.DrawCube(center: m_rightTopMapCorner.localPosition +
m_leftBottomMapCorner.localPosition,
                            size: m_rightTopMapCorner.localPosition -
m_leftBottomMapCorner.localPosition);
        }
    #endif
    }
```

CursorUpdates

```
using UnityEngine;

public class CursorUpdater : MonoBehaviour
{
    [SerializeField] private Vector2 m_maxZone;
    void FixedUpdate() => transform.position = new(x: (Input.mousePosition.x / 
Screen.width * 2 - 1) * m_maxZone.x,
                                                   y: (Input.mousePosition.y / Screen.height * 2 - 1) * 
m_maxZone.y);
}
```

FOVController

```
using System.Collections;
using System.Runtime.CompilerServices;
using UnityEngine;

namespace Assets.Scripts.Game
{
    public class FOVController : MonoBehaviour
    {
        [SerializeField] protected Camera m_camera;
        [SerializeField] protected float m_minimalFOVValue = 2f;
        [SerializeField] protected float m_maximalFOVValue = 16f;
        [SerializeField] protected AnimationCurve m_zoomCurve = 
AnimationCurve.Linear(0, 0, 0.3f, 1f);

        public Camera Camera { get => m_camera; set => m_camera = value; }
        public float TargetFOV { get; private set; }
```

```
            float time;
            float initialFOV, deltaFOV;
            private void Awake()
            {
                initialFOV = TargetFOV = m_camera.orthographicSize;
            }
            /// <param name="requiredFOV"> </param>
            public void SetFOV(float requiredFOV)
            {
                requiredFOV = Mathf.Clamp(requiredFOV, m_minimalFOVValue,
m_maximalFOVValue);
                if (requiredFOV == TargetFOV) return;
                time = 0; enabled = true;

                TargetFOV = requiredFOV;
                initialFOV = m_camera.orthographicSize;
                deltaFOV = requiredFOV - initialFOV;
            }
            private void Update()
            {
                time += Time.deltaTime;

                if (time > m_zoomCurve[m_zoomCurve.length - 1].time)
                {
                    enabled = false;
                    m_camera.orthographicSize = initialFOV + deltaFOV;
                }
                else m_camera.orthographicSize = initialFOV + deltaFOV *
m_zoomCurve.Evaluate(time);
            }
        }
    }
```

GameGenerator

```
namespace Games
{
    public enum GameModes : byte
    {
        None = 0,
        Lobby,
        TrickySquares,
    }

    public static class GameGenerator
    {
        public static BaseSaveData GenerateGameData(GameModes mode,
GeneratorArguments args)
        {
            return mode switch
            {
                GameModes.TrickySquares =>
GenerateSQZoneGameData((SQZone.SQZoneGeneratorArguments)args),
                _ => null,
            };
        }

        public static SQZone.SQZoneSaveData
GenerateSQZoneGameData(SQZone.SQZoneGeneratorArguments args)
        {
            SQZone.SQZoneSaveData data = new()
            {
                GameMode = args.GameMode,
                MatchMode = args.MatchMode,

                MapX = args.MapSizeX,
                MapY = args.MapSizeY,
```

```
            MaxShapeSizeX = args.MaxShapeSizeX,
            MaxShapeSizeY = args.MaxShapeSizeY,
            ShapeCount = args.ShapeCount,
            RandomSeed = (int)System.DateTime.Now.Ticks,
            //MapData = new byte[args.MapSizeX * args.MapSizeY],
        };
        return data;
    }
  }
}
```

GameInputs

```
using UnityEngine;

namespace Main
{
  public class GameInputs : MonoBehaviour
  {
    private void Update()
    {
      if (Input.GetKeyDown(Keybinds.Global.ExtendConsole))
      {
        GameConsole.InvertExpand();
      }
    }
  }
}
```

GameManager

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace Games
{
   public class GameManager : MonoBehaviour
   {
      public static GameManager Instance { get; private set; }
      public static BaseSaveData SaveData { get; private set; }

      private void Awake()
      {
         DontDestroyOnLoad(this);
         Instance = this;
      }

      private IEnumerator LocalLoadGame(BaseSaveData saveData, System.Action
onGameLoaded)
      {
         //if (!Library.TryGetSceneByMode(saveData.GameMode, out SceneField scene))
yield break;
         yield return
SceneManager.LoadSceneAsync(Library.GetSceneByMode(saveData.GameMode));

         BaseLocalManager.SetupSaveData(saveData);

         onGameLoaded?.Invoke();
      }

      #region Static methods
```

```
        public static void LoadGame(BaseSaveData saveData, System.Action
onGameLoaded = null)
            => Instance.StartCoroutine(Instance.LocalLoadGame(saveData, onGameLoaded));
        #endregion
    }
}
```

Library

```
using Games;
using System.Collections.Generic;
using UnityEngine;

public class Library : MonoBehaviour
{
    internal static Library Instance { get; private set; }
    public static Color DefaultColor => Instance.DefaultTeamColor;

    [SerializeField] List<ModeToSceneStruct> SceneAssociations = new();
    [SerializeField] List<TeamColorsStruct> TeamColors = new();
    [SerializeField] private Color DefaultTeamColor = Color.white;

    private void Awake()
    {
        DontDestroyOnLoad(this);
        Instance = this;
    }

    internal static SceneField GetSceneByMode(GameModes mode)
    {
        foreach (ModeToSceneStruct i in Instance.SceneAssociations)
        {
            if (i.mode == mode) return i.scene;
        }
```

```
    return null;
}
internal static bool TryGetSceneByMode(GameModes mode, out SceneField scene)
    => (scene = GetSceneByMode(mode)) is not null;
internal static Color GetTeamColor(ushort team)
{
    foreach (TeamColorsStruct i in Instance.TeamColors)
    {
        if (i.team == team) return i.color;
    }
    return Instance.DefaultTeamColor;
}


[System.Serializable]
private struct ModeToSceneStruct
{
    public GameModes mode;
    public SceneField scene;

    public ModeToSceneStruct(GameModes mode, SceneField scene)
    {
        this.mode = mode;
        this.scene = scene;
    }
}
[System.Serializable]
private struct TeamColorsStruct
{
    public ushort team;
    public Color color;

    public TeamColorsStruct(ushort team, Color color)
    {
```

```
            this.team = team;
            this.color = color;
        }
    }
}
```

GameGrid

```
using RiptideNetworking;
using System;
using System.Collections.Generic;
using UnityEngine;

namespace Games.SQZone
{
    [Serializable]
    public class GameGrid : MonoBehaviour
    {
        public int MapVolume => Grid.Length;
        public int OccupiedVolume => m_occupiedVolume;
        [SerializeField] Transform parent;
        [SerializeField] GridBlock block;

        [HideInInspector] public int Width;
        [HideInInspector] public int Height;
        [HideInInspector] public Cell[,] Grid;
        private int m_occupiedVolume;

        public readonly Dictionary<ushort, int> CellOccupation = new();
        public readonly Dictionary<Vector2Int, Cell> FadeCells = new();
        public void SetupGrid(int maxWidth, int maxHeight, GridBlock block = null)
        {
            if (block is not null)
                this.block = block;
```

```
            CellOccupation.Clear();
            Width = maxWidth;
            Height = maxHeight;
            Grid = new Cell[maxWidth, maxHeight];
            maxWidth--;
            while (maxWidth > -1)
            {
                for (int y = 0; y < maxHeight; y++)
                {
                    Grid[maxWidth, y] = new();
                }
                maxWidth--;
            }
        }

        /// <summary>
        /// Validates placement of objects in zone (Checks nearby cells for team avability)
        /// </summary>
        /// <param name="minX">Left edge of zone on game map</param>
        /// <param name="maxX">Right edge of zone</param>
        /// <param name="minY">Bottom edge of zone</param>
        /// <param name="maxY">Top edge of zone</param>
        /// <param name="team">Team of caller</param>
        /// <param name="teamAsTarget">If true - method will trying to find any cell of
given team</param>
        /// <returns>True if this cell is available for placement</returns>
        public bool ValidateFillZone(int minX, int maxX, int minY, int maxY)
        {
            // Debug.Log(nameof(ValidateNearCellZone) + ": MinX: " + minX + "  MaxX: " +
maxX + "  MinY: " + minY + "  MaxY: " + maxY);
            // Checks cell occupation in selected bounds
            for (int x = minX; x < maxX + 1; x++)
            {
```

```
        for (int y = minY; y < maxY + 1; y++)
        {
            if (Grid[x, y].team == 0) continue;
            return false;
        }
    }
    return true;

}

public bool ValidateContacts(int minX, int maxX, int minY, int maxY, ushort team)
{
    // Y Edges checkouts
    if (minX > 0) // Left Edge
    {
        minX -= 1;
        for (int y = minY; y <= maxY; y++)
        {
            if (GetValid(minX, y)) return true;
        }
        minX += 1;
    }
    if (maxX < Width - 1) // Right Edge
    {
        maxX += 1;
        for (int y = minY; y <= maxY; y++)
        {
            if (GetValid(maxX, y)) return true;
        }
        maxX -= 1;
    }

    // X Edges checkouts
    if (minY > 0) // Bottom Edge
```

```
            {
                minY -= 1;
                for (int x = minX; x <= maxX; x++)
                {
                    if (GetValid(x, minY)) return true;
                }
            }
            if (maxY < Height - 1) // Top Edge
            {
                maxY += 1;
                for (int x = minX; x <= maxX; x++)
                {
                    if (GetValid(x, maxY)) return true;
                }
            }
            return false;

            // Returns TRUE if detected required team
            bool GetValid(int x, int y) => Grid[x, y].team == team;
        }

        public bool IsAllMapOccupied() => m_occupiedVolume >= Grid.Length;
        public void RecalculateOccupation()
        {
            int occupationSize = 0;
            foreach (int item in CellOccupation.Values)
            {
                occupationSize += item;
            }
        }

        /// <summary>
        /// Fills zone with object of given team, but unsafe! (Use after <see
cref="ValidateFillZone"/>)
```

```
/// </summary>
/// <param name="minX"></param>
/// <param name="maxX"></param>
/// <param name="minY"></param>
/// <param name="maxY"></param>
/// <param name="team"></param>
public void FillZoneUnsave(int minX, int maxX, int minY, int maxY, ushort team)
{
   for (int x = minX; x < maxX + 1; x++)
   {
      for (int y = minY; y < maxY + 1; y++)
      {
         if (FadeCells.ContainsKey(new Vector2Int(x, y))) continue;
         else CreateFadeUnsave(x, y, team);
      }
   }

   Submit(team);
}

/// <summary>
/// Returns TRUE if position NOT WITHIN grid bounds (Invalid)
/// </summary>
/// <returns>TRUE if position INVALID</returns>
/// <summary>
public bool IsCellValid(int x, int y) => x < 0 || x >= Width || y < 0 || y >= Height;
/// Returns TRUE if cell OCCUPIED. Uses <see cref="IsCellValid"/> before check
/// </summary>
/// <returns>TRUE if cell OCCUPIED</returns>

public bool IsCellOccupied(int x, int y) => IsCellValid(x, y) || Grid[x, y].team != 0;



public bool RaycastBlock(int x, int y, out Cell cell)
```

```
{
    // Debug.Log(nameof(RaycastBlock) + ": X: " + x + "  Y: " + y);
    if (IsCellValid(x, y))
    {
        cell = null;
        return false;
    }

    cell = Grid[x, y];
    return cell.block is not null;
}
public bool RaycastFade(int x, int y, out Cell cell)
{
    // Debug.Log(nameof(RaycastBlock) + ": X: " + x + "  Y: " + y);
    return FadeCells.TryGetValue(new Vector2Int(x, y), out cell);
}

/// <summary>
/// Creates block, in fade status.
/// Use <see cref="Submit"/> to apply fade blocks
/// </summary>
/// <param name="x">X position of cell</param>
/// <param name="y">Y position of cell</param>
/// <param name="team">Team of cell placer</param>
/// <returns>Created Fade block</returns>
public Cell CreateFadeUnsave(int x, int y, ushort team)
{
    // Debug.Log(nameof(CreateFadeUnsave) + ": X: " + x + "  Y: " + y);
    Vector2Int pos = new(x, y);
    if (FadeCells.ContainsKey(pos)) return null;
    Cell cell = new()
    {
        block = CreateAt(x, y, team),
        team = team,
```

```
            };
            cell.block.transform.localScale = new(0.8f, 0.8f);
            FadeCells.Add(pos, cell);
            return cell;
        }
        /// <summary>
        /// Destroys block, in fade status.
        /// </summary>
        /// <param name="x">X position of cell</param>
        /// <param name="y">Y position of cell</param>
        /// <returns>TRUE - if block was destroyed</returns>
        public bool DestroyFade(int x, int y)
        {
            // Debug.Log(nameof(DestroyFade) + ": X: " + x + "  Y: " + y);
            Vector2Int pos = new(x, y);
            if (FadeCells.TryGetValue(pos, out Cell cell))
            {
                FadeCells.Remove(pos);
                cell.Destroy();
                return true;
            }
            return false;
        }
        /// <summary>
        /// Destroys block, in fade status.
        /// </summary>
        /// <param name="x">X position of cell</param>
        /// <param name="y">Y position of cell</param>
        /// <param name="team">Team of what cells must be deleted</param>
        public bool DestroyFade(int x, int y, ushort team)
        {
            // Debug.Log(nameof(DestroyFade) + ": X: " + x + "  Y: " + y + "  Team: " +
team);
            Vector2Int pos = new(x, y);
```

```
    if (FadeCells.TryGetValue(pos, out Cell cell) && cell.team == team)
    {
        cell.Destroy();
        FadeCells.Remove(pos);
        return true;
    }
    return false;
}

public void Discard()
{
    foreach (KeyValuePair<Vector2Int, Cell> cell in FadeCells)
    {
        cell.Value.block.Destroy();
    }
    FadeCells.Clear();
}

public GridBlock CreateBlock(int x, int y, ushort team)
{
    Cell cell = Grid[x, y];
    if (cell.block is null)
    {
        cell.team = team;
        return cell.block = CreateAt(x, y, team);
    }
    return cell.block;
}

public void DestroyBlock(int x, int y)
{
    // Debug.Log(nameof(DestroyBlock) + ": X: " + x + "  Y: " + y);
    if (Grid[x, y].block is null) return;
    Cell cell = Grid[x, y];
```

```
            cell.Destroy();
        }
        public void DestroyBlock(int x, int y, ushort team)
        {
            // Debug.Log(nameof(DestroyBlock) + ": X: " + x + "  Y: " + y);
            Cell cell = Grid[x, y];
            if (cell.block is null) return;
            if (cell.team == team) cell.Destroy();
        }
        private GridBlock CreateAt(int x, int y, ushort team = 0)
        {
            // Debug.Log(nameof(CreateAt) + ": X: " + x + "  Y: " + y);
            GridBlock block = Instantiate(this.block.gameObject,
parent).GetComponent<GridBlock>();
            block.transform.localPosition = new(x, y, 0);
            block.Sprite.color = Library.GetTeamColor(team);
            return block;
        }
        /// <summary>
        /// Places all created fade blocks, sendings results to cilents
        /// </summary>

        public void Submit(ushort teamID)
        {
            foreach (KeyValuePair<Vector2Int, Cell> cell in FadeCells)
            {
                Grid[cell.Key.x, cell.Key.y] = cell.Value;
                cell.Value.block.Animate(1f);
            }

            if (CellOccupation.TryGetValue(teamID, out int count))
            {
                CellOccupation[teamID] = count + FadeCells.Count;
            }
```

```
            else CellOccupation.Add(teamID, FadeCells.Count);
            m_occupiedVolume += FadeCells.Count;
            FadeCells.Clear();
        }

        public void SendShapeSkip()
        {
            Message message = Message.Create(MessageSendMode.reliable,
MulticlientID.submitShape);

            message.AddBool(true); // Is it was skip action

            if (NetworkManager.IsLocal) NetworkManager.Server.SendToAll(message,
NetworkManager.Client.Id);
            else NetworkManager.Client.Send(message);
        }

        public void SendShapeSubmit(int minX, int maxX, int minY, int maxY, ushort team)
        {
            Message message = Message.Create(MessageSendMode.reliable,
MulticlientID.submitShape);

            message.AddBool(false); // Is it was skip action
            message.AddInt(minX);
            message.AddInt(maxX);
            message.AddInt(minY);
            message.AddInt(maxY);
            message.AddUShort(team);

            if (NetworkManager.IsLocal) NetworkManager.Server.SendToAll(message,
NetworkManager.Client.Id);
            else NetworkManager.Client.Send(message);
        }
```

```
        public class Cell
        {
            public ushort team = 0;
            public GridBlock block = null;

            public void Destroy()
            {
                team = 0;
                block.Destroy();
                block = null;
            }
        }
    }
}
```

GridBlock

```
using UnityEngine;

namespace Games.SQZone
{
    public class GridBlock : MonoBehaviour
    {
        public const float MagnitudeSpeed = 4.5f;
        public int Team { get; set; }
        public SpriteRenderer Sprite => m_sprite;


        [SerializeField] protected SpriteRenderer m_sprite;


        float scale;
        public void Animate(float scale)
        {
```

```
            if (transform.localScale.x != scale)
            {
                this.scale = scale;
                enabled = true;
            }
        }
        private void Update()
        {
            if (SoftMath.Approximately(transform.localScale.x, 0f))
            {
                enabled = false;
                transform.localScale = new(scale, scale);
            }

            float delta = scale - transform.localScale.x;
            delta *= MagnitudeSpeed * Time.deltaTime;
            delta = transform.localScale.x + delta;
            transform.localScale = new(delta, delta);
        }
        public void Destroy() => Destroy(gameObject);
    }
}
```

Shape

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEngine;

namespace Games.SQZone
{
    public class Shape : MonoBehaviour
    {
        public const float MaxAnimationDuration = 6f;
```

```
        public const float AnimationSpeed = 12f;
        public static readonly List<Shape> m_activeShapes = new();

        public float SizeX => m_shapeX;
        public float SizeY => m_shapeY;

        [SerializeField] protected SpriteRenderer m_spriteRenderer;
        [SerializeField] protected Vector2 m_offset;

        protected float m_shapeX;
        protected float m_shapeY;



        float time;
        Vector3 targetPosition;
        public Shape CreateShapeFromInstance(int maxSizeX, int maxSizeY, Transform
parent = null)
        {
            Shape shape = Instantiate(gameObject, parent).GetComponent<Shape>();
            m_activeShapes.Add(shape);

            Vector2 size = new(x:
Mathf.Round((float)SQZoneLocalManager.Instance.Random.NextDouble() * maxSizeX),
                               y:
Mathf.Round((float)SQZoneLocalManager.Instance.Random.NextDouble() * maxSizeY));
            if (size.x < 1) size.x = 1;
            if (size.y < 1) size.y = 1;
            shape.m_spriteRenderer.size = size;
            shape.m_shapeX = size.x;
            shape.m_shapeY = size.y;
            shape.m_offset = new((size.x - 1) % 2 * 0.5f, (size.y - 1) % 2 * 0.5f);

            return shape;
        }
```

```
        private void Update()
        {
            transform.localPosition = transform.localPosition + AnimationSpeed *
Time.deltaTime * (targetPosition - transform.localPosition);

            time += Time.deltaTime;
            if (time > MaxAnimationDuration)
            {
                enabled = false;
            }
        }
        public void SetRelativePosition(Vector2 position)
        {
            targetPosition = new(GetXRelativeToGrid(position.x),
GetYRelativeToGrid(position.y), transform.localPosition.z);
            time = 0;
            enabled = true;
        }
        public void PlaceShape(Vector3 position)
        {
            SetRelativePosition(position);

            //m_activeShapes.Remove(this);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private float GetXRelativeToGrid(float sourceX)
            => Mathf.Round(sourceX) + m_offset.x;
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private float GetYRelativeToGrid(float sourceY)
            => Mathf.Round(sourceY) + m_offset.y;

        public static bool RaycastShapes(Vector3 position, out Shape shape)
        {
```

```
            Vector2 halfShapeSize;
            foreach (Shape s in m_activeShapes)
            {
                halfShapeSize = s.m_spriteRenderer.size * 0.5f;
                if (s.transform.localPosition.x + halfShapeSize.x < position.x) continue;
                if (position.x < s.transform.localPosition.x - halfShapeSize.x) continue;
                if (s.transform.localPosition.y + halfShapeSize.y < position.y) continue;
                if (position.y < s.transform.localPosition.y - halfShapeSize.y) continue;

                shape = s;
                return true;
            }
            shape = null;
            return false;
        }
    }
}
```

Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МУЛЬТИПЛЕЄРНОЇ КОМП'ЮТЕРНОЇ ГРИ «TRICKY SQUARES» З ВИКОРИСТАННЯМ C# ТА UNITY

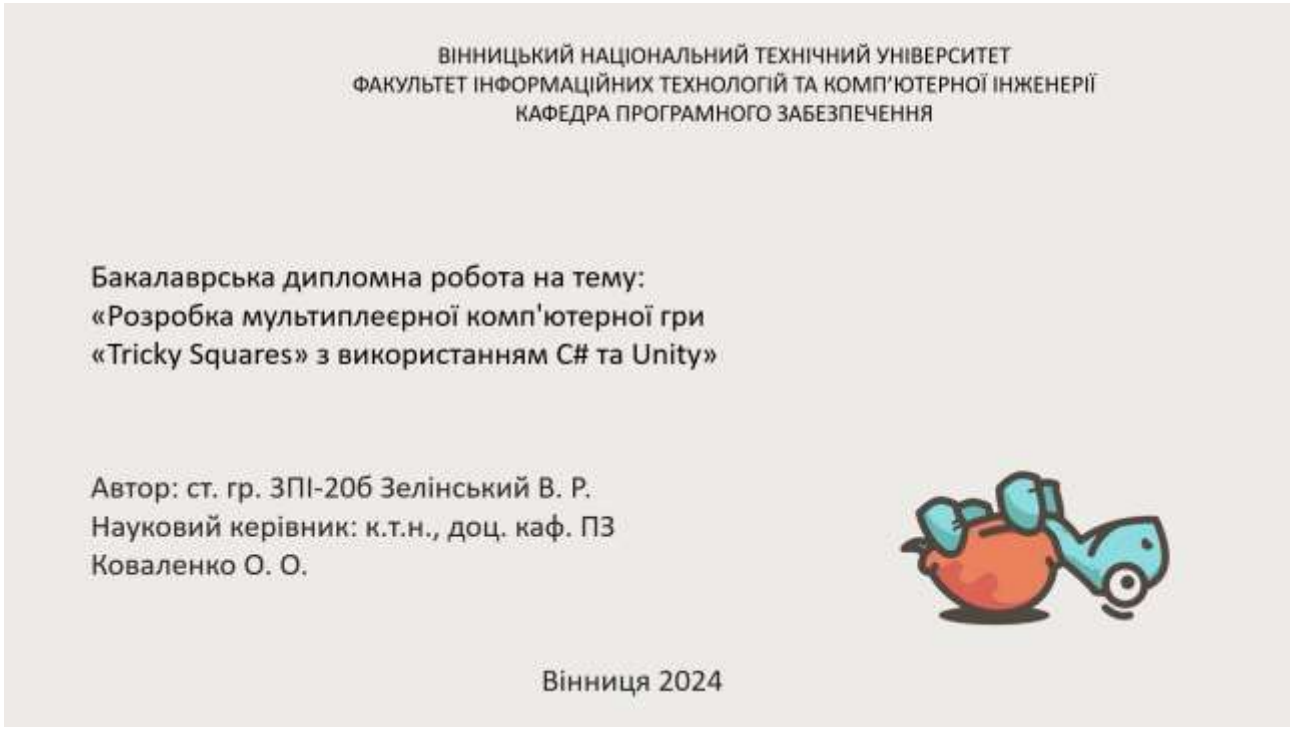

Рисунок Г.1 – Титульний слайд

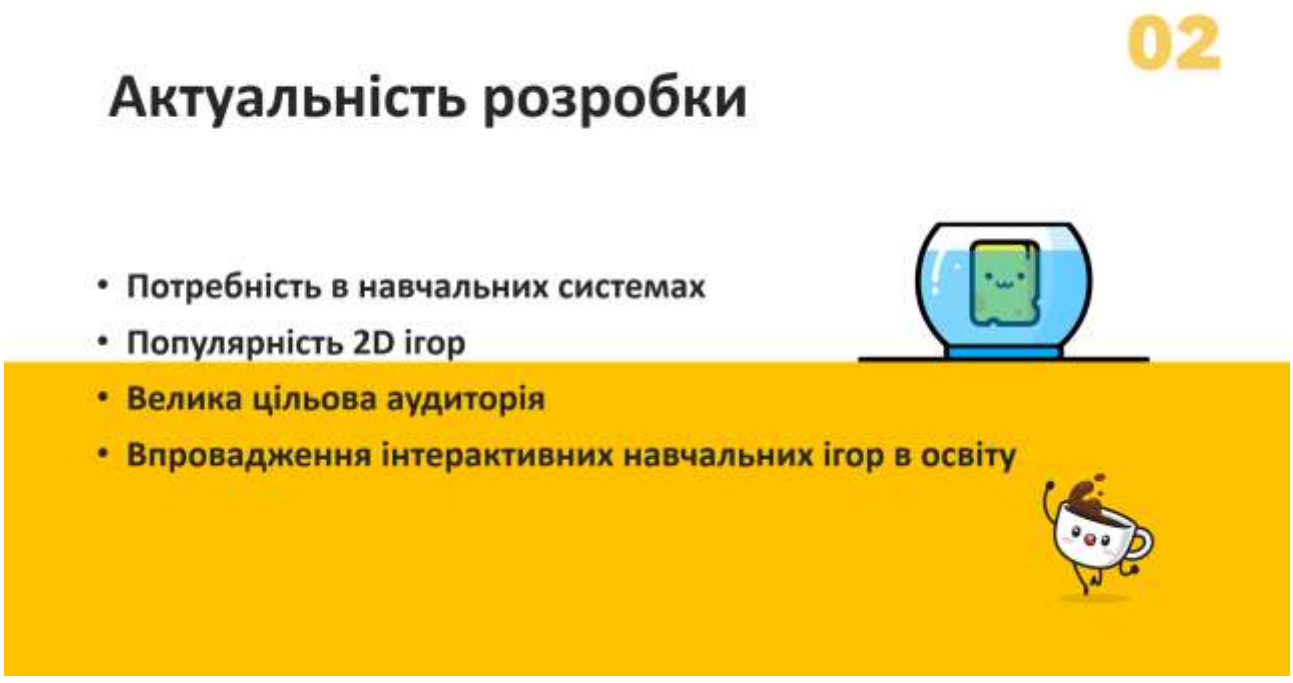

Рисунок Г.2 – Слайд «Актуальність розробки»

Рисунок Г.3 – Слайд «Мета, предмет та об’єкт дослідження»

Рисунок Г.4 – Слайд «Постановка задачі»

Рисунок Г.5 – Слайд «Новизна»

Рисунок Г.6 – Слайд «Практична цінність»

Рисунок Г.7 – Слайд «Tetris»

Рисунок Г.8 – Слайд «Tricky Towers»

Рисунок Г.9 – Слайд «Candy crush»

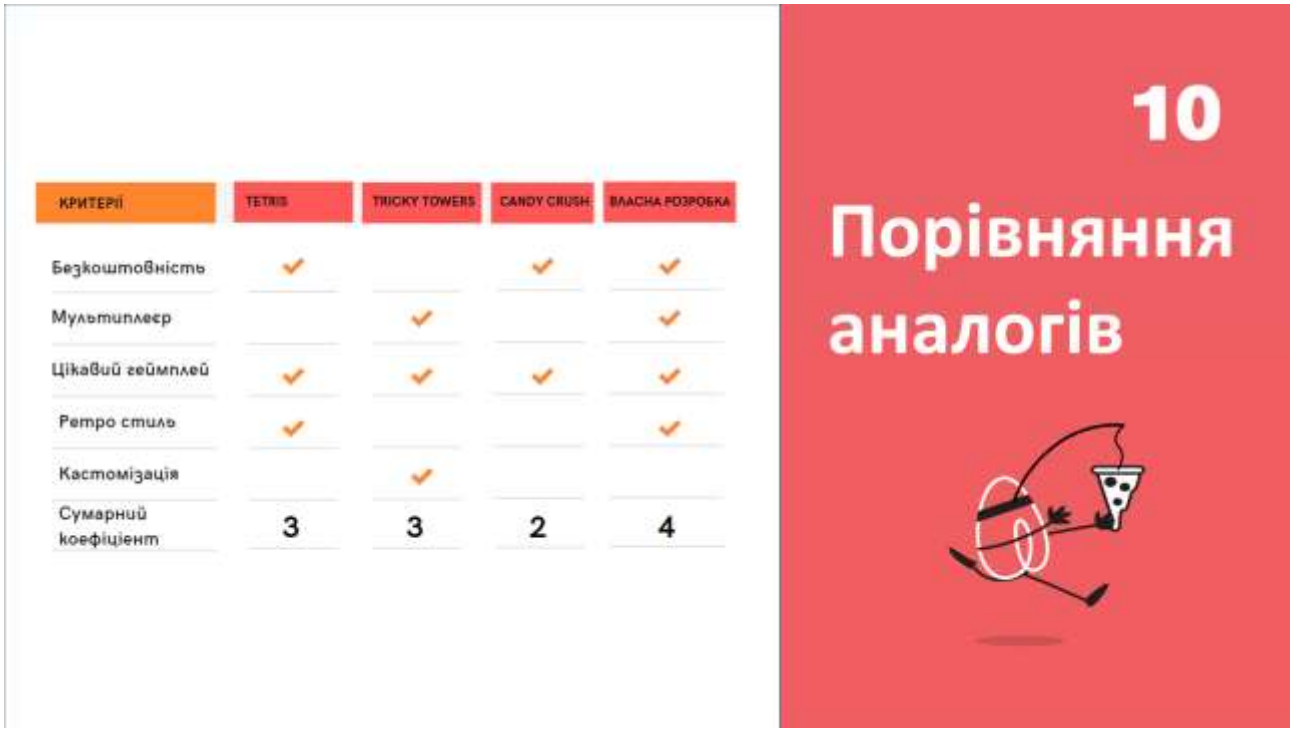

Рисунок Г.10 – Слайд «Порівняння аналогів»

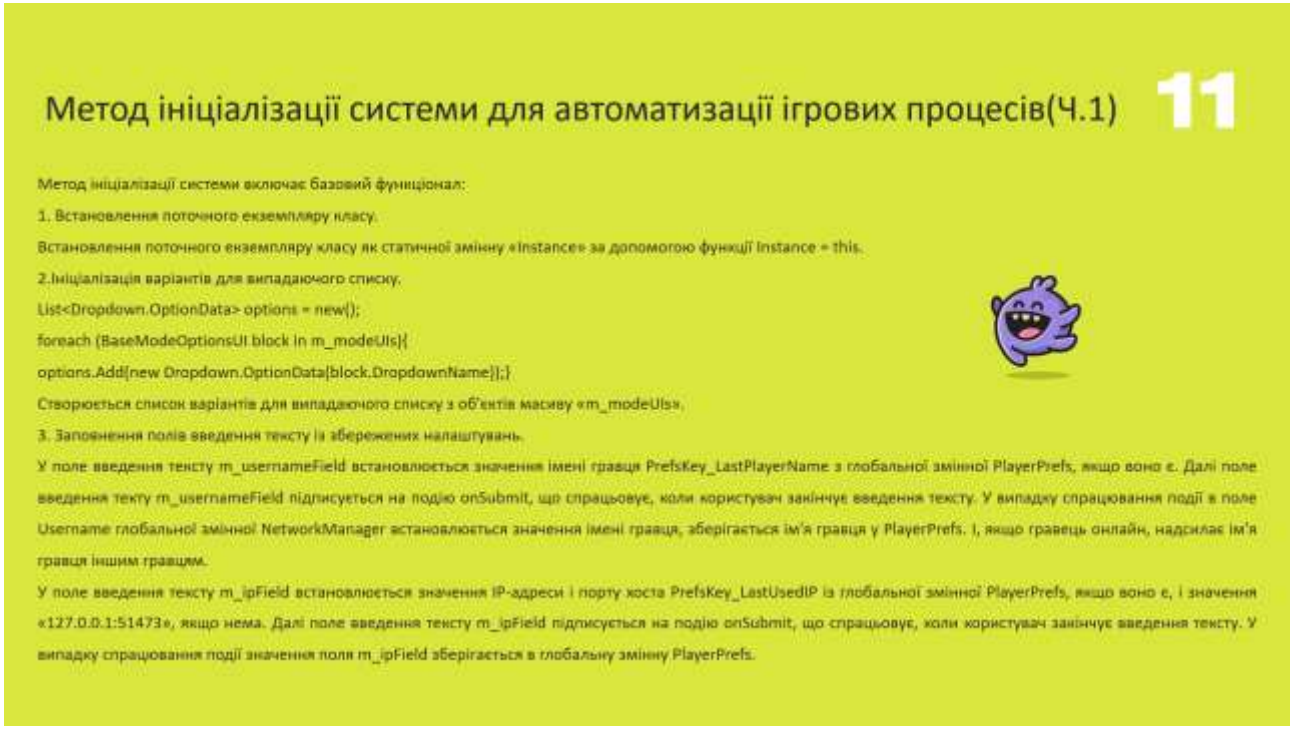

Рисунок Г.11 – Слайд «Метод ініціалізації системи для автоматизації ігрових процесів (ч.1)»

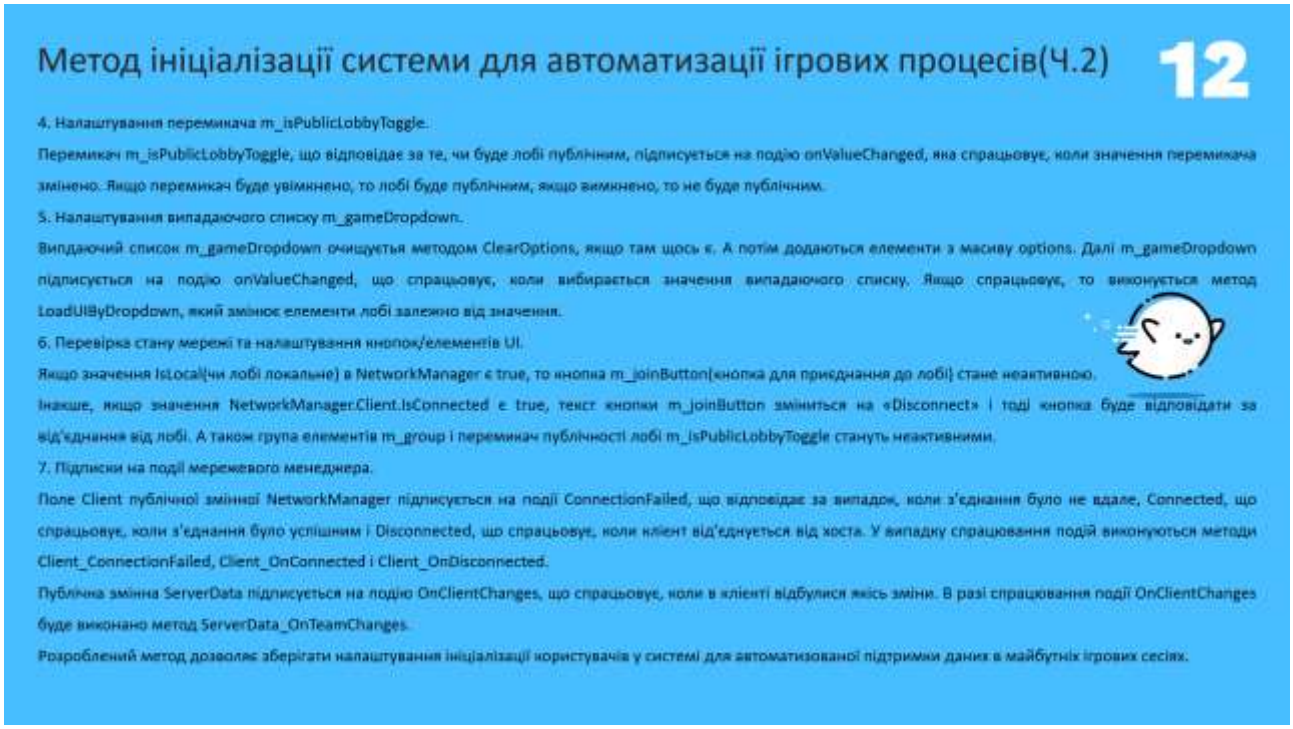

Рисунок Г.12 – Слайд «Метод ініціалізації системи для автоматизації ігрових процесів (ч.2)»

Рисунок Г.13 – Слайд «Блок-схема алгоритму роботи методу ініціалізації системи для автоматизації ігрових процесів»

Рисунок Г.14 – Слайд «Алгоритм роботи мультиплеєра»

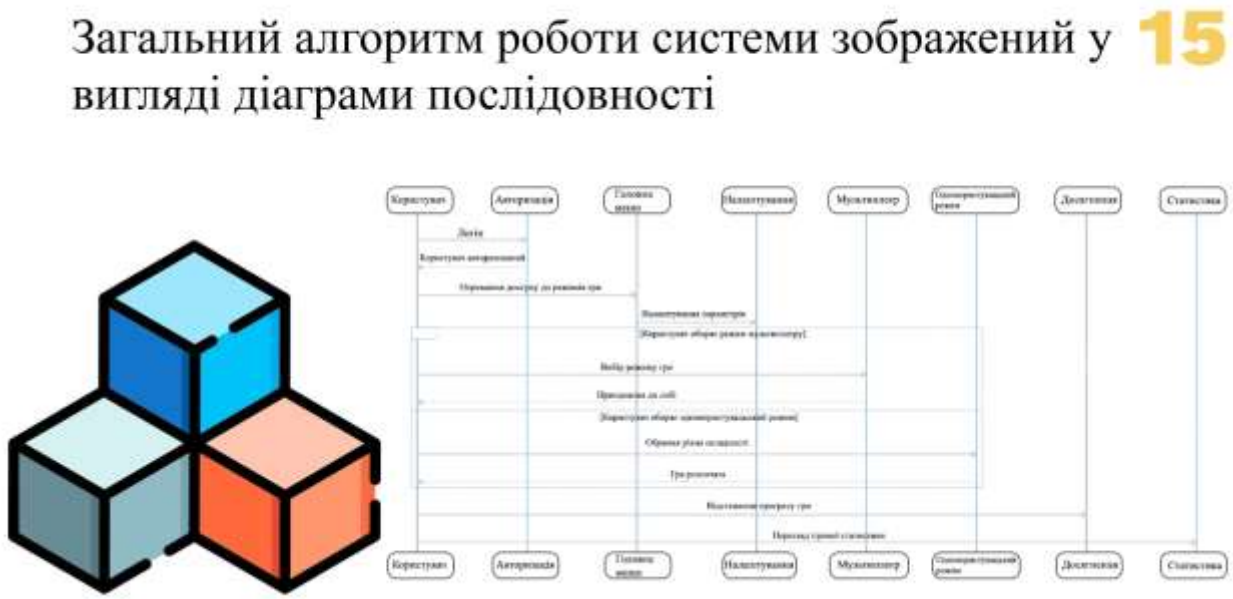

Рисунок Г.15 – Слайд «Загальний алгоритм роботи системи зображений у вигляді діаграми послідовності»

Рисунок Г.16 – Слайд «Модель системи у вигляді діаграми варіантів використання»

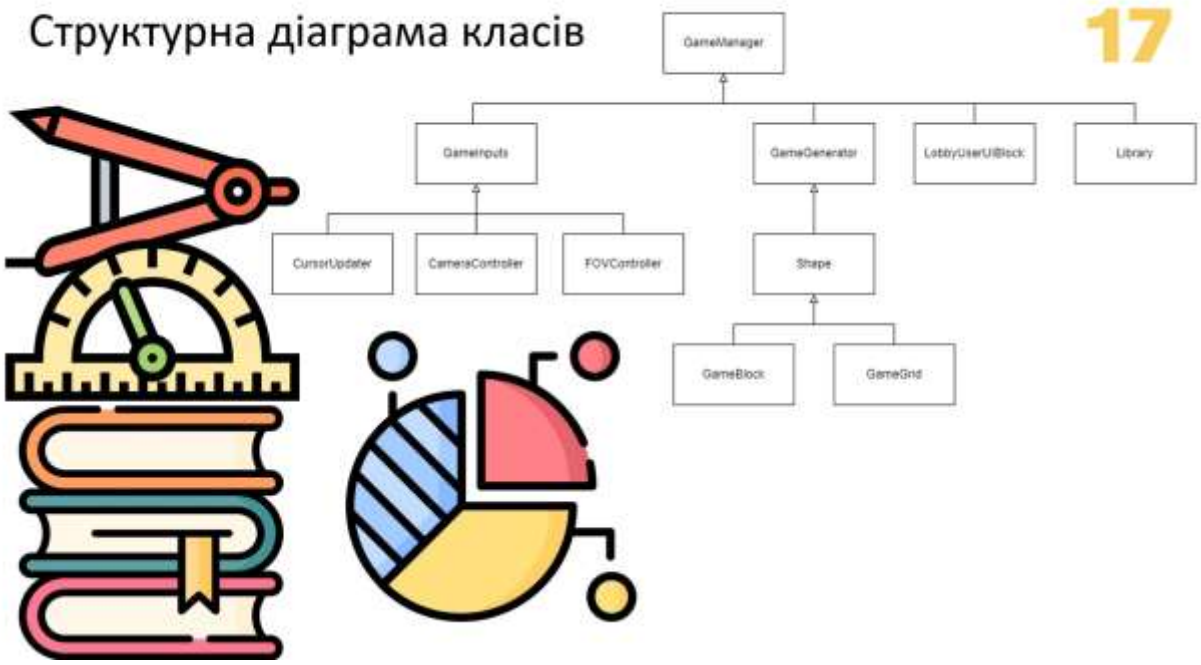

Рисунок Г.17 – Слайд «Структурна діаграма класів»

Рисунок Г.18 – Слайд «Використані технології»

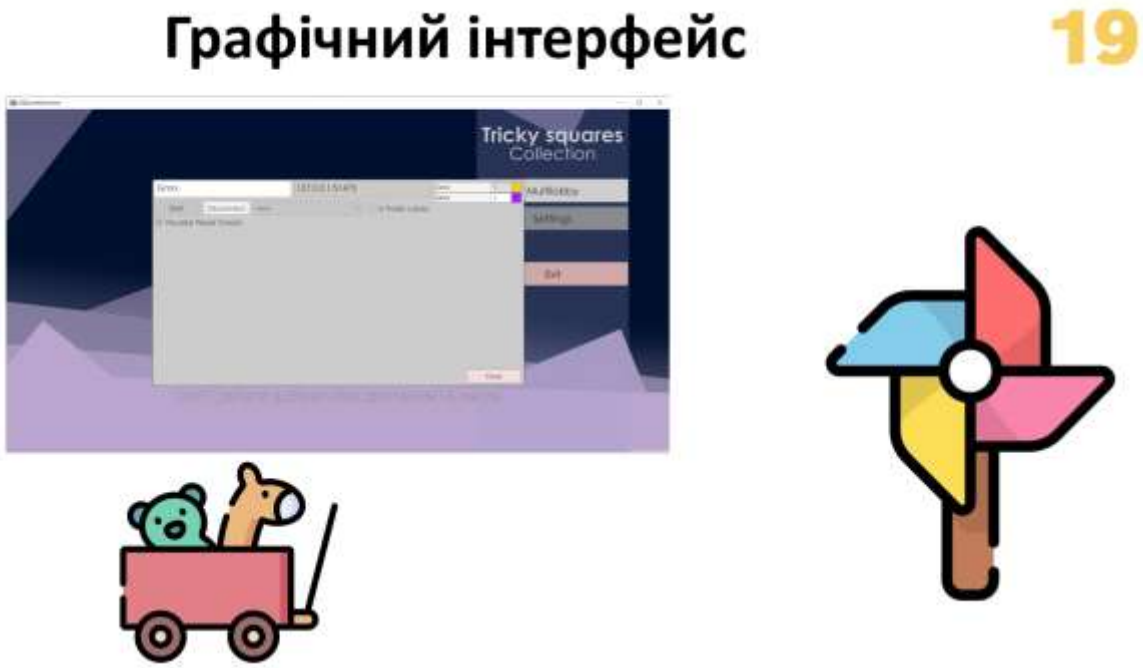

Рисунок Г.19 – Слайд «Графічний інтерфес»

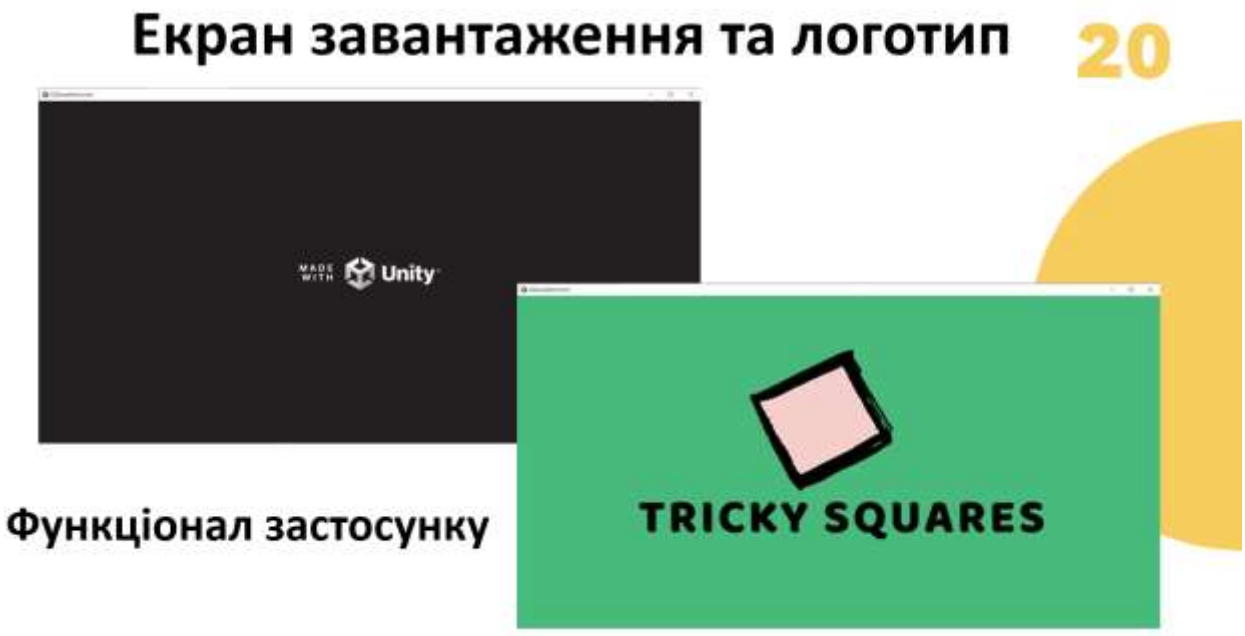

Рисунок Г.20 – Слайд «Екран завантаження та логотип»

Рисунок Г.21 – Слайд «Приєднання фігур до своєї території»

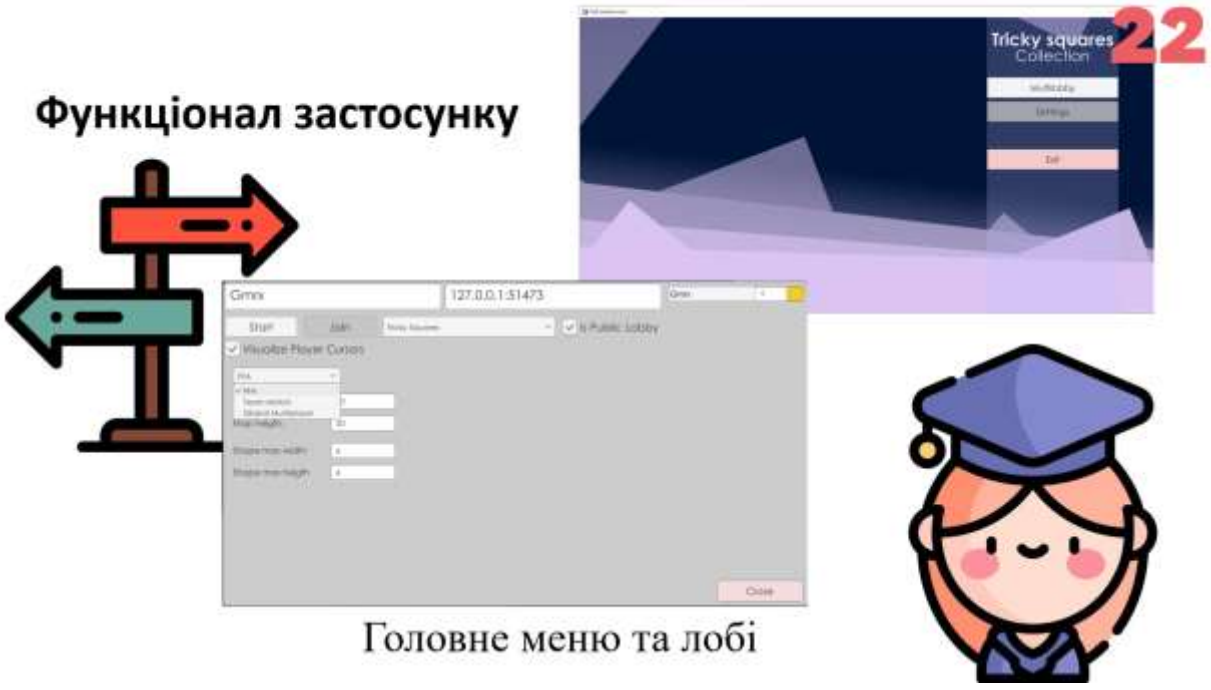

Рисунок Г.22 – Слайд «Головне меню та лобі»

Рисунок Г.23 – Слайд «Ігровий процес»

Рисунок Г.24 – Слайд «Завершення»

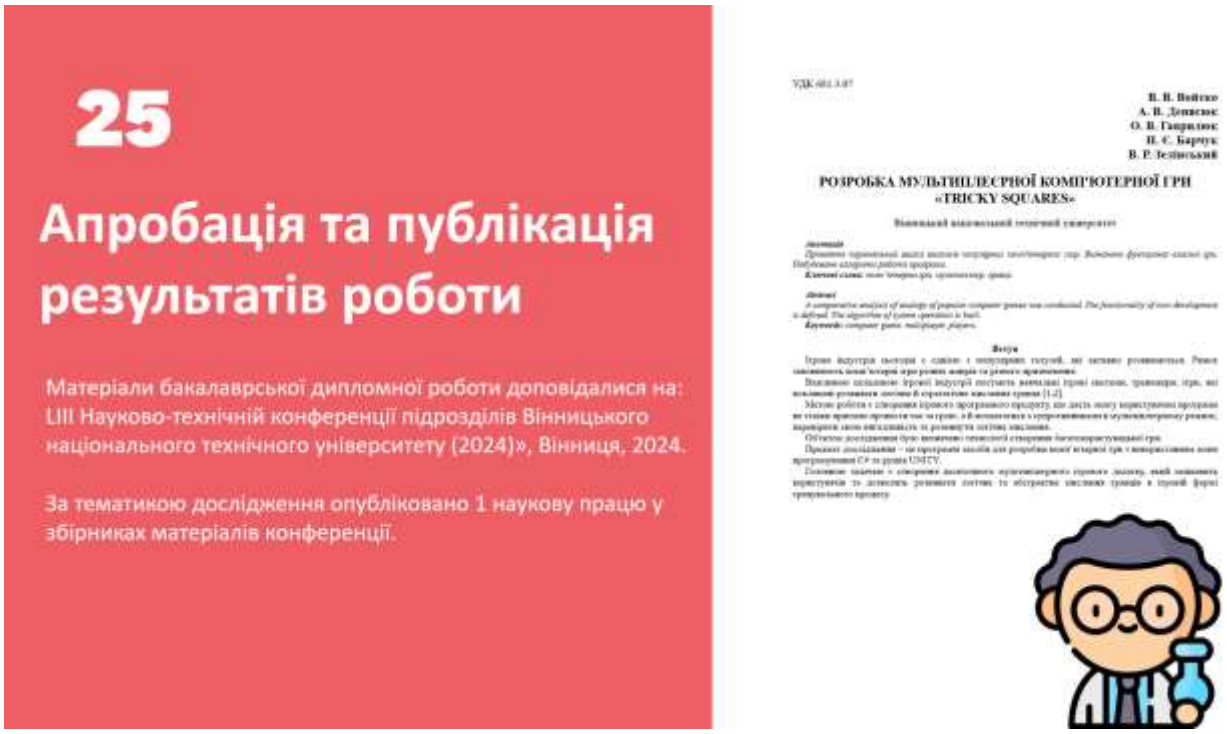

Рисунок Г.25 – Слайд «Апробація та публікація результатів роботи»

Рисунок Г.26 – Слайд «Висновки»

Рисунок Г.27 – Фінальний слайд