

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Конструктор та симулятор настільних ігор»

Виконав студент групи ІПЗ-20-1
КОЧУБЕЙ Костянтин Максимович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного
забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КОЧУБЕЙ Костянтин Максимович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Конструктор та симулятор настільних ігор»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **процес створення та симуляції настільних ігор за допомогою програмного забезпечення, яке дозволяє користувачам розробляти, тестувати та вдосконалювати ігри.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	05.03.2024р.	05.03.2024р.
2	Власенко О.В.	27.03.2024р.	27.03.2024р.
3	Власенко О.В.	06.05.2024р.	06.05.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з конструктора та симулятора настільних ігор та пояснювальної записки. Пояснювальна записка до випускної роботи містить 69 сторінок, 24 ілюстрації та 16 таблиць.

Метою роботи є розробка онлайн-сервісу для гри в настільні ігри онлайн, що також включає функції створення власних ігор, їх програмування для онлайн-сервісу, викладання у публічний доступ, тестування та спілкування з іншими користувачами.

В роботі визначено основні завдання на розробку сервісу, проаналізовано аналоги розробленого продукту та визначено головні потреби потенціальних користувачів. Наведена інструкція та можливі сценарії користування, загальна структура онлайн-сервісу, опис алгоритму роботи інструменту програмування ігор, опис об'єктної структури системи (діаграма класів), реалізованих з використанням мови програмування C#, опис структури бази даних.

Ключові слова: НАСТІЛЬНІ ІГРИ, СТВОРЕННЯ ВЛАСНИХ ІГОР, ОНЛАЙН-СЕРВІС, СПІЛКУВАННЯ, ПУБЛІЧНИЙ ДОСТУП.

					ІПЗ.КР.Б-121-24-ІПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Конструктор та симулятор настільних ігор	Літ.	Арк.	Аркушів
Розроб.		Кочубей К.М.					3	70
Керівник		Власенко О.В.						
Н. контр.		Тичина О.П.				Житомирська політехніка, група ІПЗ-20-1		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis contains a tabletop game constructor and simulator and an explanatory note. The explanatory notes for the thesis are made up of 69 pages, 24 illustrations, 16 tables.

The goal of the thesis is the development of an online service for playing board games, which includes tools for creating one's own games, coding them for the online service, bringing them out for public access, testing them as well as providing communication between developers.

The main tasks for developing the service and the main needs of the potential user have been identified and analogues of the developed product have been analysed. Instructions and possible usage scenarios are demonstrated, as well as the basic structure of the online service, the description of the algorithm of the game programming tool, the description of the system object structure (class diagram), all implemented in the C# programming language, and the description of the database is provided.

The developed online service is already being operated by users from various countries.

Keywords: BOARD GAMES, CREATION OF ONE'S OWN GAMES, ONLINE SERVICE, COMMUNICATION, PUBLIC ACCESS.

		Кочубей К.М.			ІІЗ.КР.Б-121-24-ІІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ КОНСТРУКТОРА ТА СИМУЛЯТОРА НАСТІЛЬНИХ ІГОР.....	9
1.1 Опис проекту та постановка задачі	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури симулятора настільних ігор	17
1.4 Обґрунтування вибору засобів реалізації.....	19
Висновки до першого розділу.....	20
РОЗДІЛ 2 ПРОЄКТУВАННЯ КОНСТРУКТОРА НАСТІЛЬНИХ ІГОР.....	20
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	21
2.2. Розробка бази даних системи.....	27
2.3. Проектування та реалізація алгоритмів роботи системи	32
2.4. Реалізація функціоналу конструктора настільних ігор	36
Висновки до другого розділу.....	41
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З КОНСТРУКТОРОМ ТА СИМУЛЯТОРОМ НАСТІЛЬНИХ ІГОР	42
3.1. Порядок встановлення та налаштування параметрів платформи....	42
3.2. Структура інтерфейсу додатку та порядок роботи.....	43
3.3 Тестування роботи додатку.....	50
Висновки до третього розділу.....	53
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	55
ДОДАТКИ.....	57

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MVC — Model-View-Controller архітектурний шаблон.

IDE — інтегроване середовище розробки.

СУБД — система управління базами даних.

UML — уніфікована мова моделювання.

LUA — скриптова мова програмування.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. Настільні ігри є популярним видом проведення дозвілля вже багато століть. В сучасному світі багато людей захоплюються ними, колекціонують та слідкують за їхнім ринком. Одна з головних переваг настільних ігор є те, що будь-яку гру можна змінити під себе, або ж навіть створити власну гру, маючи лише папір та олівці. Також настільні ігри мають багато елементів соціальної взаємодії, що робить їх зручним інструментом для соціалізації. Але останні роки зустрічатися з друзями, щоб зіграти вживу все складніше, тому багато настільних ігор стоять на полицях, а можливостей протестувати власні ігри – стає все менше.

Актуальність обраної теми полягає в тому, що онлайн-сервіс настільних ігор дозволить гравцям з різних міст та різних країн проводити час разом, а творцям – тестувати власні ігри не виходячи з дому. Публічна бібліотека забезпечить соціальну взаємодію людей з різних куточків світу та допоможуть знайти друзів зі схожими інтересами.

Новизна теми. В сучасному світі любителі настільних ігор мають багато потреб, які не можуть задовольнити. Основною серед таких потреб є можливість створення власних ігор, використовуючи зручні інструменти, які допоможуть розробці, тестуванню та впровадженню в широкі маси.

Тому новизною обраної теми є задоволення потреб любителів настільних ігор, завдяки створенню багатофункціонального онлайн-сервісу.

Метою випускної роботи є аналіз існуючих онлайн-сервісів настільних ігор, оцінення їх можливостей, складання списку затребуваних функцій та розробка власного аналогу.

Встановлена мета обумовлює наступні завдання:

- збір інформації про затребувані функції потенціальних користувачів;
- проведення аналізу реалізованих функцій в існуючих сервісах;
- виділення необхідних функцій для власного аналога;

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

– розробка кожної функції окремо та об’єднання їх в один онлайн-сервіс;

Об’єктом дослідження є технології реалізації власних настільних ігор та можливості використання створених ігор як на самому онлайн-сервісі, так і вдома.

Предметом дослідження є використання технології для створення багатофункціонального онлайн-сервісу настільних ігор, який задовольняє потреби гравців та творців.

В роботі було використано методи об’єктно-орієнтованого проектування, теорії реляційних баз даних, дизайну інтерфейсів, а також методи роботи з графічними движками та моделями полігонів.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ КОНСТРУКТОРА ТА СИМУЛЯТОРА НАСЛІЛЬНИХ ІГОР

1.1 Опис проекту та постановка задачі

Метою кваліфікаційної роботи бакалавра є створення додатку для конструювання і симуляції настільних ігор на платформі Unity, з використанням сучасних технологій, що забезпечить зручність користування та можливість масштабування в майбутньому.

На сьогоднішній день настільні ігри залишаються одними з найпопулярніших форм розваг у світі. Їхня вікова і соціальна доступність [1] робить їх привабливими для широкого кола людей, незалежно від їхнього віку чи інтересів. З виникненням сучасних технологій і зростанням доступу до Інтернету, настільні ігри відкривають нові горизонти через онлайн-симулятори.

Онлайн-симулятори настільних ігор представляють собою інтерактивні платформи, які дозволяють користувачам грати в улюблені настільні ігри прямо зі своїх комп'ютерів або мобільних пристроїв. Це означає, що ви можете насолоджуватись грою у шахи, монополію, каркс, або будь-яку іншу гру, навіть якщо ваші друзі або супротивники знаходяться на іншому кінці світу [2].

Однією з головних переваг таких симуляторів є їхня доступність. Вам не потрібно організовувати зустрічі чи чекати на вільну кімнату у кафе з настільними іграми, щоб насолоджуватись грою з друзями або сім'єю. Ви можете грати в будь-який час і в будь-якому місці, де є Інтернет-з'єднання. Це особливо зручно для тих, хто має розмірковувати про розташування та часові обмеження.

Онлайн-симулятори настільних ігор також відрізняються за рівнем реалізму. Деякі з них надають дуже деталізовані візуальні та звукові ефекти, щоб забезпечити максимально імерсивний досвід гри. Інші можуть бути

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

спрощеними версіями, які акцентують увагу на базових правилах гри і швидкому доступі до розваги. Це різноманіття дозволяє кожному знайти свій варіант і насолоджуватись грою у форматі, який відповідає їхнім потребам та вподобанням.

Крім того, деякі онлайн-симулятори надають можливість не тільки грати з випадковими супротивниками, але й відтворювати ігрові сесії зі своїми друзями або знайомими. Це створює можливість спілкування та взаємодії, подібної до тієї, що ви можете отримати під час особистої гри за столом. Такі ігри сприяють не лише розвагам, а й підтримці соціальних зв'язків, особливо в умовах віддалених комунікацій.

Саме тому онлайн-симулятори настільних ігор представляють собою важливий елемент сучасної ігрової культури, що поєднує традиційні аспекти настільних ігор з вигодами технологічного прогресу. Їхня популярність продовжує зростати, розширюючи коло людей, які можуть насолоджуватись цими захоплюючими ігровими досвідами.

Серед головних функцій головного проекту можна зазначити такі пункти, як:

1. Перегляд та встановлення імпортованих настільних ігор: Ця функція дозволяє користувачам переглядати широкий асортимент настільних ігор, які доступні для гри в онлайн-симуляторі. Вони можуть переглядати опис ігор, щоб обрати ігри, які їм цікаві, та встановити їх для гри.

2. Гра з іншими користувачами у встановлені ігри: Користувачі можуть запрошувати інших користувачів на гру у будь-яку імпортовану настільну гру з загальної бібліотеки. Для цього буде достатньо передати іншому користувачу код створеної кімнати.

3. Створення власних ігор та додання їх у глобальну бібліотеку: Ця функція надає користувачам можливість створювати власні настільні ігри або модифікувати існуючі, додавати власні правила чи варіації. Після цього їх можна опублікувати в глобальній бібліотеці, де інші гравці зможуть їх знайти та спробувати.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

4. Програмування власних ігор завдяки вбудованим інструментам: Онлайн-симулятор може надавати користувачам інструменти для програмування власних ігор за допомогою зручних інтуїтивних інструментів.

В результаті повинен бути створений додаток, який дозволить створювати власні настільні ігри та грати в існуючі з друзями.

Розробку додатку можна поділити на наступні етапи:

1. Побудова архітектури бази даних.
2. Побудова структури користувацької частини додатку.
3. Розробка інтерфейсу.
4. Розробка функціоналу пов'язаного з базами даних:
 - створення логіки авторизації користувачів;
 - розробка системи синхронізації списку ігор та пазлів в додатку з базою даних.

5. Розробка системи синхронізації об'єктів між різними користувачами в мультиплеєрі.

6. Розробка основного алгоритму:
 - побудова ієрархічної структури взаємодії між класами;
 - реалізація скриптів для роботи пазлів;
 - розробка інструментів для експорту та імпорту алгоритмів ігор.
7. Тестування додатку:
 - створення настільної гри з алгоритмом його роботи за допомогою розроблених інструментів та її відтворення;
 - тестування інтерфейсу.

1.2 Аналіз аналогів програмного продукту

Для створення практичного та зручного у використанні продукту було проведено дослідження аналогічних додатків та сервісів. Це завдання включало загальний огляд дизайну, можливостей та зручності їх

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

використання. У результаті було визначено переваги та недоліки цих аналогів.

Оскільки кожен гравець у настільні ігри очікує різних функцій від онлайн-сервісів, розробники створили багато продуктів, кожен з яких має свій унікальний підхід. Тому під час дослідження аналогів було визначено кілька різних критеріїв, що описують кожен із сервісів з різних сторін.

Першим серед аналогів є найвідоміший симулятор настільних ігор – “Tabletop Simulator” (рис. 1.1). Він є одним із провідних додатків для гри в настільні ігри онлайн, який приваблює користувачів своєю реалістичною графікою та широкими можливостями. Додаток пропонує інтуїтивно зрозумілий інтерфейс, що дозволяє гравцям легко взаємодіяти з віртуальним ігровим столом та його елементами. Однією з головних переваг Tabletop Simulator є підтримка величезної кількості настільних ігор, включаючи як популярні комерційні проекти, так і інді-розробки. Користувачі мають можливість створювати власні ігри, налаштовувати ігровий процес за допомогою скриптів та ділитися своїми творіннями зі спільнотою.



Рисунок 1.1 Скриншот програми “Tabletop Simulator”

Tabletop Simulator також пропонує багатокористувацький режим, який дозволяє грати з друзями або випадковими гравцями з усього світу. Вбудована система чату та голосового зв'язку полегшує спілкування між гравцями під час гри, що створює атмосферу, близьку до реальної гри за столом. Додаток надає широкий вибір налаштувань, що дозволяють користувачам персоналізувати свій ігровий досвід, включаючи зміну фону, столів, фігурок та інших елементів.

Незважаючи на численні переваги, Tabletop Simulator має деякі недоліки. Щоб надати грі певного алгоритму роботи, користувачу потрібно знати мову скриптів LUA. Крім того, додаток вимагає потужного комп'ютера для стабільної роботи, що може бути проблемою для деяких користувачів.

Наступним розглянутим аналогом є платформонезалежний сервіс “Tabletopia” (рис. 1.2). Він пропонує доступ через веб-версію, додатки для Steam, iOS і Android, що робить її досить доступною для користувачів на різних пристроях.



Рисунок 1.2. Скриншот програми “Tabletopia”

Основна перевага Tabletopia полягає в великому виборі настільних ігор, які включають як класичні, так і сучасні творіння. Користувачі можуть не лише грати в існуючі ігри, але й створювати власні, використовуючи доступні інструменти для розробки.

Однак, для повного доступу до всіх ігор і можливостей платформи, може знадобитися підписка. Також інтерфейс може здаватися складним для новачків, оскільки створення власних ігор вимагає певного часу на оволодіння інструментами.

Board Game Arena (BGA) - це популярна онлайн-платформа для настільних ігор, яка пропонує широкий вибір настільних ігор, доступних для гравців по всьому світу (рис. 1.3). Вона служить зручним середовищем для віртуального геймінгу, де користувачі можуть грати у режимі реального часу або поштовхами, знаходити опонентів для гри або приєднуватися до існуючих столів.



Рис. 1.3. Скриншот процесу гри сайту “BoardGameArena”

Однак BGA також має свої недоліки. Один з них - обмежена графіка та анімація. Порівняно з іншими платформами візуальні ефекти і деталізація графіки на BGA можуть виглядати менш привабливо для деяких користувачів, які віддають перевагу більш реалістичному або ефектному візуальному виконанню ігор.

Інший недолік - відсутні можливості кастомізації. Це може обмежувати творчість користувачів та їх можливість адаптувати ігри під власні вимоги.

Крім того, деякі функції та ігри на BGA доступні лише за платної підписки або окремої оплати, що може вплинути на доступність для

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

користувачів з обмеженим бюджетом або тих, хто переважає безкоштовні альтернативи.

Scratch — це веб-платформа, спрямована на навчання програмування шляхом використання візуальних блоків (рис. 1.4). Основна перевага Scratch полягає в його простоті і доступності для новачків у програмуванні. Користувачі можуть легко створювати анімації, інтерактивні історії та ігри, складаючи різні блоки разом без потреби писати код вручну. Це робить Scratch ідеальним інструментом для освітніх цілей, особливо для дітей і молоді.

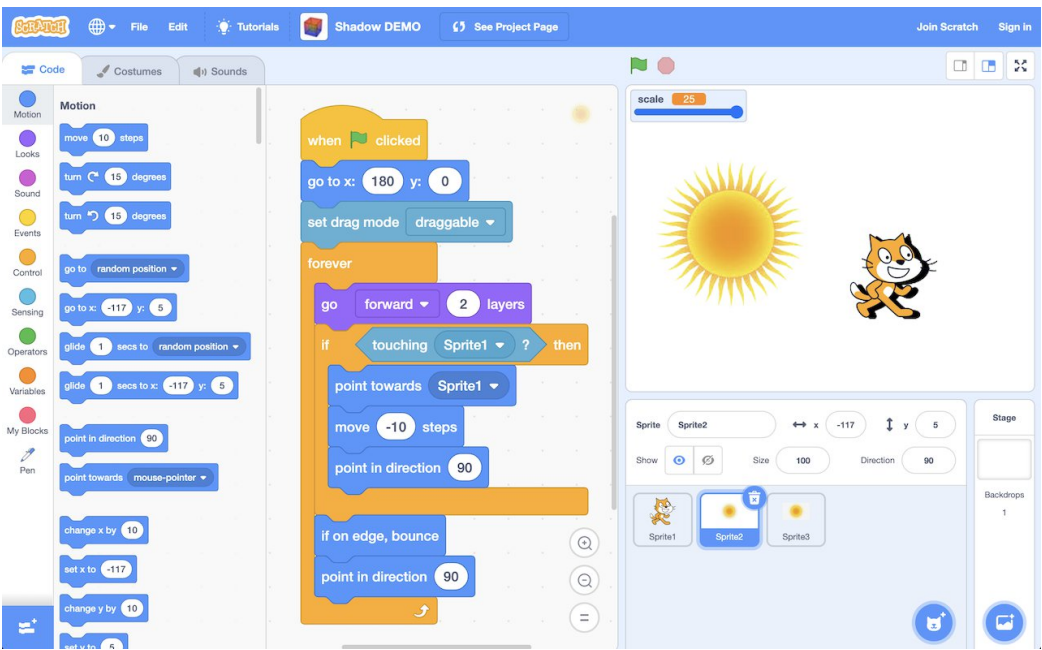


Рис. 1.4. Скриншот процесу конструювання в сервісі “Scratch”

Однак у Scratch є деякі недоліки. Через його візуальний підхід, іноді можуть бути обмеження у реалізації складних алгоритмів чи деталізованих програм. Для більш досвідчених користувачів програмування Scratch може здаватися занадто обмеженим для розвитку професійних навичок у програмуванні, які потребують використання текстових мов програмування.

Scratch, в рамках сервісів для конструювання ігор, не надає спеціалізованих інструментів, призначених спеціально для створення настільних ігор. Він скоріше орієнтований на створення інтерактивних ігор та програм з використанням блок-схем та скриптів для програмування. Такі

інструменти зазвичай менш підходять для реалізації унікальних механік та компонентів, що характерні для класичних настільних ігор.

Для зручного порівняння вище описаних додатків була створена таблиця 1.1.

Таблиця 1.1

Порівняльна характеристика аналогів програмного продукту

Характеристика	Tabletop Simulator	Tabletopia	BGA	Scratch
Мультиплеєр	Так	Так	Так	Ні
Зд елементи	Так	Так	Ні	Ні
Створення власних ігор	Так	Так	Ні	Так
Складність створення алгоритму ігор	Висока	Висока	Неможлива	Низька
Орієнтованість на настільні ігри	Так	Так	Так	Ні
Інтуїтивність інтерфейсу	Висока	Висока	Середня	Висока
Навчальний потенціал	Ні	Ні	Ні	Так

Порівнюючи чотири сервіси для настільних ігор — Tabletop Simulator, Tabletopia, Board Game Arena (BGA) та Scratch, можна виділити кілька ключових відмінностей і сильних сторін кожного з них. Tabletop Simulator і Tabletopia відрізняються можливістю мультиплеєру і використанням 3D елементів, що робить їх ідеальними для іммерсивного геймплею. BGA відмічається зручністю у користуванні та орієнтацією на класичні настільні ігри, але не має можливості створення власних ігор. Scratch, хоч і не призначений для настільних ігор, відзначається низькою складністю створення алгоритмів і великим навчальним потенціалом завдяки інтуїтивному інтерфейсу конструювання алгоритму гри.

Отже, щоб покращити конструктор та симулятор настільних ігор потрібно врахувати ці аспекти для створення найбільш зручного та універсального середовища для онлайн настільних ігор. Повинно бути надано можливість мультиплеєру і використання 3D елементів для покращення іммерсії гравців. Крім того, потрібно врахувати можливість створення власних ігор з низькою складністю використання, щоб зробити процес доступним для широкого кола користувачів. Інтуїтивний інтерфейс та високий навчальний потенціал також будуть важливими аспектами для сприяння швидкому вивченню та комфортному користуванню додатком усіма категоріями користувачів.

1.3 Вибір архітектури симулятора настільних ігор

Перед поясненням вибору архітектури для даного проекту важливо зрозуміти, що таке паттерн проектування. Паттерн проектування - це загальний шаблон або методологія, яка використовується для розв'язання типових проблем у проектуванні програмного забезпечення. Це своєрідний набір правил та підходів, які дозволяють створювати програмні продукти з високою якістю, легко зрозумілими, модифікованими та розширюваними.

Тепер щодо вибору архітектури для проекту. У даному випадку важливо було обрати таку архітектуру, яка б найбільше відповідала потребам та характеристикам самого проекту. Обрана архітектура MVC (Model-View-Controller) здавалася найбільш відповідною, оскільки вона забезпечує чітке розділення логіки програми на три компоненти: модель, представлення та контролер. [7]

Модель в MVC відповідає за обробку даних і логіку бізнес-логіки. Це дозволяє зберігати логіку додатку відокремлено від інтерфейсу користувача, що полегшує розуміння та редагування коду. Представлення відповідає за відображення даних користувачу. Це означає, що можна легко змінювати і вдосконалювати інтерфейс без впливу на логіку додатку. Контролер же

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

відповідає за прийняття вхідних запитів від користувача і вибір відповідної логіки для виконання.

Обрана архітектура MVC також сприяє відокремленню різних компонентів додатку, що полегшує тестування та розширення функціональності. Кожен компонент може бути тестований окремо, що дозволяє швидко виявляти та виправляти помилки.

Таким чином, вибір архітектури MVC був здійснений з метою забезпечення чистоти, ефективності та легкості управління кодом у даному проекті.

Архітектура MVC є зручним вибором для розробки симулятора настільних ігор з кількох причин.

По-перше, симулятор настільних ігор часто має складну бізнес-логіку та великий обсяг даних, які потрібно обробляти. Модель у паттерні MVC відповідає за цю бізнес-логіку та управління даними. Вона може включати правила гри, логіку розрахунку результатів ходів та інші аспекти, що стосуються гри. Розділення цієї логіки у модель дозволяє забезпечити її чистоту та ефективність, а також полегшує тестування цих аспектів окремо від інших компонентів.

По-друге, представлення в MVC може бути використане для відображення графічного інтерфейсу користувача. У симуляторі настільних ігор графічний інтерфейс може включати дошки, карти, фігури, лічильники, та інші елементи, які відображають стан гри. При використанні MVC, ці елементи можуть бути чітко відокремлені від логіки гри, що полегшує їх розробку та модифікацію.

Загалом, використання паттерну MVC дозволяє ефективно організувати код симулятора настільних ігор, забезпечуючи його чистоту, модульність та легкість управління. Він дозволяє розділити логіку бізнес-процесів, графічний інтерфейс та керування взаємодією з користувачем, що полегшує розробку, тестування та підтримку додатку.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

1.4 Обґрунтування вибору засобів реалізації

Unity — це потужна платформа для розробки ігор, яка дозволяє створювати інтерактивні 3D та 2D-ігри та додатки. [8] Вона підтримує широкий спектр платформ, включаючи Windows, Mac, Android, iOS, та інші, що робить її універсальним інструментом для розробників. Unity забезпечує багатий набір інструментів для створення складних ігор, включаючи вбудований фізичний рушій, засоби для роботи з графікою та анімацією, а також інтегроване середовище розробки (IDE). Для нашого додатку Unity є основною платформою, на якій базується вся візуальна і логічна частина гри. Це дозволяє реалізувати складну графіку, динамічні анімації та інтуїтивний інтерфейс користувача, що робить процес створення ігор зручним та ефективним.

Головні переваги Unity:

Можливості Unity для розробки ігор: Unity має великий набір інструментів та функціональності, спеціально призначених для розробки ігор. Це включає в себе різноманітні фізичні ефекти, анімацію, обробку графіки, управління аудіо та багато іншого, що може бути корисним при створенні реалістичних симуляцій настільних ігор.

Спільнота та підтримка: Unity має велику активну спільноту розробників, яка надає доступ до безлічі ресурсів, таких як документація, форуми, підручники та інші. Це дозволяє швидко знайти відповіді на питання та вирішити проблеми, що можуть виникнути під час розробки проекту.

Photon — це високопродуктивний сервер для розробки мультиплеєрних ігор і додатків в реальному часі. [9] Він забезпечує всі необхідні інструменти для створення мережових ігор, включаючи функціонал для синхронізації станів гравців, управління підключеннями та обміном повідомленнями. Photon відомий своєю високою швидкістю і низькою затримкою, що є критично важливим для онлайн-ігор, де реальний час має велике значення. У нашому додатку Photon використовується для

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

забезпечення онлайн-ігрового процесу між гравцями. Це дозволяє гравцям підключатись до однієї гри, обмінюватись ходами та взаємодіяти в режимі реального часу, що забезпечує плавний і безперервний ігровий досвід.

SQL Server — це реляційна система управління базами даних (СУБД), розроблена компанією Microsoft. [10] Вона використовується для зберігання, управління та обробки даних, забезпечуючи високу продуктивність, надійність та масштабованість. SQL Server підтримує широкий спектр функцій, включаючи складні запити, транзакції, реплікацію даних та аналітику, що робить її ідеальним вибором для багатьох корпоративних застосувань. У нашому проекті SQL Server використовується для зберігання даних про користувачів, ігри, результати та іншу важливу інформацію. Це дозволяє нам ефективно управляти даними, забезпечувати їх цілісність та швидкий доступ до них, що є важливим для забезпечення стабільної роботи додатку.

Висновки до першого розділу

В першому підрозділі було розглянуто особливості сервісів, визначено мету та завдання на дипломну роботу. Також було вирішено в якому середовищі програмного забезпечення буде розроблятися сайт та за допомогою яких засобів.

В другому підрозділі було розглянуто та проаналізовано існуючі сервіси-симулятори настільних ігор. А саме: симулятор «Tabletop Simulator», симулятор «Tabletopia», та сайт «BoardGameArena». Після чого було визначено особливості та функції власного додатку.

У третьому підрозділі було визначено архітектуру конструктора та симулятора настільних ігор та обрано паттерн проектування MVC.

У четвертому підрозділі було сформоване технічне завдання додатку, яким вимогам має відповідати функціонал та інтерфейс, а також розглянуто мінімальні системні вимоги для справної роботи сайту.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

РОЗДІЛ 2 ПРОЄКТУВАННЯ КОНСТРУКТОРА НАСТІЛЬНИХ ІГОР

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Розроблений конструктор та симулятор настільних ігор має інтерфейс, завдяки якому користувачі можуть переглядати список всіх ігор на сервері, шукати та встановлювати певні ігри та конструювати власні проекти. При перегляді ігор користувачі можуть отримати їх назву, опис, правила гри, користувача, який сконструював цю гру. А при перегляді проектів вони можуть змінювати алгоритм роботи цих проектів та публікувати, перетворюючи проекти на повноцінні ігри, в які можуть грати інші користувачі. Для розуміння взаємодії між елементами додатку було створено Діаграму варіантів використання (рис. 2.1).

Діаграма варіантів використання (Use Case Diagram) – це графічне зображення взаємодії користувачів із системою, яке показує всі можливі сценарії використання. Вона є компонентом мови моделювання UML. Ця діаграма ілюструє, як різні типи користувачів (актори) взаємодіють із системою через різні функції (варіанти використання). Вона допомагає зрозуміти вимоги до системи, ідентифікувати її функціональні можливості, а також служить основою для розробки більш детальних діаграм та модулів системи.

UML (Unified Modeling Language) – це стандартизована мова моделювання, яка використовується для візуалізації, специфікації, конструювання та документування компонентів програмного забезпечення. UML забезпечує набір графічних позначень для створення діаграм, які відображають різні аспекти програмних систем, включаючи структуру, поведінку та взаємодію між компонентами. Вона широко використовується в об'єктно-орієнтованому програмуванні для розробки та аналізу системи на різних етапах її життєвого циклу. [11]

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

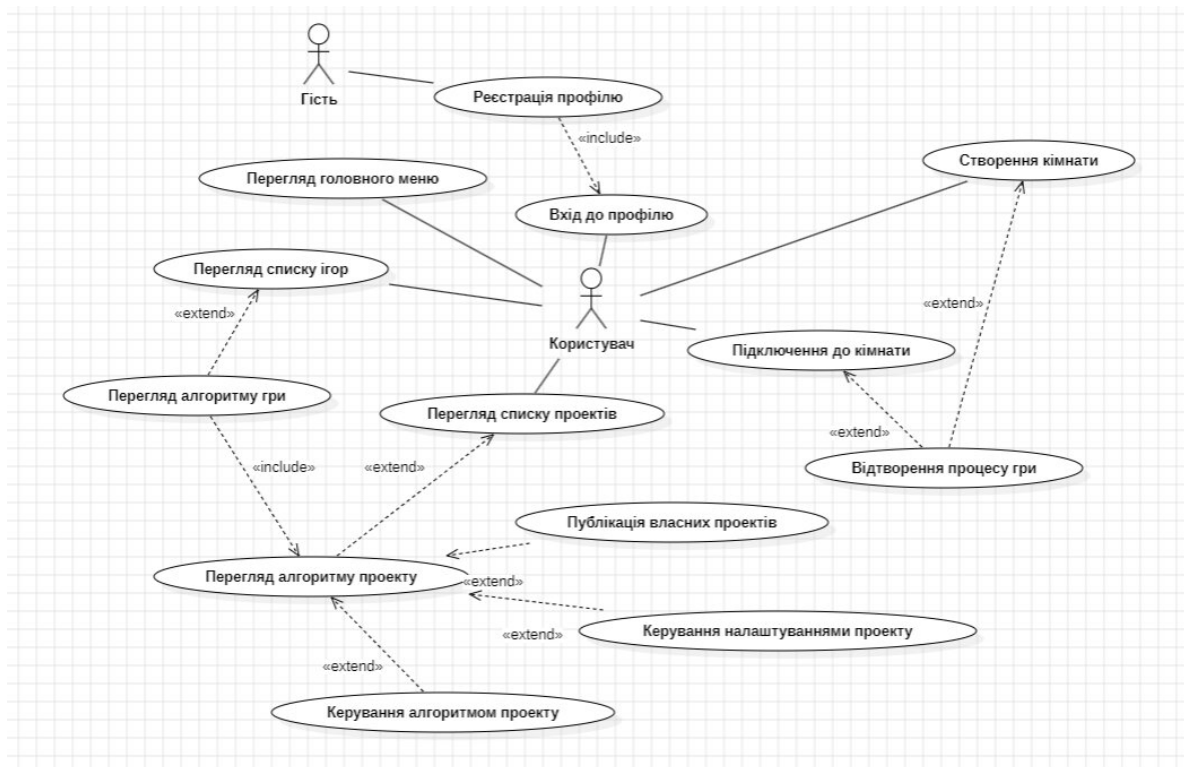


Рис. 2.1. Діаграма варіантів використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.7.

Таблиця 2.1

Опис прецеденту «Вхід/Реєстрація»

Назва варіанту використання	Вхід/Реєстрація
Основні актори	Користувач, гість
Короткий опис	Гості мають зареєструватися, щоб стати користувачем та отримати доступ до більшості функцій додатку. Гості можуть вийти в існуючий обліковий запис.

Таблиця 2.2

Опис прецеденту «Створення кімнати»

Назва варіанту використання	Створення кімнати
Основні актори	Користувач
Короткий опис	Користувачі можуть створювати кімнати з існуючою в базі грою або власним проектом, щоб інші користувачі могли приєднатися.

Таблиця 2.3

Опис прецеденту «Підключення до кімнати»

Назва варіанту використання	Підключення до кімнати
Основні актори	Користувач
Короткий опис	Користувачі можуть підключатися до кімнат інших користувачів за згенерованим кодом, який отримують від власника кімнати.

Таблиця 2.4

Опис прецеденту «Відтворення процесу гри»

Назва варіанту використання	Відтворення процесу гри
Основні актори	Користувач
Короткий опис	Після підключення або створення кімнати користувачі можуть перейти до процесу гри, яка буде відбуватися за сконструйованим алгоритмом.

Таблиця 2.5

Опис прецеденту «Перегляд алгоритму проекту»

Назва варіанту використання	Перегляд алгоритму проекту
Основні актори	Користувач
Короткий опис	Користувачі можуть переглядати алгоритми власних проектів, які були створені з нуля або сформовані на основі опублікованої іншим користувачем гри.

Таблиця 2.6

Опис прецеденту «Керування налаштуваннями проекту»

Назва варіанту використання	Керування налаштуваннями проекту
Основні актори	Користувач
Короткий опис	Користувачі можуть обирати назву для власного проекту, яка відображатися після публікації гри. Також користувачі можуть встановити опис, правила та зображення для майбутньої гри.

Опис прецеденту «Перегляд списку ігор»

Назва варіанту використання	Перегляд списку ігор
Основні актори	Користувач
Короткий опис	Користувачі можуть переглядати всі ігри, які були опубліковані всіма користувачами.

Основна ціль розробки додатку для створення та симуляції настільних ігор полягає у створенні зручного та простого інструменту для користувачів, який дозволяє легко розробляти, тестувати, редагувати та обмінюватись власними іграми. Додаток надає доступ до різноманітних інструментів для створення ігор, а також можливість пошуку ігор інших користувачів. Користувачі можуть створювати свої унікальні ігри, ділитися ними з іншими та використовувати алгоритм роботи ігор інших користувачів. Персоналізований досвід досягається через налаштування профілів, де зберігаються створені ігри та власні проекти.

Основною метою є спрощення процесу розробки та гри в настільні ігри, забезпечуючи користувачів інтуїтивно зрозумілим інструментом для втілення їхніх творчих задумів та насолоди новими ігровими враженнями [12].

Вимоги користувачів:**Внутрішній користувач – гість:**

1. Можливість зареєструватися або авторизуватися у раніше створений обліковий запис.

Внутрішній користувач – авторизований користувач:

1. Можливість вийти з облікового запису.
2. Можливість створення кімнати.
3. Можливість підключення до створеної кімнати.
4. Можливість конструювання власних проектів.
5. Можливість перегляду списку всіх ігор всіх користувачів.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

6. Можливість відтворення процесу гри опублікованих ігор або власних проектів.
7. Можливість перегляду алгоритму ігор інших користувачів.
8. Можливість публікації власних проектів.

Функціональні вимоги:

1. В системі повинна бути можливість реєстрації облікового запису та авторизації.
2. Збереження створених проектів та опублікованих ігор всіх користувачів в базі даних.
3. Отримання інформації про ігри інших користувачів.
4. Швидкодія алгоритмів, сконструйованих за допомогою вбудованих інструментів.
5. З'єднання користувачів одне з одним за допомогою синхронізованих кімнат.

Нефункціональні вимоги:

1. Сприйняття:
 - час, потрібний для освоєння функціоналу процесу гри – від 2-х до 5-ти хвилин, для освоєння функціоналу конструювання ігор – від 10-ти хвилин до 2-х годин;
 - час відповіді додатку на дії користувача не повинні перевищувати 3 секунд, а для виконання складних сконструйованих алгоритмів – 20-ти секунд;
 - інтерфейс додатку повинен бути інтуїтивно зрозумілим для будь-яких категорій користувачів.
2. Надійність:
 - сервери повинні бути доступні цілодобово.
3. Продуктивність:
 - система повинна підтримувати мінімум 100 користувачів, які взаємодіють з базою даних через головне меню та 10 користувачів, які конструюють алгоритми проектів.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

Системні вимоги:

1. Вимоги до середовища виконання:

- мінімум 4 Гб оперативної пам'яті;
- процесор з тактовою частотою не менше 2.4 ГГц;
- наявність DirectX 10 та .NET Framework [13];
- наявність доступу до мережі Інтернет.

2. Вимоги до серверу інформаційної системи:

- наявність реляційної бази даних для забезпечення правильних зв'язків між таблицями та зберігання даних відповідно до їхньої структури.

Завдяки аналізу функціональних вимог були виділені основні сутності, що відображають загальну логіку роботи програмного продукту. Це дозволяє зрозуміти зв'язки як між основними компонентами клієнтської частини, так і між таблицями бази даних, які будуть розроблені надалі. Тепер розглянемо опис діаграми класів, їх полів та методів.

Клас **TableObject** відповідає за весь загальний функціонал можливих об'єктів на столі, таких як карти, колоди та фігури, їх переміщення та відображення.

Клас **Container** відповідає за загальний функціонал об'єктів, які зберігають інші об'єкти, такі як: рука, колода.

Клас **Deck** відповідає за функціонал колод, зберігає методи тасування карт, розміщення карт та їх переміщення.

Клас **Hand** відповідає за функціонал рук гравців, в кожній з яких зберігається ідентифікатор власника, множина карт та визначається їх розміщення.

Клас **Card** відповідає за всі карти на столі. Зберігає їх параметри та методи переміщення та відображення.

Клас **PuzzleContainer** відповідає глобальні параметри та функції окремої гри, керує пазлами.

Клас **Puzzle** відповідає за пазли у кожній грі. Ці пазли визначають алгоритм процесу гри та послідовність виконуваних дій під час гри.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

На рисунку 2.2. наведена частина охарактеризованої діаграми класів.

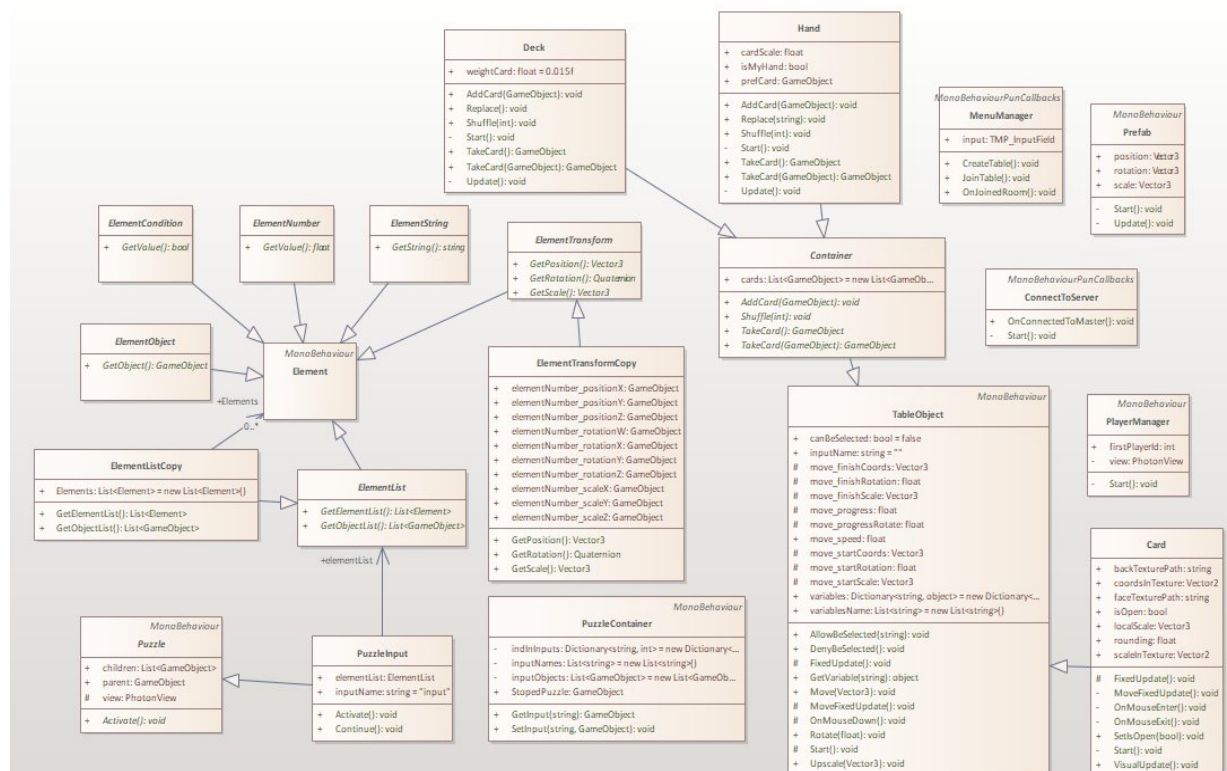


Рис. 2.2. Частина діаграми класів

2.2. Розробка бази даних системи

SQL Server є високопродуктивною СУБД, що відзначається надійністю і ефективністю у роботі з великими обсягами даних. [14] Розроблений компанією Microsoft, він пропонує розширені можливості для забезпечення стійкої роботи баз даних у критичних умовах. SQL Server використовується в

широкому спектрі застосувань від веб-додатків до корпоративних систем, завдяки своїм високим показникам надійності і безпеки. [15]

Щодо підтримки стандарту SQL, SQL Server імплементує стандартні SQL-операції, що спрощує розробку та підтримку програм, що працюють з базою даних. Це дозволяє розробникам легко використовувати знання SQL для створення запитів, оновлення та видалення даних. SQL Server також підтримує транзакційні операції, що забезпечує консистентність даних під час їх обробки, що є важливим для додатків з високими вимогами до цілісності і надійності даних.

База даних для додатку містить 8 таблиць:

1. users – таблиця для збереження даних користувачів.
2. games – таблиця для збереження параметрів ігор та проектів.
3. puzzles – таблиця для збереження реалізованих пазлів у всіх іграх та проектах.
4. puzzles_types – таблиця для збереження типів пазлів.
5. elements – таблиця для збереження реалізованих елементів у всіх іграх та проектах.
6. elements_types – таблиця для збереження типів елементів.
7. elements_data_types – таблиця для збереження типів даних, якими можуть представлятися певні типи елементів.
8. rooms – таблиця для збереження всіх активних кімнат.

Опис структури таблиць бази даних додатку наведено у таблицях 2.8 – 2.15.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

Структура таблиці «users»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор користувача
username	VARCHAR(255)	-	-	Логін
hashed_password	VARCHAR(255)	-	-	Хешований пароль

Таблиця 2.9

Структура таблиці «games»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор гри
name	VARCHAR(255)	-	-	Назва гри
first_puzzle_id	INT	-	+	Початковий пазл алгоритму гри
creator_id	INT	-	+	Творець гри
description	TEXT	-	-	Опис гри
rules_link	VARCHAR(255)	-	-	Посилання на правила гри

Таблиця 2.10

Структура таблиці «puzzles»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор реалізованого пазлу
type_id	INT	-	+	Тип реалізованого пазлу
first_child_puzzle_id	INT	-	+	Дочірній пазл
second_child_puzzle_id	INT	-	+	Опціональний дочірній пазл
element_1_id	INT	-	+	Перший реалізований елемент
element_2_id	INT	-	+	Другий реалізований елемент
element_3_id	INT	-	+	Третій реалізований елемент
element_4_id	INT	-	+	Четвертий реалізований елемент
element_5_id	INT	-	+	П'ятий реалізований елемент
element_6_id	INT	-	+	Шостий реалізований елемент
element_7_id	INT	-	+	Сьомий реалізований елемент
element_8_id	INT	-	+	Восьмий реалізований елемент

Таблиця 2.11

Структура таблиці «puzzles_types»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор типу пазлів
name	VARCHAR(255)	-	-	Назва типу пазлів
child_count	INT	-	-	Кількість дочірніх пазлів
element_data_type_1_id	INT	-	+	Тип першого елемента
element_data_type_2_id	INT	-	+	Тип другого елемента
element_data_type_3_id	INT	-	+	Тип третього елемента
element_data_type_4_id	INT	-	+	Тип четвертого елемента
element_data_type_5_id	INT	-	+	Тип п'ятого елемента
element_data_type_6_id	INT	-	+	Тип шостого елемента
element_data_type_7_id	INT	-	+	Тип сьомого елемента
element_data_type_8_id	INT	-	+	Тип восьмого елемента

Таблиця 2.12

Структура таблиці «elements»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор елемента
type_id	INT	-	+	Тип реалізованого елемента
child_element_1_id	INT	-	+	Перший дочірній елемент
child_element_2_id	INT	-	+	Другий дочірній елемент
child_element_3_id	INT	-	+	Третій дочірній елемент
child_element_4_id	INT	-	+	Четвертий дочірній елемент
child_element_5_id	INT	-	+	П'ятий дочірній елемент
child_element_6_id	INT	-	+	Шостий дочірній елемент
child_element_7_id	INT	-	+	Сьомий дочірній елемент
child_element_8_id	INT	-	+	Восьмий дочірній елемент

Таблиця 2.13

Структура таблиці «elements_types»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор типу елемента
data_type_id	INT	-	+	Тип даних типу елемента

Продовження таблиці 2.13

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

name	VARCHAR(255)	-	-	Назва типу елемента
child_element_1_id	INT	-	+	Перший дочірній елемент
child_element_2_id	INT	-	+	Другий дочірній елемент
child_element_3_id	INT	-	+	Третій дочірній елемент
child_element_4_id	INT	-	+	Четвертий дочірній елемент
child_element_5_id	INT	-	+	П'ятий дочірній елемент
child_element_6_id	INT	-	+	Шостий дочірній елемент
child_element_7_id	INT	-	+	Сьомий дочірній елемент
child_element_8_id	INT	-	+	Восьмий дочірній елемент

Таблиця 2.14

Структура таблиці «elements_data_types»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор типу даних
name	VARCHAR(255)	-	-	Назва типу даних типу елемента

Таблиця 2.15

Структура таблиці «rooms»

Назва	Тип даних	ПК	ЗК	Опис поля
id	INT	+	-	Ідентифікатор кімнати
first_player_id	INT	-	+	Перший гравець в кімнаті
second_player_id	INT	-	+	Другий гравець в кімнаті
game_id	INT	-	+	Гра кімнати
code	VARCHAR(10)	-	-	Код підключення до кімнати

На рисунку 2.3. наведено схему бази даних. Дана візуалізація демонструє, яким чином виглядають таблиці та зв'язки між ними.

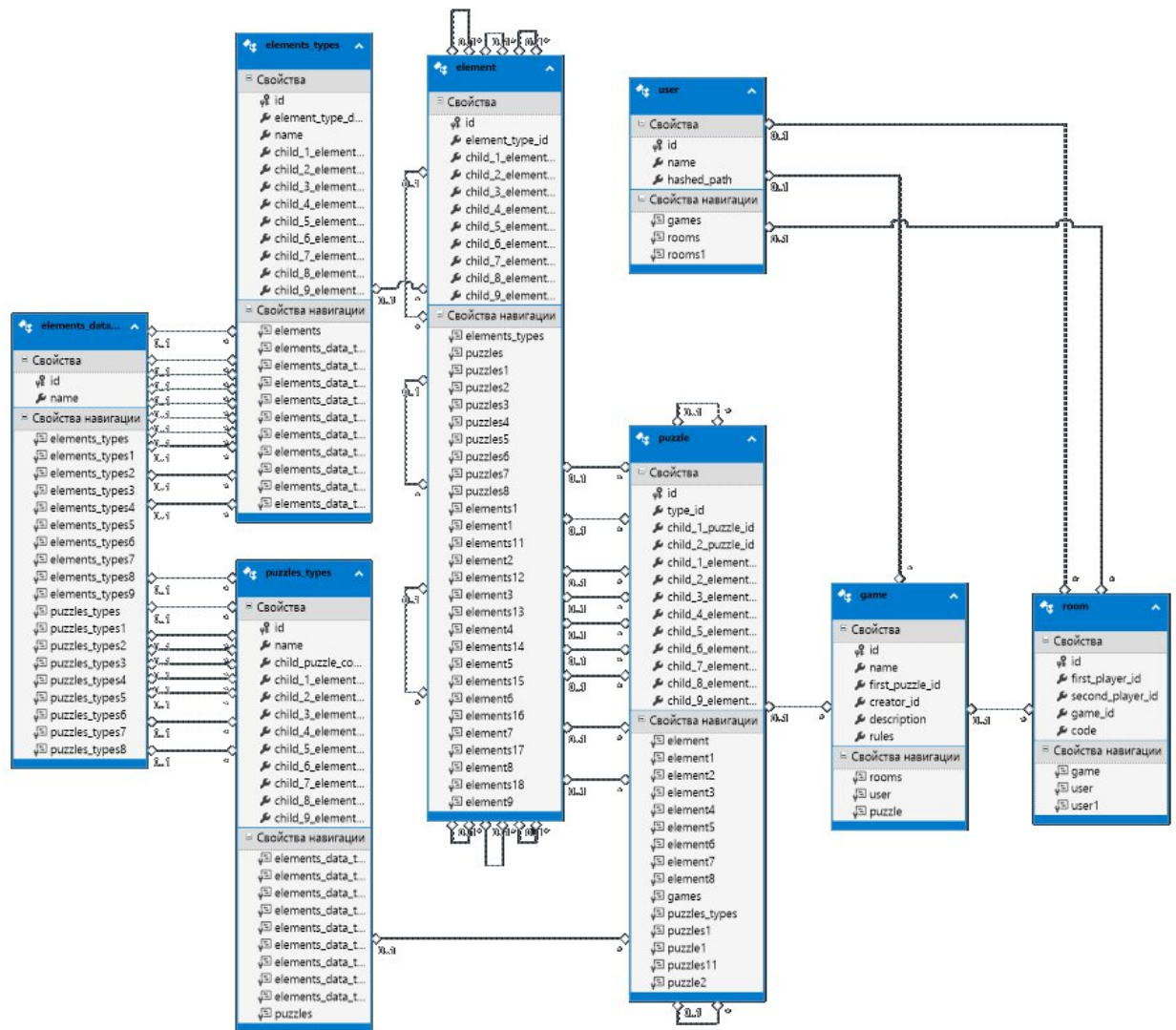


Рис. 2.3. Схема бази даних конструктора та симулятора настільних ігор

2.3. Проектування та реалізація алгоритмів роботи додатку

Алгоритм підключення до кімнати відбувається за допомогою коду, який відображається в створеній користувачем кімнаті. Коли цей користувач ділиться цим кодом з іншим користувачем, останній може приєднатися до кімнати ввівши код в спеціальне поле. Коли обидва користувача підключаються до кімнати, у створившого кімнату з'являється можливість почати гру на двох [16].

Діаграма активності для користувачів представлена на рисунку 2.4.

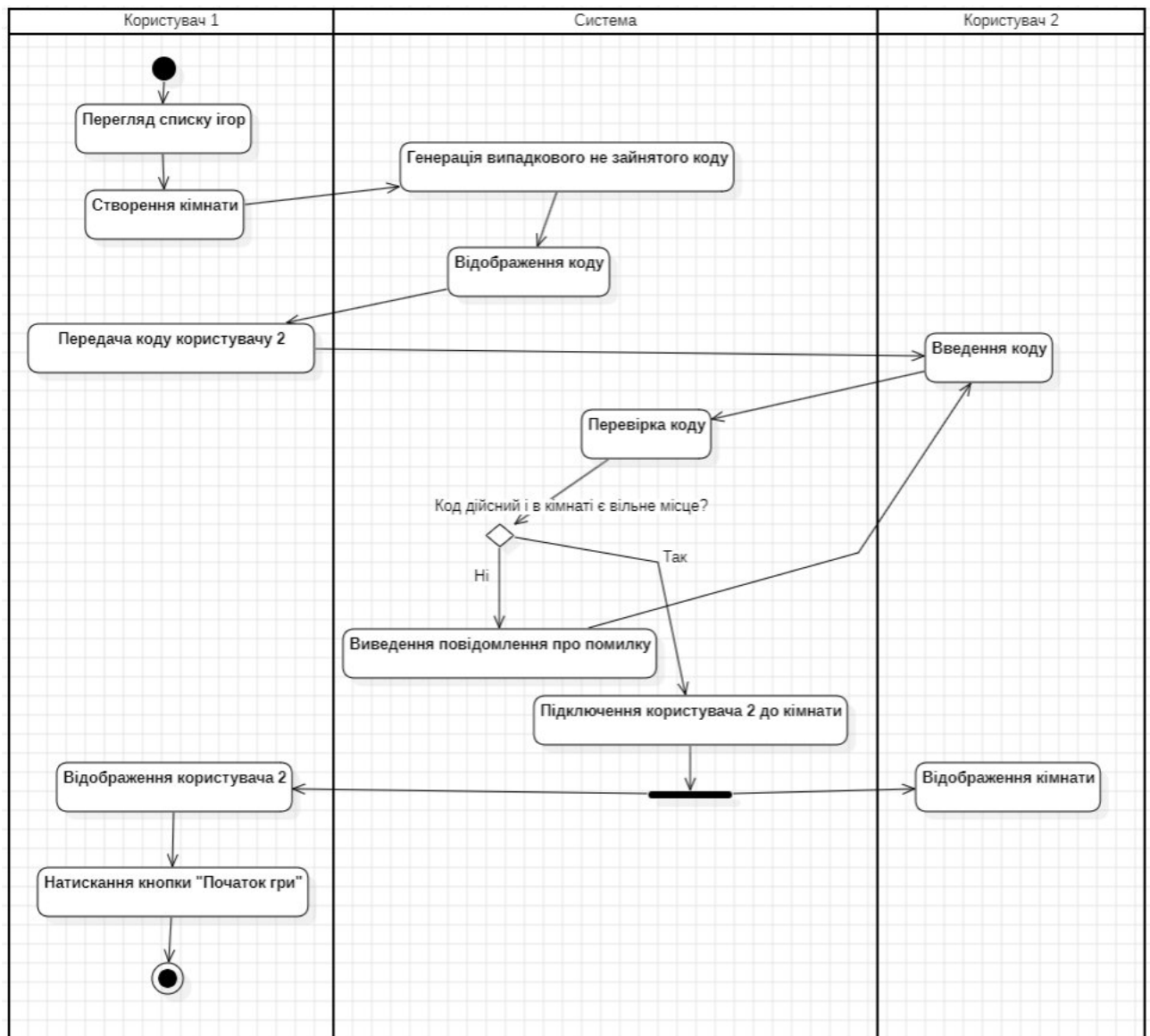


Рис. 2.4. Діаграма активності та відповідність станів на дій користувачів

Опис діаграми активності (Користувач під час гри):

1. Перегляд списку ігор – користувач може переглядати список всіх опублікованих ігор на спеціальній сторінці.
2. Створення кімнати – після натискання користувачем на гру у списку, створюється кімната.
3. Генерація випадкового не зайнятого коду – система аналізує всі кімнати та генерує випадковий не зайнятий код.
4. Відображення коду – користувачу в кімнаті відображається згенерований код.

5. Передача коду користувачу 2 – користувач в кімнаті може передати код цієї кімнати іншому користувачу наживо чи за допомогою будь-якого сервіса для спілкування.
6. Введення коду – користувач може перейти на сторінку підключення до існуючої кімнати та ввести отриманий код.
7. Перевірка коду – система визначає, чи існує в базі кімната з вказаним кодом та чи не зайнята вона іншими користувачами.
8. Виведення повідомлення про помилку – якщо користувач ввів невідний код, він побачить повідомлення про це.
9. Підключення користувача до кімнати – система звертається до серверів Photon та під'єднує користувача до кімнати.
10. Натискання кнопки “Початок гри” – користувач, що обрав гру та створив кімнату може натиснути кнопку “Початок гри” та обидва користувача перейдуть до гри.

Одним з основних алгоритмів додатку є відтворення користувацького алгоритму. Під час конструювання гри користувач складає її алгоритм з вбудованих пазлів, кожен з яких робить конкретну дію з об'єктами на столі. Коли деякі користувачі приєднуються до однієї кімнати та починають гру, спрацьовує початковий пазл алгоритму цієї гри “PuzzleStart”. При цьому у кожного пазлу є принаймні один дочірній пазл, який спрацьовує одразу ж після виконання минулого. Таким чином гра йде допоки алгоритм гри не приведе до останнього пазлу, який визначить переможця, яким може стати або один з гравців, або гра (у разі, якщо гра є кооперативною). Особливим пазлом у системі додатку є пазл “PuzzleInput”, для виконання якого користувач повинен натиснути на один з об'єктів на столі. Коли користувач це робить, програма зберігає посилання на активний об'єкт. Саме за допомогою цього пазлу гра взаємодіє з гравцями.

Діаграма активності для користувача під час процесу гри представлена на рисунку 2.5.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

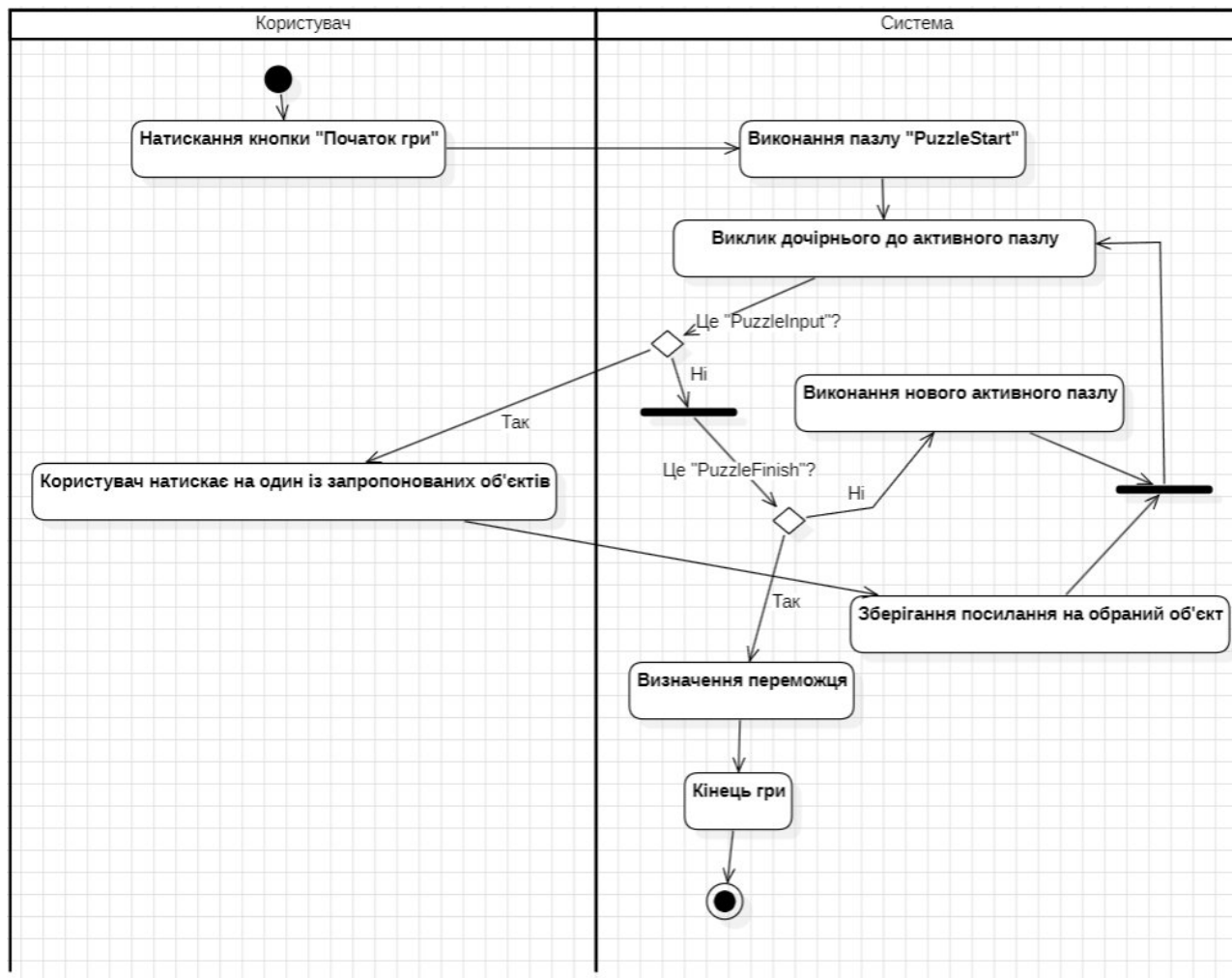


Рис. 2.5. Діаграма активності та відповідність станів на дій гравців

Опис діаграми активності (Користувач під час гри):

1. Натискання кнопки “Початок гри” – користувач, який створив кімнату та обрав гру може натиснути кнопку початку гри.
2. Виконання пазлу “PuzzleStart” – система активує початковий пазл, який позначає початок гри та додає основні об’єкти, такі як: камера, освітлення, стіл та руки гравців.
3. Виклик дочірнього до активного пазлу – дочірній пазл останнього активного пазлу стає активним.
4. Користувач натискає на один із запропонованих об’єктів – в залежності від налаштувань пазлу “PuzzleInput” система обирає певний список об’єктів, на які може натиснути користувач.

5. Зберігання посилання на обраний об'єкт – записується посилання на об'єкт, який обрав користувач.
6. Виконання нового активного пазлу – новий активний пазл виконує певну дію в залежності від свого типу та вхідних даних
7. Визначення переможця – коли пазл “PuzzleFinish” стає активним, гра закінчується і в залежності від параметрів цього пазлу, визначається переможець.

2.4. Реалізація функціоналу конструктора настільних ігор

Перейдемо до реалізації функціоналу додатку. Врахуємо те, що базу даних вже було створено заздалегідь у SQL Server Manager Studio 20. Для розробки було використане середовище Visual Studio 2022.

Для реалізації мультиплеєру було обрано мережеву технологію Photon Engine. Коли користувач запускає додаток, йому відкривається сторінка завантаження, в якій деякий час (зазвичай не більше секунди) відбувається підключення користувача до серверів Photon Engine.

Лістинг коду класу “ConnectToServer”:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using UnityEngine.SceneManagement;

public class ConnectToServer : MonoBehaviourPunCallbacks
{
    void Start()
    {
        // Підключення до сервера Photon
        PhotonNetwork.ConnectUsingSettings();
    }

    public override void OnConnectedToMaster()
    {
        // Завантаження сцени "Menu"
        SceneManager.LoadScene("Menu");
    }
}
```

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

Основним функціоналом в додатку є виконання алгоритмів ігор, побудованих користувачами. Для цього використовуються пазли різних типів. Одним з головних типів пазлів є “PuzzleStart”, який запускає алгоритм гри та створює в сцені основні об’єкти.

Спочатку в сцені гри знаходиться об’єкт з назвою гри, в якому знаходиться об’єкт PuzzleContainer, який зберігає глобальні дані для алгоритму гри. Наприклад, посилання на обрані користувачами об’єкти. В об’єкті PuzzleContainer завжди зберігається об’єкт PuzzleStart, який відповідає за початок гри та налаштування сцени. Структуру початкової сцени можна побачити на рисунку 2.6.



Рис. 2.6. Об’єкти початкової сцени будь-якої користувацької гри

Після запуску сцени виконується метод “Activate()” класу PuzzleStart, який є частиною однойменного об’єкту. В цього метода є дві головні функції.

Перша функція – завантаження сцени в обох гравців. Для цього використовується метод “RPC”, який виконує інший обраний метод на прикладах обох користувачів. В даному випадку виконується метод “LoadScene”.

В ньому відбувається завантаження стандартних для усіх ігор об’єктів з префабу, таких як: “Camera” (камера), “Directional Light” (світло), “Table” (поверхня стола), “MyHand” (власна рука) та “Opponent Hand” (рука супротивника). При цьому, камера, світло та поверхня стола вкладаються в кореневий об’єкт гри, тому мають глобальні координати, а власна та ворожа руки вкладаються в об’єкт камери задля того, щоб при переміщенні камери, зображення рук на екрані не змінювалось.

Лістинг коду методу “LoadScene” зображено на рисунку:

```

[PunRPC]
private void LoadScene()
{
    // Знаходимо об'єкт з тегом "Game"
    GameObject game = GameObject.FindWithTag("Game");
    if (game == null)
        return;

    // Видаляємо всі дочірні об'єкти, крім тих, що мають тег "PuzzleContainer"
    foreach (Transform child in game.transform)
    {
        if (child.tag != "PuzzleContainer")
            GameObject.Destroy(child.gameObject);
    }

    // Завантажуємо і створюємо нові об'єкти з ресурсів
    GameObject camera =
    Instantiate(Resources.Load<GameObject>("Base/Prefabs/Camera"), Vector3.zero,
    Quaternion.identity);
    GameObject light =
    Instantiate(Resources.Load<GameObject>("Base/Prefabs/Directional Light"),
    Vector3.zero, Quaternion.identity);
    GameObject table =
    Instantiate(Resources.Load<GameObject>("Base/Prefabs/Table"), Vector3.zero,
    Quaternion.identity);
    GameObject myHand =
    Instantiate(Resources.Load<GameObject>("Base/Prefabs/MyHand"), Vector3.zero,
    Quaternion.identity);
    GameObject opponentHand =
    Instantiate(Resources.Load<GameObject>("Base/Prefabs/OpponentHand"), Vector3.zero,
    Quaternion.identity);

    // Призначаємо батьківські об'єкти
    camera.transform.parent = game.transform;
    light.transform.parent = game.transform;
    table.transform.parent = game.transform;
    myHand.transform.parent = camera.transform;
    opponentHand.transform.parent = camera.transform;

    // Змінюємо обертання камери для другого гравця
    if (PhotonNetwork.LocalPlayer.ActorNumber !=
    game.GetComponent<PlayerManager>().firstPlayerId)
        camera.GetComponent<Prefab>().rotation.z += 180f;

    // Призначаємо матеріал і текстуру для столу
    table.GetComponent<MeshRenderer>().material =
    Resources.Load<Material>("Base/Materials/Table");

```

```

        table.GetComponent<MeshRenderer>().material.mainTexture =
Resources.Load<Texture>("Base/Textures/Table");
    }

```

Друга функція методу “Activate()” – запуск алгоритму гри. Для цього викликається той самий метод у дочірнього пазлу. Взагалі, цей метод в кожному пазлі визначає процеси, які відбуваються після його активації.

Лістинг коду цього методу для пазлу “PuzzleStart”:

```

public override void Activate()
{
    view = GetComponent<PhotonView>(); // Отримуємо PhotonView

    GameObject game = GameObject.FindWithTag("Game"); // Знаходимо об'єкт "Game"

    view.RPC("LoadScene", RpcTarget.AllBuffered); // Викликаємо RPC метод
LoadScene

    if (children.Count > 0 && children[0] != null) // Активуємо дочірній об'єкт
        children[0].GetComponent<Puzzle>().Activate();
}

```

Тепер розглянемо функціонал інших важливих пазлів, які визначають роботу алгоритму.

“PuzzleInput” – це пазл, який визначає взаємодію гри з користувачами. Головним параметром цього пазлу є елемент, який зберігає множину певних об’єктів на столі. Цей параметр визначає користувач, який конструює цю гру. Після активації пазлу всі обрані об’єкти стають “клікабельними” та при певних налаштуваннях кімнати починають кліпати, демонструючи гравцю можливість натиснути на цей об’єкт. При цьому виконання пазлу тимчасово зупиняється. Але як тільки гравець натискає на якийсь об’єкт з цієї множини, виконання пазлу продовжується, система отримує інформацію про обраний об’єкт та зберігає в глобальному об’єкті “PuzzleContainer”. Після цього “PuzzleInput” активує дочірній пазл, а обраний об’єкт алгоритм зможе використати в подальшій роботі.

Лістинг методу активації пазлу “PuzzleInput”:

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

```

public override void Activate()
{
    List<GameObject> objectList = elementList.GetObjectList();
    foreach (GameObject obj in objectList)
    {
        obj.GetComponent<TableObject>().AllowBeSelected(inputName); //
Дозволяємо об'єктам бути вибраними
        Debug.Log(obj.name);
    }
    GameObject.FindWithTag("PuzzleContainer").GetComponent<PuzzleContainer>().
.StopedPuzzle = this.gameObject; // Зупиняємо пазл
}

```

Лістинг методу продовження пазлу “PuzzleInput”, який викликається після обрання користувачем якогось об’єкту:

```

public void Continue()
{
    List<GameObject> objectList = elementList.GetObjectList();
    foreach (GameObject obj in objectList)
    {
        obj.GetComponent<TableObject>().DenyBeSelected(); // Забороняємо
об'єктам бути вибраними
    }

    Debug.Log(GameObject.FindWithTag("PuzzleContainer").GetComponent<PuzzleCo
ntainer>().GetInput("input").name);

    if (children.Count > 0 && children[0] != null)
        children[0].GetComponent<Puzzle>().Activate(); // Закінчуємо
виконання пазлу та активуємо перший дочірній об'єкт
}

```

“PuzzleCheck” також є одним з найважливіших елементів конструювання алгоритмів настільних ігор конструкторі. Він водночас виконує функцію розгалуження виконання алгоритму та циклу.

Пазл приймає єдиний параметр – елемент типу “ElementCondition”, який зберігає або певним чином визначає який з двох дочірніх пазлів потрібно активувати. Якщо дочірній пазл є новим – виконується розгалуження, а якщо він вже активувався раніше, алгоритм повертається назад, створюючи таким чином цикл.

Лістинг коду методу активації “PuzzleCheck”:

```
public override void Activate()
{
    if (condition.GetComponent<ElementCondition>().GetValue()) // Перевірка
умови
    {
        if (children.Count > 0 && children[0] != null)
            children[0].GetComponent<Puzzle>().Activate(); // Активуємо
перший дочірній об'єкт
        }
        else
        {
            if (children.Count > 1 && children[1] != null)
                children[1].GetComponent<Puzzle>().Activate(); // Активуємо
другий дочірній об'єкт
            }
        }
    }
```

Висновки до другого розділу

У другому розділі були визначені варіанти використання та описані їх прецеденти. Розглянуті вимоги користувачів, а також функціональні, нефункціональні та системні вимоги до додатку. Створено діаграму класів, що відображає загальну логіку роботи програмного продукту. На основі діаграми сутностей та загальних вимог, було спроектовано вісім таблиць для бази даних конструктора настільних ігор і створено схему бази даних.

Крім того, здійснено реалізацію ключового функціоналу додатку з включенням лістингів та опису коду.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З КОНСТРУКТОРОМ ТА СИМУЛЯТОРОМ НАСТІЛЬНИХ ІГОР

3.1. Порядок встановлення та налаштування параметрів платформи

Розроблений конструктор та симулятор настільних ігор складається з ряду компонентів, кожен з яких потребує окремого розгортання та налаштування. [17] Для візуалізації цих компонентів та процесу розгортання була створена діаграма розгортання, яка зображена на рисунку 3.1.

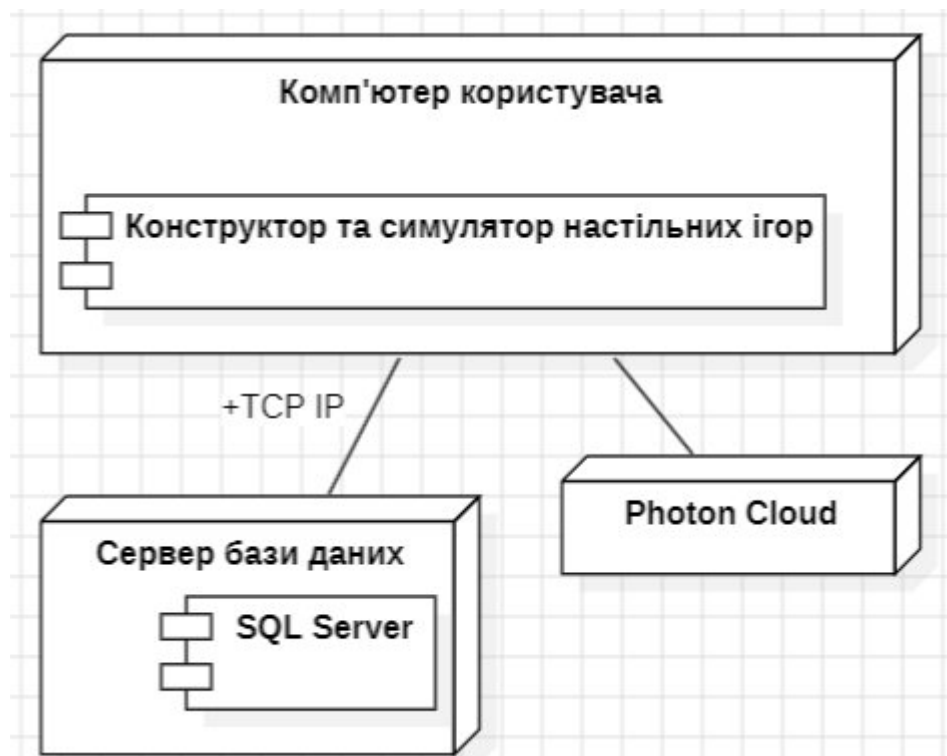


Рис. 3.1. Діаграма розгортання

Діаграма розгортання ілюструє систему, що включає три основні вузли:

1. Клієнтський додаток “Конструктор та симулятор настільних ігор”: Додаток, який встановлюється на комп'ютер користувача і забезпечує взаємодію з ігровими сесіями та іншими користувацькими функціями.
2. Сервер баз даних (SQL Server): Розташований на віддаленому сервері або в хмарі, цей вузол зберігає всі дані, пов'язані з ігровими сесіями, користувачами та іншими важливими аспектами додатку.

3. Photon Cloud: Цей сервіс забезпечує мережеве з'єднання між гравцями під час ігрових сесій. Він реалізується через Photon Cloud сервери, які забезпечують реалізацію мережевої логіки гри та синхронізацію даних між клієнтами.

Для взаємодії між комп'ютером користувача та сервером баз даних використовується протокол TCP/IP. Цей протокол забезпечує надійну та ефективну передачу даних між клієнтським додатком та сервером баз даних. [18]

Для успішного розгортання додатку необхідно налаштувати сервер баз даних SQL Server з встановленою базою даних і правильними параметрами підключення. Photon Cloud рекомендується для обслуговування мережевих запитів і забезпечення оптимальної продуктивності системи.

3.2. Структура інтерфейсу додатку та порядок роботи

Перейдемо до огляду додатку.

Після його запуску відкривається меню завантаження (рис. 3.2).



Рис. 3.2. Зовнішній вигляд меню завантаження

Через деякий час (частіше всього менше ніж за секунду), якщо пристрій користувача підключений до мережі Інтернет, користувачу відкривається вікно авторизації (рис. 3.3), де він може увійти в свій обліковий запис ввівши логін та пароль у спеціальні поля та натиснувши кнопку “Авторизуватися”.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

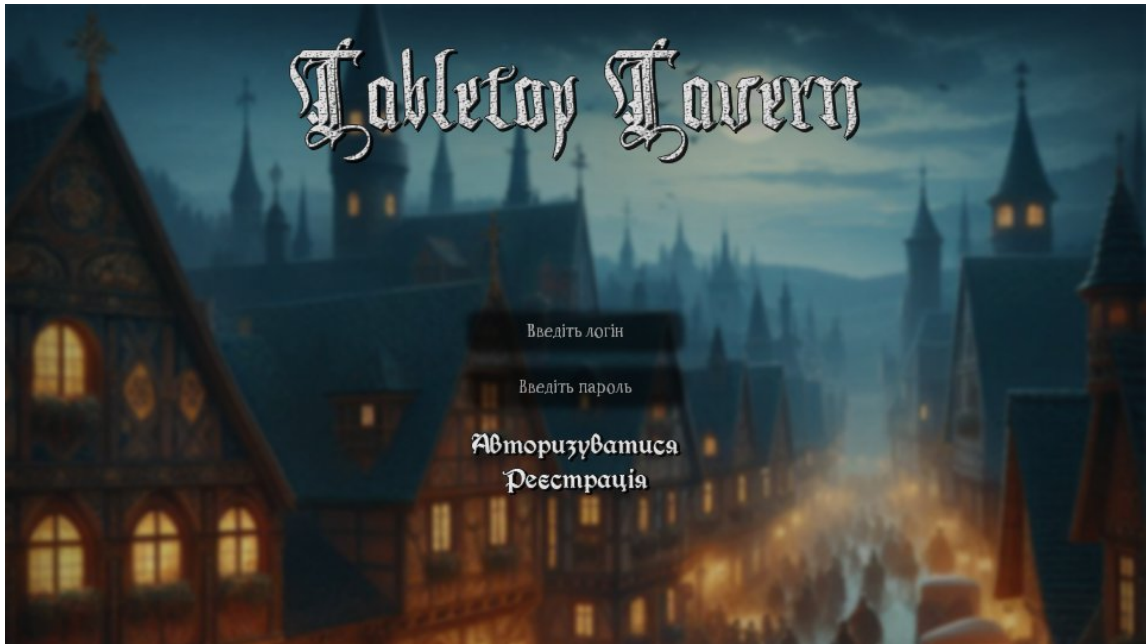


Рис. 3.3. Зовнішній вигляд меню авторизації

Якщо у користувача немає власного облікового запису, він може його створити натиснувши на кнопку “Реєстрація”, після чого відкриється меню реєстрації (рис. 3.4).

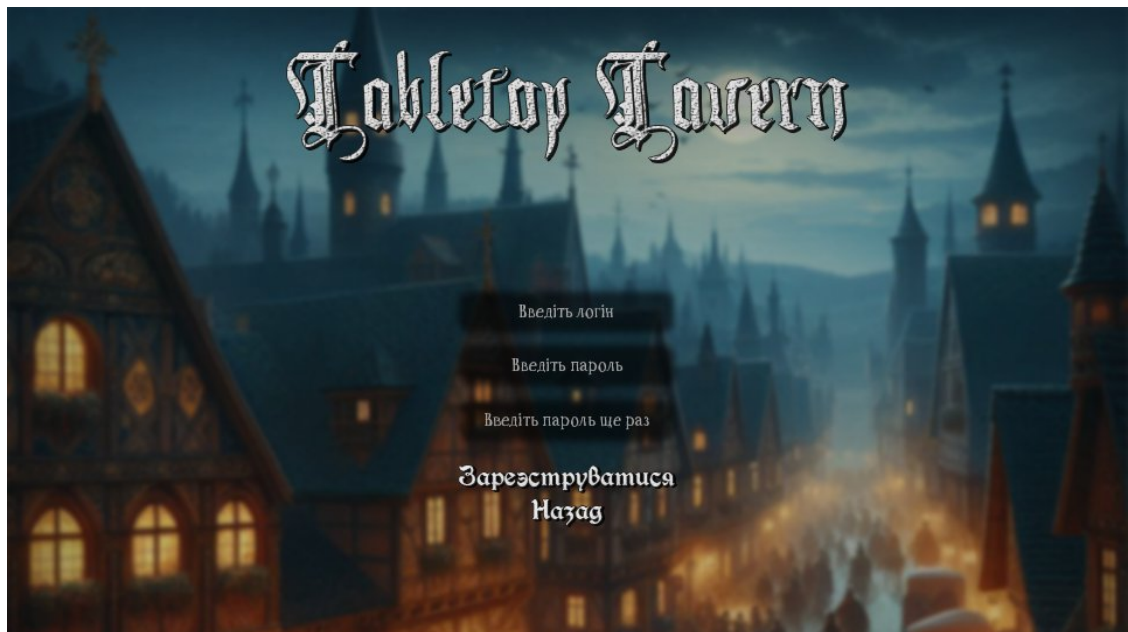


Рис. 3.4. Зовнішній вигляд меню реєстрації

Коли користувач введе необхідні дані в поля та натисне кнопку “Зареєструватися”, буде створений обліковий запис з цими даними, а користувача автоматично авторизує.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Після авторизації відкриється головне меню, яке зображене на рисунку 3.5.



Рис. 3.5. Зовнішній вигляд головного меню

Якщо користувач натисне на кнопку “Всі ігри”, відкриється список всіх опублікованих ігор різних користувачів (рис. 3.6).



Рис. 3.5. Зовнішній вигляд меню всіх ігор

В лівій частині вікна з’явився перелік всіх ігор, який можна гортати за допомогою колеса миші. Таким чином можна обрати іншу гру та переглянути її опис (рис. 3.6).

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				45
Змн.	Арк.	№ докум.	Підпис	Дата		



Рис. 3.6. Приклад обрання іншої гри

Також в цьому меню є можливість пошуку конкретної гри за назвою. Це буде в нагоді, коли кількість ігор в додатку стане великою. Ось як виглядає пошук конкретної гри за назвою (рис. 3.7).



Рис. 3.7. Приклад пошуку конкретної гри

У правому нижньому кутку знаходиться меню, за допомогою якого можна повернутися назад до головного меню, переглянути структуру алгоритму процесу гри та створити кімнату. Якщо натиснути на кнопку

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

“Створити кімнату”, відкриється кімната, до якої може приєднатися інший користувач (рис. 3.8). Для цього потрібно назвати йому код, що зображений у верхній частині вікна.



Рис. 3.8. Інтерфейс створеної кімнати

Користувач, який отримав код, повинен повернутися до головного меню та натиснути кнопку “Приєднатися до кімнати”, після чого відкриється меню (рис. 3.9) з полем для вводу отриманого коду кімнати (рис. 3.10).



Рис. 3.9. Меню підключення до кімнати

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		47



Рис. 3.10. Введення коду кімнати

Після введення коду та натискання кнопки “Приєднатися”, відкривається меню кімнати, в якій вже є перший користувач (рис.3.11).

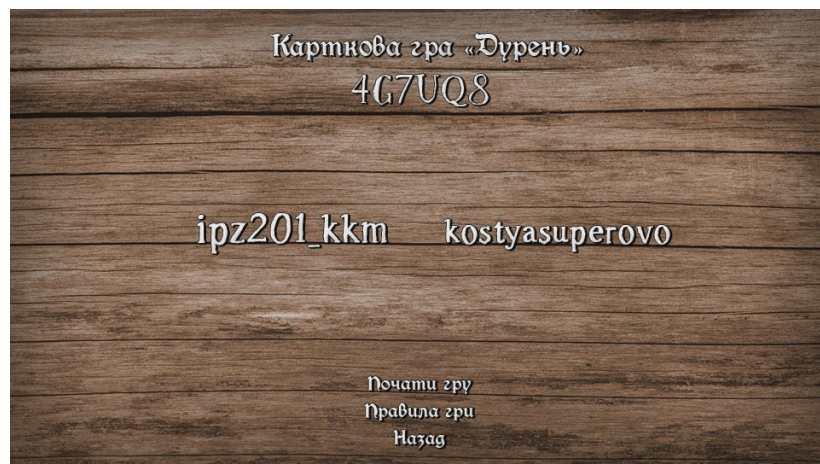


Рис. 3.11. Меню підключення до кімнати

Коли ж користувач, який створив кімнату, натисне кнопку “Почати гру”, обидва користувача перейдуть до екрану гри (рис.3.12).



Рис. 3.12. Процес гри в карткову гру “Дурень”

Повернемося до головного меню. Якщо натиснути кнопку “Побудувати гру” (рис. 3.13), відкриється список усіх проектів користувача. Знову ж таки,

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

проекти гортаються в меню за допомогою колеса миші, а також присутнє поле для пошуку (рис. 3.14).



Рис. 3.13. Кнопка головного меню “Побудувати гру”

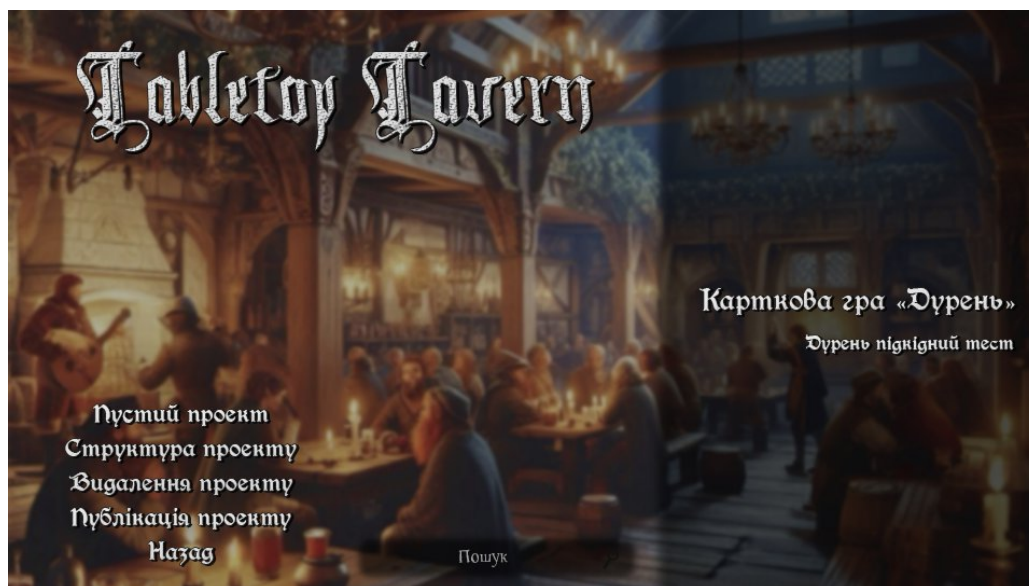


Рис. 3.13. Меню списку проектів

Щоб змінити або переглянути проект потрібно натиснути “Структура проекту”, а щоб створити новий пустий проект – натиснути кнопку “Пустий проект”. В будь-якому випадку відкриється вікно редагування алгоритму гри за допомогою вбудованих пазлів (рис. 3.14).



Рис. 3.14. Меню редагування проектів

3.3 Тестування роботи додатку.

Тестування є ключовим етапом у забезпеченні якості та надійності додатку, допомагає виявляти і усувати помилки, недоліки та потенційні проблеми. Воно також дозволяє переконатися, що додаток відповідає вимогам і очікуванням користувачів.

Тестування програмного забезпечення є критично важливим на всіх етапах розробки і після її завершення, щоб забезпечити правильність роботи всіх функцій. Після виявлення та виправлення помилок необхідно проводити повторне тестування, щоб переконатися у відсутності нових проблем. [19] Існує багато різних технік тестування, які ефективно застосовуються в масштабних додатках. Тестування гарантує високу якість програмного забезпечення і задовольняє вимоги користувачів, перевіряючи правильність виконання окремих функцій, модулів або всієї системи в цілому.

Виявлення великої кількості помилок часто можливо завдяки використанню техніки "передбачення помилок", яка дозволяє розробнику заздалегідь визначити потенційні проблеми на основі свого досвіду. Ця техніка особливо ефективна, коли додаток розробляється однією людиною, яка має глибоке розуміння всіх його аспектів. Для перевірки форм з числовими значеннями ефективною є техніка "аналізу граничних значень", яка передбачає, що помилки найчастіше виникають на межах діапазону вхідних даних. Якщо програма правильно обробляє найменші і найбільші значення, то вона, швидше за все, буде коректно працювати з іншими значеннями в межах цього діапазону.

Впровадження різних методів тестування дозволяє максимально охопити всі можливі сценарії використання додатку і забезпечити його надійність та стабільність. Кожна техніка має свої переваги і специфіку, що дозволяє більш точно та детально виявляти і усувати помилки. Таким чином, тестування є не просто етапом розробки, а безперервним процесом, який сприяє створенню якісного і надійного програмного продукту. [20]

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

1. Запуск додатку та завантаження меню

передумови: користувач запускає додаток на своєму пристрої, підключеному до Інтернету.

кроки:

- користувач запускає додаток;
- додаток відкриває меню завантаження.

очікуваний результат: меню завантаження з'являється і через деякий час відкривається вікно авторизації.

фактичний результат: меню завантаження з'являється і через деякий час відкривається вікно авторизації.

2. Авторизація користувача

передумови: користувач хоче увійти у свій обліковий запис.

кроки:

- користувач переходить на сторінку авторизації;
- користувач вводить логін та пароль;
- натискає кнопку «Авторизуватися».

очікуваний результат: користувач успішно авторизується та відкривається головне меню.

фактичний результат: користувач успішно авторизується та відкривається головне меню.

3. Реєстрація нового користувача

передумови: користувач не має власного облікового запису.

кроки:

- користувач натискає кнопку «Реєстрація»;
- вводить необхідні дані для реєстрації;
- натискає кнопку «Зареєструватися».

очікуваний результат: обліковий запис створюється, користувач автоматично авторизується та відкривається головне меню.

фактичний результат: обліковий запис створюється, користувач автоматично авторизується та відкривається головне меню.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

4. Перегляд всіх ігор

передумови: користувач знаходиться у головному меню.

кроки:

- користувач натискає кнопку «Всі ігри»;
- відкривається список всіх опублікованих ігор;
- користувач наводиться на список ігор та гортає його за допомогою колеса миші.

очікуваний результат: користувач може переглядати всі опубліковані ігри та їх описи.

фактичний результат: користувач може переглядати всі опубліковані ігри та їх описи.

5. Пошук конкретної гри

передумови: користувач знаходиться у вікні списку ігор.

кроки:

- користувач вводить назву гри в поле для пошуку;
- натискає кнопку «Пошук».

очікуваний результат: показується список ігор, що відповідають введеній назві.

фактичний результат: показується список ігор, що відповідають введеній назві.

6. Створення кімнати для гри

передумови: користувач знаходиться у вікні списку ігор.

кроки:

- Користувач натискає кнопку «Створити кімнату».
- Відкривається кімната з кодом у верхній частині вікна.

очікуваний результат: користувач отримує код кімнати, яку може використати для приєднання іншого користувача.

фактичний результат: користувач отримує код кімнати, яку може використати для приєднання іншого користувача.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

Висновки до третього розділу

У третьому розділі було детально описано процес встановлення та налаштування додатку, а також розглянуто його архітектуру за допомогою діаграми розгортання. Окрім цього, проведено аналіз роботи з ігровим процесом та конструктором в додатку, де розглянуті ключові компоненти і їх функціональні можливості.

Також був проведений інтенсивний процес тестування для виявлення потенційних помилок, що сприяє покращенню загальної якості продукту. Результати тестування свідчать про відсутність помилок у функціонуванні додатку.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра, було розроблено готовий до користування конструктор та симулятор настільних ігор.

У процесі розробки конструктора та симулятора настільних ігор було проведено великий обсяг аналізу та розробки, охоплюючи кілька ключових аспектів. Початково було визначено стратегічні цілі проекту та вибрано середовище розробки, що відповідає потребам додатку. Під час дослідження існуючих сервісів-симуляторів настільних ігор, таких як “Tabletop Simulator”, “Tabletopia”, “Scratch” і “BoardGameArena”, були визначені особливості та функції, які мали бути реалізовані в додатку.

Детальний аналіз архітектури системи дозволив обрати патерн проектування, що забезпечив ефективну структуру додатку. Розроблене технічне завдання чітко визначило вимоги до функціоналу і інтерфейсу користувача.

Далі були визначені варіанти використання, описані прецеденти та розглянуті вимоги користувачів, функціональні, нефункціональні та системні вимоги до додатку. На основі діаграм сутностей та вимог було розроблено структуру бази даних та класів додатку.

В результаті було детально розглянуто процес встановлення та налаштування додатку, включаючи діаграму розгортання. Проведено аналіз роботи з ігровим процесом та конструктором, де досліджені ключові компоненти і їх функціональні можливості. Інтенсивне тестування додатку підтвердило його надійність і відсутність помилок, що підкреслює високу якість розробленого програмного забезпечення.

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Examining Board Gameplay and Learning: A Multidisciplinary Review of Recent Research [Електронний ресурс] — Режим доступу до ресурсу: https://www.researchgate.net/publication/340690811_Examining_Board_Gameplay_and_Learning_A_Multidisciplinary_Review_of_Recent_Research
2. Rise of Board Games in Today's Tech-Dominated Culture [Електронний ресурс] — Режим доступу до ресурсу: <https://www.pitt.edu/pittwire/features-articles/rise-board-games-today-s-tech-dominated-culture>
3. Tabletop Simulator [Електронний ресурс] — Режим доступу до ресурсу: <https://www.tabletopsimulator.com/>
4. Tabletopia [Електронний ресурс] — Режим доступу до ресурсу: <https://tabletopia.com/>
5. Board Game Arena [Електронний ресурс] — Режим доступу до ресурсу: <https://uk.boardgamearena.com/>
6. Scratch [Електронний ресурс] — Режим доступу до ресурсу: <https://scratch.mit.edu/>
7. Understanding Model-View-Controller (MVC) Architecture: A Comprehensive Guide [Електронний ресурс] — Режим доступу до ресурсу: https://medium.com/@Sukumar_Sundar/understanding-model-view-controller-mvc-architecture-a-comprehensive-guide-f0be8ebb8d7f
8. Unity Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://docs.unity.com/>
9. Photon Engine [Електронний ресурс] — Режим доступу до ресурсу: <https://www.photonengine.com/>
10. Microsoft SQL Server Documentation [Електронний ресурс] — Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16>
11. Unity Learn [Електронний ресурс] — Режим доступу до ресурсу: <https://learn.unity.com/>

12. Introduction to Game Development with Unity (Udemy Course) [Электронный ресурс] — Режим доступа до ресурсу: <https://www.udemy.com/course/introduction-to-game-development-with-unity/>
13. Microsoft .NET Framework Documentation [Электронный ресурс] — Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/dotnet/framework/>
14. RDBMS (Relational Database Management System) Definition [Электронный ресурс] — Режим доступа до ресурсу: <https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system>
15. What is SQL Server? (Microsoft Documentation) [Электронный ресурс] — Режим доступа до ресурсу: <https://learn.microsoft.com/ru-ru/sql/sql-server/what-is-sql-server?view=sql-server-ver16>
16. Photon Fusion Documentation [Электронный ресурс] — Режим доступа до ресурсу: <https://doc.photonengine.com/fusion/current/fusion-intro>
17. UML Deployment Diagrams Tutorial (TutorialsPoint) [Электронный ресурс] — Режим доступа до ресурсу: https://www.tutorialspoint.com/uml/uml_deployment_diagram.htm
18. Cisco: Introduction to Routing Information Protocol (RIP) [Электронный ресурс] — Режим доступа до ресурсу: <https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html>
19. Software Engineering Body of Knowledge (SWEBOK) [Электронный ресурс] — Режим доступа до ресурсу: <https://www.computer.org/education/bodies-of-knowledge/software-engineering>
20. Testing Computer Software by Cem Kaner, Jack Falk, Hung Quoc Nguyen [Электронный ресурс] — Режим доступа до ресурсу: <https://lingualeo.com/en/jungle/testing-computer-software-cem-kaner-jack-falk-hung-quoc-nguyen-270050>

ДОДАТКИ

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

Лістинг TableObject.cs:

```

using System.Collections;
using System.Collections.Generic;
using System.Runtime.ConstrainedExecution;
using Unity.VisualScripting.Antlr3.Runtime;
using UnityEngine;
using Photon.Pun;

public class TableObject : MonoBehaviour
{
    public List<string> variablesName = new List<string>();
    public Dictionary<string, object> variables = new Dictionary<string,
object>();

    public string inputName = "";
    public bool canBeSelected = false;

    public float move_speed;
    protected Vector3 move_startCoords, move_finishCoords;
    protected Vector3 move_startScale, move_finishScale;
    protected float move_startRotation, move_finishRotation;
    protected float move_progress, move_progressRotate;

    protected virtual void Start()
    {
        move_speed = 0.16f;
    }

    protected virtual void FixedUpdate()
    {
        if (canBeSelected)
        {
            float millisec = System.DateTime.Now.Millisecond +
System.DateTime.Now.Second * 1000f + System.DateTime.Now.Hour * 60000f;
            float clr = Mathf.Sin(millisec / 280f) * 0.07f;
            float clr2 = Mathf.Sin(millisec / 280f) * 0.1f + 0.9f;
            foreach (Material material in
GetComponent<MeshRenderer>().materials)
            {
                material.color = new Color(clr2, clr2, clr2);
                material.SetColor("_EmissionColor", new Color(clr, clr, clr));
            }
        }
        MoveFixedUpdate();
    }
}

```

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІПЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

```

protected virtual void OnMouseDown()
{
    if (canBeSelected)
    {
        GameObject.FindWithTag("PuzzleContainer").GetComponent<PuzzleContainer>().SetInput(inputName, this.gameObject);
        canBeSelected = false;
    }
}

public void AllowBeSelected(string inputName)
{
    this.inputName = inputName;
    canBeSelected = true;
}

public void DenyBeSelected()
{
    inputName = "";
    canBeSelected = false;
    foreach (Material material in GetComponent<MeshRenderer>().materials)
    {
        material.color = Color.white;
        material.SetColor("_EmissionColor", Color.black);
    }
}

public object GetVariable(string name)
{
    if (variablesName.Contains(name))
        return variables[name];
    return null;
}

public void Move(Vector3 finishCoords)
{
    move_startCoords = transform.localPosition;
    move_finishCoords = finishCoords;

    move_progress = 0f;
}

public void Rotate(float finishRotation)
{
    move_startRotation = transform.localEulerAngles.y;

```

```

        move_finishRotation = finishRotation;

        while (move_startRotation < -180f)
            move_startRotation += 360f;
        while (move_startRotation >= 180f)
            move_startRotation -= 360f;

        while (move_finishRotation < -180f)
            move_finishRotation += 360f;
        while (move_finishRotation >= 180f)
            move_finishRotation -= 360f;

        move_progress = 0f;
    }

    public void Upscale(Vector3 finishScale)
    {
        move_startScale = transform.localScale;
        move_finishScale = finishScale;

        move_progress = 0f;

    }

    protected virtual void MoveFixedUpdate()
    {
        //if (view.IsMine)
        {
            if (Vector3.Distance(transform.localScale, move_finishScale) >
0.0001f || Vector3.Distance(transform.localPosition, move_finishCoords) >
0.0001f || Mathf.Abs(transform.localEulerAngles.y - move_finishRotation) >
0.01f)
            {
                transform.localScale = Vector3.Lerp(move_startScale,
move_finishScale, move_progress);
                transform.localPosition = Vector3.Lerp(move_startCoords,
move_finishCoords, move_progress);
                if (Mathf.Abs(move_startRotation - move_finishRotation) >
180f)
                {
                    transform.localEulerAngles =
                        new Vector3(transform.localEulerAngles.x,
move_startRotation - (360f - move_finishRotation +
move_startRotation) * move_progress,
transform.localEulerAngles.z);
                    if (transform.localEulerAngles.y < -180f)

```

```

        transform.localEulerAngles =
            new Vector3(transform.localEulerAngles.x,
                transform.localEulerAngles.y + 360f,
                transform.localEulerAngles.z);
    }
    else
    {
        transform.localEulerAngles =
            new Vector3(transform.localEulerAngles.x,
                (move_finishRotation - move_startRotation) *
move_progress + move_startRotation,
                transform.localEulerAngles.z);
    }
    move_progress += move_speed;
    if (move_progress > 1f)
        move_progress = 1f;
    }
    else
    {
        move_progress = 1f;
    }
    }
}
}

```

Лістинг Card.cs:

```

using System.CodeDom.Compiler;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using Photon.Pun;

public class Card : TableObject
{
    // Візуальні параметри
    public Vector3 localScale;
    public float rounding;
    public Vector2 coordsInTexture, scaleInTexture;
    public bool isOpen;

    public string faceTexturePath;
    public string backTexturePath;

    private void Start()
    {
        //view.RPC("VisualUpdate", RpcTarget.AllBuffered);
        VisualUpdate();
    }
}

```

		Кочубей К.М.			ІПЗ.КР.Б-121-24-ІЗ	Арк.
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

```

    }

    protected override void FixedUpdate()
    {
        base.FixedUpdate();

        MoveFixedUpdate();
    }

    private void MoveFixedUpdate()
    {
        //if (view.IsMine)
        {
            if (!isOpen && Mathf.Abs(transform.eulerAngles.z - 180f) > 0.01f)
            {
                Vector3 startAngles = new Vector3(transform.eulerAngles.x,
transform.eulerAngles.y, 0f);
                Vector3 finishAngles = new Vector3(transform.eulerAngles.x,
transform.eulerAngles.y, 180f);
                transform.eulerAngles = Vector3.Lerp(startAngles,
finishAngles, move_progressRotate);
                move_progressRotate += move_speed;
                if (move_progressRotate > 1f)
                    move_progressRotate = 1f;
            }
            else
            if (isOpen && Mathf.Abs(transform.eulerAngles.z - 0f) > 0.01f)
            {
                Vector3 startAngles = new Vector3(transform.eulerAngles.x,
transform.eulerAngles.y, 180f);
                Vector3 finishAngles = new Vector3(transform.eulerAngles.x,
transform.eulerAngles.y, 0f);
                transform.eulerAngles = Vector3.Lerp(startAngles,
finishAngles, move_progressRotate);
                move_progressRotate += move_speed;
                if (move_progressRotate > 1f)
                    move_progressRotate = 1f;
            }
            else
            {
                move_progressRotate = 1f;
            }
        }
    }

    private void OnMouseEnter()
    {
        if (transform.parent.tag == "MyHand")

```

```

        this.GetComponentInParent<Hand>().Replace(transform.name);
    }

    private void OnMouseExit()
    {
        if (transform.parent.tag == "MyHand")
            this.GetComponentInParent<Hand>().Replace();
    }

    public void SetIsOpen(bool finishIsOpen)
    {
        if (isOpen != finishIsOpen)
            move_progressRotate = 0f;
        isOpen = finishIsOpen;
    }

    [PunRPC]
    public void VisualUpdate()
    {
        Mesh mesh = new Mesh();
        GetComponent<MeshFilter>().mesh = mesh;
        mesh.name = "Grid";

        this.GetComponent<MeshRenderer>().materials = new Material[3]
        {
            Resources.Load<Material>("Base/Materials/CardFace"),
            Resources.Load<Material>("Base/Materials/CardBack"),
            Resources.Load<Material>("Base/Materials/White")
        };

        string gameName = GameObject.FindWithTag("Game").name;
        this.GetComponent<MeshRenderer>().materials[0].mainTexture =
Resources.Load<Texture>(gameName + "/Textures/" + backTexturePath);
        this.GetComponent<MeshRenderer>().materials[1].mainTexture =
Resources.Load<Texture>(gameName + "/Textures/" + faceTexturePath);

        Vector3 pos = transform.position;
        float skalRounding = Mathf.Min(localScale.x, localScale.z) / 2f *
rounding;
        Vector3[] vertices = new Vector3[56];
        int[][] trianglesFront = new int[2][];
        trianglesFront[0] = new int[78];
        trianglesFront[1] = new int[78];
        int[] trianglesEdges = new int[144];

```

```

Vector2[] uvs = new Vector2[vertices.Length];

for (int j = 0; j < 2; j++)
{
    float invert = j == 0 ? -1f : 1f;
    // Центральний прямокутник
    vertices[28 * j + 0] = new Vector3(-invert * localScale.x / 2f,
localScale.y * invert / 2f, -localScale.z / 2f + skalRounding);
    vertices[28 * j + 1] = new Vector3(+invert * localScale.x / 2f,
localScale.y * invert / 2f, -localScale.z / 2f + skalRounding);
    vertices[28 * j + 2] = new Vector3(-invert * localScale.x / 2f,
localScale.y * invert / 2f, +localScale.z / 2f - skalRounding);
    vertices[28 * j + 3] = new Vector3(+invert * localScale.x / 2f,
localScale.y * invert / 2f, +localScale.z / 2f - skalRounding);
    trianglesFront[j][0] = 28 * j + 0;
    trianglesFront[j][1] = 28 * j + 2;
    trianglesFront[j][2] = 28 * j + 1;
    trianglesFront[j][3] = 28 * j + 1;
    trianglesFront[j][4] = 28 * j + 2;
    trianglesFront[j][5] = 28 * j + 3;

    // Верхній прямокутник
    vertices[28 * j + 4] = new Vector3(-invert * localScale.x / 2f +
invert * skalRounding, localScale.y * invert / 2f, -localScale.z / 2f);
    vertices[28 * j + 5] = new Vector3(+invert * localScale.x / 2f -
invert * skalRounding, localScale.y * invert / 2f, -localScale.z / 2f);
    vertices[28 * j + 6] = new Vector3(-invert * localScale.x / 2f +
invert * skalRounding, localScale.y * invert / 2f, -localScale.z / 2f +
skalRounding);
    vertices[28 * j + 7] = new Vector3(+invert * localScale.x / 2f -
invert * skalRounding, localScale.y * invert / 2f, -localScale.z / 2f +
skalRounding);
    trianglesFront[j][6] = 28 * j + 4;
    trianglesFront[j][7] = 28 * j + 6;
    trianglesFront[j][8] = 28 * j + 5;
    trianglesFront[j][9] = 28 * j + 5;
    trianglesFront[j][10] = 28 * j + 6;
    trianglesFront[j][11] = 28 * j + 7;

    // Нижній прямокутник
    vertices[28 * j + 8] = new Vector3(-invert * localScale.x / 2f +
invert * skalRounding, localScale.y * invert / 2f, +localScale.z / 2f -
skalRounding);
    vertices[28 * j + 9] = new Vector3(+invert * localScale.x / 2f -
invert * skalRounding, localScale.y * invert / 2f, +localScale.z / 2f -
skalRounding);
    vertices[28 * j + 10] = new Vector3(-invert * localScale.x / 2f +
invert * skalRounding, localScale.y * invert / 2f, +localScale.z / 2f);
    vertices[28 * j + 11] = new Vector3(+invert * localScale.x / 2f -
invert * skalRounding, localScale.y * invert / 2f, +localScale.z / 2f);
}

```

```

trianglesFront[j][12] = 28 * j + 8;
trianglesFront[j][13] = 28 * j + 10;
trianglesFront[j][14] = 28 * j + 9;
trianglesFront[j][15] = 28 * j + 9;
trianglesFront[j][16] = 28 * j + 10;
trianglesFront[j][17] = 28 * j + 11;

// Четверть кола зверху праворуч
for (int i = 1; i < 5; i++)
    vertices[28 * j + 11 + i] = new Vector3(+invert * localScale.x
/ 2f - invert * skalRounding + invert * skalRounding * Mathf.Cos(Mathf.PI /
10f * i), localScale.y * invert / 2f, +localScale.z / 2f - skalRounding +
skalRounding * Mathf.Sin(Mathf.PI / 10f * i));
trianglesFront[j][18] = 28 * j + 9;
trianglesFront[j][19] = 28 * j + 12;
trianglesFront[j][20] = 28 * j + 3;
trianglesFront[j][21] = 28 * j + 9;
trianglesFront[j][22] = 28 * j + 13;
trianglesFront[j][23] = 28 * j + 12;
trianglesFront[j][24] = 28 * j + 9;
trianglesFront[j][25] = 28 * j + 14;
trianglesFront[j][26] = 28 * j + 13;
trianglesFront[j][27] = 28 * j + 9;
trianglesFront[j][28] = 28 * j + 15;
trianglesFront[j][29] = 28 * j + 14;
trianglesFront[j][30] = 28 * j + 9;
trianglesFront[j][31] = 28 * j + 11;
trianglesFront[j][32] = 28 * j + 15;

// Четверть кола зверху ліворуч
for (int i = 1; i < 5; i++)
    vertices[28 * j + 15 + i] = new Vector3(-invert * localScale.x
/ 2f + invert * skalRounding - invert * skalRounding * Mathf.Cos(Mathf.PI /
10f * i), localScale.y * invert / 2f, +localScale.z / 2f - skalRounding +
skalRounding * Mathf.Sin(Mathf.PI / 10f * i));
trianglesFront[j][33] = 28 * j + 8;
trianglesFront[j][34] = 28 * j + 2;
trianglesFront[j][35] = 28 * j + 16;
trianglesFront[j][36] = 28 * j + 8;
trianglesFront[j][37] = 28 * j + 16;
trianglesFront[j][38] = 28 * j + 17;
trianglesFront[j][39] = 28 * j + 8;
trianglesFront[j][40] = 28 * j + 17;
trianglesFront[j][41] = 28 * j + 18;
trianglesFront[j][42] = 28 * j + 8;
trianglesFront[j][43] = 28 * j + 18;
trianglesFront[j][44] = 28 * j + 19;
trianglesFront[j][45] = 28 * j + 8;
trianglesFront[j][46] = 28 * j + 19;
trianglesFront[j][47] = 28 * j + 10;

```

```

// Четверть кола знизу ліворуч
for (int i = 1; i < 5; i++)
    vertices[28 * j + 19 + i] = new Vector3(-invert * localScale.x
/ 2f + invert * skalRounding - invert * skalRounding * Mathf.Cos(Mathf.PI /
10f * i), localScale.y * invert / 2f, -localScale.z / 2f + skalRounding -
skalRounding * Mathf.Sin(Mathf.PI / 10f * i));
    trianglesFront[j][48] = 28 * j + 6;
    trianglesFront[j][49] = 28 * j + 20;
    trianglesFront[j][50] = 28 * j + 0;
    trianglesFront[j][51] = 28 * j + 6;
    trianglesFront[j][52] = 28 * j + 21;
    trianglesFront[j][53] = 28 * j + 20;
    trianglesFront[j][54] = 28 * j + 6;
    trianglesFront[j][55] = 28 * j + 22;
    trianglesFront[j][56] = 28 * j + 21;
    trianglesFront[j][57] = 28 * j + 6;
    trianglesFront[j][58] = 28 * j + 23;
    trianglesFront[j][59] = 28 * j + 22;
    trianglesFront[j][60] = 28 * j + 6;
    trianglesFront[j][61] = 28 * j + 4;
    trianglesFront[j][62] = 28 * j + 23;

// Четверть кола знизу праворуч
for (int i = 1; i < 5; i++)
    vertices[28 * j + 23 + i] = new Vector3(+invert * localScale.x
/ 2f - invert * skalRounding + invert * skalRounding * Mathf.Cos(Mathf.PI /
10f * i), localScale.y * invert / 2f, -localScale.z / 2f + skalRounding -
skalRounding * Mathf.Sin(Mathf.PI / 10f * i));
    trianglesFront[j][63] = 28 * j + 7;
    trianglesFront[j][64] = 28 * j + 1;
    trianglesFront[j][65] = 28 * j + 24;
    trianglesFront[j][66] = 28 * j + 7;
    trianglesFront[j][67] = 28 * j + 24;
    trianglesFront[j][68] = 28 * j + 25;
    trianglesFront[j][69] = 28 * j + 7;
    trianglesFront[j][70] = 28 * j + 25;
    trianglesFront[j][71] = 28 * j + 26;
    trianglesFront[j][72] = 28 * j + 7;
    trianglesFront[j][73] = 28 * j + 26;
    trianglesFront[j][74] = 28 * j + 27;
    trianglesFront[j][75] = 28 * j + 7;
    trianglesFront[j][76] = 28 * j + 27;
    trianglesFront[j][77] = 28 * j + 5;
}

int[] verEdges = { 4, 5, 27, 26, 25, 24, 1, 3, 12, 13, 14, 15, 11, 10,
19, 18, 17, 16, 2, 0, 20, 21, 22, 23, 4, 5 };
for (int i = 0; i < verEdges.Length - 2; i++)
{

```

```

        trianglesEdges[i * 6 + 0] = verEdges[i + 2];
        trianglesEdges[i * 6 + 1] = 28 + verEdges[24 - i];
        trianglesEdges[i * 6 + 2] = verEdges[i + 1];
        trianglesEdges[i * 6 + 3] = 28 + verEdges[24 - i - 1];
        trianglesEdges[i * 6 + 4] = 28 + verEdges[24 - i];
        trianglesEdges[i * 6 + 5] = verEdges[i + 2];
    }

    mesh.vertices = vertices;
    mesh.subMeshCount = 3;
    mesh.SetTriangles(trianglesFront[0], 0);
    mesh.SetTriangles(trianglesFront[1], 1);
    mesh.SetTriangles(trianglesEdges, 2);
    for (int i = 0; i < vertices.Length / 2; i++)
        uvs[i] = new Vector2((float)vertices[i].x / localScale.x - 0.5f, -
(float)vertices[i].z / localScale.z - 0.5f);
    for (int i = vertices.Length / 2; i < vertices.Length; i++)
        uvs[i] = new Vector2(-(float)vertices[i].x / localScale.x - 0.5f)
* scaleInTexture.x + coordsInTexture.x, -(float)vertices[i].z / localScale.z
- 0.5f) * scaleInTexture.y + coordsInTexture.y);
    mesh.uv = uvs;
    mesh.RecalculateNormals();

    GetComponent<BoxCollider>().size = localScale;
}
}

```

Лістинг Hand.cs:

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using UnityEngine;

public class Hand : Container
{
    public GameObject prefCard;
    public float cardScale;
    public bool isMyHand;

    private void Start()
    {
        base.Start();
        move_finishCoords = transform.GetComponent<Prefab>().position;
        move_finishRotation = transform.GetComponent<Prefab>().rotation.z;
        move_finishScale = transform.GetComponent<Prefab>().scale;
    }
}

```

```

    }

    public override void AddCard(GameObject card)
    {
        cards.Add(card);
        cards[cards.Count - 1].transform.parent = transform;
        Replace();
    }

    public override void Shuffle(int seed = -1)
    {
        if (seed == -1)
            seed = (int)(UnityEngine.Random.value * 100000000);
        if (cards.Count <= 1)
            return;

        System.Random random = new System.Random(seed);

        for (short i = 0; i < cards.Count * 10; i++)
        {
            short first = (short)(random.Next(0, cards.Count));
            short second = (short)(random.Next(0, cards.Count));
            GameObject t = cards[first];
            cards[first] = cards[second];
            cards[second] = t;
        }

        Replace();
    }

    public override GameObject TakeCard()
    {
        if (cards.Count == 0)
            return null;
        GameObject result = cards[cards.Count - 1];
        cards.RemoveAt(cards.Count - 1);
        Replace();
        return result;
    }

    public override GameObject TakeCard(GameObject card)
    {
        for (int i = 0; i < cards.Count; i++)
            if (cards[i] == card)
            {
                cards.RemoveAt(i);
                Replace();
            }
    }

```

```

        return card;
    }
    return null;
}

public void Replace(string cardName = "")
{
    float radius = 15f, selectRadius;
    if (cards.Count < 8)
    {
        float angle = 0.1f;
        for (short i = 0; i < cards.Count; i++)
        {
            if (cards[i].GetComponent<Transform>().name == cardName)
                selectRadius = 16.6f;
            else
                selectRadius = 15f;
            float locAngle = (Mathf.PI - (cards.Count - 1) * angle) / 2 +
i * angle + (isMyHand ? 0 : Mathf.PI);
            cards[i].GetComponent<Card>().Move(
                new Vector3(Mathf.Cos(locAngle) * selectRadius, -0.1f * i,
Mathf.Sin(locAngle) * selectRadius - radius));
            cards[i].GetComponent<Card>().Rotate(-(locAngle - Mathf.PI /
2f) / Mathf.PI * 180f);
            cards[i].GetComponent<Card>().Upscale(new Vector3(cardScale,
cardScale, cardScale));
            cards[i].GetComponent<Card>().SetIsOpen(isMyHand);
        }
    }
    else
    {
        float firstAngle = 17.189f;
        for (short i = 0; i < cards.Count; i++)
        {
            if (cards[i].GetComponent<Transform>().name == cardName)
                selectRadius = 16.6f;
            else
                selectRadius = 15f;
            float locAngle = (2f * firstAngle / (cards.Count - 1) * i +
90f - firstAngle) / 180f * Mathf.PI + (isMyHand ? 0 : Mathf.PI);
            cards[i].GetComponent<Card>().Move(
                new Vector3(Mathf.Cos(locAngle) * selectRadius, -0.6f /
(cards.Count - 1) * i, Mathf.Sin(locAngle) * selectRadius - radius));
            cards[i].GetComponent<Card>().Rotate(-(locAngle - Mathf.PI /
2f) / Mathf.PI * 180f);
            cards[i].GetComponent<Card>().Upscale(new Vector3(cardScale,
cardScale, cardScale));
            cards[i].GetComponent<Card>().SetIsOpen(isMyHand);
        }
    }
}
}

```