ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка карткової гри "Survive After The Apocalypse"»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Костянтин БОЙКО
(підпис)

Виконав: здобувач(ка) вищої освіти групи ПД-43

Костянтин БОЙКО

Керівник: Олександр ЯРОШЕВСЬКИЙ
асистент кафедри

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бойку Костянтину Вікторовичу

1. Тема кваліфікаційної роботи: «Розробка карткової гри “Survive After The Apocalypse”»
керівник кваліфікаційної роботи асистент кафедри Олександр ЯРОШЕВСЬКИЙ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки цифрових та карткових ігор, принципи побудови ігрових механік та організації багатокористувацького ігрового процесу, рекомендації щодо проєктування інтерфейсу користувача для ігрових додатків. Опис роботи настільних і цифрових карткових ігор у багатокористувацькому режимі. Документація ігрового рушія Unity, мови програмування C#, мережевого рішення Photon PUN 2.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд жанру карткових ігор, аналіз існуючих аналогів та визначення вимог до програмного забезпечення.

2. Огляд та аналіз засобів реалізації карткової гри «Survive After The Apocalypse».
3. Проєктування архітектури карткової гри «Survive After The Apocalypse».
4. Програмна реалізація та тестування карткової гри «Survive After The Apocalypse».

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма прецедентів.
5. Діаграма класів
6. Екранні форми.
7. Демонстрація роботи застосунку
8. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд ігрових механік, архітектурних підходів та мережевих технологій для розробки багатокористувацьких карткових ігор	14.03-20.03.2025	
4	Проєктування архітектури карткової гри, ігрових механік, мережевої взаємодії	21.03-30.03.2025	
5	Програмна реалізація клієнтської та серверної частини гри, розробка ігрових механік та мережевої взаємодії	31.03-20.04.2025	
6	Тестування гри, усунення помилок, оптимізація продуктивності	21.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Костянтин БОЙКО
(підпис)

Керівник
кваліфікаційної роботи _____ Олександр ЯРОШЕВСЬКИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 72 стор., 1 табл., 28 рис., 20 джерел

Мета роботи – виявити ефективні підходи до організації ігрового процесу в умовах соціальної взаємодії гравців шляхом створення фазової моделі онлайн-гри на виживання в бункері, яка забезпечує послідовне відкриття карт, обговорення та голосування в мультиплеєрному середовищі.

Об'єкт дослідження – процес розробки мультиплеєрних карткових ігор з використанням сучасних технологій мережевої взаємодії.

Предмет дослідження – методи та засоби реалізації мережевої синхронізації ігрового процесу та взаємодії між гравцями в карткових іграх з використанням мережевої технології Photon.

Короткий зміст роботи: У роботі проаналізовано предметну галузь карткових ігор, досліджено існуючі аналоги та методи реалізації цифрових багатокористувацьких карткових ігор. Проведено огляд популярних ігор подібного жанру. Розроблено концепцію та ігрову механіку карткової гри «Survive After The Apocalypse». Спроєктовано архітектуру клієнтської та серверної частини, реалізовано основні функціональні можливості, такі як: створення ігрових сесій, підключення гравців, роздача карт, хід гри за чергою, та визначення переможця. Для реалізації багатокористувацької взаємодії використано Photon PUN 2, для графічної складової та ігрової логіки — Unity, Adobe Illustrator та C#. Проведено тестування основного функціоналу гри, аналіз стабільності роботи та перевірку мережевої взаємодії між гравцями.

Сферою використання гри є організація соціальної взаємодії та стратегічного мислення в процесі колективного прийняття рішень в умовах постапокаліптичного виживання.

КЛЮЧОВІ СЛОВА: UNITY, C#, PHOTON PUN 2, БАГАТОКОРИСТУВАЦЬКА ГРА, ІГРОВИЙ РУШІЙ, NPC

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment

API – Application Programming Interface

RPC – Remote Procedure Call

FPS – Frames Per Second

UI – User Interface

UX – User Experience

UDP – User Datagram Protocol

TCP – Transmission Control Protocol

PUN – Photon Unity Networking

SRS – Survival Role-playing System

ID – Identifier

SDK – Software Development Kit

LAN – Local Area Network

WAN – Wide Area Network

ВСТУП

Актуальність: Розробка багатокористувацьких карткових ігор є актуальним напрямом у сфері цифрових розваг та інтерактивного дозвілля. З розвитком технологій мережевої взаємодії та популяризацією онлайн-ігор зростає потреба у створенні якісних ігрових проєктів, які поєднують у собі елементи соціальної взаємодії, стратегічного мислення та колективного прийняття рішень. Особливий інтерес викликають ігри з тематикою виживання, де гравці об’єднуються або протистоять один одному в складних умовах обмежених ресурсів.

Гра постає як всебічний пакет усіх необхідних механік для організації інтерактивного багатокористувацького ігрового процесу в одному середовищі. У ній представлені основні компоненти карткових ігор: роздача карт, етапи обговорення, фазовий перехід між ходами, а також голосування за вибування гравців. Додаткові цифрові інструменти, такі як система керування чергою ходів, обмежений час для прийняття рішень та мережеве збереження стану гри допомагають забезпечити рівні умови для всіх учасників та підтримують динаміку ігрового процесу. Більш того, мультиплеєрна платформа дозволяє користувачам долучатися до спільних сесій, взаємодіяти, обговорювати і приймати рішення у реальному часі, незалежно від їхнього місцезнаходження. Завдяки цьому гра стає більш захопливою, соціально орієнтованою та адаптованою під сучасні геймерські вимоги.

Об’єктом дослідження є процес розробки мультиплеєрних карткових ігор з використанням сучасних технологій мережевої взаємодії.

Предметом дослідження є методи та засоби реалізації мережевої синхронізації ігрового процесу та взаємодії між гравцями в карткових іграх з використанням мережевої технології Photon.

Метою роботи є виявити ефективні підходи до організації ігрового процесу в умовах соціальної взаємодії гравців шляхом створення фазової моделі онлайн-гри

на виживання в бункері, яка забезпечує послідовне відкриття карт, обговорення та голосування в мультиплеєрному середовищі.

Методи дослідження: Першим кроком було проведено аналіз існуючих рішень у сфері цифрових багатокористувацьких карткових ігор для формування початкових функціональних та нефункціональних вимог до майбутньої гри. Далі здійснено огляд сучасних технологій розробки мультиплеєрних проєктів, що відповідають специфіці ігрового процесу та обраному жанру, після чого було обрано ігровий рушій Unity для побудови клієнтської та серверної частини гри, мову програмування C# для реалізації логіки та мережеве рішення Photon PUN 2 для забезпечення взаємодії між гравцями в режимі реального часу. Для розробки дизайну користувацького інтерфейсу використано Adobe Illustrator, Дослідження процесу реалізації гри проводилося безпосередньо в процесі розробки і тестування ігрових механік та мережевої взаємодії.

Наукова новизна роботи визначається наступними функціональними складовими: впровадження фазової моделі ігрового процесу для мультиплеєрної карткової гри з соціальною взаємодією гравців; реалізація системи послідовного відкриття карт, обговорення, прийняття колективних рішень та голосування в режимі реального часу; забезпечення синхронізації станів гри між усіма гравцями з використанням мережевої технології Photon PUN 2. Гра постає як всебічний пакет сучасних ігрових механік, спрямованих на взаємодію між гравцями в онлайн-середовищі.

Практична значущість результатів полягає в можливості використання розробленої гри для організації інтерактивного дозвілля, дослідження механік соціальної взаємодії в іграх, а також створення аналогічних багатокористувацьких карткових проєктів для різних десктопних платформ.

1 АНАЛІЗ БАГАТОКОРИСТУВАЦЬКИХ КАРТКОВИХ ІГОР

1.1 Мультиплеєрна карткова гра

Мультиплеєрна карткова гра — це спеціалізоване програмне забезпечення, яке надає користувачам можливість взяти участь у спільному ігровому процесі, що відбувається у реальному часі з іншими гравцями через мережу. Такі ігри дозволяють гравцям взаємодіяти між собою, змагатися або співпрацювати, використовуючи віртуальні картки, що містять певні ігрові події, властивості чи дії.

Основна мета мультиплеєрної карткової гри полягає у створенні цікавого та динамічного ігрового середовища, де гравці можуть проявляти стратегічне мислення, комунікувати та приймати спільні рішення в умовах обмеженого часу та ресурсів.

Основні компоненти мультиплеєрної карткової гри включають:

– Створення та налаштування ігрових кімнат із заданими параметрами кількості гравців та сценарієм гри.

– Послідовний перехід між фазами відкриття карт, обговорення, голосування та підведення підсумків.

– Синхронізація ігрового стану між усіма учасниками в режимі реального часу.

– Організація прийняття колективних рішень щодо ігрових подій чи вибування гравців.

Основні цілі мультиплеєрної карткової гри:

– створення інтерактивного ігрового простору;

– розвиток стратегічного мислення гравців;

– організація соціальної взаємодії в ігровому середовищі;

– підвищення навичок прийняття колективних рішень;

– забезпечення динамічного дозвілля;

– тестування механік фазового ігрового процесу;

– перевірка стабільності мережевої взаємодії.

Оцінимо переваги та недоліки вебсайту як засобу для вивчення японської мови.

Переваги:

– Ігри доступні для користувачів з будь-якої точки світу, де є підключення до інтернету.

– Користувачі можуть самостійно створювати ігрові кімнати та налаштовувати параметри сесій.

– Можливість об’єднувати гравців у команди чи групи та приймати спільні рішення.

– Завдяки голосовим обговоренням та фазовій системі гравці отримують ігровий досвід, максимально наближений до реальної настільної гри.

– Вартість цифрових карткових ігор значно нижча, ніж друкованих аналогів, а оновлення механік відбувається швидко та без додаткових витрат.

Недоліки:

– Потреба у стабільному інтернет-з’єднанні та відповідних пристроях.

– Деякі користувачі можуть мати труднощі з опануванням правил та механік гри.

– Відсутність обов’язковості завершувати гру може призводити до дострокового виходу окремих гравців.

1.2 Специфіка розробки багатокористувацьких карткових ігор

Розробка багатокористувацьких карткових ігор має низку особливостей, які суттєво відрізняють цей процес від створення інших типів ігор, зокрема класичних настільних чи однокористувацьких проєктів. Розглянемо їх.

Механіка карткової гри:

– Наявність чітко структурованих ігрових фаз: роздача карт, обговорення, прийняття рішень, підрахунок результатів.

– Розподіл ролей і карт між гравцями з обмеженим доступом до інформації, кожен гравець бачить лише власні карти.

– Використання випадкових подій, що впливають на ігровий процес і змінюють його хід.

Організація мультиплеєрної взаємодії:

– Мережеве з’єднання всіх гравців у реальному часі з мінімальною затримкою.

– Забезпечення однакового і синхронізованого ігрового стану на клієнтах усіх учасників.

– Створення ігрових сесій з параметрами: кількість гравців, режим гри, тривалість ходів тощо.

Фазовий геймплей:

– Чітке розділення гри на послідовні етапи, де всі гравці виконують певні дії за чергою

– Забезпечення синхронного переходу між фазами для всіх користувачів.

Мережеві технології:

– Необхідність застосування ефективних рішень для обробки одночасних підключень.

– Підтримка стабільного з’єднання при можливих обриваннях чи повторному підключенні гравців.

1.3 Цільова аудиторія

Гра «Survive After The Apocalypse» орієнтована на широке коло користувачів, які відрізняються за своїми інтересами, стилем гри та цілями. До основних категорій цільової аудиторії належать:

– Прихильники настільних ігор та карткових стратегій – гравці, які полюбляють настільні карткові ігри з соціальними елементами.

– Фанати соціальних та психологічних ігор – гравці, які цікавляться іграми, побудованими на комунікації, маніпуляції, психологічному тиску та прийнятті колективних рішень.

– Онлайн-геймери, які шукають мультиплеєрні ігри з нестандартним геймплеєм, користувачі, які віддають перевагу не класичним шутерам чи MOBA, а іграм, де важливіша взаємодія між гравцями, стратегічне мислення та сюжетна напруженість.

– Контентмейкери, які створюють відеоогляди, стріми або аналітичний контент про мультиплеєрні ігри.

– Гравці, зацікавлені у виживанні та постапокаліптичних сюжетах – користувачі, яких приваблює тематика апокаліпсису, боротьби за виживання та прийняття складних рішень в екстремальних умовах.

1.4 Дослідження існуючих аналогів

На сьогоднішній день одними з найпопулярніших проєктів, схожих за концепцією або механікою на гру «Survive After The Apocalypse», є настільна гра «Бункер» та мобільна гра «Shelter». Розглянемо їх детальніше.

1.4.1 Бункер

«Бункер» — популярна настільна психологічна карткова гра, в якій гравці опиняються в умовах постапокаліпсису та повинні вирішити, хто заслуговує на виживання в бункері. Гравці отримують випадкові картки з характеристиками персонажів, їх професіями, станом здоров’я та іншими особливостями, після чого починається серія обговорень та голосувань.

Основні функції гри «Бункер» включають:

– Картки персонажів та подій: набір характеристик, що впливають на рішення гравців.

– Активна комунікація між гравцями для аргументації свого права залишитися в бункері.

– Щотурове голосування за виключення гравця.

– додаткові умови виживання або перешкоди.

Переваги:

– Соціальна взаємодія та психологічна напруга.

– Велика варіативність сценаріїв та ігрових комбінацій.

– Простота правил та швидкий старт для нових гравців.

Недоліки:

– Відсутність цифрової версії гри.

– Обмеження за кількістю гравців та необхідність фізичної присутності.

– Складнощі з тривалістю гри при великій кількості учасників.

Рис. 1.1 Приклад карток в настільній грі «Бункер»

1.4.2 Shelter

Shelter — це багатокористувацька психологічна карткова онлайн-гра, дія якої відбувається після глобальної катастрофи. Гравці беруть на себе ролі випадкових персонажів, кожен із яких має набір унікальних характеристик. Основна мета гри — забезпечити своє виживання, довівши необхідність своєї присутності в укритті, і переконати інших гравців вигнати суперників.

Особливість гри полягає у поступовому розкритті характеристик персонажів та активній соціальній взаємодії гравців — обговоренні, аргументуванні,

маніпулюванні та голосуванні за виключення найслабших або непотрібних учасників.

Основні функції Shelter включають:

– Кожен гравець отримує набір параметрів: професію, стан здоров'я, вік, стать, хобі, фобії, додаткові навички та людські якості.

– На початку гри кожен отримує по дві додаткові карти - «знання» та «дія», які можна використати в будь-який момент гри на свою користь.

– На початку першого раунду всі відкривають лише професії. У кожному наступному раунді - по одній характеристиці.

– Після відкриття кожної характеристики гравці пояснюють, чому саме вони повинні залишитися в Бункері.

– Починаючи з другого раунду, після обговорення усі гравці голосують за найменш корисного персонажа, який покидає гру.

– Гра закінчується, коли в укритті залишається половина початкових гравців.

Переваги:

– Соціальна взаємодія в реальному часі.

– Психологічна напруга та стратегічне планування.

– Велика кількість варіацій ігрових ситуацій завдяки унікальним наборам характеристик.

– Простота освоєння правил.

– Акцент на комунікацію, аргументацію та переконання.

Недоліки:

– Тривалість ігрової сесії іноді затягується від кількості учасників.

– Відсутність сюжетної кампанії або розвитку персонажів.

– Вимагає постійного онлайн-з’єднання.

Рис. 1.3 Приклад екрану ігрового процесу .

Таблиця 1.1

Зведена таблиця аналізу існуючих карткових ігор

Показник	**«Бункер»**	**Shelter (мобільна)**
Тип гри	Настільна карткова гра	Карткова гра
Соціальна взаємодія	Жива комунікація	Відсутня
Фазовий геймплей	Присутній	Відсутній
Голосування	Усне	Відсутнє
Рольова система	Випадкові персонажі	Випадкові персонажі
Додаткові карти	Відсутні	Ресурси та предмети
Доступність на платформах	Фізична гра	Android, iOS

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих карткових ігор

Показник	**«Бункер»**	**Shelter (мобільна)**
Особливості	Соціальна психологічна гра	Соціальна психологічна гра
Мотивація гравців	Переконати інших залишити	Переконати інших залишити
Недоліки	Потреба зібрати гравців вживу	Відсутність впровадження комунікації в середені гри
Цільова аудиторія	Фанати настільних ігор	Фанати виживання та стратегії

1.5 Визначення вимог до застосунку

Проаналізувавши сутність гри, її специфіку, цільову аудиторію та існуючі аналоги, було визначено наступні вимоги до багатокористувацької карткової гри «Survive After The Apocalypse»:

– Можливість введення імені гравця перед початком сесії.

–Можливість створювати ігрову кімнату з унікальним кодом та приєднуватися до неї іншим гравцям.

– Система випадкового розподілу карт персонажів, професій, особливостей, фобій та інших типів карт між гравцями на початку гри.

– Реалізація фазового ігрового процесу, при якому в кожному раунді відкривається одна з характеристик персонажа.

– Можливість перегляду історії відкриття карт кожного гравця на попередніх етапах гри.

– Оновлення станів і подій гри для всіх гравців майже одночасно (затримка не більше 1-2 секунд).

– Підтримка одночасного підключення до 10 гравців в одній ігровій кімнаті.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) – це програмне забезпечення, що об’єднує в собі набір засобів для написання, налагодження, тестування та профілювання програмного коду. Воно дозволяє розробникам ефективно створювати програмне забезпечення, зменшуючи час на налаштування та обслуговування окремих інструментів.

2.1.1 Unity

Unity — це потужний ігровий рушій, який дозволяє створювати дво- та тривимірні ігри для різноманітних платформ. Платформа підтримує широкі можливості для роботи з анімаціями, фізикою, мережевими підключеннями, візуалізацією та системами управління ресурсами. Unity активно використовується в індустрії для створення як великих проєктів, так і інді-ігор.

Для розробки багатокористувацької карткової гри «Survive After The Apocalypse» Unity було обрано завдяки вбудованим можливостям організації мережевої взаємодії, роботі зі сценаріями на мові C#, інтеграції з хмарними сервісами та простоті використання інструментів для побудови графічного інтерфейсу.

2.1.2 Visual Studio

Visual Studio від Microsoft є потужним інтегрованим середовищем розробки (IDE), яке широко використовується для програмування на C#, зокрема при створенні ігор на Unity. Воно пропонує розробникам комплексний набір інструментів, що значно прискорює та спрощує процес написання коду.

Серед ключових переваг Visual Studio для роботи з C# та Unity можна відзначити:

– Розширену підтримку синтаксису C#, включаючи підсвічування коду, інтелектуальне автодоповнення (IntelliSense) та навігацію по кодовій базі, що підвищує продуктивність розробника.

– Інтеграцію з Unity, яка дозволяє безпосередньо в IDE редагувати скрипти, запускати гру та використовувати спеціалізовані інструменти для роботи з ігровими об'єктами.

– Потужні засоби налагодження, такі як точки зупину, покрокове виконання коду та оглядачі змінних, що робить процес виявлення та виправлення помилок більш ефективним.

– Вбудовану підтримку систем контролю версій, зокрема Git, що спрощує колаборативну розробку та версійність проекту.

Visual Studio також пропонує додаткові можливості для професійних розробників, включаючи інструменти для рефакторингу коду, аналізу продуктивності та тестування. Це робить його одним з найкращих виборів для розробки ігор на Unity, поєднуючи в собі потужність, зручність та широкий функціонал для створення якісних програмних рішень.

2.1.3 Photon PUN 2

Для організації мережевої взаємодії в грі було застосовано Photon PUN 2 (Photon Unity Networking) - спеціалізований хмарний сервіс для реалізації мультиплеєрного режиму в реальному часі. Дане рішення надає комплексний набір інструментів для синхронізації ігрового процесу між різними клієнтами.

Архітектура клієнт-серверної взаємодії

– Використовується оптимізована хмарна інфраструктура Photon Cloud.

– Забезпечує надійне з'єднання з мінімальними затримками (low-latency).

– Автоматичний підбір найближчого сервера для кожного гравця.

Функціонал ігрових кімнат

– Система дозволяє створювати та керувати віртуальними кімнатами.

– Реалізовано механізми обмеження кількості гравців.

Механізми синхронізації

– Ефективна система Remote Procedure Calls (RPC) для передачі подій.

– Підтримка синхронізації ігрових об'єктів через PhotonView.

– Оптимізовані протоколи передачі даних.

Додаткові переваги

– Інтеграція з Unity через спеціалізований SDK.

– Підтримка крос-платформової взаємодії.

– Вбудована система чату та обміну повідомленнями.

Photon PUN 2 значно спрощує процес реалізації мережевого коду, дозволяючи розробникам зосередитись на ігровій логіці замість створення власної мережевої інфраструктури. Сервіс особливо ефективний для швидких action-ігор, де критичною є мінімізація затримок і плавність ігрового процесу.

2.1.4 Adobe Illustrator

Для розробки графічних елементів (карт персонажів, карт подій, іконок та інтерфейсних елементів) використовується Adobe Illustrator. Це професійний векторний графічний редактор, який забезпечує високоякісну візуалізацію, масштабованість ресурсів та можливість швидкого експорту у формати, сумісні з Unity. Adobe Illustrator став невід'ємною частиною конвеєра створення графіки для проекту, дозволяючи дизайнерам ефективно реалізовувати складні візуальні концепції та швидко адаптувати їх під вимоги ігрового рушія. Векторний підхід особливо цінний при створенні інтерфейсних елементів та адаптивних ігрових асетів, забезпечуючи високу якість відображення на будь-яких пристроях.

2.2 Мова програмування

Мова програмування — це формалізована мова, призначена для комунікації інструкцій до машини, зокрема комп'ютера. Вона використовується для створення програм, які виконують алгоритми. Більшість мов програмування складається з інструкцій для комп'ютерів. Вони дозволяють розробникам ефективно створювати програми, виражаючи розрахунки та управління даними.

2.2.1 C#

C# — є сучасною об’єктно-орієнтованою мовою програмування зі строгою типізацією, розробленою Microsoft у рамках платформи .NET. Ця мова стала основним інструментом для розробки ігор, особливо при роботі з рушієм Unity, де вона використовується для створення ігрової логіки та механік.

Об’єктно-орієнтований підхід C# дозволяє ефективно організовувати код через використання класів, інтерфейсів, наслідування та поліморфізму. Це особливо важливо в ігровій розробці, де необхідно керувати складними системами взаємодії об’єктів, фізикою, штучним інтелектом та іншими аспектами гри. Строга типізація мови допомагає уникнути багатьох помилок на етапі компіляції, що значно підвищує надійність коду.

Інтеграція C# з Unity надає розробникам потужні інструменти для реалізації ігрових механік. Unity використовує компонентний підхід, де кожен об’єкт гри може містити різноманітні скрипти, написані на C#. Такі можливості, як події, корутини та автоматичне керування пам’яттю, спрощують процес розробки та оптимізації ігор.

Крім того, C# підтримує велику кількість бібліотек і фреймворків, які розширюють його функціональність. Завдяки .NET Standard та Unity API розробники отримують доступ до готових рішень для роботи з графікою, звуком, мережевими функціями та іншими критично важливими компонентами сучасних ігор.

2.3 Організація зберігання ігрових данних

Особливістю реалізації даних карт у грі є використання системи ScriptableObject — спеціальних об’єктів у Unity, які дозволяють зберігати структуру даних окремо від ігрових об’єктів. Це дозволяє:

–Зберігати характеристики карт безпосередньо у проєкті без потреби підключення до зовнішньої бази даних.

–Централізовано редагувати властивості карт без зміни префабів.

–Забезпечити оптимізацію продуктивності, оскільки дані карт завантажуються лише один раз при запуску гри.

–Уникати дублювання даних і спростити процес налаштування та тестування.

Для візуального представлення карт використано префаби — заздалегідь налаштовані шаблони об’єктів, які динамічно підвантажуються в ігрову сцену під час виконання. Це дозволяє швидко та ефективно відображати картки з відповідним наповненням, зберігаючи стабільну продуктивність та мінімальне навантаження на пам’ять.

2.4 Система контролю версій

GitHub — це сучасна веб-платформа для розміщення та управління програмним кодом, яка ґрунтується на розподіленій системі контролю версій Git. З моменту свого заснування вона перетворилася на один із ключових інструментів у сфері розробки ПЗ, забезпечуючи розробників потужними механізмами спільної роботи, відстеження змін у коді та організації розподілених проектів.

Основні функціональні можливості GitHub — центральним структурним компонентом платформи є репозиторії, які використовуються для зберігання та управління вихідним кодом. Користувачі мають можливість створювати як публічні, так і приватні репозиторії, що дозволяє контролювати доступ до проектів. Управління версіями реалізується за допомогою Git, який забезпечує гнучке відстеження історії змін, розгалуження (branching) та злиття (merging) коду.

Модель колаборативної розробки — однією з найважливіших особливостей GitHub є механізм форкування (forking), який дає змогу користувачам створювати власні копії сторонніх репозиторіїв. Після внесення змін розробники можуть надсилати пропозиції щодо їх інтеграції у вихідний проект через пул-реквести (pull requests). Цей підхід є фундаментальним для розвитку відкритого програмного забезпечення (Open Source), оскільки сприяє глобальній співпраці між розробниками, незалежно від їхньої географічної локації.

3 ПРОЄКТУВАННЯ

3.1 Проєктування архітектури гри

Гра «Survive After The Apocalypse» реалізована за клієнт-серверною моделлю архітектури, що забезпечує оптимальний розподіл обчислювальних навантажень між клієнтом та сервером. Такий підхід дозволяє підтримувати стабільну синхронізацію ігрового процесу між гравцями, зменшувати затримки та забезпечувати гнучкість при масштабуванні.

Клієнтська частина створена на платформі Unity. Вона відповідає за відображення ігрового інтерфейсу, обробку введення користувачів та взаємодію з сервером за допомогою мережевого фреймворку Photon PUN 2. Гравці через клієнт створюють або приєднуються до кімнат, надсилають команди, відкривають картки та голосують. Unity-клієнт отримує оновлення про стан гри з сервера, що забезпечує єдину актуальну інформацію для всіх гравців у реальному часі.

Серверна частина функціонує на основі Photon Cloud — хмарного сервісу для організації мультиплеєрного взаємодії в іграх. Сервер приймає дані від усіх клієнтів, опрацьовує ігрову логіку (наприклад, роздачу карт, оновлення стану сесій, результати голосувань) та повертає синхронізовані оновлення всім учасникам гри. Це дозволяє забезпечити послідовність виконання ігрових подій та уникнути розбіжностей у відображенні станів гри на різних пристроях.

Для зберігання та обробки ігрових даних (картки професій, характеристик, подій тощо) використовується вбудована система ScriptableObject у Unity, яка дозволяє зберігати налаштовані об’єкти без використання зовнішніх баз даних. Це забезпечує високу продуктивність і мінімальні затримки під час доступу до ігрових даних.

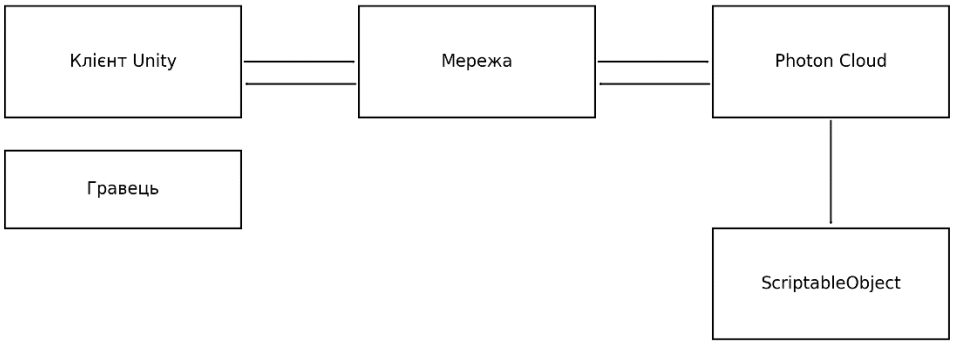


Рис. 3.1 Діаграма розгортання клієнт-серверної моделі

3.2 Архітектура клієнтської частини

Клієнтська частина гри «Survive After The Apocalypse» побудована за компонентною структурою, де кожен елемент відповідає за конкретний аспект функціонування ігрового процесу. Це дозволяє забезпечити модульність системи, полегшує підтримку та розширення проєкту.

Одним із ключових компонентів є менеджер кімнат, який відповідає за створення ігрових сесій, приєднання гравців до наявних кімнат за допомогою унікального коду, а також за організацію виходу гравців з ігрових кімнат. Цей модуль забезпечує логіку відкриття та закриття кімнат і перевірку доступності підключень до них.

Наступним важливим елементом є менеджер карт, який виконує функції динамічного завантаження префабів карток персонажів, їх властивостей і подальшого виведення на ігрове поле. У цьому модулі реалізовано взаємодію з системою ScriptableObject, що дозволяє оперативно підвантажувати дані карток із заздалегідь створених конфігураційних файлів проєкту.

За відстеження стану гравців, їх підключення та відключення від ігрової кімнати відповідає менеджер гравців. Він веде облік активних учасників, фіксує їх статуси та забезпечує актуальну синхронізацію цієї інформації з серверною частиною гри. Також цей модуль координує відображення інформації про гравців у користувацькому інтерфейсі.

Важливу роль у забезпеченні безперебійного ігрового процесу виконує менеджер фаз, який контролює перехід між фазами гри, відкриття карток, організацію обговорень та проведення голосувань. Саме цей компонент відслідковує, яка подія повинна відбутися наступною, керує послідовністю дій та надсилає команди на сервер для виконання відповідних процедур.

Окремо слід виділити інтерфейс користувача (UI), який реалізований через канваси та спеціалізовані префаби. Ця частина відповідає за відображення інформації на екрані гравця, взаємодію з елементами управління (кнопки, списки, повідомлення) та оперативне оновлення вмісту інтерфейсу в залежності від стану гри. Префабна система дозволяє легко налаштовувати вигляд і поведінку елементів інтерфейсу, що забезпечує гнучкість та зручність адаптації під різні сценарії гри.

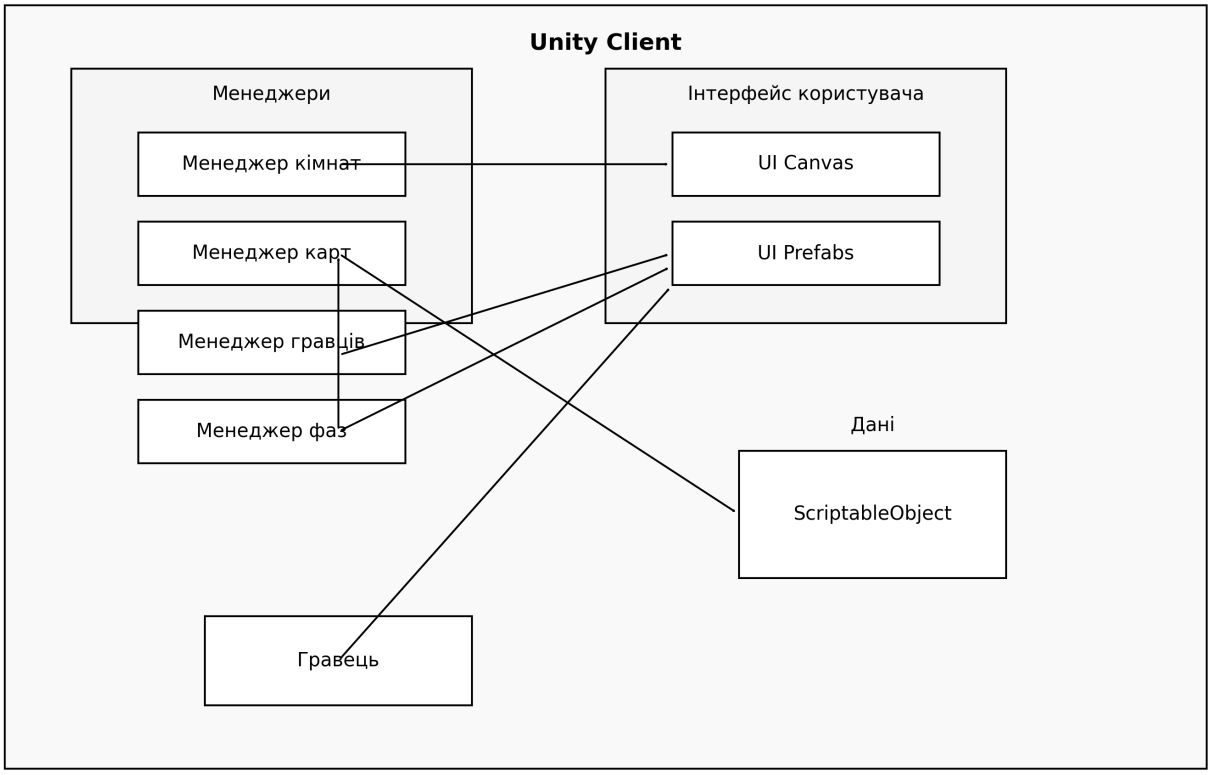


Рис. 3.2 Діаграма пакетів клієнтської частини

3.3 Архітектура серверної частини

Серверна частина гри «Survive After The Apocalypse» виконує функцію координаційного центру, який забезпечує стабільну взаємодію між усіма клієнтами, синхронізує ігрові події та контролює перебіг ігрових сесій. В основі серверної логіки лежить хмарний сервіс Photon Cloud, який обробляє усі підключення, події та повідомлення в реальному часі.

Сервер приймає підключення гравців до наявних кімнат, здійснює перевірку унікальних кодів і надає доступ до гри. Після формування складу учасників відбувається розподіл випадкових карток між гравцями. Процес роздачі організовано на стороні сервера задля забезпечення чесного та незалежного від клієнтів вибору карток.

Також сервер координує фази ігрового процесу: початок гри, розкриття карток гравців, обговорення, голосування за виключення, а також завершення сесії за умовами правил. Здійснення переходів між фазами супроводжується надсиланням відповідних повідомлень всім клієнтам, що дозволяє синхронізувати стан гри в реальному часі.

Передача актуального стану гри всім підключеним клієнтам відбувається через систему подій Photon Events, що дозволяє оперативно інформувати усіх гравців про зміну фаз, відкриття карт, результати голосувань та відключення гравців. Сервер стежить за узгодженістю даних та коректністю дій усіх учасників, запобігаючи виникненню конфліктних ситуацій.

Окрім цього, серверна частина забезпечує обробку виходу гравців із гри, контроль допустимої кількості підключень та коректне завершення ігрових сесій. У разі втрати зв’язку з гравцем, сервер або відображає його як неактивного, або завершує сесію відповідно до ігрових правил.

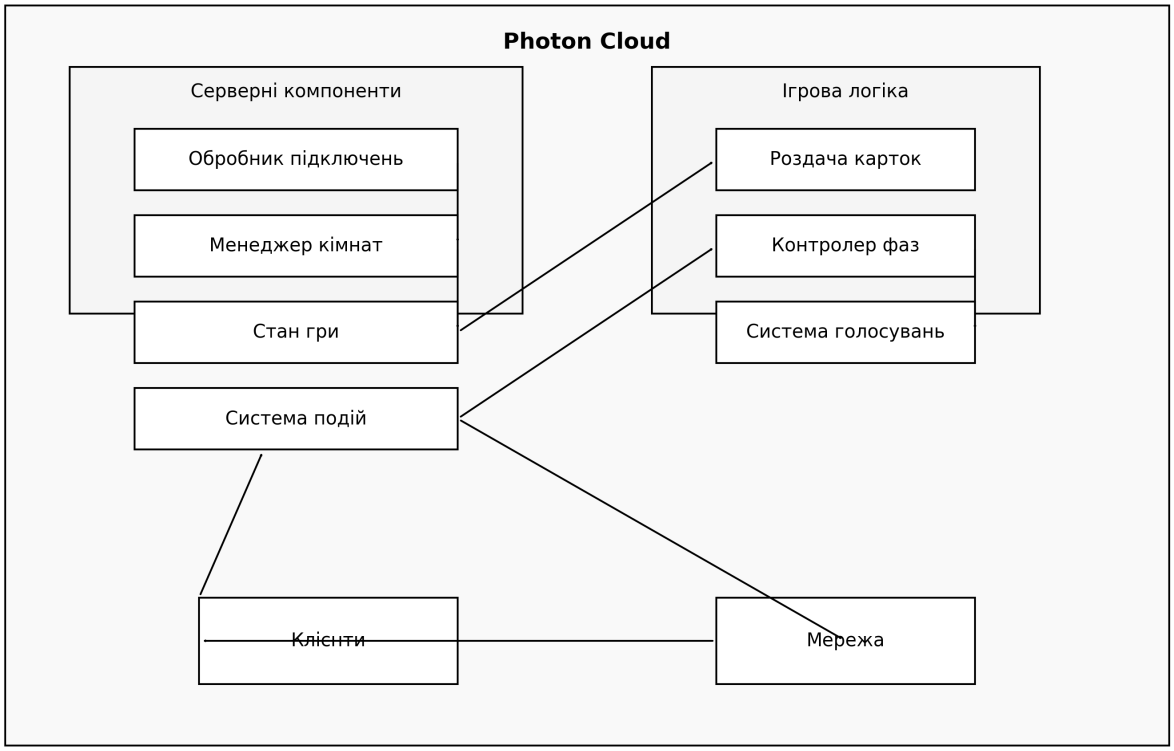


Рис. 3.3 Діаграма пакетів серверної частини

3.4 Організація зберігання ігрових даних

У проєкті «Survive After The Apocalypse» зберігання ігрових даних організовано без використання класичної системи управління базами даних. Замість традиційного підходу з реляційною структурою обрано сучасну гнучку систему на основі ScriptableObject, що є вбудованим механізмом ігрового рушія Unity. Такий підхід дозволяє значно оптимізувати роботу з даними, спрощує адміністрування контенту та забезпечує високу продуктивність під час виконання гри.

ScriptableObject являє собою спеціальний тип серіалізованих об’єктів, які зберігаються у вигляді окремих файлів у проєкті та не прив’язані до конкретної сцени чи ігрового об’єкта. Ці об’єкти дозволяють централізовано структурувати та зберігати інформацію про ігрові сутності, такі як картки персонажів, професій, станів здоров’я, фобій, особистих фактів і багажу. Використання цього механізму

дозволяє ефективно керувати ігровим контентом без залучення зовнішніх серверів чи баз даних, що особливо важливо для мультиплеєрних ігор, де швидкість доступу до даних має критичне значення.

В основі зберігання ігрових даних лежить структура, що передбачає поділ на окремі категорії. До кожної категорії відносяться об’єкти ScriptableObject, які містять набір параметрів, зокрема назву, опис, унікальний ідентифікатор, тип і додаткові властивості, що впливають на перебіг ігрової сесії. Таким чином, дані про картки зберігаються у структурованому вигляді, доступному для завантаження у будь-який момент під час виконання гри.

Усі об’єкти ScriptableObject об’єднуються у колекції — списки, які містять набір карток певного типу. Наприклад, окремі списки формуються для карток професій, особистих фактів, фобій чи станів здоров’я. Ці списки завантажуються під час запуску гри або створення нової сесії. Після завантаження дані карток використовуються протягом усієї гри без додаткового доступу до зовнішніх джерел інформації. Розподіл карток між гравцями та вибір випадкових подій відбувається безпосередньо з цих колекцій.

Додавання нових ігрових карток до проєкту здійснюється безпосередньо в редакторі Unity, де розробник має можливість створювати нові об’єкти ScriptableObject, наповнювати їх параметрами та включати до відповідних списків. Зчитування даних відбувається під час запуску проєкту або створення нової кімнати. Редагування властивостей ігрових об’єктів можливе лише на етапі розробки, що гарантує цілісність та незмінність ігрових даних під час виконання сесії. Видалення або заміна карток також виконується вручну в редакторі проєкту, що забезпечує повний контроль над наповненням ігрового контенту.

Щодо організації взаємозв’язків між даними, то замість класичних відносин типу «один до одного», «один до багатьох» чи «багато до багатьох», що характерні для реляційних баз даних, у грі використовуються списки ScriptableObject у класах керування сесією та системі менеджменту карток. Вибірка потрібної картки або

події здійснюється за допомогою унікальних ідентифікаторів та індексів у відповідних списках. Така модель дозволяє швидко опрацьовувати дані, не витрачаючи час на запити до серверів чи обробку складних запитів.

Використання ScriptableObject у цьому проєкті має низку переваг. По-перше, забезпечується висока продуктивність завдяки локальному доступу до даних без необхідності надсилати запити на сервер. По-друге, ресурси системи оптимізуються, оскільки дані завантажуються лише один раз на початку гри та використовуються протягом усієї сесії без повторного зчитування. По-третє, адміністрування карток та ігрових параметрів відбувається безпосередньо через редактор Unity, що значно спрощує підтримку проєкту. Окрім того, такий підхід унеможливлює зміну даних гравцями під час гри, що підвищує безпеку й цілісність ігрового процесу.

Серед інших переваг варто відзначити простоту реалізації та високу гнучкість. Додавання нових карток чи характеристик не потребує внесення змін у кодову базу або структуру зберігання даних. Розробнику достатньо створити новий об’єкт ScriptableObject і включити його до відповідного списку.

Таким чином, обрана архітектура організації ігрових даних дозволила забезпечити оптимальний баланс між продуктивністю, надійністю, безпекою та гнучкістю системи. Вона дозволяє ефективно керувати контентом гри, спрощує розробку і розширення проєкту, а також гарантує стабільну роботу мультиплеєрної частини без необхідності залучення сторонніх серверних рішень для зберігання даних.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Дизайн інтерфейсу

У проєкті «Survive After The Apocalypse» розробка дизайну інтерфейсу користувача виконувалася за допомогою графічного редактора Adobe Illustrator. Це професійне середовище для створення векторної графіки, яке дозволяє створювати адаптивні та якісні макети інтерфейсів для цифрових застосунків та ігор.

На етапі проектування інтерфейсу гри було розроблено комплекс макетів, що детально відображають структуру основних екранів та логіку взаємодії гравця з системою. Основна увага при цьому приділялася створенню зручного та естетично виваженого інтерфейсу, який би ефективно виконував свої функції при будь-якому сценарії використання.

Процес дизайну інтерфейсу включав такі етапи:

На початковому етапі було створено загальний концепт екранів головного меню, ігрової кімнати, карткових полів, та голосування. Для цього були визначені основні кольорові палітри, стиль іконографіки та загальна візуальна концепція.

Далі для кожного екрану було створено окремий артборд, на якому розміщувалися всі необхідні елементи: кнопки, текстові поля, іконки, картки персонажів, таймери та діалогові вікна. Розташування об’єктів на екрані враховувало особливості багатокористувацької взаємодії, необхідність швидкого доступу до основних функцій і зручності навігації під час гри.

Для карткових полів та меню створювалися окремі векторні префаби з урахуванням їх майбутньої інтеграції в Unity. Такий підхід дозволив швидко адаптувати елементи інтерфейсу без втрати якості при масштабуванні.

Окремо були підготовлені макети для анімації відкриття карток, спливаючих повідомлень про голосування та вибування гравців. Це дало змогу заздалегідь спланувати сценарії взаємодії користувача з грою та зменшити кількість правок на етапі інтеграції дизайну в рушій.

Завершальним етапом стала експортація підготовлених макетів у формати PNG для подальшого використання в Unity. Також було створено окрему документацію з описом призначення кожного екрану та взаємозв’язків між ними.

Рис. 4.1.1 Екранна форма головного меню

Рис. 4.1.2 Створенні карти в Adobe Illustrator

4.2 Реалізація клієнтської частини

Для реалізації клієнтської частини було обрано середовище Unity 2021 LTS як оптимальну платформу, що поєднує потужність, крос-платформеність та велику спільноту розробників. Використання мови C# дозволило реалізувати складну логіку гри, враховуючи її строгу типізацію та об'єктно-орієнтований підхід. Для мультиплеєрної складової була обрана бібліотека Photon PUN 2, яка надає готові рішення для синхронізації ігрового процесу між клієнтами через хмарну інфраструктуру.

Такий технологічний стек був обраний через:

–Високу продуктивність Unity для 2D/3D ігор

–Зручну інтеграцію Photon для мультиплеєрних рішень

–Можливість крос-платформової розробки

–Велику спільноту та доступність навчальних матеріалів

Архітектура проекту:

Проєкт був організований за принципами компонентно-орієнтованого дизайну з чіткою структурою каталогів:

– /Scripts - основна логіка гри

– /Prefabs - повторно використовувані об'єкти

– /Scenes - ігрові сцени

– /Resources - ScriptableObjects та інші асети

– /Cards – інформація про карти

На початковому етапі було створено новий проєкт в Unity Hub, визначено структуру каталогів для збереження сцен, скриптів, ресурсів, префабів, ScriptableObject та UI-елементів. Це дозволило організувати роботу над проєктом відповідно до сучасних практик розробки Unity-ігор.

Основними компонентами клієнтської частини є:

Менеджер кімнат — відповідає за створення, підключення та вихід з ігрових кімнат. Реалізує перевірку кількості гравців у кімнаті, дозволяє обрати ім’я гравця

та генерує унікальний код кімнати. Також забезпечує обробку виняткових ситуацій, наприклад, повторне підключення після втрати з’єднання або сповіщення про переповнення кімнати.

Менеджер гравців — забезпечує відстеження стану гравців у сесії, збереження їхніх імен, статусів, станів підключення та результатів голосування. Також синхронізує інформацію між усіма клієнтами за допомогою PhotonView. Крім цього, відповідає за передачу повідомлень про зміну статусу гравця, наприклад, коли гравець стає хостом або покидає гру.

Менеджер карт — виконує динамічне завантаження ігрових карток з даних, які зберігаються у вигляді ScriptableObject. Дані включають професії, фобії, стани здоров’я, додаткові факти та інші риси персонажів. Це дозволяє уникнути використання баз даних, оптимізувати продуктивність та спростити адміністрування проєкту. Додатково система карт підтримує легке розширення — нові типи карт можна додати без зміни основного коду, просто створивши новий ScriptableObject.

Менеджер фаз — управляє черговістю фаз гри: старт, роздача карток, відкриття характеристик, обговорення, голосування, вибування гравців, завершення гри. Забезпечує зміну станів гри у всіх клієнтів та передачу керування фазами між гравцями. При цьому використовуються налаштовані таймери, що дозволяє автоматично переходити між фазами, навіть якщо хтось із гравців зволікає.

Інтерфейс користувача (UI) реалізовано за допомогою Canvas та набору префабів, які відповідають за відображення ігрових карт, меню кімнат, діалогових вікон голосування, лобі, і сповіщень. Це дозволяє легко адаптувати інтерфейс під різні розміри екранів. Крім того, було впроваджено підтримку локалізації, що дає змогу швидко змінювати мову інтерфейсу в налаштуваннях гри.

Для підключення UI та ігрової логіки використовувалися механізми подій (Event System) та делегатів C#. Це дозволило забезпечити чисту архітектуру та мінімізувати зв’язність між різними частинами коду.

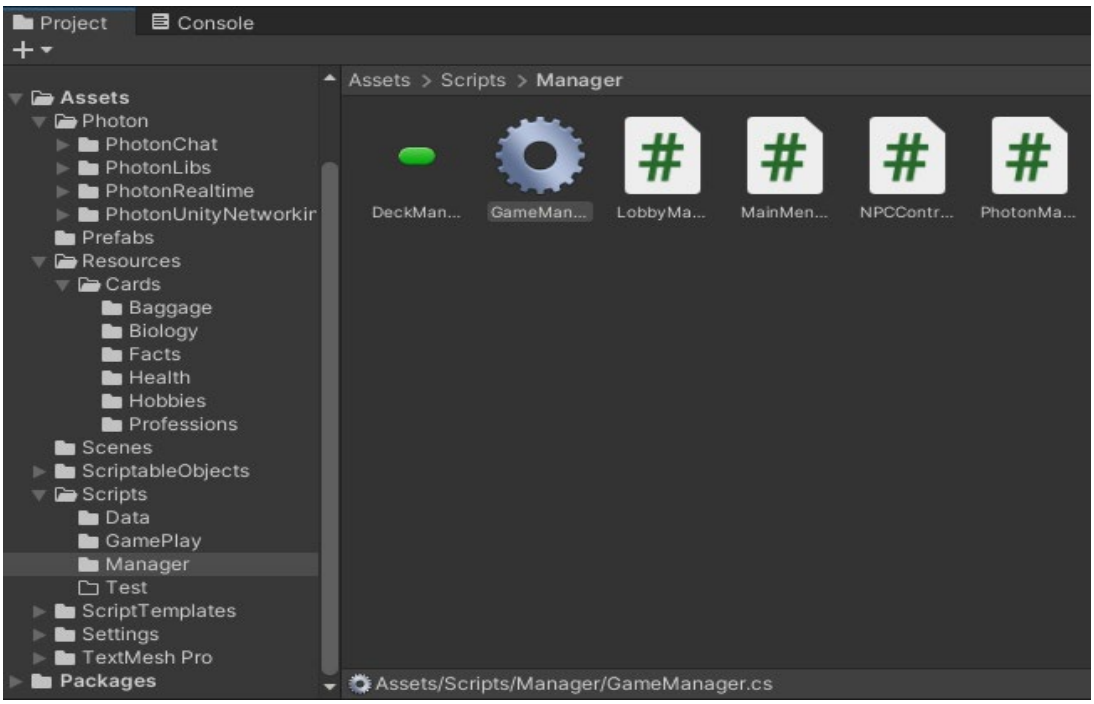


Рис 4.2.1 Приклад структури проєкту в Unity

Для ефективного управління ігровим процесом було створено низку спеціалізованих менеджерів, кожен з яких відповідає за певний аспект функціонування гри:

– GameManager - центральний координатор, який керує основними фазами ігрового процесу. Відповідає за послідовність станів гри (початок, роздача карт, обговорення, голосування, завершення), синхронізацію цих станів між усіма клієнтами та обробку переходів між фазами. компонента, декоратори та логіку компонента.

– LobbyManager - реалізує всю логіку роботи з ігровими кімнатами. Забезпечує створення нових кімнат, підключення до існуючих, перевірку доступності місць, генерацію унікальних ідентифікаторів кімнат та обробку подій, пов'язаних із зміною складу учасників.

– PlayerManager - відстежує стан усіх гравців у поточній сесії. Зберігає інформацію про імена учасників, їхні статуси, результати голосувань та інші важливі дані. Також відповідає за синхронізацію цієї інформації між клієнтами через Photon.

– CardManager - управляє всіма аспектами, пов'язаними з ігровими картами. Включає завантаження даних карт з ScriptableObjects, створення екземплярів карт

у грі, їх розподіл між гравцями, обробку взаємодії з картами та відображення їх станів.

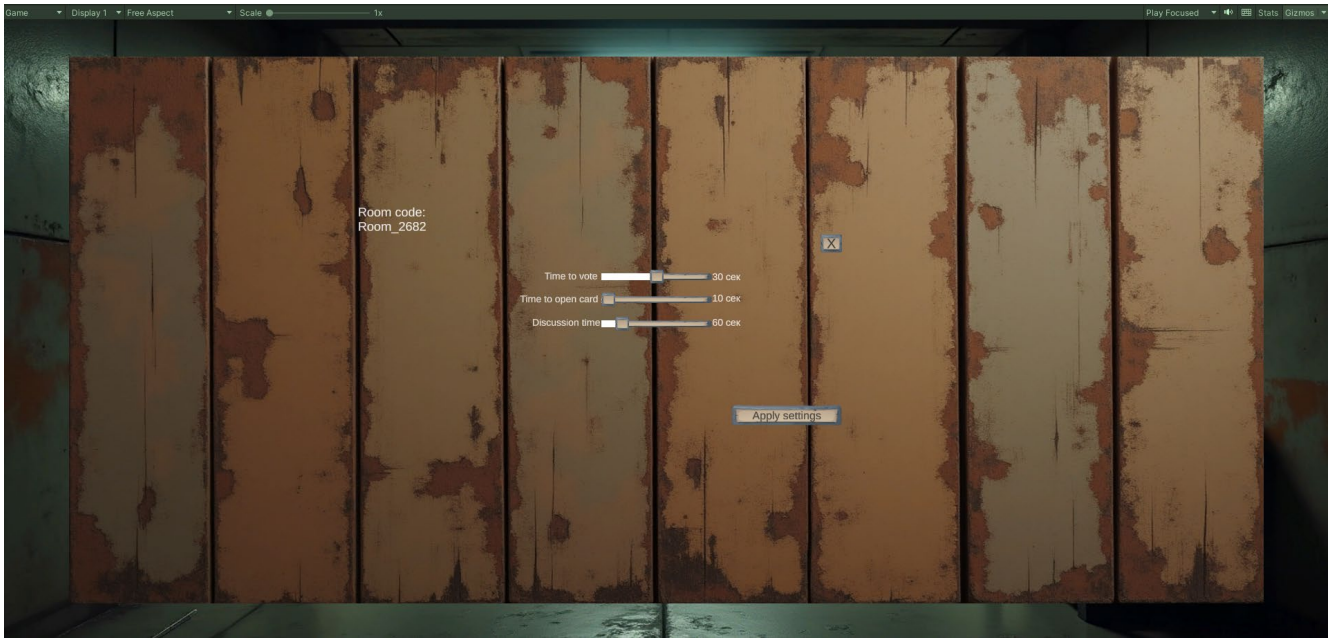


Рис 4.2.2 Приклад налаштувань гри в лоббі

Реалізація багатокористувацького режиму в грі "Survive After The Apocalypse" була побудована на базі Photon PUN 2, що забезпечило надійну та ефективну комунікацію між клієнтами. Основні аспекти мережевої взаємодії включають:

Створення та підключення до кімнат. Система дозволяє гравцям створювати нові ігрові кімнати або приєднуватися до вже існуючих. Кожна кімната має унікальний ідентифікатор, що генерується автоматично, та має налаштування:

– Максимальна кількість гравців

– Індивідуальний ID

– Вибір таймерів

Синхронізація ігрових станів

Для підтримки однакового стану гри на всіх клієнтах використовується компонент PhotonView, який дозволяє:

– Відстежувати зміни позицій об'єктів

– Синхронізувати стан ігрових елементів

– Передавати критичні події між клієнтами

– Керувати власністю об'єктів (object ownership)

Кожен синхронізований об'єкт має свій PhotonView ID для точної ідентифікації в мережі.

Віддалені виклики процедур (RPC)

Для критичних подій, що потребують миттєвої реакції всіх клієнтів, реалізовано RPC-механізми:

– Відправка повідомлень про початок/кінець гри

– Синхронізація результатів голосування

– Обробка спеціальних подій (наприклад, використання карток)

– Надання прав адміністратора кімнати

RPC виклики забезпечують надійну доставку критично важливих повідомлень.

Для мінімізації затримок було впроваджено:

– Компресію мережевих пакетів

– Пріоритезацію критичних повідомлень

– Обмеження частоти синхронізації неважливих об'єктів

– Кешування мережевих даних

– Прогнозування станів (client-side prediction)

Інтерфейсна частина гри "Survive After The Apocalypse" була реалізована з ґрунтовним підходом до забезпечення високого рівня юзабіліті та оптимальної продуктивності. Процес розробки UI включав кілька ключових етапів, кожен з яких сприяв створенню інтуїтивно зрозумілого та технічно досконалого інтерфейсу:

Canvas як основний контейнер

– Ієрархічна організація UI-елементів

– Розділення на окремі Canvas для різних екранів

– Оптимізація рендерингу через правильне налаштування компонентів

– Використання World Space Canvas для ігрових елементів

– Префаби UI-компонентів

– Стандартизовані елементи інтерфейсу (кнопки, панелі, повідомлення)

– Модульна система побудови складних інтерфейсів

– Параметризовані префаби для динамічного створення

– Оптимізація через повторне використання

Адаптивні макети

– Підтримка різних роздільних здатностей

– Гнучкі системи розміщення елементів (anchors, layout groups)

– Динамічне масштабування інтерфейсу

– Оптимізовані графичні активи для різних DPI

Після завершення розробки всіх основних систем гри наступає заключний етап - запуск та комплексне тестування проекту. Unity Editor надає потужний набір інструментів для виконання цієї задачі.

Нище наведено екранні форми декількох сторінок клієнтської частини гри.

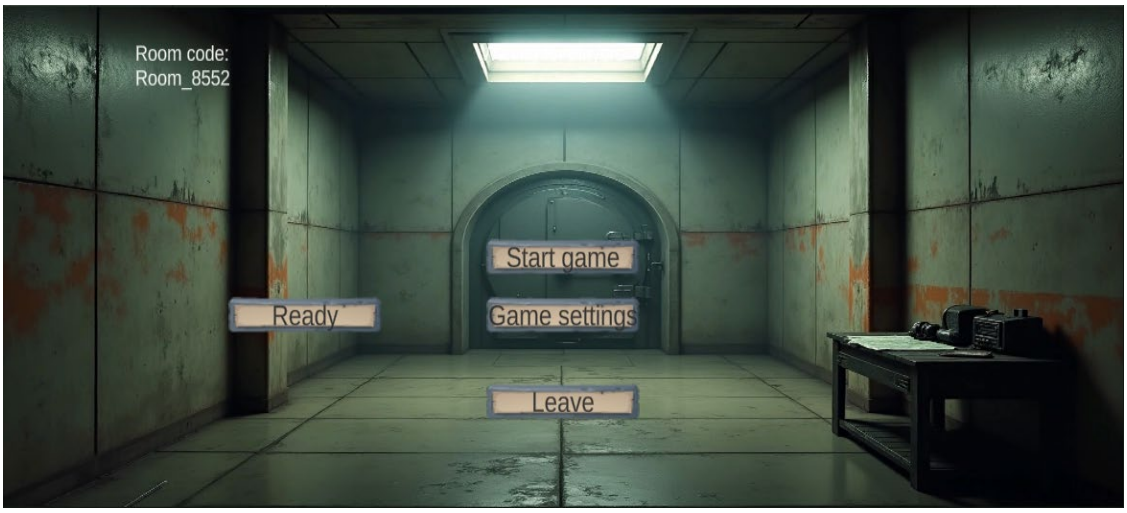


Рис. 4.2.3 Екранна форма ігрового лоббі

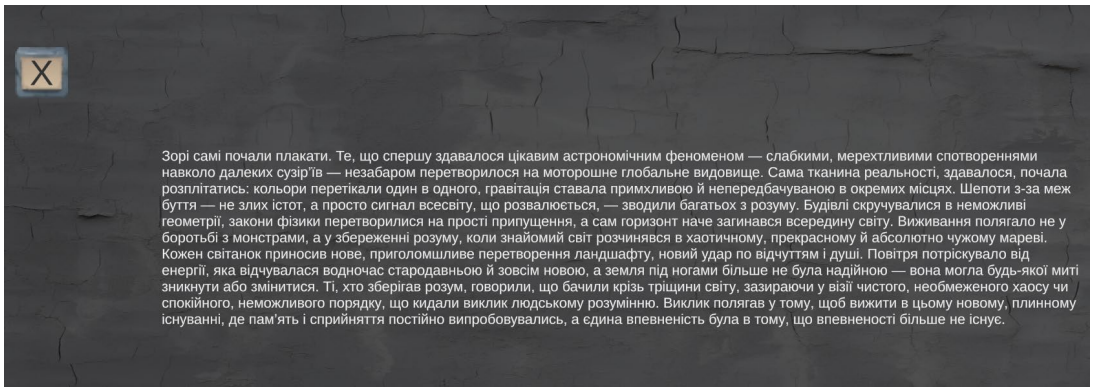


Рис 4.2.4 Панель з описом апокаліпсису

Рис. 4.2.5 Екранна форма ігрового процесу

4.3 Реалізація серверної частини

У проєкті «Survive After The Apocalypse» серверна частина реалізована за допомогою хмарної інфраструктури Photon Cloud та бібліотеки Photon PUN 2, інтегрованої безпосередньо в Unity. Це дозволяє обійти необхідність створення та налаштування окремих серверів або систем управління базами даних, забезпечуючи всю мережеву взаємодію між гравцями через хмарний сервер Photon.

Процес реалізації серверної частини складався з декількох етапів:

На початковому етапі було зареєстровано обліковий запис на платформі Photon Engine, створено проєкт та згенеровано унікальний Photon App ID. Цей ідентифікатор використовується для підключення клієнтів до ігрових сесій через хмарні сервери Photon.

У Unity встановлено та підключено пакет Photon PUN 2 з Asset Store. Далі у налаштуваннях PhotonServerSettings було задано параметри підключення до серверів, обмеження кількості гравців у кімнаті, таймаути очікування та режими відновлення сесії.

Основні функції серверної частини:

– Підключення гравців до ігрових кімнат. Гравець може створити кімнату або приєднатися до існуючої за унікальним кодом. Photon автоматично резервує слоти та сповіщає про приєднання інших гравців.

– Створення та розподіл карток гравцям. На початку гри адміністратор кімнати ініціює роздачу карток, які випадково обираються із заздалегідь підготовлених ScriptableObject. Розподіл карт синхронізується між усіма клієнтами через PhotonView і RPC-методи.

– Контроль фаз гри. Менеджер фаз відстежує стан гри та ініціює перехід до наступного етапу: роздача карток, відкриття характеристик, обговорення, голосування та визначення переможця. Зміни фаз відправляються всім клієнтам через мережу одночасно.

– Передача актуального стану гри між клієнтами. Ключові об’єкти, що потребують синхронізації (гравці, картки, голосування), мають компонент PhotonView, який забезпечує їх актуальний стан на всіх пристроях.

– Обробка відключення гравців. У разі виходу або втрати зв’язку з одним із гравців система автоматично видаляє його зі списку активних учасників. Гра продовжується без потреби перезавантаження для інших.

– Керування завершенням сесії. Після вибування необхідної кількості гравців система завершує гру, надсилаючи всім клієнтам команду про завершення сесії та відображення результатів.

Використання Photon Cloud дозволило забезпечити високу стабільність, мінімальний час відгуку та синхронізацію ігрових даних у режимі реального часу для всіх гравців. Такий підхід значно спростив розробку багатокористувацької архітектури гри, забезпечивши надійність та масштабованість проєкту «Survive After The Apocalypse».

Другим кроком під час реалізації серверної частини було створення системи обробки подій та мережевої синхронізації в межах Photon PUN 2. Замість використання окремих серверних додатків чи систем управління базами даних, усі механізми взаємодії між клієнтами реалізовані безпосередньо в Unity за допомогою Photon Cloud.

У Unity було створено окремий каталог /Scripts/Manager/PhotonNetwork, у якому розміщено скрипт для роботи з підключенням до серверу, управлінням кімнатами, обробкою статусів гравців та синхронізацією ігрових об’єктів.

Основні компоненти цієї структури:

– LobbyManager.cs — відповідає за створення, приєднання та вихід із ігрових кімнат. Визначає максимальну кількість гравців у сесії, генерує унікальні коди кімнат та контролює стан підключення користувачів.

– GameManager.cs — реалізує систему управління фазами гри: старт, роздача карт, відкриття характеристик, голосування, вибування гравців та завершення гри. Фази синхронізуються між усіма гравцями за допомогою RPC-методів Photon.

– Player.cs — містить логіку для зберігання і синхронізації даних про гравців: імена, стан підключення, активні статуси та результати голосування.

– DeckManager.cs — відповідає за генерацію випадкових карток із використанням ScriptableObject, їх відображення та синхронізацію між гравцями.

– PhotonNetwork.cs - обробляє події підключення, відключення гравців, виходу із сесії та надсилання актуального стану гри.

Всю мережеву взаємодію забезпечено компонентами PhotonView із прив’язаними ідентифікаторами для контролю стану ігрових об’єктів. Передача даних між клієнтами виконується через Photon RPC (Remote Procedure Call), що дозволяє викликати методи одночасно на всіх пристроях у сесії.

Таким чином, структура проєкту побудована на основі мережевих компонентів Photon і забезпечує повну синхронізацію ігрового процесу без використання класичних серверів чи баз даних.

Третім кроком було створенно реалізування системи зберігання ігрових даних. У проєкті «Survive After The Apocalypse» замість традиційної бази даних використовується структура ScriptableObject — спеціальний тип об’єктів у Unity, що дозволяє зберігати серіалізовані дані у вигляді окремих файлів у проєкті. Це рішення забезпечує централізоване, швидке та оптимізоване зберігання інформації про ігрові картки та сутності.

Для кожної категорії даних створено окремий клас ScriptableObject:

– CardData.cs — містить дані про конкретну картку: назву, опис, категорію (професія, фобія, хобі тощо) та параметри впливу на гру.

– CardDisplay_Interactive — зберігає список усіх доступних карток певного типу, дозволяючи виконувати вибірку або рандомізацію під час створення ігрової сесії.

Дані редагуються безпосередньо в Unity Editor через інспектор, що забезпечує зручність адміністрування контенту. Це дозволяє розробнику легко додавати, змінювати чи видаляти картки без зміни коду та без використання зовнішніх серверів чи баз даних.

Четвертим кроком під час реалізації серверної частини стало впровадження системи керування неігровими персонажами (NPC), які автоматично додаються до ігрових сесій у випадках, коли кількість реальних гравців є недостатньою для коректного функціонування ігрового процесу.

Це рішення дозволяє забезпечити стабільну роботу гри за будь-якого складу гравців та уникнути зривів сесій через нестачу учасників.

NPC реалізовані як повноцінні гравці, які мають усі необхідні ігрові атрибути: набір карток, ім’я, статус підключення та можливість брати участь у голосуваннях і відкритті характеристик. В основі логіки NPC закладено три різні типи поведінкових моделей:

– Агресивний (Aggressive NPC) — схильний до конфліктної поведінки, часто голосує проти гравців з сильними чи стратегічно важливими характеристиками.

– Дружній (Friendly NPC) — прагне підтримувати слабших гравців та ухиляється від конфліктів, голосуючи переважно за збереження нейтральних або корисних учасників.

– Лояльний (Loyal NPC) — приймає рішення, орієнтуючись на загальну вигідність для групи, враховуючи кількість уже відкритих карток, поточні фази гри та співвідношення професій і навичок серед гравців.

NPC інтегруються в гру на етапі створення кімнати, якщо кількість підключених користувачів не відповідає мінімальному чи оптимальному складу для початку сесії. Їхня взаємодія з іншими гравцями синхронізується через

мережеві виклики Photon RPC, а рішення про голосування чи відкриття карток приймаються на основі скриптів з попередньо визначеними шаблонами поведінки та випадкових факторів, що забезпечує варіативність та непередбачуваність ігрового процесу.

Реалізація NPC дозволила:

– Гарантувати повноцінну гру за нестачі гравців.

– Урізноманітнити ігрові сценарії та прийняті рішення.

– Забезпечити постійну динаміку гри без затримок на очікування додаткових користувачів.

Таким чином, інтеграція системи NPC стала важливим етапом реалізації багатокористувацької архітектури гри «Survive After The Apocalypse», що дозволяє підтримувати стабільну та цікаву ігрову сесію за будь-яких обставин.

П’ятий крок розробки серверної частини полягав у створенні системи обробників ігрових подій, що забезпечує синхронізацію станів гри між усіма учасниками сесії в режимі реального часу. Оскільки гра «Survive After The Apocalypse» є багатокористувацькою мережею, побудованою на технології Photon PUN 2, взаємодія між клієнтами та обмін ігровою інформацією здійснюється шляхом виклику віддалених процедур (RPC) та обміну подіями (RaiseEvent) через Photon Server.

На цьому етапі було реалізовано низку основних обробників, серед яких:

– Підключення та вихід гравця з кімнати — у момент входу гравець реєструється у списку активних клієнтів, отримує унікальний ідентифікатор, набір ігрових карт та статус. У разі виходу — інші гравці отримують сповіщення, а система автоматично адаптує ігрову сесію.

– Передача та розподіл карток — за допомогою Photon RPC викликається функція розподілу випадкових карток між гравцями, яка відбувається одночасно на всіх клієнтах. Картки завантажуються із ScriptableObject та прив’язуються до відповідного гравця.

– Управління фазами гри — після завершення певної фази (відкриття характеристик, обговорення, голосування, вибуття гравця) система за допомогою

викликів PhotonNetwork.RaiseEvent повідомляє всіх учасників про перехід до наступного стану. Це дозволяє всім клієнтам миттєво оновити свої локальні дані та відобразити актуальний стан гри.

– Голосування та вибір кандидатів на вибуття — під час голосування кожен клієнт надсилає свій вибір за допомогою Photon RPC, після чого серверна частина обробляє голоси, підраховує результати та передає їх усім клієнтам для синхронного відображення результатів.

– Управління NPC — у разі недостатньої кількості гравців система автоматично додає NPC з заданими типами поведінки. Поведінка NPC визначається алгоритмічно, на основі заздалегідь закладених шаблонів. NPC можуть голосувати, відкривати картки, брати участь у виборі гравців на вибуття та взаємодіяти з іншими учасниками гри відповідно до заданого стилю (агресивний, дружній, лояльний).

Останнім кроком стала збірка та тестування проєкту в середовищі Unity

Після завершення розробки основних систем клієнтської та серверної частин гри «Survive After The Apocalypse», впровадження мережевих обробників подій, системи NPC, менеджера фаз та інтеграції з Photon PUN 2, фінальним етапом стало тестування повної працездатності проєкту в середовищі Unity Editor та створення збірки для фінального тестування в реальних умовах.

На цьому етапі було виконано:

– Перевірку коректності взаємодії між усіма клієнтами через Photon Cloud. Проводилися тестові сесії гри з різною кількістю гравців для виявлення можливих помилок синхронізації та затримок оновлення станів.

– Тестування поведінки NPC у сценаріях неповних ігрових лобі. Перевірялися всі три моделі поведінки (агресивний, дружній, лояльний) на адекватність голосування, відкриття карт та прийняття рішень.

– Налагодження UI для різних роздільних здатностей екранів та форматів відображення. Особливу увагу приділили відображенню карток персонажів, меню голосування, повідомлень про зміну фаз гри та фінального екрана результатів.

– Оптимізація завантаження ресурсів. Перевірялися часи завантаження сцен, підвантаження карток з ScriptableObject та час відгуку на ігрові події, зокрема на дії типу голосування, підключення нового гравця та завершення сесії.

– Створення фінальної збірки гри за допомогою інструментів Build Settings у Unity. Збірку було протестовано на різних пристроях для виявлення можливих проблем продуктивності та сумісності.

Цей етап дозволив остаточно переконатися у працездатності всіх механік гри, стабільності мережевого з’єднання, коректності синхронізації даних між гравцями та

NPC, а також у відсутності критичних помилок у клієнтській та серверній частинах проєкту. Після завершення внутрішнього тестування проєкт отримав статус готового до демонстрації.

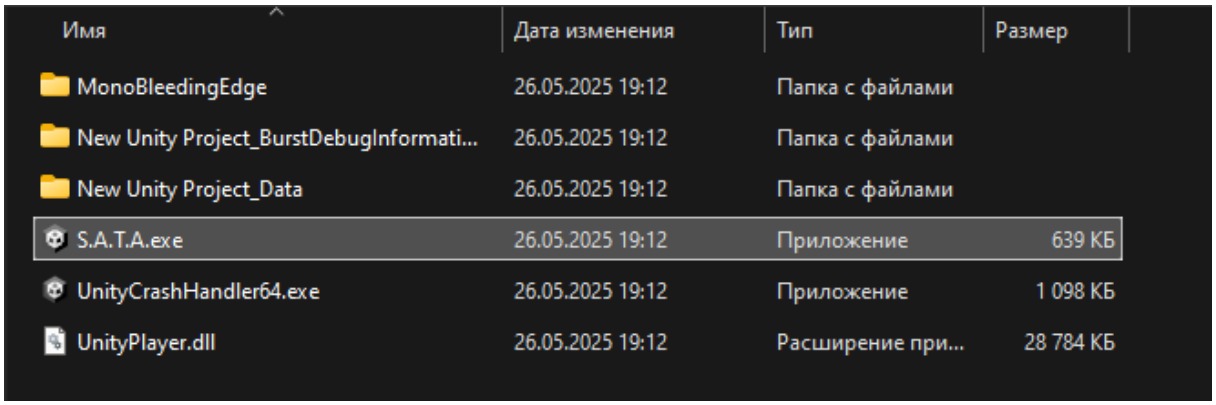


Рис 4.3 Успішно створений білд гри

4.4 Завантаження проєкту на GitHub

Після завершення основної реалізації клієнтської та серверної частини гри «Survive After The Apocalypse», важливим етапом стало розміщення всього проєкту у віддаленому репозиторії для зручності командної роботи, збереження резервних копій та подальшої демонстрації результатів розробки. Для цього було використано сервіс GitHub, який є популярною платформою для розміщення програмного коду, зокрема проєктів на Unity.

Процес розміщення проєкту складався з наступних кроків:

У кореневій директорії проєкту в Unity для ініціалізації репозиторію було виконано команду: git init

Це дозволило створити локальний репозиторій, що відстежує зміни у проєкті.

Щоб пов’язати локальний репозиторій із віддаленим на GitHub, було додано адресу існуючого репозиторію: git remote add origin

https://github.com/vassab/SurviveAfterTheApocalypse

Усі файли та каталоги проєкту були додані до репозиторію командою:

– git add .

Було створено коміт з описом реалізованого функціоналу

– git commit -m "Initial multiplayer version with Photon integration "

Останнім етапом стало завантаження проєкту у віддалений репозиторій за допомогою команди:

– git push -u origin master

Це забезпечило доступність проєкту для подальшого тестування, оновлення та презентації.

Таким чином, зберігання проєкту у віддаленому репозиторії дозволило впорядкувати структуру проєкту, налагодити контроль версій і забезпечити можливість відновлення або перегляду попередніх версій гри на будь-якому етапі розробки.

4.5 Тестування застосунку

Одним із ключових етапів у процесі розробки програмного забезпечення є тестування, яке дозволяє виявити помилки, недоліки в логіці, а також перевірити стабільність та відповідність функціональних і нефункціональних вимог. Для гри «Survive After The Apocalypse» особливу увагу було приділено інтеграційному, функціональному та інтерактивному (ігровому) тестуванню. Враховуючи мережевий характер проєкту, що працює на основі Photon PUN 2 та Unity, класичні підходи до модульного тестування виявилися менш доцільними. Основне

навантаження припадало на перевірку взаємодії між різними модулями гри та їхню стабільну роботу в умовах реального часу.

Інтеграційне тестування є важливим етапом для будь-якої багатокомпонентної системи. Його суть полягає у перевірці правильності взаємодії декількох незалежних модулів, які об’єднуються у єдину систему. В рамках даного проєкту було проведено інтеграційне тестування наступних підсистем:

– Менеджера кімнат, який відповідає за створення та підключення до ігрових лобі.

– Менеджера гравців, що забезпечує реєстрацію, збереження станів та синхронізацію інформації про гравців.

– Менеджера фаз, що контролює послідовність ігрового процесу: роздача карт, відкриття характеристик, обговорення, голосування та завершення гри.

– Системи голосування та синхронізації ігрових подій, яка забезпечує одночасне оновлення станів гри на всіх клієнтах.

Під час тестування перевірялися правильність розподілу карт між гравцями, коректність переходів між фазами, своєчасне оновлення ігрового інтерфейсу та відсутність конфліктів при підключенні, виході чи відключенні гравців.

Окремий акцент було зроблено на тестуванні стабільності гри у ситуаціях втрати мережевого підключення, відключення хоста, зміни кількості активних гравців та автоматичне додавання NPC.

Додавання NPC як частина тестування ігрових сценаріїв

Важливим нововведенням у грі стало впровадження NPC (персонажів, керованих системою), які автоматично додаються до кімнати у разі недостатньої кількості реальних гравців для старту сесії. Було реалізовано три типи NPC:

– Агресивний — активно голосує проти інших гравців, намагаючись залишити у грі найсильніших.

– Лояльний — підтримує більшість, уникає конфліктів.

– Дружній — обирає тих, хто має менше негативних характеристик, орієнтуючись на загальне благо.

Під час інтерактивного тестування NPC брали участь у всіх фазах гри: отримували випадкові картки, відкривали характеристики, брали участь в обговореннях (емулюючи репліки), голосували та могли бути вигнаними за підсумками раунду.

Було протестовано сценарії, коли у кімнаті грало від одного до дев’яти реальних гравців, а відсутні місця автоматично займали NPC. Усі ігрові події синхронізувалися без затримок, а час відгуку системи при взаємодії NPC не перевищував 1,5 секунд.

Цей тип тестування полягав у запуску повноцінних ігрових сесій з кількома гравцями та NPC у реальному часі. Під час тестування перевірялися такі параметри:

– Стабільність підключення до кімнат.

– Час роздачі карт і їх відображення.

– Коректність відкриття карт усіма гравцями.

– Правильність обрахунку голосів за вибування.

– Синхронізація завершення раундів та початку наступних.

– Обробка нештатних ситуацій: відключення хоста, випадковий вихід гравця.

Для перевірки було проведено 15 повноцінних ігрових сесій із залученням тестувальників та NPC. Усі тестові сесії успішно завершувалися без критичних збоїв або втрати даних. Спостереження за ходом гри здійснювалося через debug-консолі клієнтів та логування подій сервера Photon.

У результаті проведеного інтеграційного та інтерактивного тестування було виявлено і усунуто незначні недоліки, зокрема затримку переходу між фазами при підключенні до повної кімнати та несинхронізоване відображення голосів при дуже слабкому мережевому з’єднанні у одного з клієнтів. Після внесення виправлень усі системні компоненти функціонували відповідно до вимог.

За підсумками тестування встановлено, що:

– Час відгуку гри на ключові події (відкриття карт, початок голосування) не перевищує 2 секунд.

– Усі клієнти отримують актуальний стан гри синхронно.

– NPC поводяться відповідно до закладеної поведінкової моделі;

– Гра коректно обробляє ситуації втрати підключення та повторного входу гравця;

– Система стійка до навантажень при максимальній кількості гравців.

Таким чином, результати тестування підтвердили працездатність проєкту «Survive After The Apocalypse», його відповідність функціональним і нефункціональним вимогам, а також високу стабільність роботи в реальних ігрових умовах.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було виконано повний цикл розробки клієнт-серверного багатокористувацького застосунку «Survive After The Apocalypse» — цифрової карткової гри на виживання. Результати роботи підтверджують доцільність та ефективність обраних технічних і архітектурних рішень, а також демонструють відповідність реалізованого проєкту поставленим функціональним і нефункціональним вимогам.

У процесі дослідження та розробки було досягнуто наступних результатів:

– Проаналізовано предметну область: детально досліджено сучасний стан ігрових карткових проєктів з тематикою постапокаліпсису та моделей взаємодії гравців у таких системах. Проаналізовано ключові механіки ігрових аналогів визначено їхні сильні та слабкі сторони. На основі проведеного аналізу сформульовано перелік вимог до власної гри, враховуючи специфіку мультиплеєрного режиму, потребу в синхронізації станів гравців у реальному часі та можливість гри з NPC.

– Визначено цільову аудиторію застосунку: визначено, що основною аудиторією продукту є користувачі віком від 14 до 30 років, які цікавляться настільними та цифровими іграми з командною взаємодією, а також фанати ігор жанру виживання.

– Сформульованно вимоги до проєкту: визначено ключові функціональні вимоги, серед яких: реєстрація гравців, створення і приєднання до кімнат, роздача випадкових карт, покрокове відкриття характеристик, система голосування та управління фазами гри. До нефункціональних вимог віднесено стабільність гри при втраті гравця, синхронізацію станів у реальному часі, мінімальний час відгуку на дії користувача та підтримку до 10 гравців у сесії.

– Обґрунтовано вибір засобів реалізації: для реалізації проєкту було обрано ігровий рушій Unity, мову програмування C#, мережеву бібліотеку Photon PUN 2 для організації клієнт-серверної взаємодії, а для візуального оформлення карт і

інтерфейсів, графічний редактор Adobe Illustrator. Структурування ігрових даних реалізовано через ScriptableObject, що дозволило відмовитись від використання традиційної бази даних, забезпечивши швидкий доступ до інформації в пам’яті додатка.

– Розроблено архітектуру системи: сконструйовано архітектуру застосунку на основі клієнт-серверної моделі, яка забезпечує розподіл обчислювального навантаження між сервером Photon Cloud та клієнтами. Окремо спроєктовано клієнтську частину, яка складається з менеджерів кімнат, гравців, карт, фаз гри та системи UI, і забезпечує взаємодію з користувачем.

– Реалізовано прототип системи: створено повнофункціональний робочий прототип гри з можливістю підключення гравців, створення сесій, роздачі карт, переходу між фазами, проведення голосувань та визначення переможців. Впроваджено NPC, які автоматично додаються у разі нестачі реальних гравців.

– Проведено комплексне тестування: проведено інтеграційне, інтерактивне, ігрове тестування та тестування стійкості системи. Виявлені недоліки були усунені. За результатами тестів підтверджено стабільну роботу гри при максимальному навантаженні до 10 гравців у кімнаті та швидкість відгуку на дії гравців до 1,5 секунд. Перевірено коректну роботу механізмів голосування, вибування гравців, динамічної роздачі карт, синхронізації подій та обробки відключень клієнтів.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Бойко К.В., Ярошевський О.В., Аналіз жанрових особливостей соціальних симуляторів у контексті цифрової адаптації гри «Бункер». Метеріали: VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с 251.

2. Бойко К.В., Ярошевський О.В., Визначення вимог до карткової гри «Бункер». Метеріали: VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с 253.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unity Technologies. Unity Documentation. URL: https://docs.unity3d.com/ru/530/Manual/index.html (дата звернення: 28.03.2025).
2. Wizards Of The Coast. Magic.Wizards. URL: https://magic.wizards.com/ru/rules (дата звернення: 17.03.2025).
3. Schell J. The Art of Game Design : у 3 т. 2019. Т. 3. 652 с.
4. Fajar M., Cholid F. Design and Build Desktop-Based Educational GameSave The Wildlife "With Game Engine Construct 2 URL: https://www.ejournal.iocscience.org/index.php/mantik/article/view/722/480 (дата звернення 20.03.2025)
5. Microsoft. Visual Studio Documentation. URL: https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022 (дата звернення: 06.04.2025).
6. Adams E., Dormans J. Game Mechanics: Advanced Game Design. 2012. URL: https://books.google.pl/books?hl=ru&lr=&id=_Azio0txIdAC&oi=fnd&pg=PR7&dq=game+design+book&ots=RqkFYH8V0R&sig=vV4lGbbxwSohfo8XSvGbwFwba4I&redir_esc=y#v=onepage&q=game%20design%20book&f=false (дата звернення: 22.03.2025).
7. Sloan K., Khagendra K. Unity Networking Fundamentals. 2022. URL: https://link.springer.com/book/10.1007/978-1-4842-7358-6.
8. Hocking J. Unity in Action: Multiplatform Game Development in C#. 3rd ed. New York: Manning Publications, 2022. 400 p.
9. Ferrone H. Learning C# by Developing Games with Unity. 6th ed. Birmingham: Packt Publishing, 2023. 528 p.
10. Thorn A. Mastering Unity Game Development with C#. 2nd ed. Birmingham: Packt Publishing, 2021. 654 p.
11. Glazer J., Madhav S. Multiplayer Game Programming: Architecting Networked Games. Boston: Addison-Wesley Professional, 2021. 432 p.
12. Photon Engine. Photon Unity Networking 2 (PUN 2) Documentation. 2023. URL:https://doc.photonengine.com/pun/current/getting-started/pun-intro (дата звернення: 15.04.2025).
13. Kyaw A., Peters C., Swe T. Unity Multiplayer Games. Birmingham: Packt Publishing, 2022. 386 p.

14. Pagulayan R., Steury K., Fulton B., Medlock M. User-Centered Design in Games. The Game Design Reader.

15. Rabin S. Game AI Pro 360: Guide to Character Behavior. Boca Raton: CRC Press, 2022. 275 p.

16. Convai. Create AI Characters in Unity that Act on Your Command. 2024. URL: https://convai.com/blog/adding-actions-for-ai-characters-in-unity(дата звернення: 15.05.2025).

17. Schultz C., Bryant R., Langdell T. Game Testing: All in One. 3rd ed. Dulles: Mercury Learning & Information, 2022. 560 p.

18. Rupp C. Game QA Testing Methodologies: A Practical Guide. London: Apress, 2023. 320 p.

19. Cosp O. How to make a digital card game. 2024. URL: https://oriolcosp.com/how-to-make-a-digital-card-game/ (дата звернення: 15.05.2025).

20. Millington I., Funge J. Artificial Intelligence for Games. 3rd ed. Boca Raton: CRC Press, 2023. 896 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка карткової гри "Survive After the Apocalypse"

Виконав студент 4 курсу
Групи ПД-43
Бойко Костянтин Вікторович
Керівник роботи
Ассистент кафедри ІПЗ Ярошевський Олександр Вікторович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - виявити ефективні підходи до організації ігрового процесу в умовах соціальної взаємодії гравців шляхом створення фазової моделі онлайн-гри на виживання в бункері, яка забезпечує послідовне відкриття карт, обговорення та голосування в мультиплеєрному середовищі
- **Об’єкт дослідження -** процес розробки мультиплеєрних карткових ігор з використанням сучасних технологій мережевої взаємодії
- **Предмет дослідження -** методи та засоби реалізації мережевої синхронізації ігрового процесу та взаємодії між гравцями в карткових іграх з використанням мережевої технології Photon

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз існуючих мультиплеєрних карткових ігор та визначити їх переваги і недоліки.
2. Розробити концепцію та правила карткової гри з урахуванням особливостей мережевої взаємодії.
3. Розробити механізми синхронізації ігрового стану між гравцями з використанням технології Photon.
4. Реалізувати систему управління ігровими сесіями та лобі для підключення гравців
5. Створити користувацький інтерфейс, що забезпечує зручну взаємодію з грою
6. Розробити систему обробки ігрових подій та взаємодії між гравцями.
7. Провести тестування розробленої гри на стабільність мережевої взаємодії та ігровий баланс

Концепт гри

Назва – «Survive After The Apocalypse ».
Жанр – карткова стратегія .
Цільова аудиторія – любителі карткових ігор, соціальних ігор у компанії, віком від 14 до 30 років.
Сетинг – світ після апокаліпсису , в ізольованому бункері з обмеженими ресурсами.
Короткий огляд ідеї гри – кожна гра починається з випадкової генерації карт шістьох різних типів, та випадковим апокаліпсисом. Гравці аналізують інформацію, ведуть дискусії та голосують, кого виключити з бункеру. Основна мета – залишитися останнім у бункері.
Ключові особливості – гра реалізована у цифровому форматі, що дозволяє гравцям без фізичної взаємодії грати з різних точок світу
Платформа – Windows

АНАЛІЗ АНАЛОГІВ

Критерій / Гра	Shelter	Настільна гра «Бункер»	Survive After the Apocalypse
Покрокове відкриття карт гравцями	+	+	+
Фаза обговорення між гравцями	+	+	+
Голосування за виключення гравця	+	+	+
Історія відкритих карт гравців	-	-	+
Повна гра через інтернет із лобі	+	-	+

5

USP

1. Гнучкі сценарії апокаліпсису – кожна гра має різний тип катастрофи (ядерна війна, зомбі -вірус, повстання штучного інтелекту)
2. Динамічний соціальний експеримент – гравці не тільки голосують, а мусять переконувати, розкривати секрети інших, гра на «психологію».
3. Реалізація гри у цифровому форматі, що відрізняє її від настільних аналогів.

6

Геймплей, ігрові механіки, ігровий цикл

Геймплей - гравець взаємодіє з ігровим світом через стратегічні обговорення, аналіз карт персонажів та колективне прийняття рішень. Кожен раунд вимагає оцінки ризиків, переконання інших гравців і голосування за виключення з бункера.
Ігрові механіки - гра включає генерацію випадкових карт із унікальними характеристиками персонажів та систему голосування.
Ігровий цикл – кожна партія складається з циклічних фаз: викидання карти, обговорення та аргументація позиції, голосування за виключення гравця. Цикл повторюється, доки не визначиться 2 переможця.

7

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:
1. Реєстрація гравця
2. Створення та приєднання до кімнати через унікальний код
3. Роздача випадкових карт кожному гравцю
4. Покрокове відкриття карт гравцями
5. Гравці можуть переглянути історію відкриття карт на попередніх етапах

Нефункціональні вимоги:
1. Стани гри повинні оновлюватися для всіх гравців майже одночасно
2. Час відгуку гри на дії не більше 1 - 2 секунд
3. Збереження стабільної роботи при втрати одного з гравців
4. Дані користувачів (імена та статуси) не мають змінюватися іншими гравцями
5. Підтримка до 10 гравців одночасно в одному лобі

8

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Unity

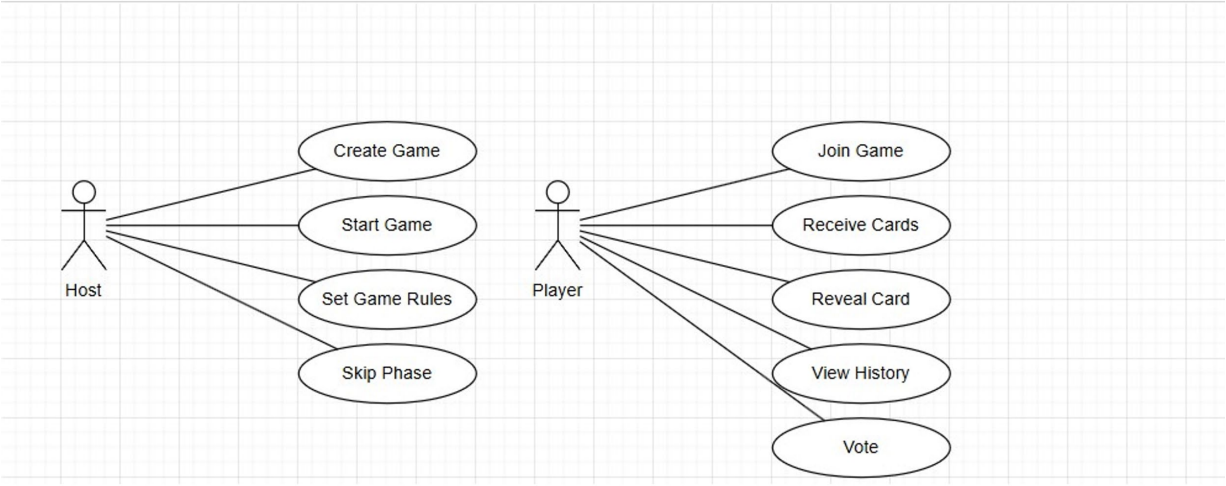
Visual Studio

Photon PUN-2 надає інструменти для підключення , синхронізації об'єктів і взаємодії між гравцями, а також для управління ігровим процесом на сервері, використовуючи платформу Photon Cloud

Adobe Illustrator

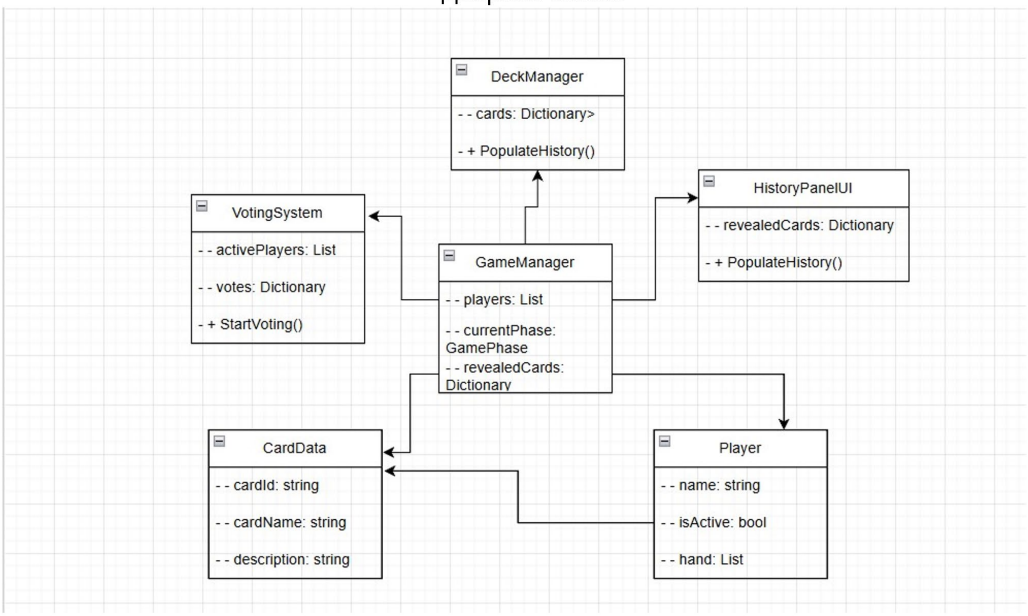
9

Діаграма прецедентів



10

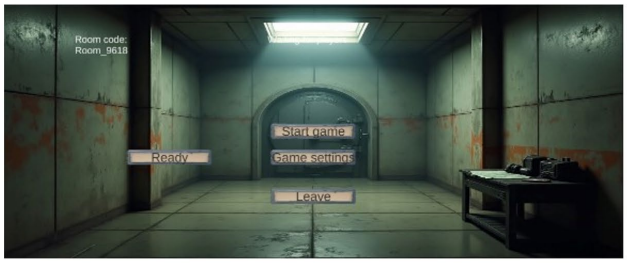
Діаграма класів



11

ЕКРАННІ ФОРМИ

Головне меню



Лобі

12

ЕКРАННІ ФОРМИ

Панель налаштувань гри

Ігровий процес

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Бойко К.В., Ярошевський О.В., Аналіз жанрових особливостей соціальних симуляторів у контексті цифрової адаптації гри «Бункер». Метеріали: VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с 251.

2. Бойко К.В., Ярошевський О.В., Визначення вимог до карткової гри «Бункер». Метеріали: VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с 253.

ВИСНОВКИ

1. Проведено аналіз наявних мультиплеєрних карткових ігор, що дозволило виявити сильні та слабкі сторони реалізації логіки взаємодії між гравцями, структури ходів та впливу гравців на результат. Це лягло в основу формування унікальної концепції гри.
2. Розроблено оригінальні правила гри, що передбачають послідовне відкриття карток, обговорення і голосування з поступовим вибуванням гравців.
3. Забезпечено надійне синхронне оновлення ігрового стану між клієнтами завдяки використанню Photon PUN 2. Реалізовано передачу стану, історії відкритих карт, ходу гравця та результатів голосування у реальному часі.
4. Створено систему управління ігровими сесіями з підтримкою хост-клієнтської моделі. Хост отримує можливість керування налаштуваннями гри та запуску сесії.
5. Створено централізовану логіку обробки ігрових подій: вибір карти, завершення ходу, запуск голосування, підрахунок голосів, вибування гравця та початок нового раунду.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections.Generic;
using System.Linq;
using Photon.Pun;

public class GameManager :
MonoBehaviourPunCallbacks
{
   public static GameManager Instance { get; private
set; }
   public bool IsVotingPhase { get; private set; }

   [Header("Game Settings")]
   [SerializeField]
   private List<CardType> startingCardTypes = new
List<CardType>
   {
      CardType.Profession,
      CardType.Biology,
      CardType.Health,
      CardType.Hobby,
      CardType.Baggage,
      CardType.Facts
   };

   [SerializeField] private int playersCount = 4;
   [SerializeField] private float timePerTurn = 60f;

   [Header("UI References")]
   [SerializeField] private CardDisplay_Interactive
cardPrefab;
   [SerializeField] private Transform cardsContainer;
   [SerializeField] private VotingSystem
votingSystem;
   [SerializeField] private GameObject endGamePanel;

   private List<Player> players = new List<Player>();
   private int currentPlayerIndex;
   private float currentTurnTime;
   private bool isGameActive;
   private List<CardDisplay_Interactive> activeCards
= new List<CardDisplay_Interactive>();

   private PhotonView photonView;
   private Dictionary<string, Dictionary<CardType,
List<CardData>>> revealedCardsHistory = new
Dictionary<string, Dictionary<CardType,
List<CardData>>>();

   [PunRPC]
   void SyncPlayerStatus(string playerName,
bool isActive)
   {
      Player player = players.Find(p => p.Name
== playerName);
      if (player != null)
      {
         player.IsActive = isActive;
      }
   }

   public void StartNextRound()
   {
      // Сбрасываем голоса всех игроков
      players.ForEach(p => p.ResetVotes());

      // Сбрасываем статусы ходов
      foreach (string playerName in
playerMoveStatus.Keys.ToList())
      {
         playerMoveStatus[playerName] = false;
      }

      // Сбрасываем флаги NPC
      var npcControllers =
FindObjectsOfType<NPCController>();
      foreach (var npc in npcControllers)
      {
         if (npc != null)
         {
            npc.ResetRoundFlags();
         }
      }

      // Синхронизируем с другими
клиентами
      if (photonView != null &&
PhotonNetwork.IsMasterClient)
      {

photonView.RPC("SyncResetMoveStatus",
RpcTarget.All);
      }

      // Проверяем количество активных
игроков
```

```
    private Dictionary<string, bool> playerMoveStatus
= new Dictionary<string, bool>();

    void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
            return;
        }
        Instance = this;

        photonView = GetComponent<PhotonView>();
        if (photonView == null)
        {
            Debug.LogError("GameManager: PhotonView
component is missing!");
        }
    }

    void Start()
    {
        InitializeGame();
    }

    public void InitializeGame()
    {
        if (startingCardTypes.Count == 0)
        {
            Debug.LogError("No starting card types
assigned!");
            return;
        }

        CreatePlayers();

        if (PhotonNetwork.IsMasterClient)
        {
            DealCards();
        }

        StartGame();
    }

    void CreatePlayers()
    {
        players.Clear();
        playerMoveStatus.Clear();

        // Только для Master Client
        if (!PhotonNetwork.IsMasterClient) return;

        // Создаём реальных игроков
        foreach (var photonPlayer in
PhotonNetwork.PlayerList)
        {
            string playerId = photonPlayer.UserId;
            int activePlayers = players.Count(p =>
p.IsActive);

            if (activePlayers >= 2)
            {
                // Начинаем новый раунд
                currentPlayerIndex = 0;
                IsVotingPhase = false;
                StartTurn(currentPlayerIndex);
            }
            else
            {
                CheckGameEnd();
            }
        }

        [PunRPC]
        void SyncResetMoveStatus()
        {
            foreach (string playerName in
playerMoveStatus.Keys.ToList())
            {
                playerMoveStatus[playerName] = false;
            }
        }

        void DealCards()
        {
            if (!PhotonNetwork.IsMasterClient)
return;

            if (DeckManager.Instance == null)
            {
                Debug.LogError("DeckManager
instance not found!");
                return;
            }

            Dictionary<string, List<string>>
playerCards = new Dictionary<string,
List<string>>();

            foreach (Player player in players)
            {
                List<string> cardIds = new
List<string>();
                foreach (CardType type in
startingCardTypes)
                {
                    CardData card =
DeckManager.Instance.GetRandomCard(type);
                    if (card != null)
```

```
            string playerName = photonPlayer.NickName;
            players.Add(new Player(playerId,
playerName));
            playerMoveStatus[playerName] = false;
        }

        // Создаём NPC (только если не хватает
игроков)
        int npcNeeded = playersCount -
PhotonNetwork.PlayerList.Length;
        for (int i = 0; i < npcNeeded; i++)
        {
            string npcId = $"NPC_{i + 1}";
            string npcName = $"NPC_{i + 1}";

            // Проверяем, не создан ли уже NPC с таким
именем
            if (!players.Any(p => p.Name == npcName))
            {
                players.Add(new Player(npcId, npcName));
                playerMoveStatus[npcName] = false;

                // Создаём GameObject для NPC
                if (GameObject.Find(npcName) == null)
                {
                    GameObject npcObj = new
GameObject(npcName);
                    NPCController npcController =
npcObj.AddComponent<NPCController>();
                    npcController.npcName = npcName;
                    npcController.Initialize(players.Last(),
npcId);
                    Debug.Log($"[DEBUG] NPC {npcName}
initialized with Player {players.Last().Name}");
                }
            }
        }
    }

    private HashSet<string>
playersWhoActedThisRound = new
HashSet<string>();

    public bool CanPlayerAct(string playerName)
    {
        return
!playersWhoActedThisRound.Contains(playerName);
    }

    public void RegisterPlayerAction(string
playerName)
    {
        playersWhoActedThisRound.Add(playerName);
    }

    public void ClearRoundActions()
    {
                {
                    cardIds.Add(card.cardId);
                }
            }
            playerCards[player.Name] = cardIds;
        }

        string serializedCards =
SerializePlayerCards(playerCards);
        photonView.RPC("SyncPlayerCards",
RpcTarget.All, serializedCards);
    }

    [PunRPC]
    void SyncPlayerCards(string
serializedCards)
    {
        try
        {
            Debug.Log($"Received cards data:
{serializedCards}");

            // Десериализация данных
            Dictionary<string, List<string>>
playerCards =
DeserializePlayerCards(serializedCards);
            if (playerCards == null)
            {
                Debug.LogError("Failed to
deserialize player cards");
                return;
            }

            foreach (var kvp in playerCards)
            {
                string playerName = kvp.Key;
                List<string> cardIds = kvp.Value;

                // Ищем игрока в списке
                Player player = players.Find(p =>
p.Name == playerName);

                // Если игрока нет — создаём и
добавляем
                if (player == null)
                {
                    Debug.LogWarning($"Player
{playerName} not found, creating new Player
instance");
                    player = new Player(playerName,
playerName); // id = name (если нет
отдельного id)
```

```
        playersWhoActedThisRound.Clear();
    }

    void StartGame()
    {
        isGameActive = true;
        currentPlayerIndex = 0;
        currentTurnTime = timePerTurn;
        StartTurn(currentPlayerIndex);
    }

    void StartTurn(int playerIndex)
    {
        if (!isGameActive || players.Count == 0)
        {
            Debug.LogWarning("Cannot start turn - game
not active or no players");
            return;
        }

        // Устанавливаем нового текущего игрока
        currentPlayerIndex = playerIndex;
        currentTurnTime = timePerTurn;

        Player currentPlayer = GetCurrentPlayer();
        if (currentPlayer == null)
        {
            Debug.LogError("Failed to get current
player!");
            return;
        }

        Debug.Log($"Starting turn for
{currentPlayer.Name} (Local:
{PhotonNetwork.NickName})");

        // Синхронизация хода для всех клиентов
        if (PhotonNetwork.IsMasterClient)
        {
            photonView.RPC("SyncCurrentTurn",
RpcTarget.All, currentPlayerIndex, currentTurnTime);
        }

        // Обновляем UI только для локального игрока
        if (currentPlayer.Name ==
PhotonNetwork.NickName)
        {
            Debug.Log("Displaying cards for local
player");
            DisplayPlayerHand(currentPlayer);
        }
        else
        {
            Debug.Log("Clearing cards container - not
local player's turn");
            ClearCardsContainer();
            players.Add(player);
        }

        // Очищаем старую руку
        player.ClearHand();

        // Добавляем карты
        foreach (string cardId in cardIds)
        {
            CardData card =
DeckManager.Instance?.GetCardById(cardId);
            if (card == null)
            {
                Debug.LogWarning($"Card
{cardId} not found in deck");
                continue;
            }
            player.AddCard(card);
        }

        Debug.Log($"Updated
{player.Name}'s hand with
{player.GetHand().Count} cards");
    }

    // Обновляем UI для локального
игрока, если его ход
    Player currentPlayer =
GetCurrentPlayer();
    if (currentPlayer != null &&
currentPlayer.Name ==
PhotonNetwork.NickName)
    {
        Debug.Log($"Updating UI for local
player: {currentPlayer.Name}");
        DisplayPlayerHand(currentPlayer);
    }
    else
    {
        Debug.Log($"Not local player's turn.
Current: {currentPlayer?.Name}, Local:
{PhotonNetwork.NickName}");
    }
  }
  catch (System.Exception e)
  {
    Debug.LogError($"Error in
SyncPlayerCards:
{e.Message}\n{e.StackTrace}");
  }
}
```

```
        // Если это NPC, запускаем его ход
        if (currentPlayer.Name.StartsWith("NPC_"))
        {
            NPCController npc =
FindNPCController(currentPlayer.Name);
            if (npc != null)
            {
                npc.StartNPCTurn();
            }
        }
    }
}

    [PunRPC]
    void SyncCardHistory(string playerName, string
cardId, int cardTypeInt)
    {
        CardType cardType = (CardType)cardTypeInt;
        CardData card =
DeckManager.Instance.GetCardById(cardId);

        if (card == null)
        {
            Debug.LogError($"[GameManager] Card with
ID {cardId} not found in SyncCardHistory!");
            return;
        }

        if
(!revealedCardsHistory.ContainsKey(playerName))
        {
            revealedCardsHistory[playerName] = new
Dictionary<CardType, List<CardData>>();
        }

        if
(!revealedCardsHistory[playerName].ContainsKey(car
dType))
        {
            revealedCardsHistory[playerName][cardType]
= new List<CardData>();
        }


revealedCardsHistory[playerName][cardType].Add(ca
rd);

        Debug.Log($"[GameManager] Synced card
{card.cardName} for {playerName}");
    }


    [PunRPC]
    void SyncCurrentTurn(int playerIndex, float
turnTime)
    {
        try
    string
SerializePlayerCards(Dictionary<string,
List<string>> playerCards)
    {
        string result = "";
        foreach (var kvp in playerCards)
        {
            result += kvp.Key + ":" +
string.Join(",", kvp.Value) + ";";
        }
        return result;
    }

    Dictionary<string, List<string>>
DeserializePlayerCards(string serialized)
    {
        var result = new Dictionary<string,
List<string>>();
        var entries = serialized.Split(';');

        foreach (var entry in entries)
        {
            if (string.IsNullOrEmpty(entry))
continue;

            var parts = entry.Split(':');
            if (parts.Length != 2) continue;

            string name = parts[0];
            var ids = parts[1].Split(',').Where(id =>
!string.IsNullOrEmpty(id)).ToList();
            result[name] = ids;
        }

        return result;
    }

    void DisplayPlayerHand(Player player)
    {
        ClearCardsContainer();

        if (player == null || cardPrefab == null ||
cardsContainer == null)
        {
            Debug.LogWarning("Invalid player or
UI references");
            return;
        }

        bool wasActive =
cardPrefab.gameObject.activeSelf;
        cardPrefab.gameObject.SetActive(true);
```

```
        {
            // Обновляем состояние на всех клиентах
            currentPlayerIndex = playerIndex;
            currentTurnTime = turnTime;

            Player currentPlayer = GetCurrentPlayer();
            if (currentPlayer == null)
            {
                Debug.LogError("SyncCurrentTurn: Current
player is null!");
                return;
            }

            Debug.Log($"[SYNC] Turn sync received.
Current player: {currentPlayer.Name} (Local:
{PhotonNetwork.NickName})");

            // Проверяем, является ли текущий игрок
локальным игроком
            if (currentPlayer.Name ==
PhotonNetwork.NickName)
            {
                Debug.Log("Displaying cards for local
player");
                DisplayPlayerHand(currentPlayer);
            }
            else
            {
                Debug.Log("Clearing UI - not local player's
turn");
                ClearCardsContainer();

                // Если это NPC, запускаем его ход
(только на Master Client)
                if (currentPlayer.Name.StartsWith("NPC_")
&& PhotonNetwork.IsMasterClient)
                {
                    NPCController npc =
FindNPCController(currentPlayer.Name);
                    if (npc != null)
                    {
                        Debug.Log($"Starting NPC turn:
{currentPlayer.Name}");
                        npc.StartNPCTurn();
                    }
                }
            }
        }
        catch (System.Exception e)
        {
            Debug.LogError($"Error in SyncCurrentTurn:
{e.Message}\n{e.StackTrace}");
        }
    }

    bool IsLocalPlayerTurn()
    {
        foreach (CardData card in
player.GetHand())
        {
            CardDisplay_Interactive cardUI =
Instantiate(cardPrefab, cardsContainer);
            cardUI.Initialize(card, player);
            activeCards.Add(cardUI);
        }


cardPrefab.gameObject.SetActive(wasActive);

LayoutRebuilder.ForceRebuildLayoutImmediat
e(cardsContainer as RectTransform);
    }

    private NPCController
FindNPCController(string npcName)
    {
        if (string.IsNullOrEmpty(npcName))
        {
            Debug.LogError("FindNPCController:
npcName is null or empty!");
            return null;
        }

        // Ищем среди всех NPC контроллеров
в сцене
        NPCController[] allNPCs =
FindObjectsOfType<NPCController>();
        foreach (NPCController npc in allNPCs)
        {
            if (npc.GetPlayer()?.Name ==
npcName)
            {
                Debug.Log($"Found NPC controller
for {npcName}");
                return npc;
            }
        }

        Debug.LogError($"NPC controller for
{npcName} not found!");
        return null;
    }

    /// <summary>
    /// Запускает ход для NPC
    /// </summary>
    public void StartNPCTurn(Player npcPlayer)
    {
```

```
        Player currentPlayer = GetCurrentPlayer();
        return currentPlayer != null &&
currentPlayer.Name == PhotonNetwork.NickName;
    }

    public void EndTurn()
    {
        if (!isGameActive) return;

        Debug.Log($"Ending turn for
{GetCurrentPlayer().Name}");

        if (GetCurrentPlayer() != null)
        {
            playerMoveStatus[GetCurrentPlayer().Name] =
true;

            if (photonView != null)
            {
                photonView.RPC("SyncPlayerMoveStatus",
RpcTarget.Others, GetCurrentPlayer().Name, true);
            }
        }

        bool allPlayersMoved = players.Where(p =>
p.IsActive).All(p => playerMoveStatus[p.Name]);
        Debug.Log($"[DEBUG] All players moved?
{allPlayersMoved}");

        if (allPlayersMoved)
        {
            StartVotingPhase();
        }
        else
        {
            int nextPlayer = (currentPlayerIndex + 1) %
players.Count;
            StartTurn(nextPlayer);
        }
        var currentPlayer = GetCurrentPlayer();
        if (currentPlayer == null)
        {
            Debug.LogError("GetCurrentPlayer() returned
null!");
            return;
        }
        Debug.Log($"[DEBUG] CurrentPlayer:
{currentPlayer.Name}");
    }

    [PunRPC]
    void SyncPlayerMoveStatus(string playerName,
bool hasMoved)
    {
        playerMoveStatus[playerName] = hasMoved;
    }

        if (!PhotonNetwork.IsMasterClient)
        {
            Debug.Log("Only master client can
start NPC turns");
            return;
        }

        if (npcPlayer == null)
        {
            Debug.LogError("StartNPCTurn:
npcPlayer is null!");
            return;
        }

        Debug.Log($"Starting NPC turn for
{npcPlayer.Name}");

        // Находим контроллер NPC
        NPCController npcController =
FindNPCController(npcPlayer.Name);
        if (npcController == null)
        {
            Debug.LogError($"Failed to find
controller for NPC {npcPlayer.Name}");
            return;
        }

        // Запускаем ход NPC
        npcController.StartNPCTurn();

        // Синхронизируем начало хода для
всех клиентов
        photonView.RPC("SyncNPCTurn",
RpcTarget.Others, npcPlayer.Name);
    }
    [PunRPC]
    private void SyncNPCTurn(string npcName)
    {
        // Этот RPC получают все клиенты,
кроме мастера (который уже обработал ход)
        Player npcPlayer = players.Find(p =>
p.Name == npcName);
        if (npcPlayer == null) return;

        Debug.Log($"Syncing NPC turn for
{npcName}");

        // Обновляем UI - очищаем карты (так
как это не наш ход)
        ClearCardsContainer();
    }
```

```
    void Update()
    {
        if (!isGameActive) return;

        currentTurnTime -= Time.deltaTime;
        if (currentTurnTime <= 0)
        {
            EndTurn();
        }
    }

    public void
OnCardSelected(CardDisplay_Interactive cardUI)
    {
        if (!isGameActive || cardUI == null) return;

        if (IsPlayerTurn(cardUI.GetOwner()))
        {
            Debug.Log($"Selected card:
{cardUI.GetCardData().cardName}");
            RegisterDiscardedCard(cardUI.GetOwner(),
cardUI.GetCardData());
        }
    }

    public void RegisterDiscardedCard(Player player,
CardData card)
    {

        Debug.Log($"[DEBUG] RegisterDiscardedCard
called for {player.Name} with card
{card.cardName}");
        if (player == null || card == null) return;

        string playerName = player.Name;

        if
(!revealedCardsHistory.ContainsKey(playerName))
        {
            revealedCardsHistory[playerName] = new
Dictionary<CardType, List<CardData>>();
        }

        if
(!revealedCardsHistory[playerName].ContainsKey(car
d.cardType))
        {

revealedCardsHistory[playerName][card.cardType] =
new List<CardData>();
        }


revealedCardsHistory[playerName][card.cardType].Ad
d(card);

    void ClearCardsContainer()
    {
        foreach (var card in activeCards)
        {
            if (card != null)
Destroy(card.gameObject);
        }
        activeCards.Clear();
    }

    public Player GetCurrentPlayer()
    {
        if (players.Count == 0 ||
currentPlayerIndex < 0 || currentPlayerIndex
>= players.Count)
            return null;
        return players[currentPlayerIndex];
    }

    public bool IsPlayerTurn(Player player)
    {
        return player != null && player.Name ==
GetCurrentPlayer().Name && player.IsActive;
    }

    void CheckGameEnd()
    {
        var activePlayers = players.Where(p =>
p.IsActive).ToList();
        if (activePlayers.Count <= 1)
        {

EndGame(activePlayers.FirstOrDefault());
        }
    }

    void EndGame(Player winner)
    {
        isGameActive = false;
        string result = winner != null ?
$"{winner.Name} wins!" : "Game ended in a
draw";
        Debug.Log(result);

        if (endGamePanel != null)
        {
            endGamePanel.SetActive(true);
            TMP_Text text =
endGamePanel.GetComponentInChildren<TM
P_Text>();
            if (text != null)
            {
```

```
        Debug.Log($"[GameManager] Registered {card.cardName} for {playerName}");


        if (PhotonNetwork.IsMasterClient)
        {
            photonView.RPC("SyncCardHistory", RpcTarget.All, playerName, card.cardId, (int)card.cardType);
        }
    }


    public Dictionary<Player, Dictionary<CardType, List<CardData>>> GetRevealedCardsHistory()
    {
        Dictionary<Player, Dictionary<CardType, List<CardData>>> result = new Dictionary<Player, Dictionary<CardType, List<CardData>>>();

        foreach (var kvp in revealedCardsHistory)
        {
            string playerName = kvp.Key;
            Player player = players.Find(p => p.Name == playerName);

            if (player != null)
            {
                result[player] = kvp.Value;
            }
        }

        return result;
    }



    private bool isVotingInProgress;

    public void StartVotingPhase()
    {
        if (isVotingInProgress) return;

        isVotingInProgress = true;
        IsVotingPhase = true;

        // Сбрасываем голоса
        votingSystem.ResetVoting();

}
                text.text = result;
            }
        }

        if (photonView != null && PhotonNetwork.IsMasterClient)
        {
            photonView.RPC("SyncGameEnd", RpcTarget.All, winner != null ? winner.Name : "");
        }
    }

    [PunRPC]
    void SyncGameEnd(string winnerName)
    {
        isGameActive = false;
        string result = !string.IsNullOrEmpty(winnerName) ? $"{winnerName} wins!" : "Game ended in a draw";

        if (endGamePanel != null)
        {
            endGamePanel.SetActive(true);
            TMP_Text text = endGamePanel.GetComponentInChildren<TMP_Text>();
            if (text != null)
            {
                text.text = result;
            }
        }
    }

    public List<Player> GetActivePlayers()
    {
        return players.Where(p => p.IsActive).ToList();
    }

    public List<Player> GetAllPlayers()
    {
        return new List<Player>(players);
    }
}
```