

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Мультиплеєр веб гра "Deck Duel" у жанрі
deckbuilding/battlerounds»**

Виконав студент групи ІПЗ-20-2
МАЦКЕВИЧ Роман Геннадійович

Керівник роботи:
ТУРЕНКО Єлизвета Єгорівна

Рецензент:
БАКАЛЮК Тетяна Анатоліївна

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

«ЗАТВЕРДЖУЮ»

Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **МАЦКЕВИЧ Роман Геннадійович**

Керівник роботи: **ТУРЕНКО Єлизавета Єгорівна**

Тема роботи: **«Мультиплеєр веб гра "Deck Duel" у жанрі deckbuilding/battlerounds»,**

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

Вихідні дані для роботи: **розробка змагальницької карткової онлайн гри, з елементами будівництва власних столів, та порівняння цих столів з іншими учасниками**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Туренко Є.Є.	27.02.2024р.	27.02.2024р.
2	Туренко Є.Є.	24.04.2024р.	24.04.2024р.
3	Туренко Є.Є.	02.05.2024р.	02.05.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного комплексу розробленої онлайн гри про створення столів з різних карток та етапу боїв цих столів між собою у різних учасників та пояснювальної записки. Пояснювальна записка до дипломної роботи містить 77 сторінок, 33 ілюстрації та 1 таблицю. Головною метою є створення онлайн гри, яка дозволить користувачам пограти у вільний час, залучити до цього своїх друзів, та визначити хто ж з гравців обирає кращу стратегію.

В роботі визначено ключові завдання, що пов'язані з розробкою гри, проведено аналіз існуючих програм, подібних до розроблюваної, а також наведено сценарії роботи програмного продукту. Представлено огляд загальної структури додатку, описано алгоритми взаємодії між окремими компонентами.

У рамках випускної роботи були проведені експерименти по оптимізації геймплею, в результаті яких були визначені оптимальні налаштування для більшості існуючих девайсів.

КЛЮЧОВІ СЛОВА: ГРА, ВЕБ, ОНЛАЙН, СИСТЕМА, МЕХАНІКА, ГЕЙМПЛЕЙ, РУШІЙ, СКРИПТ, КАРТКИ.

					ІПЗ.КР.Б – 121 – 24 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Мультиплеєр веб гра “Deck Duel” у жанрі deckbuilding/battlegrounds				Лім.	Арк.	Аркуші
Розроб.	Мацкевич Р.Г.										
Керівник	Туренко Є.Є.									3	77
									Житомирська політехніка, група ІПЗ-20-2		
Н. контр.											
Зав. каф.	Вакалюк Т.А.				Пояснювальна записка						

ABSTRACT

The bachelor's thesis consists of a software package for an online game about creating tables of different cards and the stage of battles between these tables among different participants and an explanatory note. The explanatory note to the thesis contains 77 pages, 33 illustrations, and 1 table. The main goal is to create an online game that will allow users to play in their free time, involve their friends, and determine which player chooses the best strategy.

The paper identifies the key tasks related to the development of the game, analyzes existing programs similar to the one under development, and provides scenarios for the operation of the software product. An overview of the general structure of the application is presented, and the algorithms of interaction between individual components are described.

As part of the graduation work, experiments were conducted to optimize the gameplay, as a result of which the optimal settings for or most existing devices.

KEYWORDS: GAME, WEB, ONLINE, SYSTEM, MECHANICS, GAMEPLAY, ENGINE, SCRIPT, CARDS.

		Мацкевич Р.Г.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....	8
1.1 Постановка задачі	8
1.2 Аналіз аналогів програмного продукту	10
1.3 Вибір архітектури системи.....	18
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	21
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ МУЛЬТИПЛЕЄР ВЕБ ГРИ «"Deck Duel" у жанрі deckbuilding/battlerounds»	23
2.1 Визначення варіантів використання та структури системи	23
2.2 Розробка алгоритмів та ігрових механік.....	29
2.3 Реалізація функціоналу додатку	31
Висновки до другого розділу	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ГРОЮ «Мультиплеєр веб гра "Deck Duel" у жанрі deckbuilding/battlerounds» ..	49
3.1 Порядок встановлення та налаштування параметрів додатку.....	49
3.2 Структура інтерфейсу додатку та порядок роботи	50
3.3 Тестування роботи додатку.....	58
Висновки до третього розділу	59
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	62
ДОДАТКИ	64
Додаток А.....	65
Додаток В.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення;

ОС – операційна система;

UI – інтерфейс користувача;

ЦП – центральний процесор;

ЗП – оперативна пам'ять;

AI – штучний інтелект;

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми полягає в історії розробки ігор, ще з давніх часів, люди почали розробляти різні ігри, та заробляти на цьому кошти, пізніше це перейшло в комп'ютерні технології та набрало ще більшу популярність. На даний момент більшість людей використовують браузер, для пошуку інформації та ігор. В браузерних іграх зазвичай не має сюжету та важких механік, в них легко розібратись, тому гравець швидко занурюється. Ця основа і буде покладена в нашу гру.

Метою даної роботи є розробка програми гри та реалізація механік онлайн гри в браузері. Встановлена мета обумовлює наступні завдання:

- Встановлення цільової аудиторії гравців.
- визначення архітектури та узагальненої структури системи;
- обґрунтування та вибір засобів реалізації системи;
- розробка систем для реалізації ігрових механік та зручного користування ними

Об'єктом дослідження є процес взаємодії користувачів із грою для браузерів.

Предметом дослідження є використання ключових механік та підходів, що практикуються у мобільних іграх та способи їх реалізації.

Розроблений додаток може бути використаний для гри з друзями.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Механіки мобільних ігор хоч і не є важкими та не вимагають навичок для розуміння гравцю. Проте є досить складними в реалізації, особливо якщо мова йде про онлайн гру. Мета цієї роботи полягає в розробці програми гри, яка відповідає всім вимогам користувачів, працює швидко, та використовує зрозумілий підхід під час програмування.

Необхідна реалізація таких функцій:

- 1) Аналіз проекту та збір вимог:
 - аналіз конкурентних додатків на ринку;
 - детальний опис ключових механік, з якими відбуватиметься взаємодія;
 - оцінювання майбутньої гри з точки зору гравця, що використовуватиме продукт.
- 2) Реалізація та впровадження системи:
 - реалізація двох мікросервісів;
 - реалізація клієнтської частини гри;
 - розробка системи для підключення гравців;
 - створення і конфігурація зв'язків між гравцями, обробкою вхідних даних;
 - розробка асинхронних таймерів для синхронізації користувачів.
- 3) Реалізація мікросервісу з вибору, придбання та дек-білдингу:
 - реалізувати стійке підключення гравців до мікросервісу;
 - реалізувати відправку сповіщень на клієнт;
 - створити обробник на пошук гри та реалізувати всередині розподілення користувачів по різних ігрових кімнатах;
 - втілити обробник на вибір героя, реалізувати випадкове отримання набору героїв для вибору у гравця;

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

- в середині вище вказаного, реалізувати сповіщення всіх гравців про даний етап гри, героїв та показники здоров'я всіх гравців;
 - створити обробник початку гри, та запуску таймеру після під'єднання останнього клієнту до кімнати(Таймер повинен бути синхронізований у всіх гравців);
 - в середині обробника реалізувати логіку виходу гравця, коли значення здоров'я його героя стало менше 0. Також реалізувати систему раундів (1, 2, ...);
 - в середині обробника реалізувати логіку отримання гравцем карт для вибору, отримання золота в залежності від раунду гри;
 - створити обробник переходу на наступний раунд, та скидання інформації про попередній раунд.(золото, вибір карт);
 - в середині реалізувати отримання пошкодження героєм персонажу у випадку програшу в стадії битви;
 - реалізувати обробник «заморозки», не оновлювати вибір карт після раунду;
 - створити обробник оновлення вибору карт;
 - втілити обробник придбання, виставлення карти на стіл, та продажі карти;
 - реалізувати обробник для унікальної властивості кожного з героїв;
 - реалізувати обробник відключення гравця.
- 4) Реалізація мікросервісу з обробки битви столів гравців
- реалізувати стійке підключення гравців до мікросервісу;
 - реалізувати відправку сповіщень на клієнт;
 - реалізувати підключення клієнтів та механіки по розподіленню за кімнатами;
 - реалізувати метод обрахунку наступного удару, позиції атакуючого та приймаючого удар;
 - реалізувати механіку завершення бою;
 - реалізувати обробник відключення гравця.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Турченко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

- 5) Реалізація клієнтської частини
 - розробити зручний інтерфейс;
 - розробити анімацію ударів;
 - підключити бібліотеки для сповіщень.

1.2 Аналіз аналогів програмного продукту

Наразі веб ігри користуються чималою популярністю через примітивність механік, простоту графіки та можливість грати без скачування, для них потрібен лише браузер та гарне інтернет з'єднання. Переглянемо декілька з найпопулярніших.

agar.io

Подана гра є реалізацією стратегічного онлайн шутера із простим керуванням та легкістю в освоєнні. У Agar.io не передбачена система бою в традиційному розумінні, однак в ній використовується механіка збору ресурсів: достатньо підвести свій "клітинний" персонаж до маленьких точок (агару), і вони автоматично будуть поглинені, збільшуючи розмір вашого персонажа (рис. 1.1-).

Посилання на гру: <https://agar.io/#ffa>



Рис. 1.1. – Agar.io – старт

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.Є.				10
Змн.	Арк.	№ докум.	Підпис	Дата		



Рис. 1.2. – Agar.io – гра

Переваги:

- Простота та доступність: Гра має дуже простий інтерфейс і механіку, що дозволяє швидко зрозуміти, як грати, навіть новачкам. Вона доступна для гри прямо в браузері без необхідності завантаження чи встановлення додаткового програмного забезпечення;
- Мультиплеєр: Agar.io підтримує багатокористувацький режим, де гравці з усього світу можуть змагатися один з одним у реальному часі, що додає грі конкурентного елементу і робить її захоплюючою;
- Динамічний геймплей: Гра має динамічний геймплей з постійно змінюваними ситуаціями на карті. Гравці повинні швидко адаптуватися до нових умов, що тримає їх у напрузі і забезпечує високий рівень реіграбельності;
- Безкоштовність: Agar.io можна грати безкоштовно, що робить її доступною для широкого кола користувачів. Можливість грати без фінансових витрат є великою перевагою;
- Простір для стратегії: Незважаючи на простоту гри, вона пропонує багато стратегічних можливостей. Гравці можуть використовувати різні

тактики для досягнення успіху, що додає глибини і робить гру більш захоплюючою.

Недоліки:

- Проблеми з балансом: У грі можуть виникати ситуації, коли новачки стикаються з більш досвідченими гравцями, що має значну перевагу у розмірах клітин. Це може бути фруструючим для нових гравців і робить гру менш справедливою;
- Відсутність різноманітності: Хоча гра має простий і привабливий геймплей, вона може стати одноманітною після тривалого часу гри. Відсутність нових елементів або значних оновлень може призвести до швидкого зниження інтересу;
- Чіти та боти: У грі часто зустрічаються чітери та боти, які можуть порушувати баланс і справедливість гри. Це знижує задоволення від гри і може відштовхнути чесних гравців;
- Зупинка розвитку: Вже досить багато часу гра ніяк не розвивається, та не отримує нових оновлень

Catan

Наступна гра є браузерною адаптацією класичної настільної гри "Settlers of Catan", де гравці будують поселення, дороги і міста на острові Катан, використовуючи ресурси, отримані з різних гексів. Гра включає в себе елементи будівництва, торгівлі та стратегії, з випадковим генеруванням острова для кожної партії, що забезпечує унікальний ігровий досвід. Гравці можуть торгувати ресурсами, купувати картки розвитку, захищатися від розбійників та змагатися в режимі мультиплеєра або одиночної гри. Catan Universe підтримує сучасну графіку, інтуїтивний інтерфейс і доступна на різних платформах, включаючи мобільні пристрої, що дозволяє грати будь-де і будь-коли (рис. 1.3-1.4).

Посилання на гру: <https://catanuniverse.com/en/>

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Туренко Є.Є.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

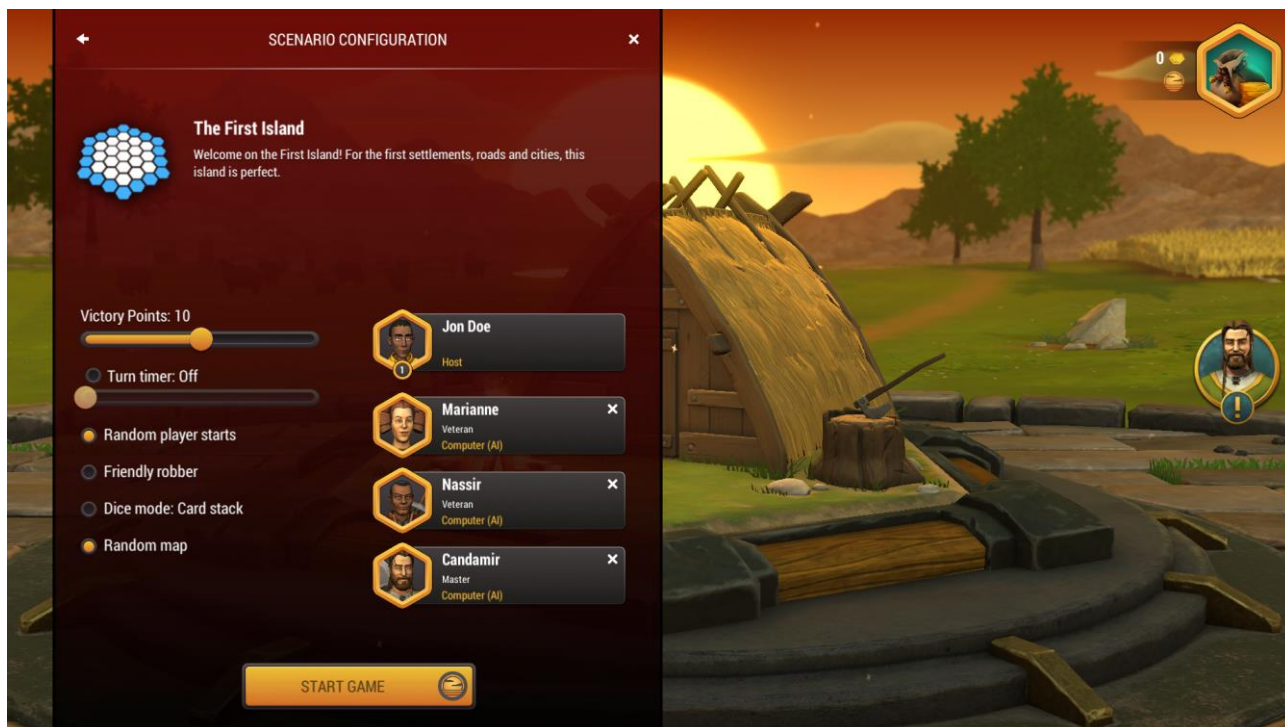


Рис. 1.3 – Login



Рис. 1.4. – Catan

Переваги:

- завдяки випадковому генеруванню острова кожна партія унікальна, що забезпечує високу реіграбельність і постійний інтерес до гри;

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

- механіка торгівлі між гравцями додає стратегії і взаємодії, дозволяючи будувати союзи та проводити вигідні обміни, що підвищує соціальний аспект гри;
- гра доступна на ПК та мобільних пристроях, що дозволяє грати в будь-який час і будь-де, забезпечуючи гнучкість для гравців;
- Catan Universe має сучасну, деталізовану графіку та інтуїтивно зрозумілий інтерфейс, що робить гру візуально привабливою та зручною у використанні;
- гра підтримує одиночний режим, мультиплеєр і турніри, що дозволяє гравцям вибирати, як вони хочуть грати, будь то проти AI чи реальних людей.

Недоліки:

- гра містить внутрішньоігрові покупки, що може створити дисбаланс між гравцями, які купують додаткові ресурси, та тими, хто грає без витрат;
- для гри потрібне стабільне інтернет-з'єднання, що може бути незручно для гравців з обмеженим доступом до мережі або з нестабільним підключенням;
- партії можуть тривати досить довго, що може бути незручним для гравців, які віддають перевагу коротким ігровим сесіям;
- щоб досягти успіху в грі, потрібно присвячувати їй багато часу, що може бути проблематично для гравців із насиченим графіком;
- гра іноді стикається з технічними проблемами, такими як збої або лаги, що може негативно вплинути на ігровий досвід.

The Duelyst

Duelyst це унікальна стратегічна карточна гра, яка відзначається комбінацією колекціонування карток і тактичних битв на дошці. Гравці складають свої колоди з різноманітних карток, кожна з яких представляє різноманітних персонажів з унікальними властивостями і способностями. Головна мета полягає в розміщенні і використанні цих карт для досягнення стратегічних переваг на бойовій дошці. Грається в режимі одиночних матчів

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

або у форматі турнірів проти інших гравців. Відмінність Duelyst полягає в тому, що вона поєднує стратегію розміщення і мобільності персонажів на дошці зі змагальністю колекціонування і елементами RNG (випадкових чисел), створюючи напружені та варіативні ситуації на кожному ході. Гра має візуально привабливий стиль і ретро-футуристичну графіку, що додає їй власного характеру серед інших карткових ігор жанру (рис. 1.5-1.6).

Посилання на гру: <https://play.duelyst2.com/>

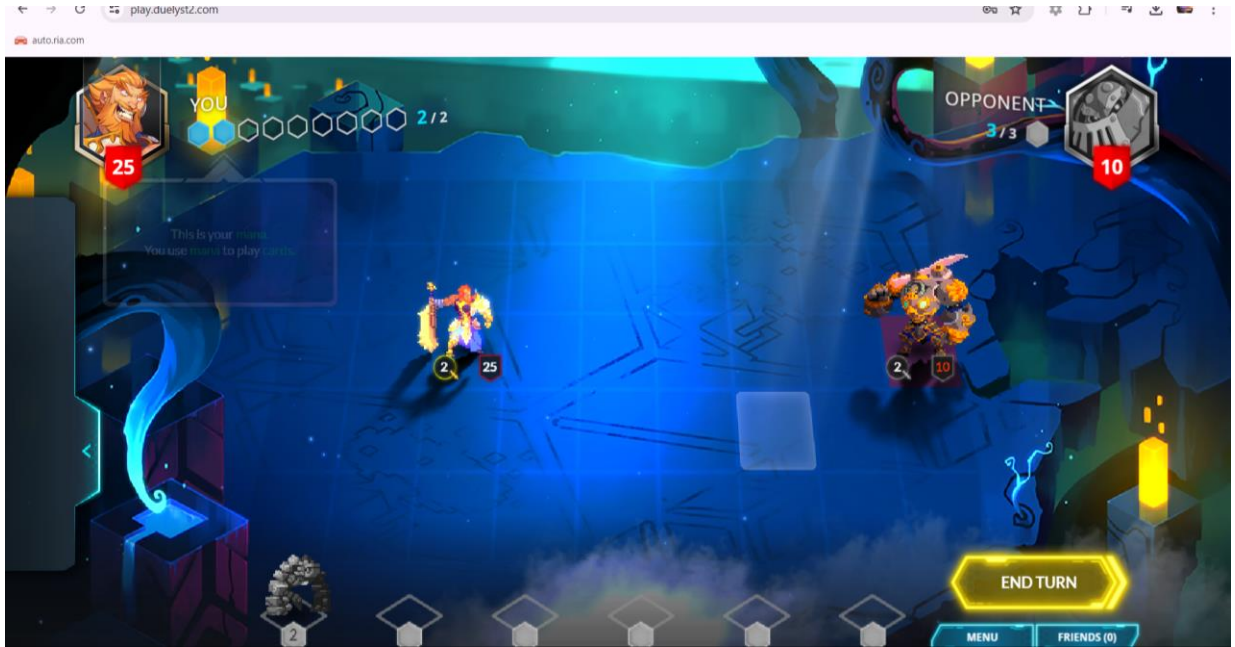


Рис. 1.5 – Duelyst



Рис. 1.6 – Duelyst

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Туренко С.С.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

Переваги:

- стратегічність і тактичність: Duelyst вимагає глибокого аналізу та стратегічного мислення. Гравці повинні розуміти взаємодію між різними картами, їх потужності і слабкі сторони, а також вміло використовувати їх в бою на дошці;
- унікальна механіка бою: Гра поєднує в собі елементи тактичних боїв на дошці зі змагальністю колекціонування карточок, що робить кожен бій варіативним і цікавим;
- глибина налаштувань колод: Гравці можуть створювати різні колоди з великою кількістю карт, що дає можливість експериментувати зі стратегіями і створювати унікальні комбінації;
- візуальний стиль: Гра має стильне візуальне оформлення і анімацію, яка додає в неї художній шар і робить геймплей захоплюючим;
- активна спільнота гравців: Duelyst має велику активну спільноту гравців, яка підтримується регулярними оновленнями і турнірами, що сприяє живому інтересу до гри.

Недоліки:

- високий вхідний поріг: Через складність стратегічного ігрового процесу, новачкам може бути важко зрозуміти всі аспекти гри і досягти конкурентоспроможності;
- часове споживання: Кожен матч у Duelyst може займати значну кількість часу, що може бути проблемою для гравців з обмеженим часом на ігри;
- баланс гри: Як і в багатьох онлайн-іграх, іноді можуть виникати проблеми з балансом між картами, що може впливати на досвід гри;
- мережеві проблеми: Оскільки гра залежить від інтернет-з'єднання, можуть виникати проблеми з мережею, такі як затримки або втрати зв'язку під час матчів;

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

- залежність від RNG: Як і у багатьох інших карточних іграх, в Duelyst є елементи випадковості (RNG), які можуть вплинути на результати гри і непередбачуваність ходів.

Далі наведена порівняльна таблиця цих ігор (таб. 1.1)

Таблиця 1.1

Порівняння аналогів

Характеристика	Agar.io	Duelyst	Catan
Гра на реакцію	+	-	-
Покрокова стратегія	-	+	+
Багатокористувацький режим	+	+	+
Наявність карт та полів	+	+	+
Різноманітність тактик	-	+	+
Легкість освоєння	+	-	-
Глибина геймплею	-	+	+
Популярність серед цільової аудиторії	+	+	+
Розмір завантаження	-	-	-

Порівнюючи agar.io, Duelyst та Catan, можна помітити значні відмінності в їх механіках та підходах до геймплею. Agar.io приваблює гравців своєю простотою, доступністю та динамічним геймплеем. Вона дозволяє швидко включитися в гру, має багатокористувацький режим і високу реіграбельність завдяки постійно змінюваним ситуаціям на карті. Однак, недоліки, такі як проблеми з балансом, відсутність різноманітності та наявність чітерів, можуть знизити задоволення від гри. Це гра, що добре підходить для коротких сесій і не вимагає значних витрат часу чи ресурсів.

Duelyst, навпаки, зосереджена на глибокій стратегії та тактиці. Гравцям необхідно ретельно аналізувати взаємодію карт та планувати свої дії, що робить гру більш складною, але й винагороджує за стратегічне мислення. Велика активна спільнота та унікальна механіка бою забезпечують живий інтерес до

гри. Однак, високий вхідний поріг, необхідність значного часу для кожного матчу та можливі проблеми з балансом можуть ускладнити гру для новачків. Catan, зі свого боку, пропонує високий рівень реіграбельності завдяки випадковому генеруванню острова та гнучкі можливості гри, підтримуючи різні режими і доступність на різних платформах. Незважаючи на це, тривалі партії, внутрішньоігрові покупки та вимоги до стабільного інтернет-з'єднання можуть стати перешкодами для деяких гравців.

1.3 Вибір архітектури системи

У сучасному світі, де інформаційні технології розвиваються дуже швидко, програмісти виявили багато типів зв'язків між різними компонентами програми. Це призвело до появи концепції архітектури і правильного планування коду. Правильний вибір підходу дозволяє зменшити обсяг роботи в майбутньому, наприклад, збільшити перевикористовуваність класів, розділяючи великі скрипти на дрібніші з конкретними завданнями.

Слово "моноліт" [\[24\]](#) має стародавнє походження і означає великий камінь. У сучасному контексті термін зберіг своє значення. У програмному забезпеченні монолітна архітектура означає створення єдиної неподільної структури. Концепція монолітного програмного забезпечення передбачає інтеграцію всіх компонентів програми в одну єдину структуру на одній платформі. Зазвичай монолітний додаток включає базу даних, інтерфейс користувача та серверну частину, які керуються централізовано.

Для визначення корисності цього архітектурного рішення для вашого бізнесу, розглянемо його переваги та недоліки.

Переваги:

- Простота розробки та розгортання: Використання інтегрованих інструментів спрощує процес розробки, а єдиний каталог полегшує розгортання. Монолітне ядро дозволяє розробникам вносити зміни та оновлення одночасно, що економить час;
- Легка інтеграція систем моніторингу: Монолітні додатки полегшують аналіз трафіку, логування та інші аспекти завдяки єдиній кодовій

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

базі. Підключення компонентів простіше, оскільки все працює в межах одного додатку;

- Вища продуктивність: Правильно побудовані монолітні програми зазвичай більш ефективні, порівняно з програмами на основі мікросервісів. Здатність ефективно взаємодіяти між компонентами через спільний код і пам'ять забезпечує кращу продуктивність.

Недоліки:

- Зростання обсягу коду: З розширенням продукту кодова база стає важчою для розуміння та редагування, особливо для нових розробників. Це ускладнює виявлення побічних ефектів та залежностей;

- Складність внесення змін: Через суцільну структуру згодом може бути важко або навіть неможливо змінити базові операції, які були додані на попередніх етапах розробки.

Мікросервісна архітектура (Microservices Architecture) [\[22\]](#) – це підхід до розробки програмного забезпечення, при якому додаток складається з невеликих незалежних служб, кожна з яких виконує чітко визначену функцію. Кожен мікросервіс працює як окрема одиниця та взаємодіє з іншими через стандартизовані API. Цей підхід відрізняється від традиційної монолітної архітектури, де всі компоненти інтегровані в одну велику систему.

Основні переваги мікросервісного програмування включають в себе:

- Незалежність розробки та розгортання: Кожен мікросервіс може бути розроблений, протестований, розгорнутий та масштабований незалежно від інших сервісів. Це дозволяє швидше вносити зміни та оновлення;

- Модульність: Кожен мікросервіс відповідає за конкретну бізнес-функцію і може бути розроблений з використанням різних мов програмування та технологій;

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

- Легкість масштабування: Мікросервіси можна масштабувати окремо, що дозволяє ефективніше використовувати ресурси та підвищувати продуктивність системи;

- Розподілена природа: Мікросервіси зазвичай розташовані на різних серверах або контейнерах, що забезпечує високу доступність і надійність системи;

- Стійкість до помилок: Якщо один сервіс виходить з ладу, це не впливає на роботу всієї системи, що підвищує її загальну надійність.

Недоліки:

- Складність управління: Управління великою кількістю незалежних сервісів вимагає складної інфраструктури та інструментів для оркестрації, моніторингу та управління;

- Затримки в комунікаціях: Взаємодія між сервісами відбувається через мережу, що може призвести до затримок і зниження продуктивності;

- Витрати на інфраструктуру: Підтримка багатьох мікросервісів потребує значних ресурсів для управління інфраструктурою, включаючи контейнеризацію, оркестрацію та моніторинг;

Серед запропонованих рішень було обрано мікросервісну архітектуру. Незважаючи на складне управління ми отримуємо перевагу у швидкості, та головне отримує легку масштабованість та чітку архітектуру.

Додаток можна логічно розділити на три сегмента(мікросервіса). Розглянемо основні частини програми:

- Мікросервіс що відповідає за основну логіку, та ініціює процеси.
- Мікросервіс що відповіде за частину боїв, отримує на вхід по 2 користувачі, та обраховує логіку бою
- Клієнт: отримує інформацію від сервіси через обробники, та демонструє гравцю в браузері потрібні елементи.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

При розробці такої гри потрібно обрати підхід та технології які нам будуть допомагати. Ось перелік основний технологій які ми будемо використовувати:

React є однією з найпопулярніших бібліотек для розробки інтерфейсів користувача. Його компонента орієнтована архітектура дозволяє створювати повторно використовувані та добре організовані компоненти, що значно спрощує процес розробки та підтримки додатку. Завдяки Virtual DOM, React забезпечує високу продуктивність, оскільки зміни в інтерфейсі відображаються швидко і ефективно. Крім того, велика спільнота та багата екосистема пакетів і інструментів підтримують швидкий розвиток і вирішення проблем.

Express[\[10\]](#) є легким і гнучким фреймворком для Node.js, який дозволяє швидко створювати серверні додатки. Його простота і мінімалістичний підхід дозволяють легко налаштовувати маршрути, обробку запитів та інтеграцію з іншими інструментами. Express добре підходить для створення RESTful API, що робить його ідеальним для взаємодії з клієнтською частиною, написаною на React. Швидке прототипування та велика кількість модулів роблять Express потужним інструментом для створення серверної логіки вашої гри.

WebSockets[\[13\]](#) забезпечують двостороннє спілкування між клієнтом і сервером у реальному часі, що є критично важливим для інтерактивних веб-ігор. Використовуючи WebSockets, ви можете реалізувати миттєве оновлення ігрового стану, обмін повідомленнями між гравцями та інші реальні інтерактивні функції без затримок. Це робить гру більш динамічною і захоплюючою для користувачів. На відміну від традиційних HTTP-запитів, WebSockets підтримують постійне з'єднання, що значно знижує накладні витрати на передачу даних.

Як мова програмування застосовується JS[\[17\]](#). Це інтерпритована мова програмування зі динамічною типізацією, для серверної частини. Для клієнту

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

буде використаний TS[8] - це строготипізований JS. Ці дві мови програмування ідеально підійдуть для роботи з websockets.

Мінімальні системні вимоги до пристрою користувача:

- 100 MB вільного простору на диску;
- Windows 7 або новіше, linux Ubuntu;
- 2GB RAM і більше;
- процесор з тактовою частотою не нижче 2GHz;
- мінімальна роздільна здатність екрану: 1024x768 пікселів;
- увімкнений JavaScript.

Висновки до першого розділу

У першому розділі було проаналізовано завдання та аналогічні ігри. При порівнянні з такими додатками, як Agar.io, Duelyst та Catan, було визначено ключові переваги, що будуть додані у наш продукт.

Було розглянуто різні типи архітектури та обрано найбільш оптимальний варіант. Для нашого проекту найбільше пасуватиме мікросервісна архітектура, що забезпечить гнучкість та адаптивність.

Під час вибору інструменту реалізації було обрано два фреймворки Express, React, що є чудовим рішенням для написання вебдодатків. Для програмування використовуватиметься мова JS, що зарекомендувала себе як надійна мова високого рівня для розробки додатків.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

РОЗДІЛ 2. ПРОЕКТУВАННЯ МУЛЬТИПЛЕЄР ВЕБ ГРИ

«"Deck Duel" у жанрі deckbuilding/buttlerrounds»

2.1 Визначення варіантів використання та структури системи

Для забезпечення неперервного виконання необхідних функцій і відповідності встановленим стандартам та специфікаціям, наведено високорівневі вимоги, які повинна задовольняти наша програма. Ці вимоги допомагають забезпечити безперебійну роботу системи та виконання всіх необхідних стандартів і специфікацій.

Бізнес вимоги

1. Основні цілі: проект створюється з метою залучення гравців до простої розслабляючої гри з елементами змагання
2. Представлення проекту: проект буде реалізовано у вигляді веб сторінки, з динамічним оновленням елементів на сторінці.

Вимоги користувачів

1. Можливість дізнаватись інформацію про картки
2. Можливість купувати, продавати, ставити на стіл картки
3. Можливість обирати героїв.
4. Можливість грати з друзями
5. Можливість використовувати унікальні властивості цих героїв
6. Можливість переглядати автоматично двобої
7. Можливість оновлювати вибір карток для покупки

Вимоги власників

1. Забезпечити безперебійну роботу додатку
2. Забезпечити комфортну гру для користувачів

Характеристика об'єкта комп'ютеризації

Користувач додатку буде мати можливість виконувати прості дії в грі, яка не потребує багато часу та зусиль на навчання, а також отримає гарне враження від швидкого занурення в гру. Після чого ж користувач може зацікавитись вивченням інших механік та ще більше зануритись в гру.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Функціональні вимоги:

1. Ігровий процес повинен проходити автоматично. Етапи гри мають змінюватись автоматично

3. Нескінченність ігрового процесу: при умові що гравці не втрачають здоров'є своїх героїв, гра, в теорії може тривати нескінечно

Нефункціональні вимоги:

1. Сприйняття

– час, необхідний для навчання звичайних користувачів – 1-5 хв;

– час, необхідний для навчання досвідчених користувачів – 10-20 ігрових годин

– інтерфейс має бути зрозумілим та з відповідними підказками.

2. Надійність

– Доступність – час, що витрачається на обслуговування додатку, не повинен перевищувати 5% від загального часу роботи.

– середній час безвідмовної роботи – 15 робочих днів.

– максимальна норма помилок та дефектів в роботі системи – 1 помилка на десять тисяч рядків коду.

3. Продуктивність

– Гра повинна легко працювати у всіх користувачів.

4. Можливість експлуатації

– Масштабування – система повинна мати можливість розвиватись у майбутньому та легко розширятись у наступних оновленнях.

Аналіз вимог користувачів дав підстави для визначення варіантів використання гіперказуальної гри про видобуток корисних копалин. Наведено діаграму використання програмної системи (рис. 2.1).

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

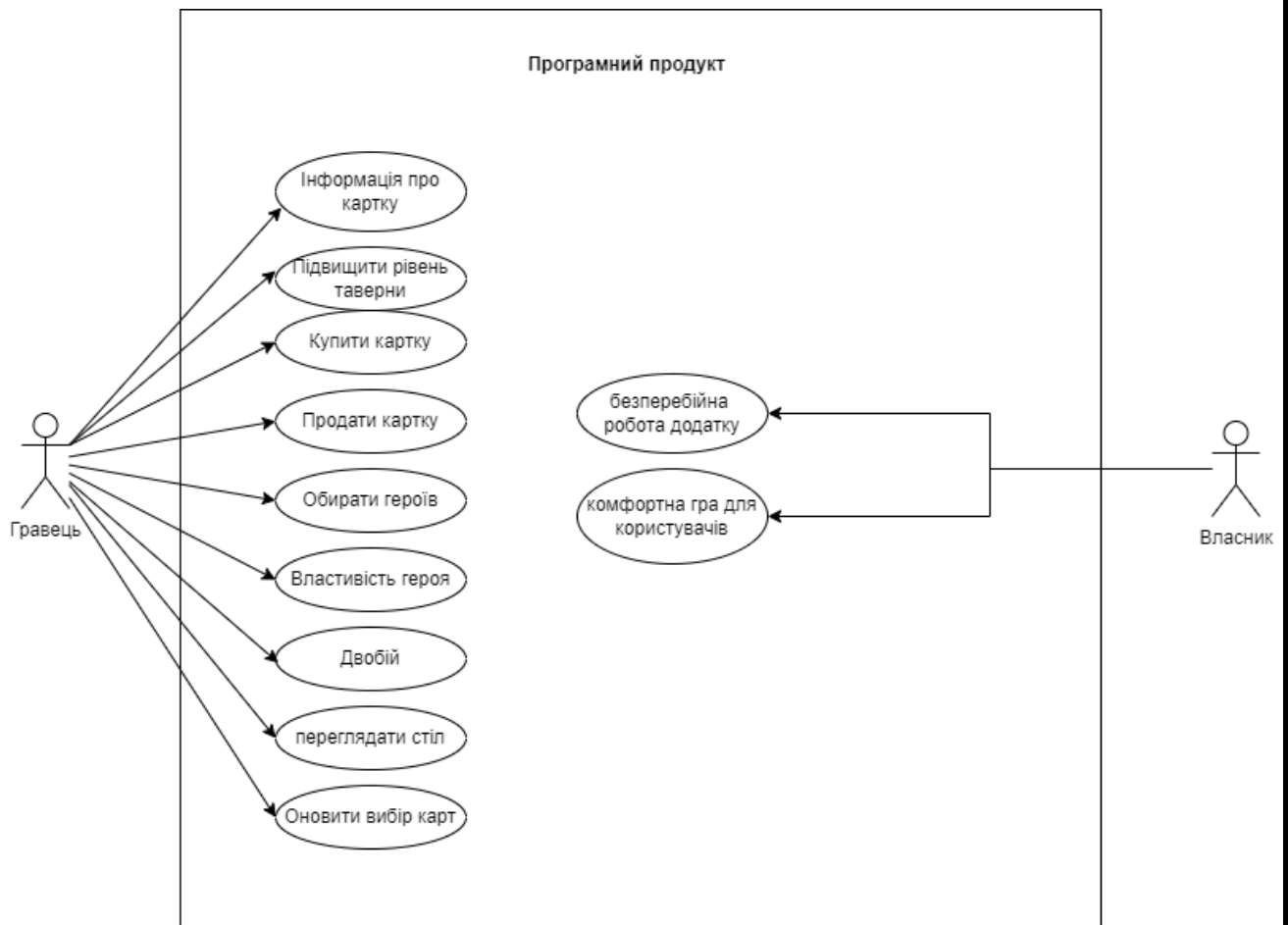


Рис. 2.1 – Діаграма використання системи

В зв'язку з тим що в грі можуть бути новачки, та люди які не знайомі з цим жанром в цілому. Буде наведений детальний опис по кожному пункту використання системи гравцем. Почати варто з стола та його розуміння в цілому в контексті гри.

ОБИРАТИ ГЕРОЯ. Гра починається з вибору героя на якому гравець буде грати. Героїв всього 8, та вони випадково розподіляються між гравцями.

ВЛАСТИВІСТЬ ГЕРОЯ. У кожного героя є своя унікальна властивість: активна чи пасивна.

ПІДВИЩИТИ РІВЕНЬ ТАВЕРНИ. В кожного гравця є свій унікальний показник рівня таверни. Від 1 до 5. В залежності від рівня, користувач має можливість купувати різні картки. Чим вище рівень, тим сильніші картки відповідно.

ПЕРЕГЛЯДАТИ ІНФОРМАЦІЮ ПРО КАРТКУ. Гра карткова та вся механіка побудована навколо цих карт. У кожної карти є свій показник рівня, атаки та здоров'я. А також унікальні властивості.

ПЕРЕГЛЯДАТИ СТІЛ. В кожного гравця є свій унікальний стіл, який складається з набору карток, завдання гравця за допомогою далі описаних механік покращувати свій стіл, збільшенням кількості карток та їх властивостей (пошкодження та здоров'я). Стіл може мати максимум 7 карток.

ОНОВИТИ ВИБІР КАРТ. Кожному гравцю в залежності від його рівня таверни дається на вибір певний вибір карток. В якому він може ..ПЕРЕГЛЯДАТИ ВЛАСТИВОСТІ КАРТОК.

КУПИТИ КАРТКУ. У кожного гравця на початку гри є 3 золот(валюта). Кожен новий раунд після двобою, кількість золота зростає на 1 до десяти. Гравець з запропонованих карт може обрати певну, та подвійним натисканням лівої кнопки миші придбати її. Коштуватиме це 3 золот. Та гравець отримає карту в руку що дасть йому можливість ..

ПОСТАВИТИ КАРТКУ НА СТІЛ. Ставить картку на стіл, тим самим покращує його «силу».

ПРОДАТИ КАРТКУ. Видаляє картку зі столу, дає гравцю один золота.

ДВОБІЙ. Найцікавіша подія в грі. Після того як скінчується час на раунд, стіл який встиг скласти за раунд гравець копіюється, також це відбувається у всіх і випадковим чином вони по 2 змагаються у двобоях, де визначають переможця. Переможений гравець отримує пошкодження свого героя в розмірі суми рівнів всіх виживших карт противника. І наступає новий раунд ,де можна будувати стіл далі.

У результаті аналізу функціональних вимог було створено діаграми активності, що відображають ймовірні сценарії розвитку подій у грі (рис. 2.2):

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

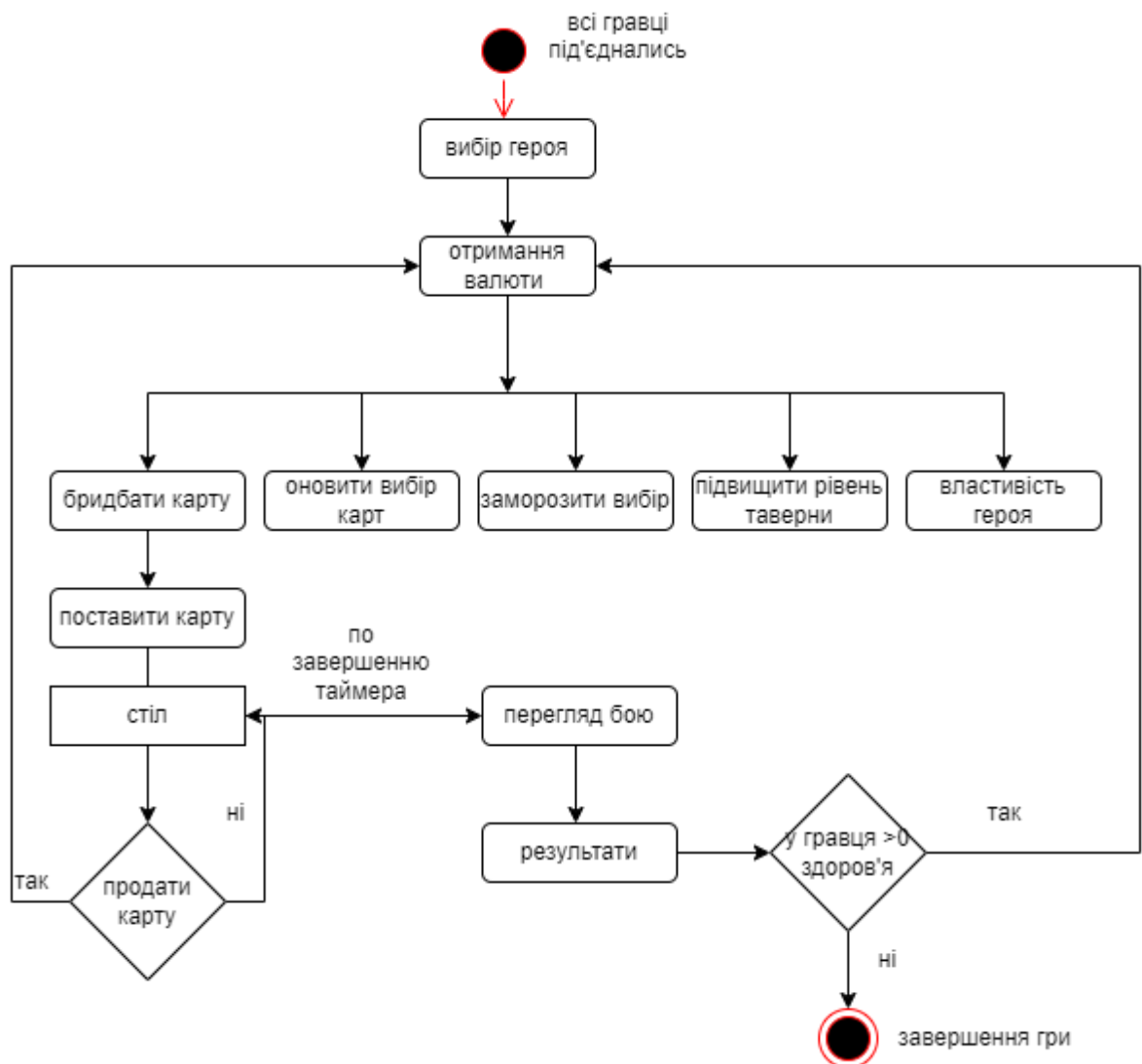


Рис. 2.2 – Діаграма активностей взаємодії користувача з основними процесами гри

Розглянемо умовні позначення на діаграмі активності:

- Початковий вузол діяльності (круг): початок потоку виконання діяльності.
- Контроль потоку (стрілка): послідовність виконання між діями.
- Дія (заокруглений прямокутник): конкретна дія, що виконується в системі.
- Вузол-рішення (ромб з розгалуженням): створює певну умову, в якій є два варіанти дій.
- Вузол-злиття (ромб після рішення): об'єднує різні шляхи виконання.

- Вузол-розгалуження (зафарбований прямокутник із стрілками): розділяє процес на паралельні потоки без умови.
- Об'єднуючий вузол (зафарбований прямокутник із стрілками): об'єднує паралельні потоки.
- Фінальний вузол діяльності (круг з контуром): завершує діаграму.
- Доріжка: групує дії, виконувані одним актором або об'єднує дії в одному потоці.

Було побудовано діаграми, що відображають основний процес гри з боку користувача, а також схема по взаємодії з картами та декбілдингу. Діаграма основних процесів гри відображає перелік можливостей користувача під час гри.

- Початковий вузол діяльності: момент коли всі гравці доєднались.
- Дія: вибір героя.
- Вузол-розгалуження;
- Дія: Придбати карту;
- Дія: Поставити картку
- Дія: Оновити вибір карт;
- Дія: Заморозити вибір
- Дія: Підвищити рівень таверни
- Дія: Використати властивість героя
- Вузол-стану(стіл);
- Вузол рішення: продати картку;
- Дія: переглядати бій;
- Дія: Переглядати результати
- Вузол-рішення;
- Кінцевий вузол: завершення гри.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

2.2 Розробка алгоритмів та ігрових механік

У нашій грі користувач наділений багатьма можливостями, однак його основною прямою розвитку буде будівництво власного столу з карток та посилення його кожен хід. Вирішено було скласти послідовну діаграму будівництва столу(рис. 2.3):

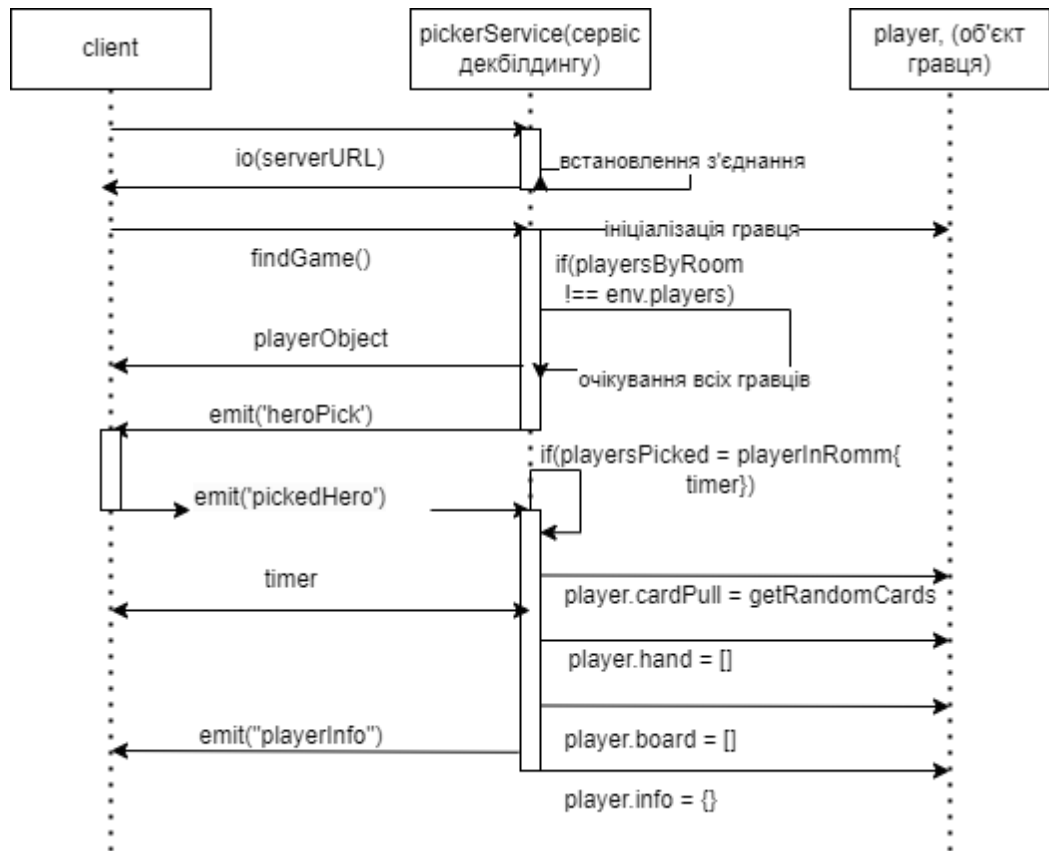


Рис. 2.3 Діаграма послідовності ініціалізації гравця

Вищезгаданий метод працює наступним чином, після того як користувач підключається до серверу, та викликає метод пошуку гри. Для нього на сервері створюється об'єкт за його айди сокета[12]. Після чого гравець очікує інших, та їм на вибір дають героїв. Після того ж як всі гравці обрали героїв, кожному клієнту віддається його об'єкт, в якому лежить інформація про гравця, його руку, стіл, та вибір карт. А також всі гравці отримують однаковий таймер, який синхронізує всіх у кімнаті.

Після стадії білдингу наступає стадія битви, давайте розглянемо діаграму послідовності роботи мікрсервісу та клієнту які за це відповідають.(рис. 2.5):

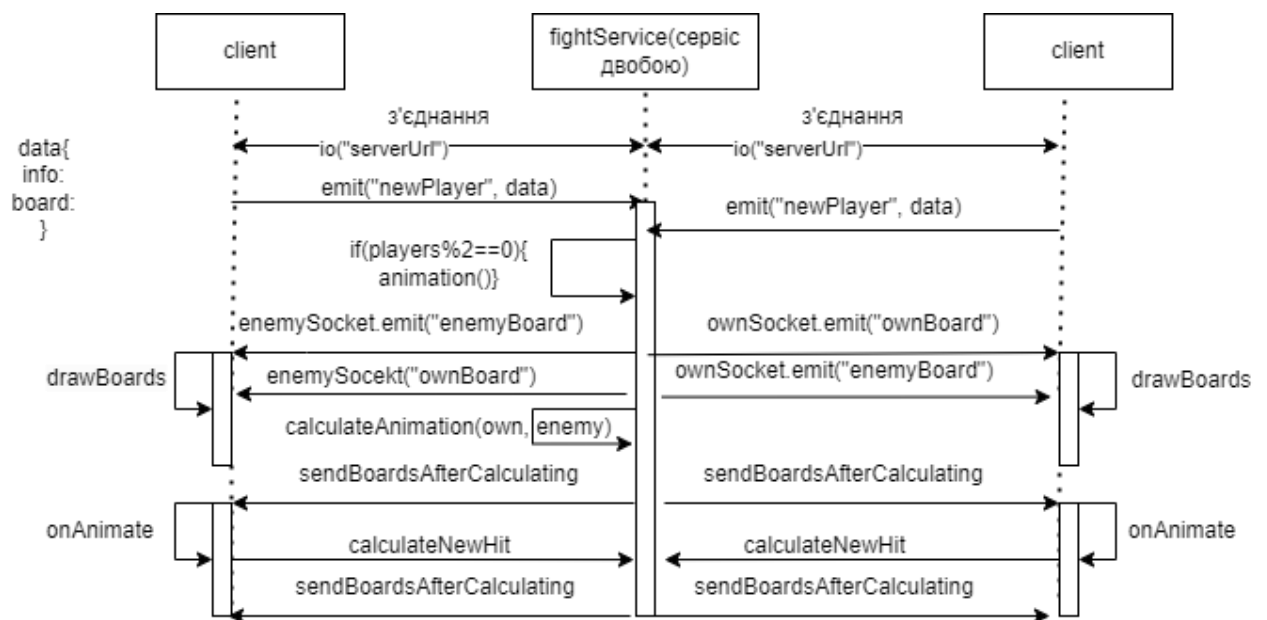


Рис. 2.4 – Діаграма послідовності обробки ударів під час битви

Діаграма демонструє наступну логіку: До сервісу під'єднуються гравці, сервіс їх зберігає в загальному об'єкті всі гравців по socket.id, у кожного гравця в об'єкті info, зберігається номер його кімнати в pickerService. Після під'єднання сервіс перевіряє кількість гравців по даному айді, якщо ж воно парне, сервіс бере двох останніх гравців які доєднались, вираховує об'єкти їх сокетів, та відсилає кожному гравцю по сокету інформацію про його противника. Паралельно викликається відбувається метод калькуляції анімації, який обраховує карту яка буде бити, карту в яку буде бити. Та порядок карт після удару.

Після обрахунку обом гравцям відсилається ідентична інформація про стоили, проте відзеркалена інформація про удар картки, оскільки гравці бачать один одного у дзеркалі. Даний алгоритм є рекурсивний. Виключення з алгоритму відбувається тоді, коли у якогось гравця або у обох стіл пустий. Тоді або нічия або хтось переміг. В такому випадку гравцям надсилається івент "emit("endBattle", {status: "draw,you win, you lose", damage: 0})". І гравці після 5 секундої затримки переходять знову до стадії декблдингу.

2.3 Реалізація функціоналу додатку

Основою інтерфейсу для користувача стає все ж таки React, та його компоненти, почнемо з реалізації підключення сокетів:

```
import React from "react";
import io, {Socket} from "socket.io-client"
import {DefaultEventsMap} from "@socket.io/component-emitter";
export interface CustomSockets{
  gameSocket: Socket<DefaultEventsMap, DefaultEventsMap>;
  fightSocket: Socket<DefaultEventsMap, DefaultEventsMap>;
}

export const gameSocket: Socket<DefaultEventsMap, DefaultEventsMap> =
io("http://localhost:5000")
export const fightSocket: Socket<DefaultEventsMap, DefaultEventsMap> =
io("http://localhost:8080")
export const SocketContext = React.createContext<null | CustomSockets>(null);
```

Ми імпортуємо метод підключення та тип даних самого Сокету що ми отримуємо, після чого створюємо об'єкти двох сокетів з якими будемо працювати, та найголовніше впроваджуємо контекст для цих сокетів.

```
const [playerInfo, setPlayerInfo] = useState<PlayerInfo | undefined>();
const [timeLeft, setTimeLeft] = useState<number>(0);
const [currentPlayers, setCurrentPlayers] = useState([]);
```

За допомогою хуків `useState` створюються стани для збереження інформації про гравця (`playerInfo`), часу, що залишився до кінця раунду (`timeLeft`), та поточних гравців у грі (`currentPlayers`).

```
const timerComponent = useMemo(() =>
  playerInfo && <div className='timer'>
    {timeLeft} S
  </div>,
  [timeLeft])
```

`useMemo` використовується для оптимізації відображення компонента таймера, щоб він оновлювався тільки тоді, коли змінюється `timeLeft`.

```
const onPlayers = useCallback((data: any) => {
  setCurrentPlayers(data)
}, []);
const onWarning = useCallback((data: any) => {
  toast(data)
}, []);
```

`useCallback` створює колбек-функції `onPlayers` і `onWarning`, які використовуються для обробки подій від веб-сокетів. `onPlayers` оновлює список поточних гравців, а `onWarning` відображає попередження за допомогою `toast`.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

```
useEffect(() => {
  gameSocket?.off("players", onPlayers).on("players", onPlayers)
  gameSocket?.off("players", onPlayers).on("players", onPlayers)
  gameSocket?.off("warning", onWarning).on("warning", onWarning)
}, []);
```

useEffect використовується для налаштування обробників подій для веб-сокетів, які оновлюються при кожному рендері. Функції onPlayers і onWarning прикріплюються до відповідних подій сокетів.

```
const ability = () => {
  gameSocket?.emit('ability')
}
```

Метод відправки запиту на сервер про властивість героя.

```
<SocketContext.Provider value={{ gameSocket, fightSocket }}>
```

Контекст з об'єктами сокетів

```
{ playerInfo &&
  <Hero
    heroImageUrl={playerInfo.hero.imgUrl}
    health={playerInfo.hero.hp}
    gold = {playerInfo.gold}
    hint = {playerInfo.hero.ability}
    abilityCost = {playerInfo.hero.abilityCost}
    ability = {ability}
  />
}
```

Відмальовування контенту за умови що в стані playerInfo, є інформація про гравця. Це основоположні методи які будуть далі використовувати в проєкті. Цікавим є використання даної структури написання: **gameSocket?.off("players", onPlayers).on("players", onPlayers).**

В цій здавалось на першій погляд простій структурі прихована дуже цікава річ, що наш **onPlayers** – а є синхронним обробником на вхід. На який ми підписуємось при кожному перерендері сторінки. Проте відписання можливе лише при відключенні від сервера, або виходу з сторінки, Тому нам потрібно механічно при кожному перерендері **.off**, викликати даний метод, щоб чистити всі підписки і наш працював коректно.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

В цілому у всієї решті компонентів логіка дуже схожа на те що наведено вище, тому зациклюватись на цьому не будемо, а ліпше розглянемо лістинг компонента, що відповідає за промальовування анімації.

```
const { board, hand, playerInfo, setPlayerInfo, setLoadingStatus } =
playerData.playerData
const sockets = useContext(SocketContext) as CustomSockets;
const { gameSocket, fightSocket } = sockets;
const [currentBoard, updateCurrentBoard] = useState<Board[]>(board)
const [animateObject, setAnimateObject] = useState<any>({})
const [animateStatus, setAnimateStatus] = useState('wait')
const [player, setPlayer] = useState(playerInfo);
const [enemyBoard, updateEnemyBoard] = useState<Board[]>([])
```

За допомогою методів useState створюються стани для збереження:

currentBoard: поточна дошка гравця.

animateObject: об'єкт анімації.

animateStatus: статус анімації.

player: інформація про гравця.

enemyBoard: дошка супротивника.

```
const onAnimate = useCallback((data: any) => {
  setAnimateObject(data)
  setAnimateStatus("go")
}, []);
const onUpdateCurrentBoard = useCallback((data: any) => {
  updateCurrentBoard(data.board)
}, []);
const onUpdateEnemyBoard = useCallback((data: any) => {
  updateEnemyBoard(data.board)
}, []);
const onEndBattle = useCallback((data: any) => {
  console.log("rmited")
  setAnimateStatus("wait")
  setTimeout(() => {
    gameSocket?.emit("nextRoundPick", {
      damage: data.collectedDamage,
      status: data.fightStatus
    });
    fightSocket.emit('leave');
  }, 5000);
}, []);
```

onAnimate: встановлює об'єкт анімації та змінює статус анімації на "go".

onUpdateCurrentBoard: оновлює поточну дошку гравця.onUpdateEnemyBoard:

оновлює дошку супротивника.onEndBattle: обробляє закінчення битви,

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Туренко С.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

встановлює статус анімації на "wait", а також через 5 секунд надсилає на сервер інформацію про наступний раунд і виходить з битви.

```
useEffect(() => {
  fightSocket.off("animation", onAnimate).on("animation", onAnimate)
  fightSocket.off("currentBoard", onUpdateCurrentBoard).on("currentBoard",
onUpdateCurrentBoard)
  fightSocket.off("enemyBoard", onUpdateEnemyBoard).on("enemyBoard",
onUpdateEnemyBoard)
  fightSocket.off('endBattle', onEndBattle).on('endBattle', onEndBattle)
  return () => {
    fightSocket.off('endBattle', onEndBattle)
  }
}, []);
```

Встановлення обробників, через метод useState

```
return (
  <Canvas style={{ width: "50vw", height: "34vh", position: "absolute", top:
"50%", left: "50%", transform: "translate(-50%, -60%)" }}>
    <Scene
      player={player}
      handLength={hand?.length}
      animateStatus={animateStatus}
      setAnimateStatus={setAnimateStatus}
      animateObject={animateObject}
      currentBoard={currentBoard}
      enemyBoard={enemyBoard}
      setCurrentBoard={updateCurrentBoard}
      setEnemyBoard={updateEnemyBoard}
      setLoadingStatus={setLoadingStatus}
    />
  </Canvas>
);
}
```

Компонент повертає «JSX», який містить:

- Елемент Canvas, який використовується для відображення анімаційної сцени. Він має стилі для позиціонування на екрані.
- Компонент Scene, якому передаються необхідні пропси для відображення анімації, включаючи:
 - player: інформація про гравця.
 - handLength: довжина руки гравця.
 - animateStatus: статус анімації.

- setAnimateStatus: функція для встановлення статусу анімації.
- animateObject: об'єкт анімації.
- currentBoard: поточна дошка гравця.
- enemyBoard: дошка супротивника.
- setCurrentBoard: функція для оновлення поточної дошки гравця.
- setEnemyBoard: функція для оновлення дошки супротивника.
- setLoadingStatus: функція для встановлення статусу завантаження.

Цей компонент відповідає за динамічне відображення та оновлення анімаційних елементів гри в реальному часі, використовуючи дані, отримані через веб-сокети. Також ми бачимо, що об'єкт павертає компонент <Scene />[\[7\]](#). Поглянемо що в середині цього компоненту.

```
useEffect(() => {
  fightSocket?.emit("newPlayer", {
    board: currentBoard,
    cardInHand: handLength,
    info: player
  })
}, []);
```

При першому відображенні сцени відправляємо запит до серверу та повідомляємо йому про наш стіл та гравця.

```
const moveCard = (card: any, speedByTick: number, direction: boolean, moveX:
boolean): void => {
  if (moveX) {
    direction ? card.position.x += speedByTick : card.position.x -= speedByTick;
  } else {
    direction ? card.position.y += speedByTick : card.position.y -= speedByTick;
  }
}
```

Основний метод руху карт при анімації, на вхід отримує об'єкт самої карти, швидкість руху, напрям руху по Y, напрям руху по X. На основі цих даних пересуває картку в потрібний бік на відстань speedByTick.

```
useFrame((state, delta) => {
  if (!hitterPos || !emitterPos) return;
```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Туренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

```

    const hiter = ownTurn ? cardRefs.current[hitterPos - 1] :
cardRefs.current[emitterPos - 1 + 7];
    const emitter = ownTurn ? cardRefs.current[emitterPos - 1 + 7] :
cardRefs.current[hitterPos - 1];

    if (!hiter || !emitter) return;

    const distanceBetweenCardsX = hiter.children[0].position.x -
emitter.children[0].position.x;
    const directionByX = distanceBetweenCardsX > 0 ? false : true;
    if (animateStatus === "go") {
        hiter.position.z = 0.001
        const moveYDirection = ownTurn ? true : false;
        if ((moveYDirection && hiter.position.y < emitter.children[0].position.y +
DISTANCE_BETWEEN_BOARDS_Y / 2) ||
(!moveYDirection && hiter.position.y > emitter.children[0].position.y -
DISTANCE_BETWEEN_BOARDS_Y / 2)) {
            moveCard(hiter, delta * (ANIMATION_CARD_SPEED *
Math.abs(distanceBetweenCardsX / DISTANCE_BETWEEN_BOARDS_Y)), directionByX, true);
            moveCard(hiter, delta * ANIMATION_CARD_SPEED, moveYDirection, false);
        } else {
            setAnimateStatus("back");
        }
    } else if (animateStatus === "back") {
        if ((ownTurn && hiter.position.y > 0) || (!ownTurn && hiter.position.y < 0))
    {
        moveCard(hiter, delta * (ANIMATION_CARD_SPEED *
Math.abs(distanceBetweenCardsX / DISTANCE_BETWEEN_BOARDS_Y)), !directionByX, true);
        moveCard(hiter, delta * ANIMATION_CARD_SPEED, !ownTurn, false);
    } else {
        setAnimateStatus("wait");
        hiter.position.z = 0;
        hiter.position.x = 0;
        hiter.position.y = 0;
        setCurrentBoard(currentPlayerBoardAfterHit);
        setEnemyBoard(enemyPlayerBoardAfterHit);
        fightSocket.emit('calculateNewHit', { currentBoard:
currentPlayerBoardAfterHit, enemyBoard: enemyPlayerBoardAfterHit });
    }
    }
    });

```

Метод нескінечно виконується, та надає нам значення $0 < \mathbf{delta} < 0.2$ [\[2\]](#). На початку відбуваються певні перевірки, щоб уникнути помилок. Та створюються об'єкти **hiter**, **emitter**. Які відповідно являються картою яка б'є, та яку б'ють. Дані про ці картки ми раніше отримали з сервера. Також є змінна **animateStatus** = “go” | “back” | “wait”. При першому статусі, ми маємо варіант

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

з рухом картки від корисувача, знизу до гори, відразу як картка зіштовхується з тією картою яку вона мала бити встановлюється статус **setAnimateStatus("back");**

Виконується умова руху до зіткнення, після зіткнення встановлюється даний статус. Після чого запускається цикл руху на попереднє місце, до координат 0, 0, 0. Відносно початкового положення.

Після того як карта повернулась на початкове положення, ми оновлюємо два набори карт, та заповнюємо їх даними після обрахунку, і запускаємо рекурсивно наступний метод на обрахунок наступного удару.

У випадку, коли нам приходить інформація про завершення бою, статус встановлюється «wait».

```
return (
  <>
  {enemyBoard && currentBoard && (
    <group>
    <ambientLight intensity={2} />
    <directionalLight intensity={2} />
    {currentBoard.map(({ name, health, damage, tavern_tier, image }, i) => (
      <Card
        tavern_tier={tavern_tier}
        textureUrl={image}
        key={name + 2 + i}
        name={name}
        hp={health}
        damage={damage}
        ref={(el: any) => (cardRefs.current[i] = el as CustomGroup)}
        position={[-7 + i * 2.5, -DISTANCE_BETWEEN_BOARDS_Y / 2.2, 0]}
      />
    ))}
    </group>
  )}
  </>
);
```

Такий вигляд має лістинг «JSX», що повертає сцена. В ній ми бачимо групу з двох масивів карток та світла, світло потрібне щоб ми бачили кольори а не суцільну темряву у сцені. В кожен картку з масиву об'єктів на вхід ми передаємо інформацію, про здоров'є, рівень та атаку цієї картки.

Цікавим є обрахунок позиції карток :

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Туренко С.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

`position={[-7 + i * 2.5, -DISTANCE_BETWEEN_BOARDS_Y / 2.2, 0]}.`

Ми бачимо передання на картку її позиції у форматі вектора, одномірного масива довжиною 3. [x,y,z]. Ми передаємо картам зсув по осі y, зсув навпіл на ширину між столами, оскільки центр проходить рівновіддалено від них, та по X, зсув відбувається на 7 одиниць вліво, для розміщення каарт з лівого краю.

```
const {gameSocket } = sockets as CustomSockets
let onDoubleClick;
const onBuyCard = () => {
  gameSocket?.emit("buyCard", {cardToBuy: card, index: index})
}

const onPlaceCard = () => {
  gameSocket?.emit("placeCard", {cardToPlace: card, index: index})
}

const onSellCard = () => {
  gameSocket?.emit("sellCard", {cardToSell: card, index: index})
}

switch (position) {
  case "pull":
    onDoubleClick = onBuyCard;
    break;
  case "hand":
    onDoubleClick = onPlaceCard;
    break;
  case "board":
    onDoubleClick = onSellCard;
    break;
  default:
    onDoubleClick = () => {};
}
```

В середині картки, ми отримуємо позицію картки, та об'єкт сокета. В залежності від позиції ми створюємо потрібний обробник для картки.

- картка знаходиться у виборі – купити;
- картка знаходиться в руці – поставити;
- картка знаходиться на столі – продати.

З клієнтською частиною та її нюансами закінчили, можемо перейти до сервісу з декбїлдингу, щоб продемонструвати вище згадані механіки у коді.

```
export const getRandomCardsByLevel = (cards, level, cardsCountPullByLevel) => {
  const tierCards = cards.filter(card => card.tavern_tier <= level);
  const selectedCards = [];
```

```

for (let i = 0; i < cardsCountPullByLevel[level]; i++) {
  const randomIndex = Math.floor(Math.random() * tierCards.length);
  selectedCards.push(tierCards[randomIndex]);
  tierCards.splice(randomIndex, 1);
}
return selectedCards;
}

export const readCardsData = (path) => {
  try {
    const data = fs.readFileSync(path, 'utf-8');
    return JSON.parse(data);
  } catch (error) {
    console.error('Помилка при читанні файлу:', error);
    return null;
  }
}

```

Почнемо з двох простих методів, які ми винесли в окремий файл, перший метод, на вхід отримує масив карток, рівень по який фільтруємо їх, та кількість карток які нам потрібні. Другий метод власне зчитує синхронно дані з файла та нівелює помилки

Об'єкт гравця:

```

export default class Player {
  constructor(name) {
    this.info = {};
    this.info.name = name;
    this.info.level = 1;
    this.info.levelUpCost = 5;
    this.info.hero = undefined;
    this.isBoardFreeze = false;
  }
}

```

Метод пошуку гри:

```

socket.on('findGame', (data) => {
  if(players[socket.id]) return;
  players[socket.id] = new Player(data.name);
  io.emit('updatePlayers', players);
  battleStatus = true;
  turn = 0;
  if (queueSet.has(players[socket.id])) {
    socket.emit('loading', 'finding game');
  } else {
    socket.emit('loading', 'finding game');
  }
}

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Турченко С.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

```

queueSet.add(socket.id);
if (queueSet.size == process.env.PLAYERS_COUNT) {
  const emptyRoom = emptyRooms.shift();
  queueSet.forEach(socketId => {
    const player = players[socketId];
    player.info.roomName = emptyRoom.name;
    emptyRoom.empty = false;
    emptyRoom.startTime = Date.now();
    const playerSocket = io.sockets.sockets.get(socketId);
    playerSocket.emit('loading', 'recruting');
    playerSocket.join(emptyRoom.name);
    shuffleArray(availableHeroes);
    const selectedHeroes = availableHeroes.slice(0,
process.env.PLAYERS_COUNT);
    playerSocket.emit('selectHero', { heroes: selectedHeroes });
  });
  queueSet.clear();
}
}
});

```

Доволі масштабний метод, коректніше буде називати це обробником на сервері, який чекаю, по **'findGame'**, виклик. Відразу бачио перевірку чи існує даний гравець, щоб не дублюватись по сокету в об'єкті.

Далі створюється по сокет айді в об'єкті новий гравець, з всіма вхідними даними, та конструктором самого об'єкту. Гравець у відповідь автоматично отримує перелік всіх гравців. Коли гравців набралась потрібна кількість для заповнення кімнати переходимо в наступну частину.

Ініціалізуємо пусту кімнату з переліку пустих кімнат, і починаємо в об'єкті сокета ініціатора, хто останній зайшов, перебирати всіх гравців в черзі,

І починаємо методом перебору, перебирати всіх гравців, та надсилати їм героїв на вибір. Та звісно присвоюємо їм ідентифікатор кімнати.Після чого очищаємо об'єкт черги, для нових гравців.

```

socket.on('heroPick', data => {

  const playersInfo = socketIdsInRoom.map(socketId => {
    const player = players[socketId];
    return {
      name: player.info.name,
      hero: player.info.hero
    };
  });
  io.to(roomName).emit('players', playersInfo);

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.С.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

```
}
});
```

Метод вибору героя, кожен гравець на клієнті обирає героя, і він присвоюється об'єкту його сокета. Після того як всі гравці обрали своїх героїв, ми по айді їхньої кімнати відправляємо сповіщення про початок гри, на клієнті в цей час відбувається відповідне перемальовування. Та перехід до обробника гри.

```
socket.on('game', () =>
  if (player.info.hero.hp <= 0) {
    socket.leave(roomName);
    delete players[socket.id];
    if (socketIdsInRoom.length === 1) {
      const winnerSocket = io.sockets.sockets.get(socketIdsInRoom[0])
      winnerSocket.leave(roomName);
      delete players[winnerSocket.id];
      winnerSocket.emit('end-battle', "Ти переможець над леймами, вітаю!!!")
      socket.emit('end-battle', "Ти нажаль програв, можливо повезе наступного
разу")
      return
    }
    const playersInfo = socketIdsInRoom.map(socketId => {
      const player = players[socketId];
      return {
        name: player.info.name,
        hero: player.info.hero
      };
    });
    socket.emit('end-battle', "Ти нажаль програв, можливо повезе наступного
разу")
    io.to(roomName).emit('players', playersInfo);
    return;
  }
```

Якщо у героя гравця закінчилося здоров'я (hp <= 0), сокет залишає кімнату і гравець видаляється з об'єкта players.

Якщо в кімнаті залишився один гравець, його сокет також залишає кімнату і він видаляється з players. Обом гравцям надсилаються повідомлення про закінчення бою (перемога або поразка)

Якщо в кімнаті залишилося більше одного гравця, всім гравцям у кімнаті надсилається оновлена інформація про гравців.

```
if(playersByRoomInPick.hasOwnProperty(roomName)){
```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.С.				41
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    playersByRoomInPick[roomName]++
  }else{
    playersByRoomInPick[roomName] = 1
  }
  if(playersByRoomInPick[roomName] == process.env.PLAYERS_COUNT){
    //20 + turn * 10
    let countdown = 10;
    const intervalId = setInterval(() => {
      if (battleStatus) {
        if (countdown > 0) {
          io.to(roomName).emit('ticker', countdown);
          countdown--;
        } else {
          clearInterval(intervalId);
          delete playersByRoomInPick[roomName]
          io.to(roomName).emit('loading', 'fight');
        }
      } else {
        clearInterval(intervalId);
      }
    }, 1000);
  }
}

```

Збільшується кількість гравців, що знаходяться в стадії підготовки до бою (playersByRoomInPick).

Якщо кількість гравців у стадії підготовки дорівнює значенню process.env.PLAYERS_COUNT, запускається зворотний відлік до початку бою (10 секунд). Якщо відлік закінчується, всім гравцям у кімнаті надсилається повідомлення про початок бою.

```

player.info.gold = 2 + turn;
player.cardsPulled = player.isBoardFreeze ? player.cardsPulled :
getRandomCardsByLevel(cards.cards, player.info.level, cardsCountPullByLevel);
player.hand = player.hand || [];
player.board = player.board || [];
socket.emit('updateBoardAndHand', player);

```

В кінці після цих перевірок гравцю призначаються його вибір карт, рука та стіл, золото. Після стадії двобою, нам потрібно знову дати відлік та почати новий хід білдингу, тому ми маємо обробник:

```

socket.on('nextRoundPick', ({ damage, status }) =>

```

```

  if(player.info.levelUpCost > 0){
    player.info.hero.abilityTurnsDisable--;
  }
}

```

Оновлюємо використання властивості гравцю

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.С.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (player.info.levelUpCost >= 0) {
    player.info.levelUpCost--;
}
Знижуємо вартість підвищення таверни на 1
if (isNaN(damage)) {
return;
}
«Заглушка», на випадок некоректної дати
    if (status === 'you lose') {
    player.info.hero.hp -= (damage);
}
Віднімання здоров'я у випадку поразки
socket.emit('loading', 'starting game...');
Метод виклику на клієнті початку раунда білдинга
const roomName = player.info.roomName;
const socketIdsInRoom = Array.from(io.sockets.adapter.rooms.get(roomName) ||
[]);
const playersInfo = socketIdsInRoom.map(socketId => {
    const player = players[socketId];
    return {
        name: player.info.name,
        hero: player.info.hero
    };
});
io.to(roomName).emit('players', playersInfo);
Оновлення інформації по гравцям, якщо хтось втратив здоров'я
Далі йдуть методи взаємодії з столом, рукою, та вибіркою карт.
const player = players[socket.id];
if (player.info.levelUpCost > player.info.gold) {
    socket.emit('warning', 'You need more money');
    return;
}
if (player.info.level >= process.env.MAX_LEVEL) {
    socket.emit('warning', 'Max level');
    return;
}
player.info.gold -= player.info.levelUpCost;
player.info.level++;
player.info.levelUpCost = levelUpPrice[player.info.level];

```

Перевіряємо чи у гравця достатньо коштів, та чи не максимальний рівень, у противному підвищуємо рівень та віднімаю плату за це.

```

if (player.info.gold > (rollCost - 1)) {
    player.info.gold -= rollCost;
    player.cardsPulled = getRandomCardsByLevel(cards.cards, player.info.level,
cardsCountPullByLevel);
    socket.emit('updateBoardAndHand', player);
}

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Туренко Є.С.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

Перевіряємо наявність коштів, та оновлюємо вибір карток та відправляємо на клієнт оновлений об'єкт гравця.

```
if (player.cardsPulled[index] && player.cardsPulled[index].name === cardToBuy.name) {
    player.hand.push(cardToBuy);
    player.cardsPulled.splice(index, 1);
    player.info.gold -= cardCost;
    socket.emit('updateBoardAndHand', player);
} else {
    socket.emit('warning', 'Ця карта недоступна для покупки');
}
```

З подібними перевітками, переміщаємо карту з масива вибору у руку, та віднімаємо кошти за це.

Схожа механіка і у виставлені карт на стіл і при продажі, з єдиною різницею що при продажі золото нараховується гравцю.Єдина відмінність, що якщо у карти є властивість «інкріз» то ми перевіряємо цю умову, і виконуємо певні дії.

```
if (cardToPlace.ability == 'інкріз-2/2') {
    targetCard.damage += 2;
    targetCard.health += 2;
}
if (cardToPlace.ability == 'інкріз-10/10') {
    targetCard.damage += 10;
    targetCard.health += 10;
}
```

Розберемо одну з унікальних властивостей:

```
if (player.info.hero.name === "Токі") {
    player.cardsPulled = getRandomCardsByLevel(cards.cards, player.info.level, cardsCountPullByLevel);
    const tierCards = cards.cards.filter(card => card.tavern_tier === (1 + player.info.level));
    player.cardsPulled.push(tierCards[randomInt(0, tierCards.length)]);
    player.info.hero.abilityTurnsDisable = 1;
    socket.emit('updateBoardAndHand', player);
    return
}
```

Це властивість героя «Токі» [\[20\]](#), його сутність що при використанні своєї властивості, в неї віднімається 1 золота проте у вибір карт дається 1 карта на рівень вище від її рівня таверни.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Туренко Є.С.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

Перейдемо до мікросервісу з двобою двох гравців, в ньому лише два обробника, проте досить великих, ми розберемо лише метод обрахунку двобою, та як відбувається синхронізація у двох гравців.

```
const calculateAnimation = (currentBoard, enemyBoard, roomId) => {
  console.log('calculating animation')
  if (!Array.isArray(currentBoard) || !Array.isArray(enemyBoard)) {
    throw new Error("Invalid board data");
  }
  if (!queueHitInRoom.hasOwnProperty(roomId)) {
    queueHitInRoom[roomId] = Math.floor(Math.random() * 2) + 1;
  } else {
    queueHitInRoom[roomId]++
  }
  const currentPlayerBoardAfterHit = JSON.parse(JSON.stringify(currentBoard));
  const enemyPlayerBoardAfterHit = JSON.parse(JSON.stringify(enemyBoard));
  const hitterPos = 1
```

Перевірка валідності та створення об'єкту порядку ударів, та ініціалізація індексів, ударник та отримувача.

```
const tauntIndices = enemyPlayerBoardAfterHit
  .map((card, index) => card.taunt ? index : -1)
  .filter(index => index !== -1);

let emitterPos;
if (tauntIndices.length > 0) {
  // Randomly select a taunt if taunts are present
  emitterPos = tauntIndices[Math.floor(Math.random() * tauntIndices.length)] + 1;
} else {
  // Select a random enemy card if no taunts are present
  emitterPos = Math.floor(Math.random() * enemyPlayerBoardAfterHit.length) + 1;
}
if (currentBoard.length === 0 && enemyBoard.length === 0) {
  return {
    fightStatus: "draw",
    collectedDamage: 0
  }
}
```

Перевірка чи є в певному столі карти з властивістю «таунт», що означає що бити потрібно саме в них.

```
if (currentBoard.length === 0) {
  const collectedDamage = enemyBoard.reduce((accumulator, item) => {
```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Турченко Є.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

```

        return accumulator + item.tavern_tier;
    }, 0)
    return {
        fightStatus: "you lose",
        collectedDamage
    }
}
if (enemyBoard.length === 0) {
    const collectedDamage = currentBoard.reduce((accumulator, item) => {
        return accumulator + item.tavern_tier;
    }, 0)
    return {
        fightStatus: "you win",
        collectedDamage
    }
}

```

Перевірка чи столи не є пустими, у противному випадку, битва закінчується і рахуються результати.

```

currentPlayerBoardAfterHit[hitterPos - 1].health -=
Number(enemyPlayerBoardAfterHit[emitterPos - 1].damage);
enemyPlayerBoardAfterHit[emitterPos - 1].health -=
Number(currentPlayerBoardAfterHit[hitterPos - 1].damage);
if (currentPlayerBoardAfterHit[hitterPos - 1].health <= 0) {
    currentPlayerBoardAfterHit.splice(hitterPos - 1, 1);
}
if (enemyPlayerBoardAfterHit[emitterPos - 1].health <= 0) {
    enemyPlayerBoardAfterHit.splice(emitterPos - 1, 1);
}
return {
    currentPlayerBoardAfterHit,
    enemyPlayerBoardAfterHit,
    hitterPos,
    emitterPos,
    ownTurn: queueHitInRoom[roomId] % 2 !== 0
}

```

Обрахунок властивостей карт після удару, чи є загинувші карти($hp < 0$), та повернення їх на клієнт.

```

const generateNewRoomId = (roomId) => {

    if (playersByRoomId.hasOwnProperty(roomId)) {
        playersByRoomId[roomId]++
    } else {
        playersByRoomId[roomId] = 1
    }

    if (playersByRoomId[roomId] <= 2) {

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return roomId + '-' + "1"
  } else {
    return roomId + '-' + "2"
  }
}

```

Метод генерації унікальних парних ідентифікаторів кімнат розрахований на 4 людей в грі.

```

const reverseFightStatus = {
  "you win": "you lose",
  "you lose": "you win",
  "draw": "draw"
}

```

Об'єкт розвороту даних.

```

if (socketsInRoom.length === 2) {
  for (const socketId of socketsInRoom) {
    if (socketId !== socket.id) {
      enemySocket = getFullSocketByID(io, socketId)
      const enemyData = players[socketId].info.hero
      const ownData = players[socket.id].info.hero
      socket.emit('enemyData', enemyData)
      enemySocket.emit('enemyData', ownData)
      enemySocket.emit('currentBoard', players[socketId])
      enemySocket.emit('enemyBoard', data)
      socket.emit('currentBoard', data)
      socket.emit('enemyBoard', players[socketId])
      const ownAnimationData = calculateAnimation(data.board,
        players[socketId].board)
      const enemyAnimationData = {
        currentPlayerBoardAfterHit: ownAnimationData.enemyPlayerBoardAfterHit,
        enemyPlayerBoardAfterHit: ownAnimationData.currentPlayerBoardAfterHit,
        hitterPos: ownAnimationData.emitterPos,
        emitterPos: ownAnimationData.hitterPos,
        ownTurn: !ownAnimationData.ownTurn
      }
    }
  }
}

```

Вище описаний код є тим, що відбувається після підключення двох гравців в одну кімнату, Ми можемо бачити як по айді 2 користувача ми дістаємо об'єкт всього сокета, та вже з ним взаємодіємо. Власне що ми надсилаємо інформацію про столи для кожного гравця, опоненту в розвернутому вигляді та інформацію про опонента

```

if (ownAnimationData.hasOwnProperty('fightStatus')) {
  socket.emit('endBattle', ownAnimationData)
  const enemyAnimationData = {
    fightStatus: reverseFightStatus[ownAnimationData.fightStatus],
  }
}

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Туренко Є.С.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        collectedDamage: ownAnimationData.collectedDamage
    }
    enemySocket.emit('endBattle', enemyAnimationData)
    socket.emit('enemyData', null)
    enemySocket.emit('enemyData', null)
} else {
    socket.emit('animation', ownAnimationData)
    enemySocket.emit('animation', enemyAnimationData)
}
}
}
}

```

Також у випадку, якщо обрахунки закінчились, та ми маємо результат двобою, ми також відправляємо кожному результати.

```

socket.on('leave', () => {
    console.log(`Player left, with id: ${socket.id}`);
    const player = players[socket.id]
    if(!player) return
    const roomName = player.roomId;

    socket.leave(roomName);
    delete playersById[players[socket.id].info.roomName]
    delete playersObject[roomName];
    delete queueHitInRoom[roomName];
    delete players[socket.id];
});

```

І в фіналі анімації, ми з кожного клієнта чистимо всі дані про себе на сервері. Це робиться для того, по перше, щоб не забивати пам'ять, та постійно не генерувати нові айдішки, а й в випадковому порядку, по послідовності підключення сортувати їх у двобої.

Висновки до другого розділу

Під час проектування та розробки мобільної гри були визначені функціональні та нефункціональні вимоги, бізнес вимоги, а також необхідний функціонал для користувачів. Відповідно до врахованих особливостей програми було спроектовано дві діаграми послідовності. Розроблена структура додатку відповідає тій, що була спланована на початку.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.С.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ГРОЮ «Мультиплеєр веб гра "Deck Duel" у жанрі deckbuilding/battlerounds»

3.1 Порядок встановлення та налаштування параметрів додатку

Для початку гри, користувачу потрібно мати гарне інтернет з'єднання, та браузер.(рис. 3.1):

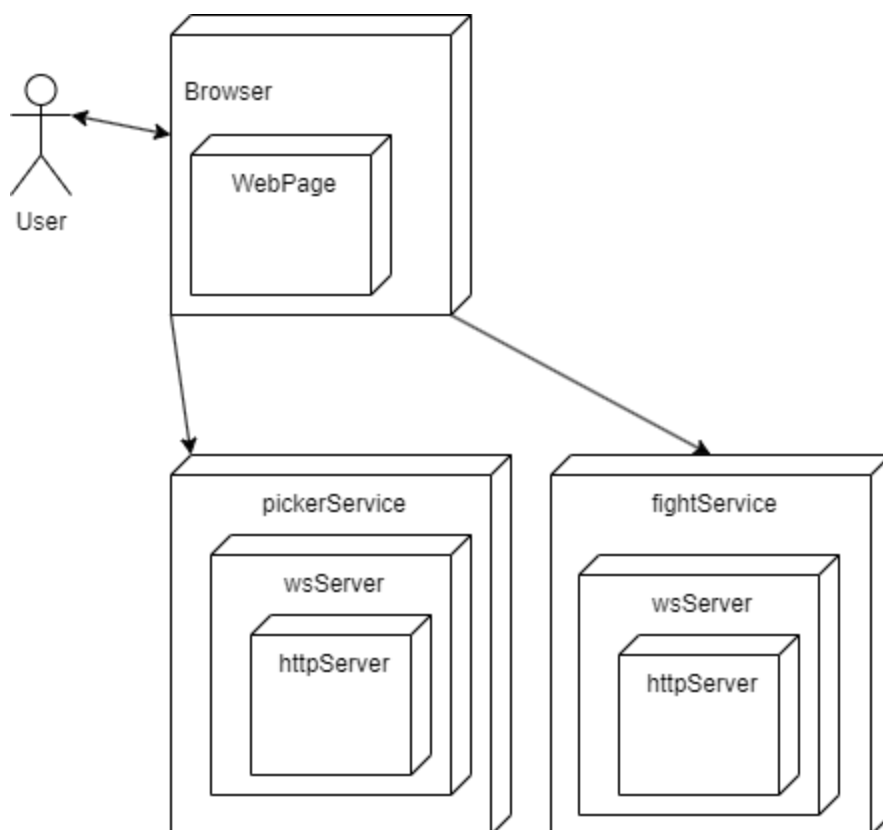


Рис. 3.1 – Діаграма розгортання

На діаграмі зображена система, що складається з трьох вузлів: браузера та двох мікросервісів. Система розроблена таким чином, щоб забезпечити ефективну взаємодію між клієнтською частиною і серверною логікою гри. Спочатку розробник повинен розгорнути два мікросервіси, кожен з яких виконує окремі функції та взаємодіє з іншими компонентами системи. Один мікросервіс може відповідати за управління даними гравців та їхньою автентифікацією, тоді як інший - за обробку ігрової логіки та механіки. Після розгортання необхідно перевірити коректність виконання запитів до кожного мікросервісу, що дозволяє впевнитися в їхній стабільній роботі та відповідності вимогам.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко Є.Є.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

Після успішної перевірки мікросервісів можна починати надсилати запити з веб-сторінки гри, що запускається у браузері. На цьому етапі розробник налаштовує взаємодію фронтенду з бекендом, забезпечуючи коректне виконання ігрових запитів, таких як реєстрація гравців, запуск матчів та обробка результатів боїв. Це дозволяє гравцям взаємодіяти з грою через веб-інтерфейс, використовуючи функціонал, розроблений та протестований на серверній стороні. Таким чином, забезпечується повноцінний геймплей, де всі компоненти системи працюють злагоджено і ефективно, створюючи стабільний ігровий досвід для користувачів.

3.2 Структура інтерфейсу додатку та порядок роботи

Для початку користувач заходить на веб сторінку, та отримує картинку де на задньому фоні має бекграунд гри, та пропозицію ввести ім'я і почати пошук кімнати.(рис. 3.2):

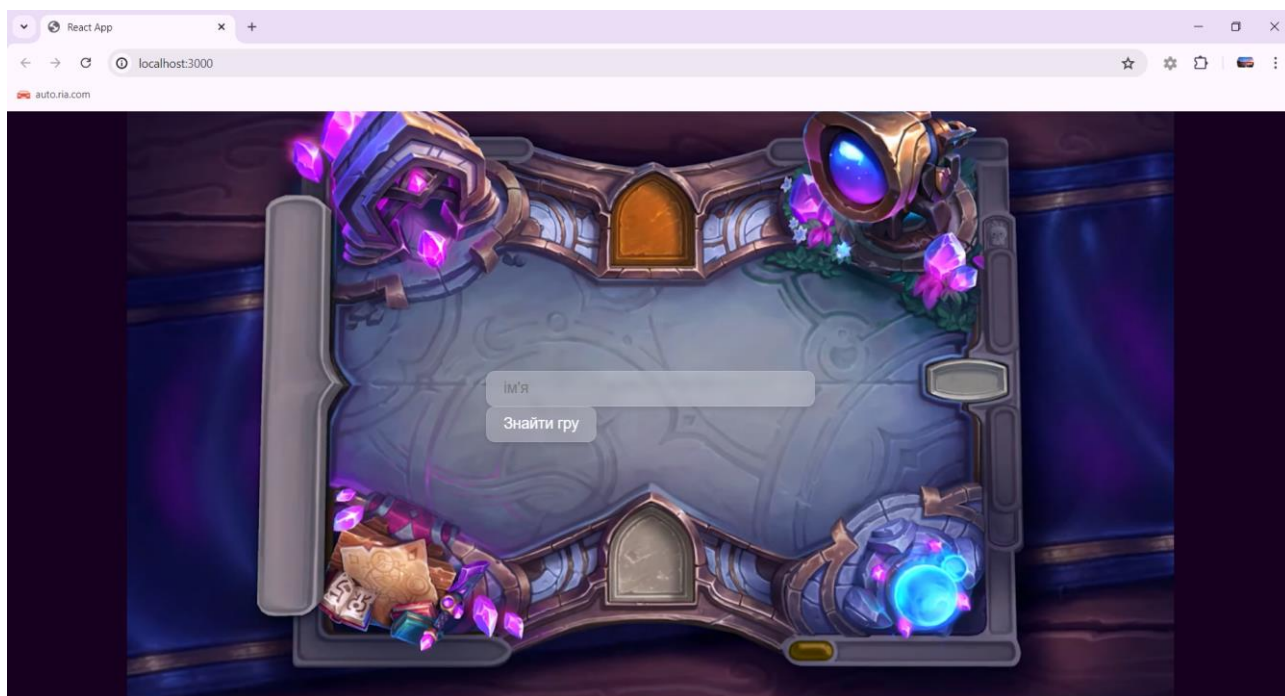


Рис. 3.2.- Введення імені та кнопка початку

Якщо це перший запуск – гравцеві відкриється міні-комікс, в якому демонструється сутність гри (рис. 3.3):

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турденко Є.Є.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

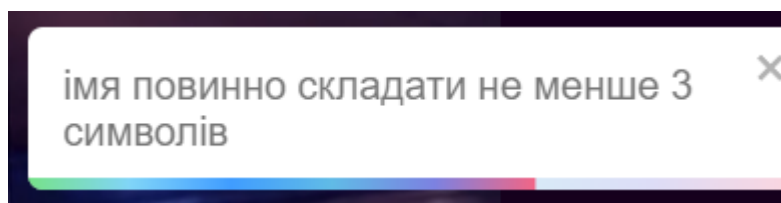
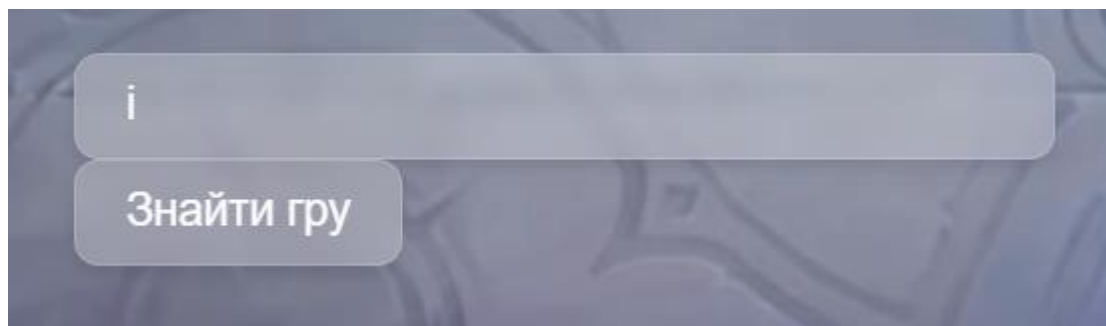


Рис. 3.3-3.4 – Помилка при введенні імені

Далі ми отримуємо екран очікування решти гравців: рис 3.5

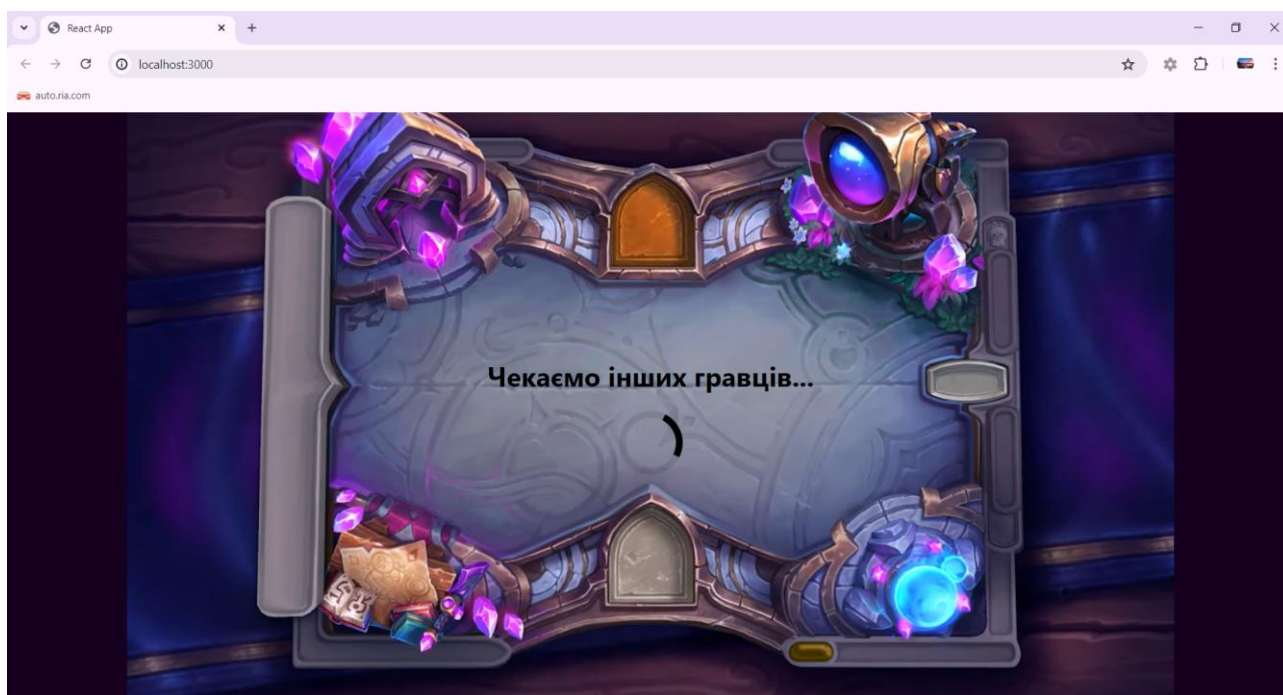


Рис 3.5 – Очікування гравців

Після того як всі гравці приєднались, нам відмальовують вибір героїв, виглядає він наступний чином, варто звернути увагу на підказки зверху над вибором.рис – 3.6

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

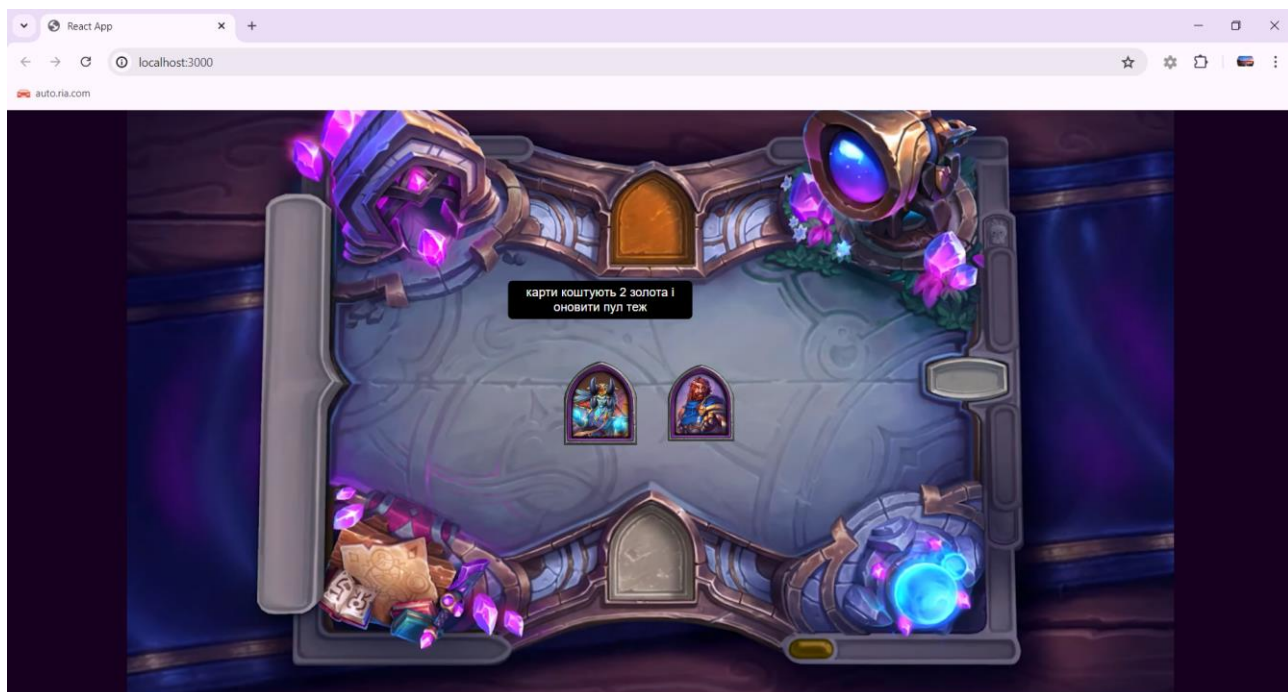


Рис 3.6 – Вибір героя

В грі на вибір існує 8 герої з їх унікальними властивостями: «Кубанчик», «Пуджерс», «Токіус», «Манаштурмус», «Ноздормус», «Бакабот», «Піраміда», «Рапан». Далі будуть показані їхні обгортки та здібності (рис. 3.7 – 3.14).

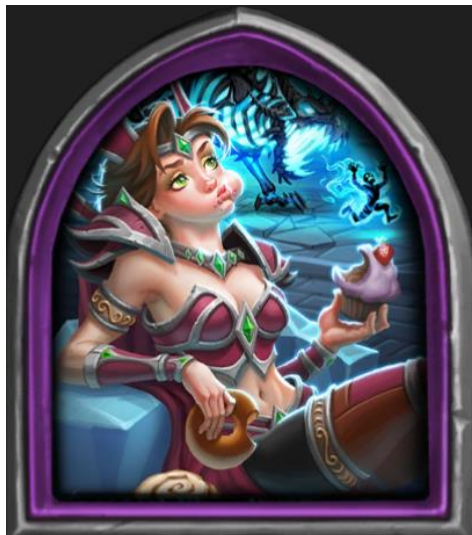


Рис 3.7 – Кубанчик

Кубанчик : дає золото від 1 до 6 коштує 1 золота, не можна використовувати впродовж тієї кількості ходів, скільки отримали золота.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				52
Змн.	Арк.	№ докум.	Підпис	Дата		



Рис 3.8 – Бакабот

Бакабот: підвищення таверни коштує на 1 менше.



Рис 3.9 – Манаштормус

Манаштормус: карти коштують два золота і оновити вибір теж.



Рис 3.10 – Ноздормус

Ноздормус: безкоштовно оновлює таверну.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		53



Рис 3.11 – Рапан

Рапан: випадкова карта отримує характеристики, що дорівнюють вашій таверні, коштує 1 золота.



Рис 3.12 – Пудж

Пудж: Здоров'є збільшено до 50.



Рис 3.13 – Піраміда

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

Піраміда: підвищує рівень здоров'я у карти на рівень вашої таверни, коштує 0 золота.

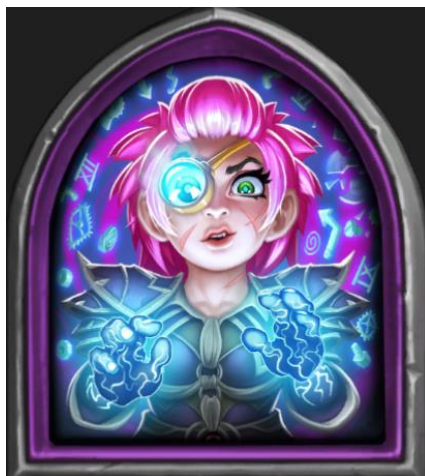


Рис 3.14 – Токі

Токі: Оновлює за один золота вибір таверни, та дає додаткову карту на рівень вище.

Після того як гравець обрав одного з героїв він переходить до основного елементу гри: «стадія декбілдингу»



Рис 3.15 – Основний інтерфейс

На ньому ми можемо спостерігати картинку нашого героя якого ми обрали знизу по центру. В правому нижньому кутку є інформація про кількість золота на даний момент у гравця. У лівій вертикальній панелі показані, інші гравці, та їхня кількість здоров'я та імена. З правого боку від героя можемо бачити зелений круг з метеором - це властивість героя. Зверху над нею є ціна за

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гурденко Є.Є.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

використання. Навпроти героя, зверху, є людина, це так званий «Власник Таверни», умовний герой, який нібито пропонує нам пограти. З правого боку на лінії розподілення, є таймер, який сигналізує нам про початок нової стадії. Розберемо детальніше верхню панель рис 3.16.



Рис 3.16 – Верхня панель керування

По центру ж знову бачимо нашого «Власника Таверни», з лівого до правого боку по черзі бачимо, кнопку підняття рівня таверни та ціну за це зверху, рівень нашої таверни, кнопка оновлення вибору карт, та кнопка заморозки вибраних карт. Переходимо до наступної стадії - двобою



Рис 3.17 – Момент удару під час двобою

На попередньому рисунку, ми бачимо як карта під два номером на столі спостерігача, наносить удар в карту під номером два противника. Після того як

у нас закінчилися карти, оскільки стіл противника був сильнішим, бачимо наступну картину (рис. 3.18).

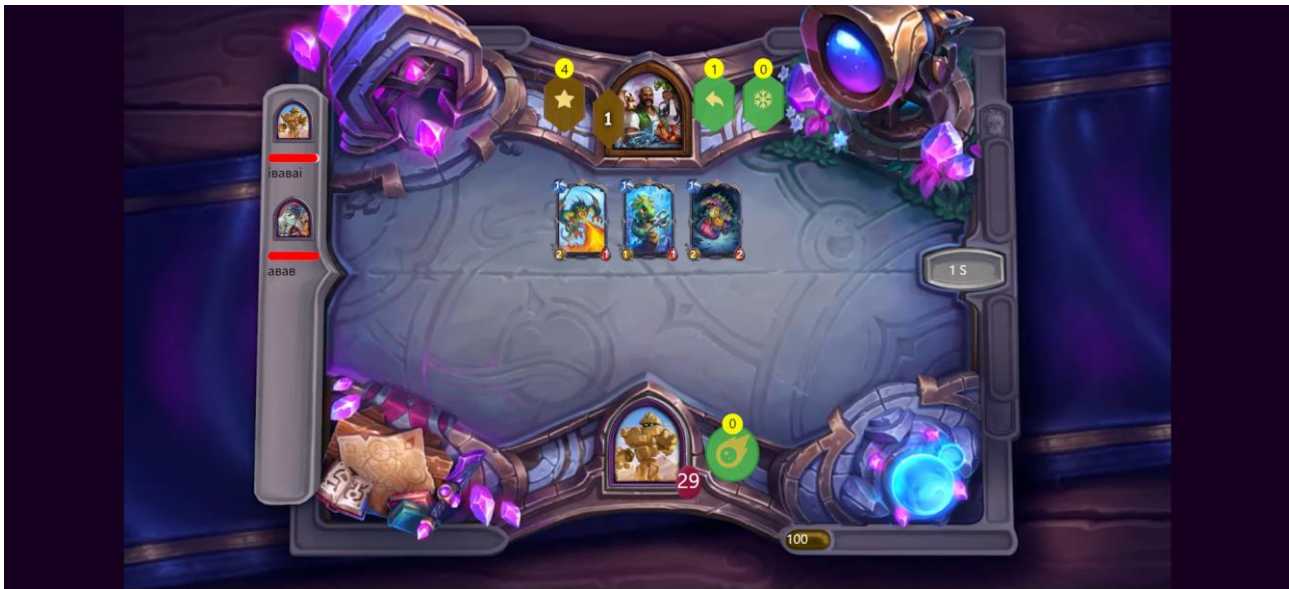


Рис. 3.18 – Перегляд отриманого пошкодження

Ми бачимо, що попередній раунд битви закінчився для нас поразкою, та ми отримали 1 пошкодження, оскільки у противника на столі залишилась карта першого рівня. Також ми бачимо, що почався наступний раунд, в нас знову оновилося золото, та зменшилась на 1 золота ціна за покращення рівня таверни. Що ж відбудеться, коли у гравця закінчиться повністю здоров'є. Він отримає сповіщення (рис 3.19) В разі перемоги, коли всі інші програли, користувач отримає наступне (рис 3.20)

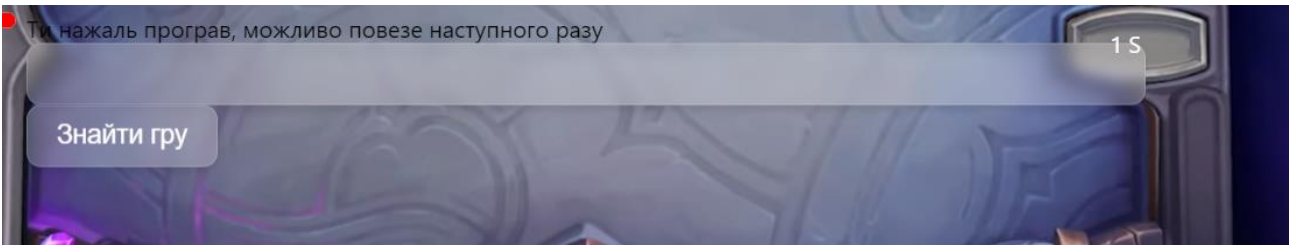


Рис. 3.19 – Сповіщення про програш

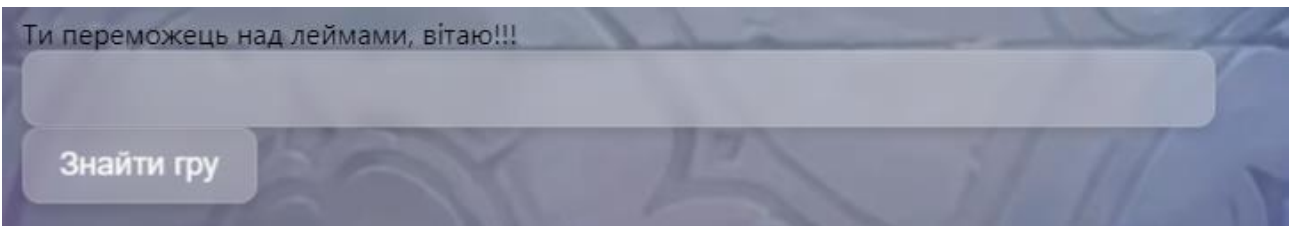


Рис. 3.20 – Сповіщення про виграш

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко С.С.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

Наприкінці, додамо трішки про карти та їхні властивості (рис 3.21-3.23)



Рис 3.21-3.23 – Карти

На малюнках зображені 3 карти з властивостями, які поміж позначень здоров'я і атаки, також мають властивості, перша карта має властивість «таунт», що означає, що всі карти першочергово мають лупцювати саме неї, друга карта має властивість «інкріз 2.2», що означає, що вона випадковій карті на полі бою, додає два пошкодження і два здоров'я. І остання карта має «Інкріз 10.10», ця карта робить те ж що і попередня, тільки додає більше, проте і має вищий рівень

3.3 Тестування роботи додатку

Тестування відіграє надзвичайно важливу роль у створенні надійного додатку та його коректності на пристроях користувачів. Крім того, воно дозволяє переконатися, що додаток відповідає всім вимогам і очікуванням користувачів. Тестування має проводитись на кожному етапі розробки програми, починаючи з базових функцій і закінчуючи кінцевим оглядом отриманого програмного забезпечення. Це необхідно через великий обсяг різноманітних завдань, які потрібно протестувати, щоб уникнути пропусків деяких аспектів або умов, які можуть призвести до помилок.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Таким чином, протягом розробки програми було виявлено такі дефекти у її роботі:

1) Гравець перезавантажує сторінку, під час гри.

Очікуваний результат: Гравець, покидає кімнату та сповіщення приходить всім іншим.

Фактичний результат: Гравець покидає кімнату, проте сповіщення не приходить і гравець фактично в ній залишається. Через що виходять помилки при роботі програмного коду.

Цей баг виявив ряд недоліків у програмному коді, після усунення яких все запрацювало коректно.

2) Користувач намагається купити п'яту карту в руку, та отримує сповіщення після цього, про ліміт. Сповіщення залишається, проте сервер перекидує на етап двобою.

Очікуваний результат: Гравець переходить на етап двобою, та спостерігає.

Фактичний результат: користувач отримує екран про помилку.

Баг також проявив помилку в бібліотеці яку було використано: `toasify`. Виявилось, що компонент цієї бібліотеки треба використовувати якомога вище архітектурно. Баг було виправлено.

3) Користувач вводить ім'я менше ніж з 3 букв, та його пропускає.

Очікуваний результат: Гравець отримує помилку про довжину імені

Фактичний результат: його пропускає до гри.

Баг було виправленням, додаванням серверної перевірки на довжину імені.

Висновки до третього розділу

Під час проектування та розробки гри було продемонстровано отримані можливості програми з огляду користувача. Відповідно до врахованих особливостей програми було спроектовано діаграму варіантів використання системи. Також було протестовано програму на помилки та виявлено деякі несправності, які згодом були виправлені.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Турченко С.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

ВИСНОВКИ

Під час розробки кваліфікаційної роботи у формі онлайн веб-гри було набуто різноманітних теоретичних знань, які вдалося успішно застосувати на практиці. Використання технологій, що забезпечують одночасний зв'язок клієнтів, у поєднанні з іншими інструментами, дозволило створити унікальний продукт, доступний користувачам з будь-якої точки світу просто за допомогою браузера та Інтернету. Цей підхід забезпечив високий рівень зручності та доступності гри, що є ключовим фактором для сучасних користувачів, які цінують швидкість та простоту доступу до розважального контенту.

У першому розділі нашої роботи ми детально оцінили ринок аналогічних ігор, вивчаючи найсучасніші тенденції та технологічні можливості, які вони пропонують. Ми ретельно переглянули наявні рішення, склали порівняльні таблиці, що дозволило нам краще зрозуміти наші сильні та слабкі сторони. Це глибоке занурення в аналіз ринку допомогло визначити оптимальні підходи для нашого проекту, що, в свою чергу, сприяло розробці конкурентоспроможного продукту. Завдяки цьому аналізу ми змогли ідентифікувати ключові аспекти, які потрібно було врахувати для досягнення успіху.

В роботі використовувалися такі потужні інструменти, як мова програмування JavaScript та бібліотеки React і Socket.io, що значно сприяло швидкій і ефективній розробці гри. Вибір цих технологій дозволив нам не лише швидко створити функціональний прототип, але й легко масштабувати його функціонал у майбутньому. Аналізуючи завдання та існуючі ігри у першому розділі, було визначено ключові переваги, які ми успішно імплементували у наш продукт. Використання мікросервісної архітектури у другому розділі забезпечило стабільність і високу продуктивність програми, що є надзвичайно важливим для підтримки високої якості користувацького досвіду.

У другому розділі ми також детально визначили всі вимоги до нашого проекту - як функціональні, так і нефункціональні. Ці вимоги включали як

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Туренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

бізнес-потреби, так і потреби кінцевих користувачів, що дозволило нам створити продукт, максимально орієнтований на свого споживача. Відповідно до цих вимог було розроблено діаграми варіантів використання системи та послідовності дій, що дозволило чітко визначити взаємодії між різними компонентами програми. Це допомогло структурувати процес розробки та забезпечити чітке розуміння всіх етапів проекту як для команди розробників, так і для замовників.

У третьому розділі нашої роботи було продемонстровано можливості гри з точки зору користувача. Ми створили інтуїтивно зрозумілий інтерфейс та функціонал, що забезпечує зручне і приємне використання гри. Особливу увагу було приділено тестуванню програми на помилки, що дозволило виявити і виправити всі несправності перед завершенням проекту. Це тестування включало різноманітні сценарії використання, що допомогло нам переконатися у високій якості та надійності нашого продукту.

Завершена розробка програми підтверджує успішне поєднання теоретичних знань із практичними навичками, що виявилися критичними для створення сучасних інтерактивних ігор. Отриманий продукт повністю відповідає високим стандартам і вимогам сучасного ігрового ринку, готовий задовольнити потреби й очікування користувачів. Наш проект демонструє, як правильний вибір технологій, глибокий аналіз ринку та чітке визначення вимог можуть призвести до створення продукту, який має всі шанси стати популярним серед широкої аудиторії гравців.

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Three Drei[Електронний ресурс]- Режим доступу до ресурсу:
<https://www.npmjs.com/package/@react-three/drei/v/9.0.1>
2. React Three Fiber[Електронний ресурс]- Режим доступу до ресурсу:
<https://docs.pmnd.rs/react-three-fiber/getting-started/introduction>
3. TypeScript[Електронний ресурс]- Режим доступу до ресурсу:
<https://react.dev/learn/typescript>
4. Three-Types[Електронний ресурс]- Режим доступу до ресурсу:
<https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types/three>
5. React Toastify Documentation[Електронний ресурс]- Режим доступу до ресурсу: <https://www.npmjs.com/package/react-toastify>
6. Socket-io-client[Електронний ресурс]- Режим доступу до ресурсу:
<https://socket.io/docs/v4/client-api/>
7. Three[Електронний ресурс]- Режим доступу до ресурсу:
<https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene>
8. TypeScript for new programmer[Електронний ресурс]- Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>
9. Dotenv[Електронний ресурс]- Режим доступу до ресурсу:<https://www.npmjs.com/package/dotenv>
- 10.Express hello World[Електронний ресурс]- Режим доступу до ресурсу:
<https://expressjs.com/en/starter/hello-world.html>
- 11.Nodemon Documentation[Електронний ресурс]- Режим доступу до ресурсу: <https://www.npmjs.com/package/nodemon>
- 12.Socket.io Get Started [Електронний ресурс]- Режим доступу до ресурсу:
<https://socket.io/docs/v4/tutorial/introduction>
- 13.The WebSocket Api (WebSockets) [Електронний ресурс]- Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

14. Writing a WebSocket server in JavaScript [Електронний ресурс]- Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_a WebSocket_server_in_JavaScript_Deno
15. Emitting events [Електронний ресурс]- Режим доступу до ресурсу: <https://socket.io/docs/v4/tutorial/step-4>
16. Handling Disconnections Sockets [Електронний ресурс]- Режим доступу до ресурсу: <https://socket.io/docs/v4/tutorial/handling-disconnections>
17. Структура коду, Сучасний посібник з JavaScript [Електронний ресурс]- Режим доступу до ресурсу: <https://uk.javascript.info/structure>
18. JavaScript Game Development [Електронний ресурс]- Режим доступу до ресурсу: <https://www.codewizardshq.com/javascript-games/2>
19. 2D breakout game using pure JavaScript [Електронний ресурс]- Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_Breakout_game_pure_JavaScript
20. Javascript Promises [Електронний ресурс]- Режим доступу до ресурсу: https://developer.mozilla.org/docs/Web/JavaScript/Guide/Using_promises
21. The What, Why and How of a Microservices Architecture [Електронний ресурс]- Режим доступу до ресурсу: <https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9>
22. Microservices vs Monolit Architecture [Електронний ресурс]- Режим доступу до ресурсу: <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>
23. Game Development [Електронний ресурс]- Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Games>

ДОДАТКИ

		Мацкевич Р.Г.			ПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

```

import { createServer } from "http";
import { Server } from "socket.io";
import { getFullSocketByID } from "../utils.js";
import 'dotenv/config'
import { randomInt } from "crypto";
const PORT = 8080;

const httpServer = createServer();
const io = new Server(httpServer, {
  cors: {
    origin: 'http://localhost:3000'
  }
});

let players = {}

const queueHitInRoom = {}

const calculateAnimation = (currentBoard, enemyBoard, roomId) => {
  console.log('calculating animation')
  if (!Array.isArray(currentBoard) || !Array.isArray(enemyBoard)) {
    throw new Error("Invalid board data");
  }
  if (!queueHitInRoom.hasOwnProperty(roomId)) {
    queueHitInRoom[roomId] = Math.floor(Math.random() * 2) + 1;
  } else {
    queueHitInRoom[roomId]++;
  }
  const currentPlayerBoardAfterHit = JSON.parse(JSON.stringify(currentBoard));
  const enemyPlayerBoardAfterHit = JSON.parse(JSON.stringify(enemyBoard));
  const hitterPos = 1

  const tauntIndices = enemyPlayerBoardAfterHit
    .map((card, index) => card.taunt ? index : -1)
    .filter(index => index !== -1);

  let emitterPos;
  if (tauntIndices.length > 0) {
    // Randomly select a taunt if taunts are present
    emitterPos = tauntIndices[Math.floor(Math.random() * tauntIndices.length)] + 1;
  } else {
    // Select a random enemy card if no taunts are present
    emitterPos = Math.floor(Math.random() * enemyPlayerBoardAfterHit.length) + 1;
  }
  if (currentBoard.length === 0 && enemyBoard.length === 0) {
    return {
      fightStatus: "draw",
    }
  }
}

```

		Мацкевич Р.Г.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Турченко Є.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

```

        collectedDamage: 0
    }
}

if (currentBoard.length === 0) {
    const collectedDamage = enemyBoard.reduce((accumulator, item) => {
        return accumulator + item.tavern_tier;
    }, 0)
    return {
        fightStatus: "you lose",
        collectedDamage
    }
}

if (enemyBoard.length === 0) {
    const collectedDamage = currentBoard.reduce((accumulator, item) => {
        return accumulator + item.tavern_tier;
    }, 0)
    return {
        fightStatus: "you win",
        collectedDamage
    }
}

currentPlayerBoardAfterHit[hitterPos - 1].health -=
Number(enemyPlayerBoardAfterHit[emitterPos - 1].damage);
enemyPlayerBoardAfterHit[emitterPos - 1].health -=
Number(currentPlayerBoardAfterHit[hitterPos - 1].damage);
if (currentPlayerBoardAfterHit[hitterPos - 1].health <= 0) {
    currentPlayerBoardAfterHit.splice(hitterPos - 1, 1);
}
if (enemyPlayerBoardAfterHit[emitterPos - 1].health <= 0) {
    enemyPlayerBoardAfterHit.splice(emitterPos - 1, 1);
}
return {
    currentPlayerBoardAfterHit,
    enemyPlayerBoardAfterHit,
    hitterPos,
    emitterPos,
    ownTurn: queueHitInRoom[roomId] % 2 !== 0
}
}

let playersByRoomId = {}
const generateNewRoomId = (roomId) => {

    if (playersByRoomId.hasOwnProperty(roomId)) {
        playersByRoomId[roomId]++
    } else {

```

		Мацкевич Р.Г.		
		Гуренко Є.Є.		
Змн.	Арк.	№ докум.	Підпис	Дата

```

    playersByRoomId[roomId] = 1
  }

  if (playersByRoomId[roomId] <= 2) {
    return roomId + '-' + "1"
  } else {
    return roomId + '-' + "2"
  }
}
const playersObject = {};

const reverseFightStatus = {
  "you win": "you lose",
  "you lose": "you win",
  "draw": "draw"
}

io.on('connect', (socket) => {
  socket.on('newPlayer', data => {
    console.log('New client connected, with id: ' + socket.id);

    players[socket.id] = data;
    if (!data.info.roomName) {
      return 'error'
    }
    const newRoomId = generateNewRoomId(data.info.roomName)
    console.log(newRoomId);
    socket.join(newRoomId)
    players[socket.id].roomId = newRoomId
    const socketsInRoom = Array.from(io.sockets.adapter.rooms.get(newRoomId) ||
    []);
    console.log(socketsInRoom)
    let enemySocket = {}
    if (socketsInRoom.length === 2) {
      for (const socketId of socketsInRoom) {
        if (socketId !== socket.id) {
          enemySocket = getFullSocketByID(io, socketId)
          const enemyData = players[socketId].info.hero
          const ownData = players[socket.id].info.hero
          socket.emit('enemyData', enemyData)
          enemySocket.emit('enemyData', ownData)
          enemySocket.emit('currentBoard', players[socketId])
          enemySocket.emit('enemyBoard', data)
          socket.emit('currentBoard', data)
          socket.emit('enemyBoard', players[socketId])
          const ownAnimationData = calculateAnimation(data.board,
          players[socketId].board)
          const enemyAnimationData = {
            currentPlayerBoardAfterHit: ownAnimationData.enemyPlayerBoardAfterHit,
            enemyPlayerBoardAfterHit: ownAnimationData.currentPlayerBoardAfterHit,

```

		Мацкевич Р.Г.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Гуренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

```

        hitterPos: ownAnimationData.emitterPos,
        emitterPos: ownAnimationData.hitterPos,
        ownTurn: !ownAnimationData.ownTurn
    }
    if (ownAnimationData.hasOwnProperty('fightStatus')) {
        socket.emit('endBattle', ownAnimationData)
        const enemyAnimationData = {
            fightStatus: reverseFightStatus[ownAnimationData.fightStatus],
            collectedDamage: ownAnimationData.collectedDamage
        }
        enemySocket.emit('endBattle', enemyAnimationData)
        socket.emit('enemyData', null)
        enemySocket.emit('enemyData', null)
    } else {
        socket.emit('animation', ownAnimationData)
        enemySocket.emit('animation', enemyAnimationData)
    }
}
}
}
});
socket.on('calculateNewHit', (boards) => {
    const roomId = players[socket.id].roomId
    const { currentBoard, enemyBoard } = boards
    let enemySocket = {}
    if (!playersObject.hasOwnProperty(roomId)) {
        playersObject[roomId] = 1
    } else {
        playersObject[roomId]++
    }
    const socketsInRoom = Array.from(io.sockets.adapter.rooms.get(roomId) || []);
    if (playersObject[roomId] === 2 || playersObject[roomId] === 4) {
        for (const socketId of socketsInRoom) {
            if (socketId !== socket.id) {
                enemySocket = getFullSocketByID(io, socketId)
                const ownAnimationData = calculateAnimation(currentBoard, enemyBoard)
                const enemyAnimationData = {
                    currentPlayerBoardAfterHit: ownAnimationData.enemyPlayerBoardAfterHit,
                    enemyPlayerBoardAfterHit: ownAnimationData.currentPlayerBoardAfterHit,
                    hitterPos: ownAnimationData.emitterPos,
                    emitterPos: ownAnimationData.hitterPos,
                    ownTurn: !ownAnimationData.ownTurn
                }
                if (ownAnimationData.hasOwnProperty('fightStatus')) {
                    socket.emit('endBattle', ownAnimationData)
                    const enemyAnimationData = {
                        fightStatus: reverseFightStatus[ownAnimationData.fightStatus],
                        collectedDamage: ownAnimationData.collectedDamage
                    }
                }
            }
        }
    }
});

```

```

        enemySocket.emit('endBattle', enemyAnimationData)
        socket.emit('enemyData', null)
        enemySocket.emit('enemyData', null)
    } else {
        socket.emit('animation', ownAnimationData)
        enemySocket.emit('animation', enemyAnimationData)
    }
    }
    }
    }
    })

socket.on('leave', () => {
    console.log(`Player left, with id: ${socket.id}`);
    const player = players[socket.id]
    if(!player) return
    const roomName = player.roomId;

    socket.leave(roomName);
    delete playersByRoomId[players[socket.id].info.roomName]
    delete playersObject[roomName];
    delete queueHitInRoom[roomName];
    delete players[socket.id];
});

socket.on('disconnect', () => {
    delete players[socket.id];
    console.log(`Socket: ${socket.id} disconnected`)
});

});

httpServer.listen(PORT, () => {
    console.log(`listen on port: ${PORT}`)
})

public override ControlType GetControlType()
{
    return ControlType.Player;
}

protected override void Start()
{
    base.Start();
    SetView((PlayerDefaultView)Kernel.DisplayManager.GetPlayerView());

    if (IsViewExists())
    {
        //Events
        View.joystick.OnPressed.AddListener(OnJoystickPress);
        View.joystick.OnReleased.AddListener(OnJoystickRelease);
    }
}
}

```

```

import { createServer } from "http";
import { Server } from "socket.io";
import Player from "../player.js";
import 'dotenv/config';
import { shuffleArray, getRandomCardsByLevel, readCardsData } from "../utils.js";
import { levelUpPrice, cardsCountPullByLevel, rooms, heroes } from "../const.js";
import { randomInt } from "crypto";
const httpServer = createServer();
const io = new Server(httpServer, {
  cors: {
    origin: 'http://localhost:3000'
  }
});

let players = {};
const queueSet = new Set();
let emptyRooms = rooms.filter(room => room.empty);
let availableHeroes = [...heroes];
const cards = readCardsData('../data/cards.json');
let playersByRoomInPick = {}

io.on('connect', (socket) => {
  let battleStatus
  let turn = 0;
  socket.on('battle-end', () => {
    battleStatus = false
  })
  socket.on('findGame', (data) => {
    if (players[socket.id]) return
    if (data.name.length < 3) {
      socket.emit('warning', 'Ім'є;я повинне бути щонайменше з 3 букв')
      return
    }
    players[socket.id] = new Player(data.name);
    io.emit('updatePlayers', players);
    battleStatus = true
    turn = 0;
    if (queueSet.has(players[socket.id])) {
      socket.emit('loading', 'finding game');
    } else {
      socket.emit('loading', 'finding game');
      queueSet.add(socket.id);
      if (queueSet.size == process.env.PLAYERS_COUNT) {
        const emptyRoom = emptyRooms.shift();
        queueSet.forEach(socketId => {
          const player = players[socketId];
          if (player) {

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Туренко Є.С.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        player.info.roomName = emptyRoom.name;
        emptyRoom.empty = false;
        emptyRoom.startTime = Date.now();

        const playerSocket = io.sockets.sockets.get(socketId);
        if (playerSocket) {
            playerSocket.emit('loading', 'recruting');
            playerSocket.join(emptyRoom.name);

            shuffleArray(availableHeroes);
            const selectedHeroes = availableHeroes.slice(0,
Number(process.env.PLAYERS_COUNT));
            playerSocket.emit('selectHero', { heroes: selectedHeroes });
        }
    }
    });
    queueSet.clear();
}
}
});

socket.on('heroPick', data => {
    turn = 1;
    players[socket.id].info.hero = data;
    if (players[socket.id].info.hero.name === "Бакабот") {
        players[socket.id].info.levelUpCost--;
    }
    socket.emit('loading', 'waiting another players');
    const roomName = players[socket.id].info.roomName;
    const playersInRoom = Object.values(players).filter(player =>
player.info.roomName === roomName);
    const allPlayersPicked = playersInRoom.every(player => player.info.hero);
    if (allPlayersPicked) {
        io.to(roomName).emit('loading', 'starting game...');
        const socketIdsInRoom = Array.from(io.sockets.adapter.rooms.get(roomName) ||
[]);
        const playersInfo = socketIdsInRoom.map(socketId => {
            const player = players[socketId];
            return {
                name: player.info.name,
                hero: player.info.hero
            };
        });
        io.to(roomName).emit('players', playersInfo);
    }
});

socket.on('game', () => {
    const player = players[socket.id];
    if (!player) return;

```

		Мацкевич Р.Г.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Гуренко Є.Є.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const roomName = player.info.roomName;
const socketIdsInRoom = Array.from(io.sockets.adapter.rooms.get(roomName) ||
[]);

if (player.info.hero.hp <= 0) {
    socket.leave(roomName);
    delete players[socket.id];

    if (socketIdsInRoom.length === 1) {
        const winnerSocket = io.sockets.sockets.get(socketIdsInRoom[0]);
        winnerSocket.leave(roomName);
        delete players[winnerSocket.id];
        winnerSocket.emit('end-battle', "Ти переможець над леймами, вітаю!!!");
        socket.emit('end-battle', "Ти нажаль програв, можливо повезе наступного
разу");
        return;
    } else if (socketIdsInRoom.length > 1) {
        const remainingSocketIds =
Array.from(io.sockets.adapter.rooms.get(roomName) || []);
        if (remainingSocketIds.length === 1) {
            const remainingPlayerSocket =
io.sockets.sockets.get(remainingSocketIds[0]);
            remainingPlayerSocket.leave(roomName);
            delete players[remainingPlayerSocket.id];
            remainingPlayerSocket.emit('end-battle', "Ти переможець над леймами,
вітаю!!!");
        }
    }

    socket.emit('end-battle', "Ти нажаль програв, можливо повезе наступного
разу");

    const playersInfo = socketIdsInRoom.map(socketId => {
        const player = players[socketId];
        if (!player) return;
        return {
            name: player.info.name,
            hero: player.info.hero
        };
    }).filter(Boolean);
    io.to(roomName).emit('players', playersInfo);
    return;
}

if (playersByRoomInPick.hasOwnProperty(roomName)) {
    playersByRoomInPick[roomName]++;
} else {
    playersByRoomInPick[roomName] = 1;
}

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Турченко Є.Є.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if (playersByRoomInPick[roomName] == process.env.PLAYERS_COUNT) {
  let countdown = 10;
  const intervalId = setInterval(() => {
    if (battleStatus) {
      if (countdown > 0) {
        io.to(roomName).emit('ticker', countdown);
        countdown--;
      } else {
        clearInterval(intervalId);
        delete playersByRoomInPick[roomName];
        io.to(roomName).emit('loading', 'fight');
      }
    } else {
      clearInterval(intervalId);
    }
  }, 1000);
}

player.info.gold = 100;
player.cardsPulled = player.isBoardFreeze ? player.cardsPulled :
getRandomCardsByLevel(cards.cards, player.info.level, cardsCountPullByLevel);
player.hand = player.hand || [];
player.board = player.board || [];
socket.emit('updateBoardAndHand', player);
});

socket.on('nextRoundPick', ({ damage, status }) => {
  const player = players[socket.id];
  if (!player) return;

  turn++;
  if (player.info.levelUpCost > 0) {
    player.info.hero.abilityTurnsDisable--;
  }
  if (player.info.levelUpCost >= 0) {
    player.info.levelUpCost--;
  }
  if (isNaN(damage)) {
    return;
  }
  if (status === 'you lose') {
    player.info.hero.hp -= (damage + 30);
  }

  socket.emit('loading', 'starting game...');
  const roomName = player.info.roomName;
  const socketIdsInRoom = Array.from(io.sockets.adapter.rooms.get(roomName) ||
[]);
  const playersInfo = socketIdsInRoom.map(socketId => {

```

```

    const player = players[socketId];
    return {
      name: player.info.name,
      hero: player.info.hero
    };
  });
  io.to(roomName).emit('players', playersInfo);
});

```

```

socket.on('freeze', () => {
  const player = players[socket.id];
  player.isBoardFreeze = !player.isBoardFreeze;
  socket.emit('updateBoardAndHand', player);
});

```

```

socket.on('levelUp', () => {
  const player = players[socket.id];
  if (player.info.levelUpCost > player.info.gold) {
    socket.emit('warning', 'You need more money');
    return;
  }
  if (player.info.level >= process.env.MAX_LEVEL) {
    socket.emit('warning', 'Max level');
    return;
  }
  player.info.gold -= player.info.levelUpCost;
  player.info.level++;
  player.info.levelUpCost = levelUpPrice[player.info.level];
  if (player.info.hero.name === "Бакабот") {
    player.info.levelUpCost--;
  }
  socket.emit('updateBoardAndHand', player);
});

```

```

socket.on('roll', () => {
  const player = players[socket.id];
  const rollCost = player.info.hero.name === "Манашторм" ? 2 : 1;
  if (player.info.gold > (rollCost - 1)) {
    player.info.gold -= rollCost;
    player.cardsPulled = getRandomCardsByLevel(cards.cards, player.info.level,
cardsCountPullByLevel);
    socket.emit('updateBoardAndHand', player);
  } else {
    socket.emit('warning', 'Недостатньо золота')
  }
});

```

```

socket.on('buyCard', ({ cardToBuy, index }) => {

```

		Мацкевич Р.Г.		
		Туренко С.С.		
Змн.	Арк.	№ докум.	Підпис	Дата

```

const player = players[socket.id];
const cardCost = player.info.hero.name === "Манашторм" ? 2 : 3;

if (player.info.gold < cardCost) {
    socket.emit('warning', 'Недостатньо коштів');
    return;
}

if (player.hand.length >= 4) {
    socket.emit('warning', 'Максимум карт в руці: 4');
    return;
}

if (player.cardsPulled[index] && player.cardsPulled[index].name ===
cardToBuy.name) {
    player.hand.push(cardToBuy);
    player.cardsPulled.splice(index, 1);
    player.info.gold -= cardCost;
    socket.emit('updateBoardAndHand', player);
} else {
    socket.emit('warning', 'Ця карта недоступна для покупки');
}
});

socket.on('placeCard', ({ cardToPlace, index }) => {
    const player = players[socket.id];
    if (player.board.length > 0) {
        const randomIndex = Math.floor(Math.random() * player.board.length);
        const targetCard = player.board[randomIndex];
        if (cardToPlace.ability === 'інкріз-2/2') {
            targetCard.damage += 2;
            targetCard.health += 2;
        }
        if (cardToPlace.ability === 'інкріз-10/10') {
            targetCard.damage += 10;
            targetCard.health += 10;
        }
    }
    if (player.board.length >= 7) {
        socket.emit('warning', 'Стіл заповнений');
        return;
    }
    if (player.hand[index] && player.hand[index].name === cardToPlace.name) {
        player.board.push(cardToPlace);
        player.hand.splice(index, 1);
        socket.emit('updateBoardAndHand', player);
    } else {
        socket.emit('warning', 'В тебе нема цієї карти в руці');
    }
});

```

Мацкевич Р.Г.

Туренко Є.С.

ІПЗ.КР.Б – 121 – 24 – ІЗ

Арк.

75

Змн.	Арк.	№ докум.	Підпис	Дата
------	------	----------	--------	------

```

socket.on('sellCard', ({ cardToSell, index }) => {
  const player = players[socket.id];
  if (player.board[index] && player.board[index].name === cardToSell.name) {
    player.board.splice(index, 1);
    player.info.gold++;
    socket.emit('updateBoardAndHand', player);
  } else {
    socket.emit('warning', 'В тебе нема цієї карти');
  }
});

socket.on('ability', () => {
  const player = players[socket.id];
  if (player.info.hero.abilityTurnsDisable > 0) {
    socket.emit('warning', 'Ти вже використав');
    return
  }
  if (player.info.gold < player.info.hero.abilityCost) {
    socket.emit('warning', 'Недостатньо грошей');
    return
  }
  player.info.gold -= player.info.hero.abilityCost
  if (player.info.hero.name === "Токі") {
    player.cardsPulled = getRandomCardsByLevel(cards.cards, player.info.level,
    cardsCountPullByLevel);
    const tierCards = cards.cards.filter(card => card.tavern_tier === (1 +
    player.info.level));
    player.cardsPulled.push(tierCards[randomInt(0, tierCards.length)]);
    player.info.hero.abilityTurnsDisable = 1;
    socket.emit('updateBoardAndHand', player);
    return
  }

  if (player.info.hero.name === "Піраміда") {
    if (player.board.length < 1) {
      socket.emit('warning', 'Стіл пустий');
      return
    }

    player.board[randomInt(0, player.board.length)].health += (2 *
    player.info.level);
    player.info.hero.abilityTurnsDisable = 1;
    socket.emit('updateBoardAndHand', player);
    return
  }

  if (player.info.hero.name === "Рапан") {
    if (player.board.length < 1) {
      socket.emit('warning', 'Стіл пустий');

```

		Мацкевич Р.Г.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Туренко Є.Є.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

```

        return
    }
    const randomCardIndex = randomInt(0, player.board.length)
    player.board[randomCardIndex].health += player.info.level;
    player.board[randomCardIndex].damage += player.info.level;
    player.info.hero.abilityTurnsDisable = 1;
    socket.emit('updateBoardAndHand', player);
    return
}

if (player.info.hero.name === "Кубанчик") {
    const gold = randomInt(1, 6);
    socket.emit('warning', gold);
    player.info.hero.abilityTurnsDisable = gold;
    player.info.gold += gold;
    socket.emit('updateBoardAndHand', player);
    return
}

if (player.info.hero.name === "Ноздормус") {
    player.cardsPulled = getRandomCardsByLevel(cards.cards, player.info.level,
cardsCountPullByLevel);
    player.info.hero.abilityTurnsDisable = 1
    socket.emit('updateBoardAndHand', player);
    return
}
});

socket.on('disconnect', () => {
    queueSet.delete(players[socket.id]);
    delete players[socket.id];
    io.emit('updatePlayers', players);
    queueSet.delete(socket.id);
});
});

httpServer.listen(process.env.PORT, () => {
    console.log(`Listening on port: ${process.env.PORT}`);
});

```