Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка гри стратегії на тематику "інженерія програмного забезпечення" з використанням фреймворків Skeleton та Alpine

Виконав: студент 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)
(підпис) Бондар М.Д.
(прізвище та ініціали)

Керівник (підпис) Стоянов Ю.М.
(прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю.М.
(прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М.Р.
(прізвище та ініціали)
Рецензент (підпис) Готович .В.А.
(прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Атестаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025. Сторінок 75, рисунків 28 джерел 28, презентація.

Тема: Розробка веб-застосунку для настільної багатокористувацької гри з інтерактивним ігровим полемВ бакалаврській кваліфікаційній роботі представлено процес проєктування та реалізації клієнт-серверного веб-застосунку для гри в реальному часі. Основна увага приділена розробці зручного інтерфейсу користувача, реалізації логіки гри, а також забезпеченню синхронізації дій між гравцями.Серверна частина реалізована з використанням платформи Node.js та веб-фреймворку Express.js. Для забезпечення двосторонньої взаємодії в реальному часі між клієнтами та сервером застосовано бібліотеку Socket.IO.

Клієнтська частина розроблена з використанням легкого JavaScript-фреймворку Alpine.js, який забезпечує реактивність інтерфейсу та зручну роботу з DOM-елементами. Для створення сучасного та адаптивного дизайну використано CSS-фреймворк Skeleton, що дозволило реалізувати чітку сіткову систему та стилізувати компоненти інтерфейсу.Ігрове поле створене на основі SVG-графіки, що дає змогу динамічно змінювати його відповідно до дій користувачів.

Для реалізації візуальних ефектів, таких як анімація руху фішок та кидок кубиків, використано CSS-анімації. Alpine.js дозволив реалізувати інтерактивні елементи керування та динамічне оновлення стану гри без перезавантаження сторінки.Проведено тестування функціональності застосунку та верифікацію його коректної роботи в умовах багатокористувацької взаємодії.

Ключові слова: веб-застосунок, Node.js, Socket.IO, Alpine.js, Skeleton, SVG, CSS-анімації, багатокористувацька гра, клієнт-серверна архітектура

ABSTRACT

Bachelor's degree thesis. Ivan Pulyuy Ternopil National Technical University, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2025. 75 pages, 28 figures, sources 28, presentation.

Topic: Development of a web application for a multiplayer board game with an interactive playing fieldThe bachelor's degree thesis presents the process of designing and implementing a client-server web application for a real-time game. The main attention is paid to the development of a convenient user interface, the implementation of the game logic, as well as ensuring synchronization of actions between players. The server part is implemented using the Node.js platform and the Express.js web framework. To ensure two-way interaction in real time between clients and the server, the Socket.IO library is used.

The client part is developed using the lightweight JavaScript framework Alpine.js, which provides interface reactivity and convenient work with DOM elements. To create a modern and adaptive design, the Skeleton CSS framework was used, which allowed to implement a clear grid system and stylize interface components. The playing field is created based on SVG graphics, which allows it to be dynamically changed according to user actions.

CSS animations were used to implement visual effects, such as animation of chip movement and dice rolling. Alpine.js allowed to implement interactive controls and dynamic updating of the game state without reloading the page. The application functionality was tested and its correct operation was verified in multi-user interaction.

Keywords: web application, Node.js, Socket.IO, Alpine.js, Skeleton, SVG, CSS animations, multi-user game, client-server architecture

ЗМІСТ

АНОТАЦІЯ 4
ABSTRACT 5
ЗМІСТ 6
ВСТУП 7
1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 8
1.1 Огляд конкурентів 8
1.2 Обґрунтування вибору напрямку дослідження 9
1.3 Технічний аспект проблеми 10
2. РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ 12
2.1 Обґрунтування вибору середовища розробки програмної системи 12
2.1.1. Розробка моделі предметної області 15
2.1.2 Розробка бізнес моделі 21
2.1.3. Проєктування архітектури 25
2.1.4 Аналіз та обґрунтування вибору процедурного підходу для реалізації проєкту 26
2.1.5 Специфікація вимог до програмного забезпечення (SRS) 28
2.2.3. Розробка GUI 34
2.2.4. Тестування програмного забезпечення та оцінка якості 35
2.2.5 Результати розробки 41
3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 49
1.10 Психологічні чинники небезпеки 49
4.30 Протипожежні вимоги до робочого освітлення 50
ВИСНОВКИ 50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 54
ДОДАТКИ 57
ДОДАТОК А 58
ДОДАТОК Б. Тези конференції 76
ДОДАТОК В. Диск з роботою 78

ВСТУП

Сучасний розвиток веб-технологій відкриває широкі можливості для створення інтерактивних застосунків, зокрема в галузі онлайн-ігор. Завдяки стрімкому зростанню популярності браузерних ігор, особливо з функціональністю реального часу, виникає потреба у створенні ефективних, динамічних та зручних для користувача рішень, що забезпечують комфортну взаємодію між гравцями без необхідності встановлення додаткового програмного забезпечення.

Актуальність теми зумовлена постійним зростанням попиту на багатокористувацькі ігрові рішення, які працюють безпосередньо в браузері та підтримують обмін даними в реальному часі. Такі рішення вимагають ефективної організації клієнт-серверної архітектури, використання сучасних технологій синхронізації, а також зручного графічного інтерфейсу.

Метою даної бакалаврської роботи є розробка веб-застосунку для багатокористувацької настільної гри, яка забезпечує інтерактивну взаємодію гравців через Інтернет. Основними завданнями є:

- проєктування та реалізація серверної частини застосунку з використанням Node.js та Express.js;
- реалізація обміну даними між клієнтами в реальному часі за допомогою Socket.IO;
- створення графічного інтерфейсу з використанням HTML, CSS, SVG та JavaScript;
- впровадження візуальних ефектів для покращення ігрового досвіду;
- тестування застосунку в умовах реального використання.

Об'єктом дослідження є процес взаємодії клієнтів у багатокористувацькому середовищі.

Предметом дослідження — програмні засоби та технології для реалізації багатокористувацького веб-застосунку.

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

Сфера настільних ігор в онлайн-форматі активно розвивається. Все більше користувачів обирають можливість грати в улюблені ігри прямо у браузері, без встановлення додаткових програм. Це створює попит на зручні, доступні та водночас функціональні веб-застосунки.

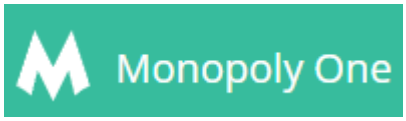

Рис. 1.1 – Логотип конкурента Monopoly One

Одним із прикладів подібного рішення є Monopoly One — онлайн-адаптація класичної "Монополії". Гра реалізована як веб-додаток, у якому користувачі можуть створювати кімнати, приєднуватися до ігор з друзями або з випадковими учасниками, кидати кубики, здійснювати покупки та спілкуватися в чаті. Весь процес відбувається в режимі реального часу, а інтерфейс інтуїтивно зрозумілий і приємний на вигляд. Простота доступу — через браузер — робить гру зручною навіть для тих, хто не має технічного досвіду.

Monopoly One приваблює передусім своєю знайомістю та тим, що дозволяє відтворити атмосферу настільної гри в онлайн-середовищі. Користувачі цінують можливість швидко зібрати компанію, почати гру та провести час у невимушеному форматі.

У рамках моєї роботи реалізується подібний підхід — багатокористувацька настільна гра, яка працює в браузері, підтримує взаємодію гравців у реальному часі та візуалізує ігрове поле. Водночас у розробці акцент зроблено на гнучкість і адаптивність: застосунок проєктувався так, щоб його можна було легко змінювати — додавати нову логіку гри, стилі чи навіть повністю переробити правила.

1.2 Обґрунтування вибору напрямку дослідження

При розробці багатокористувацької настільної гри було прийнято рішення створити веб-застосунок, доступ до якого здійснюється напряму з браузера. Це дозволяє зробити гру максимально зручною для користувача — без потреби щось встановлювати, з гарантованою підтримкою на більшості сучасних пристроїв.

Вибір основної мови програмування зупинився на JavaScript [6], оскільки вона є стандартом для веб-розробки як на клієнтській, так і на серверній стороні. Завдяки Node.js стало можливим використовувати JavaScript і для створення серверної логіки, що значно спростило структуру проєкту: одна мова — одна екосистема.

Клієнтська частина побудована з використанням HTML та CSS, включно з CSS-анімаціями, які відповідають за візуальні ефекти в грі (рух фішок, кидок кубиків тощо). Для базового стилювання застосовано легкий CSS-фреймворк Skeleton, що дозволив швидко сформувати адаптивну й охайну верстку без надмірностей.

Для додавання реактивної поведінки елементів інтерфейсу на клієнтській стороні використано Alpine.js [3] — невеликий JavaScript-фреймворк, який дає змогу реалізовувати інтерактивну поведінку без складності повноцінних фреймворків на кшталт React чи Vue. Alpine ідеально підходить для простих, але динамічних сценаріїв, таких як показ повідомлень, перемикання станів інтерфейсу чи обробка натискань.

Сервер реалізовано на базі Node.js [1], з використанням Express.js [5] — популярного веб-фреймворку, який дозволяє зручно обробляти HTTP-запити, налаштовувати маршрутизацію та інтегрувати сторонні модулі.

Для синхронізації дій гравців у реальному часі використовується Socket.IO [2] — бібліотека, яка забезпечує двосторонній обмін повідомленнями між клієнтом і сервером. Це дає змогу досягти миттєвого оновлення ігрового поля в усіх учасників гри без необхідності ручного перезавантаження сторінки.

Графічне представлення поля реалізоване за допомогою SVG [7], що дозволяє створювати масштабовану, якісну векторну графіку з можливістю динамічного керування окремими елементами через JavaScript.

1.3 Технічний аспект проблеми

Реалізація багатокористувацької гри в режимі реального часу — це не лише питання логіки гри, а й цілий комплекс технічних викликів. Ключова складність полягає в тому, що всі дії гравців мають бути синхронізовані миттєво й узгоджено: якщо один учасник кидає кубики або пересуває фішку, всі інші повинні бачити зміни практично без затримок.

Це вимагає стабільного двостороннього з’єднання між клієнтом і сервером, що реалізується за допомогою Socket.IO [2]. На відміну від класичних HTTP-запитів, які працюють за принципом "клієнт запитує — сервер відповідає", WebSocket [8] -з'єднання дозволяє як серверу, так і клієнту ініціювати передачу даних у будь-який момент. Це критично важливо для забезпечення живої, інтерактивної гри без пауз і оновлень сторінки.

Ще один технічний момент — зберігання та обробка стану гри. Сервер повинен тримати в пам’яті актуальну інформацію про гравців, позиції фішок, стан клітинок, черговість ходу та інші параметри. Це вимагає ретельно продуманої моделі даних і системи керування подіями, щоб уникнути розсинхронізації чи некоректних дій при одночасному надходженні запитів від кількох користувачів.

Графічна частина також має свою специфіку. Використання SVG як засобу відображення ігрового поля дає широкі можливості для взаємодії з векторними елементами. Проте воно вимагає більш детального опрацювання координат, масштабування та анімацій. З цим завданням добре справляється комбінація

JavaScript, CSS-анімацій і бібліотек на кшталт Alpine.js, яка дозволяє легко керувати станом інтерфейсу без надмірної складності.

Окремо варто згадати виклики, пов’язані з адаптивністю інтерфейсу. Оскільки користувачі можуть запускати гру на різних пристроях — від десктопів до смартфонів, — важливо забезпечити коректне відображення та керування грою в різних умовах. Для цього застосовано легкий CSS-фреймворк Skeleton, що дозволяє створювати адаптивні макети без зайвого навантаження на сторінку.

У підсумку, технічна реалізація проєкту поєднує вимоги до швидкодії, синхронізації, стабільності та зручності інтерфейсу. Кожне з цих завдань вирішується за допомогою відповідних інструментів, підібраних так, щоб система залишалась легкою, гнучкою та придатною до масштабування.

2. РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Обґрунтування вибору середовища розробки програмної системи

При розробці онлайн-версії гри "Монополія" постало питання вибору оптимального середовища розробки, яке б дозволило створити стабільну багатокористувацьку гру з можливістю взаємодії в реальному часі.Спочатку було розглянуто кілька варіантів технологій для серверної частини: PHP з WebSocket розширенням, Java з Spring Framework, Python з Django та Node.js.

Рис. 2.1 – Лого Node.js

Після тестування та оцінки цих варіантів, перевагу отримав Node.js через кілька важливих особливостей.Головна причина вибору Node.js – його асинхронна природа та здатність обробляти багато одночасних з'єднань без створення окремих потоків для кожного клієнта. Це критично важливо для гри, де до 24 гравців можуть одночасно взаємодіяти, а сервер повинен миттєво обробляти їхні дії та синхронізувати стан гри.Крім того, використання JavaScript як на сервері, так і на клієнті значно спрощує розробку. Можна використовувати ті самі структури даних та функції в обох частинах програми, що скорочує час розробки і зменшує кількість помилок при передачі даних між клієнтом і сервером.

Рис. 2.2 – Лого Express.js

Для веб-сервера було обрано Express.js – простий, але потужний фреймворк для Node.js. Вибір обумовлений тим, що Express дозволяє швидко налаштувати маршрутизацію та обробку запитів. До того ж, більшість хостинг-провайдерів добре підтримують Node.js з Express, що спрощує розгортання проекту.

Рис. 2.3 – Лого Socket.IO

Найскладнішою частиною розробки була організація комунікації в реальному часі між гравцями. При експериментах з чистими WebSocket виникли проблеми сумісності в деяких браузерах та корпоративних мережах, де WebSocket часто блокуються. Тому було вирішено використати бібліотеку Socket.IO, яка автоматично підбирає найкращий доступний транспорт для з'єднання. Це рішення відмінно спрацювало – гра стабільно працює навіть на повільних з'єднаннях, а гравці не відчувають затримок.Для клієнтської частини було вирішено не використовувати важкі фреймворки на кшталт React чи Angular. Справа в тому, що гра "Монополія" має досить специфічний інтерфейс з ігровим полем та фішками, і використання компонентного підходу не дало б значної переваги. Натомість ставку зроблено на чистий JavaScript з мінімальними залежностями,

що дозволило зменшити розмір клієнтського додатка та прискорити його завантаження.Для відображення ігрового поля було випробувано різні підходи: звичайні HTML елементи з абсолютним позиціонуванням, Canvas та SVG.

Після тестування вибір зупинився на SVG, оскільки він дозволяє легко маніпулювати окремими елементами гри (фішками, будинками) та анімувати їх, а також добре масштабується на різних екранах. Це виявився правильний вибір – гра однаково добре виглядає і на великих моніторах, і на планшетах.Важливим аспектом був вибір інструментів для розробки та відлагодження.

Робота велася у Visual Studio Code [11] через його хорошу підтримку JavaScript та Node.js.Особливо корисною виявилася можливість налагодження як клієнтського, так і серверного коду в одній IDE.Для налагодження WebSocket комунікації були створені спеціальні утиліти, які відображають всі повідомлення між клієнтом і сервером у консолі. Це дуже допомогло при пошуку помилок у логіці гри.Під час розробки виникло кілька проблем. Наприклад, при втраті з'єднання гравця його стан в грі "зависав", що впливало на всіх інших учасників. Проблему вдалося вирішити шляхом зберігання стану гри на сервері та автоматичного відновлення з'єднання. Це вимагало додаткової роботи, але значно покращило користувацький досвід.Для тестування спочатку використовувалося ручне тестування з кількома браузерами, але швидко стало зрозуміло, що цього недостатньо. Тому було додано автоматизовані тести з використанням Mocha та Chai [10], що дозволило швидше виявляти регресії при додаванні нових функцій.Розгортання проекту виявилося простішим, ніж очікувалося, завдяки використанню Docker [9]. Було створено контейнер з усією необхідною інфраструктурою, що дозволило легко переносити проект між середовищами розробки та виробництва.Загалом, вибір Node.js з Express та Socket.IO для серверної частини та HTML5/CSS3/JavaScript з SVG для клієнтської частини виявився оптимальним для розробки онлайн-гри "Монополія". Ця комбінація забезпечила достатню продуктивність, гнучкість та зручність розробки

2.1.1. Розробка моделі предметної області

Розробка моделі предметної області для програмного комплексу онлайн-гри "Монополія" є ключовим етапом проектування системи, який визначає основні сутності, їх властивості та взаємозв'язки. Модель предметної області служить фундаментом для подальшої реалізації програмного коду та забезпечує єдине розуміння структури системи.При аналізі предметної області гри "Монополія" було виділено основні сутності, які формують ядро ігрової механіки. Ключовими компонентами є гравці, ігрове поле з клітинками різних типів, фішки, гральні кубики, картки власності, ігрова валюта та торгові угоди між гравцями.Сутність "Гравець" представляє користувача системи, який бере участь у грі. Кожен гравець має унікальний ідентифікатор, ім'я, поточний баланс ігрової валюти та набір володінь. Також гравець асоціюється з фішкою певного кольору, яка представляє його позицію на ігровому полі. В рамках гри гравець може виконувати різні дії: кидати кубики, купувати та продавати власність, пропонувати обмін іншим гравцям, платити ренту тощо.Ігрове поле складається з клітинок, розташованих по периметру квадрата. Клітинки поділяються на кілька типів: старт (Start), власність (компанії технологічного сектору як Apple, Microsoft, Google, Facebook тощо), податки (Fee), спеціальні клітинки (Airdrop, Day Trade, Ico), та особливі кутові клітинки:

- "Banned!" - відповідає концепції в'язниці в класичній Монополії
- "Warning" - аналог поля "Просто відвідування" біля в'язниці
- "Nap Time" - аналог безкоштовної стоянки

Кожен тип клітинки має свої унікальні властивості та поведінку, яка визначає, що відбувається, коли гравець потрапляє на неї. Клітинки власності (компанії) є найскладнішим типом, оскільки мають додаткові атрибути: вартість покупки, вартість ренти (яка може змінюватись), вартість розвитку.

Важливою властивістю є колірна група, до якої належить компанія, що впливає на стратегію гри та вартість ренти.Гральні кубики є важливим елементом гри, оскільки визначають переміщення гравців по полю.

У реалізації використовуються два шестигранні кубики з 3D-анімацією, сума чисел на яких визначає кількість клітинок для переміщення гравця. Розроблена система візуалізації кидків кубиків через SVG та CSS-анімації створює відчуття реалістичного кидка.Ігрова валюта використовується для всіх фінансових операцій у грі: інвестування в компанії (купівля власності), отримання прибутку (рента), сплата податків та штрафів. Кожен гравець починає з певною сумою, яка змінюється протягом гри.Торгові угоди між гравцями реалізовані через спеціальний інтерфейс, що дозволяє обмінюватися власністю та грошима. Гравець може запропонувати обмін, вказавши, яку власність і гроші він віддає, та яку власність і гроші хоче отримати від іншого гравця. Інший гравець може прийняти або відхилити пропозицію.На основі аналізу предметної області було розроблено концептуальну модель, яка відображає основні сутності та їх взаємозв'язки:

1. Game - центральний клас, який представляє ігрову сесію та координує взаємодію між іншими компонентами. Містить посилання на гравців, ігрове поле, поточний стан гри та методи для управління ігровим процесом.

2. Player - представляє гравця з його атрибутами (ім'я, баланс, власність) та методами для виконання ігрових дій.

3. GameBoard - представляє ігрове поле, яке складається з набору клітинок.

4. Cell (тематична назва поля) - абстрактний клас, який представляє клітинку на ігровому полі. Має підкласи для різних типів клітинок: Клітинки власності (компанії): Технологічні компанії різних кольорових груп: Apple, Microsoft, Google, Amazon, Facebook, Twitter, LinkedIn, Instagram, Nvidia, AMD,

Intel, IBM, Oracle, SAP, Tesla, SpaceX, Samsung, Huawei, Xiaomi, Sony, Cisco, Dell

Клітинки податків: Fee 1, Fee 2, Fee 3, Fee 4 (червоним кольором) FeeCell - клітинка податку

Спеціальні клітинки:

- Airdrop 1, Airdrop 2, Airdrop 3 (світло-бірюзовим кольором)
- Ico 1, Ico 2, Ico 3, Ico 4 (сірим кольором)
- Day Trade 1, Day Trade 2, Day Trade 3 (світло-блакитним кольором)

Кутові та особливі клітинки:

- Start (початкова клітинка, жовтого кольору)
- Banned! (аналог в'язниці, знаходиться в кутку)
- Warning (аналог відвідування в'язниці, знаходиться в протилежному кутку)
- Nap Time (аналог безкоштовної стоянки, знаходиться в кутку)

Property - представляє власність, яку можна купити, продати або обміняти. Має різні підкласи відповідно до типу компанії та кольорової групи.

6. Dice - представляє гральний кубик з методом для генерації випадкового числа та анімації кидка.
7. Trade - представляє торгову угоду між гравцями, включаючи перелік власності та грошей, що пропонуються для обміну.
8. Room - представляє ігрову кімнату (стіл), де знаходяться гравці, які беруть участь в одній грі.

Для реалізації взаємодії між гравцями в реальному часі було розроблено модель повідомлень, які передаються між клієнтською та серверною частинами через Socket.IO. Кожне повідомлення має тип та набір даних. Основні типи повідомлень включають:

- 'join1', 'join2', 'join3', 'join4' - повідомлення для процесу приєднання до гри
- 'start1', 'start2', 'start3' - повідомлення для початку гри
- 'roll1', 'roll2' - повідомлення для кидка кубиків та обробки результату
- 'next1', 'next2' - повідомлення для переходу ходу до наступного гравця
- 'buy1', 'sell1' - повідомлення для купівлі/продажу власності
- 'mess1', 'mess2' - повідомлення для чату
- 'trade_offer', 'trade_accept' - повідомлення для обміну власністю між гравцями

Для відображення ігрового поля та взаємодії з ним використовується технологія SVG, яка дозволяє створювати векторну графіку з можливістю маніпуляції окремими елементами. Кожна клітинка ігрового поля представлена як SVG-прямокутник з унікальним ідентифікатором, що дозволяє легко змінювати її вигляд та обробляти події.Фішки гравців представлені як SVG-кола різних кольорів, які переміщуються по полю за допомогою CSS-анімацій. Це створює плавні та візуально привабливі переходи між клітинками.Кубики реалізовані як 3D-об'єкти з використанням CSS-трансформацій та анімацій.

Вони мають реалістичну фізику падіння та обертання, що імітує справжній кидок кубиків.Для синхронізації стану гри між різними клієнтами використовується механізм подій Socket.IO. Коли один гравець виконує дію, сервер отримує повідомлення, обробляє його та розсилає оновлення стану гри всім гравцям у кімнаті. Це забезпечує однакове відображення ігрового поля та стану гри для всіх учасників.Для обробки особливих ситуацій, як-от потрапляння на клітинку "Banned!" або спеціальний перехід між клітинками, реалізовані окремі функції, наприклад, specialMoveToBannedToWarning(), яка обробляє переміщення з клітинки "Banned!" до клітинки "Warning".Розроблена модель предметної області забезпечує чітке розуміння структури системи та взаємодії

її компонентів, що є основою для подальшої реалізації програмного коду. Вона також враховує особливості конкретної реалізації гри "Монополія" з її унікальними елементами та механізмами.

2.1.2. Розробка бізнес моделі

Основна мета бізнес-моделі — створити інтерактивне середовище, яке дозволяє гравцям взаємодіяти в реальному часі, виконувати ігрові дії, такі як кидок кубиків, купівля/продаж полів, обмін ресурсами, а також спілкуватися через чат.

Основні компоненти бізнес-моделі:

- Гравці є основними учасниками гри. Вони можуть приєднуватися до кімнат, виконувати ігрові дії, взаємодіяти з іншими гравцями через обмін ресурсами та спілкування.
- Кожен гравець має унікальний ідентифікатор, ім'я, баланс (гроші) та список власних полів.

Кімнати (ігрові столи):

- Кімнати є віртуальними просторами, де відбувається гра. Кожна кімната має обмежену кількість місць для гравців.
- У кімнаті зберігається інформація про активного гравця, результати останніх кидків кубиків, а також пропозиції обміну між гравцями.

Ігрові дії:

- Кидок кубиків: Гравці кидають кубики, щоб визначити кількість кроків на ігровому полі.

- Купівля/продаж полів: Гравці можуть купувати або продавати поля, які вони займають, за певну вартість.
- Обмін ресурсами: Гравці можуть пропонувати обмін полями або грошима іншим гравцям у кімнаті.

Сервер:

- Сервер є центральним компонентом, який обробляє всі запити від клієнтів, синхронізує стан гри між гравцями та забезпечує обробку ігрових подій.
- Сервер зберігає інформацію про кімнати, гравців, пропозиції обміну та результати кидків кубиків.

Клієнтська частина:

- Клієнтська частина відповідає за відображення ігрового інтерфейсу, взаємодію гравців із системою та передачу даних на сервер.
- Вона включає елементи управління, такі як кнопки для кидка кубиків, початку гри, обміну ресурсами тощо.

Взаємодія компонентів:

1. Гравець підключається до сервера та вибирає кімнату для гри.
2. Сервер додає гравця до відповідної кімнати та повідомляє інших учасників про нового гравця.
3. Гравці виконують ігрові дії, які передаються на сервер для обробки.
4. Сервер синхронізує стан гри між усіма клієнтами в кімнаті.
5. У разі обміну ресурсами сервер перевіряє коректність пропозиції та надсилає її іншому гравцю.
6. Після завершення гри сервер повідомляє всіх учасників про результати.

Логіка обміну ресурсами:

- Гравець створює пропозицію обміну, вказуючи поля, гроші та іншого гравця.
- Сервер перевіряє, чи існує гравець, якому адресована пропозиція, та надсилає її.
- Інший гравець може прийняти або відхилити пропозицію.
- У разі прийняття сервер оновлює стан гравців і повідомляє всіх учасників кімнати про успішний обмін.

Переваги бізнес-моделі:

- Масштабованість: Система підтримує кілька кімнат, що дозволяє одночасно грати багатьом гравцям.
- Інтерактивність: Гравці можуть взаємодіяти в реальному часі через обмін ресурсами та чат.
- Гнучкість: Можливість додавання нових функцій, таких як спеціальні правила гри або додаткові ігрові дії.

2.1.2 Розробка бізнес моделі

Основна мета цієї системи — забезпечити зручний і доступний спосіб проведення гри "Монополія" незалежно від місцезнаходження гравців. Розробка бізнес-моделі дозволяє зрозуміти, для кого створено продукт, які проблеми він вирішує, та які можливості розвитку або масштабування має в подальшому.Цільова аудиторія:Цільовою аудиторією розробленого веб-додатку є переважно молоді люди віком від 14 до 35 років, які мають інтерес до економічних настільних ігор, стратегій та соціальних взаємодій. Це можуть бути: школярі, студенти економічних спеціальностей, учасники бізнес-клубів, а також

групи друзів, які цікавляться економічними іграми.Кожен тип користувачів мають дещо різні очікування від застосунку, проте до головних потреб можна віднести:

- Зручний інтерфейс з можливістю швидкої навігації по ігровому полю
- Швидкодія застосунку з мінімальними затримками
- Можливість грати в групи до 6 осіб
- Доступ з будь-якого пристрою через веб-браузер
- Вбудований чат для комунікації між гравцями
- Система обміну та купівлі-продажу власності

Запропонована система вирішує низку актуальних проблем, пов'язаних з проведенням гри "Монополія" у онлайн режимі:

- Відсутність фізичної присутності

У сучасному світі не завжди можливо зібрати всіх гравців офлайн. Система дозволяє організовувати гру дистанційно, об'єднуючи користувачів у спільному лобі через інтернет.

- Автоматизація ігрового процесу

Система автоматично відстежує черговість ходів, результати кидків кубиків, стан гравців та їх власності, що виключає помилки та спрощує процес гри.

- Складність у веденні обліку

У фізичній версії гри складно відстежувати всі транзакції, власність та стан гравців. Система автоматично веде облік всіх операцій, що робить гру більш прозорою та зрозумілою.

- Відсутність можливості збереження стану гри

У фізичній версії неможливо зберегти стан гри для продовження пізніше. Система дозволяє зберігати прогрес та продовжувати гру в будь-який час.

- Складність для новачків

Завдяки інтуїтивному дизайну, автоматизованому веденню обліку та підказкам, система підходить як для досвідчених гравців, так і для новачків.

- Відсутність універсального інструменту

Багато онлайн-альтернатив мають обмежену функціональність або вимагають встановлення додатків. Запропонована система працює як веб-додаток

у браузері, доступний з будь-якого пристрою.Після аналізу цільової аудиторії, її основних потреб та способів, якими запропонована система здатна їх задовольнити, було визначено перелік ключових функціональних можливостей системи:

- Створення ігрових кімнат з підтримкою до 6 гравців
- Система кидків кубиків з візуалізацією результатів
- Автоматичне відстеження черговості ходів
- Система купівлі та продажу власності
- Вбудований чат для комунікації між гравцями
- Система обміну власності між гравцями
- Автоматичний розрахунок ренти та інших платежів

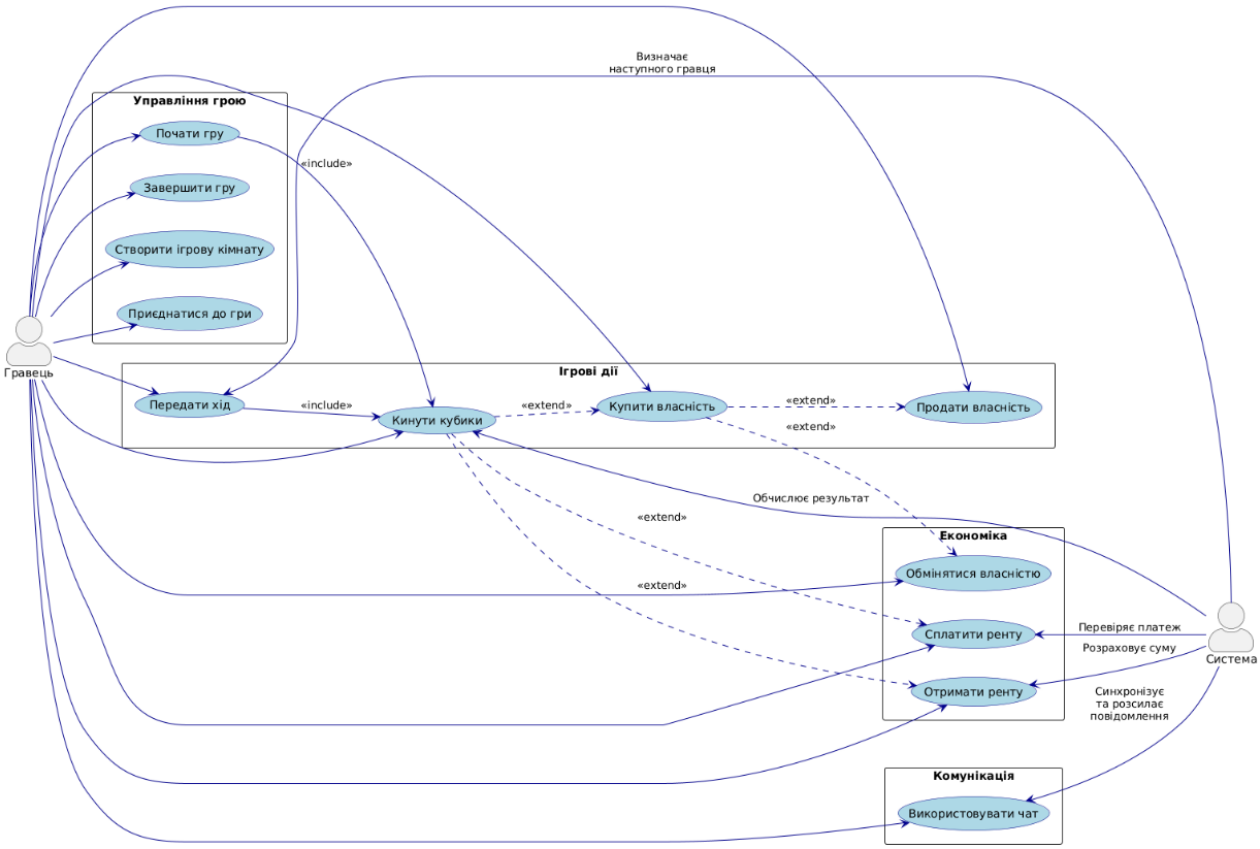

Рис. 2.4 – Діаграма варіантів використання.

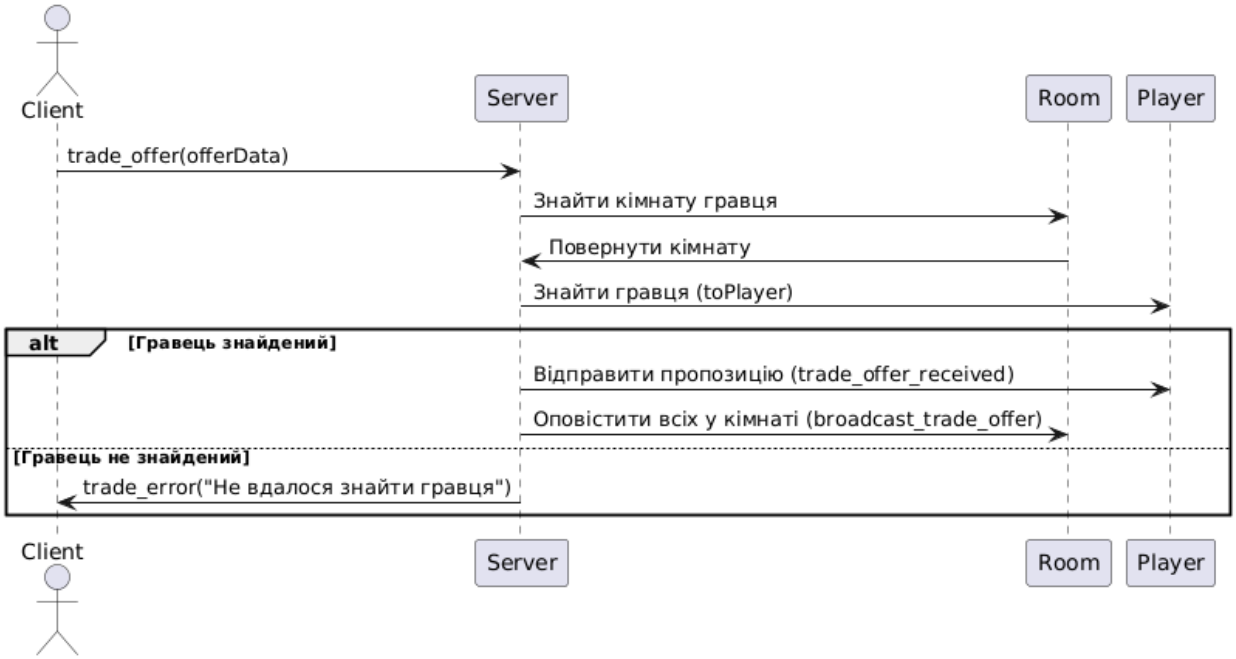

Рис. 2.5 – Діаграма послідовності.

Модель монетизації системи включає наступні елементи:

- Базовий функціонал безкоштовний для всіх користувачів
- Рекламна модель для підтримки розвитку проекту
- Преміум-аккаунт
- Розширені аватари
- Скіни гравців і філій

Перспективи розвитку системи:

- Додавання нових ігрових режимів (швидка гра, турнірний режим)
- Впровадження системи досягнень та нагород
- Інтеграція з популярними месенджерами для запрошення друзів

Така бізнес-модель дозволяє забезпечити стабільний розвиток системи, зберігаючи баланс між доступністю для користувачів та економічною ефективністю проекту.

2.1.3. Проєктування архітектури

Розроблена система базується на клієнт-серверній архітектурі з використанням веб-технологій. Архітектура додатку складається з наступних основних компонентів:

Серверна частина (Backend):

Побудована на Node.js з використанням фреймворку Express.js.

Використовує Socket.IO для реалізації двостороннього зв'язку в реальному часі.

Відповідає за:

- Управління ігровими кімнатами
- Обробку подій гри
- Синхронізацію стану між гравцями
- Керування ходами гравців
- Обробку кидків кубиків
- Управління торговими пропозиціями

Клієнтська частина (Frontend):

Реалізована з використанням HTML, CSS та JavaScript.

Структурована наступним чином:

- /public/index.html - головний файл інтерфейсу
- /public/styles/ - каталог зі стилями
- /public/source/ - каталог з клієнтськими скриптами
- /public/images/ - каталог з графічними ресурсами

Комунікаційний шар:

Базується на протоколі WebSocket через Socket.IO.

Забезпечує:

- Миттєву передачу даних між клієнтом та сервером
- Синхронізацію стану гри між всіма гравцями
- Обмін повідомленнями в реальному часі

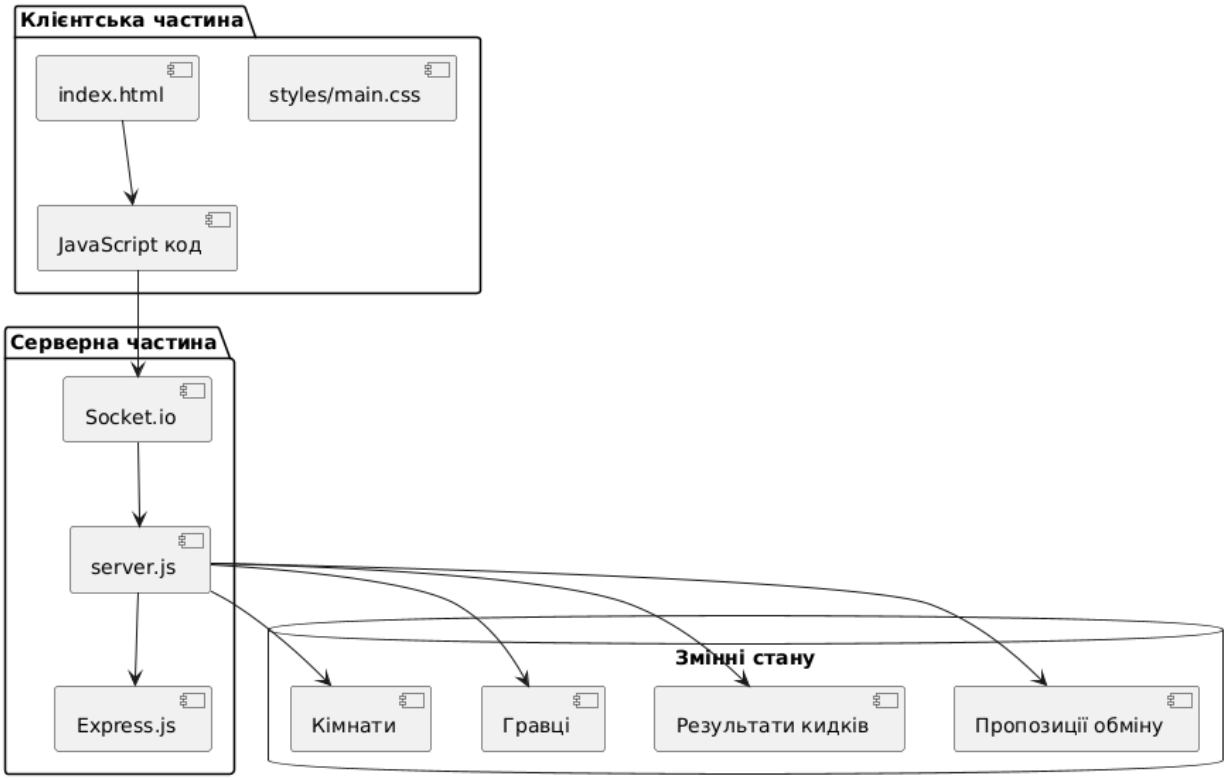

Рис.2.6 – Діаграма компонентів

2.1.4 Аналіз та обґрунтування вибору процедурного підходу для реалізації проєкту

У сучасному розвитку програмного забезпечення існує багато різних підходів до розробки, кожен з яких має свої переваги та недоліки. Для реалізації проекту багатокористувацької настільної гри "Монополія" було обрано процедурний підхід, який є одним з найбільш традиційних та перевірених методів програмування. Цей вибір був зроблений після ретельного аналізу вимог проекту, його специфіки та можливих варіантів реалізації.

Процедурний підхід передбачає розбиття програми на окремі процедури (функції), які виконують конкретні завдання та можуть викликати одна одну. Така структура особливо ефективна для реалізації настільних ігор, де кожна дія гравця може бути чітко визначена та оброблена окремою процедурою. Крім того, цей

підхід дозволяє легко керувати станом гри та синхронізувати дії між різними гравцями в реальному часі.

Обґрунтування вибору процедурного підходу для даного проекту можна представити наступними пунктами:

Простота реалізації та розуміння:

- Процедурний підхід дозволяє розбити складну логіку гри на окремі функції, які легко зрозуміти та підтримувати
- Кожна функція відповідає за конкретну дію (наприклад, roll1 для кидання кубиків, join1 для приєднання до кімнати)

Ефективність для мережевої гри:

- Процедурний підхід добре підходить для обробки мережевих подій через Socket.IO
- Кожна подія (event) обробляється окремою функцією, що спрощує відстеження та налагодження
- Це дозволяє ефективно керувати станом гри та синхронізувати дії між гравцями

Оптимізація ресурсів:

- Процедурний підхід дозволяє ефективно використовувати пам'ять, оскільки стан гри зберігається в простих структурах даних
- Змінні як rooms, players, names легко оновлюються та відслідковуються
- Це особливо важливо для багатокористувацької гри, де потрібно ефективно керувати ресурсами

Масштабованість:

- Код легко розширюється додаванням нових функцій для нової функціональності

- Існуюча структура дозволяє просто додавати нові кімнати або правила гри
- Процедурний підхід спрощує тестування окремих компонентів системи

Відповідність вимогам проекту:

- Для реалізації настільної гри "Монополія" процедурний підхід є достатнім, оскільки основна логіка гри є послідовною
- Всі дії в грі (кидання кубиків, купівля власності, обмін) можуть бути представлені як послідовність процедур
- Це спрощує реалізацію правил гри та їх перевірку

Зручність розробки:

- Процедурний підхід дозволив швидко розробити базову версію гри
- Легко вносити зміни та виправляти помилки, оскільки кожна функція відповідає за конкретну дію
- Код можна поступово покращувати та оптимізувати без значних змін архітектури

2.1.5 Специфікація вимог до програмного забезпечення (SRS)

Документ SRS описує веб-гру "Монополія", призначену для онлайн-гри між користувачами. Програмне забезпечення реалізує класичні правила настільної гри, дозволяючи гравцям змагатися через Інтернет. Ключові функції включають підключення до ігрових столів (кімнат), управління фінансовим станом гравців (гаманці, банк), переміщення фішок гравців по віртуальному ігровому полю з різними типами клітинок, механіку кидання віртуальних кубиків, можливість купівлі, продажу та обміну власності (полів), нарахування та сплату

ренти/комісій, отримання бонусів (аірдропів), функціонал текстового чату для взаємодії гравців та визначення переможця після банкрутства інших гравців.

1. Загальна характеристика

Система є клієнт-серверним веб-застосунком. Клієнтська частина (браузер користувача) відображає ігрове поле, стан гри, рахунки гравців та чат, а також приймає дії користувача (кидки кубиків, купівля, продаж, обмін, повідомлення чату). Серверна частина управляє ігровою логікою, синхронізує стан гри між усіма підключеними гравцями в одній кімнаті та обробляє їхні запити. Цільовою аудиторією є користувачі, зацікавлені в грі "Монополія" онлайн.

2. Функціональні вимоги

Система повинна дозволяти користувачеві: ввести ім'я користувача; вибрати та приєднатися до одного з доступних ігрових столів (до 6 гравців на стіл); зайняти місце за столом; після початку гри (коли достатньо гравців), по черзі кидати віртуальні кубики (один або два, судячи з інтерфейсу); автоматично переміщувати фішку гравця на відповідну кількість клітинок; виконувати дії, пов'язані з клітинкою, на яку потрапив (наприклад, побачити ціну, купити, продати, заплатити комісію, отримати бонус); ініціювати та брати участь в обміні власністю та/або грошима з іншими гравцями через модальні вікна; відправляти та отримувати текстові повідомлення в ігровому чаті; бачити оновлення балансів гаманців гравців та банку; бачити, хто зараз робить хід; система має автоматично обробляти банкрутство гравців та визначати переможця.

3. Вимоги до інтерфейсу користувача

Інтерфейс реалізовано за допомогою HTML, CSS (зокрема, з використанням Skeleton для базової стилізації) та JavaScript (включаючи Alpine.js для додавання інтерактивності без складних фреймворків) для візуалізації дошки та фішок. Використання SVG для відображення ігрового поля та фішок. Взаємодія здійснюється через кнопки та поля введення. Візуальні елементи включають: SVG-поле з різними клітинками, що змінюють колір залежно від власника; SVG-фішки для кожного гравця, що рухаються по полю; текстові елементи для відображення імені поточного гравця, результату кидка кубиків, поточної позиції

гравця, ціни/дії на клітинці; таблиця або список з іменами гравців та їхніми поточними балансами гаманців; відображення балансу банку; область чату зі списком повідомлень та полем введення/кнопкою відправки; модальні вікна для створення та обробки пропозицій обміну (власність + гроші); анімація кидка кубиків.

4. Вимоги до виконання

Анімація кидка кубиків та руху фішок повинна бути плавною, без помітних затримок. Оновлення стану гри (переміщення фішок, зміна кольору полів, оновлення балансів, повідомлення чату) повинно відбуватися в реальному часі для всіх гравців за столом. Система має підтримувати одночасну гру декількох груп гравців за різними столами. Час відгуку на дії користувача та оновлення даних з сервера має бути мінімальним для комфортної гри.

5. Нефункціональні атрибути

Інтерфейс гри має бути інтуїтивно зрозумілим для користувачів, які знайомі з правилами "Монополії". Гра повинна стабільно працювати під час ігрової сесії без раптових вильотів чи помилок, які призводять до втрати прогресу. Система повинна забезпечувати коректну обробку всіх ігрових транзакцій (купівля, продаж, обмін, комісії, бонуси) та оновлення балансів без помилок. Необхідно запобігти можливості гравцям впливати на ігрову логіку або дані (наприклад, змінювати свій баланс чи позицію) на клієнтській стороні.

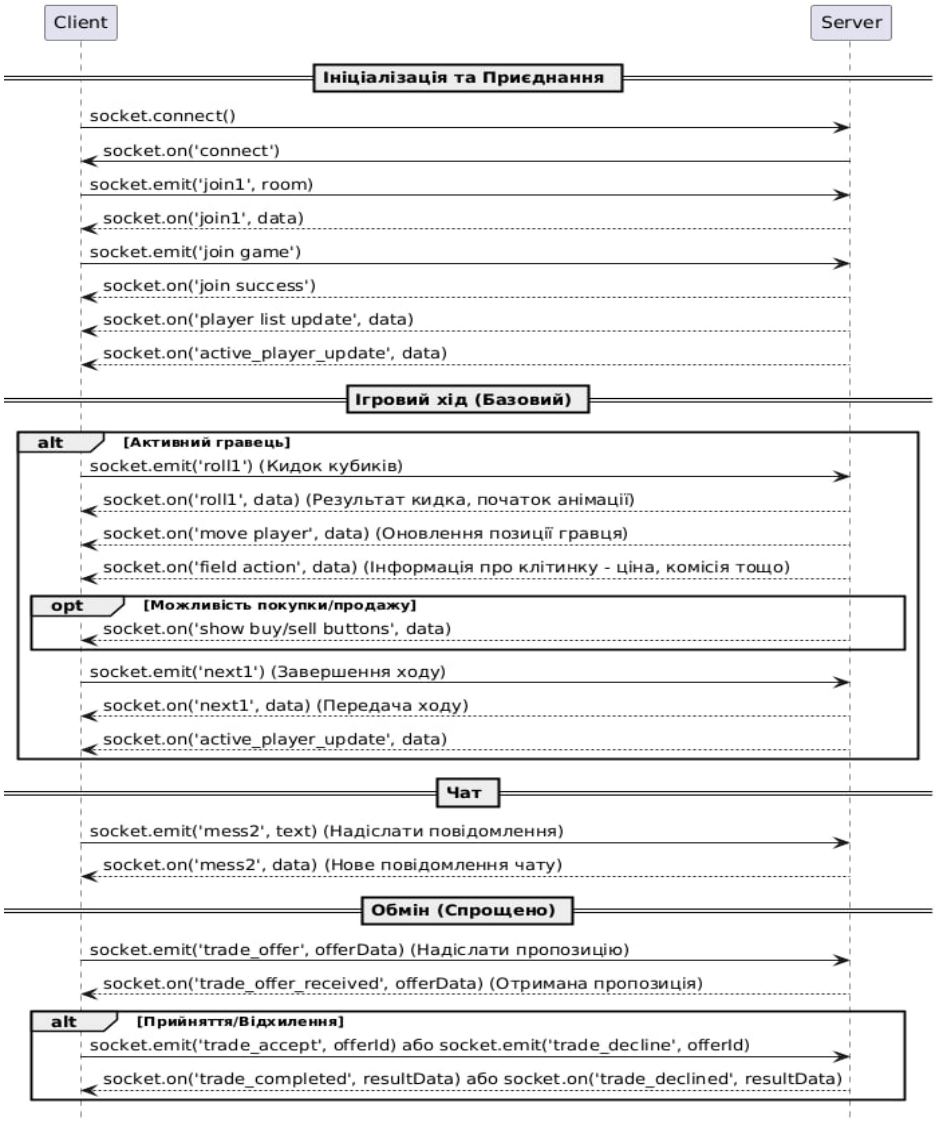

Рис.2.7 – Діаграма взаємодії клієнта та сервера через Socket.IO

2.2 Конструювання

У цьому розділі детально описано процес реалізації основних компонентів програмного продукту — багатокористувацької гри "Монополія". Наведено ключові рішення щодо вибору технологій, структури коду, організації взаємодії між клієнтом і сервером, а також особливості інтеграції окремих модулів у єдину систему.

2.2.1 Вибір технологій та середовища розробки

Для реалізації серверної частини було обрано платформу Node.js, яка забезпечує високу продуктивність та зручність роботи з асинхронними подіями. Для організації двосторонньої комунікації між клієнтами та сервером використовується бібліотека Socket.IO.

Клієнтська частина реалізована з використанням HTML, CSS та JavaScript, що забезпечує кросплатформеність та доступність гри з будь-якого сучасного браузера. Для спрощення та прискорення розробки інтерфейсу було використано CSS-фреймворк Skeleton, який дозволяє швидко створювати адаптивні та мінімалістичні макети без зайвого перевантаження стилями. Це дало змогу зосередитися на функціональності та зручності користувача.

Для організації реактивної поведінки елементів інтерфейсу та керування станом на клієнті застосовано JavaScript-фреймворк Alpine.js. Його використання дозволило реалізувати динамічні елементи (наприклад, відображення/приховування кнопок, анімації, інтерактивні підказки) без необхідності підключення важких бібліотек на кшталт Vue чи React. Alpine.js ідеально підходить для невеликих SPA-додатків та інтерактивних сторінок, де важлива швидкість завантаження та простота підтримки коду.

2.2.2 Реалізація серверної частини

Серверна логіка зосереджена у файлі server.js. Основні функції сервера:

- обробка підключень гравців;
- створення ігрових кімнат;
- синхронізація стану гри між усіма учасниками;

- обробка ігрових подій (кидання кубиків, купівля/продаж власності, обмін, чат).

Для зберігання стану гри використовуються прості структури даних: масиви для гравців, кімнат, імен, результатів кидків тощо. Кожна подія, що надходить від клієнта, обробляється окремою функцією, яка оновлює відповідний стан і розсилає зміни всім учасникам кімнати.

Підключення та ініціалізація:

```
const express = require('express');
const app = express();
const http = require('http').createServer(app);
const io = require('socket.io')(http);
app.use(express.static('public'));
const rooms = [];
const players = {};
const names = {};
const diceResults = {};
```

Обробка підключень гравців:

```
io.on('connection', (socket) => {
  console.log('Користувач підключився:', socket.id);
  socket.on('join', (data) => {
    const { roomId, playerName } = data;
    if (!rooms.includes(roomId)) {
      rooms.push(roomId);
    }
    players[socket.id] = roomId;
    names[socket.id] = playerName;
    socket.join(roomId);
    io.to(roomId).emit('playerJoined', { playerName, playerId:
socket.id });
  });
});
```

Обробка ігрових подій:

```
socket.on('rollDice', () => {
  const roomId = players[socket.id];
  const result = Math.floor(Math.random() * 6) + 1;
  diceResults[socket.id] = result;
  io.to(roomId).emit('diceResult', { playerId: socket.id, result });
});
```

Купівля/продаж власності:

```
socket.on('buyProperty', (propertyId) => {
  const roomId = players[socket.id];
  io.to(roomId).emit('propertyBought', { playerId: socket.id,
propertyId });
});
socket.on('sellProperty', (propertyId) => {
  const roomId = players[socket.id];
  io.to(roomId).emit('propertySold', { playerId: socket.id,
propertyId });
```

```
});
```

Обмін власністю:

```
socket.on('tradeProperty', (data) => {
  const { targetPlayerId, propertyId } = data;
  const roomId = players[socket.id];
  io.to(roomId).emit('propertyTraded', { fromPlayerId: socket.id, toPlayerId: targetPlayerId, propertyId });
});
```

Чат:

```
socket.on('chatMessage', (message) => {
  const roomId = players[socket.id];
  io.to(roomId).emit('chatMessage', { playerId: socket.id, playerName: names[socket.id], message });
});
```

Запуск сервера:

```
const PORT = process.env.PORT || 3000;
http.listen(PORT, () => {
  console.log(`Сервер запущено на порту ${PORT}`);
});
```

Сервер використовує Socket.IO для обробки підключень та подій у реальному часі.

Глобальні змінні (rooms, players, names, diceResults) зберігають стан гри.

Кожна подія (підключення, кидання кубиків, купівля/продаж, обмін, чат) обробляється окремою функцією, яка оновлює стан і розсилає зміни всім учасникам кімнати.

2.2.3. Розробка GUI

Під час проектування інтерфейсу гри "Монополія" основна увага приділялася простоті, інтуїтивності та зручності для користувача. На макеті видно, що ігрове поле розташоване по центру екрану, що дозволяє гравцям легко орієнтуватися у грі та швидко сприймати зміни на полі. Праворуч від ігрового поля розміщено чат, який забезпечує ефективну комунікацію між учасниками гри в режимі реального часу.

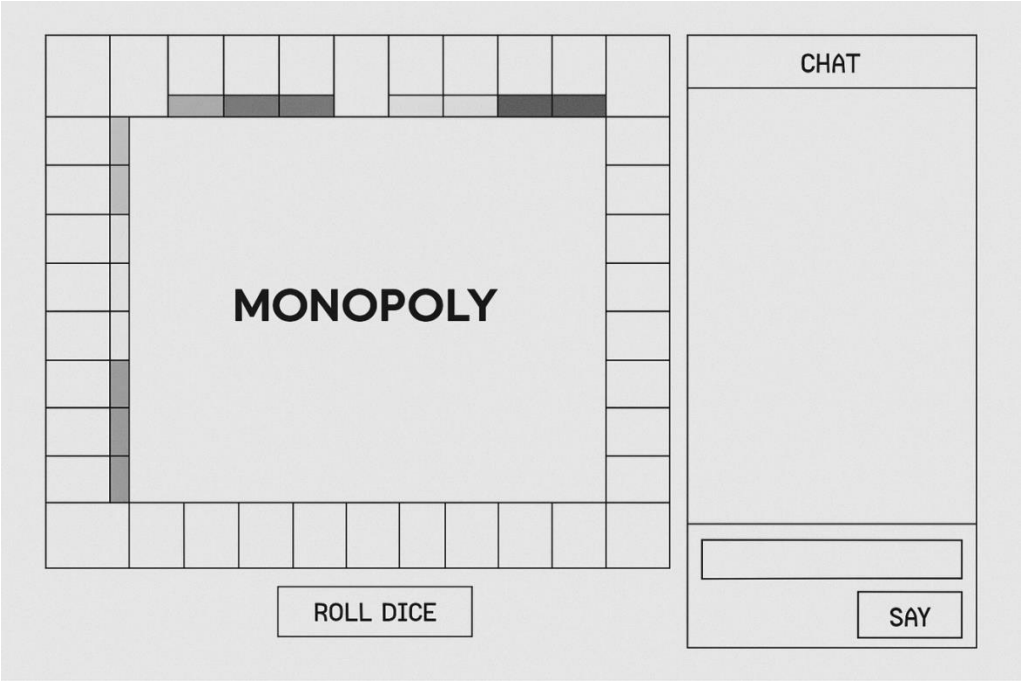

Рис.2.8 – Проектування інтерфейсу (макет)

У нижній частині екрану розташована кнопка для кидка кубиків ("ROLL DICE"), яка є основною дією під час ходу гравця. Під чатом знаходиться поле для введення повідомлення та кнопка "SAY" для надсилання тексту, що робить процес спілкування максимально зручним.

Весь інтерфейс виконано у мінімалістичному стилі, що дозволяє зосередитися на ігровому процесі та не відволікатися на зайві елементи. Такий підхід сприяє кращому користувацькому досвіду та забезпечує легкість у використанні навіть для новачків.

2.2.4. Тестування програмного забезпечення та оцінка якості

Тестування програмного забезпечення є невід’ємною частиною процесу розробки, що дозволяє виявити помилки, оцінити стабільність та якість роботи системи. Для перевірки працездатності серверної частини гри «Монополія» було

проведено комплексне ручне тестування із використанням спеціально створеного тестового клієнта на основі бібліотеки socket.io-client версії 2.1.1.

Метою тестування було перевірити коректність обробки основних ігрових подій, стабільність роботи серверу при багатокористувацькій взаємодії, а також оцінити якість реалізації функціоналу гри.

Інструменти тестування:

Для тестування було створено окрему HTML-сторінку (test.html), яка підключається до серверу через socket.io-client тієї ж версії, що й серверна частина. Даний клієнт дозволяє вручну надсилати будь-які події (join1, join3, roll1, mess2 тощо) та отримувати відповіді від серверу, що забезпечує повний контроль над процесом тестування.

Сценарії тестування:

Під час тестування були перевірені такі основні сценарії:

- Підключення до серверу та створення ігрової кімнати (подія join1)
- Додавання гравців до кімнати (join1, join3)
- Початок гри та кидання кубиків (start1, roll1)
- Надсилання повідомлень у чат (mess2)
- Проведення обміну між гравцями (trade_offer, trade_accept)
- Перевірка синхронізації стану гри між кількома клієнтами

Для кожного сценарію фіксувалися відправлені події, отримані відповіді та статус виконання. Всі логи тестування зберігалися у файл для подальшого аналізу.

Рис. 2.9 – Меню клієнта для тестування запитів

Модуль підключення:

- Встановлення з'єднання з сервером
- Відстеження статусу підключення
- Обробка модулів ігрових подій

Цей клієнт є інтегральним інструментом для розробки та тестування гри "Монополія", забезпечуючи повний контроль над усіма аспектами взаємодії між клієнтом та сервером.

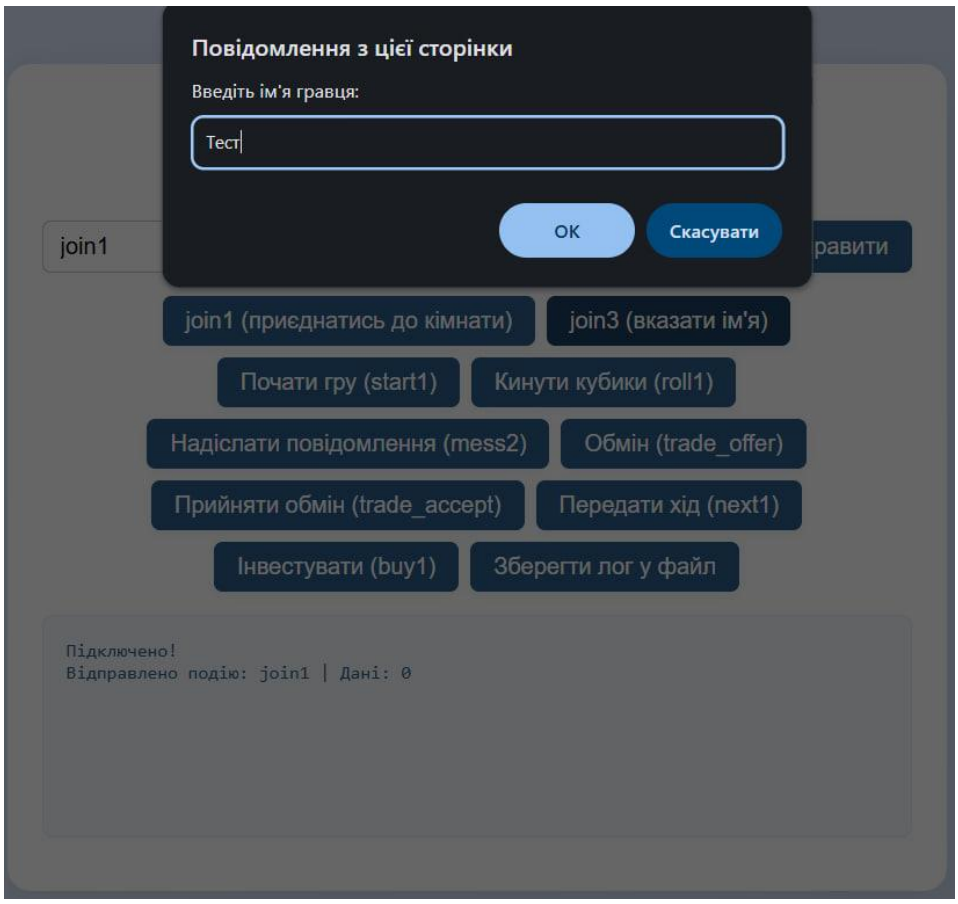

Рис.2.10 – Підключення тестового клієнта

Рис.2.11 – Успішне підключення тестового клієнта

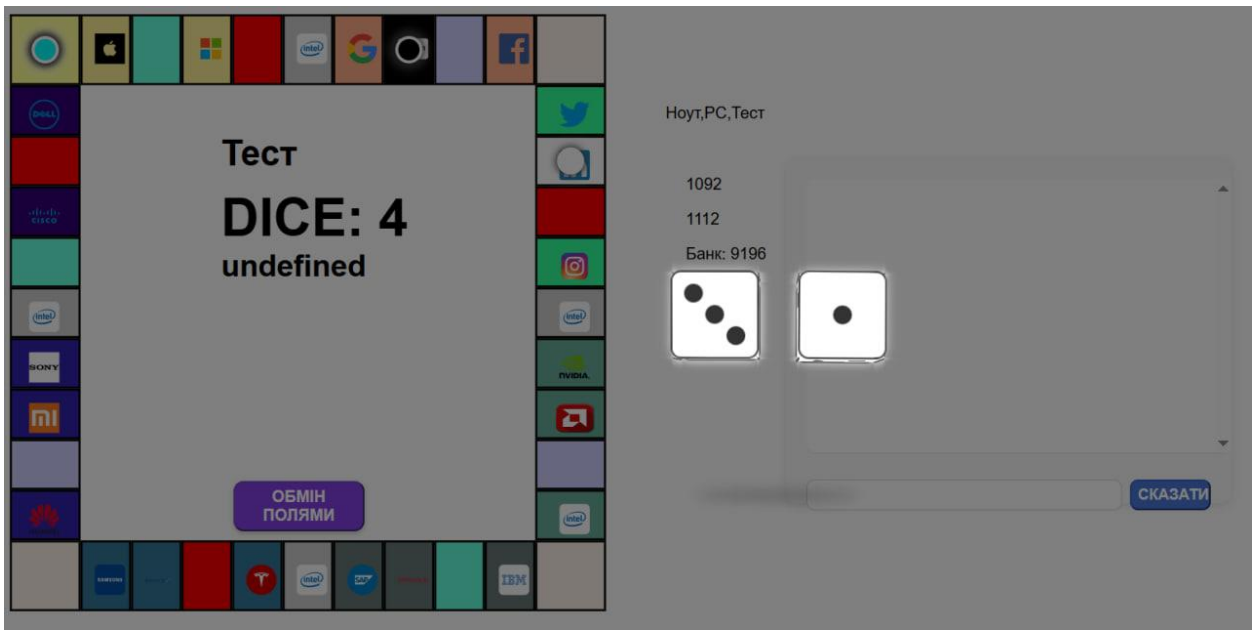

Рис.2.12 – Успішне тестування події кидків кубиків

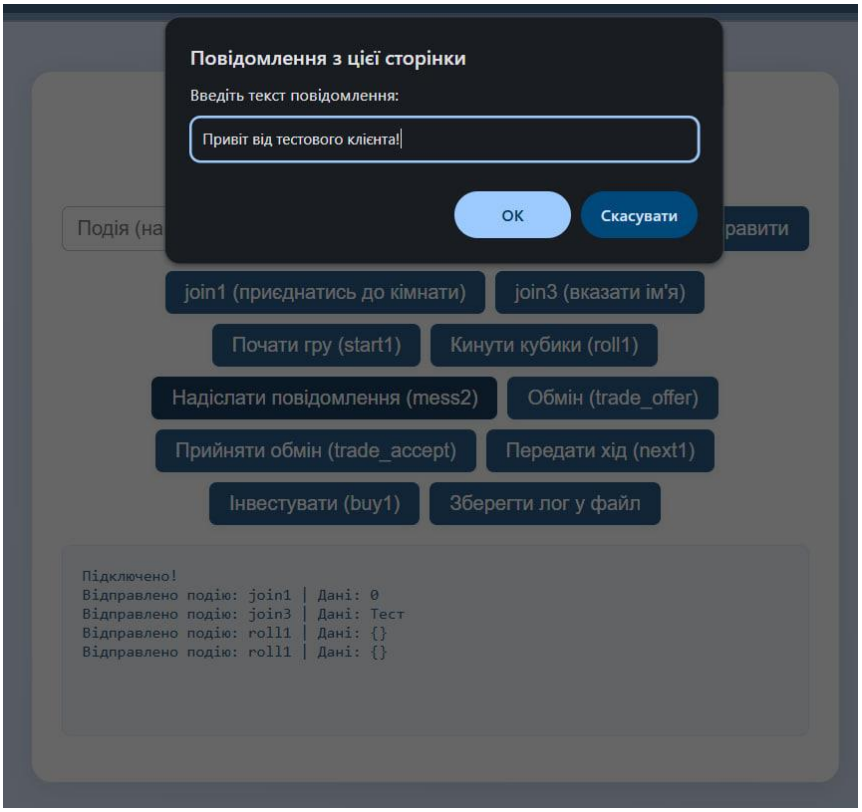

Рис.2.13 – Відправлення події з клінта (повідомлення)

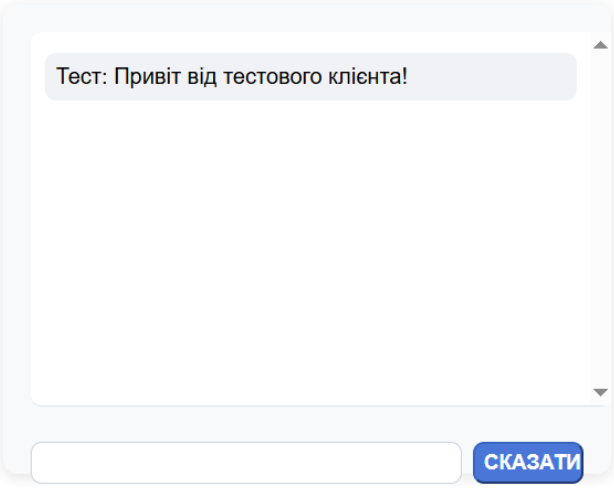

Рис.2.14 – Успішне тестування події (повідомлення)

```
Підключено!
Відправлено подію: join1 | Дані: 0
Відправлено подію: join3 | Дані: Тест
Відправлено подію: mess2 | Дані: "Привіт від тестового клієнта!"
Відправлено подію: roll1 | Дані: {}
Відправлено подію: next1 | Дані: {}
```

Рис.2.15 – Логи клієнта відправлених подій

2.2.4.1 Оцінка якості

За результатами тестування встановлено, що:

- Сервер коректно обробляє всі основні ігрові події.
- Взаємодія між кількома клієнтами відбувається без затримок та втрати даних.
- Всі повідомлення та ігрові дії синхронізуються між гравцями.
- Критичних помилок або збоїв у роботі серверу не виявлено.

Проведене тестування підтвердило працездатність та стабільність програмного забезпечення. Використання власного тестового клієнта дозволило гнучко перевірити всі аспекти роботи серверу, а також зафіксувати результати для подальшого аналізу. Якість реалізації відповідає вимогам до

багатокористувацьких ігор у реальному часі.

2.2.5 Результати розробки

В результаті розробки було створено веб-додаток для онлайн-гри "Монополія" з наступними ключовими характеристиками:

1. Архітектура системи:
   - Серверна частина реалізована на Node.js з використанням Express.js
   - Реалізовано систему кімнат (столів) для одночасних ігор
   - Використано Socket.IO для реалізації реального часу комунікації
2. Функціональні можливості:
   - Підтримка до 4 ігрових столів одночасно
   - До 6 гравців на кожному столі
   - Система чергування гравців
   - Інтегрований чат для спілкування між гравцями
   - Система купівлі та продажу власності
   - Механіка кидання кубиків
   - Система обміну власністю між гравцями
3. Технічні особливості:
   - Реалізовано систему синхронізації стану гри
   - Забезпечено збереження результатів кидків кубиків
   - Реалізовано механізм відстеження активного гравця
   - Створено систему пропозицій по обміну
4. Інтерфейс користувача:
   - Адаптивний веб-інтерфейс на основі Skeleton CSS фреймворку
   - Реактивний інтерфейс з використанням Alpine.js
   - Інтуїтивно зрозуміла система керування

- Візуальне відображення ігрового процесу
- Інтегрований чат для комунікації
- Мінімалістичний та сучасний дизайн завдяки Skeleton
- Реактивні компоненти інтерфейсу на базі Alpine.js

5. Безпека та надійність:
   - Реалізовано систему перевірки дій гравців
   - Забезпечено захист від некоректних дій
   - Реалізовано механізм відновлення стану гри при перепідключенні
6. Масштабованість:
   - Модульна архітектура дозволяє легко розширювати функціонал
   - Підтримка одночасних ігрових сесій
   - Можливість додавання нових ігрових механік
7. Технічний стек:
   - Backend: Node.js, Express.js
   - Реалізація реального часу: Socket.IO
   - Frontend:
   - HTML, CSS, JavaScript
   - Alpine.js для реактивного інтерфейсу
   - Skeleton CSS для адаптивного дизайну
   - Система контролю версій: Git

Кінцевий результат видно на даному Рис.2.16 сам інтерфейс і ігрове поле.

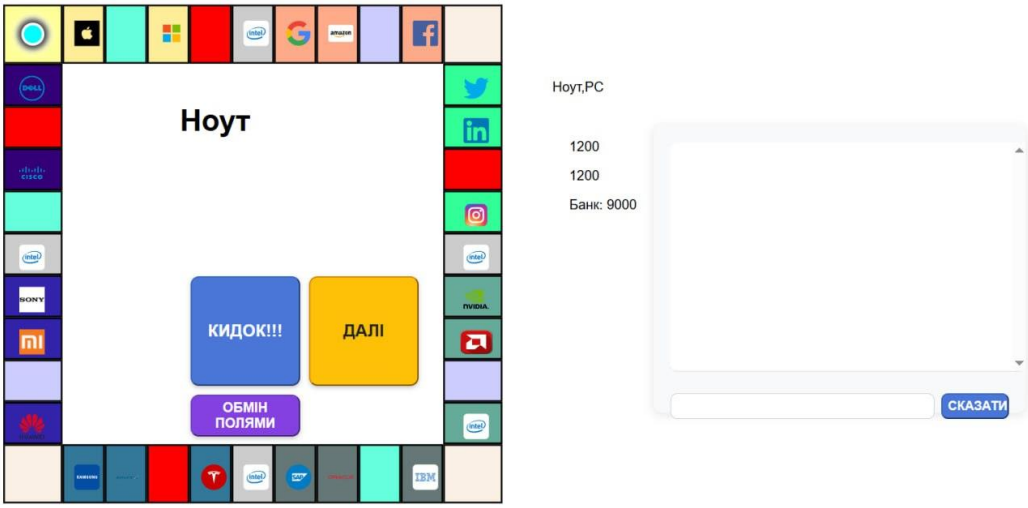

Рис.2.16 – Ігрове поле

Реалізовано система лоббі, саме лоббі обновляється в реальному часі і дає інформацію про присутність гравців в самій кімнаті, видно на Рис.2.17 (в даному випадку ігровий стіл).

Рис.2.17 – Демонстрація присутності гравця в лоббі (стіл)

При вході в лоббі (стіл) гравця чекає поле реєстрації, де потрібно себе індетифікувати і всети своє ігрове ім’я на Рис.2.18, після чого просто зайняти своє місце і чекати коли головний гравець лоббі розпочне гру, сам головний гравець лоббі в якого появиться кнопка “СТАРТ” визначається наступним чином, хто перший зайде в лоббі і займе місце, в того буде право розпочати гру на Рис.2.19.

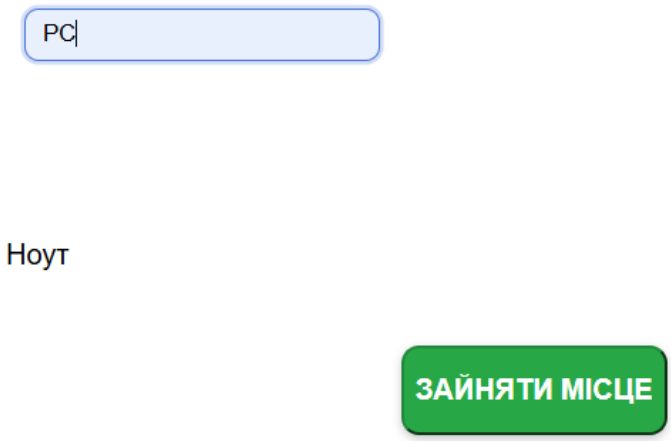

Рис.2.18 – Поле для реєстрації гравця

Рис.2.19 – Інтерфейс головного гравця для старту гри

Ігрова сесія відбувається після того як всі гравці зареєструються (індитефікують себе) після чого головний гравець лоббі розпочне гру нажавши кнопку старт.

Головний функціонал у гравців:

- Кидання кубиків
- Інвестування філій (ігрових полів)
- Обмін філій (ігрових полів) між гравцями
- Писати повідомлення в ігровий чат

Самі ігрові події відбуваються з красивою анімацією які синхронізовані між усіма гравцями ігрової сесії (лоббі).

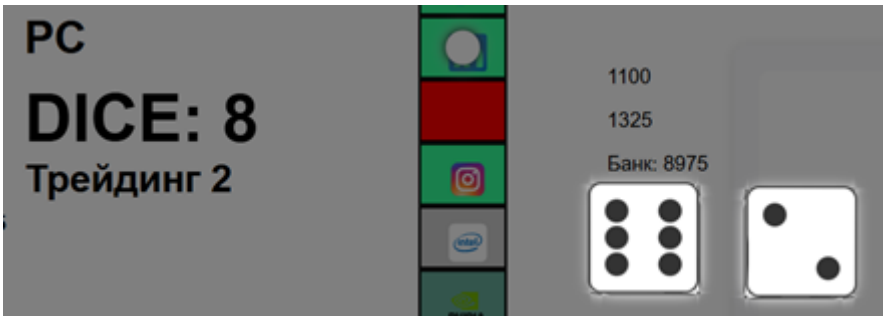

Рис.2.20 – Кидання кубиків з анімацією

Механіка інвестування філій (ігрових полів) відбувається наступним чином, гравець після кидка кубиків виконує n – ну кількість ходів відносного того яке загальне число випало на кубиках, після переходу на філію (ігрове поле) у гравця є можливість нажати на кнопку “ІНВЕСТУВАТИ” або передати хід далі, коли вибір впав на інветсування фон філії краситься в характерний колір гравця на Рис.2.21.

Рис.2.21 – Інвестування філії (ігрового поля)

Інвестування гравця красить філію в колір гравця, бренд компанії самої філії зберігається по верх поля що дозволяє бачити конкретну філії і ким вона вже зайнята.

Обмін полями дозволяє гравцям між собою складати трейденгову пропозицію, в саму пропозицію входить вибір філій (ігрових полів), ігрову валюту. Пропозиція обміну проходить наступним чином, гравець нажимає на кнопку “ОБМІН ПОЛЯМИ” після чого появляється інтерфейс на Рис.2.22 в якому можна створити саму трейденгову пропозицію, в неї входить філії (ігрові поля) якими володіють гравці та поле для введеня суми, після створення пропозиції в чаті гри буде відображена сама пропозиція для гравця якому це було відправлено.

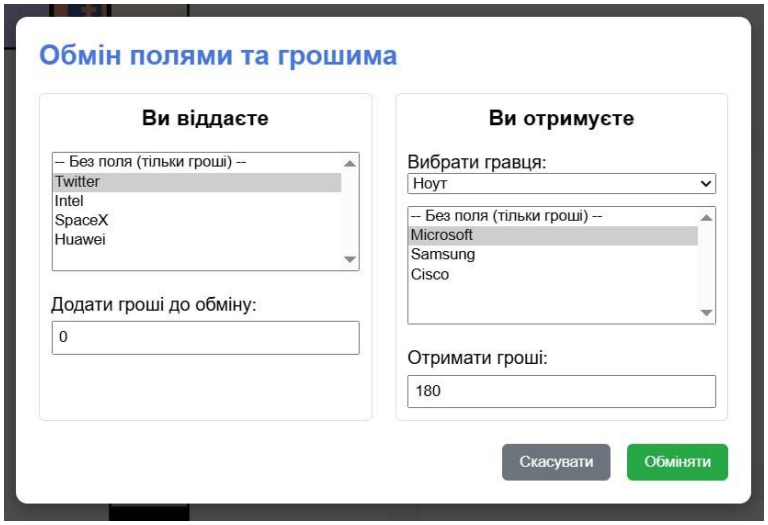

Рис.2.22 – Інтерфейс для створення пропозиції для обміну

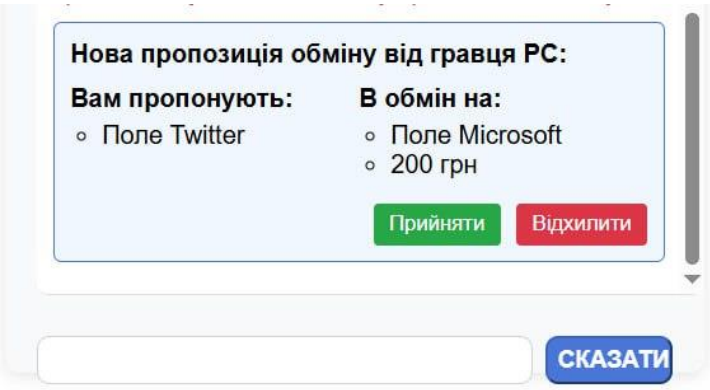

Рис.2.23 – Повідомлення в чаті (пропозиція обміну)

Гравці мають можливість як і прийняти пропозицію обміну так і відхилити її, після прийнятя пропозиції в чаті буде повідомлення про успішний обмін або про відхилення пропозиції залежучи від вибору гравця.

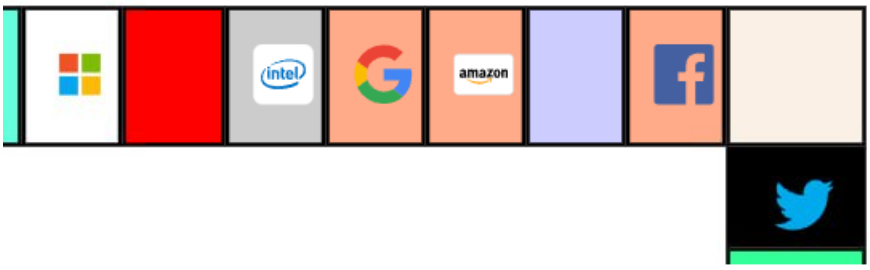

Рис.2.24 – Філія “Майкрософт” і “Твітер” до обміну

Рис.2.25 – Філія “Майкрософт” і “Твітер” після обміну

Коли пропозиція обміну завершена в чаті буде повідомлення яке містить інформацію про самі філії і суму, також ім’я гравців.

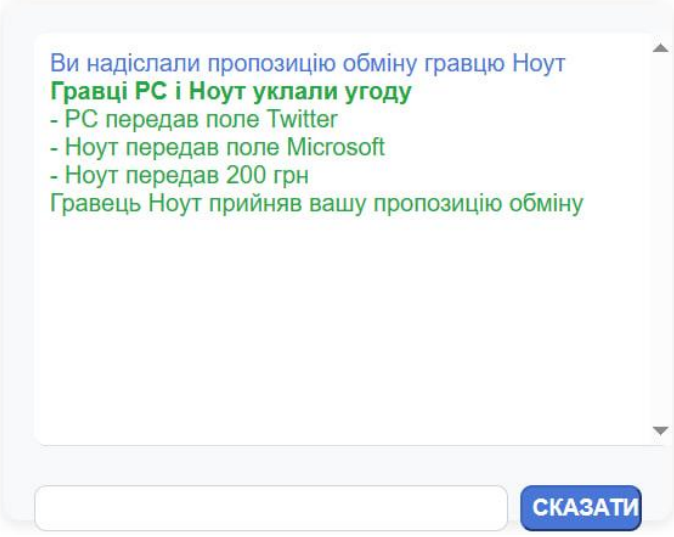

Рис.2.26 – Повідомлення про успішний обмін

Один з важливих аспектів є ігровий чат, гравці можуть між собою спілкуватися за допомогою ігрового чату, в самому чаті видно ім’я гравця а також те що він написав на Рис.2.27 також на Рис.2.28.

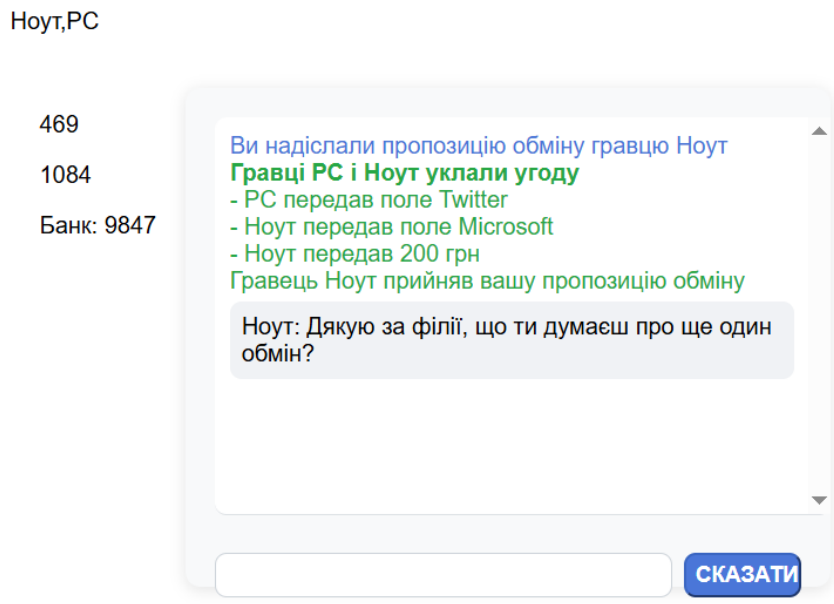

Рис.2.27 – Повідомлення від гравця ”Ноут” в чаті

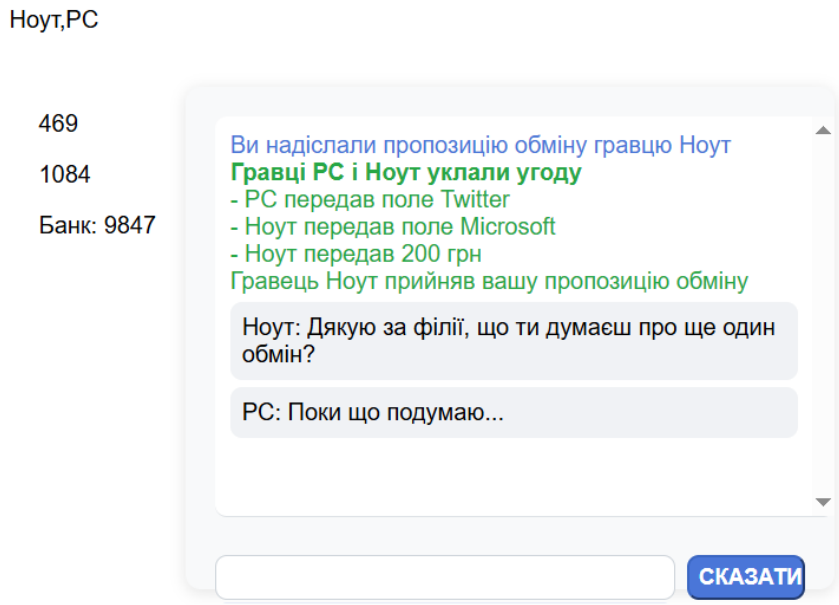

Рис.2.28 – Повідомлення від гравця ”PC” в чаті

Ігровий чат синхронізований для всіх гравців в лобі (ігровій сесії), без потрібності обновляти сторінку.

3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Психологічні чинники небезпеки

Психофізіологічні чинники відіграють надзвичайно важливу роль у забезпеченні безпеки життєдіяльності людини, особливо в професійній сфері. Ці чинники охоплюють як психоемоційні, так і фізіологічні особливості особистості, які можуть сприяти або, навпаки, знижувати рівень безпеки праці.

Мобінг — це тривалий психологічний тиск на працівника з боку колективу або керівництва, що має на меті його ізоляцію, дискредитацію або звільнення. Згідно зі змінами до Кодексу законів про працю України (ст. 2-2, введено у 2022 році), мобінг визнається формою насильства у трудовій сфері. Він негативно впливає на психофізіологічний стан працівника, викликає хронічний стрес, підвищену тривожність, дратівливість, розлади сну та зниження концентрації, що значно підвищує ризик нещасного випадку на виробництві.

Булінг — це агресивна поведінка, яка проявляється у формі фізичного або психологічного насильства, систематичного цькування, переважно в шкільному або студентському середовищі. Булінг є адміністративним правопорушенням (КУпАП, ст. 173-4) і має серйозні наслідки для психофізіологічного здоров’я потерпілого. Жертви булінгу часто мають ознаки депресії, тривожних розладів, а також труднощі в соціальній адаптації.

Поза соціальними загрозами, психофізіологічні небезпеки також включають:

– **Неврівноваженість емоційних процесів**: імпульсивність, різкі зміни настрою, підвищена збудливість. Такі риси у працівника можуть бути джерелом неочікуваної поведінки в стресових умовах.

– **Стан втоми та емоційного вигорання**: тривала монотонна робота або постійні перевантаження викликають зниження швидкості реакцій, уважності, продуктивності та критичності мислення.

– **Дефіцит сенсомоторної координації**: погіршення зв’язку між сенсорними та руховими реакціями часто призводить до зниження точності дій та затримки реакції на небезпечні ситуації.

– **Алкогольна чи наркотична залежність**: вкрай несприятливо впливає на психофізіологічний стан людини, її свідомість, координацію, швидкість реакції та критичність мислення.

– **Незадоволення роботою**: працівник, який не має інтересу до виконуваних обов’язків, часто виявляє неуважність, байдужість, не схильний дотримуватись правил безпеки, що прямо пов’язано з підвищенням ризиків травматизму.

Для зменшення впливу вищезазначених факторів доцільно використовувати методи профілактики: впровадження психогігієнічних заходів, створення комфортного мікроклімату в колективі, навчання управлінню стресом, періодичні психофізіологічні тести та тренінги, спрямовані на підвищення стресостійкості та емоційної стабільності.
Таким чином, своєчасне виявлення та аналіз психофізіологічних чинників дозволяє значно знизити ризики виробничого травматизму та покращити загальну ефективність праці.

3.2 Протипожежні вимоги до робочого освітлення

Робоче освітлення на об'єктах повинно відповідати не лише ергономічним і гігієнічним вимогам, а й вимогам пожежної безпеки. Відповідно до чинних нормативно-правових актів України, таких як ДСТУ, ДБН та НАПБ, робоче та аварійне освітлення мають забезпечувати належний рівень безпеки під час нормальної експлуатації об’єктів, а також у разі пожежі або евакуації.

Матеріали та обладнання, які використовуються в системах освітлення, мають бути виготовлені з негорючих або важкозаймистих матеріалів. Не допускається використання кабелів і проводів, що виділяють токсичні продукти горіння. У приміщеннях підвищеної пожежної небезпеки повинно застосовуватись електрообладнання у вибухозахищеному або вогнестійкому виконанні відповідно до Правил пожежної безпеки в Україні (НАПБ А.01.001-2014).

У місцях масового перебування людей, а також на шляхах евакуації, обов'язковим є аварійне освітлення. Його увімкнення повинно відбуватись автоматично у разі зникнення основного живлення. Мінімальна освітленість має становити не менше 1 лк на рівні підлоги евакуаційного шляху. Аварійне освітлення повинно забезпечувати видимість виходів, пожежного обладнання, щитів та інших важливих елементів. Тривалість роботи аварійного освітлення повинна становити щонайменше 60 хвилин.

Рис.3.1 – Приклад розміщення аварійного освітлення в коридорі

Світильники не повинні заважати евакуації або ускладнювати проходи. Їх розміщення має виключати можливість контакту з легкозаймистими матеріалами або джерелами вибухонебезпечного пилу чи парів. Прокладення електричних кабелів освітлення повинно здійснюватись із дотриманням норм протипожежного захисту.

Електроживлення аварійного освітлення повинно бути незалежним від основної електромережі, з використанням автономних джерел енергії — акумуляторів або генераторів. Усі елементи аварійної системи освітлення повинні перевірятись на справність у встановлені терміни, а результати перевірок фіксуватись у відповідній документації.

Таким чином, протипожежні вимоги до робочого освітлення включають як технічні параметри систем освітлення, так і експлуатаційні обмеження, спрямовані на запобігання пожежам і забезпечення безпечної евакуації людей.

ВИСНОВКИ

Основним досягненням роботи є створення повноцінної системи для проведення онлайн-ігор з підтримкою до 4 ігрових столів та 6 гравців на кожному столі. Використання Node.js та Socket.IO дозволило реалізувати надійну систему комунікації в реальному часі, що є критично важливим для багатокористувацьких ігор. Особливу увагу заслуговує реалізація системи синхронізації стану гри між гравцями, яка забезпечує безперебійний ігровий процес.

Важливим аспектом розробки стало використання сучасних frontend-технологій. Застосування Alpine.js дозволило створити реактивний інтерфейс з мінімальним навантаженням на браузер, а Skeleton CSS забезпечив адаптивний дизайн, що робить гру доступною на різних пристроях.

Практична цінність роботи полягає в тому, що розроблена система може служити основою для створення інших онлайн-ігор. Архітектура проекту дозволяє легко розширювати функціонал та додавати нові ігрові механіки.

Розроблений тестовий клієнт дозволив перевірити надійність системи та виявити потенційні проблеми на ранніх етапах розробки. Це підтверджує важливість комплексного підходу до тестування подібних систем.

Перспективи розвитку проекту включають можливість додавання системи рейтингу гравців, розширення кількості одночасних ігрових сесій, інтеграцію додаткових ігрових механік та створення системи досягнень. Розроблена архітектура дозволяє реалізувати ці покращення без значних змін в базовій структурі системи.

Загалом, дипломна робота продемонструвала можливість створення складних багатокористувацьких систем з використанням сучасних веб-технологій. Розроблений додаток є повноцінним продуктом, готовим до використання, і може служити основою для подальших досліджень у галузі розробки онлайн-ігор.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/
2. Socket.IO Documentation [Електронний ресурс]. – Режим доступу: https://socket.io/docs/v4/
3. Alpine.js Documentation [Електронний ресурс]. – Режим доступу: https://alpinejs.dev/
4. Skeleton CSS Documentation [Електронний ресурс]. – Режим доступу: http://getskeleton.com/
5. Express.js Documentation [Електронний ресурс]. – Режим доступу: https://expressjs.com/
6. MDN Web Docs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/
7. JavaScript.info [Електронний ресурс]. – Режим доступу: https://javascript.info/
8. W3Schools Online Web Tutorials [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/
9. GitHub Documentation [Електронний ресурс]. – Режим доступу: https://docs.github.com/
10. Stack Overflow [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/
11. CSS-Tricks [Електронний ресурс]. – Режим доступу: https://css-tricks.com/
12. WebSocket API [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket
13. HTML Living Standard [Електронний ресурс]. – Режим доступу: https://html.spec.whatwg.org/

14. ECMAScript 2023 Language Specification [Електронний ресурс]. – Режим доступу: https://tc39.es/ecma262/

15. Web Development Best Practices [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Guide/Best_practices

16. Real-time Web Applications with Socket.IO [Електронний ресурс]. – Режим доступу: https://socket.io/get-started/chat

17. Node.js Best Practices [Електронний ресурс]. – Режим доступу: https://github.com/goldbergyoni/nodebestpractices

18. Modern JavaScript Tutorial [Електронний ресурс]. – Режим доступу: https://javascript.info/

19. Web Development Guidelines [Електронний ресурс]. – Режим доступу: https://www.w3.org/standards/webdesign/

20. Responsive Web Design [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design

21. Кодекс законів України. Стаття 2-2. (Мобінг) [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2759-20/#Text

22. НАПБ А.01.001-2014. Правила пожежної безпеки в Україні [Електронний ресурс]. – Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=60541

23. ДСТУ EN 50172:2019. Системи евакуаційного освітлення [Електронний ресурс]. – Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=83261

24. Окіпний І.І., Гурик З.В Методичні вказівки до виконання курсових і розрахунково – графічних робіт з дисципліни “Охорона праці та безпеки життєдіяльності”. – Івано – Франківськ: ІФНТУНГ, 2020. – 42 с.

25. Пономарьов А.П., Гудзь С.М. Безпека життєдіяльності: підручник – Київ: Кондор, 2021. – 368 с.

26. Гандзюк М.П., Невядомський Л.І Охорона праці: навч. Посібник – Харків: ФОП Бровін О.В., 2020 – 312 с.

27. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

28. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

ДОДАТКИ

ДОДАТОК А

Лістинг коду (Ініціалізація змінних та підключення):

```
// Підключення до сервера
var socket = io.connect();

// Основні елементи DOM
var board = document.getElementById('board');
var joinB1 = document.getElementById('joinGroup1');
var joinB2 = document.getElementById('joinGroup2');
var joinB3 = document.getElementById('joinGroup3');
var joinB4 = document.getElementById('joinGroup4');
var joinGame = document.getElementById('joinGame');
var startButton = document.getElementById('startB');
var rollButton = document.getElementById('rollB');
var nextButton = document.getElementById('nextB');
var chatButton = document.getElementById('chatB');
var buyButton = document.getElementById('buyB');
var sellButton = document.getElementById('sellB');

// Елементи для анімації кубиків
var diceContainer = document.getElementById('dice-container');
var dice1Element = document.getElementById('dice1');
var dice2Element = document.getElementById('dice2');
var overlay = document.getElementById('overlay');

// Стан гри
var diceAnimationInProgress = false;
var activePlayerId = '';
var shouldShowRollButton = true;
var chatdiv = document.getElementById('chatdiv');
var joinButtons = [joinB1, joinB2, joinB3, joinB4];
var joinButtonsText = ['СТІЛ 1', 'СТІЛ 2', 'СТІЛ 3', 'СТІЛ 4'];

// Масиви даних гравців
var playerIDsDefault = [11, 12, 13, 14, 15, 16];
var playerIDs = [];
var wallets = [];
var walletsDefault = [1200, 1200, 1200, 1200, 1200, 1200];
var walletLabels = [];
var walletLabelsDefault = ['w1', 'w2', 'w3', 'w4', 'w5', 'w6'];
var playerColors = [];
var playerColorsDefault = ['white', 'black', 'purple', 'yellow', 'pink',
'cyan'];
var counter = [];
var counterDefault = [0, 0, 0, 0, 0, 0];
var circles = [];
var circlesDefault = ['circle0', 'circle1', 'circle2', 'circle3',
'circle4', 'circle5'];
var names = [];

// Ігрові константи
var bankfunds = 9000;
var i = 0;
```

```
var text = '';
var dice = 0;
var spot = 0;
var reward = 0;
var price = 0;
var daytrade = 0;
var pq = 0;
var q = 0;
var air = 0;
var a = 0;
var b1 = 0;
var b2 = 0;
var c1 = 0;
var c2 = 0;
var c3 = 0;
var nums = [];

// Координати центру
var centx = 36;
var centy = 36;

// Масиви ігрових даних
var positions = ['Старт', 'Apple', 'Аірдроп 1', 'Microsoft', 'Комісія 1',
'ICO 1', 'Google', 'Amazon', 'Трейдинг 1', 'Facebook', 'Попередження',
'Twitter', 'LinkedIn', 'Комісія 2', 'Instagram', 'ICO 2', 'Nvidia', 'AMD',
'Трейдинг 2', 'Intel', 'Відпочинок', 'IBM', 'Аірдроп 2', 'Oracle', 'SAP',
'ICO 3', 'Tesla', 'Комісія 3', 'SpaceX', 'Samsung', 'Блокування!',
'Huawei', 'Трейдинг 3', 'Xiaomi', 'Sony', 'ICO 4', 'Аірдроп 3', 'Cisco',
'Комісія 4', 'Dell'];
var upvote = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45];
var sellingPrice = [32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58,
60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96,
98, 100, 102, 104, 106, 108, 110, 112];
var posIDs = [200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211,
212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226,
227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239];
var posIDs2 = [200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211,
212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226,
227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239];
var propColors = ['#FFFF99', '#FFEE99', '#66FFDD', '#FFEE99', 'red',
'#CCCCCC', '#FFAA88', '#FFAA88', '#CCCCFF', '#FFAA88', 'linen', '#33FF99',
'#33FF99', 'red', '#33FF99', '#CCCCCC', '#66AA99', '#66AA99', '#CCCCFF',
'#66AA99', 'linen', '#667777', '#66FFDD', '#667777', '#667777', '#CCCCCC',
'#337799', 'red', '#337799', '#337799', 'linen', '#3322AA', '#CCCCFF',
'#3322AA', '#3322AA', '#CCCCCC', '#66FFDD', '#330077', 'red', '#330077'];
var airdrops = [30, 60, 90, 120, 150, 180];
var shits = ["Рахунок за електроенергію: ", "Комісії за транзакції: ",
"Програш в казино: ", "Вас відмітили негативно: ", "Благодійність: ",
"Інтернет-рахунок: ", "Шантаж: "];
var fees = [100, 125, 150, 175, 200, 225];

// Початковий стан інтерфейсу
board.style.display = 'none';
joinGame.style.display = 'none';
startButton.style.display = 'none';
rollButton.style.display = 'none';
```

```
nextButton.style.display = 'none';
buyButton.style.display = 'none';
sellButton.style.display = 'none';
chatdiv.style.display = 'none';
document.getElementById('username').style.display = 'none';
```

Лістинг коду (Функції керування грою):

```
function joiningGroup(room) {
    socket.emit('join1', room);
    joinB1.style.display = 'none';
    joinB2.style.display = 'none';
    joinB3.style.display = 'none';
    joinB4.style.display = 'none';
    joinGame.style.display = 'block';
    document.getElementById('username').style.display = 'block';
}
function joiningGame() {
    socket.emit('join2');
    let name = document.getElementById('username').value;
    socket.emit('join3', name);
    document.getElementById('username').style.display = 'none';
    joinGame.style.display = 'none';
    socket.emit('join4');
}
function starting() {
    socket.emit('start1');
    startButton.style.display = 'none';
    socket.emit('start2');
    socket.emit('start3');
    createBoardImages();
    socket.emit('board_ready');
}
function rolling() {
    if (diceAnimationInProgress) return;
    shouldShowRollButton = false;
    disableAllButtons();
    buyButton.style.display = 'none';
    sellButton.style.display = 'none';
    socket.emit('roll1');
    socket.emit('start3');
}
function nexting() {
    if (diceAnimationInProgress) return;
    shouldShowRollButton = true;
    disableAllButtons();
    socket.emit('next1');
    socket.emit('next2');
    nextButton.style.display = 'none';
}
function chatting() {
    text = document.getElementById('say').value;
    socket.emit('mess1');
    socket.emit('mess2', text);
}
function buying() {
    disableAllButtons();
```

```
    socket.emit('buy1');
    socket.emit('next1');
    socket.emit('next2');
}
function selling() {
    disableAllButtons();
    socket.emit('sell1');
    setTimeout(function() {
        nextButton.style.display = 'block';
    }, 500);
}
```

Лістинг коду (Функції керування грою):

```
function joiningGroup(room) {
    socket.emit('join1', room);
    joinB1.style.display = 'none';
    joinB2.style.display = 'none';
    joinB3.style.display = 'none';
    joinB4.style.display = 'none';
    joinGame.style.display = 'block';
    document.getElementById('username').style.display = 'block';
}
function joiningGame() {
    socket.emit('join2');
    let name = document.getElementById('username').value;
    socket.emit('join3', name);
    document.getElementById('username').style.display = 'none';
    joinGame.style.display = 'none';
    socket.emit('join4');
}


function starting() {
    socket.emit('start1');
    startButton.style.display = 'none';
    socket.emit('start2');
    socket.emit('start3');
    createBoardImages();
    socket.emit('board_ready');
}
function rolling() {
    if (diceAnimationInProgress) return;
    shouldShowRollButton = false;
    disableAllButtons();
    buyButton.style.display = 'none';
    sellButton.style.display = 'none';
    socket.emit('roll1');
    socket.emit('start3');
}
function nexting() {
    if (diceAnimationInProgress) return;
    shouldShowRollButton = true;
    disableAllButtons();
    socket.emit('next1');
    socket.emit('next2');
```

```
    nextButton.style.display = 'none';
}
function chatting() {
    text = document.getElementById('say').value;
    socket.emit('mess1');
    socket.emit('mess2', text);
}
function buying() {
    disableAllButtons();
    socket.emit('buy1');
    socket.emit('next1');
    socket.emit('next2');
}

function selling() {
    disableAllButtons();
    socket.emit('sell1');
    setTimeout(function() {
        nextButton.style.display = 'block';
    }, 500);
}
```

Лістинг коду (Функції руху та анімації):

```
function animateDice(die1Value, die2Value) {
    diceAnimationInProgress = true;
    overlay.style.display = 'block';
    diceContainer.style.display = 'block';

    let faces = document.querySelectorAll('.face');
    faces.forEach(face => {
        face.style.animation = 'face-glow 0.5s infinite';
    });

    dice1Element.style.left = "40%";
    dice2Element.style.right = "40%";

    dice1Element.style.transition = 'transform 0.05s linear';
    dice2Element.style.transition = 'transform 0.05s linear';

    dice1Element.style.transform = `rotate3d(${Math.random()*2-1},
${Math.random()*2-1}, ${Math.random()*2-1}, ${Math.random() * 360}deg)`;
    dice2Element.style.transform = `rotate3d(${Math.random()*2-1},
${Math.random()*2-1}, ${Math.random()*2-1}, ${Math.random() * 360}deg)`;

    setTimeout(() => {
        diceContainer.style.animation = 'dice-fall 0.8s cubic-bezier(0.17,
0.89, 0.32, 1.25) forwards';
        dice1Element.style.animation = 'dice1-spread 0.8s forwards';
        dice2Element.style.animation = 'dice2-spread 0.8s forwards';

        function chaosSpinDice() {
            dice1Element.style.transform = `rotate3d(${Math.random()*2-1},
${Math.random()*2-1}, ${Math.random()*2-1}, ${Math.random() * 720 +
360}deg)`;
            dice2Element.style.transform = `rotate3d(${Math.random()*2-1},
${Math.random()*2-1}, ${Math.random()*2-1}, ${Math.random() * 720 +
```

```
360}deg)`;
        }

        let spinInterval = setInterval(chaosSpinDice, 60);

        setTimeout(() => {
            clearInterval(spinInterval);
            dice1Element.style.transition = 'transform 0.6s cubic-
bezier(0.1, 0.8, 0.2, 1)';
            dice2Element.style.transition = 'transform 0.6s cubic-
bezier(0.1, 0.8, 0.2, 1)';
            dice1Element.style.transform = getDiceTransform(die1Value);
            dice2Element.style.transform = getDiceTransform(die2Value);

            faces.forEach(face => {
                face.style.animation = 'dice-pulse 0.8s infinite';
            });

            setTimeout(() => {
                dice1Element.style.animation = '';
                dice2Element.style.animation = '';
                dice1Element.style.left = "0";
                dice2Element.style.right = "0";
                diceContainer.style.display = 'none';
                overlay.style.display = 'none';
                diceAnimationInProgress = false;
                socket.emit('roll2');

                if (nextButton && socket.id === activePlayerId) {
                    nextButton.style.display = 'block';
                }
            }, 800);
        }, 600);
    }, 50);
}

function movement() {
    let startX =
parseFloat(document.getElementById(circles[i]).getAttribute("cx"));
    let startY =
parseFloat(document.getElementById(circles[i]).getAttribute("cy"));
    let endX = 0;
    let endY = 0;

    if (spot === 0) { endX = 36; endY = 36; }
    if (spot === 1) { endX = 96; endY = 36; }
    if (spot === 2) { endX = 146; endY = 36; }
    if (spot === 3) { endX = 196; endY = 36; }
    if (spot === 4) { endX = 246; endY = 36; }
    if (spot === 5) { endX = 296; endY = 36; }
    if (spot === 6) { endX = 346; endY = 36; }
    if (spot === 7) { endX = 396; endY = 36; }
    if (spot === 8) { endX = 446; endY = 36; }
    if (spot === 9) { endX = 496; endY = 36; }
    if (spot === 10) { endX = 556; endY = 36; }
    if (spot === 11) { endX = 556; endY = 96; }
    if (spot === 12) { endX = 556; endY = 146; }
    if (spot === 13) { endX = 556; endY = 196; }
```

```
    if (spot === 14) { endX = 556; endY = 246; }
    if (spot === 15) { endX = 556; endY = 296; }
    if (spot === 16) { endX = 556; endY = 346; }
    if (spot === 17) { endX = 556; endY = 396; }
    if (spot === 18) { endX = 556; endY = 446; }
    if (spot === 19) { endX = 556; endY = 496; }
    if (spot === 20) { endX = 556; endY = 556; }
    if (spot === 21) { endX = 496; endY = 556; }
    if (spot === 22) { endX = 446; endY = 556; }
    if (spot === 23) { endX = 396; endY = 556; }
    if (spot === 24) { endX = 346; endY = 556; }
    if (spot === 25) { endX = 296; endY = 556; }
    if (spot === 26) { endX = 246; endY = 556; }
    if (spot === 27) { endX = 196; endY = 556; }
    if (spot === 28) { endX = 146; endY = 556; }
    if (spot === 29) { endX = 96; endY = 556; }
    if (spot === 30) { endX = 36; endY = 556; }
    if (spot === 31) { endX = 36; endY = 496; }
    if (spot === 32) { endX = 36; endY = 446; }
    if (spot === 33) { endX = 36; endY = 396; }
    if (spot === 34) { endX = 36; endY = 346; }
    if (spot === 35) { endX = 36; endY = 296; }
    if (spot === 36) { endX = 36; endY = 246; }
    if (spot === 37) { endX = 36; endY = 196; }
    if (spot === 38) { endX = 36; endY = 146; }
    if (spot === 39) { endX = 36; endY = 96; }

    let circle = document.getElementById(circles[i]);

    setTimeout(function() {
        circle.setAttribute("cx", endX);
        circle.setAttribute("cy", endY);

        setTimeout(function() {
            let svgBoard = document.getElementById('board');
            let clone = circle.cloneNode(true);
            circle.remove();
            svgBoard.appendChild(clone);
        }, 1000);
    }, diceAnimationInProgress ? 1600 : 0);
}
```

Лістинг коду (Функції економічної системи):

```
function game() {
    if (spot != 0 && spot != 2 && spot != 4 && spot != 8 && spot != 10 &&
spot != 13 && spot != 18 && spot != 20 && spot != 22 && spot != 27 && spot
!= 30 && spot != 32 && spot != 36 && spot != 38 && spot != 5 && spot != 15
&& spot != 25 && spot != 35){
        price = 60 + (spot*4);
        document.getElementById('price').innerHTML = 'Ціна: ' + price;
    }

    if (spot === 5 || spot === 15 || spot === 25 || spot === 35){
        price = 50 + spot*2;
        document.getElementById('price').innerHTML = 'Ціна: ' + price;
    }

    if (spot === 2 || spot === 22 || spot === 36) {
        wallets[i] = wallets[i] + airdrops[air];
        document.getElementById('price').innerHTML = 'Аірдроп: ' +
airdrops[air];
        bankfunds = bankfunds - airdrops[air];
        air++;
        if(air===6){
            air = 0;
        }
    }

    if (spot === 8 || spot === 18 || spot === 32){
        daytrade = 125;
        wallets[i] = wallets[i] + daytrade;
        document.getElementById('price').innerHTML = 'Трейдинг: ' +
daytrade;
        bankfunds = bankfunds - daytrade;
    }

    if (spot === 4 || spot === 13 || spot === 27 || spot === 38) {
        wallets[i] = wallets[i] - fees[q];
        document.getElementById('price').innerHTML = shits[pq] + fees[q];
        bankfunds = bankfunds + fees[q];
        pq++;
        q++;
        if(q===6){
            q = 0;
        }
        if(pq===7){
            pq=0;
        }
    }

    if (spot === 10) {
        document.getElementById('price').innerHTML = "Обережно, податкова
за вашою спиною";
    }

    if (spot === 30) {
        document.getElementById('price').innerHTML = "Штрафи податкової: -
300 !!!";
```

```
        wallets[i] = wallets[i] - 300;
        bankfunds = bankfunds + 300;

        setTimeout(function() {
            specialMoveToBannedToWarning();
        }, 1000);
    }
    document.getElementById('bankF').innerHTML = 'Банк: ' + bankfunds;
}

function upvotes() {
    if (posIDs[1] === posIDs[3]) {
        upvote[1] = 28;
        upvote[3] = 36;
    }
    if (posIDs[6] === posIDs[7] && posIDs[6] === posIDs[9]) {
        upvote[6] = 48;
        upvote[7] = 52;
        upvote[9] = 60;
    }
    if (posIDs[11] === posIDs[12] && posIDs[11] === posIDs[14]) {
        upvote[11] = 68;
        upvote[12] = 72;
        upvote[14] = 80;
    }
    if (posIDs[16] === posIDs[17] && posIDs[16] === posIDs[19]) {
        upvote[16] = 88;
        upvote[17] = 92;
        upvote[19] = 100;
    }
    if (posIDs[21] === posIDs[23] && posIDs[21] === posIDs[24]) {
        upvote[21] = 108;
        upvote[23] = 116;
        upvote[24] = 120;
    }
    if (posIDs[26] === posIDs[28] && posIDs[26] === posIDs[29]) {
        upvote[26] = 128;
        upvote[28] = 136;
        upvote[29] = 140;
    }
    if (posIDs[31] === posIDs[33] && posIDs[31] === posIDs[34]) {
        upvote[31] = 148;
        upvote[33] = 156;
        upvote[34] = 160;
    }
    if (posIDs[37] === posIDs[39]) {
        upvote[37] = 172;
        upvote[39] = 180;
    }
}
function props() {
    for(let brr = 0; brr<names.length; brr++){
        if(brr != i){
            if (posIDs[spot]===playerIDs[brr]){
                document.getElementById('price').innerHTML = 'Ваш коментар
на пості від ' + names[brr] +' був позначений негативно на ' +
upvote[spot]*2;
                wallets[i] = wallets[i] - upvote[spot]*2;
```

```
                wallets[brr] = wallets[brr] + upvote[spot]*2;
                document.getElementById('bankF').innerHTML = 'Банк: ' +
bankfunds;
            }

            if (posIDs[spot]===playerIDs[i]){
                document.getElementById('price').innerHTML = 'Вже ваше!
Позитивна оцінка: ' + upvote[spot];
                wallets[i] = wallets[i] + upvote[spot];
                bankfunds = bankfunds - upvote[spot];
                document.getElementById('bankF').innerHTML = 'Банк: ' +
bankfunds;
            }

            if (posIDs[spot]===playerIDs[i]*2){
                reward = price*3;
                document.getElementById('price').innerHTML = 'Нагорода: ' +
reward;
                wallets[i] = wallets[i] + reward;
                bankfunds = bankfunds - reward;
                posIDs[spot] = posIDs2[spot];
                document.getElementById('bankF').innerHTML = 'Банк: ' +
bankfunds;
                let places = positions[spot];
                document.getElementById(places).setAttribute('fill',
propColors[spot]);
            }
        }
    }
}
```

Лістинг коду (Функції системи обміну):

```
function performDirectTrade() {
    var yourFieldSelect = document.getElementById('yourFieldSelect');
    var otherFieldSelect = document.getElementById('otherFieldSelect');
    var otherPlayerSelect = document.getElementById('otherPlayerSelect');
    var yourMoneyInput = document.getElementById('yourMoneyInput');
    var otherMoneyInput = document.getElementById('otherMoneyInput');

    var yourFieldIndex = parseInt(yourFieldSelect.value);
    var otherFieldIndex = parseInt(otherFieldSelect.value);
    var otherPlayerIndex = parseInt(otherPlayerSelect.value);
    var yourMoney = parseInt(yourMoneyInput.value) || 0;
    var otherMoney = parseInt(otherMoneyInput.value) || 0;

    if (isNaN(otherPlayerIndex)) {
        alert('Спочатку виберіть гравця для обміну!');
        return;
    }

    if ((yourFieldIndex === -1 || isNaN(yourFieldIndex)) &&
        (otherFieldIndex === -1 || isNaN(otherFieldIndex)) &&
        yourMoney === 0 && otherMoney === 0) {
        alert('Виберіть принаймні одне поле або додайте гроші для
обміну!');
```

```
        return;
    }

    if (yourMoney > wallets[i]) {
        alert('У вас недостатньо грошей для цього обміну!');
        return;
    }

    if (otherMoney > wallets[otherPlayerIndex]) {
        alert('У гравця ' + names[otherPlayerIndex] + ' недостатньо грошей
для цього обміну!');
        return;
    }

    var tradeOffer = {
        fromPlayer: i,
        toPlayer: otherPlayerIndex,
        fromProperty: yourFieldIndex,
        toProperty: otherFieldIndex,
        fromMoney: yourMoney,
        toMoney: otherMoney
    };

    socket.emit('trade_offer', tradeOffer);

    let li = document.createElement('li');
    li.innerHTML = `<span style="color: #4a76d8;">Ви надіслали пропозицію
обміну гравцю ${names[otherPlayerIndex]}</span>`;
    document.getElementById('events').appendChild(li);

    document.getElementById('events').scrollTop =
document.getElementById('events').scrollHeight;

    closeDirectTradeModal();

    alert('Пропозицію обміну надіслано гравцю ' + names[otherPlayerIndex] +
'. Очікуйте відповіді.');
}

function acceptTradeOffer(offerId) {
    socket.emit('trade_accept', { offerId: offerId });
}

function declineTradeOffer(offerId) {
    socket.emit('trade_decline', { offerId: offerId });
}
```

Лістинг коду (Функції створення та оновлення ігрового поля):

```
function createBoardImages() {
    let oldImages = document.querySelectorAll('svg image');
    for (let i = 0; i < oldImages.length; i++) {
        oldImages[i].remove();
    }

    let svgBoard = document.getElementById('board');

    // Apple і Microsoft (верхній ряд)
    appendImage(svgBoard, 85, 15, 30, 40, '/images/apple.svg');
    appendImage(svgBoard, 185, 15, 30, 40, '/images/microsoft.svg');

    // Google, Amazon, Facebook (верхній ряд)
    appendImage(svgBoard, 335, 15, 30, 40, '/images/google.svg');
    appendImage(svgBoard, 385, 15, 30, 40, '/images/amazon.svg');
    appendImage(svgBoard, 485, 15, 30, 40, '/images/facebook.svg');

    // Twitter, LinkedIn, Instagram (правий ряд)
    appendImage(svgBoard, 540, 85, 40, 30, '/images/x.svg');
    appendImage(svgBoard, 540, 135, 40, 30, '/images/linkedin.svg');
    appendImage(svgBoard, 540, 235, 40, 30, '/images/instagram.svg');

    // Nvidia, AMD, Intel (правий ряд)
    appendImage(svgBoard, 540, 335, 40, 30, '/images/nvidia.svg');
    appendImage(svgBoard, 540, 385, 40, 30, '/images/amd.svg');
    appendImage(svgBoard, 540, 485, 40, 30, '/images/intel.svg');

    // IBM, Oracle, SAP (нижній ряд)
    appendImage(svgBoard, 485, 540, 30, 40, '/images/ibm.svg');
    appendImage(svgBoard, 385, 540, 30, 40, '/images/oracle.svg');
    appendImage(svgBoard, 335, 540, 30, 40, '/images/sap.svg');

    // Tesla, SpaceX, Samsung (нижній ряд)
    appendImage(svgBoard, 235, 540, 30, 40, '/images/tesla.svg');
    appendImage(svgBoard, 135, 540, 30, 40, '/images/spacex.svg');
    appendImage(svgBoard, 85, 540, 30, 40, '/images/samsung.svg');

    // Huawei, Xiaomi, Sony (лівий ряд)
    appendImage(svgBoard, 15, 485, 40, 30, '/images/huawei.svg');
    appendImage(svgBoard, 15, 385, 40, 30, '/images/xiaomi.svg');
    appendImage(svgBoard, 15, 335, 40, 30, '/images/sony.svg');

    // Cisco, Dell (лівий ряд)
    appendImage(svgBoard, 15, 185, 40, 30, '/images/cisco.svg');
    appendImage(svgBoard, 15, 85, 40, 30, '/images/dell.svg');

    // Ico поля
    appendImage(svgBoard, 285, 15, 30, 40, '/images/intel.svg');
    appendImage(svgBoard, 540, 285, 40, 30, '/images/intel.svg');
    appendImage(svgBoard, 285, 540, 30, 40, '/images/intel.svg');
    appendImage(svgBoard, 15, 285, 40, 30, '/images/intel.svg');

    addPlayerCirclesToFront();
}
```

```
function appendImage(parent, x, y, width, height, href) {
    let ns = 'http://www.w3.org/2000/svg';
    let image = document.createElementNS(ns, 'image');
    image.setAttributeNS(null, 'x', x);
    image.setAttributeNS(null, 'y', y);
    image.setAttributeNS(null, 'width', width);
    image.setAttributeNS(null, 'height', height);
    image.setAttributeNS(null, 'preserveAspectRatio', 'xMidYMid meet');
    image.setAttributeNS('http://www.w3.org/1999/xlink', 'xlink:href',
href);
    parent.appendChild(image);
}

function addPlayerCirclesToFront() {
    let oldCircles = document.querySelectorAll('#board circle');
    for (let i = 0; i < oldCircles.length; i++) {
        oldCircles[i].remove();
    }

    let svgBoard = document.getElementById('board');
    let circlesNS = 'http://www.w3.org/2000/svg';

    for (let i = 0; i <= 5; i++) {
        let circle = document.createElementNS(circlesNS, 'circle');
        circle.setAttribute('id', 'circle' + i);
        circle.setAttribute('cx', '36');
        circle.setAttribute('cy', '36');
        circle.setAttribute('r', '12');
        circle.setAttribute('stroke', 'white');
        circle.setAttribute('stroke-width', '3');

        let fillColor;
        switch(i) {
            case 0: fillColor = '#FFFFFF'; break;
            case 1: fillColor = '#000000'; break;
            case 2: fillColor = '#9900FF'; break;
            case 3: fillColor = '#FFFF00'; break;
            case 4: fillColor = '#FF66FF'; break;
            case 5: fillColor = '#00FFFF'; break;
            default: fillColor = '#AAAAAA';
        }

        circle.setAttribute('fill', fillColor);
        svgBoard.appendChild(circle);
    }
}
```

Лістинг коду (Функції керування станом гри):

```
function disableAllButtons() {
    rollButton.style.display = 'none';
    nextButton.style.display = 'none';
    buyButton.style.display = 'none';
    sellButton.style.display = 'none';
    document.getElementById('price').innerHTML = '';
}
function enableButtonsForCurrentState() {
    if (diceAnimationInProgress) return;

    if (names.length === 1 && names[0] && names[0].length > 0) {
        rollButton.style.display = 'none';
        nextButton.style.display = 'none';
        buyButton.style.display = 'none';
        sellButton.style.display = 'none';
        document.getElementById('price').innerHTML = 'ПЕРЕМОЖЕЦЬ';
        return;
    }

    if (wallets[i] < 0) {
        disableAllButtons();
        socket.emit('next1');
        socket.emit('next2');
        return;
    }

    if (socket.id === activePlayerId) {
        rollButton.style.display = 'block';
    } else {
        rollButton.style.display = 'none';
        nextButton.style.display = 'none';
    }
}

function updateActivePlayerControls() {
    if (rollButton) {
        if (socket.id === activePlayerId) {
            rollButton.style.display = 'block';
        } else {
            rollButton.style.display = 'none';
        }
    }

    if (nextButton) {
        if (socket.id === activePlayerId && !diceAnimationInProgress) {
            nextButton.style.display = 'block';
        } else {
            nextButton.style.display = 'none';
        }
    }
}
```

Лістинг коду (Обробники подій):

```
document.getElementById('say').addEventListener("keyup", function(event) {
    event.preventDefault();
    if (event.keyCode === 13) {
        chatButton.click();
    }
});
window.addEventListener('load', function() {
    socket.emit('board_ready');
    createBoardImages();
});

function scrollChatToBottom() {
    let chatContainer = document.getElementById('events');
    if (chatContainer) {
        chatContainer.scrollTop = chatContainer.scrollHeight;
    }
}
joinB1.addEventListener('click', joinG1);
joinB2.addEventListener('click', joinG2);
joinB3.addEventListener('click', joinG3);
joinB4.addEventListener('click', joinG4);
joinGame.addEventListener('click', joiningGame);
startButton.addEventListener('click', starting);
chatButton.addEventListener('click', chatting);
rollButton.addEventListener('click', rolling);
nextButton.addEventListener('click', nexting);
buyButton.addEventListener('click', buying);
sellButton.addEventListener('click', selling);
```

Лістинг коду (Сокет-події):

```
socket.on('join1', function(data){
    let mj = data;
    socket.on('room1', function(rr) {
        joinButtons[rr].innerHTML = joinButtonsText[rr] + ' ' +mj+'/6';
        if (mj>5){
            joinButtons[rr].disabled = true;
        }
    })
});
socket.on('sync_images', function() {
    createBoardImages();
});

socket.on('create_images', function() {
    createBoardImages();
    setTimeout(function() {
        let svgBoard = document.getElementById('board');
        for (let i = 0; i <= 5; i++) {
            let circle = document.getElementById('circle' + i);
            if (circle && svgBoard) {
                let clone = circle.cloneNode(true);
                circle.remove();
                svgBoard.appendChild(clone);
```

```
            }
        }
    }, 500);
});
socket.on('join2', function(data){
    i = data;
    playerIDs.push(playerIDsDefault[i]);
    wallets.push(walletsDefault[i]);
    walletLabels.push(walletLabelsDefault[i]);
    playerColors.push(playerColorsDefault[i]);
    counter.push(counterDefault[i]);
    circles.push(circlesDefault[i]);
});

socket.on('join3', function(name){
    names.push(name);
    document.getElementById('players').innerHTML = names;
    nums.push(name);
});

socket.on('join4', function(data){
    startButton.style.display = data;
});

socket.on('start1', function(data){
    chatdiv.style.display = data;
});
socket.on('start2', function(data){
    if (socket.id === activePlayerId) {
        rollButton.style.display = data;
    }
});
socket.on('start3', function(data){
    i = data;
    document.getElementById('nowPlaying').innerHTML = names[i];
    for (let fac1 = 0; fac1 < walletLabels.length; fac1++) {
        document.getElementById(walletLabels[fac1]).innerHTML =
wallets[fac1];
    }
    document.getElementById('bankF').innerHTML = 'Банк: ' + bankfunds;
    board.style.display = 'block';
});

socket.on('roll1', function(data){
    let diceData = data;
    animateDice(diceData.die1, diceData.die2);
    dice = diceData.total;
    document.getElementById('dice').innerHTML = 'DICE: ' + dice;

    counter[i] = counter[i] + dice;
    if (counter[i]>39){
        counter[i]=counter[i]-40;
    }

    spot = counter[i];
    document.getElementById('position').innerHTML = positions[spot];

    movement();
```

```
    game();
    upvotes();
    props();
});
socket.on('next1', function(data){
    i = data;
    setTimeout(function() {
        enableButtonsForCurrentState();
    }, 500);
});
socket.on('next2', function(data){
    i = data;
    document.getElementById('nowPlaying').innerHTML = names[i];
    document.getElementById('position').innerHTML = '';
    document.getElementById('dice').innerHTML = '';

    for (let fac1 = 0; fac1 < walletLabels.length; fac1++) {
        document.getElementById(walletLabels[fac1]).innerHTML =
wallets[fac1];
    }
    document.getElementById('bankF').innerHTML = 'Банк: ' + bankfunds;

    let e = i - 1;
    if(e<0) {
        e = e + playerIDs.length;
    }
    if (wallets[e]<0) {
        nums = nums.filter(nn => nn != names[e])
        for (let bankrupt = 0; bankrupt<40; bankrupt++){
            if (posIDs[bankrupt] === playerIDs[e] || posIDs[bankrupt] ===
playerIDs[e]*2) {
                posIDs[bankrupt] = posIDs2[bankrupt];
                let places = positions[bankrupt];
                document.getElementById(places).setAttribute('fill',
propColors[bankrupt]);
            }
        }
    }
    for (let krr = 0; krr<circles.length; krr++) {
        document.getElementById(circles[krr]).style.display = 'block';
    }

    buyButton.style.display = 'none';
    sellButton.style.display = 'none';
    document.getElementById('price').innerHTML = '';
});
socket.on('buy1', function(data){
    let place = positions[spot];
    document.getElementById(place).setAttribute('fill', playerColors[i]);

    investing();
    wallets[i] = wallets[i] - price;
    bankfunds = bankfunds + price;
    document.getElementById('bankF').innerHTML = 'Банк: ' + bankfunds;
});
socket.on('sell1', function(data){
    if (posIDs[spot] === playerIDs[i]){
        short();
```

```
        let place = positions[spot];
        document.getElementById(place).setAttribute('fill',
propColors[spot]);
        wallets[i] = wallets[i] + sellingPrice[spot];
    }
});
socket.on('mess1', function(data){
    i = data;
});

socket.on('mess2', function(data){
    let parent = document.getElementById('events');
    let child = document.createElement('child');
    child.innerHTML = names[i] + ': ' + data + "<br />";
    parent.appendChild(child);
    document.getElementById('say').value = '';

    setTimeout(scrollChatToBottom, 100);
});
socket.on('trade_offer_received', function(data) {
    activeTradeOffers.push(data.offer);

    let li = document.createElement('li');
    li.innerHTML = `
        <div style="border: 1px solid #4a76d8; padding: 10px; border-
radius: 5px; margin: 10px 0; background-color: #f0f7ff;">
            <p style="margin: 0 0 10px 0; font-weight: bold;">Нова
пропозиція обміну від гравця ${data.fromPlayerName}:</p>
            <div style="display: flex; justify-content: space-between;
margin-bottom: 10px;">
                <div style="flex: 1;">
                    <p style="margin: 0; font-weight: bold;">Вам
пропонують:</p>
                    <ul style="margin: 5px 0; padding-left: 20px;">
                        ${data.offer.fromProperty >= 0 ? `<li>Поле
${positions[data.offer.fromProperty]}</li>` : ''}
                        ${data.offer.fromMoney > 0 ?
`<li>${data.offer.fromMoney} грн</li>` : ''}
                    </ul>
                </div>
                <div style="flex: 1;">
                    <p style="margin: 0; font-weight: bold;">В обмін
на:</p>
                    <ul style="margin: 5px 0; padding-left: 20px;">
                        ${data.offer.toProperty >= 0 ? `<li>Поле
${positions[data.offer.toProperty]}</li>` : ''}
                        ${data.offer.toMoney > 0 ?
`<li>${data.offer.toMoney} грн</li>` : ''}
                    </ul>
                </div>
            </div>
            <div style="display: flex; justify-content: flex-end; gap:
10px;">
                <button onclick="acceptTradeOffer(${data.offer.id})"
style="b
```

ДОДАТОК Б. Тези конференції

УДК 004.4
Боднар М., ст.гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя, Україна

СУЧАСНІ ПІДХОДИ ДО РОЗРОБКИ ВЕБ-ІНТЕРАКТИВНИХ СТРАТЕГІЧНИХ ІГОР

Науковий керівник: к.т.н., доц. каф. ПІ Стоянов Ю.М. UDC 004.4
M. Bondar
Ternopil Ivan Puluj National Technical University

MODERN APPROACHES TO THE DEVELOPMENT OF WEB-INTERACTIVE STRATEGY GAMES

Supervisor: PhD Yurii Stoianov

Наявність онлайн-версій настільних ігор стала звичною частиною цифрової культури. Водночас, в україномовному сегменті інтернету вибір таких продуктів залишається обмеженим. Розробка знайомих ігор із використанням сучасних технологій, адаптованих під українську аудиторію, сприяє зміцненню позицій України в галузі веб- розробки, задовольняючи попит на локалізовані розважальні веб-застосунки.

Для реалізації стратегічної гри найліпше підходить поєднання легких, але ефективних інструментів, таких як Skeleton [1] для створення адаптивного дизайну та Alpine.js [2] для додання інтерактивності без використання важких JavaScript-бібліотек. Це дозволяє забезпечити швидкодію, простоту розробки й зручність подальшої підтримки.

Для забезпечення взаємодії гравців у режимі реального часу доцільно використовувати WebSocket — технологію двостороннього зв’язку, що забезпечує миттєвий обмін даними між клієнтом і сервером. Вона дозволяє ефективно реалізувати ключову механіку стратегічних ігор: зміни стану, дії гравців, черги ходів, тощо.

В рамках процедурного підходу, основний стан гри (інформація про гравців, їхні позиції на полі, баланси гаманців, власність на поля, стан банку тощо) зберігається в оперативній пам'яті (in-memory) на серверній стороні. Кожна активна ігрова сесія (стіл) підтримує свій власний стан. Процедури (функції) на сервері безпосередньо оперують цими даними в пам'яті для виконання ігрової логіки: обробки кидків кубиків, переміщення гравців, зміни балансів при купівлі/продажу чи сплаті комісій/ренти, обміну власності. Клієнтська частина (браузери гравців) отримує оновлення цього стану з сервера через Socket.IO та відображає його, підтримуючи локальну копію стану (як видно з JavaScript-коду в index.html), але сервер залишається основним джерелом правди. Цей підхід ефективний для швидкої синхронізації стану та обробки дій у реальному часі протягом однієї ігрової сесії.Архітектура застосунку базується на логічному розділенні функціональності: ігрове поле, панель керування, інформаційні вікна,

модулі обробки подій тощо. Така структура полегшує навігацію, масштабування й дозволяє легко оновлювати окремі компоненти без порушення роботи всієї системи.

Література:

1. Skeleton: A dead simple, responsive boilerplate URL:https://getskeleton.com
2. Reactive Web Interfaces the Lightweight Way with Alpine. Retrieved URL:https://alpinejs.dev

ДОДАТОК В. Диск з роботою