Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка методу та засобів побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури»

Виконав: студент IV курсу
групи 3ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Безкревний О. С.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. ПЗ Войтко В. В.
(прізвище та ініціали)

Рецензент: к.т.н., доцент каф. ОТ Городецька О. С.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ професор Романюк О.Н.
«06» 06 2025 р.

ВНТУ – 2025

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

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

Безкревному Олексію Сергійовичу

1. Тема роботи – розробка методу та засобів побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури.

Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи: 2 червня 2025.

3. Вихідні дані до роботи: середовище розробки – PyCharm, мови розробки – Python, JavaScript, HTML/CSS;

4. Зміст текстової частини: вступ, аналіз стану розвитку багатокористувацьких ігор з сесійною мультиплеєрною архітектурою та постановка задач розробки, розробка методу, моделі та алгоритмів роботи програми, розробка програмного забезпечення, тестування програми, висновки, список використаних джерел, додатки, ілюстративна частина.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науковий керівник бакалаврської кваліфікаційної роботи, тема; актуальність розробки; мета, об'єкт та предмет дослідження; постановка задачі; наукова новизна; практична цінність; аналіз аналогів; метод генерації прямокутників; діаграма класів моделі системи; загальний алгоритм роботи системи; блок-схема ігрової логіки; використані технології; функціонал розробленої системи; апробація та публікація результатів роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Войтко В. В., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку розробки методу та засобів побудови багатокористувацьких ігор з сесійною мультиплеєрною архітектурою та постановка задач розробки	25.03.25 – 11.04.25	Виконано
2	Розробка методів, моделі та алгоритмів роботи програми	14.04.25 – 25.04.25	Виконано
3	Розробка програмного забезпечення	28.04.25 – 16.05.25	Виконано
4	Тестування програми	19.05.25 – 23.05.25	Виконано
5	Оформлення матеріалів до захисту БКР	26.05.25 – 30.05.25	Виконано

Студент _____ **Безкревний О.С**
(підпис) (ініціали та прізвище

Керівник бакалаврської кваліфікаційної роботи _____ **Войтко В. В.**
(підпис) (ініціали та прізвище

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота містить 106 сторінок формату А4, на яких наведено 47 рисунків і 3 таблиці. Список використаних джерел складається з 20 найменувань.

У бакалаврській кваліфікаційній роботі було проведено аналіз сучасного стану та підходів до розробки багатокористувацьких ігор із сесійною серверною архітектурою. Було розглянуто існуючі рішення, їх функціональні можливості, визначено їхні переваги й недоліки, а також обґрунтовано необхідність створення власної реалізації, адаптованої до вимог ігрового процесу.

У роботі обґрунтовано вибір мови програмування, середовища розробки та технологій, використаних під час створення вебзастосунку. Розроблено серверну частину багатокористувацької гри, що геймплейно реалізує сесійну мультиплеєрну архітектуру, а також інтерфейс користувача для взаємодії з грою. Особливу увагу приділено покращенню випадковості генерації об'єктів у грі.

Для реалізації системи було використано мову програмування Python, JavaScript та інструменти для побудови веб сайту, зокрема HTML та CSS. У якості середовища розробки використано PyCharm.

Результати бакалаврської кваліфікаційної роботи можуть бути використані як основа для розробки ігрових систем з аналітичними можливостями або для подальших досліджень у галузі геймдизайну та візуалізації даних.

Ключові слова: сесійна мультиплеєрна архітектура, багатокористувацька гра, браузерна гра, логічне мислення.

ABSTRACT

The bachelor's thesis contains 106 pages of A4 format, which contain 47 figures and 3 tables. The list of used sources consists of 20 items.

The bachelor's thesis analyzed the current state and approaches to the development of multiplayer games with a session-based server architecture. Existing solutions, their functionality, their advantages and disadvantages were considered, and the need to create our own implementation adapted to the requirements of the game process was justified.

The work justified the choice of programming language, development environment and technologies used in creating a web application. The server part of the multiplayer game was developed, which implements the session multiplayer architecture in gameplay, as well as the user interface for interacting with the game. Special attention was paid to improving the randomness of object generation in the game.

The system was implemented using the Python programming language, JavaScript, and tools for building a website, in particular HTML and CSS. PyCharm was used as the development environment.

The results of the bachelor's thesis can be used as a basis for the development of game systems with analytical capabilities or for further research in the field of game design and data visualization.

Keywords: sessional multiplayer architecture, multiplayer game, browser game, logical thinking.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному цифровому світі багатокористувацькі ігри стали важливою складовою індустрії розваг, пропонуючи гравцям можливість взаємодіяти в реальному часі в межах віртуальних середовищ [1]. Розвиток мережевих технологій та зростання потужності обчислювальних пристроїв дозволяють створювати складні та високонавантажені системи, що підтримують одночасну участь великої кількості користувачів. У зв’язку з розвитком ігрової індустрії постає можливість створити одну з таких систем у сфері багатокористувацьких ігор, якою є сесійна мультиплеєрна архітектура, яка дозволяє організовувати ігровий процес у вигляді незалежних сесій, або, як їх ще називають, лобі, до яких можуть підключатися учасники [2]. Такий підхід сприяє гнучкому керуванню мережевими ресурсами та забезпечує якісний користувацький досвід. Проте реалізація сесійної архітектури пов’язана з низкою технічних викликів, зокрема з питаннями масштабування обчислювальних ресурсів сервера, обробки даних та управління підключеннями. Одним із ключових аспектів є забезпечення стабільної синхронізації стану гри між усіма учасниками, що вимагає ефективної обробки мережевого трафіку. Таким чином, впровадження сесійної мультиплеєрної архітектури не лише відповідає сучасним вимогам ігрової індустрії, а й є актуальним напрямом для подальших досліджень і розробок.

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

Мета та завдання дослідження. Метою роботи є процес покращення досвіду гравців у багатокористувацьких іграх за рахунок розробки власної гри на основі сесійної мультиплеєрної архітектури та впровадження алгоритму генерації

випадкових прямокутників, що дозволяє реалізувати гру, орієнтовану на розвиток логічного мислення, у мультиплеєрному режимі.

Відповідно до поставленої мети потрібно вирішити такі завдання:

– провести аналіз існуючих аналогів багатокористувацьких ігор;

– розробити модель та алгоритм роботи багатокористувацької гри на основі сесійної мультиплеєрної архітектури;

– розробити функціонал програми з можливістю підключення до випадкового лобі;

– можливість створення та підключення до конкретного лобі, який визначається унікальним ідентифікатором;

– можливість налаштовування створеної кімнати для гри за допомогою параметрів;

– функціонал для кастомізації внутрішньоігрового кольору блоків;

– можливість змінювати ім’я користувача;

– можливість спілкування між гравцями у внутрішньоігровому чаті.

Об'єктом дослідження є процес розробки багатокористувацької гри на основі сесійної мультиплеєрної архітектури.

Предметом дослідження є методи і засоби розробки багатокористувацької гри, яка буде основана на сесійній мультиплеєрній архітектурі з використанням мови програмування Python та середовища програмування PyCharm.

Методи дослідження. У процесі дослідження використовувалися наступні методи:

– методи побудови користувацького інтерфейсу на базі HTML, CSS та JavaScript для забезпечення інтерактивної взаємодії користувача з ігровим середовищем;

– метод симуляції масових підключень для тестування продуктивності та стійкості серверної частини під час високих навантажень;

– метод випадкової генерації прямокутників для забезпечення динамічного вирівнювання розподілу за площею та чесності ігрового процесу;

– метод створення вебсокет-з’єднань для забезпечення обміну даними в реальному часі між клієнтом і сервером.

Новизна отриманих результатів.

1. Подальшого розвитку набув метод випадкової генерації прямокутників, який, на відміну від стандартних підходів, дозволяє забезпечити рівномірний розподіл усіх можливих варіантів площ, навіть якщо деякі з них математично мають більше комбінацій сторін, що дозволяє досягти контрольованого, збалансованого динамічного вирівнювання розподілу прямокутників за площею при збереженні ефекту випадковості, що покращує ігровий процес та забезпечує рівні умови для гравців.

2. Подальшого розвитку отримала модель багатокористувацької гри на основі сесійної мультиплеєрної архітектури, яка, на відміну від існуючих, дозволяє користувачам створювати ігрові кімнати з власними налаштуваннями, приєднуватися до випадкових або запрошених сесій, що дозволяє користувачам грати у гру, яка розвиває логічне мислення, у мультиплеєрному режимі.

Практичне значення бакалаврської кваліфікаційної роботи полягає у можливості використання вебзастосунку, який може бути використаний як платформа для організації багатокористувацької взаємодії в реальному часі на основі сесійної архітектури. Розроблена система дозволяє користувачам швидко створювати та налаштовувати ігрові кімнати, забезпечує стабільну взаємодію та може бути використана як основа для створення інших онлайн ігор. Гра розвиває логічне мислення та допомагає поліпшити розумові навички. Рішення є універсальним і може застосовуватись як для навчальних, так і для розважальних

проєктів, а також як тестовий майданчик для дослідження продуктивності сесійної мультиплеєрної архітектури. Програмні модулі системи легко масштабуються, розширюються та адаптуються для різних жанрів ігрових механік, що робить її корисною не лише для кінцевих користувачів, але й для розробників у сфері геймдеву.

Особистий внесок здобувача. Усі результати, подані в бакалаврській кваліфікаційній роботі, були отримані автором особисто. У науковій праці [3], опублікованій у співавторстві, автору належать такі результати: таблиця порівняння функціональних характеристик аналогів, програмна реалізація розроблених функцій, алгоритм роботи серверу.

Апробація та публікація результатів проєкту. Результати бакалаврської кваліфікаційної роботи доповідалися на міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» та опубліковані у вигляді тез доповідей на цій конференції.

Публікації. Результати дослідження були опубліковані у матеріалах конференції [3] – «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Аналіз. У пояснювальній записці до бакалаврської кваліфікаційної роботи було розглянуто чотири розділи та було використано 20 літературних джерел.

1 АНАЛІЗ СТАНУ РОЗВИТКУ БАГАТОКОРИСТУВАЦЬКИХ ІГОР З СЕСІЙНОЮ МУЛЬТИПЛЕЄРНОЮ АРХІТЕКТУРОЮ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз предметної області

У сучасних умовах стрімкого розвитку комп’ютерних технологій та мережевої інфраструктури, багатокористувацькі ігри займають провідне місце в індустрії розваг та програмного забезпечення. Зростання попиту на інтерактивні ігрові продукти стимулює розробників до створення нових архітектурних підходів, що забезпечують масштабованість, стабільність і ефективну взаємодію між великою кількістю гравців у режимі реального часу [4].

Одним із перспективних підходів до побудови таких систем є використання кімнатної серверної архітектури, яка передбачає розподіл користувачів на окремі логічні кімнати (сесії), де вони можуть взаємодіяти в межах одного ігрового простору, так званого лобі. Такий підхід дозволяє знизити навантаження на сервер, підвищити продуктивність системи та спростити управління ігровими подіями в межах конкретної кімнати [5].

Сесійна архітектура особливо ефективна у випадках, коли гра передбачає обмежену кількість учасників у межах однієї сесії (наприклад, PvP-матчі або кооперативні режими). Завдяки цьому підходу можна реалізувати гнучке масштабування, динамічне створення і видалення кімнат, а також ізоляцію ігрових подій, що дозволяє уникнути конфліктів між різними групами користувачів [6].

Таким чином, дослідження та реалізація кімнатної серверної архітектури у контексті створення багатокористувацької гри є актуальним напрямом, що дозволяє забезпечити високу продуктивність, зручність масштабування та якісну взаємодію між гравцями. Це відкриває нові можливості для інновацій у сфері геймдеву, сприяє розвитку технологій реального часу та покращенню досвіду користувачів.

1.2 Аналіз аналогів розроблюваного програмного застосунку

Сесійна мультиплеєрна архітектура не нова, на сьогодні існує багато ігор з такою архітектурою. Кожна з них має свої особливості, проте також містить деякі недоліки, які можна усунути в рамках запропонованого проєкту. Проведемо аналіз популярних вебігор з лобі системами: Diep.io, Krunker.io, Territories.

Diep.io – багатокористувацька онлайн гра, в якій гравці керують танками, знищують об'єкти та противників, заробляють досвід і покращують характеристики своєї машини [7]. Головною перевагою Diep.io є простота геймплею, що поєднується з глибокою системою прокачки (рис. 1.1). Проте гра має значні недоліки, зокрема, перевантаження серверів під час одночасної участі великої кількості гравців, що може призводити до збоїв роботи програми. Також значним недоліком є нерівномірний баланс між гравцями, оскільки новачки часто стикаються з досвідченими суперниками, що ускладнює комфортний вхід у гру (рисунок 1.2).

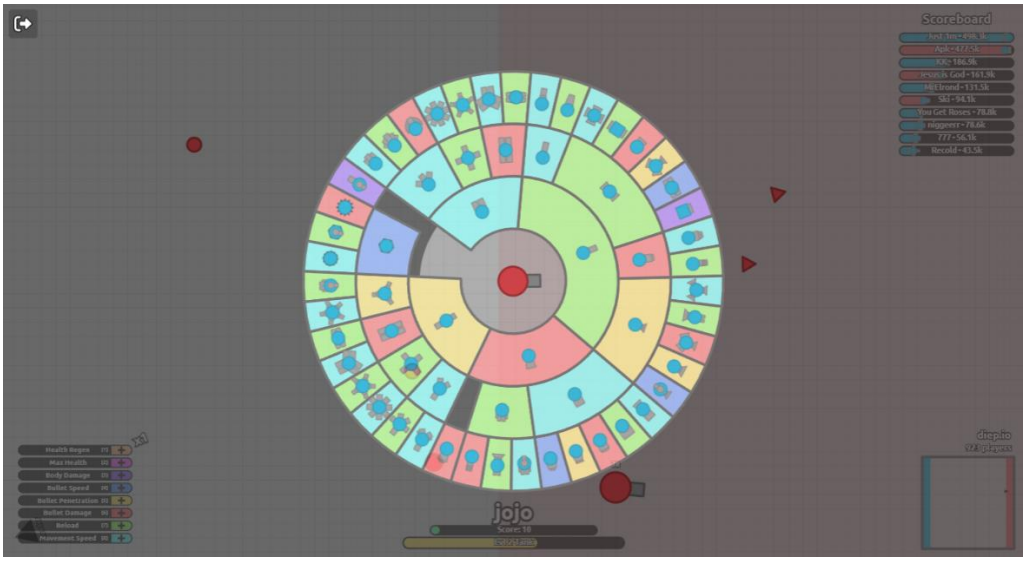


Рисунок 1.1 – Система прокачки в diep.io

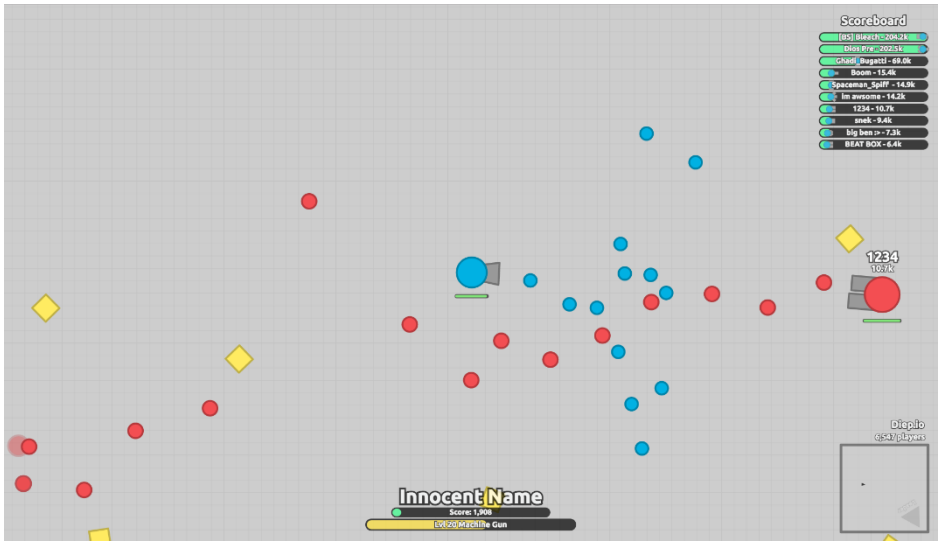


Рисунок 1.2 – Геймплей diep.io

Krunker.io – це браузерний шутер від першої особи, що пропонує швидкий геймплей у стилі класичних арена-шутерів [8]. Його перевага – плавна анімація, добре оптимізований рушій та можливість створення власних карт (рис. 1.3).

Рисунок 1.3 – Геймплей Krunker.io

Головним недоліком є висока чутливість гри до якості інтернет-з’єднання. Через архітектуру, що не передбачає ефективної компенсації затримок, навіть незначні лаги можуть призводити до втрати контролю над персонажем, запізнення пострілів або несправедливих поразок. Це створює ситуації, у яких перемога або поразка залежать не від навичок гравця, а від якості його мережі. Такий дисбаланс особливо критичний для змагального ігрового процесу, де важлива кожна секунда реакції. Крім того, проблема читів є актуальною. Відсутність належної античіт-системи дозволяє зловмисникам використовувати скрипти для отримання переваги, що уможливлює шахрайські дії й погіршує користувацький досвід (рис. 1.4).

Рисунок 1.4 – Приклад читів в Krunker.io

Territories – це мобільна гра для розвитку логічного мислення, розроблена Олексієм Безкревним у 2022 році [9]. Це покрокова гра на платформі Android у жанрі тайм-кілер/стратегія. Недоліками гри є необхідність встановлення застосунку на смартфон, відсутність онлайн мультиплеєру та деяка обмеженість геймплею (рис. 1.5).

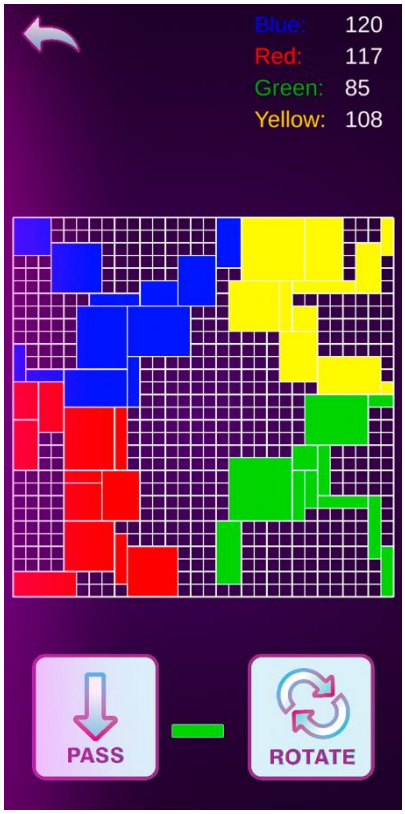


Рисунок 1.5 – Геймплей Territories

Перевагами ж Territories розвиток логічного мислення та планування дій. Особливо це актуально, оскільки гравцям необхідно взаємодіяти з іншими учасниками, координувати дії та адаптуватися до змінних умов гри.

Ці приклади демонструють важливість і перспективність використання сучасних підходів у сфері розробки багатокористувацьких ігор, забезпечуючи гравцям зручний та динамічний ігровий досвід. Створення подібних рішень вимагає глибокого розуміння очікувань цільової аудиторії та високої технічної майстерності для проєктування гнучких, надійних і масштабованих систем, які дозволяють ефективно керувати ігровими сесіями, підтримувати стабільний зв’язок між клієнтами та забезпечувати якісну взаємодію в реальному часі.

Для демонстрації особливостей розглянутих ігор їх переваги й недоліки зведено у таблицю порівняння (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристика онлайн платформ

Критерій порівняння	Diep.io	Krunker.io	Territories	Власна розробка
Чат для спілкування	0	1	0	1
Незалежність від затримок	0	0	1	1
Унеможливлення читів	1	0	1	1
Підтримка приватних лобі	1	1	0	1
Наявність командних режимів	1	1	0	1
Сумарний коефіцієнт	3	3	2	5

Проаналізувавши переваги і недоліки розглянутих ігор (табл. 1), можна зазначити, що сумарний коефіцієнт ефективності власної розробки за обраними критеріями на 60% (100% - 2/5*100% = 60%) вищий за Territories і на 40% (100% - 3/5*100% = 40%) вищий за Krunker.io та Diep.io, що підтверджує доцільність власної розробки мультиплеєрної гри.

Розроблена вебгра дозволяє провести тестування можливостей сесійної мультиплеєрної архітектури. Вебгра дозволяє створювати кімнату зі своїми налаштуваннями, приєднуватися до спеціально створеної чи випадково обраної кімнати. Розроблена гра також може слугувати експериментальним майданчиком для тестування нових підходів до організації сесійної мультиплеєрної архітектури, що може бути корисним для подальшого розвитку онлайн-ігор та мережевих сервісів.

1.3 Аналіз методів розв’язання задачі

Проєктування багатокористувацької гри вимагає ретельного вибору архітектурних та технологічних рішень для забезпечення стабільної та ефективної взаємодії між гравцями в режимі реального часу. Існує кілька підходів до реалізації

мережевої частини таких ігор, кожен з яких має свої сильні сторони та обмеження, що впливають на масштабованість, продуктивність і зручність розробки.

Монолітна архітектура передбачає централізоване оброблення всієї логіки гри в межах одного серверного застосунку. Такий підхід значно спрощує процес розробки, особливо на ранніх етапах, дозволяє уникнути надлишкової складності, пов'язаної з координацією між модулями, і забезпечує легший контроль за внутрішнім станом системи. При належній оптимізації монолітний сервер здатен обробляти десятки або навіть сотні одночасних з’єднань, що робить його ефективним рішенням для невеликих або середніх за навантаженням проєктів.

Для забезпечення ізоляції між групами гравців у межах однієї серверної системи використовується ігровий поділ на логічні кімнати або лобі, в яких відбувається незалежна взаємодія користувачів. Такий підхід дозволяє ефективно керувати геймплейною частиною гри без необхідності фізичного розділення на кілька серверів або процесів. Він також спрощує синхронізацію подій у межах кімнат, зменшує ризик конфліктів і полегшує масштабування структури гри в межах монолітної архітектури.

Серед альтернативних підходів можна розглядати мікросервісну архітектуру або використання сторонніх платформ для багатокористувацької взаємодії, однак вони часто вимагають значно більше ресурсів, технічної складності та інфраструктурної підтримки. У контексті веб-ігрового застосунку з акцентом на лобі-систему, монолітна модель виявляється раціональним і збалансованим рішенням.

У результаті аналізу було обрано монолітну архітектуру з внутрішнім розподілом гравців по окремих лобі. Це дозволяє зберегти цілісність системи, зменшити технічні витрати та забезпечити ефективне керування ігровим процесом у режимі реального часу.

1.4 Постановка задач дослідження

Після аналізу існуючих підходів до розробки багатокористувацьких ігор, було визначено основні напрямки для реалізації ефективного програмного рішення:

– розробити ігрову архітектуру, яка дозволяє логічно розділяти користувачів на ізольовані сесії всередині єдиного серверного середовища, забезпечуючи стабільну взаємодію між учасниками кожної сесії;

– розробити гру за такими правилами: є поле з клітинками, на якому почергово ходять 4 гравці. Кожному дається випадково згенерований прямокутник, який він має розмістити біля своєї території. Гравцю нараховуються бали розміром в кількість клітинок, які зайняв прямокутник. Якщо ж прямокутник не вдається розмістити або гравець не хоче його ставити, то у гравця віднімаються бали і він пропускає свій хід. Також можуть вводитись додаткові правила залежно від налаштувань кімнати, в якій проводиться гра. Перемагає той, хто набере найбільшу кількість балів;

– створити зручний і зрозумілий графічний інтерфейс, орієнтований на веб-платформу та адаптований до різних розширень екранів і типів пристроїв;

– реалізувати основні механіки гри з урахуванням багатокористувацької взаємодії, включаючи синхронізацію стану гри, обробку подій у режимі реального часу та керування сесіями;

– провести всебічне тестування функціональності гри для перевірки стабільності та зручності використання у реальних умовах.

Поставлені завдання мають на меті створення технічно надійної та масштабованої багатокористувацької гри, побудованої на основі сесійної мультиплеєрної архітектури. Такий підхід дозволяє ефективно організовувати окремі ігрові сесії, забезпечуючи стабільну взаємодію гравців без перевантаження сервера.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було здійснено аналіз сучасних веб-ігрових платформ, зокрема таких, як Krunker, Diep.io та Territories.

Дослідження цих рішень виявило, що попри їхню популярність і динамічний ігровий процес, вони мають низку недоліків, які створюють бар’єри для нових або випадкових користувачів. Серед основних проблем варто відзначити високу концентрацію досвідчених гравців, що створює нерівні умови гри, наявність чітингу через відкритість клієнтської частини або слабкий захист, залежність від постійного онлайн-з'єднання, а також складність в доступності встановлення.

Ці фактори підкреслюють необхідність створення нової гри, яка б врахувала потреби різних категорій гравців, забезпечила чесну багатокористувацьку взаємодію, а також мала зручний інтерфейс та універсальний доступ через веб-браузер без потреби в установці.

На основі проведеного аналізу було визначено перелік завдань для реалізації проєкту. Зокрема, передбачається розробка монолітного серверного застосунку з геймплейним поділом на сесії (лобі), що дозволяє обмежити взаємодію між окремими групами гравців. Також планується створення адаптивного інтерфейсу для гри у браузері, реалізація базових ігрових механік у реальному часі, а також проведення тестування для перевірки стабільності й зручності системи.

Результати цього етапу підтверджують, що розробка власного багатокористувацького веб додатку має потенціал для усунення ключових недоліків існуючих рішень, підвищення доступності та якості ігрового процесу для широкої аудиторії.

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМУ РОБОТИ СИСТЕМИ

2.1 Аналіз вхідних даних системи

У багатокористувацьких веб-іграх обробка даних відіграє критичну роль у забезпеченні узгодженості ігрового процесу, надійної синхронізації гравців і збереженні прогресу. Дані, які обробляються системою, охоплюють інформацію про гравців, їхню автентифікацію, дії в грі, результати матчів, а також технічні параметри для підключення й комунікації з сервером. У рамках розроблюваної гри з сервером, який поділяє гравців на окремі кімнати, система отримує та обробляє вхідні дані в реальному часі. Це включає вибір ігрової кімнати (лобі) та інформацію про гру в кімнаті.

Модуль ігрової логіки відповідає за обробку внутрішніх подій гри, перевірку правил, зміну станів об’єктів та взаємодію гравців з елементами ігрового середовища. Цей компонент функціонує на сервері та слугує основою для синхронізації ігрового стану між усіма учасниками сесії. Вхідними даними є дії гравців, отримані з клієнтської частини за допомогою JSON-повідомлення та поточний стан кімнати. Вихідними – оновлена інформація про гру, яка надсилається всім учасникам для відображення змін у реальному часі.

Модуль розподілу гравців відповідає за формування груп гравців у рамках одного ігрового простору. Система розподіляє користувачів за кімнатами відповідно до кількості зіграних ігор, щоб забезпечити збалансований геймплей. Вхідними даними є інтерфейс головної сторінка лобі, обраний режим і параметри матчу. Вихідними – запуск сесії та початок обміну повідомленнями між клієнтом і сервером.

Модуль чату між гравцями відповідає за обмін текстовими повідомленнями між учасниками однієї сесії в реальному часі. Система забезпечує передачу повідомлень через WebSocket-з'єднання, дозволяючи гравцям спілкуватися до, під

час та після завершення матчу. Вхідними даними є ідентифікатор сесії, нікнейм користувача та текст повідомлення. Вихідними – динамічне оновлення чату на клієнті та виведення повідомлення на екран.

Реалізація якісної обробки даних у багатокористувацькій грі дозволяє створити стабільне та динамічне ігрове середовище, де кожна дія має значення, а зібрана інформація допомагає формувати глибший досвід взаємодії користувачів із системою.

2.2 Розробка методу випадкової генерації прямокутників

У межах реалізації гри було розроблено спеціальний механізм генерації прямокутників, розміри яких варіюються в межах від 1 до 5 одиниць по кожній зі сторін. Ключова задача полягала в тому, щоб забезпечити рівномірний розподіл площ згенерованих прямокутників на великій вибірці, зберігаючи при цьому випадковість самої генерації. При простому рівномірному виборі розмірів прямокутника виникає статистичне зміщення розподілу площ. Наприклад, площа 1 (тобто прямокутник 1×1) може з'являтися значно рідше, ніж площа 6, яку можна отримати як 2×3 чи 3×2. Це створює нерівномірний розподіл площ, що може негативно впливати як на геймплей, так і на наочність ігрової візуалізаці у межах геймплейного досвіду.

Замість того, щоб одразу визначати площу чи розміри прямокутника за жорстким розподілом, було вирішено використати адаптивний алгоритм, що підлаштовує ймовірності вибору на основі попередніх генерацій. Таким чином, алгоритм у реальному часі аналізує поточний розподіл і компенсує перекоси, віддаючи перевагу менш представленим площам. Схему алгоритму показано на рисунку 2.1.

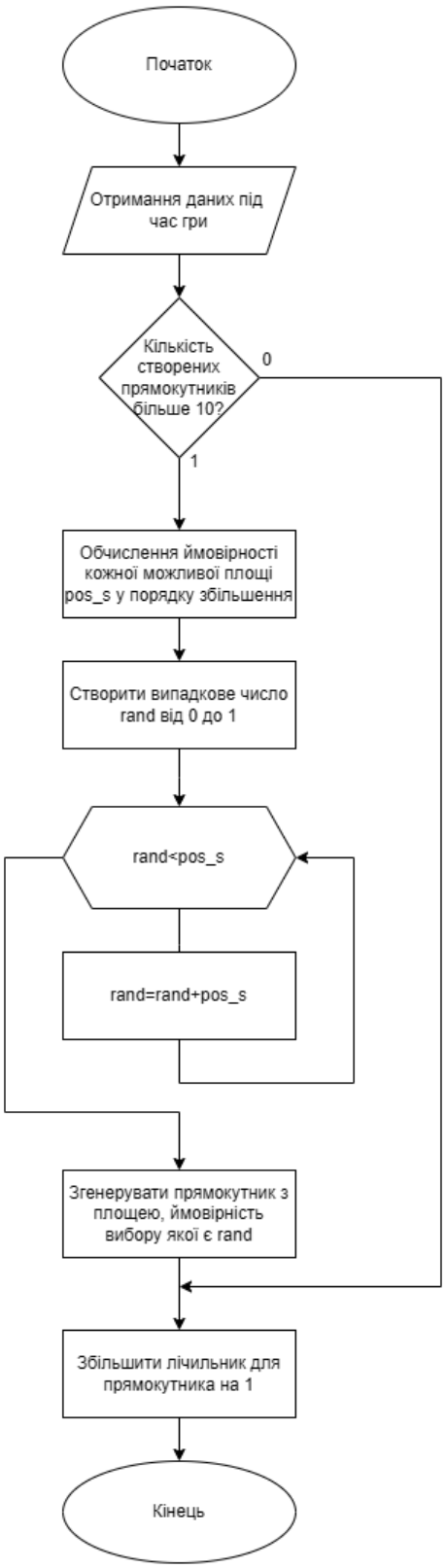


Рисунок 2.1 – Алгоритм випадкової генерації прямокутника

Метод випадкової генерації прямокутників працює за такою схемою:

1. Зі збереженого словника rect_counter, який містить кількість генерацій кожної площі, передаються статистичні дані у функцію createNum;
2. Якщо загальна кількість згенерованих прямокутників менша за 10, вибір сторін відбувається випадково;
3. Розраховуються ймовірності обернені ваги для кожної площі pos_s;
4. Використовується функція random.random() для генерації числа rand від 0 до 1;
5. Допоки rand менший за ймовірність pos_s, сумуються ймовірностей rand та pos_s для кожної з площ;
6. Присвоїти height та width числові значення довжини та ширини прямокутника із ймовірністю вибору rand;
7. Збільшити лічильник rect_counter на 1.

Кожного разу, коли генерується новий прямокутник, алгоритм переглядає, які площі вже часто з’являлися, і надає пріоритет менш представленим. Таким чином, система поступово саморегулюється і наближається до рівномірного розподілу.

2.3 Розробка моделі системи

На етапі розробки моделі системи визначаються ключові елементи архітектури, способи їх взаємодії, типи даних, які будуть передаватись у процесі гри, та принципи управління ігровими сесіями. Так як у межах цієї роботи було вирішено реалізувати багатокористувацьку гру із використанням сесійної серверної архітектури, то передбачається поділ гравців на ізольовані ігрові кімнати (сесії), де кожна з них функціонує як незалежна одиниця зі своєю логікою, полем гри та набором гравців.

Архітектура проєкту побудована на основі моделі «клієнт-сервер», де сервер відповідає за управління ігровими кімнатами, координацію гравців та обробку

логіки гри, а клієнтська частина – за візуалізацію ігрового процесу та взаємодію користувача з системою. Однією з ключових технологій, яка забезпечує стабільну та ефективну комунікацію між клієнтською і серверною частинами в реальному часі є протокол WebSocket [10]. Він був існує як відповідь на обмеження класичної моделі HTTP-запитів, де ініціатива завжди йде від клієнта, а сервер виступає лише у ролі пасивного обробника. У традиційному підході клієнт періодично надсилає запити до сервера для перевірки наявності нових даних. Це створює додаткове навантаження на сервер, збільшує затримку у доставці інформації та спричиняє зайві мережеві витрати, особливо при великій кількості одночасних клієнтів. Такий механізм є непридатним для систем, які потребують миттєвої реакції та постійної синхронізації стану, зокрема багатокористувацьких ігор. WebSocket вирішує ці проблеми шляхом встановлення постійного двостороннього з’єднання між клієнтом і сервером. Після початкового рукостискання, що відбувається через стандартний HTTP-запит, з’єднання переводиться у режим WebSocket, і обидві сторони можуть обмінюватися повідомленнями асинхронно, без необхідності повторного відкриття з’єднання для кожного обміну даними. Завдяки своїм перевагам WebSocket став стандартом для розробки сучасних веб додатків які працюють у реальному часі, такі як онлайн ігри, біржі, системи повідомлень, чати та панелі моніторингу.

Для опису логічної структури системи та взаємозв’язків між її основними класами була створена UML-діаграма класів (рис. 2.2). Діаграма включає ключові компоненти, зокрема Room (ігрова кімната), Player (гравець), Grid (ігрове поле) та Block (ігровий блок), і відображає їхні атрибути, методи та зв’язки між ними. Такий підхід дозволив формалізувати модель ігрового процесу, зробити її більш зрозумілою для подальшої реалізації та тестування.

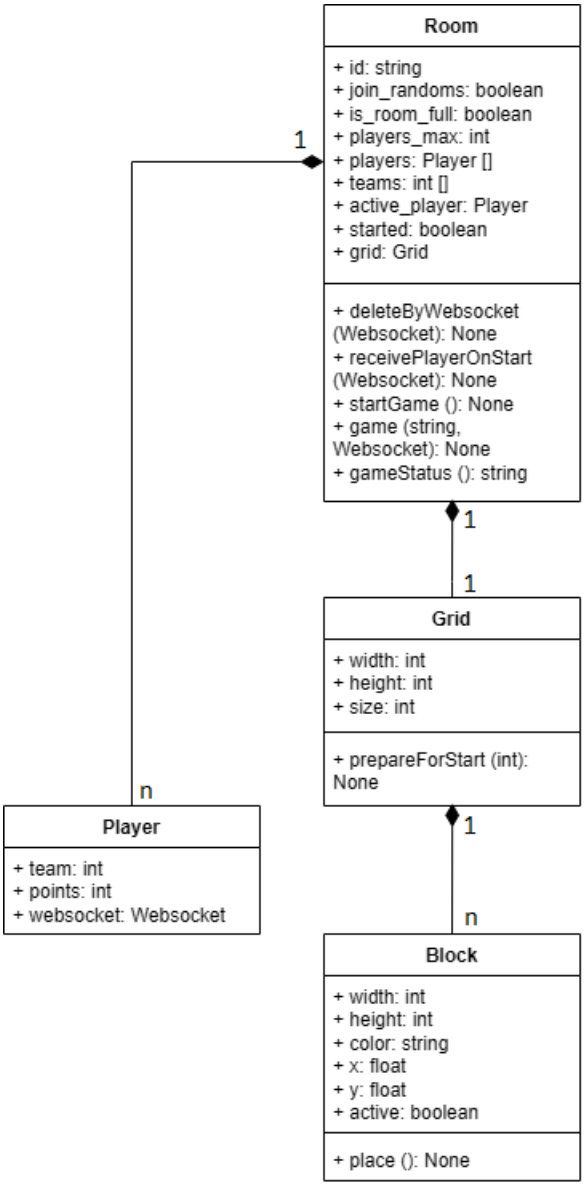


Рисунок 2.2 – Модель роботи системи у вигляді діаграми класів

На рисунку 2.2 наведено UML-діаграму класів, яка відображає основні компоненти архітектури багатокористувацької гри. Вона демонструє взаємозв'язки між класами Room, Grid, Block та Player, які реалізують основну логіку ігрової сесії.

Центральним елементом системи є клас Room, що представляє собою ігрову сесію. Він містить унікальний ідентифікатор id, логічні прапорці join_randoms і is_room_full, максимальну кількість гравців players_max, масив об'єктів Player,

перелік команд teams, а також посилання на активного гравця active_player. Атрибут started вказує на те, чи гра вже почалася, а grid — це зв'язок із класом Grid, який представляє ігрове поле.

Grid моделює ігрове поле кімнати. Він містить параметри розміру — width, height і size, що визначають сітку гри. Має метод prepareForStart(int), який готує поле до початку сесії. Клас Grid також має зв’язок 1:n із класом Block, тобто в ігрове поле можна поставити ігровий прямокутник (Block), в чому й заключається суть гри.

Block представляє собою ігровий прямокутник. Він має атрибути ширини та висоти (width, height), колір (color), координати (x, y) та логічний прапорець active, який визначає, чи активний блок. Метод place() використовується для розміщення елемента на ігровому полі.

Player це гравець в системі. Кожен гравець має належність до команди (team), кількість набраних очок (points) та WebSocket-з’єднання (websocket), через яке відбувається обмін даними між клієнтом і сервером. Зв’язок між Room і Player має тип 1:n, що відображає наявність багатьох гравців у межах однієї кімнати.

2.4 Розробка загального алгоритму роботи системи

Для розробки багатокористувацької гри на основі сесійної мультиплеєрної архітектури необхідно створити загальний алгоритм, що визначає логіку взаємодії користувача із системою, а також послідовність виконання дій на кожному етапі використання веб сайту. Цей алгоритм відображено у вигляді блок-схеми на рисунку 2.3, що наочно ілюструє увесь життєвий цикл користувача від входу на сайт та участі у грі до виходу з сайту.

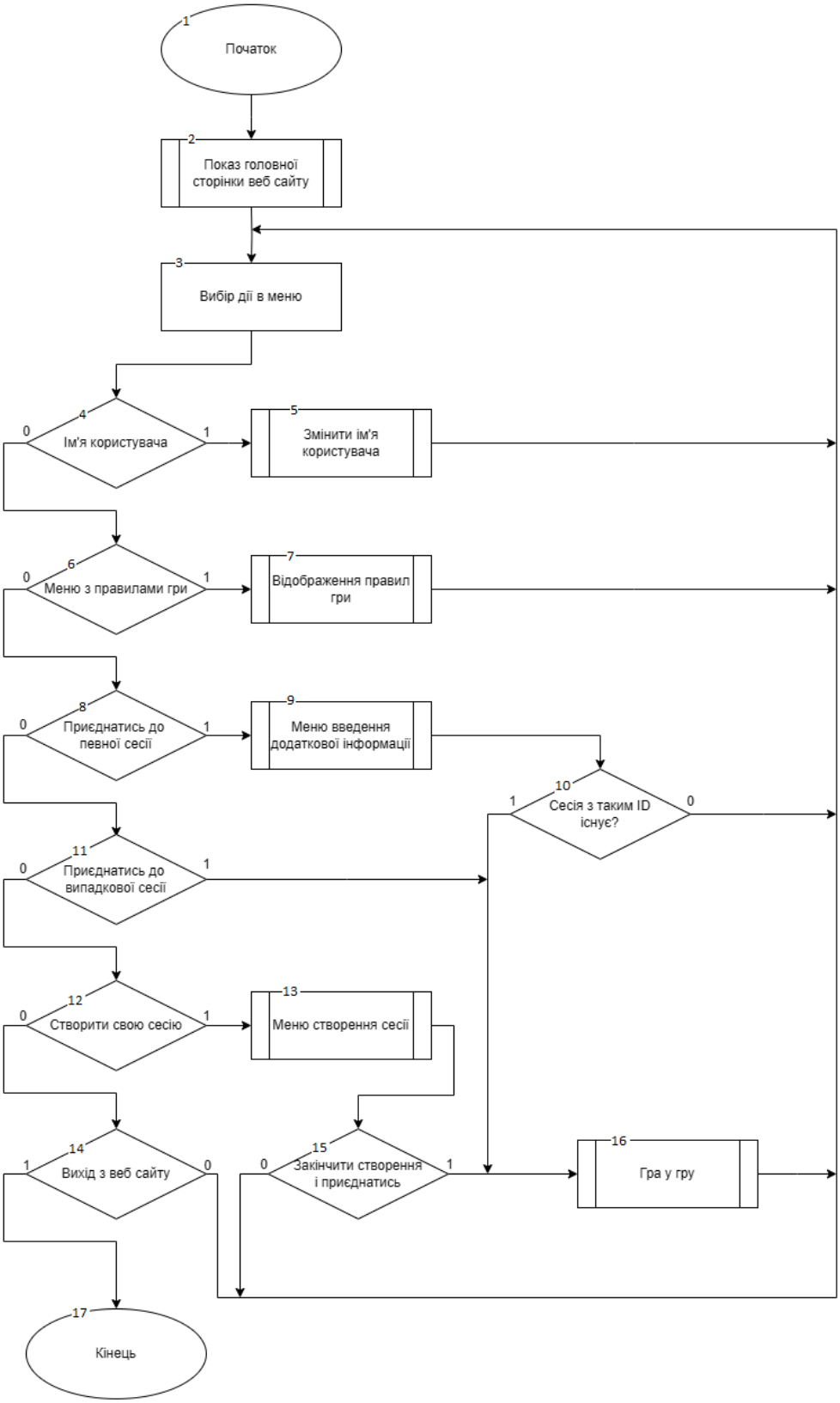


Рисунок 2.3 – Загальний алгоритм роботи веб сайту

Після того, як користувач заходить на веб сайт, він бачить головну сторінку та меню, у якому пропонується низка дій, наприклад, ввести ім’я користувача. Користувач може звернутись до відповідного меню, де він може вказати або змінити своє ім’я. Це необхідно для подальшої участі у сесіях гри, адже ім’я дозволяє ідентифікувати гравця серед інших учасників. Якщо ж ім’я не задане, воно буде присвоєне автоматично під час гри.

Після підтвердження імені користувач повертається до меню дій, де може ознайомитись із правилами гри. Обравши відповідний пункт меню, користувач переходить до сторінки з описом основних механік, правил участі та цілей гри. Ознайомлення з цими правилами є важливим етапом перед початком гри, особливо для новачків, оскільки дозволяє краще зрозуміти геймплей і логіку взаємодії з іншими гравцями.

Наступним можливим етапом є приєднання до ігрової сесії. Користувач може обрати один із кількох сценаріїв підключення. У разі наявності конкретного ID сесії, користувач має можливість ввести цей ідентифікатор і перейти до меню введення додаткової інформації, необхідної для підключення. Далі система перевіряє, чи існує сесія з вказаним ID. Якщо така сесія існує, то користувач приєднується до неї, і починається ігровий процес. Якщо ж ID виявляється неправильним або сесія вже заповнена – користувач повертається назад до меню вибору дій, щоб повторити спробу або обрати інший варіант.

Також веб сайт надає можливість приєднатися до випадкової сесії, що дозволяє швидко потрапити у гру без необхідності пошуку чи створення власної сесії. Такий підхід зменшує час очікування і спрощує взаємодію для користувачів, які хочуть одразу почати грати.

Ще один важливий функціональний блок алгоритму – створення власної сесії. Якщо користувач обирає цей варіант, він переходить до спеціального меню

створення, де задає параметри майбутньої сесії. Після завершення конфігурації користувач має змогу приєднатися до своєї створеної сесії, після чого також починається гра. Такий підхід дозволяє гравцям формувати свої власні ігрові кімнати під конкретні потреби або грати у закритому колі друзів.

У будь-який момент користувач може вийти з веб сайту, що призводить до завершення поточної взаємодії з системою. Це реалізується через закриття сайту у браузері.

2.5 Висновки

У другому розділі детально розглянуто основні аспекти розробки багатокористувацької гри з сесійною мультиплеєрною архітектурою.

Розглянуто принципи побудови веб інтерфейсу для гри з використанням HTML, CSS та JavaScript, що дозволяє забезпечити кросплатформенну доступність через браузер. Серверна логіка реалізована на мові програмування Python, яка обробляє створення та управління ігровими кімнатами, підключення користувачів до сесії, пересилання даних між клієнтами.

Загальний алгоритм роботи системи охоплює всі основні етапи: створення нової гри, приєднання гравців, запуск сесії, перегляд правил гри, завершення гри. Щоб уникнути математично нерівномірного розподілу площ прямокутників, реалізовано адаптивний механізм корекції: система аналізує частоту появи кожної площі та коригує ймовірності, щоб забезпечити рівномірний розподіл. Використання структури сесійної мультиплеєрної архітектури забезпечує масштабованість ігрового процесу та спрощує ізоляцію сесій, що є особливо важливим при реалізації одночасних багатокористувацьких ігор.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Python – високорівнева мова програмування з чітким і лаконічним синтаксисом, яка широко використовується в розробці серверних частин вебзастосунків [11]. Завдяки популярним фреймворкам, як-от FastAPI, Flask чи Django, Python дозволяє швидко реалізувати WebSocket-з’єднання, маршрутизацію запитів, авторизацію користувачів та обробку внутрішньої логіки гри.

Особливою перевагою Python є велика кількість бібліотек для обробки та візуалізації статистичних даних, зокрема Pandas, Matplotlib, Plotly, що дозволяє зручно збирати інформацію після завершення гри та показувати користувачу графіки, які відображають, наприклад, ступінь випадковості генерації елементів. Простота інтеграції з базами даних (наприклад, через SQLAlchemy або Firebase SDK) робить Python ефективним інструментом для зберігання результатів сесій, логів дій користувача, координат та подій гри.

Go (Golang) – це компільована мова програмування від Google, створена для високопродуктивних і масштабованих серверних рішень [12]. Go має вбудовану підтримку конкурентності (goroutines), що робить її ідеальним вибором для обробки великої кількості WebSocket-з’єднань. Вона часто використовується в ігрових серверних частинах або мікросервісах. Для збору та аналізу статистики Go теж надає необхідні бібліотеки, хоча не така зручна, як Python у плані візуалізації даних.

Rust – це системна мова програмування, яка поєднує високу швидкодію з безпечним управлінням пам’яттю [13]. Rust ідеально підходить для критично важливих частин бекенду гри, де важлива стабільність і продуктивність. Для роботи з WebSocket існують бібліотеки на кшталт Tokio та Warp, які дозволяють обробляти з’єднання ефективно. Хоча Rust менш зручний для швидкого прототипування або

аналітики, його використання доцільне у великих системах з високим навантаженням.

Java – класична об’єктно-орієнтована мова, яка традиційно використовується для серверних рішень [14]. За допомогою фреймворків таких як Spring Boot, Java дозволяє реалізувати WebSocket-сервер, обробляти дані користувачів та генерувати звіти. Java добре масштабується і широко використовується в індустрії. Водночас, розробка на Java часто займає більше часу порівняно з Python або Go, через більшу "вагу" коду та конфігурацій.

Для того, щоб обрати мову програмування, було створено порівняльну таблицю мов програмування (таблиця 3.1).

Таблиця 3.1 – Таблиця порівняння функціональних характеристик мов програмування

Характеристика	Go	Rust	Java	Python
Асинхронне програмування	1	1	1	1
Запуск без компіляції	1	0	0	1
Збирач сміття	1	0	1	1
Динамічна типізація	0	0	0	1
Запуск з командного рядка	1	0	0	1
Підтримка інтерактивних середовищ	0	1	0	1
Загальний коефіцієнт	4	2	2	6

Отже, для розробки багатокористувацької гри з кімнатною серверною архітектурою було обрано саме мову програмування Python, адже вона має найбільший загальний коефіцієнт серед інших мов програмування.

Python підтримується великою кількістю середовищ розробки, які забезпечують зручну роботу з кодом, відлагодженням, тестуванням та інтеграцією зі сторонніми бібліотеками. Розглянемо чотири найпопулярніші середовища для програмування на Python.

PyCharm – це потужне інтегроване середовище розробки (IDE), створене компанією JetBrains спеціально для Python [15]. Воно підтримує автоматичне завершення коду, рефакторинг, навігацію по проєкту, інтеграцію з системами контролю версій (наприклад, Git), вбудоване тестування, запуск скриптів, підтримку Docker, Jupyter Notebook, середовище віртуальних інтерпретаторів (venv) та багато іншого. PyCharm існує у двох версіях – Community (безкоштовна) та Professional (комерційна з додатковим функціоналом). Включає в себе інструменти для управління залежностями та інтеграцію з популярними фреймворками, такими як Django та Flask, що робить його ідеальним для розробки складних веб-додатків. Завдяки потужним засобам для налагодження та аналізу коду, PyCharm дозволяє ефективно працювати з великими кодовими базами та покращувати продуктивність команди розробників (рис. 3.1).

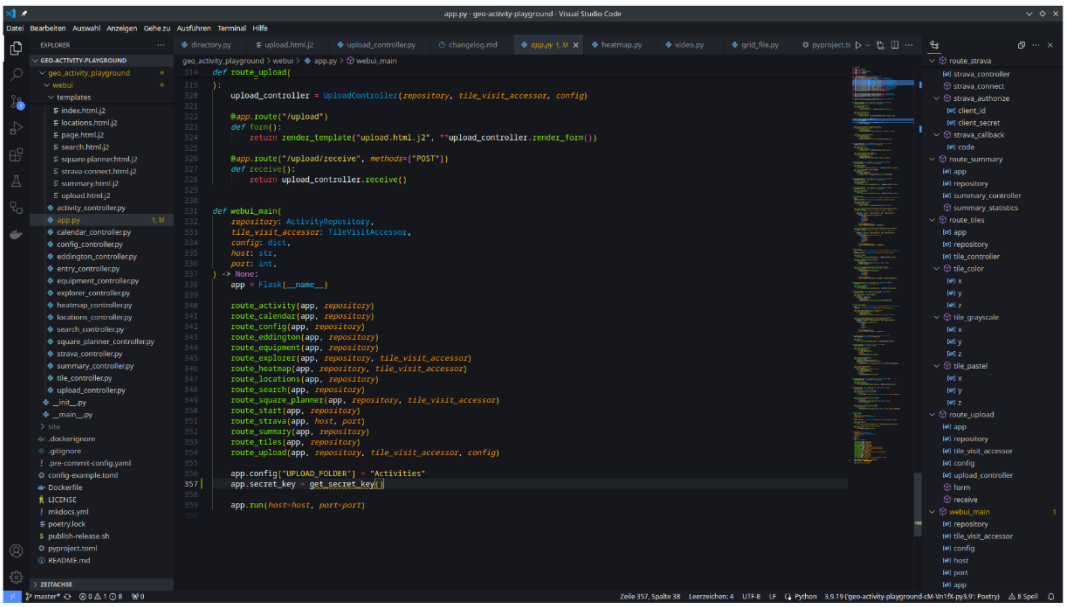


Рисунок 3.1 – Середовище розробки PyCharm

Visual Studio Code – це редактор коду з підтримкою багатьох мов програмування, включаючи Python, завдяки офіційному розширенню від Microsoft [16]. VS Code пропонує підсвічування синтаксису, автодоповнення, інтеграцію з Git, відлагодження, вбудований термінал, Jupyter-блокноти та підтримку віртуальних середовищ. Воно також забезпечує гнучкість завдяки широкому спектру плагінів і налаштувань, що дозволяють кастомізувати робоче середовище під конкретні вимоги проекту. Завдяки підтримці багатьох розширень, таких як для Docker та CI/CD, VS Code ідеально підходить для розробки як невеликих проєктів, так і великих корпоративних рішень. Це середовище є зручним як для початківців, так і для досвідчених розробників завдяки своїй легкості та швидкості роботи (рисунок 3.2).

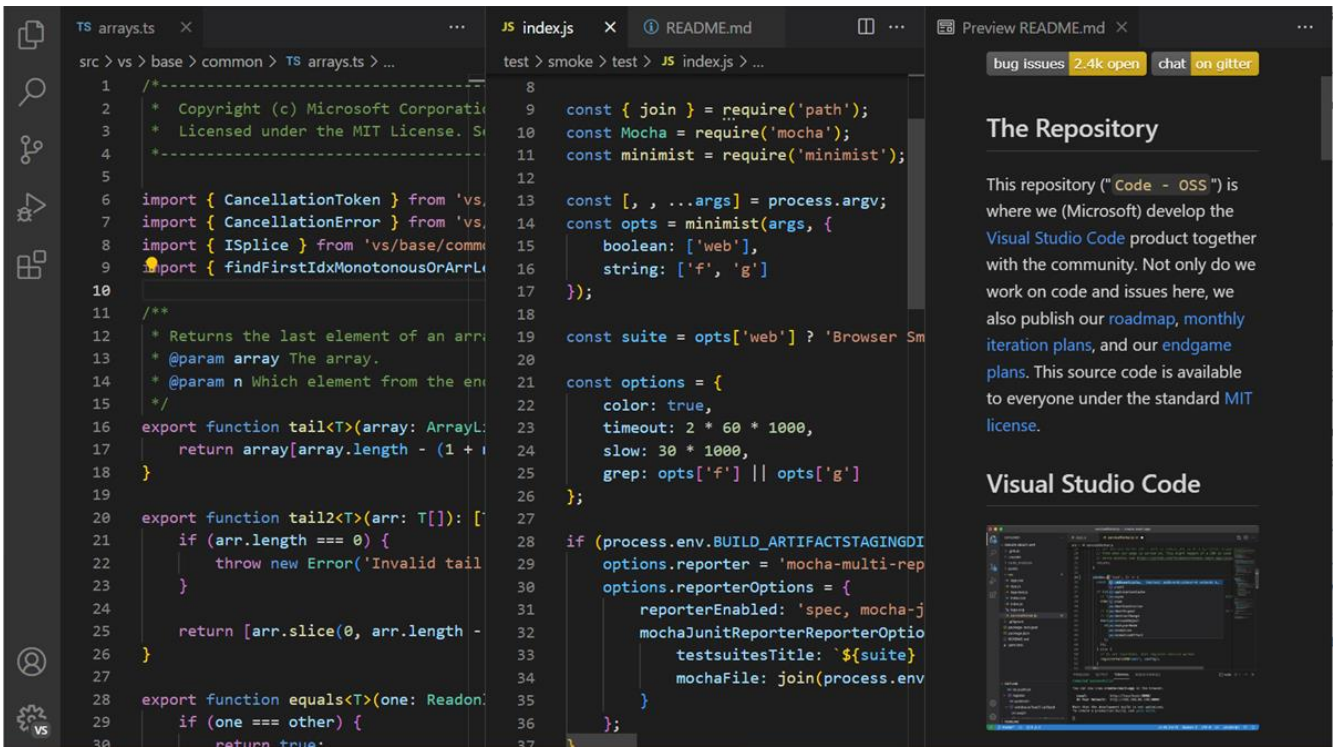


Рисунок 3.2 – Середовище розробки Visual Studio Code

Jupyter Notebook – це інтерактивне середовище для написання та виконання Python-коду у вигляді клітинок, де також можна вставляти текст у форматі Markdown, формули LaTeX, графіки, таблиці тощо [17]. Воно широко використовується у сфері аналізу даних, машинного навчання та дослідницької діяльності. Завдяки інтерактивному характеру середовище дозволяє миттєво бачити результат виконання кожної клітинки, що є надзвичайно зручним при роботі з великими обсягами даних та при тестуванні алгоритмів. Крім того, Jupyter підтримує інтеграцію з різними бібліотеками, такими як pandas, matplotlib, seaborn, scikit-learn, що дозволяє ефективно працювати з даними та побудовувати моделі машинного навчання (рис. 3.3).

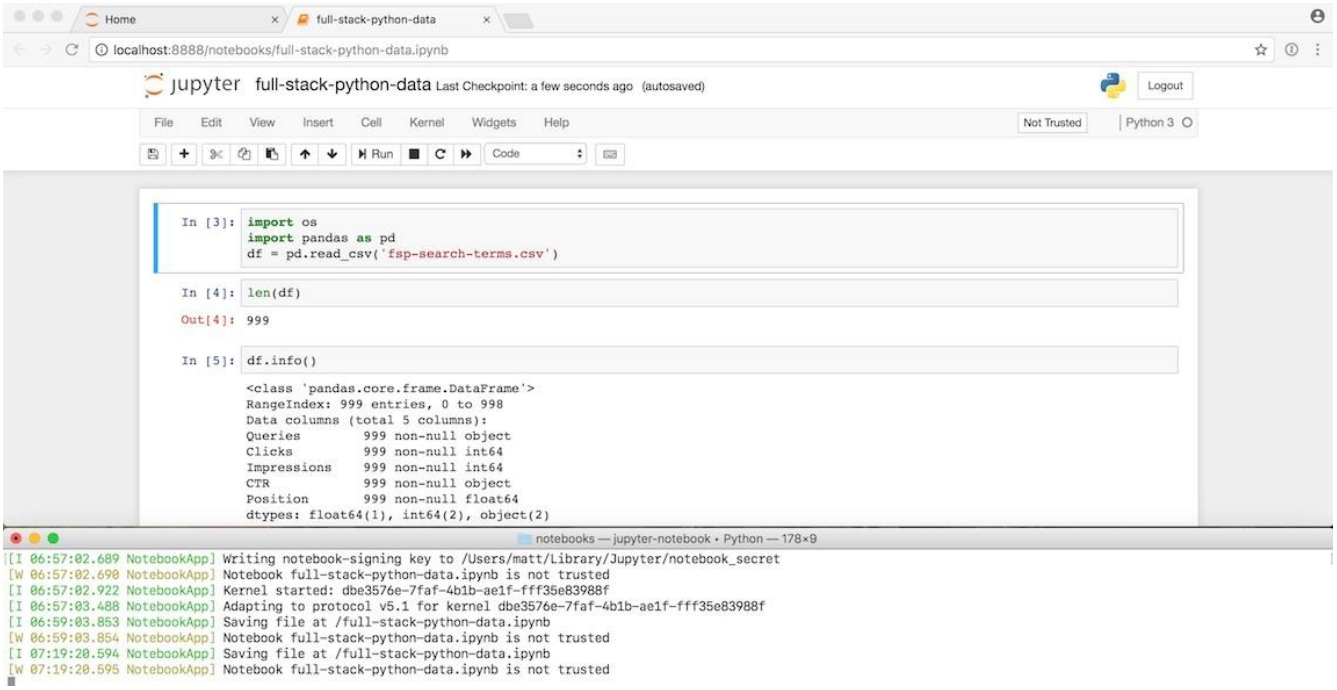


Рисунок 3.3 – Середовище розробки Jupyter Notebook

Thonny – це просте середовище розробки, орієнтоване на початківців у програмуванні [18]. Воно має мінімалістичний інтерфейс, вбудований інтерпретатор Python, зручну систему відлагодження, покрокове виконання коду та візуалізацію змінних. Завдяки цьому, Thonny ідеально підходить для навчання

основам програмування та роботи з Python, не перевантажуючи користувача зайвими функціями. Вбудована система дебагінгу дозволяє відслідковувати виконання коду покроково, що допомагає зрозуміти, як змінюються значення змінних і які саме операції виконуються на кожному етапі. Також, наявність вбудованого інтерпретатора полегшує налаштування середовища, що робить Thonny ще зручнішим для новачків. Крім того, Thonny підтримує автоматичне виявлення помилок синтаксису та підсвічування коду, що значно покращує зручність написання програм. Середовище також підтримує встановлення сторонніх бібліотек через простий інтерфейс, що дозволяє розширювати функціональність Python-проєктів без складної конфігурації (рис. 3.4).

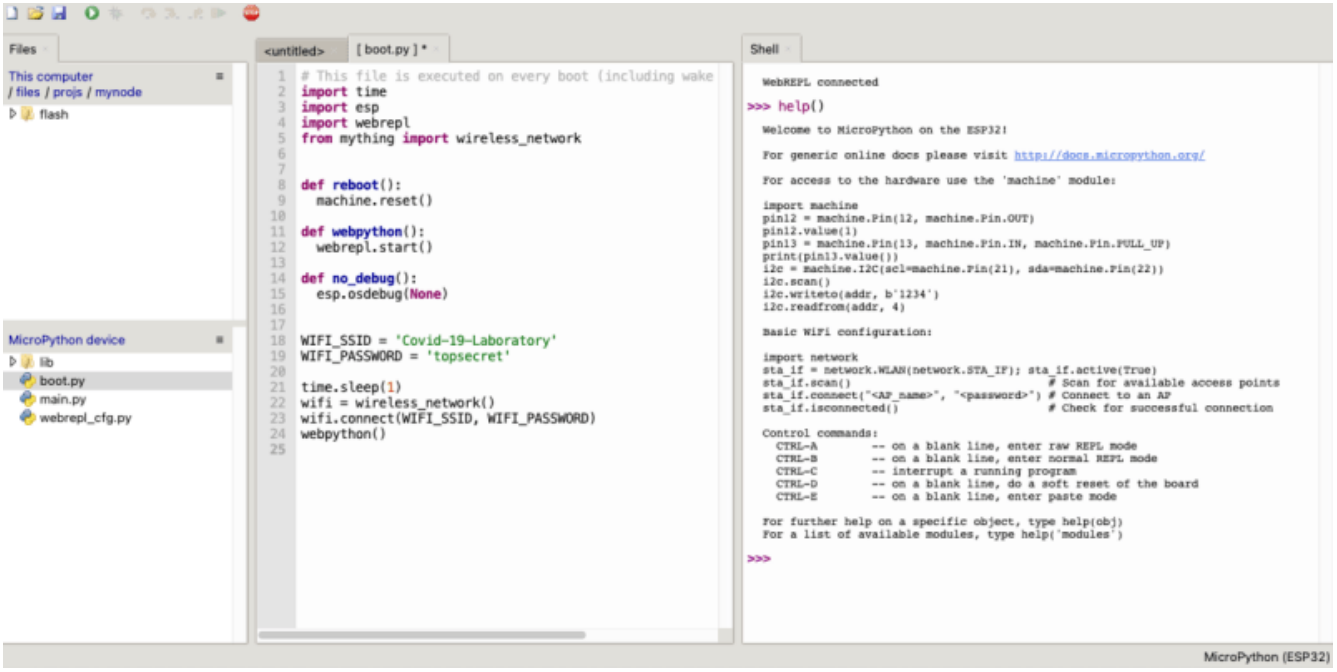


Рисунок 3.4 – Середовище розробки Thonny

Порівняння функціональних характеристик середовищ наведено в таблиці 3.2.

Таблиця 3.2 – Таблиця порівняння функціональних характеристик середовищ програмування

Характеристика	VS Code	Jupyter Notebook	Thonny	PyCharm
Автодоповнення коду	1	0	0	1
Інтеграція з Git	1	1	0	1
Підтримка віртуальних середовищ (venv)	1	0	1	1
Покрокове виконання коду (дебаггер)	1	0	1	1
Вбудоване тестування	1	0	1	1
Підтримка розширень та плагінів	1	1	0	1
Вбудований менеджер залежностей	0	0	1	1
Загальний коефіцієнт	6	2	4	7

Проаналізувавши таблицю, PyCharm має найбільше пунктів в функціональних характеристиках серед інших середовищ для розробки. Отже, для розробки на мові програмування Python було обрано середовище програмування PyCharm Community Edition.

3.2 Розробка модуля ігрової логіки

Модуль ігрової логіки відповідає за обробку дій гравців під час гри, базуючись на повідомленнях, які надходять від клієнтів через websocket-з’єднання. Робота модуля починається з моменту отримання JSON-повідомлення від клієнта. Алгоритм, представлений на рисунку 3.5, описує повну послідовність обробки вхідних даних, перевірки умов та змін ігрового стану.

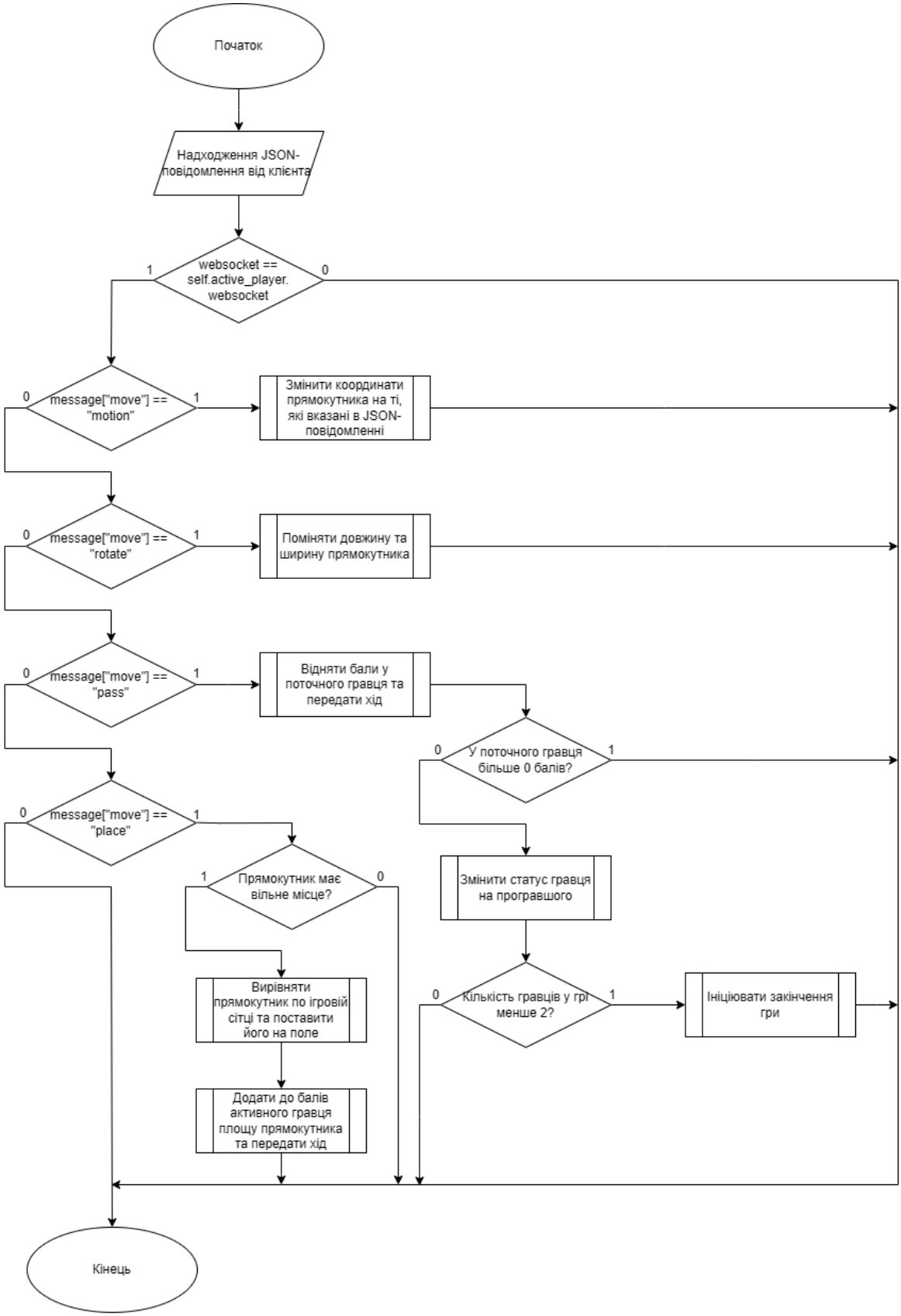


Рисунок 3.5 – Блок-схема алгоритму модуля ігрової логіки

Після надходження JSON-повідомлення модуль перевіряє, чи надійшло воно від того гравця, чий зараз хід. Якщо так, то обробка продовжується, якщо ж повідомлення надійшло від іншого гравця, воно ігнорується. Таким чином, завдяки тому що основна ігрова логіка обробляється на сервері, у грі не можна махлювати.

Далі відбувається перевірка типу дії, переданої у повідомленні. У полі move JSON-об’єкта можуть бути вказані такі команди:

– "motion" – гравець намагається перемістити прямокутник. У цьому випадку система змінює координати прямокутника відповідно до даних, переданих у повідомленні;

– "rotate" – гравець повертає прямокутник на 90 градусів. Тоді змінюються його довжина і ширина, тобто відбувається обмін місцями значень ширини та висоти, таким чином, відбувається поворот блоку;

– "pass" – гравець пропускає хід. У такому випадку система віднімає у гравця певну кількість балів та передає хід наступному гравцю. Якщо в результаті цієї дії у гравця залишилося менше 0 балів, він вважається таким, що програв, і його статус змінюється відповідно. Після цього система перевіряє, чи ще залишилось у грі більше одного гравця, і якщо кількість активних гравців є меншою за два, тобто залишився лише один гравець з позитивною кількістю балів, то ініціюється завершення гри;

– "place" – гравець намагається розмістити прямокутник на ігровому полі. Проводиться перевірка, чи є в прямокутника вільне місце для розміщення. Якщо вільне місце є, прямокутник вирівнюється по ігровій сітці та встановлюється на полі. Після цього до рахунку активного гравця додається площа прямокутника як кількість балів і хід передається наступному гравцю. Якщо вільного місця немає, то команда ігнорується.

Таким чином, алгоритм забезпечує обробку всіх ключових дій гравця та контроль ігрового процесу в реальному часі. Це дозволяє справедливо керувати

ходами учасників, оновлювати стан гри після кожної дії та завершувати гру при досягненні кінцевих умов.

3.3 Розробка модуля розподілу гравців у кімнати

Розробка модуля розподілу гравців у кімнати становить одну з найважливіших передумов для побудови стабільної й масштабованої багатокористувацької гри, що ґрунтується на сесійній мультиплеєрній архітектурі. Насамперед саме цей модуль забезпечує ізоляцію ігрових сесій, тому що кожна кімната функціонує як самостійна одиниця із власним набором станів і подій, тож дії учасників усередині однієї кімнати жодним чином не впливають на інші паралельні сесії. Ізольована логіка в межах кожної кімнати дозволяє зменшити обсяг оперативної пам’яті та мережевих повідомлень, які потрібно обробити. Серверу не потрібно передавати оновлення всім гравцям гри – лише тим, що перебувають у тій самій кімнаті. Це мінімізує обмін даними та покращує загальну продуктивність. Така ізольованість дає змогу обслуговувати значну кількість одночасних ігор, не створюючи конфліктів у спільному ігровому просторі й не погіршуючи досвід користувачів.

Крім продуктивності, модуль розподілу надає значну гнучкість дизайну геймплею. Адміністратор або самі гравці можуть створювати кімнати з різними правилами, режимами, кількістю учасників або типом карт. Це відкриває шлях до впровадження як класичних дуелей, так і складних командних баталій чи кооперативних сценаріїв, між якими гравці можуть перемикатися. Модуль розподілу гравців у кімнати виступає фундаментом, на якому тримається вся сесійна мультиплеєрна архітектура. Модуль реалізовано у вигляді алгоритму, що обробляє вхідні запити користувача та здійснює розподіл відповідно до заданих умов.

На рисунку 3.6 наведено блок-схему алгоритму роботи модуля розподілу гравців у кімнати.

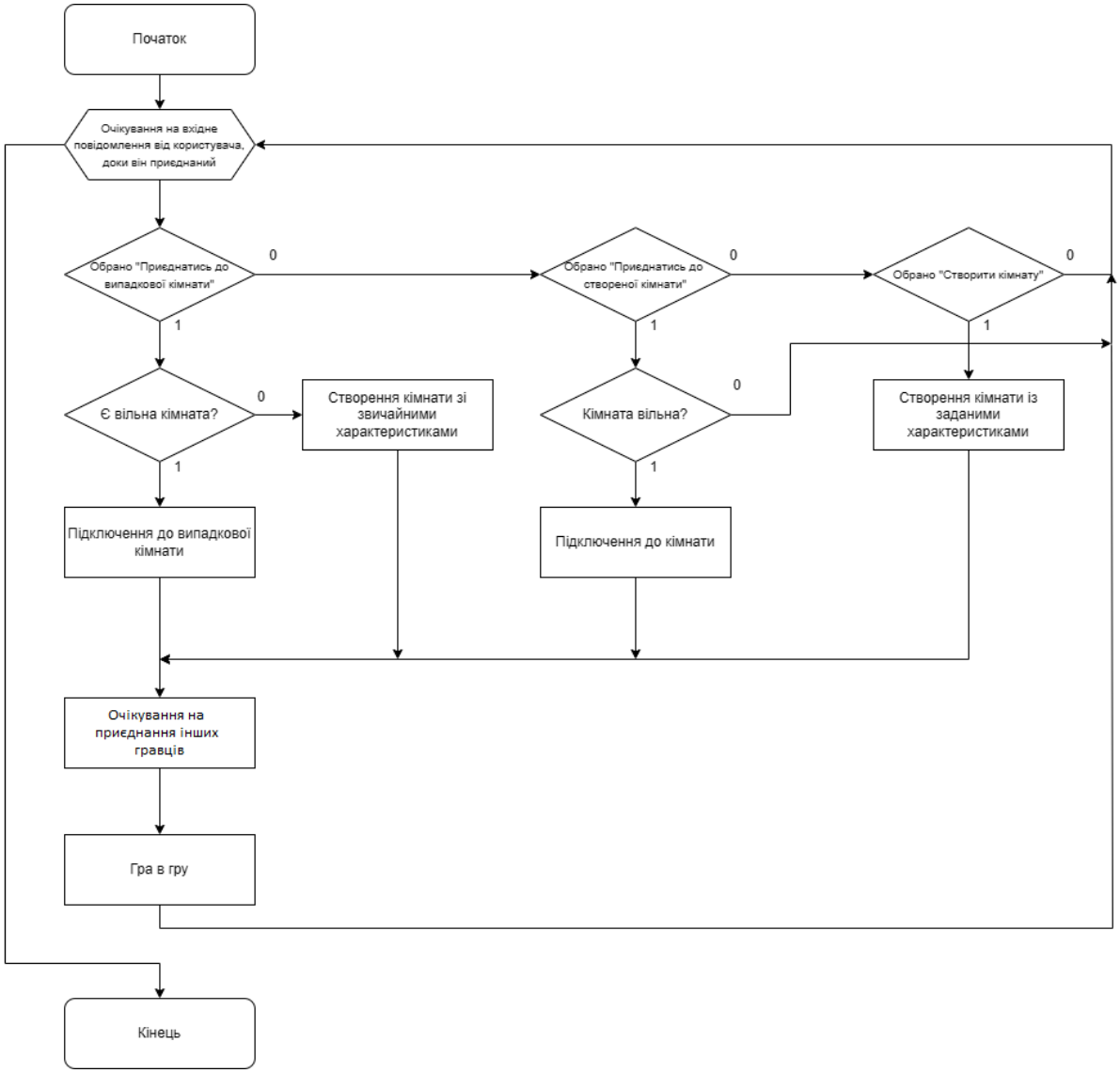


Рисунок 3.6 – Блок-схема алгоритму модуля розподілу гравців у кімнати

Алгоритм розпочинається з очікування запиту від користувача щодо приєднання до гри. Далі він обробляє обраний варіант дії:

1) Приєднання до випадкової кімнати. У разі вибору цього варіанту перевіряється наявність вільної кімнати із звичайними параметрами (тобто, створеної без спеціальних налаштувань). Якщо така кімната є – користувач до неї

підключається. Якщо ні – створюється нова кімната з типовими характеристиками, і користувача до неї додають.

2) Приєднання до вже створеної кімнати. У цьому випадку здійснюється перевірка, чи вказана кімната доступна. Якщо вона вільна (тобто не перевищено максимальну кількість гравців, і кімната активна) – користувач приєднується. Якщо кімната недоступна, система повертає користувача на початковий етап вибору.

3) Створення нової кімнати. Якщо користувач обирає цей варіант, він може вказати характеристики майбутньої кімнати (наприклад, кількість гравців, час очікування, додаткові правила). Після створення кімнати користувач автоматично приєднується до неї.

Після приєднання до кімнати система переходить у стан очікування інших гравців. Як тільки всі необхідні гравці приєднаються — запускається ігрова сесія.

Такий підхід дозволяє реалізувати як швидке автоматичне приєднання до випадкових ігор, так і створення приватних сесій із заданими параметрами, що забезпечує гнучкість ігрового процесу. Модуль також дозволяє масштабувати гру без додаткової ускладненої логіки з боку клієнта.

3.3 Розробка модуля сервера

Окрім реалізації базової ігрової логіки, важливим завданням серверного модуля є координація обробки подій та синхронізація всіх гравців щодо змін у грі. Для цього розроблено окремий цикл обробки подій, що забезпечує цілісність ігрового процесу, синхронізацію стану та роботу з мережею. Відповідний алгоритм подано на рисунку 3.7.

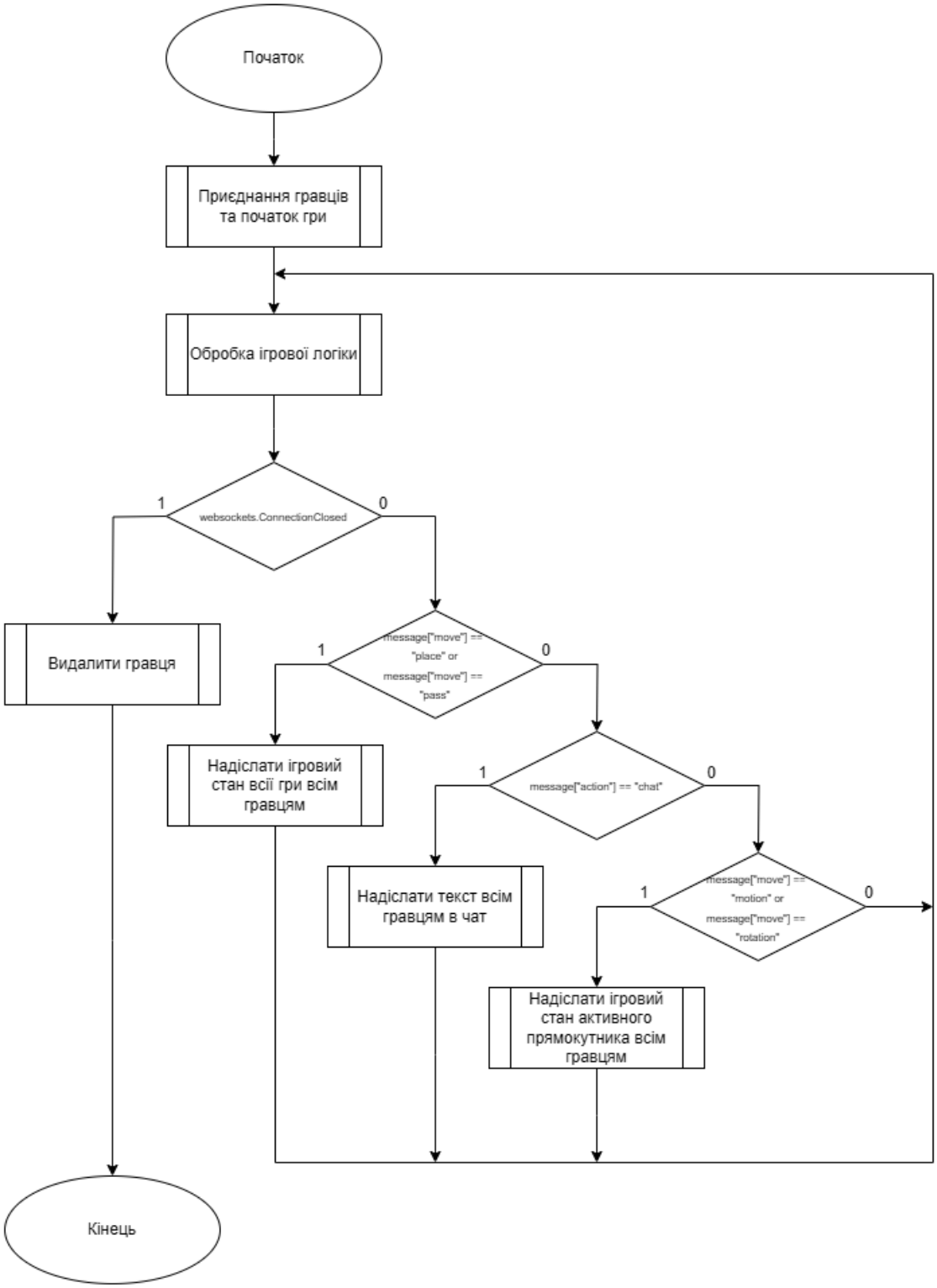


Рисунок 3.7 – Блок-схема алгоритму роботи модуля сервера

На початку відбувається приєднання гравців та запуск ігрової сесії. Цей етап включає ініціалізацію websocket-з'єднань, перевірку мінімальної кількості учасників для старту, створення стартового стану гри та відправку початкових даних клієнтам, як описано в модулі розподілу гравців в кімнати.

Після цього запускається основний цикл, який безперервно оброблює ігрову логіку. У цьому циклі обробляються події від користувачів (через отримані повідомлення) та здійснюється відповідна реакція сервера, як описано в модулі ігрової логіки.

Першим кроком у циклі модуля серверу є перевірка, чи не було розірвано з’єднання з клієнтом. Якщо websocket-з'єднання з якимось із гравців було втрачено, тобто згенеровано виняток websockets.ConnectionClosed, цей гравець видаляється з активної гри (рис. 3.8).

```
except websockets.ConnectionClosed:
    print("Клієнт відключився!")
    for room in rooms:
        room.deleteByWebsocket(websocket)
```

Рисунок 3.8 – Виклик функції deleteByWebsocket сервером

За допомогою функції deleteByWeboscket, яка належить об'єкту Room, функція виконує локальний пошук гравця в межах конкретної кімнати (рис. 3.9). Вона перебирає список self.players, який містить об’єкти гравців, і перевіряє, чи відповідає websocket того гравця вебсокету, який був переданий у функцію. Якщо збіг знайдено, відповідного гравця видаляють зі списку учасників кімнати методом self.players.remove(player).

```
def deleteByWebsocket(self, websocket):
    for player in self.players:
        if player.websocket == websocket:
            self.players.remove(player)
```

Рисунок 3.9 – Функція видалення гравця

Цей механізм виконує одразу кілька важливих задач:

– звільнення ресурсів, адже після відключення гравець більше не бере участі в грі і його потрібно прибрати з кімнати;

– запобігання зависанню логіки гри, оскільки якщо не видалити гравця, сервер може продовжити чекати його дій;

– гнучкість, оскільки через те, що гравець може бути в будь-якій кімнаті, пошук через всі кімнати гарантує що з'єднання буде оброблено коректно;

– надійність роботи, бо винятки, пов'язані з втратою з'єднання, є очікуваними в реальному середовищі, тож обробка ConnectionClosed дозволяє системі залишатися стабільною й не завершувати роботу через помилку.

У випадку, якщо з'єднання залишається активним, відбувається обробка ігрового повідомлення. Алгоритм підтримує декілька варіантів:

Якщо отримане повідомлення містить message["move"] == "place" або message["move"] == "pass", це означає, що активний гравець завершив свій хід. У такому випадку сервер обов'язково надсилає всім клієнтам повний оновлений стан всієї гри, який містить актуальні зміни поля (рис. 3.10). Це важливо для того, щоб постійно не оновлювати стан гри, який не змінюється, а лише оновлювати його раз в хід.

Якщо ж у повідомленні є message["action"] == "chat", отже, гравець написав повідомлення в чат. У відповідь сервер оновлює ігровий чат та розсилає іншим

учасникам подію про зміну чату та саме текстове повідомлення, підтримуючи комунікацію між учасниками гри.

```
def gameStatusOnceAMove(self):
    blocks_info = []
    for block in self.grid.blocks:
        block_data = {
            "x": block.x,
            "y": block.y,
            "width": block.width,
            "height": block.height,
            "color": block.color,
        }
        blocks_info.append(block_data)

    game_state = {
        "blocks": blocks_info
    }

    return game_state
```

Рисунок 3.10 – Функція оновлення стану гри

Для команд, що стосуються рухомих дій гравця з фігурою, тобто "motion" або "rotation", сервер оновлює лише частковий стан – а саме положення та розміри прямокутника активного гравця (рис. 3.11). Така інформація також надсилається всім учасникам гри, але без повної синхронізації всього ігрового поля, що дозволяє зекономити ресурси.

Функція gameStatus виконує ключову роль у забезпеченні синхронізації стану гри між усіма учасниками певної ігрової сесії. Її основне призначення – зібрати та повернути актуальну інформацію про активний ігровий блок, яким у даний момент керує один із гравців.

Ця функція створює словник info, що містить координати активного блока (x та y), його розміри (height і width), а також колір (color). Ці параметри описують візуальні та просторові характеристики блока на ігровому полі. Саме ці дані потрібні іншим гравцям для того, щоб бачити у своїх клієнтах точне положення та стан об'єкта, яким оперує поточний гравець.

```
def gameStatus(self):
    #  Це знадобиться для передачі стану блока всім користувачам цієї гри
    info = {
            "active_block_x" : self.active_block.x,
            "active_block_y" : self.active_block.y,
            "active_block_height" : self.active_block.height,
            "active_block_width" : self.active_block.width,
            "active_block_color": self.active_block.color
            }
    return info
```

Рисунок 3.11 – Функція передачі даних про активний прямокутник

Оскільки багатокористувацька взаємодія відбувається через обмін JSON-повідомленнями по WebSocket, інформація, сформована у gameStatus, конвертується у формат JSON і розсилається всім підключеним до кімнати клієнтам. Таким чином, ця функція виступає в ролі механізму оновлення інтерфейсів користувачів у реальному часі, щоб забезпечити однакове бачення стану гри для кожного гравця.

Кожен з описаних шляхів у алгоритмі закінчується поверненням у цикл обробки логіки, забезпечуючи постійне реагування на події. Таким чином, серверна частина гри побудована як подійно-орієнтований цикл завдяки вебсокетам, що враховує як логіку гри, так і підтримку стану з’єднання та інтеграцію чату.

3.4 Розробка модуля чату

Блок-схема на рисунку 3.12 описує логіку роботи модуля чату у багатокористувацькій грі. Основна мета модуля – забезпечення обміну текстовими повідомленнями між гравцями та автоматичний фільтр нецензурної лексики.

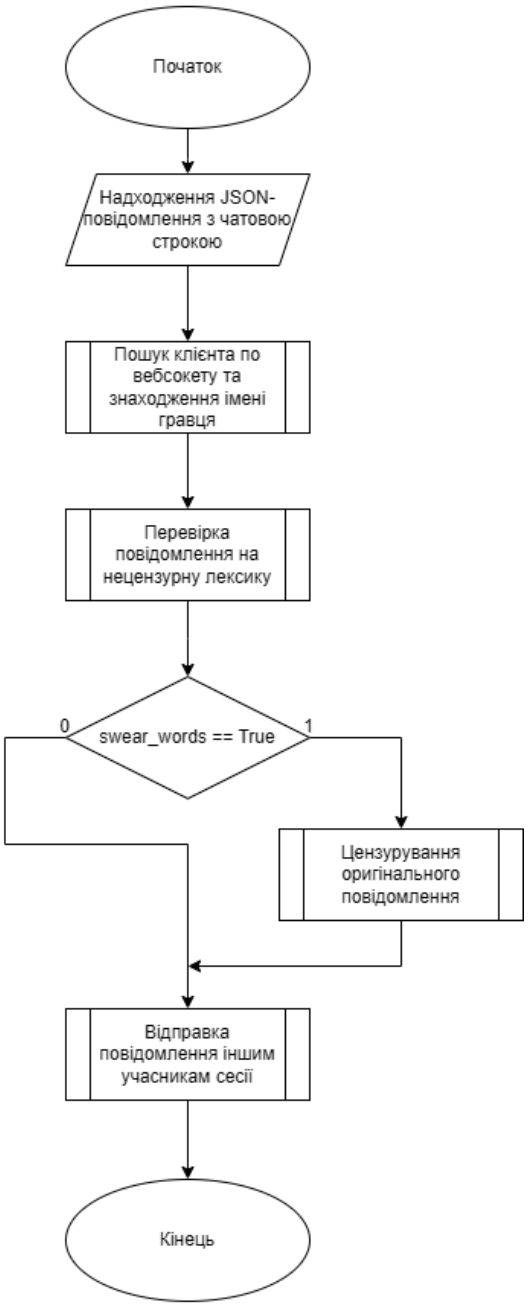


Рисунок 3.12 – Блок-схема алгоритму роботи модуля чату

Процес починається з того, що сервер отримує JSON-повідомлення, яке містить чатову строку – текст, який один з гравців хоче надіслати іншим учасникам гри.

Далі система виконує ідентифікацію гравця, який надіслав повідомлення. Це здійснюється через пошук відповідного вебсокет-з'єднання та отримання імені гравця. Цей крок важливий для того, щоб знати, хто є автором повідомлення, а також щоб правильно сформувати повідомлення для інших гравців.

Наступним етапом є перевірка вмісту повідомлення на наявність нецензурних слів або образливих виразів. За допомогою бібліотеки profanity-check результат перевірки фіксується у змінній swear_words. Якщо swear_words == True (тобто повідомлення містить лайку), тоді відбувається цензурування – заміна образливих слів на зірочки. Якщо swear_words == False, повідомлення залишається без змін.

Після обробки повідомлення воно розсилається іншим учасникам ігрової сесії. Це дає змогу підтримувати безперервний чат між гравцями в рамках однієї кімнати. На цьому цикл для одного повідомлення завершується, і сервер готовий обробляти наступні повідомлення.

Цей модуль відіграє важливу роль у підтриманні етичної поведінки в грі та забезпечує базову модерацію чату. Він також відповідає за підтримку реального часу взаємодії між гравцями, що є критично важливим для соціального та комунікаційного аспекту багатокористувацької гри. Такий підхід сприяє створенню комфортного середовища для всіх учасників гри.

3.5 Розробка модуля фіксації координат по сітці

У грі, де користувачі взаємодіють із прямокутниками, важливим аспектом є зручність позиціонування об'єктів на площині. Для цього було реалізовано окремий модуль, що відповідає за фіксацію (магнітне прилипання) прямокутників до ліній поля.

Програмний код, який виконує округлення координат до значення розміру клітинки сітки, показано на рисунку 3.13.

```
e=0
while e<x:
    e=e+step
    if (e-x)<(step/2):
        xPutRect=e
    else:
        xPutRuct=(e-step)

e=0
while e<y:
    e=e+step
    if (e-y)<(step/2):
        yPutRect=e
    else:
        yPutRuct=(e-step)
```

Рисунок 3.13 – Код для округлення координат до найближчих ліній сітки поля

Такий підхід є поширеним у багатьох графічних редакторах, конструкторах рівнів та візуальних редакторах інтерфейсів, оскільки значно полегшує точне розміщення елементів та зменшує кількість неточностей при ручному розташуванні.

3.6 Розробка інтерфейсу веб сайту

Розробка вебсайту – це ключовий етап створення багатокористувацької системи, адже саме через веб інтерфейс користувач взаємодіє з програмною логікою. Інтерфейс виступає посередником між внутрішніми механізмами гри чи системи та кінцевим користувачем. У контексті розробки веб інтерфейсу важливо

не лише забезпечити функціональність, а й створити зрозумілий, інтуїтивний і привабливий для користувача інтерфейс.

На етапі проєктування веб інтерфейсу було прийнято рішення створити односторінковий застосунок (SPA), який забезпечує динамічне оновлення вмісту без перезавантаження сторінки [19]. Такий підхід дозволяє користувачам отримувати миттєвий зворотній зв’язок, що особливо важливо у реальному часі під час гри.

Інтерфейс реалізовано за допомогою HTML, CSS та JavaScript. Веб інтерфейс підключається до серверної частини за допомогою WebSocket API, через яке передаються всі події, такі як рухи об'єктів, хід гри, повідомлення чату, оновлення статусу гри.

На рисунку 3.14 показано макет головної сторінки веб сайту.

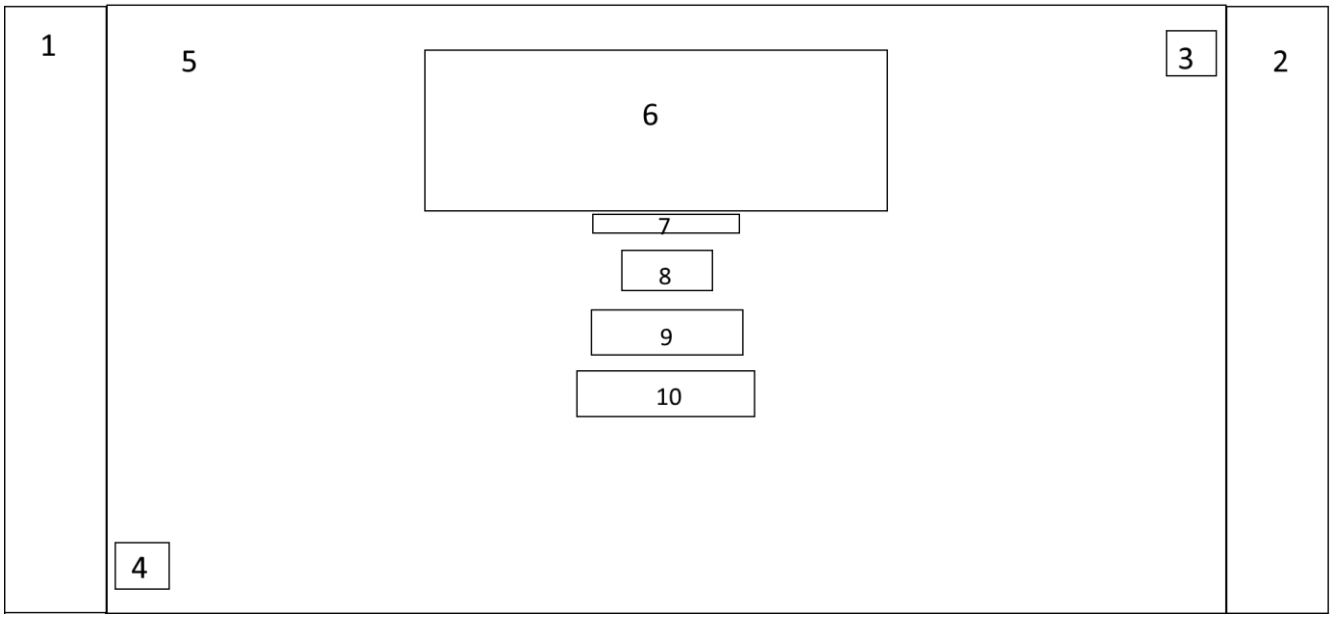


Рисунок 3.14 – Макет головної сторінки веб сайту

На головній сторінці веб інтерфейсу кожен елемент має своє функціональне та візуальне призначення, і загальна композиція дозволяє користувачу швидко зорієнтуватися та взаємодіяти з системою інтуїтивно. Елементи сторінки:

1. Рекламний банер, який розташовується у лівій частині вебсторінки. Його призначення – відображення рекламного контенту. Банер не заважає основному ігровому процесу та інтерактивним елементам, проте візуально виділяється, привертаючи увагу користувача.

2. Рекламний банер, який розташовується у правій частині вебсторінки.

3. Кнопка «Налаштування», яка відкриває панель з параметрами користувача. Дозволяє персоналізувати колір прямокутника у грі, адаптуючи його під уподобання гравця. Розташовується у правому верхньому куті сторінки.

4. Кнопка «Правила», яка відповідає за доступ до короткого опису ігрової механіки, основних правил та умов гри. Після натискання відкривається вікно з текстовим поясненням. Цей елемент особливо важливий для нових користувачів, які ще не знайомі з принципами гри.

5. Фон сторінки (бекграунд), який виконує декоративну роль і задає стилістику та атмосферу гри. У розробленому веб інтерфейсі це статичне зображення. Головна вимога до нього – не відволікати увагу від інтерактивних елементів та забезпечувати загальну естетичну узгодженість інтерфейсу.

6. Логотип гри, який є центральним елементом ідентичності гри на сторінці. Він допомагає користувачу швидко зрозуміти, в яку гру він потрапив, і виконує як естетичну, так і брендову функцію.

7. Форма введення нікнейму – поле, в яке користувач вводить свій ігровий нік. Ім’я, введене тут, буде використовуватись у чаті, під час гри, а також для відображення вкінці гри.

8. Кнопка «Грати», яка активує наступний етап – запуск гри, де користувач доєднується до випадкових гравців. Це головна кнопка запуску взаємодії з ігровим середовищем.

9. Кнопка «Підключитись». Після натискання на неї користувач може ввести код сесії та швидко долучитися до вже існуючої гри, організованої іншими

гравцями. Приєднання до кімнати відбувається через модуль розподілу гравців у кімнати, логіка якого обробляється на сервері.

10. Кнопка «Створити кімнату», яка дозволяє гравцю розпочати нову гру, ставши хостом сесії. Після натискання користувач вказує унікальний ідентифікатор кімнати, до якої зможуть приєднатися інші користувачі. Також користувач може обрати параметри, з якими він буде створювати сесію.

На рисунку 3.15 показана головна сторінка веб сайту.

Рисунок 3.15 – Головна сторінка веб-сайту

Ці елементи у поєднанні створюють зручний, зрозумілий та сучасний інтерфейс, який швидко орієнтує гравця на ключові дії та не перевантажує зайвими деталями.

3.7 Висновки

У третьому розділі було проаналізовано мови програмування та середовща програмування та обрано Python та PyCharm відповідно. Описано процес розробки модулів багатокористувацької гри на основі сесійної мультиплеєрної архітектури, а саме модулів сервера, ігрової логіки, чату, округлення координат та модуля

розподілу гравців у кімнати. Окрему увагу приділено створенню модуля ігрової логіки, зокрема функціям передачі стану гри, обробки дій гравців. Описано механізм обробки відключень гравців та видалення їх з відповідних кімнат. Описано створення веб інтерфейсу гри, з поясненням кожного з його елементів, що забезпечують інтуїтивну взаємодію користувача з системою.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування багатокористувацьких вебсистем є важливим етапом у процесі їх розробки, оскільки такі системи працюють у режимі реального часу, забезпечують взаємодію багатьох користувачів одночасно і повинні гарантувати стабільність, продуктивність та безпечність на кожному етапі використання. Особливості архітектури, зокрема модуль розподілу гравців по кімнатах, взаємодія через WebSocket-з’єднання та реалізація спільної ігрової логіки, створюють підвищені вимоги до системи тестування. Виявлення та усунення помилок на ранніх етапах допомагає уникнути збоїв у грі, втрати даних, несанкціонованого доступу та інших критичних проблем.

Методи тестування для такого типу систем поділяються на кілька рівнів. Найбільш базовим є модульне тестування, у межах якого перевіряються окремі компоненти системи, такі як функції обробки дій гравця, функції оновлення стану гри або обробка відключень. Це дозволяє виявити локальні помилки у логіці до того, як модулі починають взаємодіяти між собою. Далі здійснюється інтеграційне тестування, яке дозволяє перевірити взаємодію між модулями – наприклад, як передається інформація між клієнтом та ігровим середовищем. Важливою частиною цього є перевірка обробки повідомлень у чаті, зміни стану гри в реальному часі.

Системне тестування є наступним рівнем, де перевіряється робота усієї системи в цілому. У цьому випадку враховується не лише логіка гри, а й обробка підключень, створення сесій та розподіл гравців по кімнатах. У рамках тестування проводиться перевірка з точки зору кінцевого користувача – наскільки зручно використовувати інтерфейс, чи працюють кнопки підключення до сесії, створення імені гравця, чи немає затримок у оновленні гри або повідомлень чату.

З огляду на специфіку системи, ефективним виявляється використання динамічних методів тестування, зокрема методу "чорної скриньки", що дозволяє перевірити функціональність без заглиблення у внутрішню реалізацію, наприклад, при тестуванні веб інтерфейсу або взаємодії гравців у сесії. Також у деяких випадках доцільно застосовувати метод "сірої скриньки", коли відомі загальні принципи роботи модулів, наприклад, модуль логіки розподілу гравців, що дозволяє точніше виявляти помилки у взаємодії. Для внутрішніх частин логіки, таких як обробка websocket-з’єднань, актуальним є також метод "білої скриньки", який дозволяє тестувати конкретні гілки виконання програми.

У рамках розробки також враховувались статичні методи тестування, такі як аналіз коду вручну та автоматична перевірка структури на наявність синтаксичних або логічних помилок. Такі методи допомагають виявляти потенційні проблеми на ранньому етапі, ще до запуску системи.

Під час тестування особливу увагу приділено викликам, характерним для багатокористувацьких систем. Зокрема, необхідно враховувати можливість одночасних дій багатьох користувачів, непередбачені розриви зв’язку, перевантаження сервера та затримки в передачі повідомлень. Для зменшення впливу цих проблем можуть використовуватись модульні перевірки websocket-підключень, логування дій гравців та сценарії симульованих навантажень.

Таким чином, проведений аналіз методів тестування дозволяє сформувати комплексну систему контролю якості, яка охоплює як окремі компоненти гри, так і систему в цілому. Завдяки поєднанню різних підходів можна забезпечити стабільність і надійність багатокористувацької гри, що відповідає вимогам для розробки методів та засобів побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури.

4.2 Тестування системи

На головній сторінці перевірено відображення графічних елементів, зокрема кольорового фону, логотипу гри, а також кнопок і текстових полів. Кнопка «Підключитись» викликає поле для введення назви кімнати та додаткову кнопка підтвердження (рис. 4.1). Також перевірено роботу форми введення нікнейму.

Рисунок 4.1 – Головна сторінка після натиснення кнопки «Підключитись»

Поле введення нікнейму приймає рядок будь-якої довжини, однак при підтвердженні порожнього значення гравецю буде видаватись автоматично згенерований нікнейм, такий як Player 1 або Player 2.

Кнопка "Підтвердити" успішно передає введені значення на сервер і перевіряє наявність відповідної кімнати. При успішному введенні користувач приєднується до неї.

При неправильному імені кімнати з’являється повідомлення про її відсутність, що дозволяє користувачу повторити спробу (рис. 4.2).

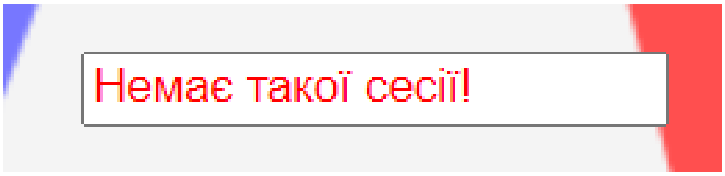


Рисунок 4.2 – Повідомлення про відсутність сесії з вказаною назвою

Кнопка «Створити кімнату» відкриває меню з опціями налаштування параметрів кімнати (рис. 4.3). Перевірено працездатність форм.

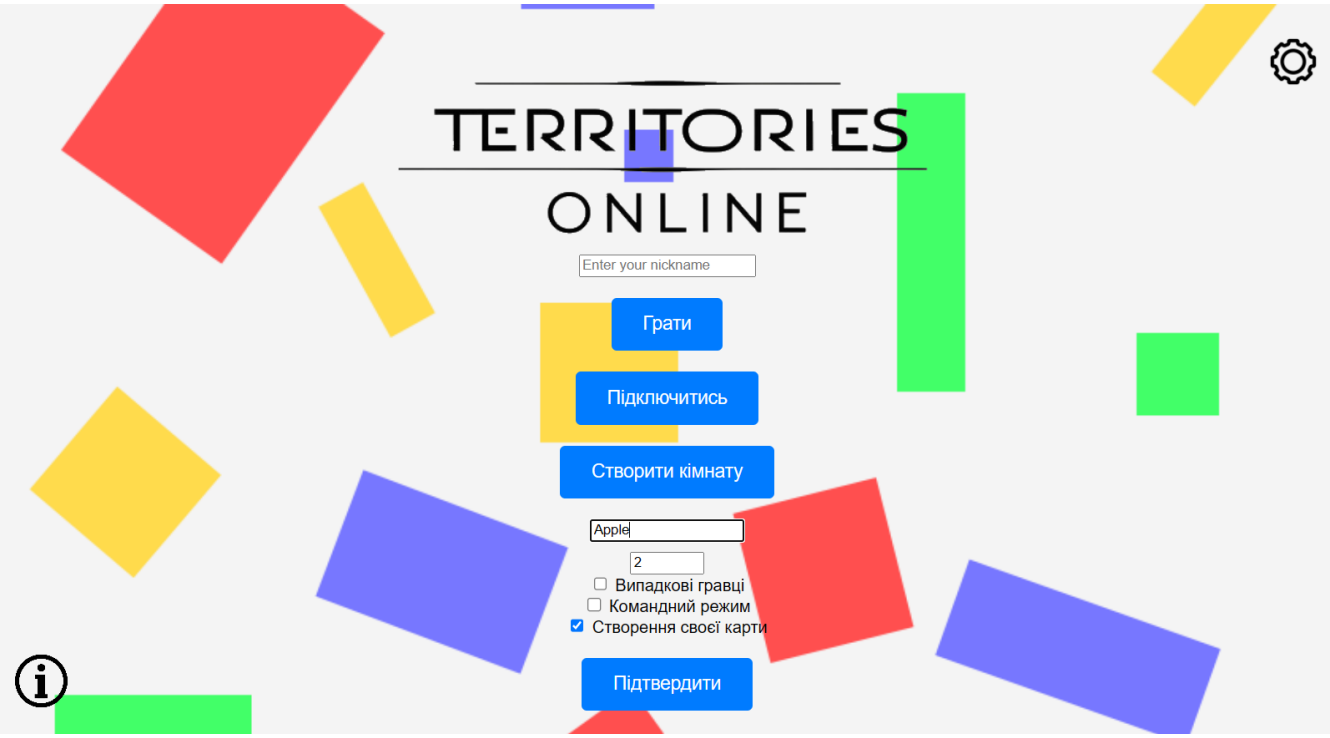


Рисунок 4.3 – Головна сторінка після натиснення кнопки «Створити кімнату»

Кнопки «Налаштування» і «Правила» перевірені на коректність відкриття відповідних розділів (рис. 4.4). Кнопка «Правила» відкриває правила гри, де новий користувач, який ще не знає правил гри, може ознайомитись з геймплеєм Territories Online.

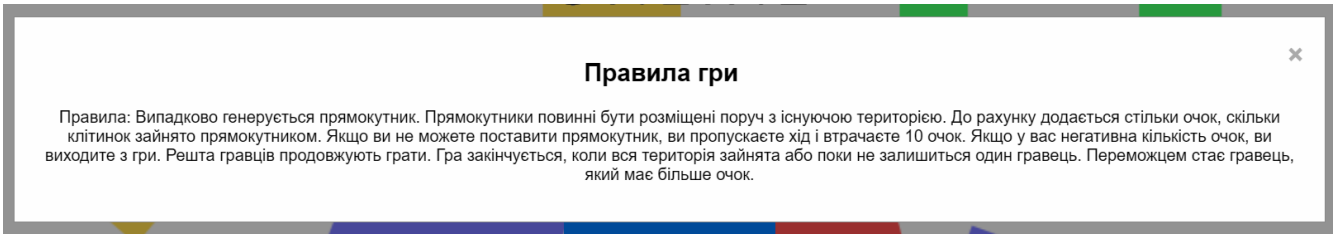


Рисунок 4.4 – Результат натиснення кнопки «Правила»

Після натискання кнопки «Налаштування», що знаходиться у правому верхньому куті інтерфейсу головної сторінки у вигляді піктограми, відкривається меню вибору кольору. Це меню дозволяє користувачу обрати базовий колір, яким будуть позначатися прямокутники, які він ставить під час гри.

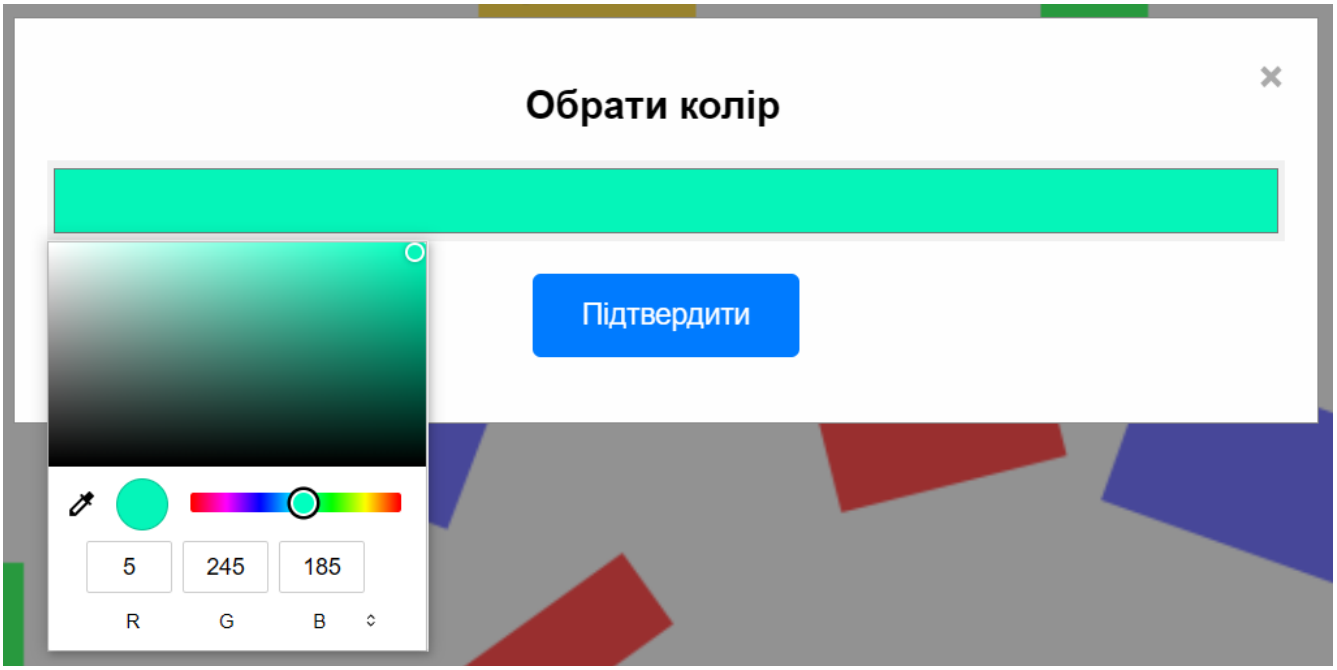


Рисунок 4.5 – Результат натиснення кнопки «Налаштування»

Кнопка «Грати» успішно ініціює старт сесії гри у стандартному режимі. Під час цього використовуються стандартні правила, описані у правилах гри, тобто 4 гравці одне проти одного (рис. 4.6).

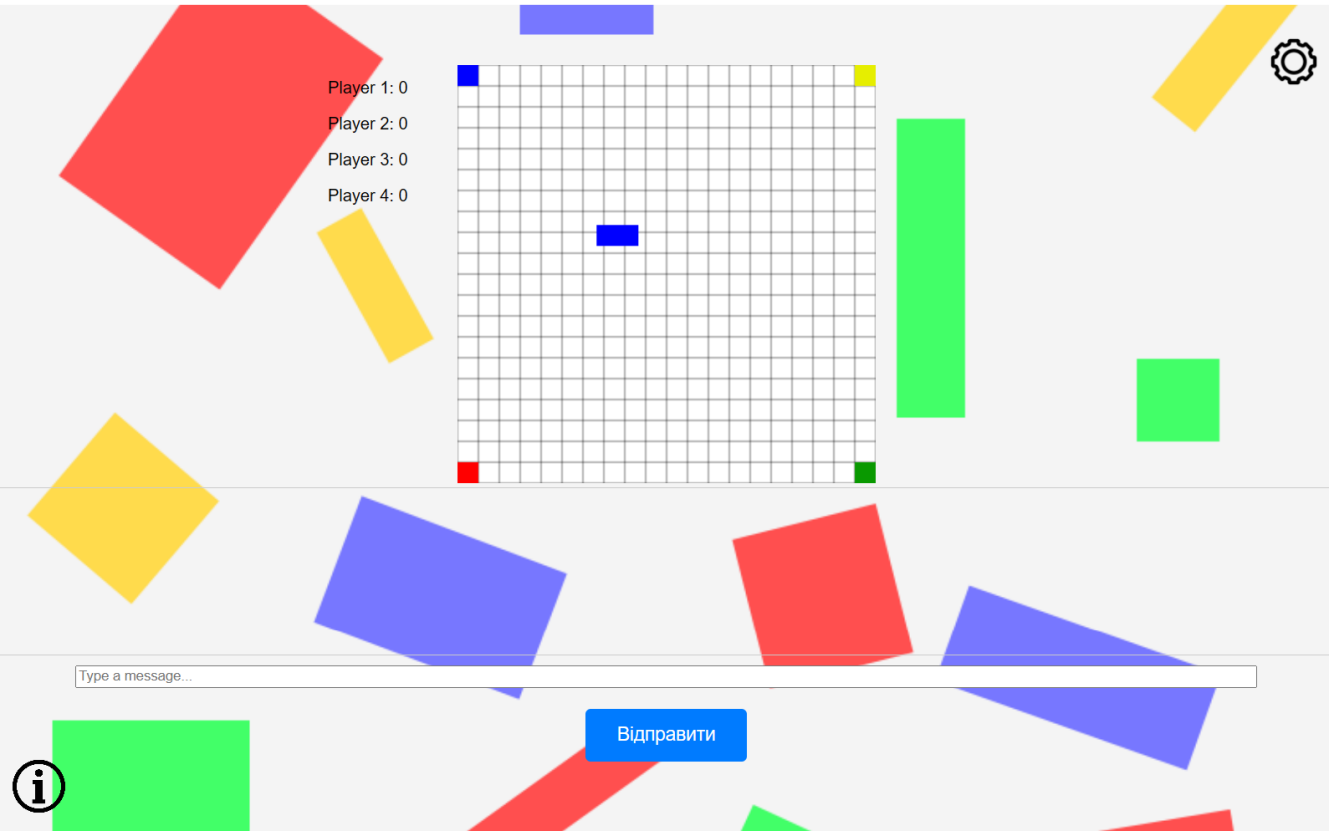


Рисунок 4.6 – Результат натиснення кнопки «Грати» та підключення інших користувачів

Під час гри перевірено коректність ігрової логіки, усі прямокутники розміщуються згідно з правилами гри – перевіряється, щоб вони не перетиналися з уже розміщеними об'єктами, інакше хід вважається недійсним. Також було протестовано механізм нарахування балів: кожному гравцеві правильно зараховується кількість очок відповідно до площі встановленого прямокутника. Прямокутники, згідно з правилами, можна повертати, можна пропускати свій хід, через що у гравця віднімаються бали. Завдяки цим перевіркам забезпечено коректну роботу гри та дотримання правил у кожній сесії. Всі прямокутники ставляться правильно, як показано на рисунку 4.7.

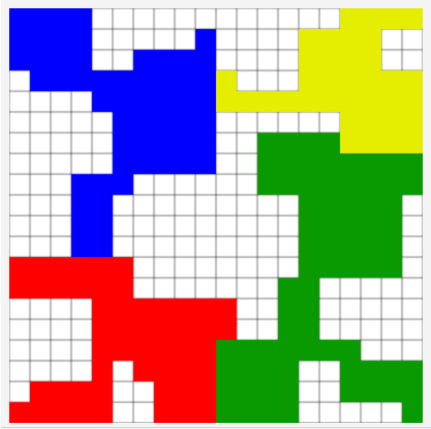


Рисунок 4.7 – Тестування ігрової логіки

Також для перевірки стійкості та продуктивності серверної частини гри було проведено симуляційне тестування, яке імітує масове підключення клієнтів, надсилання ігрових дій та взаємодію між гравцями (рис. 4.8). У рамках тестування було реалізовано сценарій, в якому до сервера одночасно підключалися до 100 віртуальних клієнтів. Результати показали, що серверна частина системи стабільно обробляє запити та підтримує одночасне обслуговування великої кількості користувачів. Середня затримка обробки дій клієнтів склала від 20 до 140 мілісекунд, що є прийнятним показником для онлайн гри.

```
[Client_38] -> action: move, server latency: 49.0 ms
[Client_39] -> action: pass, server latency: 74.9 ms
[Client_40] -> action: move, server latency: 135.4 ms
[Client_41] -> action: rotate, server latency: 40.3 ms
[Client_42] -> action: move, server latency: 122.4 ms
[Client_43] -> action: rotate, server latency: 104.2 ms
[Client_44] -> action: rotate, server latency: 139.7 ms
[Client_45] -> action: pass, server latency: 34.8 ms
[Client_46] -> action: move, server latency: 130.7 ms
[Client_47] -> action: move, server latency: 41.6 ms
[Client_48] -> action: place, server latency: 67.5 ms
[Client_49] -> action: place, server latency: 147.5 ms
```

Рисунок 4.8 – Тестування навантаження

Результати тестування підтверджують готовність серверної частини до використання в реальних умовах.

4.3 Висновки

У четвертому розділі розглянуто результати тестування вебінтерфейсу багатокористувацької гри Territories Online, перевірено коректність роботи головної сторінки та ігрової логіки, виконано тестування навантаження.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено методи та засоби побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури.

Для розробки було використано середовище програмної розробки PyCharm та мову програмування Python, для розробки сторони клієнта було використано Javascript, HTML та CSS.

Бакалаврська кваліфікаційна робота оформлена згідно з методичними вказівками [20].

Під час виконання бакалаврської кваліфікаційної роботи проведено детальний аналіз особливостей сучасних багатокористувацьких ігор, розглянуто приклади реалізації подібних систем та виявлено ключові технічні виклики. Сформульовано мету та задачі дослідження, зокрема потребу у створенні системи з незалежними ігровими мультиплеєрними сесіями.

Було розроблено схему загального алгоритму роботи системи. Також було розроблено модель системи з використанням UML діаграми класів.

Відповідно до поставлених задач було:

– розроблено модель багатокористувацької сесійної гри, яка включає логіку ігрового процесу, взаємодії гравців та розподіл по кімнатах;

– розроблено метод випадкової генерації прямокутників з адаптивним алгоритмом розподілу;

– реалізовано модуль сервера, який керує підключеннями користувачів, обробкою подій у реальному часі та синхронізацією стану гри;

– розроблено модуль розподілу гравців, що дозволяє динамічно створювати сесії;

– створено модуль чату, що забезпечує текстову комунікацію між учасниками однієї сесії;

– розроблено інтерфейс вебсайту, який забезпечує доступ до гри через зручний та адаптивний клієнтський інтерфейс;

– здійснено тестування системи на відповідність функціональним вимогам, зокрема працездатність і стабільність основних модулів.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Matt Rogers: Game Development Step by Step: From Concept to Console: A Non-Technical Introduction to Game Design (Step By Step Subject Guides). Amazon Kindle; Kindle Edition, 2024. 147 с.

2. Guidance for Multiplayer Session-Based Game Hosting on AWS [Електронний ресурс]. – Режим доступу до ресурсу: https://aws.amazon.com/ru/solutions/guidance/multiplayer-session-based-game-hosting-on-aws/

3. Бевз С.В. Розробка методу побудови багатокористувацької програми для розвитку логічногомислення / С.В. Бевз, С.М. Бурбело, О.С. Безкревний // Матеріали Міжнародної науково-практичної інтернет-конференції "Молодь в науці: дослідження, проблеми, перспективи (МН-2025)". [Електронний ресурс]. – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/24943/20586

4. A Beginner’s Guide to Multiplayer Game Development [Електронний ресурс]. – Режим доступу до ресурсу: https://www.ixiegaming.com/blog/guide-to-multiplayer-game-development/

5. Intro to Multiplayer Games and How to Create Your Own [Електронний ресурс]. – Режим доступу до ресурсу: https://www.gamedesigning.org/learn/multiplayer/

6. The Rise of Online Multiplayer [Електронний ресурс]. – Режим доступу до ресурсу: https://gameopedia.com/blogs/the-rise-of-online-multiplayer

7. Diep.io [Електронний ресурс]. – Режим доступу до ресурсу: https://diep.io/

8. Krunker.io [Електронний ресурс]. – Режим доступу до ресурсу: https://krunker.io/

9. Territories by obezkr [Електронний ресурс]. – Режим доступу до ресурсу: https://gamejolt.com/games/territories2/813358

10. Websockets 15.0.1 documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://websockets.readthedocs.io/en/stable/

11. Python [Електронний ресурс]. – Режим доступу до ресурсу: https://www.python.org/

12. The Go Programming Language [Електронний ресурс]. – Режим доступу до ресурсу: https://go.dev/

13. Rust Programming Language [Електронний ресурс]. – Режим доступу до ресурсу: https://www.rust-lang.org/

14. Java Oracle [Електронний ресурс]. – Режим доступу до ресурсу: https://www.java.com/en/

15. PyCharm – the only Python IDE you need [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/#

16. Visual Studio Code – Code Editing. Redefined [Електронний ресурс]. – Режим доступу до ресурсу: https://code.visualstudio.com/

17. Project Jupyter Home [Електронний ресурс]. – Режим доступу до ресурсу: https://jupyter.org/

18. Thonny, Python IDE for beginners [Електронний ресурс]. – Режим доступу до ресурсу: https://thonny.org/

19. Що таке SPA-додатки [Електронний ресурс]. – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/chto-takoe-spa-prilozheniya

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

ДОДАТКИ

Додаток А. Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 21 » березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка методу та засобів побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Войтко В. В.
« 21 » березня 2025 р.

Виконав:
студент гр. ЗПІ-21б Безкревний О. С.
« 21 » березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка методу та засобів побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури».

Галузь застосування – ігри та розвиток мислення.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки

Метою роботи є процес покращення досвіду гравців у багатокористувацьких іграх за рахунок розробки власної гри на основі сесійної мультиплеєрної архітектури та впровадження алгоритму генерації випадкових прямокутників, що дозволяє реалізувати гру, орієнтовану на розвиток логічного мислення, у мультиплеєрному режимі.

4. Вихідні дані для проведення НДР

1. Guidance for Multiplayer Session-Based Game Hosting on AWS [Електронний ресурс]. – Режим доступу до ресурсу: https://aws.amazon.com/ru/solutions/guidance/multiplayer-session-based-game-hosting-on-aws/

2. Python [Електронний ресурс]. – Режим доступу до ресурсу: https://www.python.org/

3. Websockets 15.0.1 documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://websockets.readthedocs.io/en/stable/

4. A Beginner’s Guide to Multiplayer Game Development [Електронний ресурс]. – Режим доступу до ресурсу: https://www.ixiegaming.com/blog/guide-to-multiplayer-game-development/

5. Технічні вимоги

Серверний комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 4 ГБ, розмір жорсткого диску 128 ГБ. Операційна система Windows 10.

З клієнтського боку: ОС Windows, браузер Google Chrome.

6. Конструктивні вимоги

Веб сайт повинен відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

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

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку розробки методу та засобів побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури та постановка задач розробки	25.03.25 – 11.04.25	
2	Розробка методів, моделі та алгоритмів роботи програми	14.04.25 – 25.04.25	
3	Розробка програмного забезпечення	28.04.25 – 16.05.25	

4	Тестування програми	19.05.25 – 23.05.25	
5	Оформлення матеріалів до захисту БКР	26.05.25 – 30.05.25	

10. Порядок контролю та прийняття.

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

Додаток Б. Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка методу та засобів побудови багатокористувацької гри на основі сесійної мультиплеєрної архітектури»

Тип роботи: БКР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Войтко В. В.

Оригінальність	97 %
Схожість	3 %

Аналіз звіту подібності

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

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

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism.

Автор роботи _____ Безкревний О. С.

Керівник роботи _____ Войтко В. В.

Додаток В. Лістинг програми

Лістинг файлу main.py

```
from Room import Room
from Player import Player
import asyncio
import json

import websockets


rooms = [Room("TEST1", players_max=4, is_room_full=False, join_randoms=False)]
room_id_count = 10000


'''
When connected to the server, websocket is not assigned to any room.
It will be, when user presses the button to play the game or to join
or create room.

Each room have id and 4 websockets. id is combination of 5 symbols,
and when user wants to join someone, he will type those symbols to
connect to specific room.


'''


async def handler(websocket, path=''):
    global room_id_count
```

```
    print(f"Новий клієнт підключився за шляхом: {path}")
    in_play = False  # Гравець може грати, якщо він грає, йому надсилаються повідомлення зі станом гри.
    #id_of_local_room = -1
    try:
        while True:
            message = await websocket.recv()
            print(f"Отримано повідомлення: {message}")
            json_message = json.loads(message)
            print (json_message)
            if (json_message["action"] == "play"):
                # Put player in the random not full room
                is_there_free_room = False
                for room in rooms:
                    if (room.is_room_full == False and room.join_randoms == True):
                        # Ми додаємо гравця в кімнату, а вже кімната потім вирішує,
                        # що з ним далі робити, які в нього характеристики і все інше
                        room.receivePlayerOnStart(websocket)
                        in_play = True
                        id_of_local_room = room.id
                        is_there_free_room = True
                if (is_there_free_room == False):
                    # Якщо вільної кімнати немає, створюємо її
                    room = Room(room_id_count, players_max=4, is_room_full=False, join_randoms=True)
                    room.receivePlayerOnStart(websocket)
                    in_play = True
                    rooms.append(room)
                    id_of_local_room = room.id
                    room_id_count = room_id_count + 1
                for i in rooms:
                    print (i)
```

```
        elif (json_message["action"] == "join"):
            # Put player in the room with given id
            for room in rooms:
                if json_message["room"] == room.id:
                    if room.is_room_full == False:
                        room.receivePlayerOnStart(websocket)
                        in_play = True

        elif (json_message["action"] == "create"):
            # Create room with received parameters
            pass

        elif (json_message["action"] == "game"):
            # На дії гравця відбуваються події у грі
            print(id_of_local_room)
            for room in rooms:
                if room.id==id_of_local_room:
                    local_room=room

            local_room.game(json_message, websocket) # вебсокет передається щоб зрозуміти який гравець щось робить
            if json_message["move"] != "motion":
                for i in local_room.players:
                    await i.websocket.send(json.dumps(local_room.gameStatusOnceAMove()))
                    await i.websocket.send(json.dumps(local_room.gameStatus()))
            else:
                for i in local_room.players:
                    await i.websocket.send(json.dumps(local_room.gameStatus()))
        print (id_of_local_room)

        # Якщо кімната заповнилась, то розпочинаємо гру
        if (room.started == False and in_play == True and room.is_room_full == True):
```

```
                room.started = True
                room.startGame()
                for i in room.players:
                    await i.websocket.send(json.dumps(room.gameStatusOnceAMove()))
                    await i.websocket.send(json.dumps(room.gameStatus()))




            # await websocket.send(f"Сервер отримав: {message}")
    except websockets.ConnectionClosed:
        print("Клієнт відключився!")
        for room in rooms:
            room.deleteByWebsocket(websocket)
            '''Ми не знаємо, в якій кімнаті знаходиться гравець з цим вебсокетом, тому ми шукаємо
його'''


async def main():
    async with websockets.serve(handler, '0.0.0.0', 8765):
        print("Сервер запущено!")
        await asyncio.Future()  # Блокування сервера

if __name__ == "__main__":
    asyncio.run(main())
```

Лістинг файлу Room.py

```
import json

import Grid
import Block
import random
```

```
import Player


class Room:
    def __init__(self, id, players_max, is_room_full=False, join_randoms=True, grid_width=600, grid_height=600, grid_size=20):
        self.join_randoms = join_randoms
        self.is_room_full = is_room_full
        self.players_max = players_max
        self.players = []
        self.id = id
        self.teams = []
        self.active_player = None
        self.active_block = None
        self.started = False
        self.grid = Grid.Grid(grid_width, grid_height, grid_size)
        for i in range(players_max):
            self.teams.append(i)



    def __str__(self):
        return (str([self.id, self.players, self.players_max, self.is_room_full, self.join_randoms, self.teams]))

    def deleteByWebsocket(self, websocket):
        for player in self.players:
            if player.websocket == websocket:
                self.players.remove(player)

    def receivePlayerOnStart(self, websocket):
        team = len(self.players)+1  # 4 players will be 1 2 3 4
        self.players.append(Player.Player(team=team, websocket=websocket))
```

```
        if (len(self.players) == self.players_max):
            self.is_room_full = True

    def createRect(self, active_player_color):
        width = random.randint(1, 5)  # звичайно ж не так
        height = random.randint (1, 5)
        block = Block.Block(0, 0, width=width*self.grid.size, height=height*self.grid.size,
color=active_player_color)
        return block

    def placeRect(self, rect, grid):
        pass

    def passMove(self):
        pass

    def moveRect(self):
        pass

    def startGame(self):
        self.active_player = self.players[0]
        block = self.createRect(active_player_color=self.active_player.color)
        self.active_block = block
        self.grid.prepareForStart(self.players)
        self.grid.placeBlock(Block.Block(x=0, y=0, height=self.grid.size, width=self.grid.size,
color=self.players[0].color))
        self.grid.placeBlock(
            Block.Block(x=0, y=380, height=self.grid.size, width=self.grid.size,
color=self.players[1].color))
        self.grid.placeBlock(
            Block.Block(x=380, y=380, height=self.grid.size, width=self.grid.size,
color=self.players[2].color))
```

```
        self.grid.placeBlock(
            Block.Block(x=380, y=0, height=self.grid.size, width=self.grid.size,
color=self.players[3].color))

        self.gameStatusOnceAMove()

    def game(self, message, websocket):
        """
        Firstly, we compare player_to_move's websocket and websocket from whom message is.
        If it is really message from player whos turn is, then we proceed with game logic.

        Message is in JSON format like this:
        {'action' : 'game'
        'move' : 'motion'
                'pass'
                'rotate'
                'place'
        }

        If 'motion', then it's just moving block around.
        If 'pass', then player is passing their turn and so on.
        """
        if websocket == self.active_player.websocket:
            if message["move"] == "motion":
                self.active_block.x = message["coordinate_x"]
                self.active_block.y = message["coordinate_y"]

            elif message["move"] == "pass":
                temp_index = self.players.index(self.active_player)
                temp_index = temp_index + 1
                if temp_index >= self.players_max:
                    temp_index = 0
```

```
            self.active_player = self.players[temp_index]
            self.active_block = None
            self.active_player.points = self.active_player.points - 10
            self.createRect(active_player_color=self.active_player.color)
            self.active_block = self.createRect(active_player_color=self.active_player.color)

        elif message["move"] == "rotate":
            self.active_block.width, self.active_block.height = self.active_block.height,
self.active_block.width

        elif message["move"] == "place":
            self.active_block.active = False
            self.grid.placeBlock(self.active_block)
            # then, if placed, then pass move to other player, if not, do nothing todo
            temp_index = self.players.index(self.active_player)
            temp_index = temp_index + 1
            if temp_index >= self.players_max:
                temp_index = 0
            self.active_player = self.players[temp_index]
            self.active_block = self.createRect(active_player_color=self.active_player.color)


    def gameStatus(self):
        #  Це знадобиться для передачі стану блока всім користувачам цієї гри
        #print (self.grid)
        info = {
            "active_block_x" : self.active_block.x,
            "active_block_y" : self.active_block.y,
            "active_block_height" : self.active_block.height,
            "active_block_width" : self.active_block.width,
            "active_block_color": self.active_block.color
            }
```

```
        print (info)
        return info

    def gameStatusOnceAMove(self):
        blocks_info = []
        for block in self.grid.blocks:
            block_data = {
                "x": block.x,
                "y": block.y,
                "width": block.width,
                "height": block.height,
                "color": block.color,
            }
            blocks_info.append(block_data)

        game_state = {
            "blocks": blocks_info
        }
        json_string = json.dumps(game_state)
        with open("game_state.json", "w") as file:
            file.write(json_string)
        #print (1)
        return game_state
```

Лістинг файлу Player.py

```
class Player:
    def __init__(self, team=None, websocket=''):
        self.team = team
        self.points = 0
        if self.team == 1:
            self.color = "#0000ff"
```

```
        elif self.team == 2:
            self.color = "#ff0000"
        elif self.team == 3:
            self.color = "#099900"
        elif self.team == 4:
            self.color = "#e6ee00"
        self.websocket = websocket
```

Лістинг файлу Block.py

```
class Block:
    def __init__(self, x, y, width, height, color):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.active = True
        self.color = color
```

Лістинг файлу Grid.py

```
class Grid:
    def __init__(self, width, height, size):
        self.width = width
        self.height = height
        self.size = size
        self.cells = []
        for i in range(0, int(self.width / self.size)):
            self.cells.append([])
            for j in range(0, int(self.height / self.size)):
                self.cells[i].append([0]) # якщо 0 це free
        self.blocks = []
```

```
    def __str__(self):
        for i in self.cells:
            for j in i:
                print (j)

    def prepareForStart(self, players):
        if len(players) == 2:
            self.cells[0][0].append(players[0].team)
            self.cells[int(self.width / self.size)-1][int(self.height / self.size)-1].append(players[1].team)
        elif len(players) == 4:
            self.cells[0][0].append(players[0].team)
            self.cells[int(self.width / self.size)-1][0].append(players[1].team)
            self.cells[0][int(self.height / self.size)-1].append(players[2].team)
            self.cells[int(self.width / self.size)-1][int(self.height / self.size)-1].append(players[3].team)
```

Лістинг файлу index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Territories Online</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            background-color: #f4f4f4; /* Light gray color */
            background-image: url('bg.png');
            background-size: cover;
            background-position: center;
            background-repeat: no-repeat;
            padding: 50px;
```

```
}
button {
    display: block;
    margin: 20px auto;
    padding: 15px 30px;
    font-size: 18px;
    background-color: #007BFF;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}
button:hover {
    background-color: #0056b3;
}
.hidden { display: none; }
.banner {
    position: fixed;
    top: 0;
    bottom: 0;
    width: 120px;
    background-color: #ccc;
    text-align: center;
    font-size: 20px;
    color: #333;
}
#leftBanner {
    left: 0;
}
#rightBanner {
    right: 0;
```

```
}
.fixed-button {
    position: fixed;
    width: 50px;
    height: 50px;
    background-color: transparent; /* Remove default color */
    border: none;
    cursor: pointer;
    background-size: contain; /* Ensure the entire image fits within the button */
    background-position: center;
    background-repeat: no-repeat;
}
#bottomLeftButton {
    bottom: 10px;
    left: 130px;
    background-image: url('bottom-left.png'); /* Replace with actual image path */
}
#topRightButton {
    top: 10px;
    right: 130px;
    background-image: url('top-right.png'); /* Replace with actual image path */
}
.modal {
    display: none; /* Hidden by default */
    position: fixed;
    z-index: 1;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto;
    background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
```

```
}
.modal-content {
    background-color: #fefefe;
    margin: 15% auto;
    padding: 20px;
    border: 1px solid #888;
    width: 80%;
    max-height: 70%;
    overflow-y: auto;
}
.close {
    color: #aaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}
.close:hover,
.close:focus {
    color: black;
    text-decoration: none;
    cursor: pointer;
}
.color-modal {
    display: none; /* Hidden by default */
    position: fixed;
    z-index: 1;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto;
    background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
```

```
        }
        .color-modal-content {
            background-color: #fefefe;
            margin: 15% auto;
            padding: 20px;
            border: 1px solid #888;
            width: 50%;
            max-height: 50%;
            overflow-y: auto;
        }
        .color-option {
            display: inline-block;
            width: 30px;
            height: 30px;
            margin: 5px;
            cursor: pointer;
            border: 1px solid #ccc;
        }
        #scoreSidebar {
            position: absolute;
            left: 420px; /* Position it right next to the canvas */
            top: 50px;
            width: 100px; /* Make it smaller */
            background-color: transparent; /* Remove background */
            padding: 5px;
            border: none; /* Remove border */
            display: none; /* Hidden by default */
        }
    </style>
</head>

<body>
```

```
<div id="leftBanner" class="banner">
   <a href="https://example.com/ad1" target="_blank">
      <img src="https://via.placeholder.com/120x600" alt="Ad 1">
   </a>
</div>
<div id="rightBanner" class="banner">
   <a href="https://example.com/ad2" target="_blank">
      <img src="https://via.placeholder.com/120x600" alt="Ad 2">
   </a>
</div>
<img id="logo" src="logo.png" alt="Logo" style="max-width: 100%; height: auto;" />
<form id="nicknameForm">
   <input type="text" id="nickname" placeholder="Enter your nickname" required />
</form>
<button id="playButton">Грати</button>
<button onclick="showJoinRoom()" id="joinButton">Підключитись</button>
<button onclick="showCreateRoom()" id="createButton">Створити кімнату</button>

<!--
<div id="roomInput">
   <input type="text" id="roomName" placeholder="Введіть ім'я кімнати">
   <button id="confirmRoomButton">Підтвердити</button>
</div>
-->

<div id="joinRoom" class="hidden">
   <input type="text" id="roomName" placeholder="Enter room name" />
   <button id="joinRoomButton">Підтвердити</button>
</div>

<div id="createRoom" class="hidden">
```

```
        <input type="text" id="createRoomName" placeholder="Enter room name" style="width:
10%; margin-bottom: 10px;" />
        <br>
        <input type="number" id="maxPlayers" placeholder="Max players" min="1" max="100"
/>
        <br>
        <label><input type="checkbox" id="allowRandoms" /> Випадкові гравці</label>
        <br>
        <label><input type="checkbox" id="teamMode" /> Командний режим</label>
        <br>
        <label><input type="checkbox" id="customMap" /> Створення своєї карти</label>
        <br>
        <button id="createRoomButton" >Підтвердити</button>
    </div>

    <canvas id="gameCanvas" width="400" height="400" class="hidden"></canvas>

    <div id="chatContainer" class="hidden">
        <div id="chatMessages" style="height: 150px; overflow-y: auto; border: 1px solid #ccc;
margin-bottom: 10px; padding: 5px;"></div>
        <input type="text" id="chatInput" placeholder="Type a message..." style="width: 80%;"
/>
        <button id="sendChatButton">Відправити</button>
    </div>

    <button id="bottomLeftButton" class="fixed-button"></button>
    <button id="topRightButton" class="fixed-button"></button>

    <!-- Modal structure -->
    <div id="rulesModal" class="modal">
        <div class="modal-content">
            <span class="close" id="closeRules">&times;</span>
```

```
            <h2>Правила гри</h2>
            <p>Правила:
                Випадково генерується прямокутник. Прямокутники повинні бути розміщені поруч з існуючою територією.
                До рахунку додається стільки очок, скільки клітинок зайнято прямокутником.
                Якщо ви не можете поставити прямокутник, ви пропускаєте хід і втрачаєте 10 очок.
                Якщо у вас негативна кількість очок, ви виходите з гри. Решта гравців продовжують грати.
                Гра закінчується, коли вся територія зайнята або поки не залишиться один гравець. Переможцем стає гравець, який має більше очок.</p>
        </div>
    </div>

    <!-- Color palette modal structure -->
    <div id="colorModal" class="color-modal">
        <div class="color-modal-content">
            <span class="close" id="closeColorModal">&times;</span>
            <h2>Обрати колір</h2>
            <input type="color" id="colorPicker" value="#ff0000" style="width: 100%; height: 50px; border: none;" />
            <button id="confirmColorButton">Підтвердити</button>
        </div>
    </div>

    <script type="text/javascript" src="script.js">
    </script>
</body>
</html>
```

Лістинг script.js

```
const canvas = document.getElementById("gameCanvas");
```

```
const ctx = canvas.getContext("2d");

let activeRect = {
    x: 0,
    y: 0,
    width: 0,
    height: 0,
    color: "black" // Default color
};

let isDragging = false;
let offsetX, offsetY;

// Global array to store blocks
let blocks = [];

function showJoinRoom() {
    document.getElementById("joinRoom").classList.remove("hidden");
    document.getElementById("createRoom").classList.add("hidden");
}

function showCreateRoom() {
    document.getElementById("createRoom").classList.remove("hidden");
    document.getElementById("joinRoom").classList.add("hidden");
}

function joinRoom() {
    const roomName = document.getElementById("roomName").value.trim();
    if (roomName === "") {
        alert("Please enter a room name.");
        return;
    }
```

```
    const message = {
        action: "join",
        room: roomName
    };

    socket.send(JSON.stringify(message));
    console.log("Sent:", message);
}

function createRoom() {
    const maxPlayers = parseInt(document.getElementById("maxPlayers").value, 10);
    const allowRandoms = document.getElementById("allowRandoms").checked;

    if (isNaN(maxPlayers) || maxPlayers < 1) {
        alert("Please enter a valid number of players.");
        return;
    }

    const message = {
        action: "create",
        maxPlayers: maxPlayers,
        allowRandoms: allowRandoms
    };

    socket.send(JSON.stringify(message));
    console.log("Sent:", message);
}

function hideMenuAndShowGame() {
    document.getElementById("playButton").style.display = "none";
    document.getElementById("joinButton").style.display = "none";
```

```
    document.getElementById("createButton").style.display = "none";
    document.getElementById("joinRoom").classList.add("hidden");
    document.getElementById("createRoom").classList.add("hidden");
    document.getElementById("nicknameForm").style.display = "none";
    document.getElementById("logo").style.display = "none";

    document.getElementById("gameCanvas").classList.remove("hidden");
    document.getElementById("scoreSidebar").style.display = "block";
    showChat();
    drawGrid();
}

// Функція для малювання прямокутника
function drawActiveRect() {
    //ctx.clearRect(0, 0, canvas.width, canvas.height);
    //drawGrid(); // якщо хочеш сітку
    ctx.fillStyle = activeRect.color;
    ctx.fillRect(activeRect.x, activeRect.y, activeRect.width, activeRect.height);
}

// Функція для оновлення прямокутника на основі даних з сервера
function updateActiveRectFromServer(data) {
    activeRect.x = data.active_block_x;
    activeRect.y = data.active_block_y;
    activeRect.width = data.active_block_width;
    activeRect.height = data.active_block_height;
    activeRect.color = data.active_block_color;
    drawActiveRect();
}

// Події мишки для перетягування
canvas.addEventListener("mousedown", (e) => {
```

```
    const rect = canvas.getBoundingClientRect();
    const mouseX = e.clientX - rect.left;
    const mouseY = e.clientY - rect.top;

    if (
        mouseX >= activeRect.x &&
        mouseX <= activeRect.x + activeRect.width &&
        mouseY >= activeRect.y &&
        mouseY <= activeRect.y + activeRect.height
    ) {
        isDragging = true;
        offsetX = mouseX - activeRect.x;
        offsetY = mouseY - activeRect.y;
    }
});

function sendRectangleMove() {
    const message = {
        action: "game",
        move: "motion",
        coordinate_x: activeRect.x,
        coordinate_y: activeRect.y
    };

    if (socket.readyState === WebSocket.OPEN) {
        socket.send(JSON.stringify(message));
        console.log('Sent:', message);
    } else {
        console.error('WebSocket is not connected!');
    }
}
```

```
// Modify the mousemove event listener to send the message when the rectangle moves
canvas.addEventListener("mousemove", (e) => {
    if (isDragging) {
        const rect = canvas.getBoundingClientRect();
        const mouseX = e.clientX - rect.left;
        const mouseY = e.clientY - rect.top;

        activeRect.x = mouseX - offsetX;
        activeRect.y = mouseY - offsetY;

        drawActiveRect();
        sendRectangleMove(); // Send the message when the rectangle moves
    }
});

canvas.addEventListener("mouseup", () => {
    isDragging = false;

    // Send a message with action 'place' when the rectangle is positioned
    const message = {
        action: "game",
        move: "place",
        coordinate_x: activeRect.x,
        coordinate_y: activeRect.y
    };

    if (socket.readyState === WebSocket.OPEN) {
        socket.send(JSON.stringify(message));
        console.log('Sent:', message);
    } else {
        console.error('WebSocket is not connected!');
    }
```

```
});

canvas.addEventListener("mouseleave", () => {
    isDragging = false;
});


// Підключення до сервера WebSocket
const socket = new WebSocket('ws://localhost:8765');

socket.onopen = () => {
    console.log('Підключення до сервера WebSocket встановлено!');
};

socket.onmessage = (event) => {
    console.log(`Received message from server: ${event.data}`);
    try {
        const data = JSON.parse(event.data);
        if (data.action === "chat") {
            const chatMessages = document.getElementById("chatMessages");
            const newMessage = document.createElement("div");
            newMessage.textContent = `${data.sender}: ${data.message}`;
            chatMessages.appendChild(newMessage);
            chatMessages.scrollTop = chatMessages.scrollHeight; // Auto-scroll to the latest message
        }
        if (data.blocks) {
            // Update the global blocks array
            blocks = data.blocks;
        }

        if (data.active_block_x !== undefined) {
            updateActiveRectFromServer(data);
```

```
        }

        // Clear the canvas once
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        ctx.fillStyle = "#ffffff";
        ctx.fillRect(0, 0, 400, 400);
        drawGrid();
        // Redraw all blocks
        blocks.forEach(block => {
            ctx.fillStyle = block.color || "#000000"; // Default to black if no color is specified
            ctx.fillRect(block.x, block.y, block.width, block.height);
            console.log(block);
        });

        // Draw the active rectangle
        drawActiveRect();
    } catch (error) {
        console.error('Error:', error);
    }
};

socket.onclose = () => {
    console.log('З\'єднання з сервером закрито.');
};

socket.onerror = (error) => {
    console.error('Помилка WebSocket:', error);
};

// Елементи сторінки
const playButton = document.getElementById('playButton');
const joinButton = document.getElementById('joinButton');
```

```
const createButton = document.getElementById('createButton');

const joinRoomButton = document.getElementById('joinRoomButton');
const createRoomButton = document.getElementById('createRoomButton');

const roomNameInput = document.getElementById('roomName');


// Логіка для кнопки "Play"
playButton.addEventListener('click', () => {
    if (socket.readyState === WebSocket.OPEN) {
        socket.send(JSON.stringify({ action: 'play', value: 0 })); // Відправляємо "0"
        console.log('Відправлено: "play", value: 0');
    } else {
        console.error('WebSocket не підключено!');
    }
});


document.getElementById("playButton").addEventListener("click", hideMenuAndShowGame);
document.getElementById("joinRoomButton").addEventListener("click", () => {
    joinRoom();
    hideMenuAndShowGame();
});
document.getElementById("createRoomButton").addEventListener("click", () => {
    createRoom();
    hideMenuAndShowGame();
});

// Show chat when the game starts
function showChat() {
    document.getElementById("chatContainer").classList.remove("hidden");
```

```
}

// Modify the event listener for sending chat messages to include the sender's name
document.getElementById("sendChatButton").addEventListener("click", () => {
    const chatInput = document.getElementById("chatInput");
    const message = chatInput.value.trim();
    const nickname = document.getElementById("nickname").value.trim(); // Get the sender's name
    if (message !== "" && nickname !== "") {
        const chatMessage = {
            action: "chat",
            sender: nickname, // Include the sender's name
            message: message
        };
        if (socket.readyState === WebSocket.OPEN) {
            socket.send(JSON.stringify(chatMessage));
            console.log('Sent:', chatMessage);
            chatInput.value = ""; // Clear input after sending
        } else {
            console.error('WebSocket is not connected!');
        }
    }
});

// Function to draw grid lines on the canvas
function drawGrid() {
    const gridSize = 20;
    ctx.strokeStyle = '#000000'; // Black color for grid lines
    ctx.lineWidth = 0.5;

    // Draw vertical lines
    for (let x = 0; x <= canvas.width; x += gridSize) {
        ctx.beginPath();
```

```
        ctx.moveTo(x, 0);
        ctx.lineTo(x, canvas.height);
        ctx.stroke();
    }

    // Draw horizontal lines
    for (let y = 0; y <= canvas.height; y += gridSize) {
        ctx.beginPath();
        ctx.moveTo(0, y);
        ctx.lineTo(canvas.width, y);
        ctx.stroke();
    }
}

// Get the modal and close button elements
const rulesModal = document.getElementById("rulesModal");
const closeRules = document.getElementById("closeRules");

// Show the modal when the 'Rules' button is clicked
document.getElementById("bottomLeftButton").addEventListener("click", () => {
    rulesModal.style.display = "block";
});

// Hide the modal when the close button is clicked
closeRules.addEventListener("click", () => {
    rulesModal.style.display = "none";
});

// Hide the modal when clicking outside of it
window.addEventListener("click", (event) => {
    if (event.target === rulesModal) {
        rulesModal.style.display = "none";
```

```
    }
});

// Get the color modal and close button elements
const colorModal = document.getElementById("colorModal");
const closeColorModal = document.getElementById("closeColorModal");
let selectedColor = null;

// Show the color modal when the 'Settings' button is clicked
document.getElementById("topRightButton").addEventListener("click", () => {
    colorModal.style.display = "block";
});

// Hide the color modal when the close button is clicked
closeColorModal.addEventListener("click", () => {
    colorModal.style.display = "none";
});

// Hide the color modal when clicking outside of it
window.addEventListener("click", (event) => {
    if (event.target === colorModal) {
        colorModal.style.display = "none";
    }
});

// Handle color selection
const colorOptions = document.querySelectorAll(".color-option");
colorOptions.forEach(option => {
    option.addEventListener("click", () => {
        // Remove selection from all options
        colorOptions.forEach(opt => opt.style.border = "1px solid #ccc");
        // Highlight the selected option
```

```
        option.style.border = "2px solid black";
        selectedColor = option.style.backgroundColor;
    });
});

// Confirm color selection
const confirmColorButton = document.getElementById("confirmColorButton");
confirmColorButton.addEventListener("click", () => {
    const selectedColor = document.getElementById("colorPicker").value;
    console.log("Selected color:", selectedColor);
    colorModal.style.display = "none";
});
```

Додаток Г. Ілюстративна частина

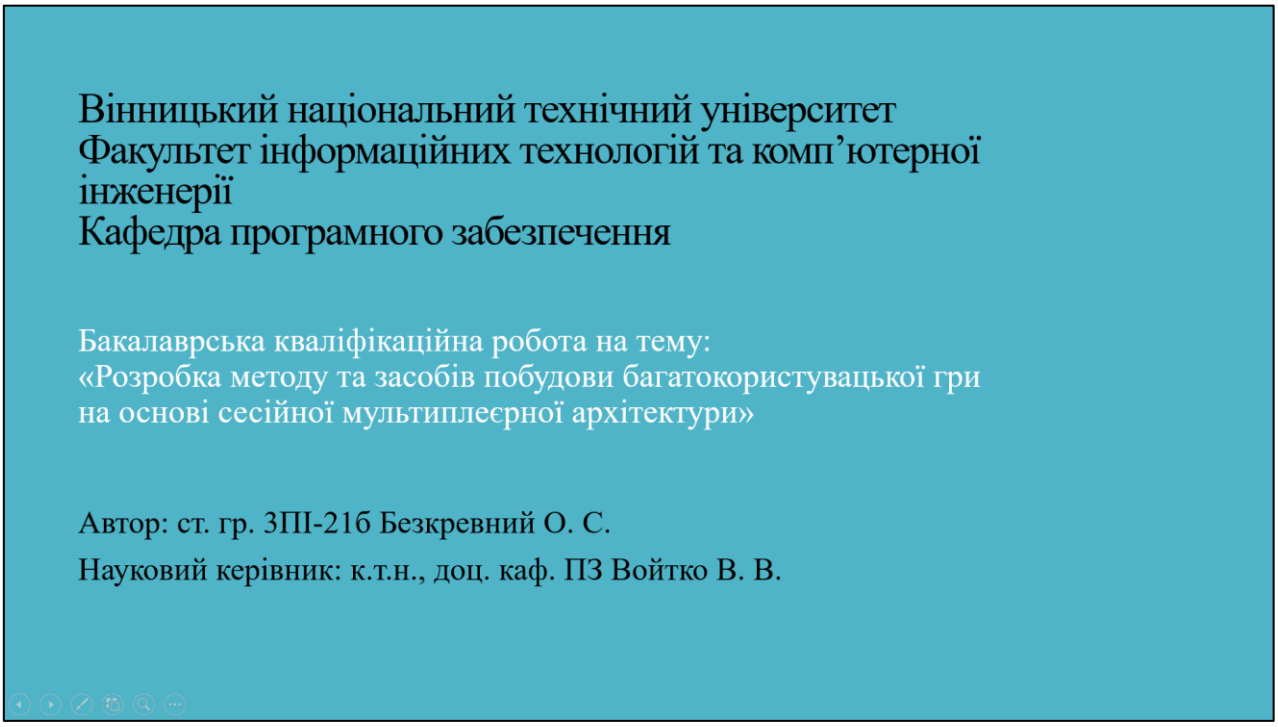


Рисунок Г.1 – Титульний слайд

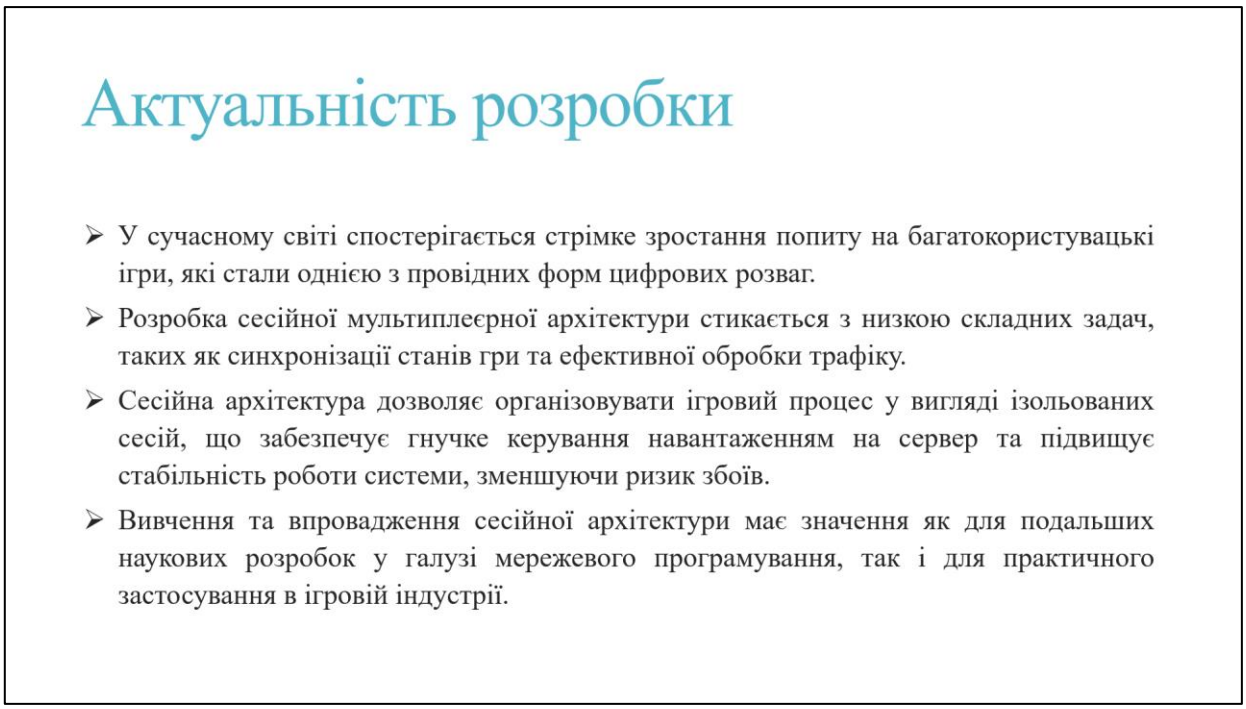


Рисунок Г.2 – Актуальність розробки

Мета, предмет та об’єкт дослідження

Метою роботи є процес покращення досвіду гравців у багатокористувацьких іграх за рахунок розробки власної гри на основі сесійної мультиплеєрної архітектури та впровадження алгоритму генерації випадкових прямокутників, що дозволяє реалізувати гру, орієнтовану на розвиток логічного мислення, у мультиплеєрному режимі.

Об'єктом дослідження є процес розробки багатокористувацької гри на основі сесійної мультиплеєрної архітектури.

Предметом дослідження є методи і засоби розробки багатокористувацької гри, яка буде основана на сесійній мультиплеєрній архітектурі, використання мови програмування Python та способи використання середовища програмування PyCharm.

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

Наукова новизна

1. Подальшого розвитку набув метод випадкової генерації прямокутників, який, на відміну від стандартних підходів, дозволяє забезпечити рівномірний розподіл усіх можливих варіантів площ, навіть якщо деякі з них математично мають більше комбінацій сторін, що дозволяє досягти контрольованого, збалансованого динамічного вирівнювання розподілу прямокутників за площею при збереженні ефекту випадковості, що покращує ігровий процес та забезпечує рівні умови для гравців.

2. Подальшого розвитку отримала модель багатокористувацької гри на основі сесійної мультиплеєрної архітектури, яка, на відміну від існуючих, дозволяє користувачам створювати ігрові кімнати з власними налаштуваннями, приєднуватися до випадкових або запрошених сесій, що дозволяє користувачам грати у гру, яка розвиває логічне мислення, у мультиплеєрному режимі.

Рисунок Г.4 – Наукова новизна

Постановка задачі

- провести аналіз існуючих аналогів багатокористувацьких ігор;
- розробити модель та алгоритм роботи багатокористувацької гри на основі сесійної мультиплеєрної архітектури;
- розробити функціонал програми з можливістю підключення до випадкового лобі;
- можливість створення та підключення до конкретного лобі, який визначається унікальним ідентифікатором;
- можливість налаштовування створеної кімнати для гри за допомогою параметрів;
- функціонал для кастомізації внутрішньоігрового кольору блоків;
- можливість змінювати ім’я користувача;
- можливість спілкування між гравцями у внутрішньоігровому чаті.

Рисунок Г.5 – Постановка задачі

Практична цінність отриманих результатів

Практичне значення проєкту полягає у можливості використання вебзастосунку, який може бути використаний як платформа для організації багатокористувацької взаємодії в реальному часі на основі сесійної архітектури. Розроблена система дозволяє користувачам швидко створювати та налаштовувати ігрові кімнати, забезпечує стабільну взаємодію та може бути використана як основа для створення інших онлайн ігор. Гра розвиває логічне мислення та допомагає поліпшити розумові навички. Рішення є універсальним і може застосовуватись як для навчальних, так і для розважальних проєктів, а також як тестовий майданчик для дослідження продуктивності сесійної мультиплеєрної архітектури. Програмні модулі системи легко масштабуються, розширюються та адаптуються для різних жанрів ігрових механік, що робить її корисною не лише для кінцевих користувачів, але й для розробників у сфері геймдеву.

Рисунок Г.6 – Практична цінність отриманих результатів

Аналіз аналогів

Критерій порівняння	Diep.io	Krunker.io	Territories	Власна розробка
Чат для спілкування	0	1	0	1
Незалежність від затримок	0	0	1	1
Унеможливлення читів	1	0	1	1
Підтримка приватних лобі	1	1	0	1
Наявність командних режимів	1	1	0	1
Сумарний коефіцієнт	3	3	2	5

Рисунок Г.7 – Аналіз аналогів

Метод генерації прямокутників

1. Зі збереженого словника rect_counter, який містить кількість генерацій кожної площі, передаються статистичні дані у функцію createNum;
2. Якщо загальна кількість згенерованих прямокутників менша за 10, вибір сторін відбувається випадково;
3. Розраховуються ймовірності обернені ваги для кожної площі pos_s;
4. Використовується функція random.random() для генерації числа rand від 0 до 1;
5. Допоки rand менший за ймовірність pos_s, сумуються ймовірностей rand та pos_s для кожної з площ;
6. Присвоїти height та width числові значення довжини та ширини прямокутника із ймовірністю вибору rand;
7. Збільшити лічильник rect_counter на 1.

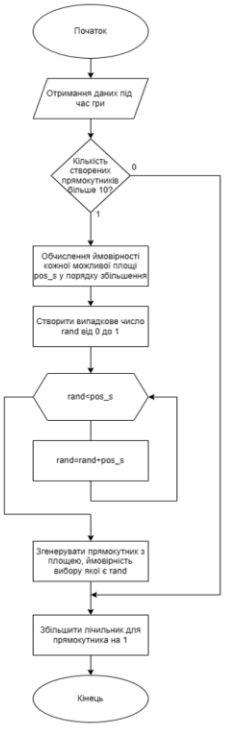


Рисунок Г.8 – Метод генерації прямокутників

Діаграма класів моделі системи

Діаграма включає ключові компоненти, зокрема Room (ігрова кімната), Player (гравець), Grid (ігрове поле) та Block (ігровий блок), і відображає їхні атрибути, методи та зв'язки між ними. Такий підхід дозволив формалізувати модель ігрового процесу, зробити її більш зрозумілою для подальшої реалізації та тестування.

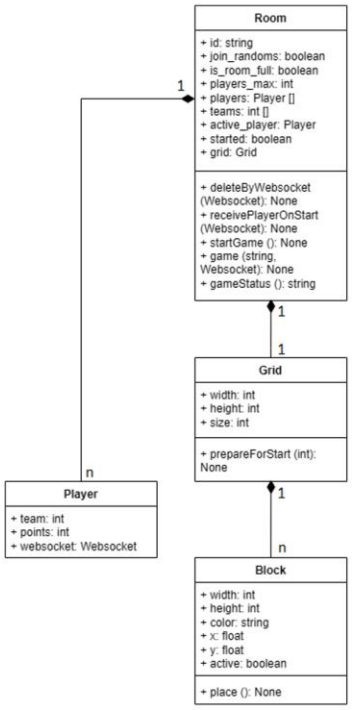
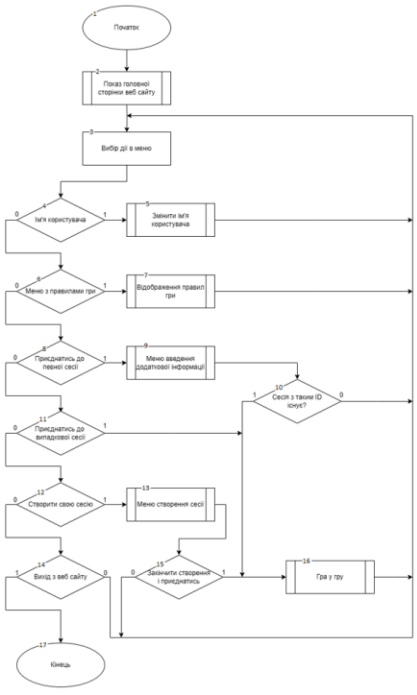


Рисунок Г.9 – Діаграма класів моделі системи



Загальний алгоритм роботи системи

Для розробки багатокористувацької гри на основі сесійної мультиплеєрної архітектури необхідно створити загальний алгоритм, що визначає логіку взаємодії користувача із системою, а також послідовність виконання дій на кожному етапі використання веб сайту. Цей алгоритм відображено у вигляді блок-схеми.

Рисунок Г.10 – Загальний алгоритм роботи системи

Модуль ігрової логіки

Модуль ігрової логіки відповідає за обробку дій гравців під час гри, базуючись на повідомленнях, які надходять від клієнтів через websocket-з'єднання. Робота модуля починається з моменту отримання JSON-повідомлення від клієнта.

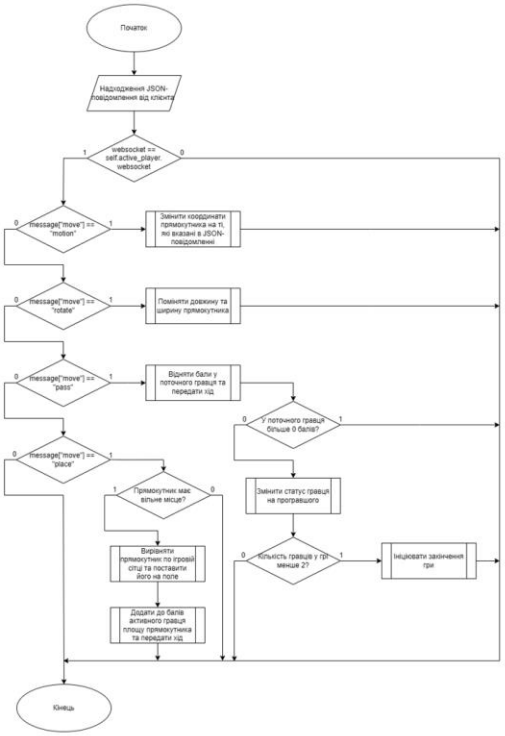


Рисунок Г.11 – Модуль ігрової логіки

Використані технології

Рисунок Г.12 – Використані технології

Тестування сайту

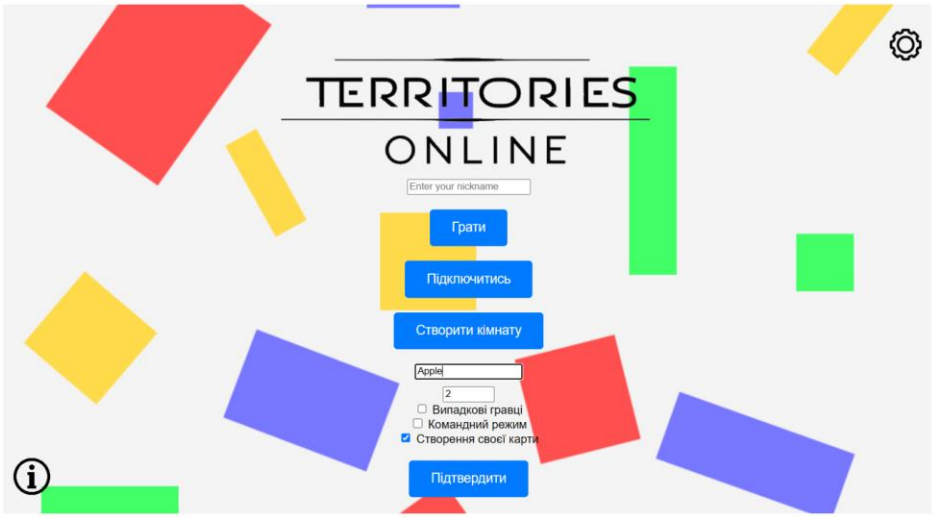


Рисунок Г.13 – Тестування сайту

Апробація та публікація результатів роботи

Результати бакалаврської кваліфікаційної роботи були представлені на міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» та опубліковані у вигляді тез доповідей на цій конференції.

За тематикою дослідження опубліковано 1 наукову працю у збірниках матеріалів конференції.

Рисунок Г.14 – Апробація та публікація результатів

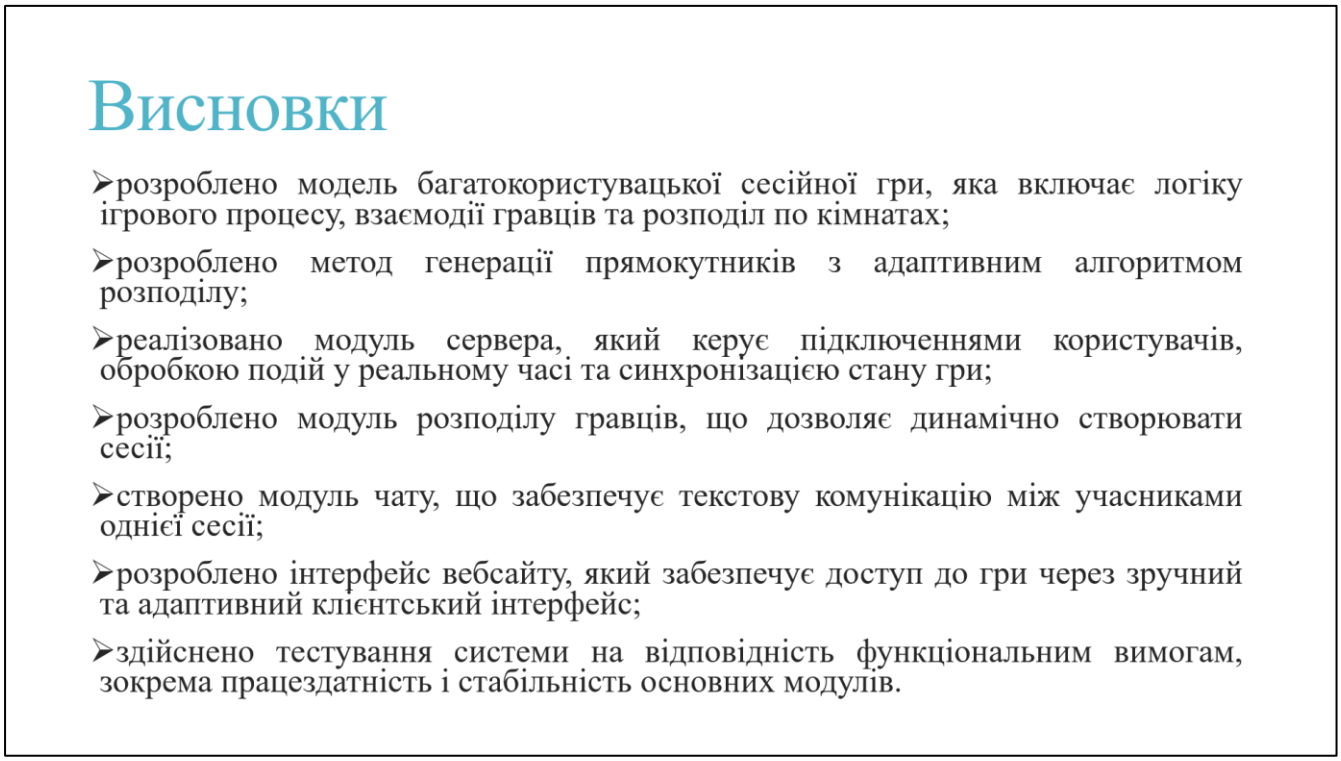


Рисунок Г.15 – Висновки

Рисунок Г.16 – Фінальний слайд