

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Вебплатформа настільно-рольових ігор: Модуль віртуальний ігровий стіл»

Виконав студент групи ІПЗ-22-4

БОРДЮГ Денис Вікторович

Керівник роботи:

ТАЛАВЕР Олег Володимирович

Рецензент:

КОНЦИДАЙЛО Андрій Михайлович

Житомир - 2026

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

«ЗАТВЕРДЖУЮ»

Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БОРДЮГ Денис Вікторович**

Керівник роботи: **ТАЛАВЕР Олег Володимирович**

Тема роботи: **«Вебплатформа настільно-рольових ігор: Модуль віртуальний ігровий стіл»,**

затверджена Наказом закладу вищої освіти від «27» лютого 2026 р., №92/с

Вихідні дані для роботи: **розробка модуля віртуального ігрового столу для вебплатформи настільно-рольових ігор, що забезпечує візуалізацію ігрових карт, управління токенами та автоматизацію кидків кубиків.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Талавер О.В.	02.03.2026р.	11.03.2026р.
2	Талавер О.В.	15.04.2026р.	29.04.2026р.
3	Талавер О.В.	07.05.2026р.	22.05.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел із 34 найменувань, 1 додатків (10 сторінок), 15 таблиць, та 46 ілюстрацій по тексту. Повний обсяг роботи становить 107 сторінок, з них 97 сторінок основного тексту.

Метою роботи є проектування та розробка вебплатформи для настільних рольових ігор та забезпечення просторової візуалізації ігрового процесу і інтерактивного управління об'єктами в реальному часі.

У першому розділі охарактеризовано предметну область настільних рольових ігор, виконано огляд та порівняльний аналіз існуючих платформ віртуальних столів, сформульовано задачу дослідження та обґрунтовано вибір технологій реалізації.

У другому розділі визначено вимоги до модуля віртуального ігрового столу, обрано архітектурний підхід та інструментальні засоби розробки. Спроектовано архітектуру об'єктної моделі ігрового середовища, алгоритми управління станом застосунку та механізми взаємодії між клієнтом і сервером.

У третьому розділі описано реалізацію клієнтської та серверної частин, процеси рендерингу карти та синхронізації дій у реальному часі. Наведено приклади функціоналу та результати тестування системи

В результаті виконання роботи розроблено вебплатформу, що забезпечує повноцінну візуалізацію та проведення настільних рольових ігор онлайн: від завантаження ігрових карт і управління токенами до автоматизації кидків кубиків та миттєвої синхронізації ігрового стану між усіма учасниками сесії.

КЛЮЧОВІ СЛОВА: НАСТІЛЬНІ РОЛЬОВІ ІГРИ, ВІРТУАЛЬНИЙ СТІЛ, CANVAS, WEBSOCKET, PIXIJS, REACT.

					ІПЗ.КР.Б - 121 - 26 - ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.		Бордюг Д. В.			«Вебплатформа настільно-рольових ігор: Модуль віртуальний ігровий стіл» Пояснювальна записка	Лім.	Арк.	Аркушів	
Керівник		Талавер О. В.					3	107	
Рецензент		Концидайло А.М.				Житомирська політехніка, група ІПЗ-22-4			
Зав. каф.		Вакалюк Т.А.							

ABSTRACT

The bachelor's thesis consists of an introduction, three chapters, conclusions, a list of references with 34 sources, 1 appendices (10 pages), 15 tables, and 46 illustrations. The total volume of the thesis is 107 pages, of which 97 pages constitute the main body.

The main goal of the thesis is to design and develop a web platform for conducting tabletop role-playing games, ensuring spatial visualization of the gaming process and interactive object management in real time.

The first chapter analyzes the subject area of tabletop role-playing games and existing virtual tabletops, formulates the research problem, and justifies the choice of implementation technologies.

The second chapter defines the requirements for the virtual tabletop module and selects the architectural approach. The object model of the game environment, application state management algorithms, and client-server interaction mechanisms are designed.

The third chapter describes the software implementation of the client and server parts, the map rendering processes, and real-time action synchronization. Graphic optimization measures and system testing results are presented.

As a result of the work, a web platform has been developed that provides full-fledged visualization and online conduction of tabletop role-playing games: from loading game maps and managing tokens to automating dice rolls and instantly synchronizing the game state among all session participants.

KEYWORDS: TABLETOP ROLE-PLAYING GAMES, VIRTUAL TABLETOP, CANVAS, WEBSOCKET, PIXI.JS, REACT.

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	12
1.3 Вибір архітектури системи	16
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	19
Висновки до першого розділу	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ НАСТІЛЬНО РОЛЬОВИХ ІГОР: МОДУЛЬ ВІРТУАЛЬНИЙ ІГРОВИЙ СТІЛ	26
2.1 Визначення вимог до системи	26
2.2 Розробка бази даних системи	32
2.3 Проєктування та реалізація алгоритмів роботи системи	37
2.4 Реалізація функціоналу модуля віртуального ігрового столу	43
Висновки до другого розділу	56
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ НАСТІЛЬНО-РОЛЬОВИХ ІГОР	57
3.1 Порядок встановлення та налаштування параметрів платформи	57
3.2 Структура інтерфейсу платформи	60
3.3 Тестування платформи	86
Висновки до третього розділу	92
ВИСНОВКИ	93
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	95
ДОДАТКИ	98
Додаток А	99

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HPI - настільні рольові ігри.

API (Application Programming Interface) - інтерфейс прикладного програмування.

CSR (Client-Side Rendering) - рендеринг (відмальовування) на стороні клієнта.

DOM (Document Object Model) - об'єктна модель документа.

JSON (JavaScript Object Notation) - текстовий формат обміну даними.

ORM (Object-Relational Mapping) - об'єктно-реляційне відображення.

SPA (Single Page Application) - односторінковий вебзастосунок.

VT (Virtual Tabletop) - віртуальний ігровий стіл.

WebGL (Web Graphics Library) - кросплатформне API для апаратного прискорення рендерингу 2D та 3D графіки в браузері.

DDD (Domain-Driven Design) - предметно-орієнтоване проектування.

WS (WebSocket) - протокол повнодуплексного двостороннього зв'язку в реальному часі.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність. Наразі індустрія настільних рольових ігор, таких як Dungeons & Dragons, Pathfinder та інші, переживає період стрімкого зростання та диджиталізації [28]. Гравці та майстри гри все частіше переходять до онлайн-форматів або використовують цифрові інструменти для проведення живих сесій. Проте повноцінна візуалізація ігрового процесу у дистанційному форматі залишається складним завданням, що вимагає забезпечення просторової уяви, відстеження позицій персонажів та швидкого обчислення ігрових механік. Застосування сучасних архітектурних рішень та засобів вебпрограмування дає змогу реалізувати ефективний рендеринг інтерактивної карти у браузері, а також гарантувати миттєву синхронізацію просторового стану для всіх користувачів.

Існуючі комерційні аналоги часто є перевантаженими складним інтерфейсом або вимагають високої абонентської плати за доступ до базового функціоналу (наприклад, завантаження кастомних карт), що створює значний поріг входження. Внаслідок відсутності зручного безкоштовного рішення значна частина гравців змушена витратити час на рутинні математичні підрахунки або використовувати примітивні графічні редактори. Тому розробка спеціалізованого модуля віртуального столу, який об'єднує засоби відображення ігрової дошки, інтерактивне керування токенами та автоматизацію кидків кубиків у межах єдиної екосистеми, є актуальним завданням для покращення користувацького досвіду та динаміки гри.

Метою роботи є проєктування та розробка вебплатформи для проведення настільних рольових ігор із забезпеченням візуалізації ігрового середовища та засобів інтерактивної взаємодії з віртуальними об'єктами.

Для досягнення поставленої мети необхідно вирішити такі завдання: проаналізувати предметну область віртуальних ігрових столів та особливості візуалізації настільних рольових ігор; виконати огляд існуючих програмних рішень для відображення ігрового простору та визначити їхні обмеження; обґрунтувати вибір архітектурного підходу та інструментальних засобів

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

реалізації; спроектувати структуру клієнтської та серверної частин модуля і модель даних; програмно реалізувати механізми рендерингу графіки та двосторонньої синхронізації в реальному часі; провести функціональне тестування розробленого програмного продукту.

Об'єктом дослідження є візуалізація та автоматизація ігрових механік настільних рольових ігор у дистанційному режимі, що охоплює просторову взаємодію гравців на віртуальній карті.

Предметом дослідження є методи, алгоритми та програмні засоби розробки веборієнтованих систем для відображення ігрового простору (VTT) та миттєвої синхронізації стану застосунку.

Методи дослідження. У роботі застосовано комплекс методів системного аналізу предметної області, принципи об'єктно-орієнтованого проєктування та мову UML для моделювання архітектури. Програмна реалізація базується на сучасних підходах до вебпрограмування, методах рендерингу графіки та технологіях передачі даних у реальному часі, а перевірка працездатності системи здійснювалася за допомогою методів функціонального тестування.

Практичне значення отриманих результатів полягає у створеному програмному рішенні, яке автоматизує рутинні ігрові процеси: маніпуляцію ігровими об'єктами на карті та розрахунок результатів кидків кубиків. Розроблений модуль віртуального столу спрощує просторову взаємодію учасників з ігровим світом та забезпечує централізоване середовище для візуалізації, що підвищує динамічність гри та покращує загальний користувацький досвід.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

1.1 Постановка задачі

Попри стрімке зростання популярності настільних рольових ігор та масовий перехід ігрових спільнот в онлайн-простір, організація та візуалізація ігрових сесій у дистанційному форматі залишається технічно складним процесом.

Ключовою проблемою є те, що гра у віддаленому режимі без використання спеціалізованого програмного забезпечення суттєво ускладнює просторову уяву учасників, перешкоджає тактичній взаємодії на ігровому полі та ускладнює відстеження позицій персонажів під час бойових зіткнень. Традиційні настільні ігри передбачають наявність фізичного столу, мініатюр та паперових карт, що робить ігровий процес інтуїтивно зрозумілим. При переході в цифровий формат цю просторову наочність необхідно ефективно імітувати.

Існуючі цифрові платформи для візуалізації ігрового простору, або так звані VTT, мають низку суттєвих недоліків, що обмежують їх масове використання. Значна частина з них має перевантажений надлишковим функціоналом інтерфейс, що створює високий поріг входження для нових гравців. Інші популярні рішення вимагають платної підписки для доступу до базового функціоналу, такого як завантаження власних ігрових карт або використання інтерактивних листів персонажів.

Окрім того, ручні підрахунки результатів кидків кубиків та визначення дистанцій на карті забирають у гравців та майстра гри значну частину часу, що негативно впливає на динаміку сюжету. Зважаючи на це, постає задача розробки спеціалізованого модуля віртуального ігрового столу у складі єдиної вебплатформи, який вирішить проблему візуалізації та забезпечить зручний інструментарій для взаємодії з ігровими об'єктами у режимі реального часу.

У межах розробки модуля необхідно вирішити комплекс функціональних завдань, першим з них є візуальне керування ігровим простором та маніпулювання тактичними картами. Це передбачає розробку оптимізованого

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

механізму завантаження та масштабування графічних файлів у браузері. Для забезпечення гнучкості відображення потрібна реалізація системи шарів: базового фоновому шару для карти, шару токенів для розміщення персонажів, шару для вільного малювання та окремого шару для відображення ігрової сітки. Інтерактивна ігрова сітка, у поєднанні з інструментом віртуальної лінійки, має дозволяти системі та гравцям точно вимірювати ігрові відстані і коректно позиціювати об'єкти з прив'язкою до координат поля.

Наступним завданням є автоматизація базових ігрових механік для зменшення когнітивного навантаження на учасників. Необхідно розробити підсистему віртуальних тривимірних кидків кубиків із підтримкою парсингу математичних формул, що автоматизує застосування ігрових модифікаторів до результатів.

Окрім цього, розробка має включати реалізацію інтерактивних листів персонажів та істот для керування інвентарем і навичками, а також вбудований трекер ініціативи для зручного відстеження черговості ходів у бою. Усі ігрові події мають фіксуватися шляхом ведення автоматизованого журналу, доступного для перегляду всім учасникам поточної сесії.

Критичним технічним завданням є двостороння синхронізація дій усіх учасників. Для імітації фізичної настільної гри потрібне впровадження механізму вільного захоплення та перетягування ігрових фішок на віртуальній карті.

Оскільки процес передбачає одночасну участь групи користувачів, система має забезпечувати безперервну та миттєву синхронізацію просторових позицій токенів, малюнків на карті та результатів кидків кубиків для всіх підключених клієнтів.

Для глибшого розуміння специфіки розробки візуального модуля необхідно детальніше розкрити саму суть настільних рольових ігор (НРІ). Настільна рольова гра - це форма спільного наративного дозвілля, де учасники словесно описують дії своїх персонажів у вигаданому світі. На відміну від класичних відеоігор із жорстко заданими алгоритмами, головним рушієм

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

ігрового процесу в НРІ є людська уява. Гравці не просто переміщують фішки по дошці, а приймають абсолютно нестандартні рішення в унікальних ситуаціях, які виникають під час розвитку сюжету.

Центральною фігурою будь-якої НРІ є Майстер гри (Game Master). Він виконує роль головного оповідача, архітектора світу та неупередженого арбітра. Майстер керує всіма неігровими персонажами (NPC), монстрами, описує навколишнє середовище та проєктує бойові зіткнення на тактичній карті. Саме він встановлює правила гри та визначає наслідки дій героїв.

Кожен гравець, у свою чергу, керує лише своїм протагоністом. Хід гравця базується на словесній заявці про намір: він описує, що хоче зробити (наприклад, перестрибнути прірву чи атакувати ворога). Для визначення успішності цієї дії використовуються кидки багатогранних гральних кубиків (зокрема, d20). До результату на гранях додаються математичні модифікатори з листа характеристик персонажа. Майстер аналізує цю суму і описує фінальний результат дії.

Ключовою особливістю такого процесу є його абсолютна гнучкість, оскільки гравці можуть вигадати нескінченну кількість варіантів вирішення однієї проблеми. Через цю непередбачуваність спроба повної програмної автоматизації всіх ігрових механік у віртуальному столі є надзвичайно складною і, здебільшого, недоцільною задачею. Жорстка автоматизація правил перетворить НРІ на звичайну відеогру, позбавивши її свободи вибору. Тому віртуальний ігровий стіл має проєктуватися як гнучка «цифрова пісочниця». Його мета - надати зручний інструментарій (інтерактивні карти, токени, 3D-кубики), який візуалізує простір та прискорює рутинні підрахунки, залишаючи при цьому повний контроль над сюжетом і правилами виключно в руках гравців та Майстра.

Результатом вирішення цієї задачі стане створення інтерактивного середовища вебдодатка, що гарантує стабільний ігровий процес та покращує рівень залученості гравців у спільну історію.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2 Аналіз аналогів програмного продукту

Ринок цифрових інструментів для настільних рольових ігор характеризується високою різноманітністю та постійним розвитком. Існуючі рішення зазвичай мають різний фокус: від комплексних платформ з глибокою автоматизацією правил до мінімалістичних онлайн-дошок, призначених виключно для малювання та переміщення фішок. Для обґрунтування доцільності розробки власного модуля віртуального столу було проведено детальний аналіз трьох найбільш популярних рішень у цій ніші: Roll20 [29], Foundry VTT [30] та Owlbear Rodeo [31]. Аналіз проводився з фокусом на можливості візуалізації ігрового простору, оптимізацію роботи з графічними ресурсами, зручність маніпуляції об'єктами та загальну продуктивність системи під час багатокористувацької взаємодії.

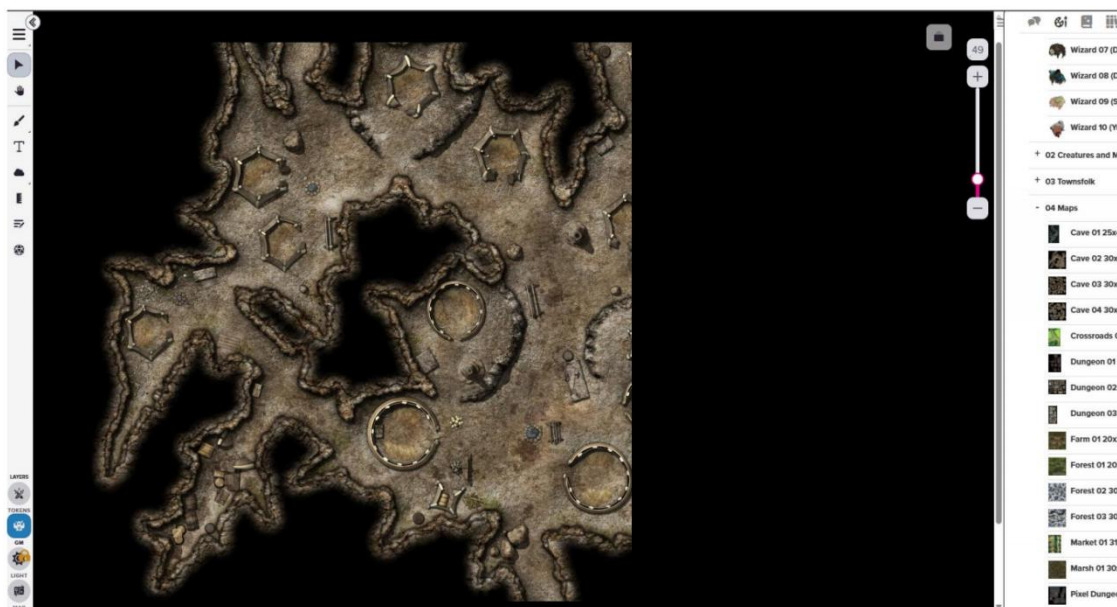


Рис. 1.1. Інтерфейс віртуального столу Roll20

Платформа Roll20 є найпопулярнішою та однією з найстаріших вебплатформ для проведення настільних рольових ігор. Її головною особливістю є те, що вона працює безпосередньо у веббраузері та не вимагає встановлення додаткового програмного забезпечення чи налаштування локальних серверів. Основний функціонал віртуального столу Roll20 зосереджений на двовимірному ігровому просторі, де майстер гри може розміщувати карти, налаштовувати сітку та керувати токенами персонажів.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Платформа пропонує вбудовану систему кидання кубиків та інтегровані інтерактивні листи персонажів для популярних ігрових систем (рис. 1.1).

Проте, незважаючи на широку базу користувачів, архітектура Roll20 має суттєві недоліки. Користувачі часто стикаються з низькою продуктивністю та затримками під час завантаження великих графічних файлів або карт із значною кількістю об'єктів. Це пов'язано із застарілими підходами до рендерингу графіки в браузері. Крім того, інтерфейс платформи вважається перевантаженим і неінтуїтивним для новачків, а доступ до розширеного функціоналу візуалізації вимагає оформлення платної щомісячної підписки.

Foundry VTT є іншим класом віртуальних столів - це потужний програмний комплекс, який вимагає одноразового придбання ліцензії та самостійного хостингу сервера майстром гри. Гравці при цьому підключаються до створеної гри через звичайний веббраузер. Ключовою перевагою Foundry VTT є використання сучасних технологій апаратного прискорення графіки, що забезпечує надзвичайно високу продуктивність рендерингу навіть при відображенні складних карт із великою кількістю динамічних елементів та анімацій (рис. 1.2).



Рис. 1.2. Інтерфейс Foundry VTT

Платформа має глибоку систему автоматизації ігрових правил, дозволяючи налаштовувати складні математичні формули для кидків кубиків та створювати власні макроси. Однак, такий потужний функціонал супроводжується дуже високим порогом входження. Необхідність самостійного

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

налаштування мережі, наприклад, конфігурації портів роутера для забезпечення підключення гравців, або орендувати спеціалізований хмарний сервер що робить цей продукт занадто складним для пересічних користувачів. Також відсутність безкоштовної базової версії обмежує можливості ознайомлення з платформою для нових ігрових груп.

Owlbear Rodeo є яскравим прикладом мінімалістичного підходу до проєктування віртуальних ігрових столів. На відміну від попередніх аналогів, цей вебзастосунок фокусується виключно на базових потребах гри: відображенні ігрової карти та переміщенні токенів у двовимірному просторі (рис. 1.3).



Рис. 1.3. Ігровий простір Owlbear Rodeo

Сервіс працює у браузері, не вимагає реєстрації акаунтів для гравців і запускається миттєво. Завдяки відмові від складних систем автоматизації та інтеграції ігрових правил, розробникам вдалося досягти високої швидкості роботи та стабільності навіть на застарілих або мобільних пристроях.

Інтерфейс Owlbear Rodeo є максимально чистим та зрозумілим. Проте такий мінімалізм є водночас і найбільшим недоліком платформи для довготривалих ігор. Відсутність вбудованих листів персонажів, складних математичних парсерів для тривимірних кидків кубиків та автоматизованих

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

журналів подій змушує гравців паралельно використовувати інші інструменти для ведення своєї ігрової статистики, що руйнує цілісність ігрового процесу.

Для наочного зіставлення ключових технічних та функціональних характеристик розглянутих платформ було сформовано порівняльну таблицю. (табл. 1.1).

Таблиця 1.1.

Порівняльна характеристика розглянутих програмних платформ

Характеристика	Roll20	Foundry VTT	Owlbear Rodeo	(Власна розробка) TTRPG
Хмарний вебдоступ без встановлення ПЗ	+	-	+	+
Використання сучасних технологій (WebGL/WebGPU) для рендерингу графіки	-	+	+	+
Стабільна робота з великою кількістю об'єктів	-	+	+	+
Автоматизація та парсинг кидків кубиків	+	+	-	+
Вбудовані інтерактивні листи персонажів	+	+	-	+
Інструменти автоматизації бойових зіткнень	+	+	-	+
Детальна кастомізація ігрової сітки (Grid)	-	+	-	+
Інтуїтивний інтерфейс та швидкий старт	-	-	+	+
Наявність повністю безкоштовної версії	+	-	+	+

Аналіз даних таблиці 1.1 демонструє, що існуючі аналоги мають чітку спеціалізацію. Roll20 надає базові інструменти, але має застарілий інтерфейс та

проблеми з продуктивністю при великих навантаженнях. Foundry VTT пропонує максимальну потужність та глибоку автоматизацію, але є складним у розгортанні та вимагає платної ліцензії. Owlbear Rodeo вирішує проблему швидкості та доступності, проте повністю позбавлений засобів автоматизації рутини. Це підтверджує актуальність розробки власного модуля віртуального столу (TTRPG), який об'єднає простоту хмарного вебдоступу із сучасною архітектурою рендерингу та необхідним рівнем автоматизації механік без перевантаження інтерфейсу.

1.3 Вибір архітектури системи

Проектування архітектури для системи віртуального ігрового столу вимагає вирішення комплексу специфічних технічних завдань. Насамперед це забезпечення стабільної просторової синхронізації ігрового стану між багатьма клієнтами в режимі реального часу з мінімальною затримкою, а також оптимізація рендерингу важкої графіки безпосередньо у вікні браузера. Класичний підхід на основі монолітної структури з рендерингом сторінок на стороні сервера є абсолютно неприйнятним для таких динамічних застосунків, оскільки постійні перезавантаження вікон руйнуватимуть ігровий процес та перериватимуть мережеві з'єднання. Натомість використання мікросервісної архітектури на початковому етапі розробки продукту вносить надлишкову складність в інфраструктуру та адміністрування, особливо враховуючи необхідність тісної взаємодії між модулем організаційного ядра та підсистемою віртуального столу [3].

З огляду на ці фактори, для розробки платформи було обрано клієнт-серверну архітектуру, побудовану за принципом модульного моноліту (рис. 1.4) та організовану в межах єдиного репозиторію. Це рішення є оптимальним компромісом, який дозволяє ефективно розподілити завдання під час розробки між організаційним контуром та ігровим модулем, зберігаючи при цьому єдину точку розгортання та спільну базу даних.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

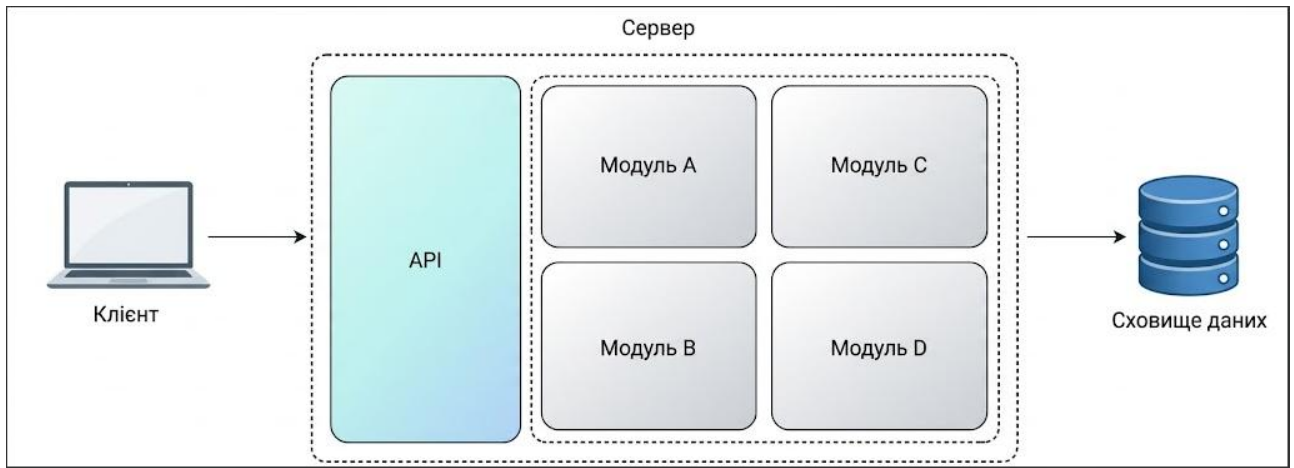


Рис. 1.4. Схема архітектури модульного моноліту

Логічна структура серверної частини базується на багат шаровому підході та концепціях DDD (рис. 1.5), що дозволяє чітко розмежувати відповідальність між обробкою стандартних мережевих запитів та бізнес-логікою ігрової синхронізації [1, 2].

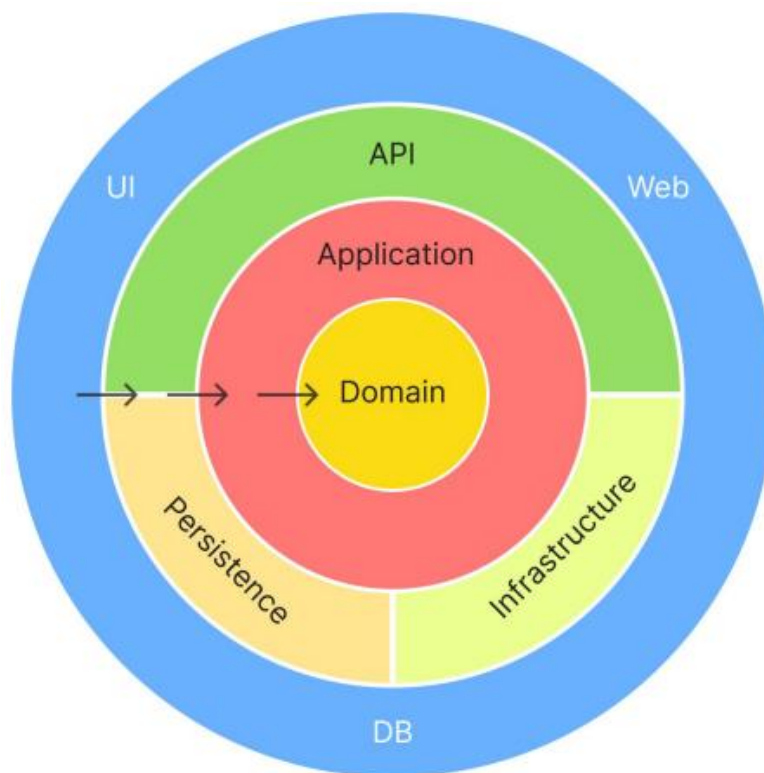


Рис. 1.5. Схематичне зображення шарів

Клієнтська частина системи проєктується як односторінковий вебзастосунок (SPA) [32]. Такий архітектурний підхід гарантує безшовну навігацію та миттєвий відгук інтерфейсу на дії користувача. Оскільки віртуальний стіл вимагає високого ступеня інтерактивності для маніпуляції

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

фішками та роботи з ігровими інструментами, архітектура SPA дозволяє завантажити основний каркас додатку лише один раз. Після початкового завантаження застосунк динамічно оновлює лише ті фрагменти інтерфейсу, де відбулися зміни, що кардинально знижує обсяг переданих даних між клієнтом і сервером під час активної ігрової сесії.

Ключовим архітектурним рішенням на стороні клієнта є чітке розділення відповідальності між стандартним інтерфейсом користувача та графічним рушієм для візуалізації ігрової карти. Для відмальовування самої ігрової дошки, фонових зображень та переміщення tokenів використовується спеціалізований підхід на базі технологій Canvas та WebGL. Виділення графічного рушія в окремий архітектурний шар переносить основне обчислювальне навантаження з рендерингу візуальних елементів на апаратні потужності клієнтського пристрою. Це рішення не лише забезпечує стабільну частоту кадрів для плавної роботи під час гри, але й суттєво мінімізує навантаження на центральний сервер платформи, який повністю звільняється від необхідності обробки графічної інформації [9].

АРХІТЕКТУРА СИСТЕМИ (SYSTEM ARCHITECTURE)

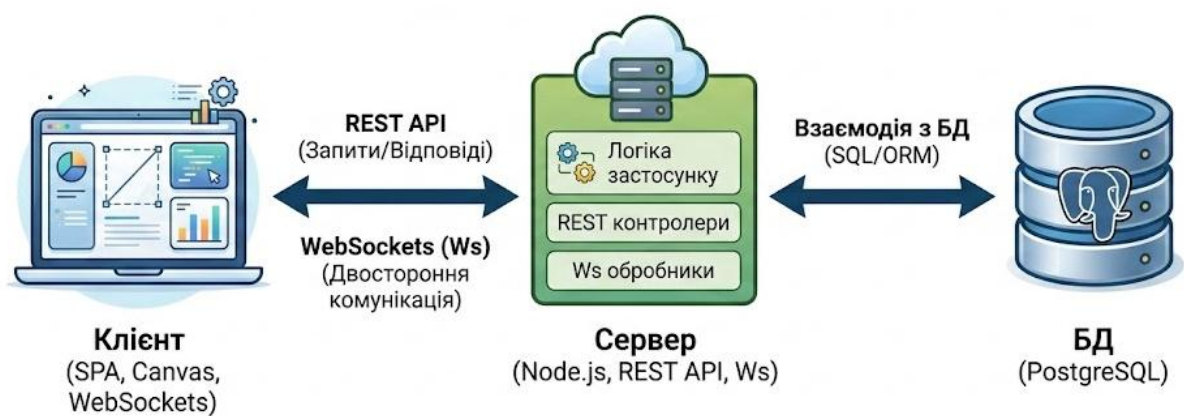


Рис. 1.6. Загальна архітектура розробленої платформи

Рівень доступу до даних реалізує взаємодію з єдиною реляційною базою даних PostgreSQL. Для збереження персистентної інформації, такої як конфігурації ігрових світів та параметри персонажів, використовується

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

класичне об'єктно-реляційне відображення, що гарантує надійність транзакцій та цілісність інформації. Водночас динамічний стан ігрової дошки під час активних ігрових фаз обробляється за допомогою швидкісних з'єднань, що мінімізує затримки під час інтенсивної взаємодії гравців. Узагальнену модель взаємодії між цими ключовими рівнями - клієнтським односторінковим застосунком, сервером та базою даних - наведено на рисунку 1.6.

Таким чином, обрана гібридна клієнт-серверна архітектура з використанням модульного моноліту, підходу односторінкового застосунку та апаратно-прискореного графічного рендерингу формує надійний технологічний фундамент для створення сучасного модуля віртуального ігрового столу.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір програмних засобів для реалізації модуля віртуального ігрового столу безпосередньо впливає з архітектурних рішень, прийнятих у попередньому підрозділі. Враховуючи, що ядром підсистеми є високонавантажений графічний інтерфейс та механізми синхронізації в реальному часі, основними критеріями вибору інструментарію є продуктивність рендерингу, забезпечення стабільного двостороннього зв'язку та наявність розвиненої екосистеми для масштабування.

Таблиця 1.2.

Порівняльна характеристика технологій рендерингу вебграфіки

Характеристика	Стандартний DOM / CSS	HTML5 Canvas 2D	WebGL / WebGPU
Апаратне прискорення (GPU)	Часткове (силами браузера)	Базове	Повне (прямий доступ)
Продуктивність (тис. об'єктів)	Низька	Середня	Дуже висока
Накладні витрати пам'яті	Високі	Середні	Низькі
Застосування у платформі	Інтерфейс користувача (UI)	Обробка зображень та резервний рендеринг	Ігрова карта (VTT)

Для обґрунтування вибору WebGL як основи для візуалізації ігрової дошки було проведено порівняння сучасних підходів до рендерингу вебграфіки (табл. 1.2).

Для серверної частини архітектури обрано подієво-орієнтовану неблокуючу модель виконання, що є критично важливим критерієм для управління великою кількістю одночасних підключень у багатокористувацьких застосунках. Оскільки ігровий модуль спеціалізується на обробці подій у реальному часі, архітектура вимагає наявності швидкісної підсистеми передачі даних. Основним інструментом для двосторонньої взаємодії між віртуальним столом та сервером обрано протокол повнодуплексного зв'язку WebSocket. На відміну від традиційних HTTP-запитів, які створюють значні додаткові витрати на встановлення з'єднання при кожній дії, відкритий канал зв'язку гарантує миттєву трансляцію координат переміщених токенів та результатів кидків кубиків усім учасникам конкретної ігрової кімнати.

Вибір WebSocket обґрунтовується критичними вимогами до затримки (latency) під час гри. Детальне порівняння протоколів реального часу наведено в таблиці 1.3.

Таблиця 1.3.

Порівняння протоколів передачі даних у реальному часі

Характеристика	HTTP Polling (Опитування)	Server-Sent Events (SSE)	WebSocket
Напрямок зв'язку	Від клієнта до сервера	Від сервера до клієнта	Повноцінний двосторонній
Накладні витрати на пакет	Дуже високі (заголовки HTTP)	Середні	Мінімальні (байт-фрейми)
Затримка (Latency)	Висока (залежить від таймера)	Низька	Мінімальна (реальний час)
Постійне з'єднання	Ні	Так	Так
Застосування у платформі	Не використовується	Системні сповіщення	Синхронізація візуалізації

Для реалізації базового клієнтського інтерфейсу користувача було обрано бібліотеку React. Її використання обумовлене компонентно-орієнтованим підходом, що ідеально підходить для створення складних та інтерактивних панелей інструментів, таких як меню керування токенами, трекер ініціативи чи інтерактивні листи персонажів [4]. Використання механізму Virtual DOM дозволяє React ефективно оновлювати лише необхідні елементи (наприклад, показники здоров'я персонажа) без перемальовування всієї сторінки, що критично важливо під час гри. Візуальна стилізація цих DOM-елементів виконується за допомогою утилітарного CSS-фреймворку Tailwind CSS [25], що дозволяє досягти єдиної дизайн-системи. Інструментом збирання проєкту обрано Vite [26], який завдяки механізму "гарячої заміни модулів" значно пришвидшує процес розробки та тестування інтерфейсних рішень.

Для ефективного управління складним локальним станом віртуального столу обрано бібліотеку Zustand. Враховуючи величезну кількість динамічних параметрів (активний інструмент малювання, обраний шар карти, виділені токени), Zustand дозволяє створити легковагові та масштабовані сховища даних без надмірного ускладнення архітектури коду. Використання цієї бібліотеки гарантує швидку реакцію інтерфейсу на дії користувача без необхідності передавати параметри через безліч компонентів [5].

Окрім локального стану, платформа активно працює із серверними даними (наприклад, завантаження списку доступних ігрових сцен, загальних налаштувань кампанії або профілів гравців). Для ефективного отримання, кешування та синхронізації цих даних із сервером застосовано бібліотеку TanStack React Query. На відміну від традиційних методів завантаження даних, React Query бере на себе управління життєвим циклом серверного стану. Вона автоматично обробляє стани завантаження та помилок, виконує фонове оновлення застарілих даних і значно зменшує кількість надлишкових запитів до API завдяки продуманому механізму кешування. Це звільняє Zustand від необхідності зберігати серверні дані, залишаючи йому виключно швидкісне керування локальним UI-станом [6].

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

Центральним інструментом для розробки візуальної частини віртуального столу було обрано графічну бібліотеку Pixi.js. На відміну від стандартних засобів маніпуляції DOM-деревом, які є занадто повільними для ігор, Pixi.js є спеціалізованим високопродуктивним рушієм для створення інтерактивної 2D-графіки. Він автоматично використовує можливості сучасних API WebGPU та WebGL, надаючи розробнику зручний абстрактний шар у вигляді "графа сцени" (Scene Graph) [7]. Вибір Pixi.js дозволяє системі ефективно відмальовувати багат шарові ігрові карти високої роздільної здатності, реалізовувати інструменти малювання (Drawing Layer) та забезпечувати плавну анімацію сотень ігрових об'єктів шляхом пакетної обробки (batching) графічних викликів до відеокарти.

Для реалізації підсистеми віртуальних кидків кубиків інтегровано спеціалізовану екосистему на базі бібліотеки @3d-dice/dice-box-threejs [33]. Цей інструмент поєднує 3D-рушій Three.js для візуалізації кубиків та фізичний рушій для розрахунку їх зіткнень. На відміну від простого генерування випадкових чисел, використання фізичної 3D-симуляції створює для гравців ефект присутності за реальним столом. Бібліотека також містить вбудований парсер, який здатний обробляти складні текстові ігрові формули (наприклад, "2d6+3") та автоматично переводити їх у фізичні моделі на екрані [8].

Серверна частина платформи базується на використанні середовища виконання Node.js та фреймворку Express [11]. Подієво-орієнтована неблокуюча модель обробки введення-виведення Node.js є ідеальною для додатків, які вимагають утримання великої кількості відкритих з'єднань одночасно, що є типовим для VTT [10].

Для забезпечення миттєвої синхронізації ігрового процесу використовується технологія WebSocket. Її реалізацію виконано за допомогою легкої бібліотеки ws для середовища Node.js. Вона дозволяє створити повноцінний дуплексний канал, через який координати перетягнутих токенів, нові лінії на карті та результати кидків кубиків передаються всім підключеним клієнтам у текстовому або бінарному форматах із мінімально можливими

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

затримками. На відміну від масивних рішень на кшталт Socket.io, бібліотека ws забезпечує максимальну продуктивність та значно менші накладні витрати оперативної пам'яті сервера [12].

Як систему керування базами даних для збереження персистентного стану (збереження позицій токенів після завершення сесії, налаштування сцен) використано реляційна СУБД PostgreSQL у зв'язці з інструментом Prisma ORM. Для забезпечення стабільності розгортання та ізоляції середовища виконання всіх компонентів системи обрано технологію контейнеризації Docker та інструмент оркестрації Docker Compose [19].

Вимоги до апаратного та програмного забезпечення для стабільного функціонування розробленого модуля наведено у таблиці 1.4.

Таблиця 1.4.

Вимоги до апаратно-програмного забезпечення

Тип забезпечення	Складова частина	Мінімальні вимоги
Програмне	Операційна система (Сервер)	Linux (Ubuntu 22.04 LTS або аналогічна)
	Операційна система (Клієнт)	Windows 10/11, macOS, Linux
	Середовище розгортання	Docker Engine 24.0+, Docker Compose 2.0+
	Веббраузер клієнта	Google Chrome, Mozilla Firefox, Safari (з підтримкою WebGL)

Апаратне	Процесор (Сервер)	2-ядерний процесор з частотою від 2.4 ГГц
	Оперативна пам'ять (Сервер)	Не менше 2 ГБ ОЗП
	Накопичувач (Сервер)	40 ГБ SSD
	Відеоадаптер (Клієнт)	Інтегрована або дискретна відеокарта з підтримкою апаратного прискорення графіки
	Оперативна пам'ять (Клієнт)	4 ГБ ОЗП і вище
	Мережеве обладнання	Доступ до мережі Інтернет на швидкості від 10 Мбіт/с

Наведена конфігурація є базовою для забезпечення комфортного ігрового процесу. Основне навантаження при роботі з модулем віртуального столу лягає на клієнтський пристрій (зокрема на відеоадаптер та оперативну пам'ять) через необхідність постійного рендерингу графіки за допомогою `Pixi.js` та `Three.js`, у той час як серверні потужності масштабуються залежно від кількості одночасно активних ігрових кімнат та інтенсивності обміну WebSocket-повідомленнями.

Висновки до першого розділу

У першому розділі кваліфікаційної роботи проведено комплексний аналіз предметної області та обґрунтовано технологічний фундамент для створення модуля віртуального ігрового столу (VTT). Виконане дослідження дозволило чітко окреслити проблематику дистанційного проведення настільних рольових

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

ігор, визначити обмеження існуючих рішень та сформувати архітектурне бачення проєктуваної вебплатформи.

Аналіз предметної області виявив, що перехід ігрових спільнот в онлайн-простір супроводжується втратою просторової наочності. Тому визначено ключові завдання розробки: необхідність створення інтерактивної ігрової дошки, автоматизації розрахунків кидків кубиків та реалізації безперервної синхронізації дій усіх учасників сесії.

Під час порівняльного аналізу комерційних аналогів (Roll20, Foundry VTT, Owlbear Rodeo) було виявлено, що існуючі рішення пропонують або перевантажений функціонал із платною ліцензією, або надто мінімалістичний підхід. Це довело доцільність розробки власного модуля, який об'єднає переваги безкоштовного хмарного вебдоступу з високою продуктивністю рендерингу.

Для досягнення поставлених завдань обґрунтовано вибір гібридної клієнт-серверної архітектури за шаблоном модульного моноліту. Доведено неприйнятність традиційного серверного рендерингу для динамічних VTT-додатків, що зумовило вибір підходу односторінкового застосунку (SPA) на стороні клієнта із розділенням відповідальності між стандартним DOM-інтерфейсом та спеціалізованим графічним рушієм.

На основі спроектованої архітектури здійснено остаточний вибір інструментальних засобів розробки. Для забезпечення інтерактивності клієнтської частини обрано екосистему React зі стейт-менеджером Zustand, а для рендерингу 2D/3D графіки - апаратно-прискорені бібліотеки Pixi.js та Three.js. Серверну логіку вирішено будувати у середовищі Node.js із використанням протоколу WebSocket для мінімізації мережових затримок та бази даних PostgreSQL для збереження прогресу.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ НАСТІЛЬНО РОЛЬОВИХ ІГОР: МОДУЛЬ ВІРТУАЛЬНИЙ ІГРОВИЙ СТІЛ

2.1 Визначення вимог до системи

Основою для успішного проєктування будь-якого програмного комплексу є детальний аналіз вимог до майбутнього продукту, що дозволяє визначити його функціональні та нефункціональні характеристики з урахуванням потреб користувачів різних типів. У контексті розробки модуля віртуального ігрового столу, головним завданням є створення багатофункціонального середовища для графічної візуалізації ігрового світу та маніпуляції його елементами в режимі реального часу. Відповідно до цього, проєктована система повинна підтримувати складну логіку керування ігровими сценами, токенами та математичними обчисленнями.

Задля формування ядра програмного продукту та опису конкретних дій, які користувачі зможуть виконувати в межах платформи, сформулюємо функціональні вимоги до візуального модуля. Вони охоплюють весь цикл взаємодії з ігровою дошкою: від завантаження карт до симуляції кидків кубиків. Деталізований перелік основних функціональних вимог наведено у таблиці 2.1.

Таблиця 2.1.

Функціональні вимоги до візуального модуля системи

Назва вимоги	Опис вимоги	Пріоритет
Управління ігровими сценами	Надання інструментів для створення сцен, завантаження фонових карт, керування їх списком (Scenes Browser) та перемикання активної сцени для всієї ігрової групи.	Високий
Інтерактивні листи персонажів та істот	Можливість створення та редагування динамічних листів персонажів (Character Sheets) і монстрів (Creatures) з відображенням їхніх характеристик, інвентарю та автоматичним розрахунком модифікаторів навичок.	Високий

Автоматизація кидків кубиків	Підтримка 3D-візуалізації кидків (DiceRoller 3D), парсинг складних текстових ігрових формул, ведення загального журналу результатів (Dice Log) та механізм швидких кидків (Quick Roll).	Високий
Маніпуляція токенами та сіткою	Механізм вільного перетягування ігрових фішок по карті з прив'язкою до ігрової сітки (Grid) та автоматичною синхронізацією координат.	Високий
Трекер бойової ініціативи	Вбудований інструмент для автоматизованого відстеження черговості ходів у бою, управління станом здоров'я учасників та переходу між раундами.	Середній
Інструменти малювання та вимірювання	Надання можливості нанесення вільних ліній, геометричних фігур на окремий шар карти (Drawing Layer) та використання віртуальної лінійки для точного розрахунку відстаней.	Середній
Інтегровані засоби комунікації	Забезпечення синхронного обміну текстовими повідомленнями (Floating Chat) та підтримка аудіо-/відеозв'язку (Floating Call) безпосередньо поверх інтерфейсу ігрової карти.	Високий

Окрім функціональних можливостей, критично важливим аспектом є дотримання нефункціональних вимог. Оскільки візуальний модуль відмальовує графіку в реальному часі за допомогою WebGL, система повинна забезпечувати стабільну частоту кадрів для плавної анімації об'єктів. Архітектура також вимагає мінімальної мережевої затримки (latency) під час трансляції подій через WebSocket, щоб унеможливити ефект "розсинхронізації" станів між клієнтами. Крім того, архітектура повинна бути гнучкою, що дозволить у майбутньому додавати нові графічні шари чи механіки без переробки ядра (наприклад, динамічне освітлення).

Для коректної роботи описаного функціоналу необхідно детально розглянути акторів системи. У межах модуля віртуального столу глобальна рольова модель платформи звужується до двох специфічних сутностей: Гравець

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

та Майстер гри (GM). Чітке розмежування їхніх можливостей є необхідним для безпечного ведення ігрового процесу. Короткий опис акторів та їхніх функцій у межах VTT представлено в таблиці 2.2.

Таблиця 2.2.

Опис акторів візуального модуля та їхніх ключових функцій

Гравець (Player)	Базовий учасник ігрової сесії. Зосереджений на керуванні своїм персонажем. Має доступ до власного інтерактивного листа персонажа, може переміщувати лише свої токени на активній карті, ініціювати кидки кубиків, користуватися лінійкою та базовими інструментами малювання. Не має впливу на загальні налаштування сцени. Також гравець має повний доступ до вбудованих засобів текстового та голосового спілкування з іншими учасниками сесії.
Майстер гри (GM)	Організатор та модератор віртуального простору. Успадковує всі права Гравця, але наділений ексклюзивними привілеями: створення і керування всіма ігровими сценами (картами), зміна налаштувань сітки, створення великої кількості токенів і монстрів (GM Creatures), повне управління трекером ініціативи та керування об'єктами інших гравців.

На рисунку 2.1 наведено діаграму варіантів використання, яка описує доменну логіку модуля віртуального столу та взаємодію ключових ігрових ролей: Гравця та Майстра гри.

Як видно з діаграми, базовий учасник (Гравець) має набір прецедентів, зосереджених виключно навколо процесу взаємодії зі своїм персонажем та картою. Прецедент «Маніпуляція токенами» дозволяє гравцеві переміщувати власну фішку, а «Здійснення кидків кубиків» автоматично ініціює запис у загальний ігровий журнал через зв'язок <<include>>. Використання лінійки та засобів вільного малювання є допоміжними інструментами комунікації.

Майстер гри успадковує всі прецеденти Гравця, але його функціонал значно розширено адміністративними привілеями. Окремим блоком виділено управління сценами, що включає створення нових карт та перемикання між ними. Найважливішим прецедентом Майстра є «Управління ініціативою», який виступає оркестратором бойових зіткнень, дозволяючи керувати ходами всіх учасників та змінювати їхні стани.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

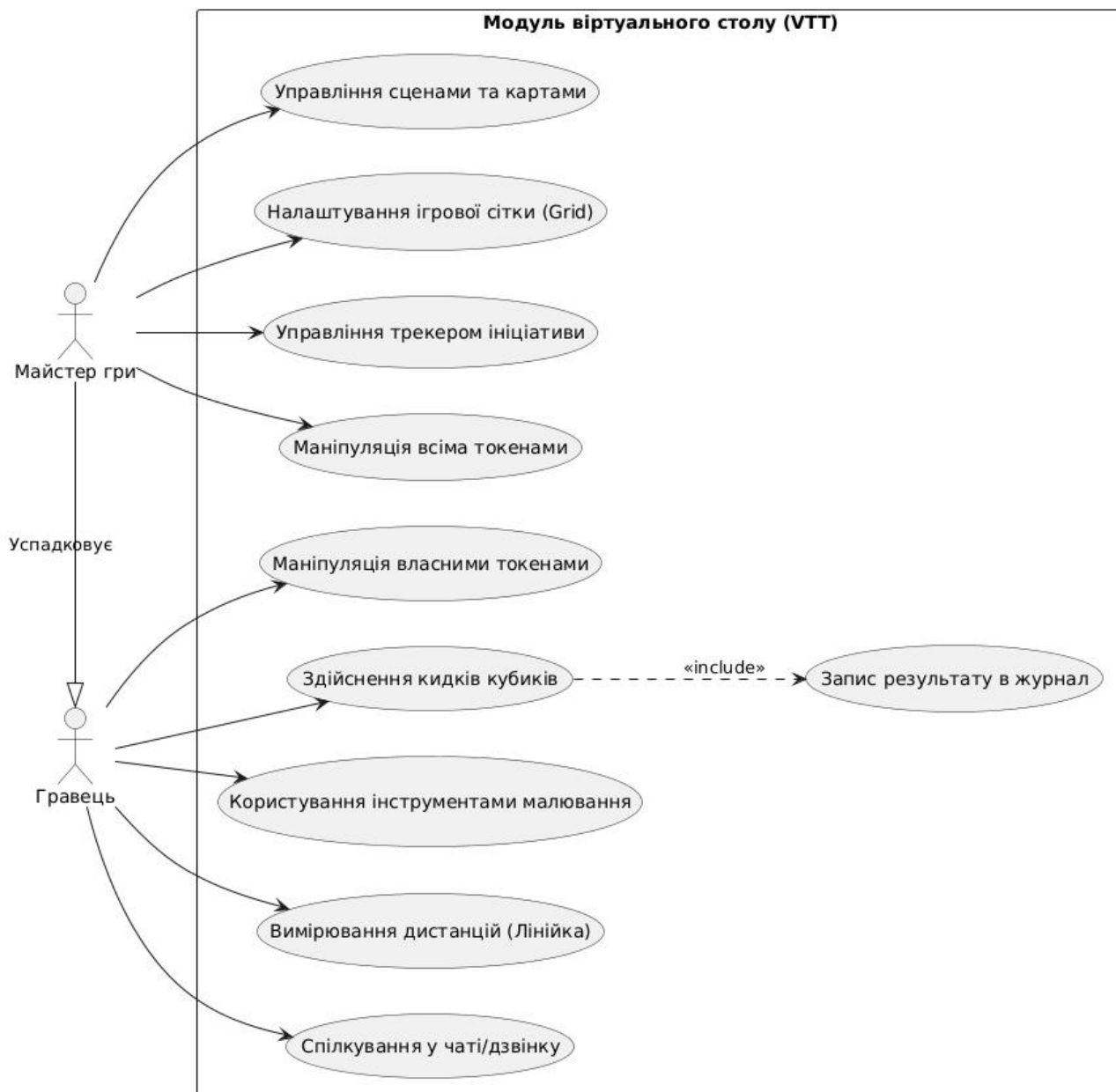


Рис. 2.1. Діаграма сценаріїв використання

Після визначення функціональних вимог та сценаріїв використання візуального модуля, наступним етапом логічного проектування є розробка об'єктно-орієнтованої моделі. Для візуалізації статичної структури системи, її базових ігрових сутностей, їхніх атрибутів, методів та взаємозв'язків між ними побудовано UML-діаграму класів (Class Diagram). Ця модель є перехідною ланкою між клієнтським інтерфейсом віртуального столу та структурою даних, оскільки вона не лише структурує інформацію, а й інкапсулює бізнес-логіку рендерингу та взаємодії через відповідні методи.

Спроектована об'єктно-орієнтована структура забезпечує цілісність даних та дозволяє візуальному рушію (Pixi.js) ефективно відмальовувати кожен об'єкт відповідно до його стану. На рисунку 2.2 наведено діаграму класів доменної області віртуального столу.

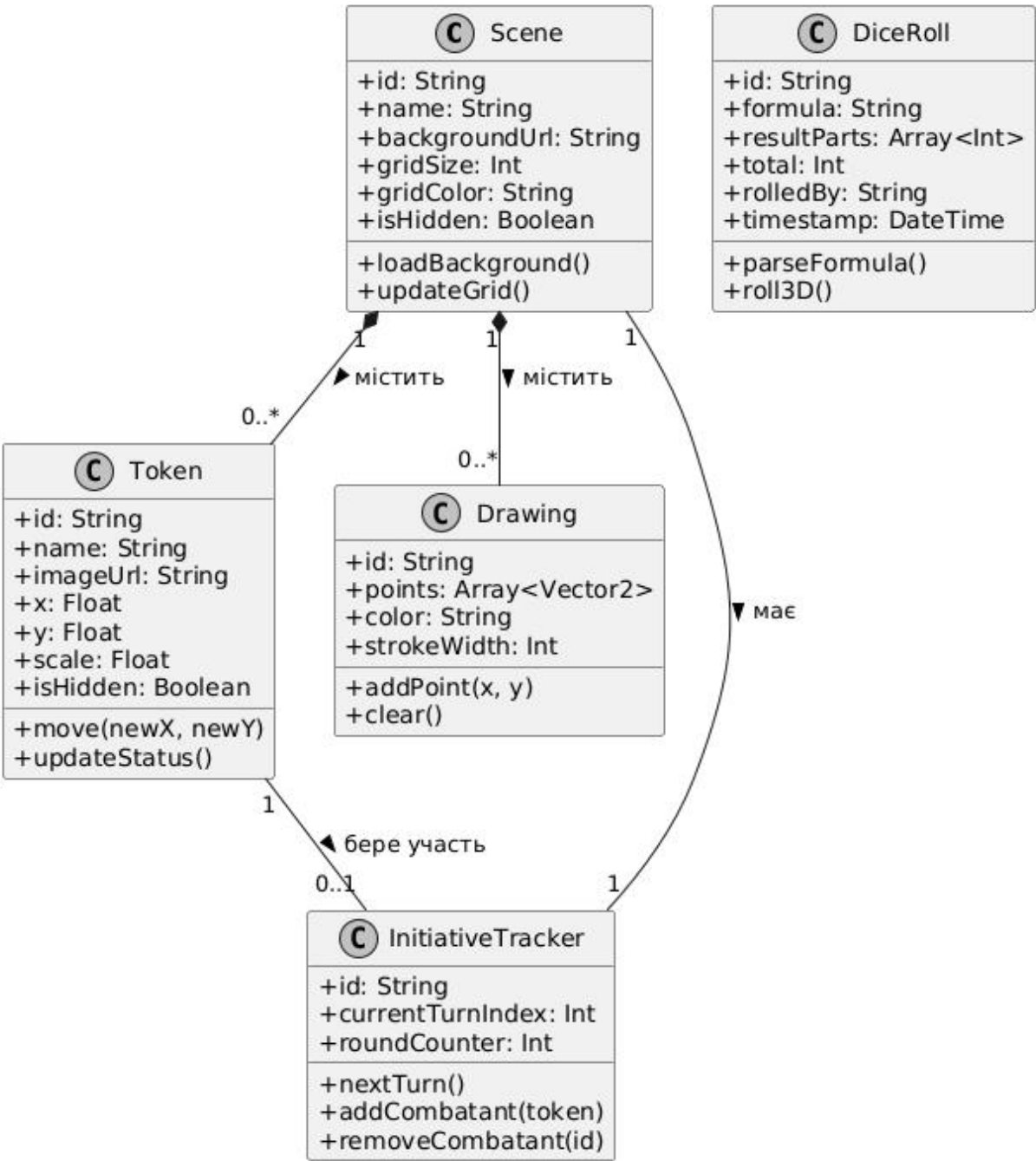


Рис. 2.2. Діаграма класів доменної області віртуального столу

Як видно з діаграми класів, архітектура візуального модуля базується на чіткій композиційній структурі, де ігрова сцена виступає головним контейнером для всіх просторових об'єктів. Відношення агрегації та композиції між сутностями гарантують, що при видаленні або зміні сцени всі пов'язані з

нею токени, малюнки та налаштування ініціативи будуть коректно опрацьовані графічним рушієм. Така об'єктно-орієнтована декомпозиція не лише оптимізує управління клієнтським станом у React-компонентах, але й створює чітку проєкцію для розробки реляційної схеми бази даних, яка детально розглядається у наступному підрозділі.

Архітектура візуального модуля платформи побудована навколо кількох ключових доменних сутностей (класів), що формують ігровий простір:

Клас Scene (Ігрова сцена): Є базовою сутністю віртуального столу, яка відображає конкретну локацію або карту. Вона містить параметри фонового зображення (backgroundUrl), налаштування сітки (gridSize, gridColor) та стан видимості для гравців (isHidden). Клас керує завантаженням графічних ресурсів та має композитний зв'язок (містить у собі) з токенами та малюнками.

Клас Token (Ігрова фішка): Відображає динамічний об'єкт на карті (персонажа, монстра або предмет). Зберігає свої просторові координати (x, y), масштаб (scale) та візуальне представлення (imageUrl). Ключовим методом класу є move(x, y), який активується під час drag-and-drop переміщення і запускає трансляцію нових координат через WebSocket.

Клас Drawing (Елемент малювання): Інкапсулює дані про вільні лінії та фігури, нанесені гравцями поверх карти. Зберігає масив точок (points), колір (color) та товщину пензля (strokeWidth).

Клас InitiativeTracker (Трекер ініціативи): Сервісний клас для управління бойовими зіткненнями. Відстежує поточний раунд (roundCounter), черговість ходів (currentTurnIndex) та має методи для передачі ходу (nextTurn()). Цей клас має агрегаційний зв'язок із класом Token, оскільки керує їхньою бойовою поведінкою.

Клас DiceRoll (Кидок кубика): Сутність, що інкапсулює логіку підсистеми 3D-кубиків. Зберігає вихідну текстову формулу (formula), масив результатів окремих кубиків (resultParts) та підсумкове значення (total). Метод roll3D() відповідає за ініціалізацію фізичної симуляції на клієнті.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

2.2 Розробка бази даних системи

Для збереження персистентного стану віртуального ігрового столу та забезпечення стабільної роботи платформи обрано реляційну систему керування базами даних PostgreSQL [15]. Як прошарок доступу до даних (Data Access Layer) використовується інструмент Prisma ORM [16], що дозволяє декларативно описувати об'єктно-реляційне відображення, гарантує сувору типізацію запитів та автоматизує керування міграціями.

Специфіка візуального модуля вимагає розділення загальних даних платформи на організаційний контур (кампанії, сесії, користувачі) та ігровий контур (сцени, токени, листи персонажів).

Оскільки організаційне ядро було розглянуто в межах загальної архітектури платформи, даний підрозділ зосереджений виключно на структурі даних модуля віртуального столу (VTT).

Загальну структуру даних візуального модуля, а також логіку їх прив'язки до організаційного контуру системи (Campaign та Session), наведено на схемі бази даних (рис. 2.3).

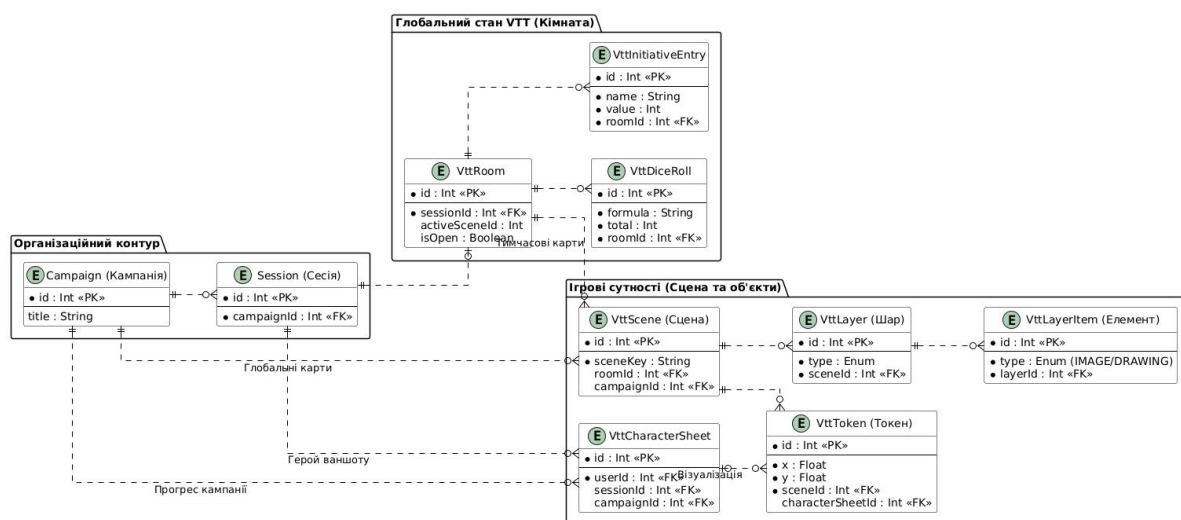


Рис. 2.3. Логічна структура реляційної бази даних модуля віртуального ігрового столу

Як демонструє наведена ER-діаграма, розроблена структура бази даних забезпечує надійне збереження складної ієрархії об'єктів ігрового простору. Використання гнучкої системи паралельних зовнішніх ключів дозволяє платформі ефективно розмежовувати глобальний багатосесійний прогрес

кампаній та ізольовані дані тимчасових ігор. Спроектowana реляційна модель створює стійкий і масштабований фундамент для управління ігровим станом, що дозволяє перейти до етапу проектування алгоритмів клієнт-серверної взаємодії в реальному часі.

Детальний опис структури бази даних наведено у таблицях 2.3-2.10. Головним контейнером для віртуальної кімнати, що зберігає її глобальний стан, виступає таблиця VttRoom (табл. 2.3). Таблиця ігрових карт із налаштуваннями розмірів, фону та координатної сітки описана сутністю VttScene (табл. 2.4).

Таблиця 2.3.

Опис структури таблиці VttRoom

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор ігрової кімнати
sessionId	Int (FK)	Зовнішній ключ, прив'язка до організаційної сесії
isOpen	Boolean	Статус доступу до віртуального столу
activeSceneId	Int	Ідентифікатор поточної відкритої сцени для гравців
lastSavedAt	DateTime	Час останнього системного автозбереження стану

Таблиця 2.4.

Опис структури таблиці VttScene

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор сцени
sceneKey	String	Клієнтський UUID для in-memory сумісності
campaignId	Int (FK)	Прив'язка до батьківської кампанії (глобальна карта)
roomId	Int (FK)	Прив'язка до ігрової кімнати (локальна карта сесії)
gridSize	Int	Розмір однієї клітинки координатної сітки

Особливої уваги заслуговує архітектурне рішення щодо збереження ігрових сцен та аркушів персонажів. База підтримує гнучку систему відомостей через паралельні зовнішні ключі. Наприклад, сцена може мати зв'язок roomId, що робить її тимчасовою для однієї сесії. Альтернативно, вона може прив'язуватися до кампанії через campaignId, дозволяючи Майстру гри створити глобальну карту, що переноситься в усі наступні ігри. Аналогічний підхід застосовано для збереження прогресу гравців. Динамічні фішки, що зберігають просторові координати та масштаб, описані у VttToken (табл. 2.5). Інтерактивні листи персонажів та істот із повним набором характеристик структурує VttCharacterSheet (табл. 2.6).

Таблиця 2.5.

Опис структури таблиці VttToken

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор фішки
sceneId	Int (FK)	Зовнішній ключ, прив'язка до конкретної сцени
x / y	Float	Точні просторові координати фішки на карті
size / scale	Float	Масштаб відображення токена
characterSheetId	Int (FK)	Опціональний зв'язок із листом персонажа гравця

Таблиця 2.6.

Опис структури таблиці VttCharacterSheet

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор листа персонажа
userId	Int (FK)	Ідентифікатор користувача-власника
campaignId	Int (FK)	Прив'язка до кампанії (збереження прогресу)
sessionId	Int (FK)	Прив'язка до сесії (якщо це персонаж для ваншоту)
stats / skills	Json	Серіалізовані об'єкти з характеристиками та навичками

Журнал збережених результатів фізичних кидків кубиків фіксує сутність VttDiceRoll (табл. 2.7). Ієрархія графічних шарів (малюнки, лінії, маркери) реалізована через таблицю VttLayer (табл. 2.8) та пов'язані з нею елементи.

Таблиця 2.7.

Опис структури таблиці VttDiceRoll

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор запису кидка
formula	String	Текстова математична формула (наприклад "2d20+5")
total	Int	Підсумковий результат кидка після розрахунку фізики
details	Json	Масив із результатами падіння кожного окремого кубика
roomId	Int (FK)	Прив'язка кидка до конкретної ігрової кімнати

Таблиця 2.8.

Опис структури таблиці VttLayer

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор шару карти
layerKey	String	Клієнтський UUID шару для забезпечення in-memory синхронізації
type	Enum (VttLayerType)	Тип шару (Background, Grid, Token, Drawing тощо)
isVisible	Boolean	Прапорець публічної видимості шару для гравців сесії
sceneId	Int (FK)	Зовнішній ключ, жорстка прив'язка шару до конкретної сцени

Структура графічного елемента шару, що зберігає координати для побудови векторних ліній, наведена у VttLayerItem (табл. 2.9). Збереження стану трекера ініціативи для відновлення черги після перепідключення описує VttInitiativeEntry (табл. 2.10).

Таблиця 2.9.

Опис структури таблиці VttLayerItem

Поле	Тип даних	Призначення
id	Int (PK)	Унікальний ідентифікатор графічного елемента шару
type	Enum (VttItemType)	Тип завантаженого елемента (зображення або векторна фігура)
points	Json	Масив математичних координат для побудови векторних ліній
url	String	Посилання на ресурс зображення (для типу IMAGE)
layerId	Int (FK)	Зовнішній ключ, прив'язка елемента до батьківського графічного шару

Таблиця 2.10.

Опис структури таблиці VttInitiativeEntry

Поле	Тип даних	Призначення
entryKey	String	Клієнтський ідентифікатор для оновлення значення у реальному часі
name	String	Відображуване текстове ім'я бойової одиниці (персонажа чи монстра)
value	Int	Математичне значення ініціативи (зазвичай результат кидка 1d20 + модифікатор)
roomId	Int (FK)	Зовнішній ключ, прив'язка списку безпосередньо до ігрової кімнати

Спроектowana реляційна структура повністю відповідає вимогам нормалізації. Для оптимізації високонавантажених вибірок, таких як отримання всіх токенів на активній сцені або завантаження логу кубиків, на рівні бази даних передбачено відповідні індекси за полями sceneId та roomId. Використання типу даних Json для зберігання ігрових характеристик дозволяє

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

зберегти максимальну гнучкість системи до зміни правил гри без необхідності проведення складних міграцій.

2.3 Проектування та реалізація алгоритмів роботи системи

Для розуміння динаміки роботи модуля віртуального столу недостатньо визначити лише його статичну структуру та базу даних. Оскільки система функціонує в режимі реального часу, критично важливим є проектування безпомилкових алгоритмів обміну даними між клієнтом, сервером та графічним рушієм. У даному підрозділі розглянуто ключові поведінкові аспекти системи за допомогою діаграм UML, що описують процеси ініціалізації ігрового простору, просторової синхронізації об'єктів та обробки ігрових механік.

Першим і найважливішим процесом є алгоритм завантаження віртуальної ігрової кімнати та ініціалізація графічного рушія Pixi.js. Цей процес вимагає суворої послідовності дій: клієнтський застосунок повинен спочатку отримати персистентні дані сцени з бази даних, завантажити важкі графічні активи (текстури фону, токени), підключитися до WebSocket-каналу для прослуховування оновлень, і лише після цього розпочати рендеринг графіки.

На рисунку 2.4 наведено діаграму активності (Activity Diagram), яка описує покроковий алгоритм дій системи під час входу користувача в ігрову кімнату.

Як видно з діаграми, процес розгалужується після перевірки наявності активної сцени. Якщо Майстер гри ще не активував жодну карту, система переводить гравців у режим очікування ("default Scene"). Якщо ж сцена існує, система ініціалізує WebGL-контекст та починає паралельне завантаження текстур токенів. Будь-яка помилка на етапі завантаження ресурсів безпечно обробляється через інтерфейс React, що гарантує відмовостійкість застосунку.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

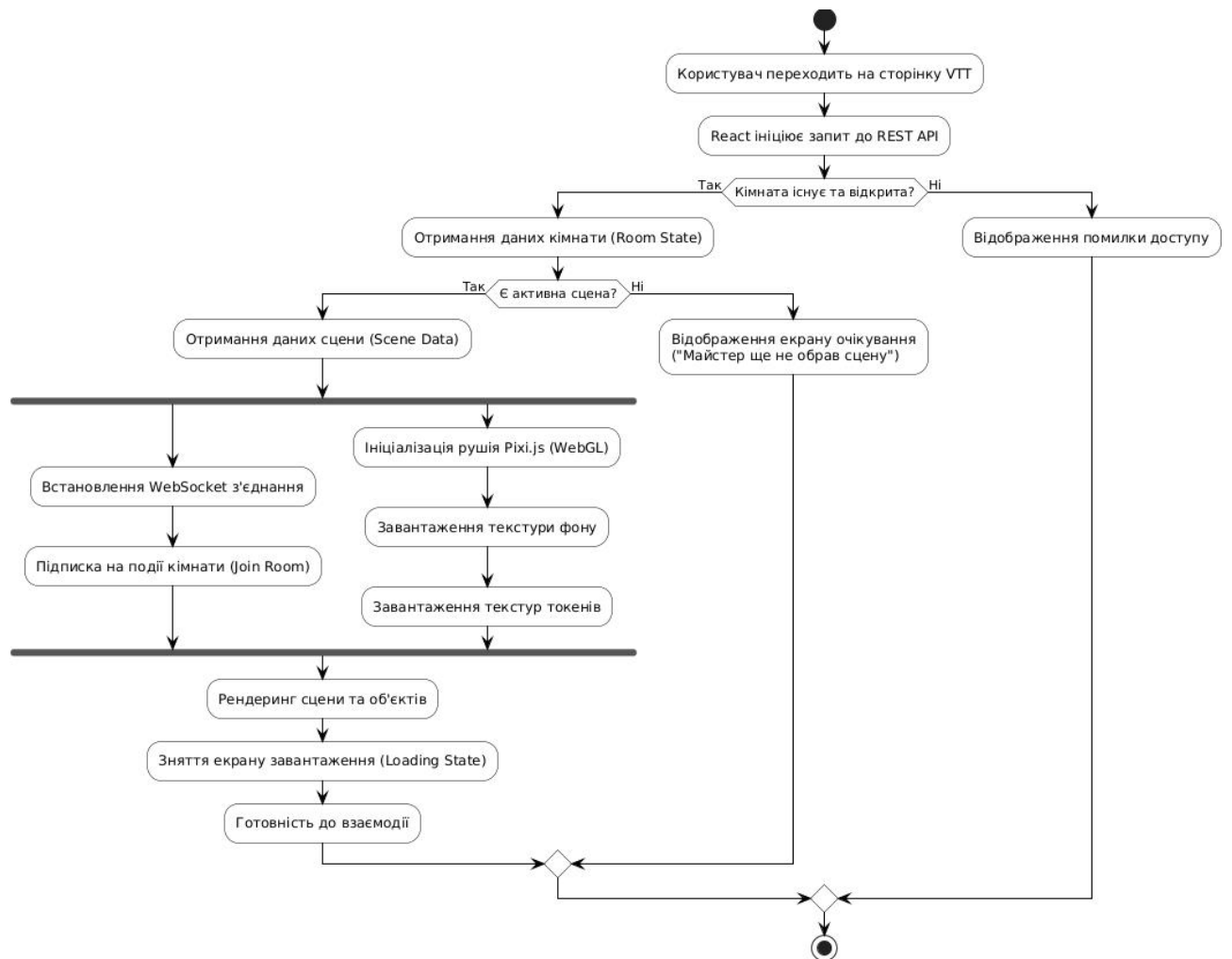


Рис. 2.4. Діаграма активності приєднання гравця

Наступним критично важливим алгоритмом є процес просторової синхронізації ігрових об'єктів у реальному часі. Переміщення токена по ігровій карті вимагає миттєвої реакції системи, щоб усі учасники сесії бачили зміни без візуальних затримок. Для моделювання взаємодії між клієнтськими застосунками та сервером під час цієї події побудовано діаграму послідовності (Sequence Diagram), яку наведено на рисунку 2.5.

З метою забезпечення плавності ігрового процесу (30 FPS), у системі застосовано підхід "оптимістичного інтерфейсу" (Optimistic UI). Коли Гравець 1 починає перетягувати токен за допомогою комп'ютерної миші (drag-and-drop), графічний рушій клієнта миттєво оновлює його координати на локальному екрані, не чекаючи підтвердження від сервера.

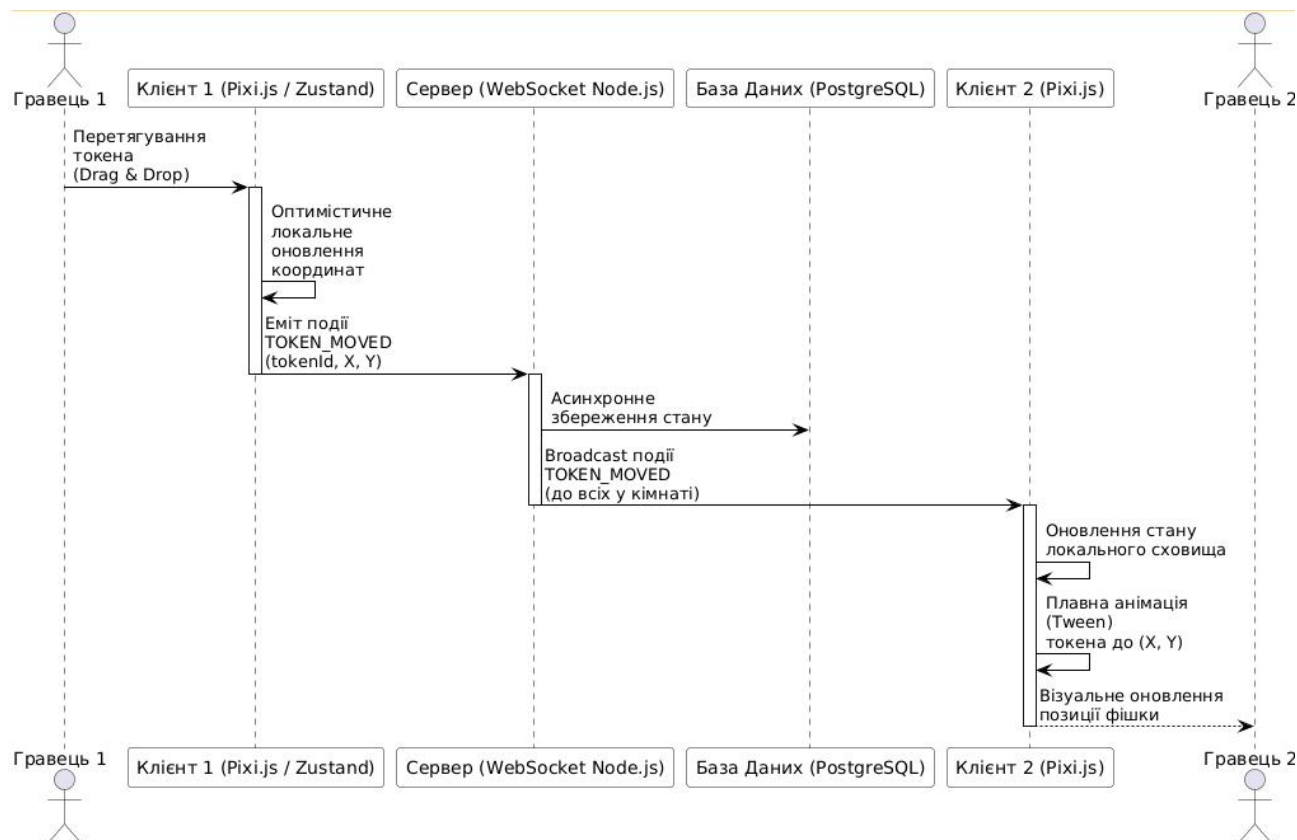


Рис. 2.5. Діаграма Послідовності синхронізації ігрових об'єктів у реальному часі.

Після завершення переміщення, клієнтський застосунок формує бінарний або текстовий пакет (payload), що містить ідентифікатор токена та його нові координати (X, Y), і відправляє його на сервер через відкритий WebSocket-канал. Серверний обробник (Node.js) приймає подію, асинхронно ініціює збереження нових координат у базу даних для забезпечення персистентності, та миттєво ретранслює (broadcast) цю подію всім іншим підключеним клієнтам у поточній ігровій кімнаті. Отримавши подію, застосунок Гравця 2 оновлює свій локальний стан у Zustand, що своєю чергою дає команду Pixi.js плавно анімувати (Tween) переміщення токена на нову позицію. Такий асинхронний алгоритм мінімізує вплив мережевої затримки на загальний користувацький досвід.

Останнім алгоритмом, який потребує детального опису через свою комплексну природу, є процес генерації та обробки кидків кубиків. Оскільки візуальний модуль використовує спеціалізовану бібліотеку для 3D-візуалізації з повноцінною симуляцією фізики твердих тіл, цей процес проходить через серію

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

чітко визначених станів. Крім того, підсистема кидків тісно інтегрована з іншими модулями (наприклад, трекером ініціативи), що ускладнює життєвий цикл події. Для ілюстрації цих переходів побудовано діаграму станів (State Machine Diagram), що наведена на рисунку 2.6.

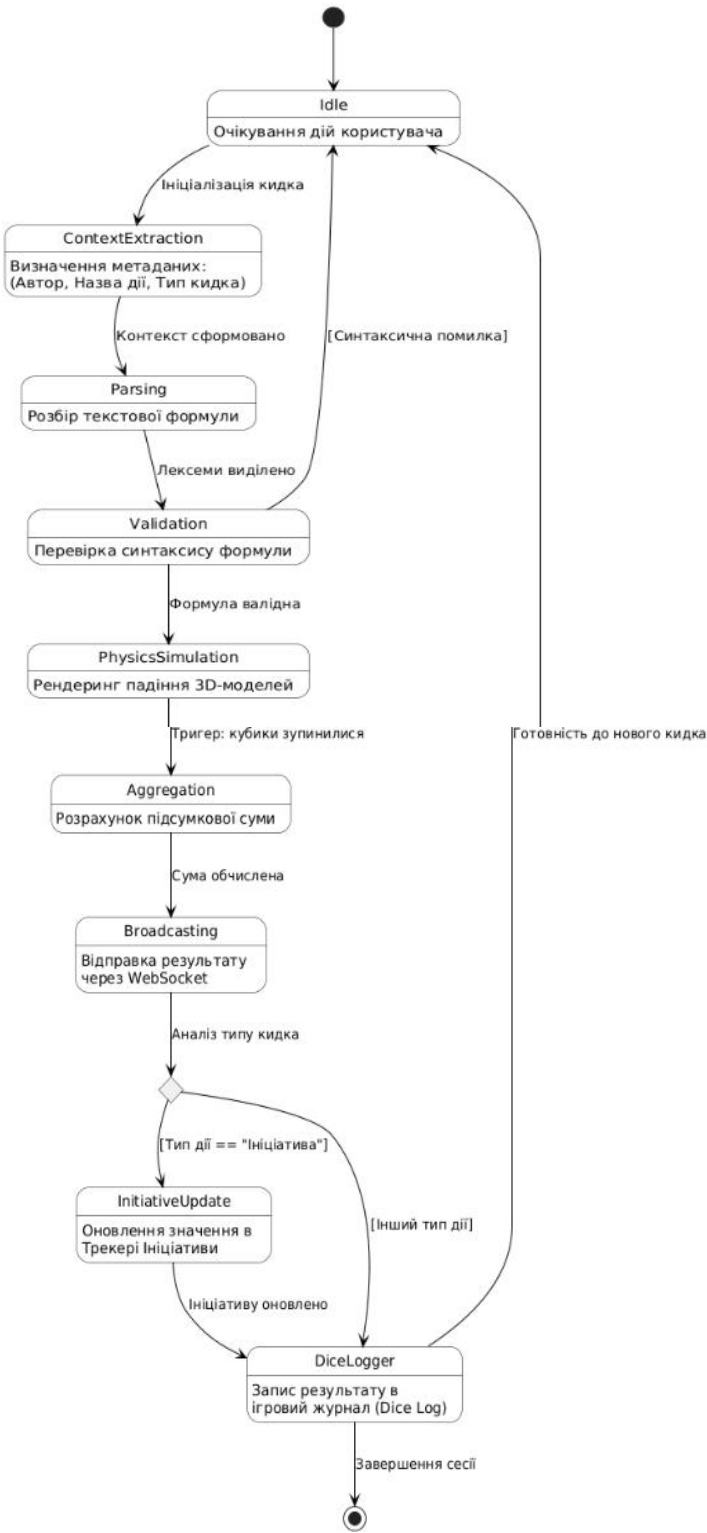


Рис. 2.6. Діаграма станів процесу генерації та обробки кидка кубика

Алгоритм починає роботу зі стану очікування («Idle»). Коли гравець ініціює дію, система спочатку переходить у стан вилучення контексту («Context Extraction»): фіксується автор кидка, тип дії та її назва (наприклад, атака конкретною зброєю чи перевірка навички). Далі текстова формула (наприклад, "1d20+5") розбирається на лексеми та проходить синтаксичну валідацію. У разі успіху активується стан фізичної симуляції у Three.js.

Перехід до наступного стану («Агрегація») відбувається виключно за умови триггеру події повної зупинки всіх 3D-моделей на екрані. Після обчислення підсумкової суми та трансляції результату через WebSocket, алгоритм проходить через умовний вузол (Type Check). На цьому етапі система перевіряє метадані кидка: якщо дія ідентифікована як "Ініціатива", результат автоматично маршрутизується до стану оновлення Трекера Ініціативи («Initiative Update»), що впливає на черговість ходів у бою. Незалежно від гілки розгалуження, фінальним станом є запис усіх зібраних даних у модуль DiceLogger, який забезпечує персистентне збереження логу та відображення деталізованої інформації (хто кинув, що кинув і які грані випали) в інтерфейсі всіх учасників.

Ще одним вагомим функціональним алгоритмом візуального модуля є процес обробки інструментів вільного малювання (Drawing Tool) та розрахунку відстаней за допомогою віртуальної лінійки (Ruler Tool). Ці інструменти відіграють ключову роль у тактичному плануванні під час гри. Їхня реалізація вимагає безперервної математичної взаємодії з параметрами ігрової сітки (Grid) та частого оновлення графічного контексту під час переміщення вказівника користувача.

Для опису цієї логіки побудовано діаграму активності (Activity Diagram), яку наведено на рисунку 2.7. Вона відображає послідовність дій системи від моменту натискання кнопки миші до фіксації результату на сервері.

Алгоритм активується в момент події pointerdown на графічному полотні (Canvas), коли система фіксує початкові координати вказівника. Далі запускається цикл обробки події pointermove. Якщо користувач обрав

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

інструмент "Пензель", система додає нові координати до тимчасового масиву векторів. Щоб уникнути переповнення оперативної пам'яті через тисячі подій переміщення за секунду, до координат застосовується механізм проріджування (Throttling), після чого лінія рендериться на локальному екрані гравця.

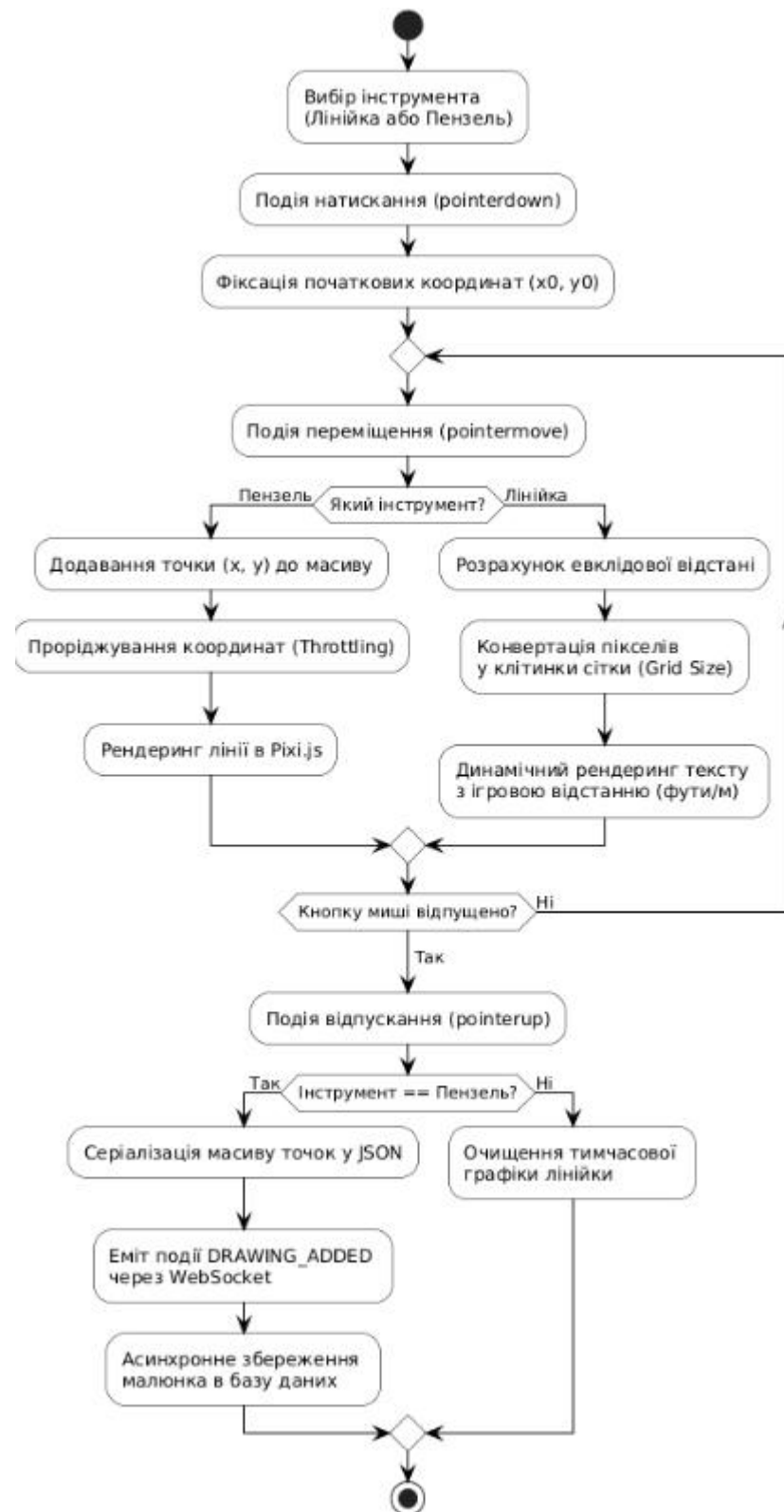


Рис. 2.7. Діаграми Активності Малювання

У випадку використання інструмента "Лінійка", алгоритм на кожному кроці циклу вираховує евклідову відстань між стартовою та поточною точками. Отримане значення у пікселях динамічно ділиться на розмір клітинки сітки (gridSize) поточної сцени, після чого конвертується в ігрові одиниці виміру (наприклад, фути або метри) і відмальовується у вигляді тексту над лінією.

Цикл переривається настанням події pointerup. Для лінійки цей етап є кінцевим і система просто очищає тимчасову графіку. Однак для інструмента малювання система виконує серіалізацію зібраного масиву точок у формат JSON, після чого ініціює відправку даних на сервер через WebSocket та паралельне збереження векторного об'єкта в базі даних (таблиця VttLayerItem). Такий підхід гарантує точність малювання та мінімізує навантаження на мережу.

Загалом, спроектовані алгоритми ініціалізації віртуальної кімнати, просторової синхронізації ігрових об'єктів та обробки результатів кидків кубиків формують надійне поведінкове ядро візуального модуля системи. Використання сучасних асинхронних підходів, механізму оптимістичного оновлення інтерфейсу та гнучкої маршрутизації мережевих подій дозволяє платформі підтримувати високу швидкість відгуку навіть за умов інтенсивного навантаження або нестабільного інтернет-з'єднання. Особливого значення в архітектурі набуває синергія між подієво-орієнтованою моделлю WebSocket для швидкісного обміну даними та апаратним прискоренням графіки на базі WebGL, що в комплексі забезпечують ефект безперервної взаємодії користувачів за віртуальним столом. Побудовані алгоритмічні моделі не лише детально ілюструють складну бізнес-логіку програми, але й повністю покривають усі технічні вимоги до багатокористувацької платформи.

2.4 Реалізація функціоналу модуля віртуального ігрового столу

Для забезпечення максимальної продуктивності та гнучкості, клієнтську частину візуального модуля було реалізовано з використанням сучасної екосистеми React у тісній інтеграції з високопродуктивним графічним рушієм PIXI.js. Такий архітектурний підхід вимагає вирішення специфічних завдань:

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

поєднання декларативної моделі DOM з імперативним світом WebGL, швидкісне управління глобальним станом клієнта (через Zustand) та асинхронна комунікація в реальному часі через WebSockets. У даному підрозділі детально розглянуто програмну реалізацію цих ключових компонентів системи.

Проте, перш ніж переходити безпосередньо до рендерингу UI-елементів та налаштування мережових з'єднань, важливо зазначити, що фундаментом стабільної роботи візуального клієнта виступає його суворі доменна бізнес-логіка. Увесь ігровий простір побудований навколо набору базових сутностей (сцен, токенів, елементів малювання, трекерів ініціативи та симуляторів кубиків), які відповідають за обробку координат, фізику та трансформацію даних перед їхньою відправкою на сервер.

Для глибшого розуміння внутрішньої архітектури та життєвого циклу цих базових класів необхідно деталізувати їхні ключові методи та поведінкові патерни, що інкапсульовані в об'єктно-орієнтованій моделі. Оскільки візуальний модуль працює на стику DOM-дерева та WebGL-контексту, кожен клас несе пряму відповідальність за оптимізацію використання пам'яті.

У класі Scene центральним архітектурним методом є `loadBackground()`. Його алгоритм передбачає не лише завантаження статичного зображення за вказаною URL-адресою, але й асинхронну ініціалізацію базової текстури (`BaseTexture`) безпосередньо в пам'яті графічного рушія `Pixi.js`. Це гарантує, що рендеринг картини відбудеться лише після повного завантаження активу в оперативну пам'ять відеокарти клієнтського пристрою, унеможливлюючи появу "артефактів" при малюванні екрану. Своєю чергою, метод `updateGrid()` відповідає за динамічний перерахунок координат відмальовування клітинок ігрової сітки залежно від поточного масштабу (`zoom`) та зсуву камери (`pan`) гравця. Цей метод оптимізований для виконання в межах циклу `requestAnimationFrame`, що забезпечує стабільну частоту оновлення інтерфейсу (30 FPS) під час навігації картою.

Поведінка класу `Token` зосереджена навколо просторової маніпуляції. Метод `move(newX, newY)` містить комплексну логіку валідації координат.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Перед фактичним візуальним переміщенням об'єкта алгоритм перевіряє, чи увімкнено на активній сцені режим прив'язки до сітки (Snap to grid). Якщо ця функція активна, передані сирі координати вказівника миші математично округлюються до найближчого центру або кута клітинки, залежно від розміру токена. Метод `updateStatus()` виконує синхронізацію візуального відображення токена (наприклад, колір рамки залежно від налаштувань Майстра, або відображення смужки здоров'я) із поточним станом прив'язаного аркуша персонажа (`CharacterSheet`). Таким чином у системі реалізується патерн Спостерігач (`Observer`), де об'єкт токена виступає "слухачем" змін стану свого прототипу в базі даних.

Клас `Drawing` переважно керується методом `addPoint(x, y)`, який безперервно викликається під час обробки події переміщення вказівника (`pointermove`) на полотні. Для уникнення перевантаження оперативної пам'яті клієнта та мережевого каналу масивом із тисяч координат, алгоритм використовує підхід проріджування точок (`Throttling`) у поєднанні з базовою інтерполяцією для згладжування векторних ліній. Це дозволяє передавати на сервер компактний набір контрольних точок, який згодом з високою точністю реконструюється на клієнтах інших учасників ігрової сесії. Метод `clear()` забезпечує повне очищення графічного контексту шару малювання, форсовано звільняючи виділену під об'єкт пам'ять відеокарти.

Для сервісного класу `InitiativeTracker` життєво необхідним є метод `nextTurn()`. Його логіка не обмежується простим інкрементуванням індексу активного гравця. Алгоритм постійно перевіряє умови завершення поточного бойового раунду: якщо індекс ходу досягає останнього елемента в відсортованому масиві бойових одиниць, метод автоматично збільшує загальний лічильник раундів (`roundCounter`) та обнуляє індекс черги. Це гарантує цілісність покрокового циклу гри без необхідності ручного втручання Майстра гри. Окрім цього, методи додавання та видалення учасників автоматично пересортують чергу на основі параметру ініціативи (`value`), що є критичним при появі нових монстрів під час бою.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

Нарешті, у класі DiceRoll метод `parseFormula()` відіграє роль повноцінного лексичного та синтаксичного аналізатора (парсера). Він перетворює нестандартні рядкові команди користувачів (на кшталт складних формул "2d20+1d4+5") на структурований масив об'єктів-завдань, зрозумілих для фізичного рушія Three.js. Після ініціалізації фізичної симуляції падіння 3D-моделей кубиків, управління перехоплює метод агрегації, який формує масив проміжних результатів `resultParts` (значення, що випали на окремих гранях кубиків) та обчислює фінальне математичне значення `total`. Лише після успішного завершення всіх обчислень об'єкт серіалізується для логування та трансляції.

Основою візуалізації ігрової карти є головний компонент `VttBattlefield`, який слугує логічним мостом між станом вебдодатка та графічним полотном (Canvas) відеокарти клієнта. Для безперешкодного використання об'єктів `Pixi.js` всередині стандартного синтаксису JSX використовується пакет `@pixi/react` [34]. Перед початком рендерингу відбувається явна реєстрація базових компонентів рушія (`Container`, `Graphics`, `Sprite`, `Text`), що дозволяє інкапсулювати складні виклики WebGL у зручні React-теги.

Уся ігрова сцена структурно розділена на логічні шари, що відповідають патерну графа сцени (Scene Graph): `BackgroundLayer` (фонова текстура), `GridLayer` (координатна сітка), `TokenLayer` (динамічні фішки гравців та монстрів), `DrawingLayer` (шар вільного малювання) та `RulerLayer` (тимчасовий шар лінійки). Такий підхід забезпечує абсолютну ізоляцію перемальовування: наприклад, зміна положення одного токена не змушує графічний рушій витрачати ресурси на перемальовування важкої фонові карти або сітки.

Фрагмент програмного коду ініціалізації середовища та обробки подій вказівника (Pointer Events) для інтерактивних шарів наведено у лістингу 2.1.

```
import { Application, extend } from '@pixi/react';
import { Container, Graphics, Sprite, Text } from 'pixi.js';
extend({ Container, Graphics, Sprite, Text });
function BattlefieldContent({ viewport, vttConnection, viewerId }) {
  const drawingTool = useBattlefieldStore(s => s.drawingTool);
```

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

```

const currentScene = useBattlefieldStore(s => s.activeScene);
const handlePointerDown = useCallback((e) => {
  if (e.button !== 0) return;
  const localPos = e.data.getLocalPosition(e.currentTarget.parent);
  if (drawingTool) {
    const newDrawing = {
      type: drawingTool,
      points: [localPos.x, localPos.y],
      color: drawingColor,
      userId: viewerId};
    setLocalDrawing(newDrawing);
    vttConnection?.sendVttSceneDrawPreview(currentScene.id, viewerId, newDrawing);}},
  [drawingTool, currentScene, viewerId, drawingColor]);}

```

Лістинг 2.1. Фрагмент коду ініціалізації середовища PIXI.js та обробки подій вказівника

Як видно з лістингу 2.1, при натисканні на графічне полотно, компонент автоматично розраховує локальні координати події відносно масштабу сцени. Після цього формується об'єкт малюнка, який одночасно записується в локальний стан для миттєвого відображення (optimistic update) та паралельно відправляється по WebSocket-каналу всім підключеним клієнтам для забезпечення просторової синхронізації.

Для забезпечення високої продуктивності та надійної синхронізації даних в умовах багатокористувацької сесії, в архітектурі візуального модуля застосовано гібридний підхід до управління станом. Замість використання монолітних та важких рішень (на кшталт Redux), архітектуру логічно поділено на два рівні: серверний стан (Server State) та локальний стан інтерфейсу (Client State).

Для управління серверним станом (кешування HTTP-запитів до бази даних, фонові оновлення, пагінація) використовується бібліотека TanStack React Query. Вона автоматично обробляє життєві цикли мережевих запитів (стадії isLoading, isSuccess, isError), а також дозволяє миттєво інвалідувати

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

(очищати) локальний кеш після виконання асинхронних мутацій (наприклад, після збереження змін у листі персонажа чи оновлення налаштувань кімнати).

Своєю чергою, для надшвидкого управління локальним станом віртуальної кімнати застосовується Zustand. Це мінімалістична бібліотека на основі хуків, яка вирішує проблему "Prop Drilling" (передачі пропсів через десятки компонентів). Вона дозволяє графічному рушію Pixi.js та React-інтерфейсу звертатися до єдиного сховища (наприклад, за поточними координатами вказівника або обраним інструментом малювання) без необхідності повного перерендерингу всього дерева компонентів, що критично для підтримки 30 FPS.

Фрагмент програмного коду, що демонструє взаємодію цих двох інструментів у межах єдиної архітектури, наведено у лістингу 2.2.

```
export const useBattlefieldStore = create((set) => ({
  activeTool: 'pointer',
  drawingColor: '#FF0000',
  setActiveTool: (tool) => set({ activeTool: tool }),
}));
export function useVttQueries(roomId) {
  const queryClient = useQueryClient();
  const roomQuery = useQuery({
    queryKey: ['vttRoom', roomId],
    queryFn: async () => {
      const response = await api.get(`/vtt/rooms/${roomId}`);
      return response.data;
    },
  });
  const updateSceneMutation = useMutation({
    mutationFn: async (sceneData) => {
      return await api.put(`/vtt/scenes/${sceneData.id}`, sceneData);
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['vttRoom', roomId] });
    },
  });
  return { roomQuery, updateSceneMutation };
}
```

Лістинг 2.2. Фрагмент використання Zustand та React Query

Як видно з лістингу 2.2, Zustand (useBattlefieldStore) забезпечує миттєву зміну інструментів або параметрів без жодних мережових затримок. Водночас React Query бере на себе повну відповідальність за безпечне завантаження

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

важких масивів даних з PostgreSQL та гарантує консистентність даних (Data Consistency) між клієнтом та сервером.

Одним із найскладніших інженерних завдань при розробці модуля віртуального ігрового столу є забезпечення мінімальної затримки (Low Latency) під час обміну просторовими даними між гравцями. Використання класичного HTTP-протоколу з архітектурою регулярних опитувань (Short/Long Polling) не відповідає вимогам високої інтерактивності, оскільки створює надмірне навантаження на сервер та генерує візуальні затримки на клієнті. Саме тому для синхронізації координат токенів, трансляції вільного малювання та поширення результатів кидків кубиків було імплементовано повнодуплексний протокол WebSocket.

Для ефективного управління мережевим з'єднанням на стороні клієнта розроблено спеціалізований React-хук, який інкапсулює складну логіку підключення, автентифікації, автоматичного відновлення з'єднання (Reconnection) та маршрутизації вхідних подій. Усі ігрові події, що транслуються сервером, проходять через єдиний комунікаційний шлюз, після чого передаються до локального сховища стану.

Фрагмент програмного коду, що демонструє процес ініціалізації WebSocket-з'єднання та підписки на події оновлення ігрової сцени, наведено у лістингу 2.3.

```
import { useEffect } from 'react';
import { io } from 'socket.io-client';
import { useBattlefieldStore } from './useBattlefieldStore';
export function useVttConnection(roomId, userToken) {
  useEffect(() => {
    const socket = io(process.env.REACT_APP_WS_URL, {
      auth: { token: userToken },
      query: { roomId } });
    socket.on('vtt:tokenMoved', (payload) => {
      const { tokenId, x, y } = payload;
      useBattlefieldStore.getState().updateTokenPosition(tokenId, x, y);
    });
    socket.on('vtt:drawPreview', (payload) => {
```

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        useBattlefieldStore.getState().setDrawPreview(payload.userId, payload.drawing);});
socket.on('vtt:diceRolled', (payload) => {
    useBattlefieldStore.getState().addDiceRollToLog(payload);});
return () => {
    socket.disconnect();}},
[roomId, userToken]);}

```

Лістинг 2.3. Фрагмент коду обробки WebSocket-подій на клієнті

Як проілюстровано в лістингу 2.3, архітектура обробки подій тісно інтегрована із системою управління локальним станом (Zustand). Коли клієнт отримує бінарний пакет або JSON-повідомлення через WebSocket, він не взаємодіє з графічним рушієм PIXI.js напряму. Натомість оновлюється виключно локальний стан useBattlefieldStore, що своєю чергою автоматично тригерить відповідні плавні анімації (Tweening) об'єктів на ігровому полотні. Такий строгий однонаправлений потік даних (Unidirectional Data Flow) робить архітектуру передбачуваною і надійно захищає систему від виникнення непередбачуваних гонок даних (Race Conditions) при інтенсивному потоці повідомлень.

Генерація випадкових чисел у настільних рольових іграх є ключовою базовою механікою. Для максимального занурення користувачів у процес гри, замість простої алгоритмічної генерації псевдовипадкових значень, у візуальному модулі було імплементовано повноцінну 3D-симуляцію фізичного кидка кубиків. Цей функціонал реалізовано на базі інтеграції спеціалізованих бібліотек екосистеми WebGL - Three.js для рендерингу графіки та Cannon.js для розрахунку фізики твердих тіл.

Інтеграція даного компонента вимагала реалізації складного конвеєра обробки даних: від парсингу текстових формул (наприклад, "2d6+3") у масив об'єктів-завдань для фізичного рушія, до подальшої математичної агрегації результатів після повної зупинки всіх тривимірних моделей на екрані гравця.

Фрагмент програмного коду, що відповідає за ініціалізацію компонента 3D-кубиків та асинхронну обробку результатів симуляції, наведено у лістингу 2.4.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

```

export function useDiceSimulator(containerSelector) {
  const [diceBox, setDiceBox] = useState(null);
  useEffect(() => {
    const box = new DiceBox(containerSelector, {
      assetPath: '/assets/dice-box/', // Шлях до 3D-моделей та текстур
      theme: 'default',
      scale: 6, gravity: 2, friction: 0.8 });
    box.init().then(() => setDiceBox(box));
  }, [containerSelector]);
  const rollDice = useCallback(async (formula, modifier = 0) => {
    const details = result.map(diceGroup => {
      diceGroup.rolls.forEach(roll => {
        totalDiceValue += roll.value;});
      return { type: diceGroup.type, rolls: diceGroup.rolls };});
    const finalTotal = totalDiceValue + modifier;
    return { formula: `${formula}${modifier > 0 ? '+' + modifier : ''}`,
      total: finalTotal,
      details;}, [diceBox]);
  return { rollDice, isReady: !!diceBox };}

```

Лістинг 2.4. Фрагмент коду інтеграції фізичної 3D-симуляції кубиків

Як продемонстровано у лістингу 2.4, симулятор кубиків ініціалізується асинхронно, підвантажуючи важкі текстури та 3D-моделі геометрії багатогранників без блокування головного потоку додатка (Main Thread). Метод rollDice використовує механізм промісів (Promises): виконання асинхронного коду призупиняється (await diceBox.roll()) рівно до моменту, поки фізичний рушій не зафіксує стану спокою (повної зупинки) всіх віртуальних тіл на сцені. Лише після цього алгоритм циклічно проходить по масиву результатів кожної грані, додає до суми константний модифікатор з листа персонажа та формує фінальний об'єкт події (Payload), який готовий до відправки в локальний журнал (Dice Logger) та паралельної трансляції іншим учасникам ігрової кімнати.

Візуалізація характеристик персонажів (Character Sheets), управління інвентарем та відображення параметрів монстрів вимагають рендерингу

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

значного обсягу текстових і числових даних на екрані. Оскільки архітектура бази даних підтримує динамічне збереження ігрових параметрів у гнучкому форматі JSON, інтерфейс користувача (UI) повинен бути максимально універсальним і композитним.

Для реалізації інтерактивних листів застосовано модульний підхід на базі функціональних компонентів React. Стилзація інтерфейсу виконана за допомогою сучасного утилітарного CSS-фреймворку TailwindCSS, що дозволяє досягти високої швидкості розробки та гарантує адаптивність компонентів без створення надмірної кількості зовнішніх таблиць стилів. Окрім візуальної репрезентації, ці компоненти виконують роль "пультів управління" ігровим процесом.

Фрагмент програмного коду, що ілюструє типову структуру компонента характеристик з інтегрованими тригерами ініціації кидків кубиків, наведено у лістингу 2.5.

```
const triggerDiceRoll = useBattlefieldStore(s => s.triggerDiceRoll);
const handleStatRoll = useCallback((statName, statValue) => {
  const modifier = Math.floor((statValue - 10) / 2);
  triggerDiceRoll('1d20', modifier, {
    author: sheetData.name,
    actionType: `Перевірка характеристики: ${statName.toUpperCase()}` });
}, [sheetData.name, triggerDiceRoll]); return (
<div className="flex flex-col gap-4 p-4 bg-gray-900 text-white rounded-lg shadow-xl">
  <div className="border-b border-gray-700 pb-2">
    <h2 className="text-2xl font-bold">{sheetData.name}</h2>
    <div className="flex justify-between text-sm text-gray-400 mt-1">
      <span>Клас броні (AC): {sheetData.stats?.ac || 10}</span>
      <span>Здоров'я (HP): {sheetData.hpCurrent} / {sheetData.hpMax}</span>
    </div>
    <div className="grid grid-cols-3 gap-2 mt-4">
      {Object.entries(sheetData.stats || {}).map(([key, value]) => (
        <button key={key}
          onClick={() => handleStatRoll(key, value)}
          className="flex flex-col items-center p-3 bg-gray-800 hover:bg-gray-700
            border border-gray-600 rounded cursor-pointer transition-colors"
```

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

```

title={`Кинути перевірку ${key}`}>
<span className="text-xs uppercase font-semibold text-gray-400">{key}</span>

```

Лістинг 2.5. Фрагмент React-компонента інтерактивного аркуша персонажа

Як продемонстровано в лістингу 2.5, компонент використовує методологію Data-Driven Design (проектування на основі даних). Ітерація по масиву JSON-об'єкта характеристик (`Object.entries(sheetData.stats)`) автоматично генерує набір інтерактивних кнопок. Кожна кнопка виконує динамічну калькуляцію математичного ігрового модифікатора і зв'язується з функцією `handleStatRoll`. Ця функція, своєю чергою, формує об'єкт контекстного запиту і передає управління до глобального стану Zustand (`triggerDiceRoll`), замикаючи архітектурний цикл системи: від взаємодії гравця з інтерфейсом до виконання фізичної 3D-симуляції та мережевої синхронізації події.

Окрім клієнтської частини, критично важливою складовою візуального модуля є серверна архітектура, яка виступає єдиним джерелом істини (Single Source of Truth) для всіх ігрових сесій. Оскільки обробка кожної дії через базу даних PostgreSQL призвела б до неприпустимих затримок, для керування просторовим станом у реальному часі було розроблено спеціалізований in-memory менеджер `VttStateManager`. Цей сервісний клас зберігає глобальний стан кожної активної ігрової кімнати в оперативній пам'яті сервера (використовуючи структуру Map), забезпечуючи миттєвий доступ до ієрархії сцен, шарів та токенів.

Для гарантування безпеки даних під час мережевої взаємодії, `VttStateManager` імплементує сувору політику валідації вхідних параметрів. Замість прямого перезапису об'єктів, застосовується алгоритм фільтрації за "білим списком" (Whitelist). Фрагмент програмного коду, що демонструє логіку безпечного оновлення властивостей ігрової сцени, наведено у лістингу 2.6.

```

const ALLOWED_SCENE_UPDATE_FIELDS = new Set([
  'name', 'width', 'height', 'gridSize', 'backgroundColor',
  'gridEnabled', 'gridType', 'gridColor', 'gridOpacity', 'gridScale']);

```

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

```

function filterUpdates(updates, allowedFields) {
  if (!updates || typeof updates !== 'object')
    return {};
  return Object.fromEntries(
    Object.entries(updates).filter(([key]) => allowedFields.has(key)) );
}

updateScene(sessionId, sceneId, updates) {
  const room = this._ensureRoom(sessionId);
  if (!room.scenes[sceneId]) return null;
  const safeUpdates = filterUpdates(updates, ALLOWED_SCENE_UPDATE_FIELDS);
  room.scenes[sceneId] =
  { ...room.scenes[sceneId], ...safeUpdates };
  return room.scenes[sceneId];
}

```

Лістинг 2.6. Фрагмент серверної логіки безпечного оновлення стану сцени

Як видно з лістингу 2.6, метод `filterUpdates` перехоплює будь-який пакет даних від клієнта і відсікає всі незадокументовані або критичні поля (наприклад, `id` або масив `layers`). Це унеможливорює компрометацію ігрової кімнати злоумисниками через підробку WebSocket-пакетів. Успішно відфільтровані дані миттєво об'єднуються з поточним станом сцени в оперативній пам'яті.

Після валідації та локального збереження стану, сервер повинен забезпечити миттєву реплікацію оновлень на клієнти. Цей процес координується модулем обробки WebSocket-подій. Особливістю реалізованого алгоритму є асинхронна гібридна модель збереження: подія спочатку транслюється всім учасникам сесії для забезпечення ефекту "нульової затримки", і лише паралельно (у фоновому режимі) записується до реляційної бази даних PostgreSQL. Фрагмент коду, що ілюструє цей механізм на прикладі кидка кубика, наведено у лістингу 2.7.

```

async function handleVttDiceRoll(socket, payload, roomManager) {
  const sessionId = parseSessionId(payload.sessionId);
  const { formula, name, strength, visibility, characterName } = payload;
  const player = characterName || socket.user?.username || 'Гравець';
  const initiatorId = socket.user?.id ? String(socket.user.id) : null;
  const rollResult = { ...rollDice(formula, player),
    name, strength: strength || 1, visibility: visibility || 'PUBLIC', initiatorId };
}

```

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

```

const entry = vttStateManager.addDiceRoll(sessionId, rollResult);
vttSceneService.saveDiceRoll(sessionId, entry, Number(initiatorId))
  catch((err) => logger.error({ err }, '[VTT] dice roll DB save failed'));
if (entry.visibility === 'GM_ONLY') {
  const gmId = String(vttStateManager.getVttState(sessionId).openedBy);
  roomManager.broadcastFilter(getVttRoom(sessionId), {
    type: 'vtt:dice:result', sessionId, roll: entry}, (clientSocket) => {
    const clientId = String(clientSocket.user?.id);
    return clientId === gmId || clientId === initiatorId;});} else
  {roomManager.broadcast(getVttRoom(sessionId), {
    type: 'vtt:dice:result', sessionId, roll: entry});}}

```

Лістинг 2.7. Фрагмент обробника WebSocket-подій для трансляції ігрових кидків

Алгоритм, представлений у лістингу 2.7, генерує об'єкт кидка на сервері для запобігання чітерству з боку клієнтів (метод `rollDice`), фіксує його в оперативній пам'яті (`addDiceRoll`), після чого ініціює збереження (метод `saveDiceRoll`). Завдяки відсутності ключового слова `await` перед викликом бази даних, основний потік виконання (Event Loop Node.js) не блокується в очікуванні відповіді від жорсткого диску. Замість цього сервер одразу переходить до розсилання повідомлення. Крім того, блок умовних операторів реалізує механізм прихованих кидків (`GM_ONLY`), фільтруючи цільових отримувачів пакета за ідентифікаторами. Такий архітектурний підхід забезпечує стабільну пропускну здатність платформи та гарантує консистентність даних під час пікових навантажень.

Таким чином, практична програмна реалізація всіх функціональних підсистем візуального модуля спирається на єдину, чітко структуровану архітектуру. Суворий розподіл відповідальності між шарами графічного рушія, менеджерами глобального та локального станів, мережевими WebSocket-слухачами та інтерактивними UI-компонентами гарантує високу відмовостійкість коду. Такий підхід не лише вирішує поточні завдання рендерингу та синхронізації, але й забезпечує надзвичайну легкість подальшого масштабування системи при додаванні нових ігрових механік.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

Висновки до другого розділу

У другому розділі кваліфікаційної роботи здійснено комплексне проектування та програмну реалізацію архітектури модуля віртуального ігрового столу. Застосування методів об'єктно-орієнтованого проектування дозволило створити стійку логічну структуру додатка, що охоплює управління станом, бази даних та складні алгоритми мережевої взаємодії у реальному часі

На етапі визначення вимог до системи було сформовано перелік пріоритетних функціональних завдань та виокремлено дві ключові ігрові ролі: Гравця та Майстра гри. За допомогою UML-діаграм варіантів використання та класів було формалізовано доменну логіку модуля, яка виступає гнучким контейнером для всіх просторових об'єктів (сцен, токенів, малюнків).

Під час розробки бази даних спроектовано реляційну схему на базі PostgreSQL з використанням інструмента Prisma ORM. Для забезпечення цілісності ієрархії ігрового простору застосовано систему паралельних зовнішніх ключів. Такий підхід дозволив платформі ефективно розділяти ізольовані дані тимчасових ігрових кімнат та глобальний багатосесійний прогрес сюжетних кампаній без втрати швидкодії та масштабованості.

Для опису динаміки роботи модуля спроектовано ключові поведінкові алгоритми системи. За допомогою UML-діаграм було детально формалізовано логіку ініціалізації ігрової кімнати, процеси просторової синхронізації фішок через WebSocket, а також математичні механізми вимірювання дистанцій, векторного малювання та 3D-генерації результатів кидків кубиків.

На етапі реалізації функціоналу успішно імплементовано гібридний підхід до управління станом за допомогою екосистеми React. Розподіл відповідальності між локальним менеджером Zustand (для надшвидкого оновлення графіки) та серверним кешем TanStack React Query дозволив уникнути перевантаження браузера. Доведено високу ефективність застосування апаратно-прискореного рушія PIXI.js та 3D-симулятора Three.js, які у синергії з протоколом WebSocket забезпечили миттєвий відгук інтерфейсу під час багатокористувацької гри.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ
НАСТІЛЬНО-РОЛЬОВИХ ІГОР

3.1 Порядок встановлення та налаштування параметрів платформи

Процес розгортання та встановлення вебплатформи уніфіковано для всіх її підсистем за допомогою технології контейнеризації Docker та інструменту оркестрації Docker Compose. Оскільки загальна інфраструктура системи, налаштування бази даних PostgreSQL, кешування Redis [17] та маршрутизація Nginx є спільними для всієї платформи, у даному підрозділі увагу зосереджено на специфіці конфігурування модуля віртуального ігрового столу (VTT).

Варто зазначити, що процес локальної розробки та тестування платформи здійснювався за допомогою підсистеми Windows Subsystem for Linux (WSL 2) з використанням дистрибутива Ubuntu. Такий підхід забезпечив абсолютну архітектурну ідентичність середовища виконання між локальною робочою станцією розробника (Windows) та цільовим хмарним сервером (Linux). Використання WSL у синергії з Docker Desktop унеможливило виникнення конфліктів файлових систем та гарантувало стабільну роботу мережеских мостів (Network Bridges) під час розгортання контейнерів на етапі тестування.

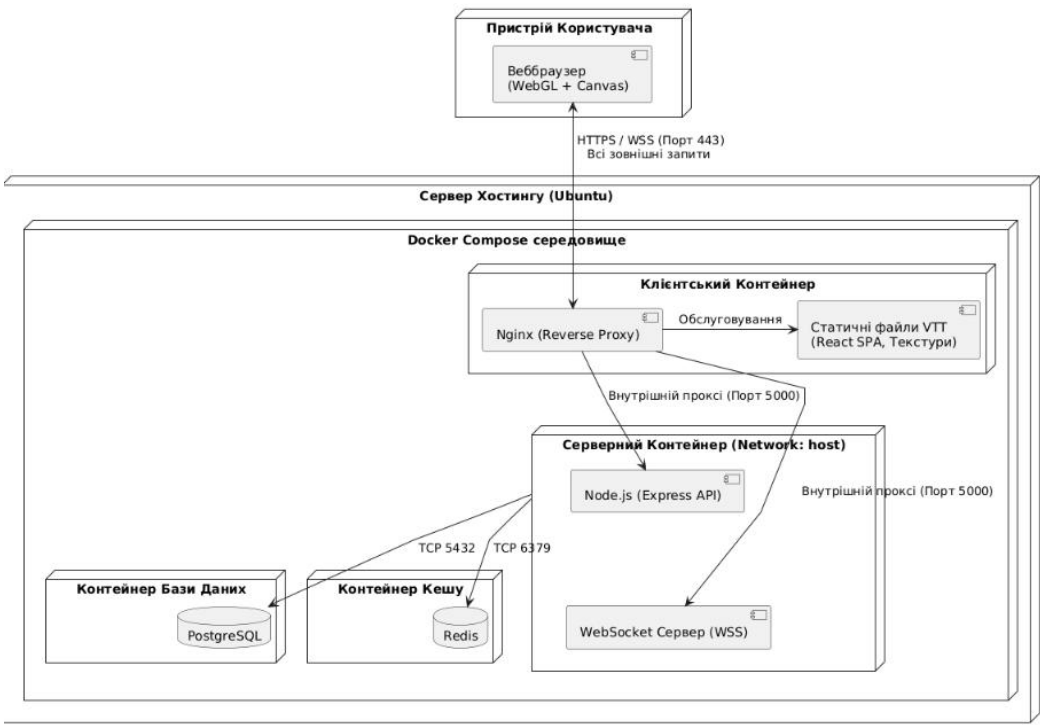


Рис. 3.1. Діаграма розгортання

Для наочної демонстрації фізичної архітектури апаратних вузлів та розміщених на них програмних артефактів розроблено UML-діаграму розгортання (Deployment Diagram). Вона ілюструє мережеву взаємодію між клієнтським браузером, вебсервером статичних файлів та контейнеризованими бекенд-сервісами (рис. 3.1).

Візуальний модуль є найбільш ресурсоємною частиною платформи. Для забезпечення безперебійної обробки сотень WebSocket-повідомлень за секунду (під час синхронного переміщення токенів), критично важливо правильно розподілити апаратні ресурси між контейнерами. У таблиці 3.1 наведено специфікацію ресурсних обмежень Docker-контейнерів.

Таблиця 3.1.

Склад контейнерів та ресурсні обмеження системи

Сервіс (Контейнер)	Базовий образ	Ліміт CPU	Ліміт RAM	Цільове призначення
Database	postgres:15	0.75 Core	512 МБ	Збереження персистентного стану ігрових сесій
Cache	redis:7-alpine	0.25 Core	128 МБ	Кешування сесій та Rate Limiting для WebSockets
Server	node (Express + mediasoup)	1.0 Core	768 МБ	Обробка бізнес-логіки та маршрутизація WSS подій
Client	nginx	0.25 Core	64 МБ	Роздача скомпільованого SPA та статичних 3D-ассетів

Важливою складовою налаштування системи є конфігурування змінних середовища (файл .env), що визначають поведінку мережевих протоколів та інфраструктури безпеки. Для коректного функціонування підсистеми реального часу (WebSockets та WebRTC) обов'язковим є задання пулу портів для маршрутизації медіапотоків (MEDIASOUP_MIN_PORT та MEDIASOUP_MAX_PORT), які виділені в діапазоні 40000-40499 для безперешкодної передачі UDP-трафіку.

Крім того, для гарантування стабільного P2P та SFU з'єднання між учасниками з різними типами мережевих підключень (особливо у випадках подвійного NAT або використання провайдерами строгих брандмауерів),

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

архітектура платформи вимагає налаштування параметрів TURN-сервера. За це відповідають критичні змінні TURN_ENABLED, TURN_URL, TURN_REALM та TURN_SHARED_SECRET. Вони передаються у серверний контейнер під час старту Docker Compose і ніколи не зберігаються у відкритому вигляді в репозиторії системи контролю версій.

Оскільки візуальний модуль VTT передбачає неперервний обмін великими обсягами даних, конфігурація клієнтського Nginx-контейнера включає оптимізацію зворотного проксіювання (Reverse Proxy) [20]. Спеціальні директиви в конфігураційному файлі дозволяють підтримувати довготривалі з'єднання (keep-alive) для протоколу WebSocket та уникати автоматичного розриву з'єднання (timeout) під час тривалих ігрових сесій (які можуть тривати понад 4 години). Це рішення суттєво підвищує загальну стабільність роботи віртуального столу під час пікових навантажень або нестабільного інтернет-з'єднання з боку клієнта.

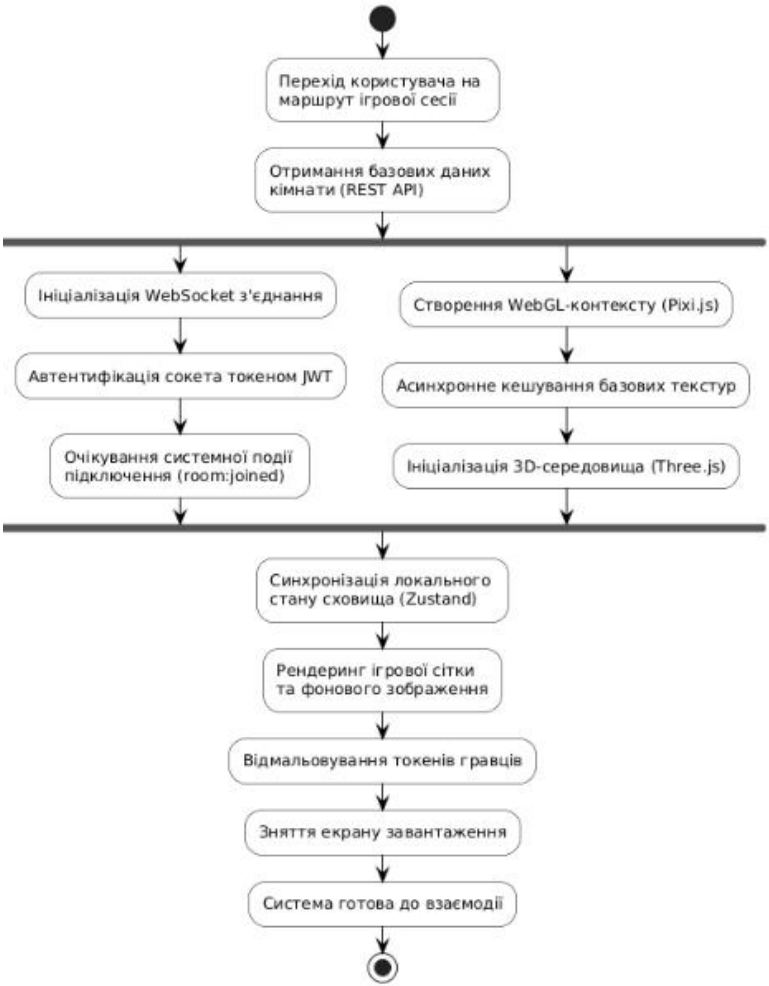


Рис. 3.2. Діаграма активностей (Запуск клієнтського VTT)

Окремої уваги заслуговує процес старту візуального модуля на стороні користувача. Оскільки VTT-клієнт використовує важкі графічні рушії, його запуск є складним багатопотоковим процесом. Для формалізації цього етапу розроблено діаграму активностей ініціалізації клієнтського середовища (рис. 3.2).

Як видно з діаграми, процес стартується переходом користувача на захищений маршрут ігрової кімнати. Алгоритм розгалужується на два паралельні потоки: перший відповідає за встановлення та автентифікацію повнодуплексного WebSocket-з'єднання, а другий - за ініціалізацію WebGL-контексту (Pixi.js) та завантаження 3D-текстур (Three.js).

Лише після успішного завершення обох паралельних процесів система переходить до агрегації локального стану (через Zustand), малювання сітки та розстановки токенів. Такий синхронізований підхід до ініціалізації гарантує, що гравець ніколи не побачить «зламаний» або неповний інтерфейс, а взаємодія з картою буде доступна тільки за наявності підключення до сервера.

У разі виникнення непередбачуваних мережевих помилок на будь-якому з етапів завантаження, система автоматично призупиняє ініціалізацію та виводить користувачеві відповідне вікно з пропозицією безпечного перепідключення. Крім того, такий архітектурний поділ дозволяє ефективно керувати оперативною пам'яттю клієнтського пристрою (Garbage Collection): важкі графічні активи та WebGL-контекст автоматично вивантажуються з пам'яті браузера одразу після закриття вкладки ігрової сесії, запобігаючи витокам пам'яті (memory leaks) при тривалому використанні додатку.

3.2 Структура інтерфейсу платформи

Головним завданням при проєктуванні інтерфейсу візуального модуля (віртуального ігрового столу) було створення ергономічного та неперевантаженого робочого простору. Оскільки ключовим елементом взаємодії користувача з системою є ігрова карта (графічне полотно WebGL), класичний підхід зі статичними бічними панелями навігації був відкинутий.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

Натомість інтерфейс побудовано на основі концепції «плаваючих віджетів» (Floating UI Components).

Усі елементи управління системою - панель графічних інструментів, текстовий чат, журнал кидків кубиків, трекер ініціативи, інтегрований відеодзвінок та налаштування сцени - існують у вигляді незалежних інтерактивних вікон, що розміщуються поверх ігрового полотна. Користувач має можливість вільно переміщувати їх по екрану (Drag-and-Drop), згортати для максимального звільнення візуального простору (Collapse) та програмно блокувати їхнє положення (Lock), запобігаючи випадковому зсуву віджетів під час інтенсивного ігрового процесу.

Така модульна архітектура інтерфейсу забезпечує високу персоналізацію робочого середовища як для гравців, так і для майстра гри, дозволяючи адаптувати систему під будь-який розмір діагоналі монітора без втрати функціональності.

Для детального ознайомлення з візуальними та функціональними можливостями розробленого модуля, нижче наведено послідовний огляд ключових компонентів віртуального ігрового столу. Взаємодія користувача з ігровим середовищем розпочинається з головного екрана сцени, який слугує базовим полотном для всіх подальших ігрових та комунікаційних операцій (рис. 3.3).

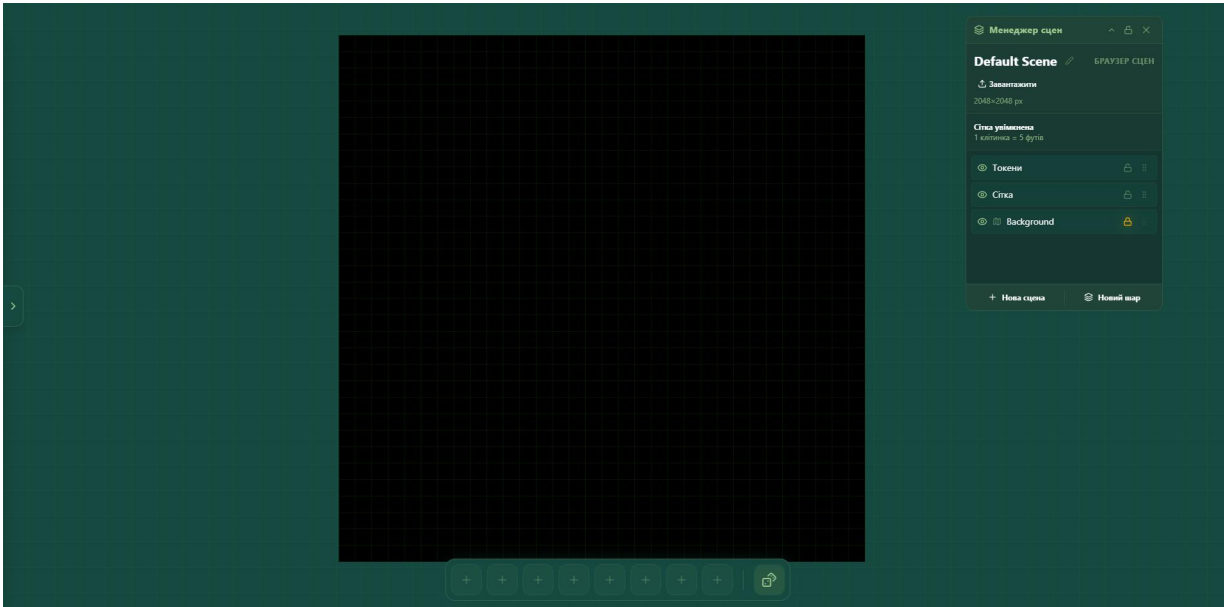


Рис. 3.3. Початковий вигляд віртуального ігрового столу для Майстра

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

На рисунку 3.3 зображено початковий стан ігрової кімнати одразу після успішної ініціалізації VTT-клієнта. Більшу частину екрана займає інтерактивне графічне полотно (Canvas), яке автоматично масштабується під розмір вікна браузера. За замовчуванням на полотні рендериться базова ігрова сітка, розмірність якої адаптована до загальноприйнятих правил настільних рольових ігор (де одна клітинка сітки еквівалентна 5 футам віртуального простору).

У правій частині екрана розташовано плаваючий віджет «Менеджер сцен», який слугує основним інструментом майстра гри для конфігурування локацій. Даний модуль реалізує повноцінну систему управління шарами (Layers System), концептуально подібну до професійних графічних редакторів. Усі об'єкти на екрані чітко структуровані за Z-індексом глибини. Зокрема, нижній шар (Background) призначений для завантаження та відмальовування статичної карти місцевості, наприклад, у роздільній здатності 2048x2048 пікселів. Над ним розташовується проміжний шар (Сітка), який відповідає за математичну прив'язку координат (Grid Snapping) та визначення масштабу ігрового поля. Своєю чергою, верхній шар (Токени) містить динамічні ігрові фішки персонажів, що переміщуються в реальному часі під час сесії.

Користувач з правами майстра має змогу гнучко керувати кожним шаром: приховувати його від гравців (кнопка видимості) або блокувати від випадкових змін (кнопка замка). Наявність елементів керування вікном у верхньому правому куті віджета (згортання, блокування позиції на екрані та закриття) наочно демонструє практичне застосування концепції Floating UI, про яку йшлося раніше. Це дозволяє майстру тримати необхідні інструменти під рукою, не перекриваючи огляд тактичної ситуації на карті.

Окрім менеджера сцен, навігація кімнатою здійснюється за допомогою панелі інструментів (рис. 3.4), розташованої у нижній частині екрана, та висувних бічних панелей для доступу до додаткових модулів. Завдяки такій продуманій організації простору гарантується висока ергономічність ігрового процесу та швидка адаптація інтерфейсу під будь-які потреби Майстра.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

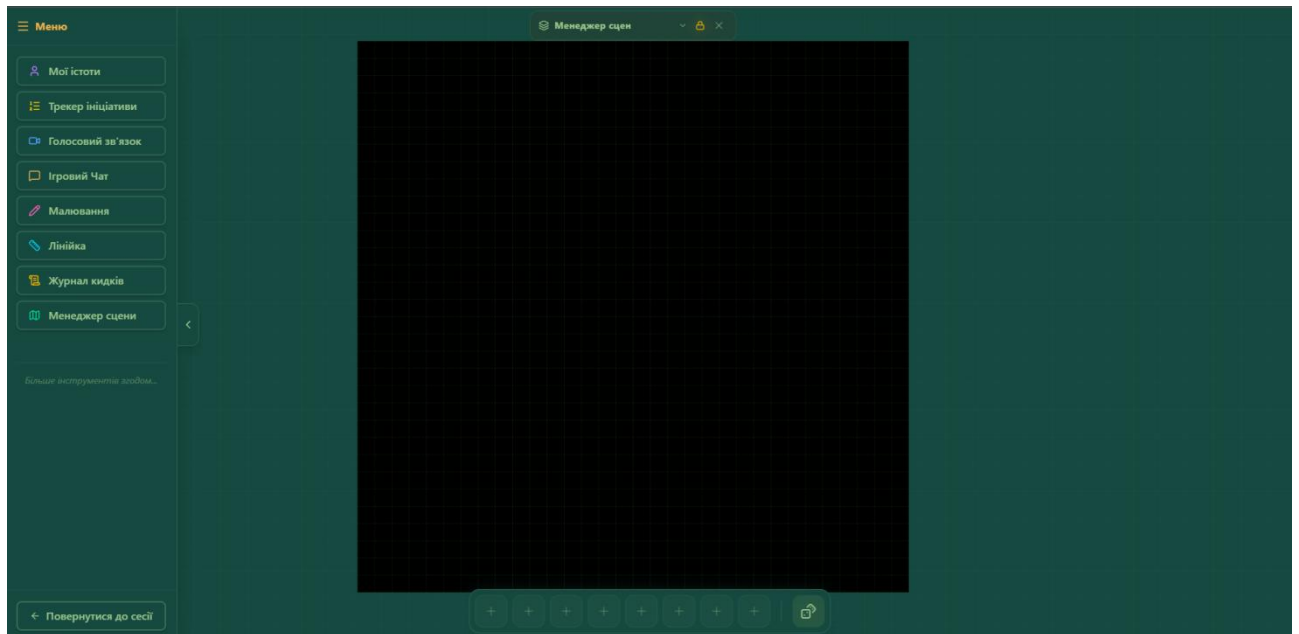


Рис. 3.4. Розгорнуте навігаційне меню майстра гри

Основним елементом глобальної навігації у віртуальній кімнаті є висувна бічна панель (Sidebar), зображена на рисунку 3.4. Ця панель виконує роль єдиного диспетчера для виклику плаваючих віджетів та доступу до різноманітних ігрових інструментів. Меню спроектовано за принципом висувної шухляди (Off-canvas Menu), що дозволяє приховувати його за межі екрана за допомогою спеціальної кнопки, звільняючи максимум простору для тактичної взаємодії з картою.

Перелік доступних пунктів у бічній панелі формується динамічно залежно від авторизованої ролі користувача, що гарантує суворе розмежування прав доступу безпосередньо на рівні клієнтського інтерфейсу. На рисунку 3.4 представлено інтерфейс майстра гри (GM), який містить вичерпний набір модулів: управління масовими сутностями («Мої істоти»), бойовий координатор («Трекер ініціативи»), модулі комунікації («Голосовий зв'язок», «Ігровий Чат»), графічні інструменти («Малювання», «Лінійка»), агрегатор результатів («Журнал кидків»), а також ексклюзивний інструмент глобального конфігурування - «Менеджер сцени». Слід також звернути увагу, що віджет менеджера сцен на даному скріншоті згорнуто у верхній частині екрана (Collapse-режим), що демонструє ергономічність керування плаваючими вікнами.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

Водночас, інтерфейс звичайного гравця (Player) зазнає суттєвих структурних змін задля унеможливлення несанкціонованого втручання в ігровий світ. Для ролі гравця з навігаційної панелі повністю вилучається пункт «Менеджер сцени», а вкладка керування NPC «Мої істоти» замінюється на вузькоспеціалізований розділ «Мій Персонаж», який надає доступ виключно до власного інтерактивного листа характеристик. Візуальна демонстрація інтерфейсу гравця з урахуванням зазначених обмежень наведена на рисунку 3.5.

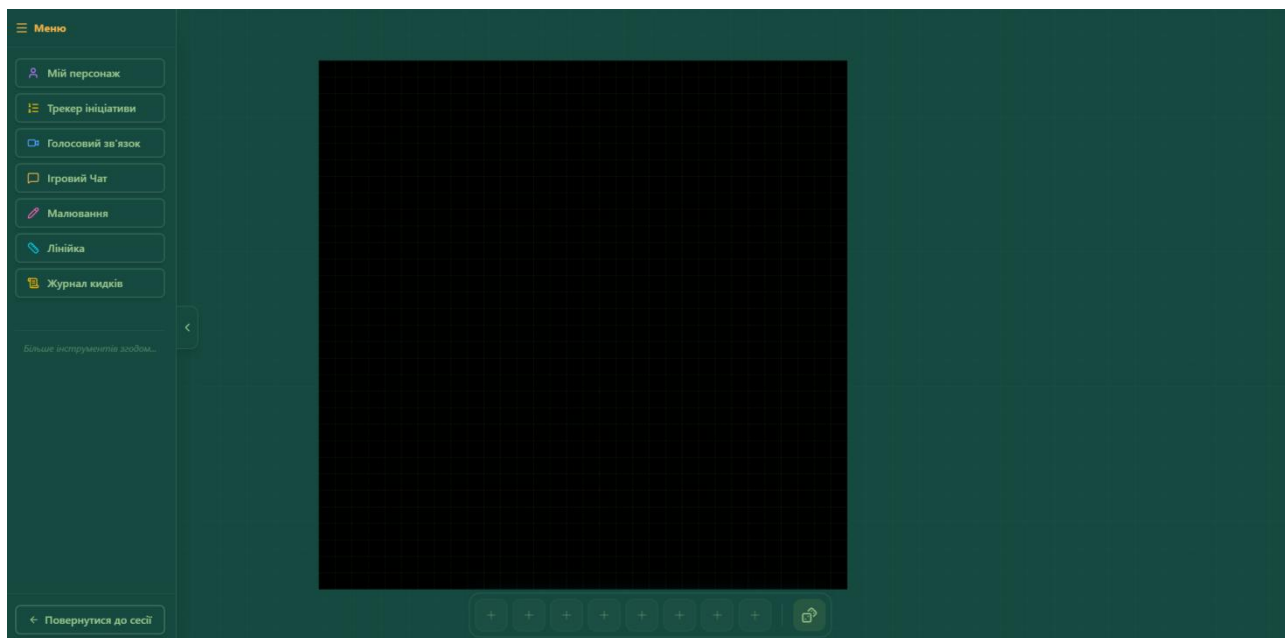


Рис. 3.5. Навігаційне меню звичайної ролі гравця

Решта інструментів, необхідних для повноцінної участі в сесії (комунікація, малювання, кубики, трекер ініціативи), залишаються доступними в повному обсязі. Такий рольовий підхід (Role-Based Access Control) не лише гарантує цілісність сюжету від втручання гравців, але й суттєво знижує когнітивне навантаження на нових користувачів, приховуючи від них складний інструментарій налаштування.

Після завантаження базового середовища, майстер гри має можливість наповнити сцену ігровими об'єктами. Цей процес розпочинається у модулі управління сутностями («Мої істоти»). Для зручності адміністрування та сортування система пропонує типізоване створення об'єктів: неігрових персонажів (NPC) або ворогів, що дозволяє логічно розділяти їх у майбутньому бойовому трекері ініціативи (рис. 3.6).

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

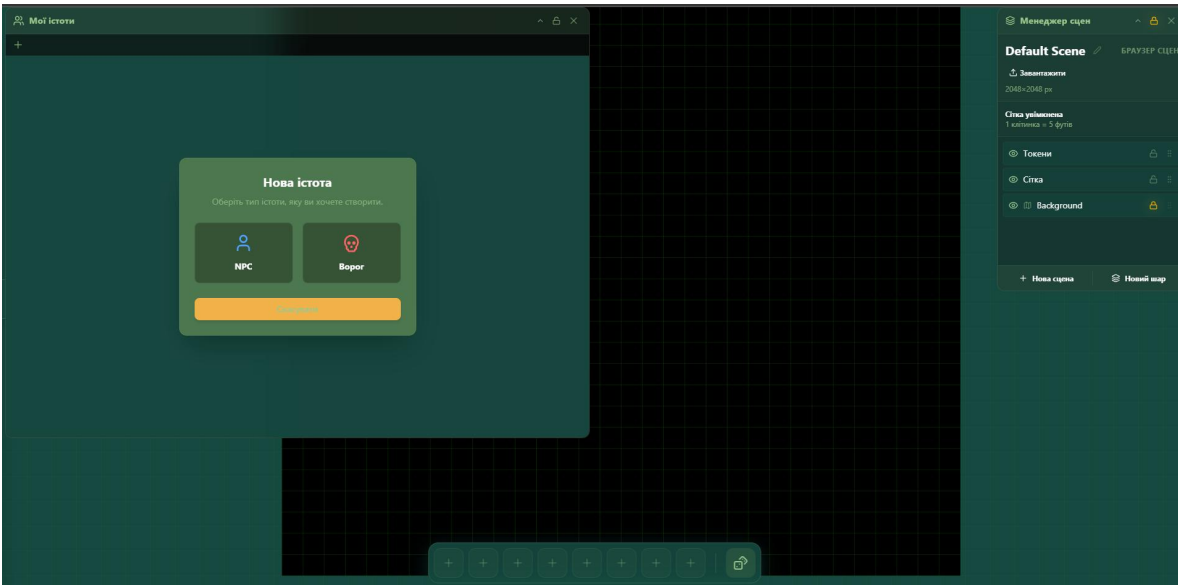


Рис. 3.6. Меню створення нової ігрової сутності

Після вибору типу сутності відкривається інтерактивний лист персонажа (Character Sheet), який інкапсулює всю математичну логіку обраної рольової системи (рис. 3.7). Інтерфейс листа побудовано з використанням сучасних патернів UI-дизайну: характеристики розбиті на логічні блоки (базові показники, навички, інвентар), а зміна будь-якого базового атрибута автоматично перераховує залежні ігрові модифікатори. Важливою архітектурною деталлю є кнопка «+ Додати Токен» у лівій панелі, яка слугує мостом між Data-рівнем системи: вона створює зв'язок між JSON-даними цього конкретного листа та фізичним графічним об'єктом (токеном) на полотні рушія Pixi.js.

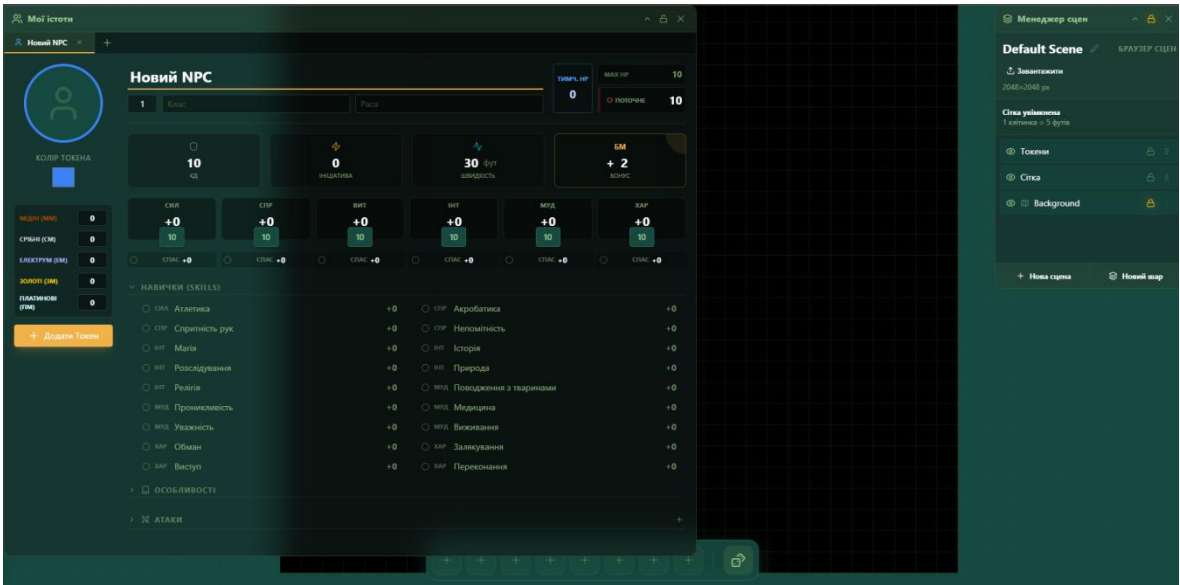


Рис. 3.7. Інтерактивний лист нового неігрового персонажа

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

Для демонстрації роботи реактивної форми на рисунку 3.8 наведено приклад заповненого профілю персонажа. Завдяки компонентній архітектурі React та локальному менеджеру стану Zustand, додавання зображення аватара, зміна характеристик та відмітка володіння навичками (жовті маркери) миттєво оновлюють візуальний стан компонента. Усі показники, виведені на цьому екрані, не лише зберігаються в базі даних PostgreSQL, але й безпосередньо використовуються математичним ядром VTT для автоматичного розрахунку бонусів при кидках кубиків, коли користувач взаємодіє з елементами інтерфейсу.

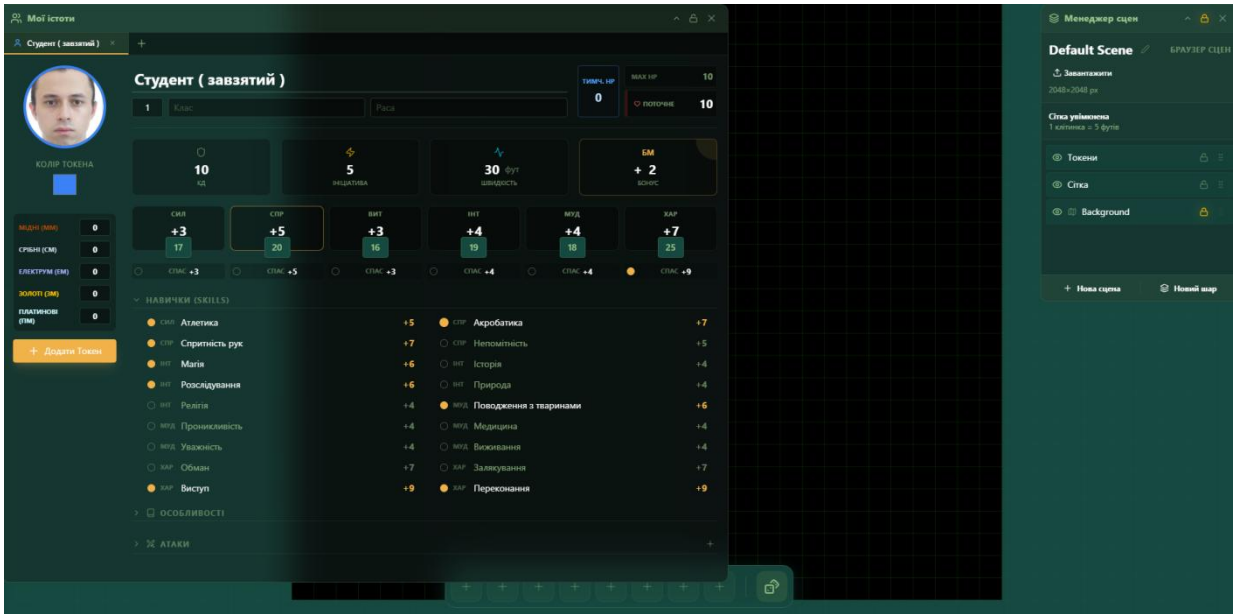


Рис. 3.8. Приклад заповненого профілю ігрової сутності

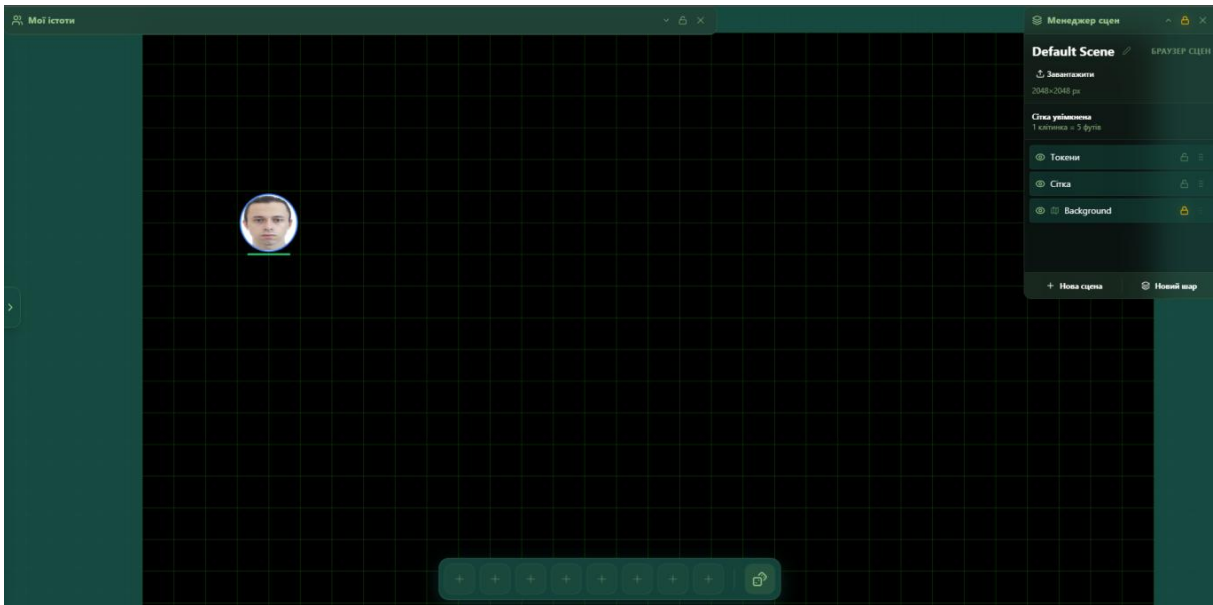


Рис. 3.9. Відмальовування ігрового токена на тактичній сітці

Рендеринг токена є складним процесом композиції. Базове зображення аватара обрізається за допомогою математичної маски (Graphics Mask), після чого до нього застосовуються динамічні UI-елементи: кольорова облямівка (яка може маркувати тип істоти: союзник, нейтрал чи ворог) та індикатор запасу здоров'я (Health Bar) у нижній частині. Оскільки токен і лист персонажа використовують єдине джерело істини в стані Zustand, зміна поточного показника HP у вікні характеристик миттєво ініціює перемальовування (re-render) довжини та кольору смуги здоров'я безпосередньо на ігровій карті (двостороннє зв'язування даних). На рисунку 3.10 продемонстровано взаємодію кількох токенів на ігровій сцені.

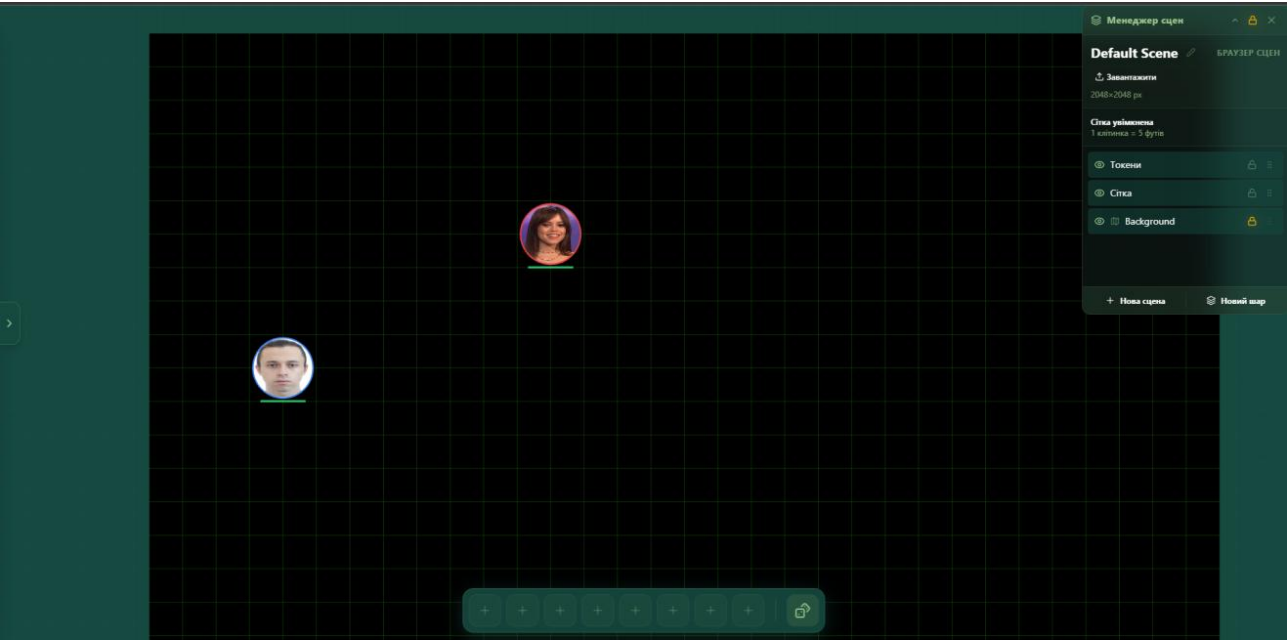


Рис. 3.10. Взаємодія кількох ігрових токенів на сцені

Графічний рушій платформи перехоплює та обробляє події вказівника (миші або сенсорного екрана), дозволяючи користувачам вільно переміщувати власні фішки методом Drag-and-Drop. Для зручності тактичного бою алгоритм переміщення автоматично виконує прив'язку координат об'єкта до найближчого центру клітинки (алгоритм Grid Snapping). Після відпускання токена (подія "pointerup"), графічний клієнт генерує оптимізований пакет даних із новими координатами (X, Y) та унікальним ідентифікатором токена. Цей пакет негайно відправляється через WebSocket на сервер і ретранслюється решті учасників,

що забезпечує плавну візуальну синхронізацію тактичної обстановки для всіх гравців у кімнаті.

Глибока інтеграція графічних об'єктів на тактичній сітці з даними бази даних реалізується через двосторонній зв'язок токена та його листа характеристик. При подвійному натисканні (Double-click) на графічний токен, система перехоплює подію рушія Pixi.js, вилучає унікальний ідентифікатор об'єкта та автоматично розгортає поверх полотна відповідний плаваючий лист персонажа. Це забезпечує гравцям та майстру миттєвий доступ до статистичних даних істоти безпосередньо під час тактичного бою.

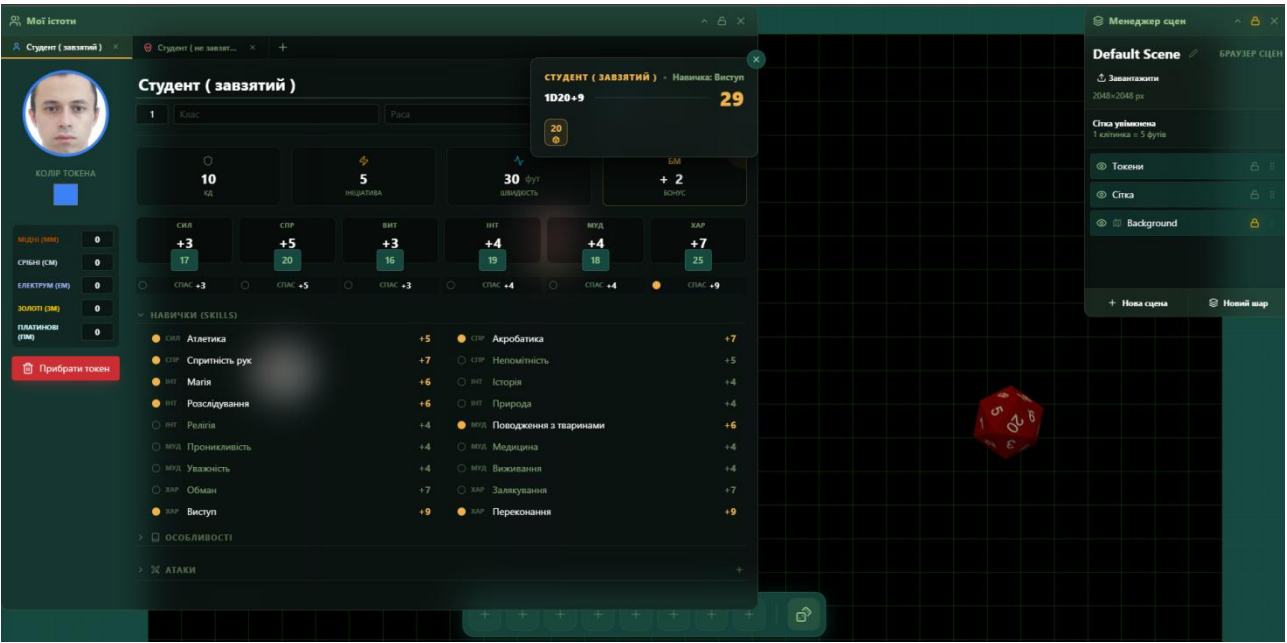


Рис. 3.11. Автоматизована генерація 3D-кидка на основі даних листа

Крім того, лист персонажа не є простою статичною таблицею. Кожна навичка, атака чи характеристика в ньому виступає інтерактивним тригером для фізичного рушія. Як показано на рисунку 3.11, при натисканні на конкретну навичку (наприклад, «Виступ»), алгоритм динамічно зчитує поточний математичний модифікатор із JSON-структури (+9), формує стандартну формулу перевірки (1d20+9) та передає її до підсистеми 3D-кубиків. На ігровому полотні ініціюється фізичне падіння тривимірного багатогранника, після повної зупинки якого агрегований результат миттєво обчислюється та виводиться у вигляді плаваючого сповіщення для всіх учасників кімнати.

Для оптимізації управління запасом здоров'я (НР) реалізовано спеціалізований функціональний модуль. При натисканні на блок НР у листі персонажа, поточний інтерфейс затемнюється і користувачу відкривається модальне вікно обробки поранень та зцілення (рис. 3.12).

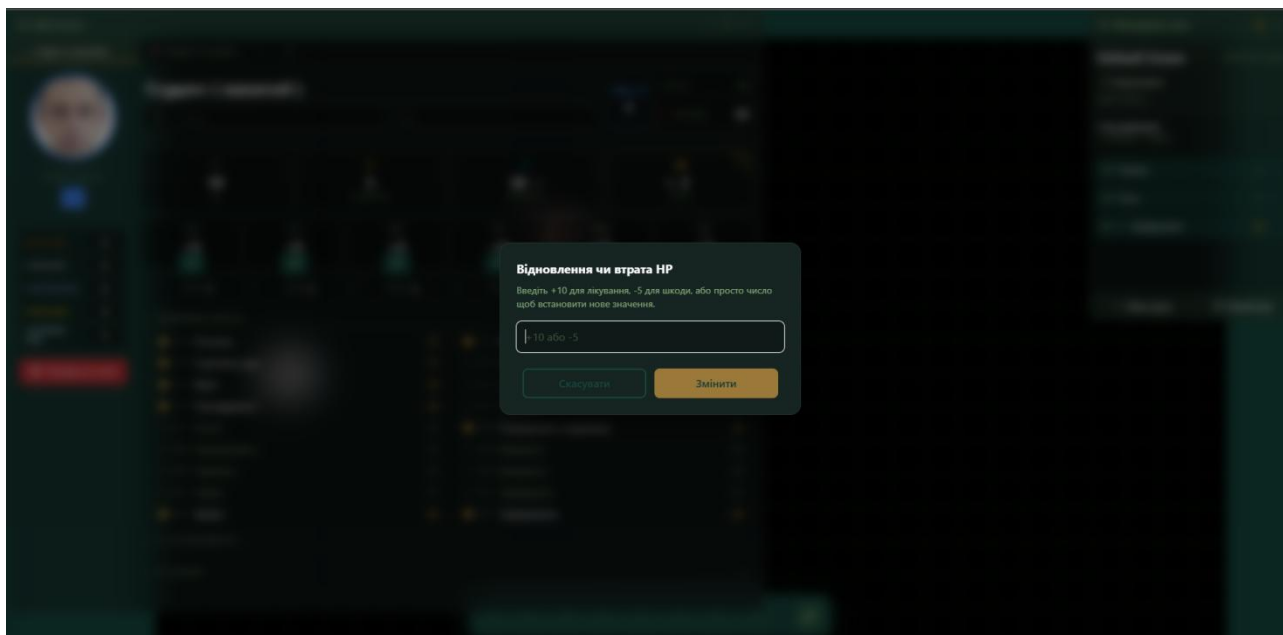


Рис. 3.12. Модальне вікно автоматизованого обчислення поранень та зцілення

Замість ручного математичного перерахунку залишку здоров'я, гравець або майстер вводить у поле отриману шкоду (наприклад, зі знаком мінус: -15) чи обсяг зцілення (зі знаком плюс: +10). Система автоматично валідує введені значення (не дозволяючи НР опуститися нижче нуля або перевищити максимальний поріг), оновлює поточний стан персонажа у глобальному сховищі та синхронно зменшує чи збільшує візуальну смужку здоров'я безпосередньо під токеном на тактичній карті.

Для автоматизації та координації бойових зіткнень, що є критично важливим аспектом настільних рольових ігор, у системі розроблено плаваючий модуль Трекера ініціативи (рис. 3.13). Цей інструмент забезпечує впорядкування учасників бою за результатом їхніх кидків та синхронізацію активної фази ходу між усіма підключеними клієнтами в режимі реального часу.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

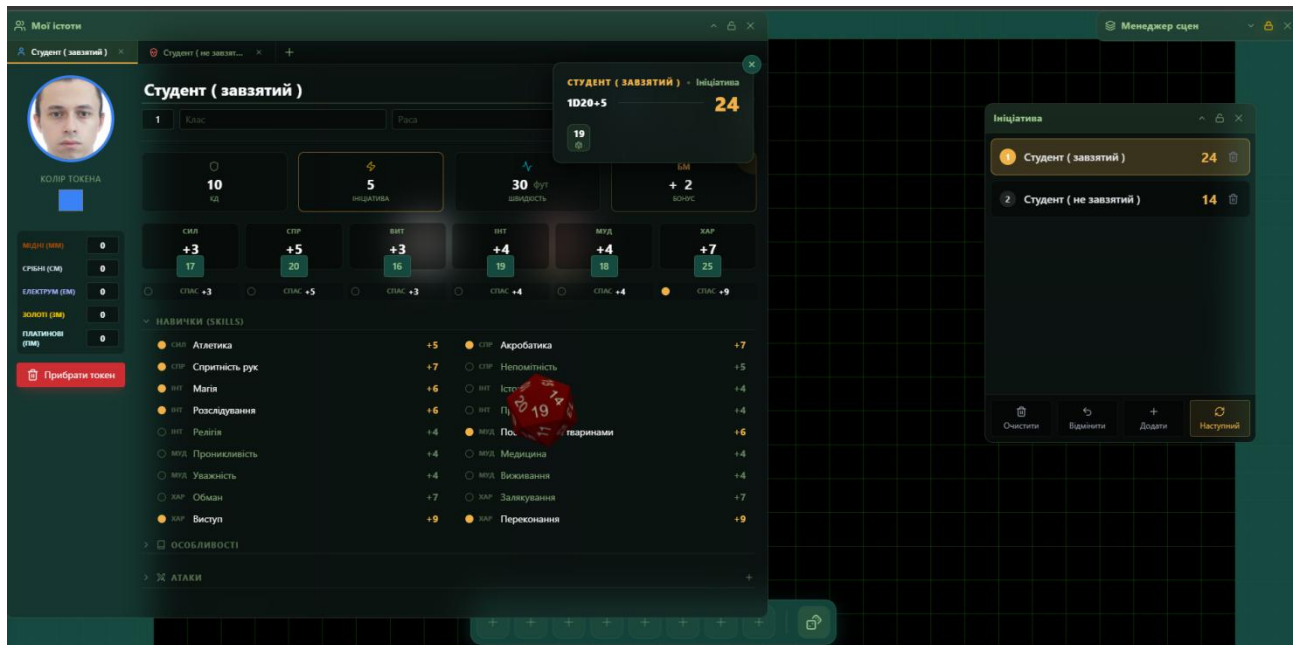


Рис. 3.13. Процес визначення ініціативи та оновлення черги ходу

Процес ініціалізації та ведення бою побудований на тісній взаємодії кількох підсистем вебплатформи. Усе розпочинається з ініціатії кидка: гравець або Майстер гри здійснює клік по інтерактивному блоку «Ініціатива» на панелі персонажа, після чого система зчитує поточний модифікатор (наприклад, +5) та формує запит на генерацію значення за формулою $1d20+5$. Далі відбувається візуалізація та обчислення результату. Запит передається до клієнтського 3D-рушія на базі Three.js, який генерує фізичну симуляцію падіння кубика. Отримане чисте значення додається до модифікатора персонажа, виводиться у спливаючому вікні та записується в історію ходів.

Наступним кроком є синхронізація через WebSocket: клієнт відправляє на сервер JSON-пакет з ідентифікатором токена та результатом. Сервер оновлює глобальний стан сцени та розсилає дані всім активним клієнтам.

На фінальному етапі менеджер стану клієнта (Zustand) автоматично сортує чергу трекера за спаданням результату кидка. При цьому активний хід завжди виділяється візуальним золотистим підсвічуванням картки сутності.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

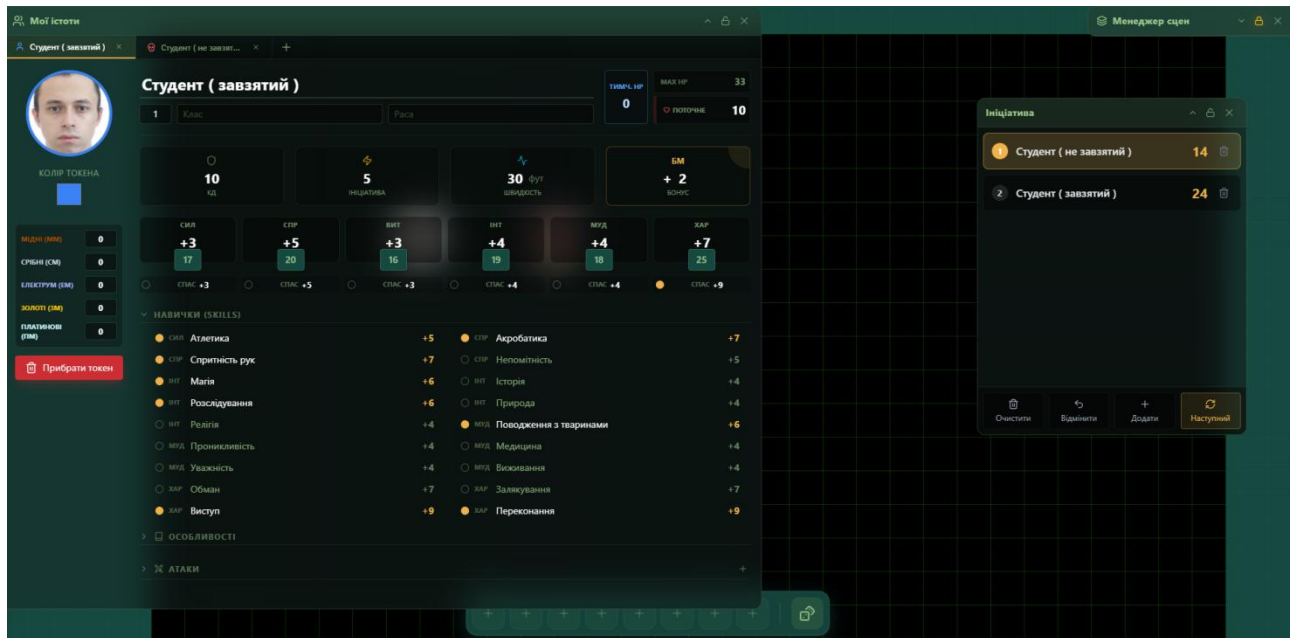


Рис. 3.14. Інтерфейс трекера ініціативи після переходу ходу до наступного учасника

На рис. 3.14 продемонстровано інтерфейс системи після активації Майстром гри переходу ходу (натискання кнопки «Наступний»).

Управління чергою ходу покладається на Майстра гри через інтерфейсну панель керування, розміщену в нижній частині віджета трекера ініціативи. Ця панель містить набір керуючих елементів для повного контролю за бойовою сценою. Зокрема, функція «Очистити» (Clear) забезпечує повне скидання бойового стану, очищення черги та видалення всіх сутностей з трекера після завершення зіткнення. У разі випадкового натискання або зміни рішення гравцем передбачена кнопка «Відмінити» (Undo), що дозволяє повернутися до попереднього ходу. Для гнучкого реагування на події під час бою використовується опція «Додати» (Add), яка надає можливість ручного додавання нових сутностей, таких як додаткові вороги або нейтральні NPC. Крім того, базовим інструментом просування ігрового часу є кнопка «Наступний» (Next), призначена для передачі ходу наступному за чергою учаснику.

Для забезпечення комунікації між учасниками ігрової сесії безпосередньо всередині платформи інтегровано два модулі зв'язку: Ігровий текстовий чат та інтегрований Модуль голосового й відеозв'язку. Завдяки плаваючій архітектурі

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

вікон користувачі можуть гнучко налаштовувати інтерфейс під поточні потреби від лаконічного вигляду з фокусом на ігровому полотні до детального робочого столу Майстра гри (GM) з великою кількістю відкритих інструментальних панелей. На рис. 3.15 представлено інтерфейс платформи у лаконічному режимі, де на екрані активовано лише ігрове полотно та текстовий чат.

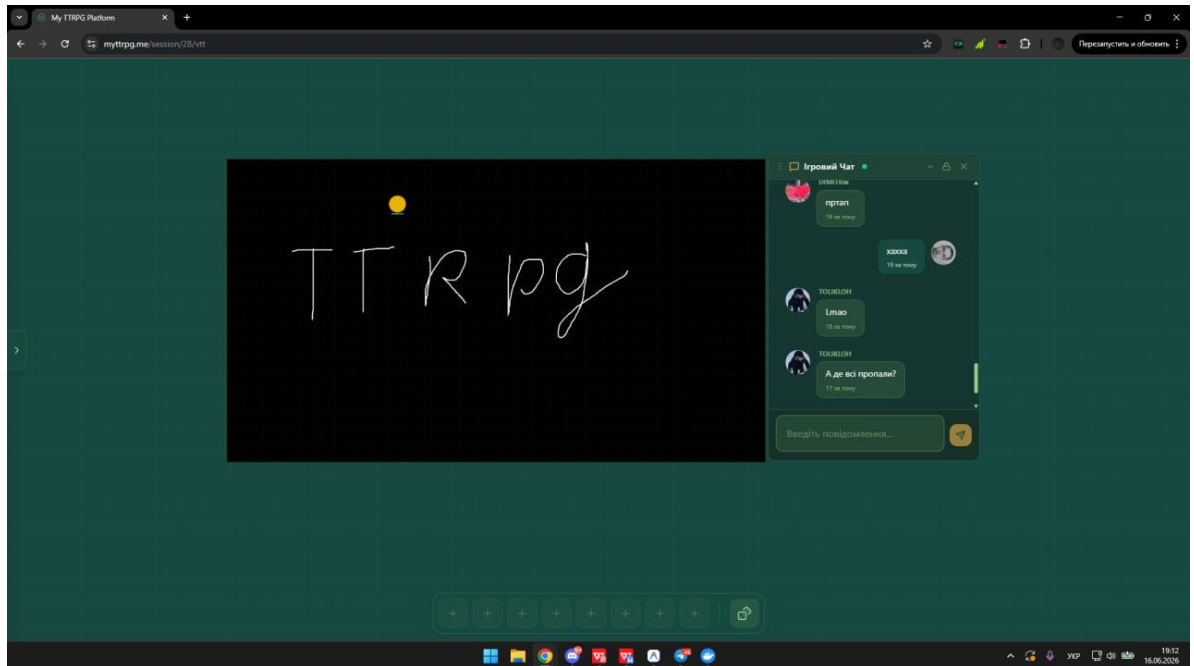


Рис. 3.15. Інтерфейс ігрового текстового чату в лаконічному режимі відображення

Текстовий ігровий чат (див. рис. 3.15, праворуч) розроблено на основі технології WebSockets, де клієнтська частина реалізована як динамічний React-компонент, що взаємодіє з сервером для миттєвої доставки повідомлень. Важливою характеристикою системи є асинхронний обмін даними, при якому повідомлення надсилаються у форматі JSON-пакетів, що містять ключові метадані автора, такі як ім'я, роль, аватар, а також точну мітку часу. Окрім цього, віджет відзначається високою гнучкістю інтерфейсу, оскільки повною мірою підтримує згортання, перетягування та блокування позиції на екрані. Його візуальний стиль гармонійно адаптовано під загальну колірну гаму платформи. Такий архітектурний підхід забезпечує ефективну економію екранного простору, адже у лаконічному режимі мінімізація додаткових інструментів дозволяє виділити понад 80% площі дисплея під головне графічне поле `Pixi.js` для комфортного відстеження токенів та елементів малювання.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

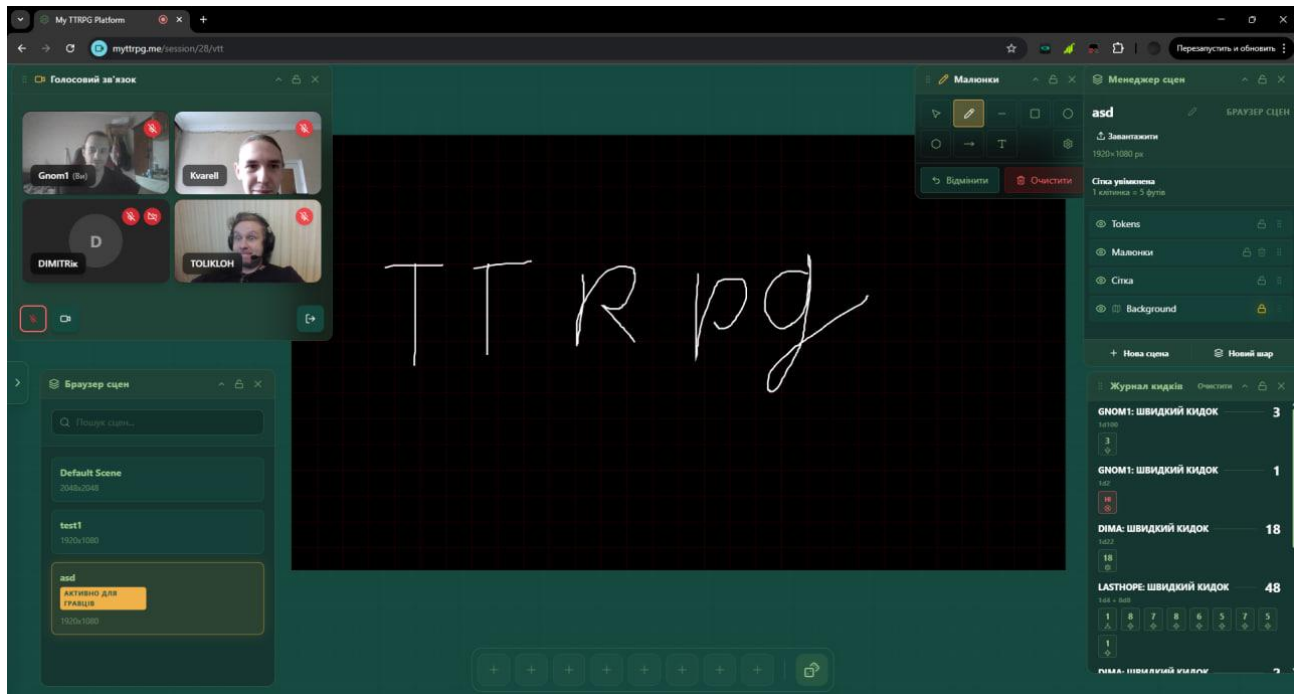


Рис. 3.16. Вигляд робочого простору VTT з активованим модулем голосового та відеозв'язку та додатковими панелями інструментів

На рисунку 3.16 продемонстровано інтегровану роботу модуля голосового та відеозв'язку, який є критично важливим для забезпечення повноцінної живої комунікації між учасниками сесії без необхідності використання сторонніх програмних додатків. Цей компонент побудовано за сучасною технологією WebRTC (Web Real-Time Communication) [13], що дозволяє передавати мультимедійні аудіо- та відеопотоки безпосередньо між браузерами користувачів або через виділений оптимізований медіасервер (Mediasoup) [14]. Такий архітектурний підхід не лише мінімізує мережеву затримку (latency) під час розмови, але й суттєво знижує навантаження на основний WebSocket-сервер платформи, який паралельно продовжує безперебійно обробляти тактичні ігрові події.

На інтерфейсній панелі зв'язку кожен підключений учасник сесії представлений окремою персональною карткою з відображенням живого відеопотоку або ідентифікаційної заглушки з ініціалами у разі вимкненої камери. Для зручності моніторингу на кожній картці передбачено індикатори мовлення та іконки поточного статусу мікрофона. Користувачеві також доступні швидкі елементи управління власним апаратним забезпеченням,

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						73

компактно розташовані у нижній частині віджета: інструменти для деактивації мікрофона, вимкнення веб-камери та кнопка безпечного виходу з активного голосового каналу. Будь-яка з цих дій миттєво змінює локальний стан відповідних медіатреків у браузері та паралельно сповіщає інших учасників віртуальної кімнати про зміну статусу підключення гравця. Завдяки реалізації у вигляді незалежного плаваючого вікна, модуль відеозв'язку можна легко перемістити у будь-яку зручну зону екрана або ж повністю згорнути до заголовка, залишаючи активним лише фоновий звуковий потік для збереження максимальної площі ігрового полотна.

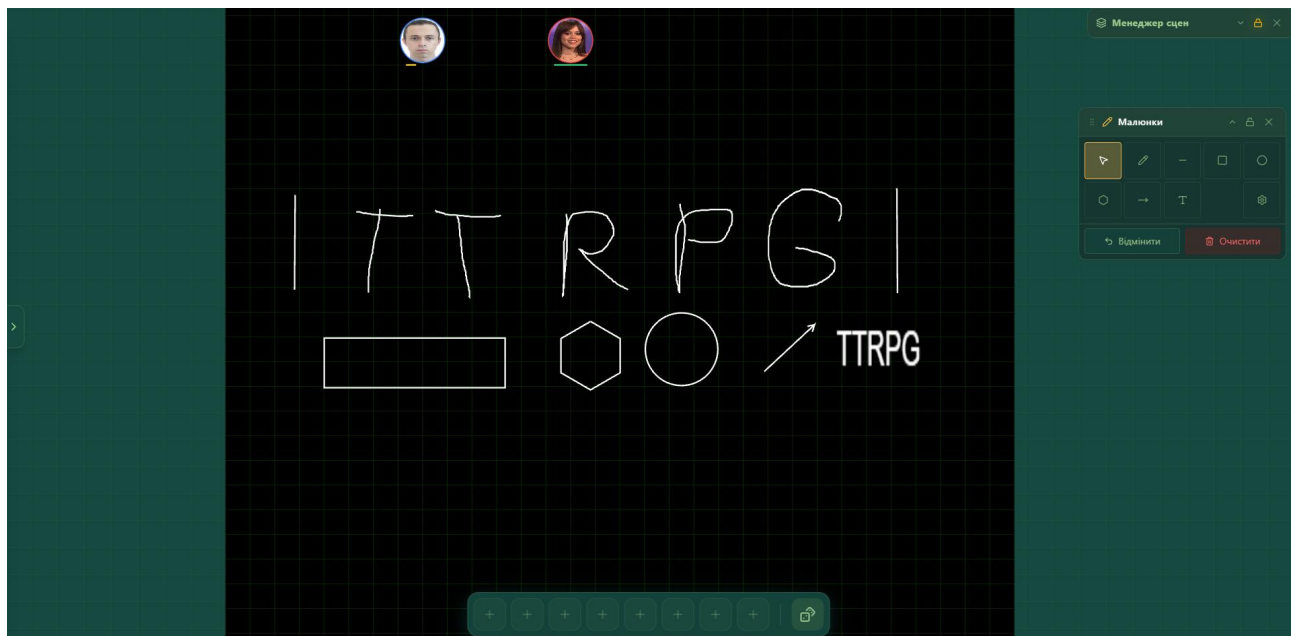


Рис. 3.17. Панель графічних інструментів та результати їх застосування на ігровому полотні

Для швидкого створення тактичних позначок, малювання схематичних планів або візуалізації зон дії магічних заклинань безпосередньо під час ігрової сесії, платформу оснащено спеціалізованим модулем графічної взаємодії (рис. 3.17). Цей інструментарій реалізований у вигляді плаваючої панелі «Малюнки», яка надає користувачам швидкий доступ до базового набору векторних інструментів. Серед доступного функціоналу передбачено вільне малювання пензлем, створення прямих ліній, побудову класичних геометричних фігур (прямокутників, кіл та багатокутників), малювання вказівних стрілок, а також можливість розміщення друкованого тексту. Усі створені графічні елементи

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

динамічно відмальовуються клієнтським рушієм Pixi.js на окремому виділеному шарі сцени. Такий архітектурний поділ гарантує абсолютну незалежність малюнків від базового фонового зображення або шару інтерактивних ігрових токенів. Крім того, графічна панель містить зручні функції скасування останньої дії (Undo) та повного очищення локального шару малюнків, що дозволяє Майстру гри миттєво коригувати тактичну обстановку у разі помилки або зміни ігрової ситуації.

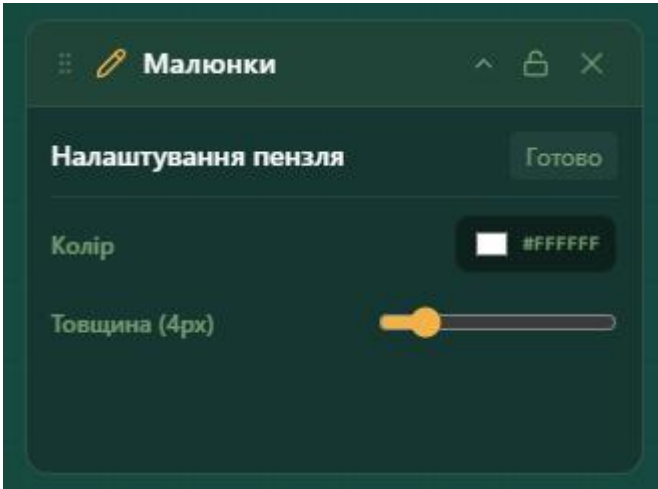


Рис. 3.18. Меню налаштування параметрів пензля для малювання

Для забезпечення більшої гнучкості та виразності при створенні візуальних позначок передбачено додаткове інтерактивне підменю конфігурації графічних інструментів (рис. 3.18). При натисканні на відповідну іконку розгортається вікно «Налаштування пензля», де користувач має змогу детально відкалібрувати візуальні характеристики поточного інструмента. Інтерфейс модуля дозволяє обрати бажаний відтінок у форматі HEX-коду за допомогою вбудованої колірної палітри, а також плавно відрегулювати товщину ліній за допомогою інтерактивного повзунка. Усі модифіковані параметри миттєво фіксуються у локальному стані React-компонента та автоматично застосовуються до всіх наступних згенерованих об'єктів. Завдяки системі синхронізації на базі WebSocket, кожна нова намальована лінія чи фігура у вигляді оптимізованого масиву координат вектора миттєво серіалізується, передається на сервер і реплікується на екранах усіх інших учасників віртуальної кімнати, формуючи ефект повноцінної спільної інтерактивної дошки (Whiteboard).

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

Для прозорого відстеження результатів генерації випадкових чисел у системі реалізовано спеціалізований плаваючий віджет «Журнал кидків» (рис. 3.19). Цей модуль виконує роль глобального агрегатора, що фіксує та деталізує історію всіх кидків віртуальних кубиків, здійснених учасниками під час ігрової сесії. Кожен запис у журналі формується динамічно і містить вичерпну метаінформацію: ідентифікатор користувача, який ініціював дію (наприклад, SEED_GM_ALEX), семантичну назву кидка (наприклад, «Швидкий кидок» або назва конкретної атаки) та точну математичну формулу, за якою здійснювався розрахунок ($2d6+7$ або $20d20$). Підсумковий сумарний результат виводиться великим шрифтом для швидкого зчитування, тоді як нижче розгортається покомпонентна візуалізація кожної окремої гральної кості. Для покращення користувацького досвіду (UX) кожен тип кубика супроводжується власною графічною іконкою, що дозволяє гравцям миттєво розпізнавати складові багатокомпонентних кидків (наприклад, при змішуванні $1d6 + 1d20 + 1d100$).

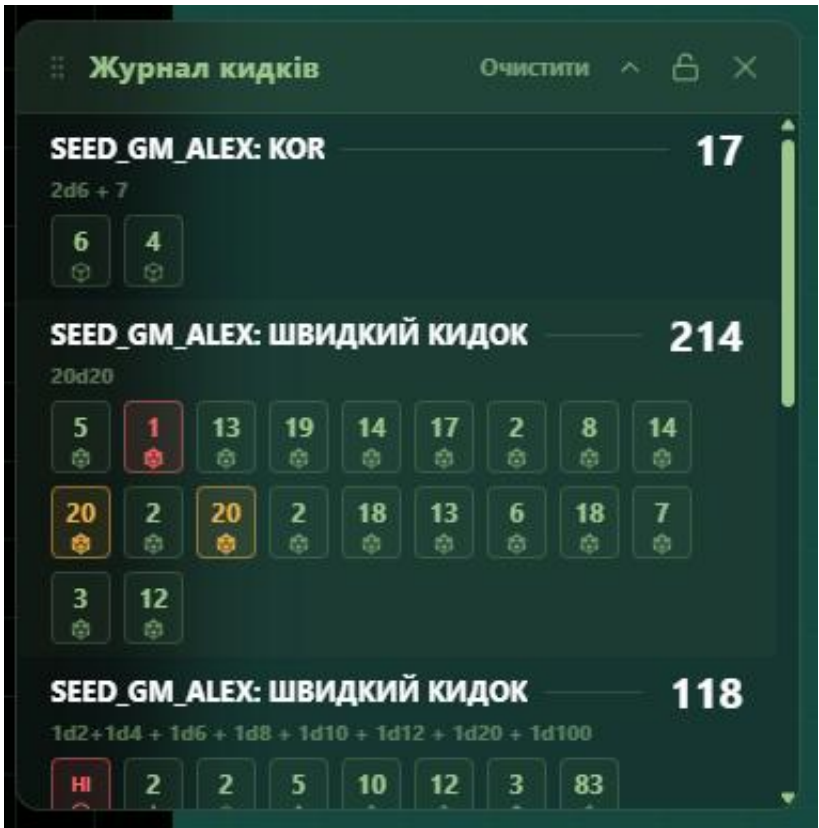


Рис. 3.19. Віджет журналу кидків віртуальних кубиків

Окрему увагу при проєктуванні компонента журналу було приділено інтеграції загальноприйнятих правил настільних рольових ігор, згідно з якими

випадіння максимального або мінімального значення (найчастіше на двадцятигранному кубіку) трактується як критичний успіх або критична невдача відповідно.

Для відображення цього механізму алгоритм рендерингу клієнта перевіряє отримані числові значення та застосовує умовне колірне кодування інтерфейсу: критичний успіх (значення 20) виділяється яскравим золотистим контуром, а критична невдача (значення 1 або мінімальне для даної кості) - акцентним червоним кольором. Це значно прискорює візуальне сприйняття результатів та полегшує роботу Майстра гри при обробці екстремальних ігрових ситуацій. Водночас, з метою оптимізації продуктивності клієнтського додатка та запобігання переповненню оперативної пам'яті надмірною кількістю DOM-елементів, історія журналу управляється алгоритмом черги FIFO (First-In-First-Out) із жорстким обмеженням у 10 останніх записів. При досягненні цього ліміту реєстрація кожного нового кидка кубиків автоматично витісняє найстаріший запис з історії, зберігаючи віджет інформативним, ергономічним та стабільним у роботі.

Для детального контролю над візуальною структурою ігрової локації в системі розроблено плаваючий віджет «Менеджер сцен» (рис. 3.20). Цей інструмент надає Майстру гри розширені можливості для управління ієрархією графічних шарів, на яких базується архітектура відмальовування карти. Кожен композиційний рівень (шар інтерактивних токенів, шар малюнків, тактична сітка та базове фонове зображення) оснащений незалежними індикаторами та елементами управління: кнопкою видимості для приховування секретних зон від гравців та іконкою замка для апаратного блокування статичних об'єктів від випадкових модифікацій.

Безпосередньо з цього віджета адміністратор сесії має змогу оперативно завантажити нове фонове зображення для карти, перевірити поточну роздільну здатність полотна в пікселях, а також ініціювати створення нових шарів чи повністю нових ігрових сцен.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

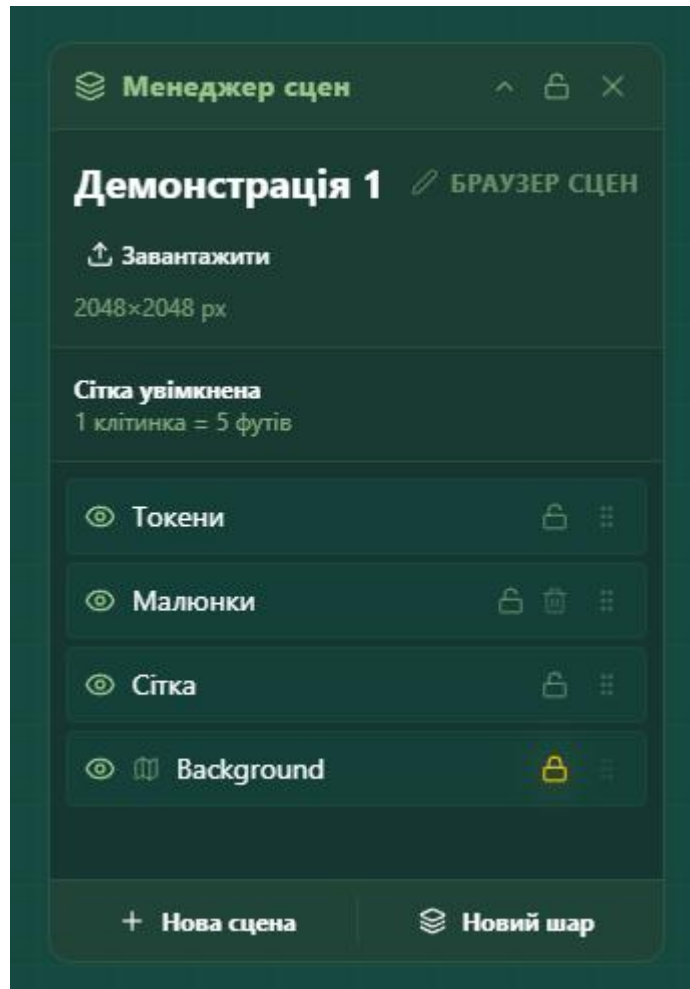


Рис. 3.20. Компонент Менеджер сцен

Глобальна навігація між усіма підготовленими локаціями сюжетної кампанії здійснюється через модуль «Браузер сцен» (рис. 3.21). Інтерфейс цього компонента генерує динамічний перелік усіх доступних карт із чітким візуальним маркуванням їхнього поточного статусу. Це дозволяє Майстру гри миттєво відрізнити активну сцену, яку наразі бачать гравці, від тієї, що перебуває в режимі прихованої підготовки. Важливою UX-особливістю платформи є інтегрований механізм запобігання десинхронізації ігрового процесу: якщо Майстер переходить на іншу сцену для її редагування, у «Менеджері сцен» автоматично активується акцентне системне попередження про те, що гравці фізично знаходяться на іншій локації. За допомогою вбудованої кнопки Майстер може в один клік згенерувати WebSocket-івент, який примусово оновить глобальний стан кімнати та перемкне активну сцену на клієнтських додатках усіх підключених учасників.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

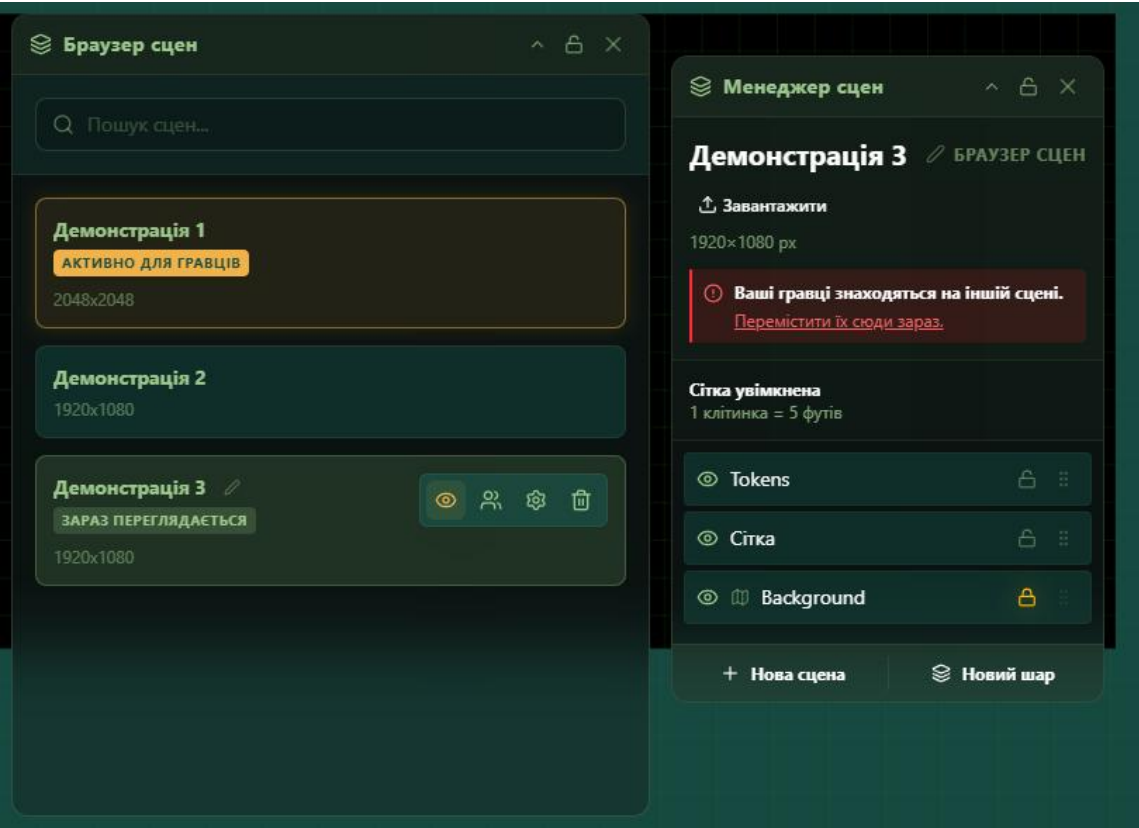


Рис. 3.21. Браузер ігрових сцен та механізм примусової синхронізації гравців

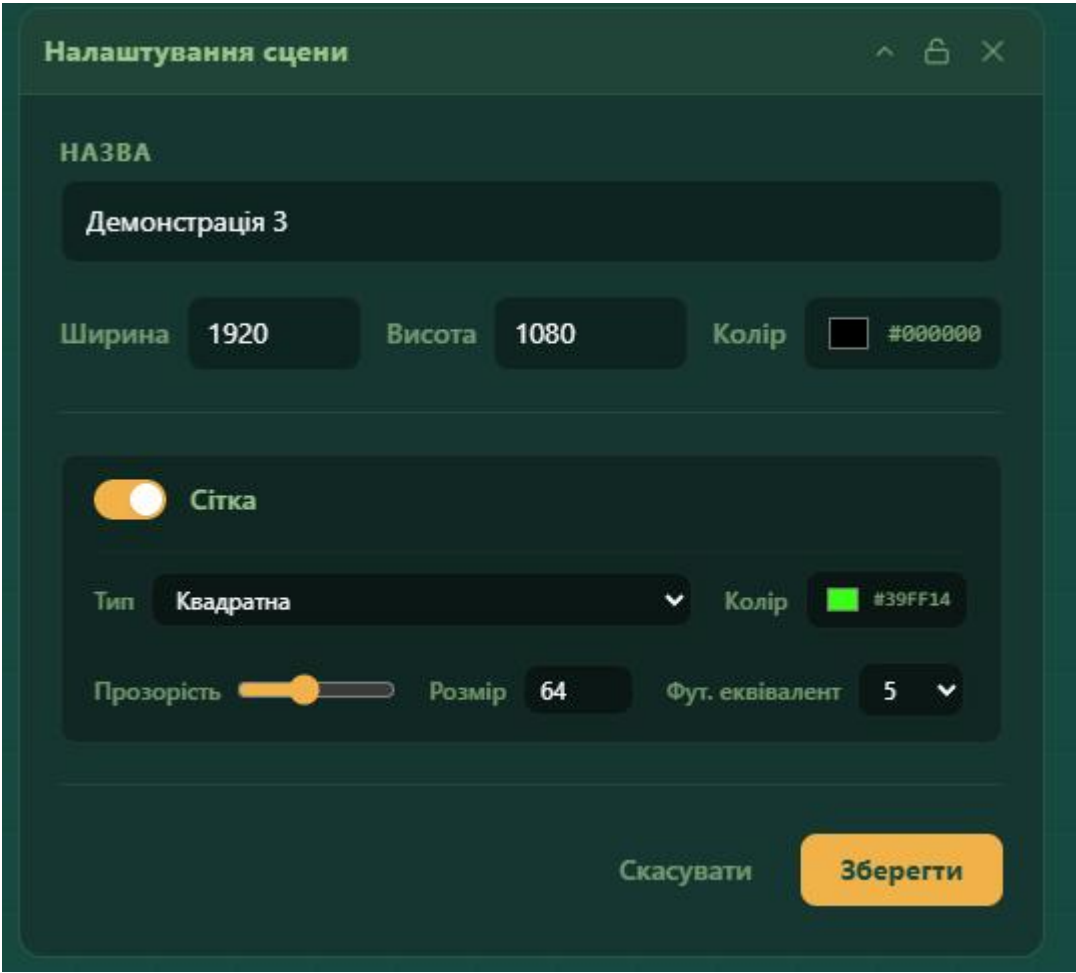


Рис. 3.22. Інтерфейс налаштування візуальних властивостей локації та сітки

Для здійснення точного налаштування фізичних та візуальних параметрів конкретної карти реалізовано спеціалізоване вікно конфігурації сцени (рис. 3.22). У цьому меню адміністратор може задати суворі розміри ігрового полотна (ширину та висоту в пікселях), встановити базовий колір фону у форматі HEX та детально відкалібрувати параметри відображення тактичної сітки. Інтерфейс дозволяє обрати геометричний тип сітки (наприклад, класична квадратна розмітка), налаштувати її колірну гаму, відрегулювати ступінь прозорості ліній за допомогою повзунка та встановити базовий розмір однієї клітинки в пікселях із прив'язкою до ігрового еквівалента відстані (стандартні 5 футів). Клієнтський додаток автоматично валідує введені дані, оновлює запис сцени в базі даних та ініціює миттєве перемальовування координатної сітки рушієм Pixi.js без необхідності перезавантаження сторінки.

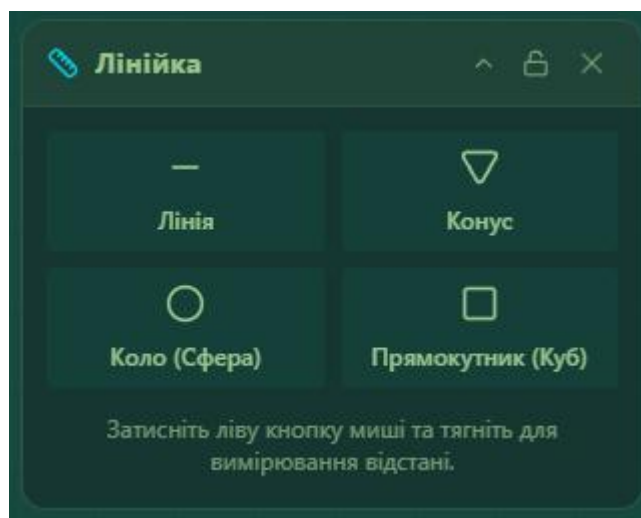


Рис. 3.23. Інтерфейс віджета «Лінійка» для вимірювання відстаней та площ

Для точного просторового орієнтування та дотримання правил тактичного бою платформу оснащено інтерактивним інструментом вимірювання - «Лінійкою» (рис. 3.23). У настільних рольових іграх обчислення дистанцій є критично важливим аспектом для визначення дальності переміщення персонажів, радіуса дії магічних заклинань чи площі ураження зброєю (Area of Effect). Відповідний плаваючий віджет надає користувачам доступ до чотирьох ключових геометричних шаблонів, що покривають базові потреби більшості рольових систем: пряма лінія, конус, коло (сфера) та прямокутник (куб). Взаємодія з модулем є максимально інтуїтивною -

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

користувачу достатньо обрати бажану форму, затиснути ліву кнопку миші на стартовій точці ігрового полотна та потягнути курсор для динамічного розгортання зони вимірювання.

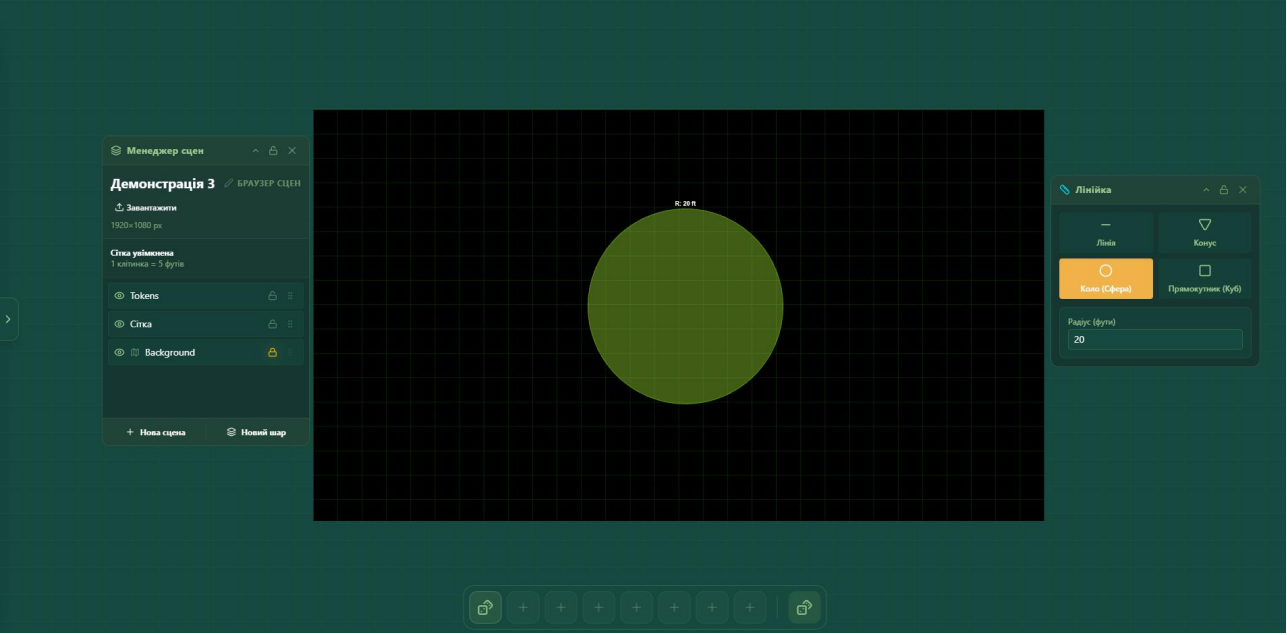


Рис. 3.24. Візуалізація інструмента Коло (Сфера)

Математичне ядро інструмента тісно інтегроване з параметрами поточної сцени, що забезпечує автоматичну конвертацію піксельних координат у внутрішньоігрові одиниці виміру. Як показано на рисунку 3.24, при застосуванні шаблону «Коло» на стандартній квадратній сітці, система динамічно відмальовує напівпрозору зону радіусом 20 футів, чітко відображаючи цей числовий показник над фігурою. Графічний рушій Pixi.js у режимі реального часу оновлює координати полігонів під час руху курсора, а бізнес-логіка клієнта зчитує масштаб сцени (наприклад, 1 клітинка еквівалентна 5 футам) та математично перетворює фізичний вектор на екрані у зрозумілий ігровий результат. Окрім інтерактивного розтягування мишею, віджет підтримує апаратне введення точного значення радіуса через числове поле, що гарантує ідеальну геометричну побудову зони заклинання без похибок.

Гнучкість обчислювальних алгоритмів лінійки також повністю адаптована під різні топології ігрового поля. На рисунку 3.25 продемонстровано роботу базового інструмента «Лінія» на гексагональній (шестикутній) сітці, де масштаб штучно змінено на 15 футів за одну клітинку.

При проведенні вектора переміщення система коректно розраховує перетин граней гексів і виводить підсумкову дистанцію у 60 футів. Повна автоматизація просторових обчислень позбавляє гравців та Майстра гри необхідності вручну рахувати клітинки на екрані, що суттєво зменшує когнітивне навантаження та прискорює загальну динаміку ігрового процесу. Як і інші взаємодії у середовищі VTT, процес побудови вимірювальних фігур реплікується через WebSocket-з'єднання, дозволяючи всім учасникам віртуальної кімнати одночасно спостерігати за тактичним плануванням колег.

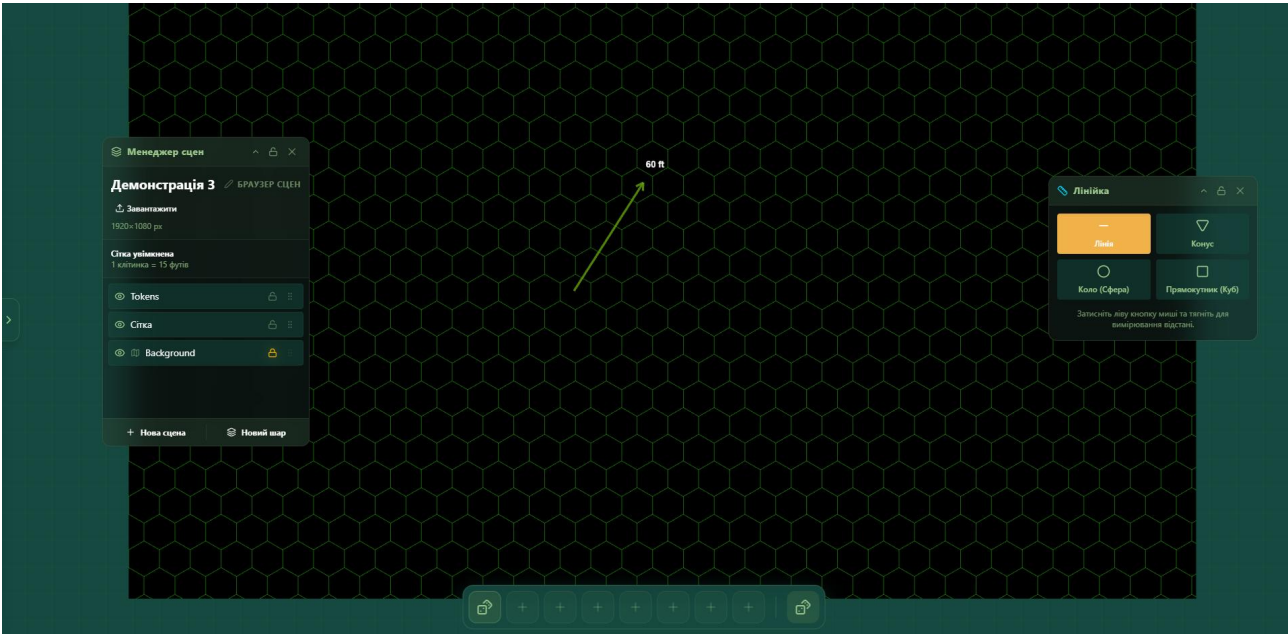


Рис. 3.25. Візуалізація інструмента Лінія



Рис. 3.26. Панель швидких та збережених кидків у розгорнутому стані

Для швидкої та зручної генерації випадкових чисел у системі передбачено спеціалізований інструмент - бар швидких кидків кубиків (рис. 3.26). Цей віджет ергономічно розташований у нижній частині екрана та спроектований із функцією згортання, що дозволяє за потреби приховувати його за межі робочого простору для максимального звільнення площі тактичного полотна. Архітектура бару логічно поділена на два основні функціональні модулі:

базовий конструктор для створення одноразових швидких кидків (ліва кнопка) та панель порожніх слотів для конфігурації і збереження сталих кидків, які гравець використовує найчастіше під час сесії.

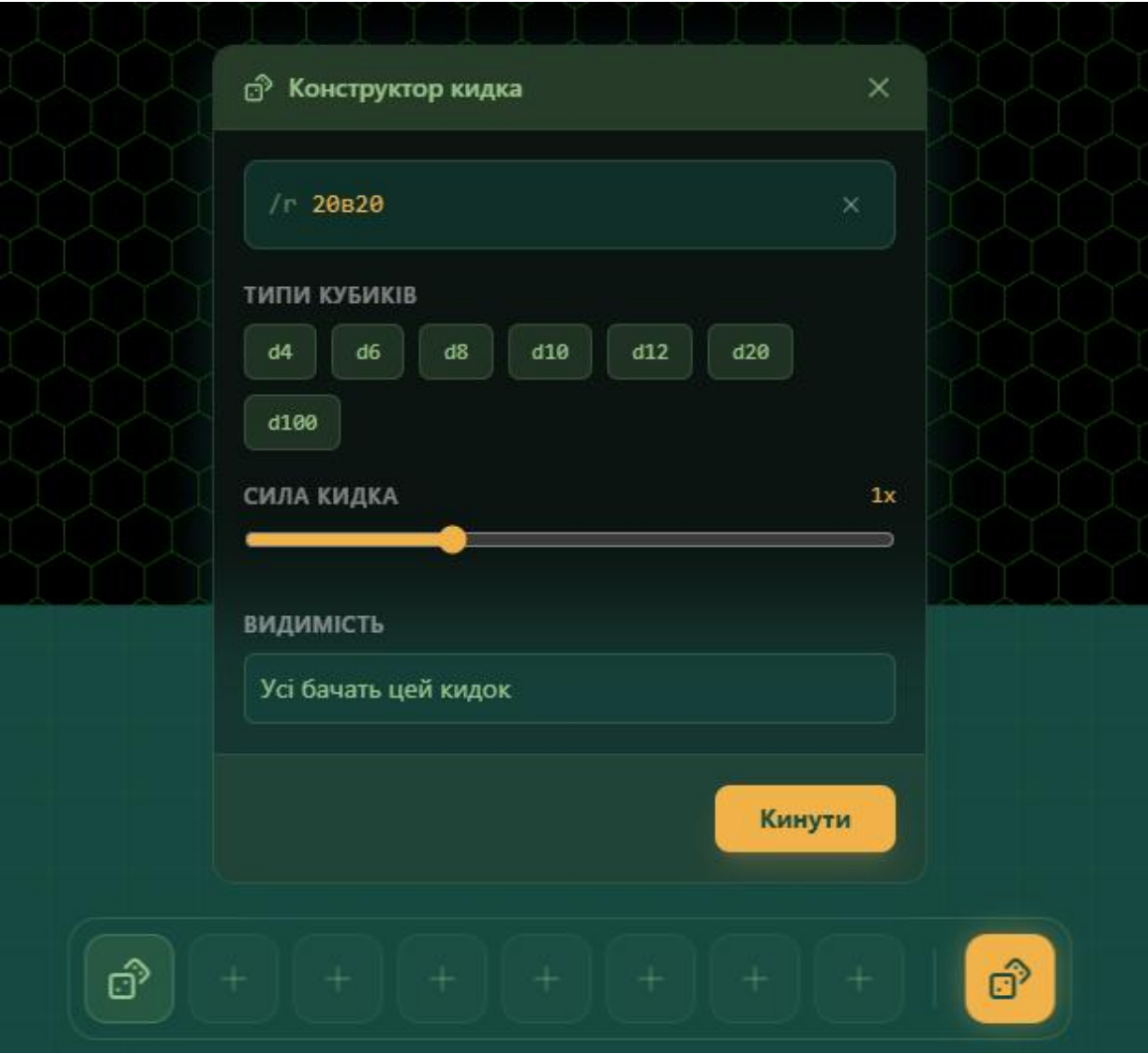


Рис. 3.27. Інтерфейс конструктора швидкого кидка

Конструктор швидкого кидка (рис. 3.27) дозволяє користувачу гнучко налаштовувати незаплановані або одноразові формули кидка. Окрім вибору гральних костей, інтерфейс надає змогу регулювати фізичну силу кидка за допомогою відповідного повзунка, що безпосередньо впливає на початкову швидкість та імпульс падіння 3D-моделей кубиків на ігровій сцені. Також реалізовано гнучкий контроль приватності: користувач може налаштувати видимість результату, обравши публічний кидок (доступний усім) або прихований (видимий виключно Майстру гри). Важливою інженерною особливістю цього модуля є розробка власного лексичного парсера для обробки

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

ручного введення формул. Оскільки в різних мовних традиціях настільних ігор гральні кості позначаються різними літерами (англійське «d» від dice, українське «к» від кубик), користувачі часто допускають синтаксичні помилки. Розроблений парсер алгоритмічно розпізнає ці синоніми та автоматично виправляє типові помилки розкладки клавіатури. Наприклад, якщо користувач забув змінити мовну розкладку і ввів «2в6» замість «2d6» (оскільки кирилична «в» та латинська «d» знаходяться на одній клавіші), або ввів «2r6» замість «2к6», алгоритм автоматично нормалізує рядок до валідного системного формату перед виконанням кидка, що критично знижує кількість помилок та покращує користувацький досвід (UX).

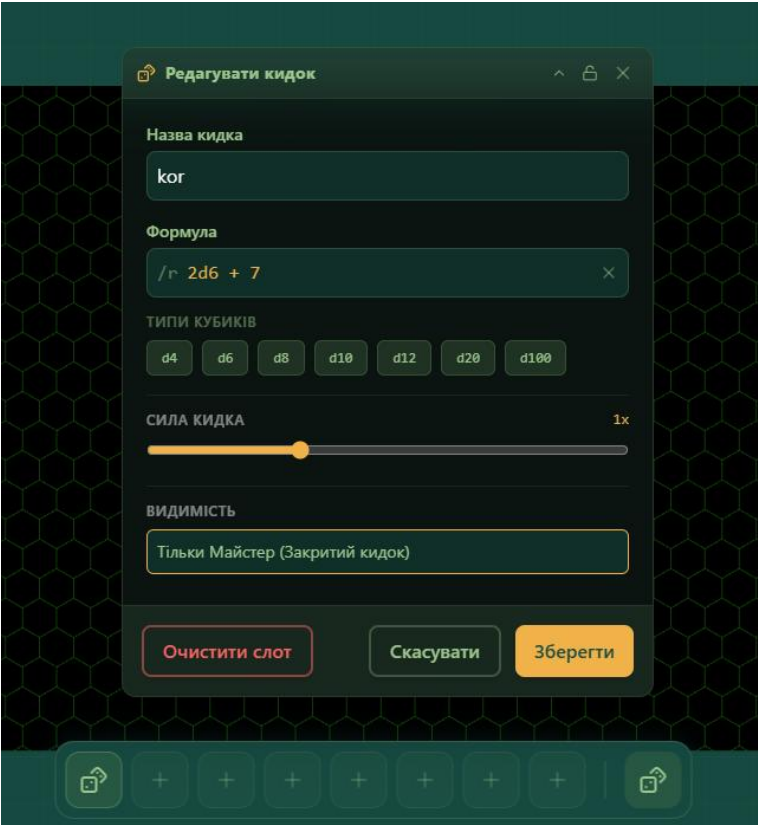


Рис. 3.28. Меню редагування збереженого слота кидка з можливістю конфігурації формули

Для оптимізації ігрового процесу та автоматизації рутини гравці можуть зберігати часто використовувані формули (наприклад, атаки основною зброєю) у спеціальні вільні слоти на панелі. При натисканні правою кнопкою миші на вже збережений слот система перехоплює контекстну подію та відкриває модальне вікно редагування (рис. 3.28). У цьому меню користувач має

можливість перейменувати дію (наприклад, вказати назву зброї), відредагувати саму формулу, а також налаштувати індивідуальні параметри видимості для конкретної комірки (зокрема, встановити статус «Тільки Майстер (Закритий кидок)»). Усі конфігурації збережених слотів серіалізуються та кешуються у локальному сховищі браузера (LocalStorage) за допомогою механізму персистентності менеджера стану Zustand. Дані структуруються з прив'язкою до ідентифікатора поточної ігрової сесії, що гарантує повне збереження індивідуальних налаштувань інтерфейсу після перезавантаження сторінки або тимчасової втрати з'єднання, при цьому не створюючи надлишкового навантаження на основну базу даних платформи.



Рис. 3.29. Комплексна схематична симуляція бойової ситуації з інтегрованим використанням модулів віртуального ігрового столу

На рисунку 3.29 представлено комплексну схематичну симуляцію типової ігрової ситуації, яка наочно демонструє безшовну інтеграцію та паралельну роботу всіх розроблених модулів віртуального ігрового столу під час бойового зіткнення. На деталізованому фоновому зображенні локації розміщено графічні токени ігрових персонажів та супротивників, переміщення яких відслідковується клієнтським рушієм та синхронізується з сервером. Водночас трекер ініціативи автоматизує управління черговістю ходів учасників, а інтегрований 3D-рушій у реальному часі генерує фізичну симуляцію падіння

гральних кубиків безпосередньо поверх графічної карти під час заявленої атаки. Підсумковий результат математичного розрахунку миттєво виводиться у вигляді центрального інформаційного спливаючого вікна та паралельно фіксується у глобальному журналі кидків для збереження історії дій. Така глибока архітектурна узгодженість незалежних плаваючих віджетів та ядра Pixi.js забезпечує користувачам цілісний, інтуїтивно зрозумілий та стабільний ігровий досвід без необхідності залишати єдину сторінку вебдодатка.

3.3 Тестування платформи

Розробка складної клієнт-серверної архітектури віртуального ігрового столу (VTT), що функціонує в режимі реального часу, вимагає застосування комплексного підходу до забезпечення якості програмного забезпечення (Quality Assurance). Оскільки платформа одночасно обробляє множинні WebSocket-з'єднання, здійснює рендеринг складної 2D/3D графіки за допомогою рушіїв Pixi.js та Three.js, а також підтримує передачу медіапотоків через WebRTC, виникнення помилок у коді може призвести до критичних збоїв у синхронізації ігрового процесу. Для мінімізації технічних ризиків та гарантування безперебійної роботи системи було впроваджено багаторівневу стратегію тестування та автоматизованої інспекції вихідного коду. Ця стратегія охоплює етапи від превентивного статичного аналізу на етапі розробки до глибокого інтеграційного та End-to-End тестування (Playwright) [24] з використанням сучасних Continuous Integration (CI) пайплайнів на базі інфраструктури GitLab.

Першим та найбільш раннім рівнем забезпечення якості в життєвому циклі розробки вебплатформи є статичний аналіз вихідного коду (Linting). Для клієнтської частини застосунку, побудованої на компонентній базі React, було інтегровано інструмент ESLint [23] із набором спеціалізованих конфігураційних плагінів. Головною метою використання цього аналізатора є не лише підтримання єдиного стандартизованого стилю написання коду серед розробників, але й превентивне виявлення потенційних логічних вразливостей ще до етапу компіляції чи тестування в браузері. Наприклад, використання

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

модуля «eslint-plugin-react-hooks» є критично важливим для коректної роботи менеджера станів Zustand та контролю життєвого циклу плаваючих віджетів. Цей плагін алгоритмічно перевіряє правильність використання хуків (зокрема useEffect) та суворо контролює масиви їхніх залежностей. Відсутність такого контролю в динамічному VTT-додатку найчастіше призводить до зайвих циклів рендерингу ігрового полотна або до виникнення витоків пам'яті (memory leaks), коли розірвані WebSocket-підписки продовжують накопичуватися у фоновому режимі браузера.

Паралельно з цим, для забезпечення інклюзивності інтерфейсу застосовується аналізатор «jsx-a11y», який сканує DOM-структуру на відповідність стандартам доступності (Accessibility), гарантуючи коректну взаємодію з інтерфейсними елементами. Серверна частина платформи також проходить етап суворого статичного контролю за допомогою спеціалізованих скриптів лінтингу, розгорнутих у середовищі Node.js. Завдяки налаштованим CI/CD сценаріям у файлі конфігурації GitLab, кожне нове надходження коду до сховища (коміт) автоматично ініціює фоновий запуск процесів перевірки (рис. 3.30). Якщо інструмент виявляє відхилення від прийнятих архітектурних конвенцій або наявність синтаксичних антипатернів, процес злиття гілок (merge) блокується системою до моменту ручного виправлення помилок розробником. Такий жорсткий превентивний підхід гарантує абсолютну чистоту кодової бази та суттєво зменшує загальний обсяг технічного боргу проєкту ще до початку написання модульних тестів.

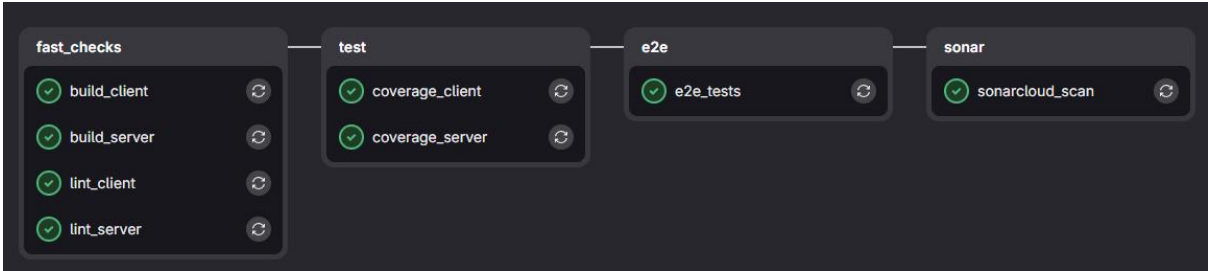


Рис. 3.30. Автоматизований запуск процесів статичного аналізу коду (Linting) у середовищі GitLab CI

Наступним рівнем забезпечення надійності програмного продукту є безперервна інспекція якості вихідного коду (Continuous Inspection) та

автоматизований статичний аналіз безпеки застосунків (Static Application Security Testing - SAST). З цією метою до інфраструктури проєкту було інтегровано аналітичну платформу Sonar [22], яка здійснює глибокий семантичний аналіз кодової бази після кожного оновлення репозиторію. Інтеграція інструмента реалізована через конфігураційний файл і повністю автоматизована в рамках конвеєра GitLab CI/CD. Алгоритми платформи паралельно сканують як серверну (Node.js), так і клієнтську (React) частини застосунку. При цьому директорії зі згенерованими білдами та тестовими модулями виключаються з процесу перевірки для забезпечення максимальної репрезентативності фінальних метрик.

Ключовою перевагою використання Sonar є його здатність не лише виявляти локальні синтаксичні помилки, але й об'єктивно оцінювати загальний архітектурний стан проєкту за допомогою метрики «Технічний борг» (Technical Debt). Платформа математично розраховує приблизний час, необхідний для виправлення всіх знайдених архітектурних недоліків, що дозволяє розробникам раціонально планувати етапи рефакторингу. Під час аналізу система класифікує виявлені проблеми на декілька категорій: «Запахи коду» (Code Smells), які вказують на заплутану або важкопідтримувану бізнес-логіку, «Баги» (Bugs), що позначають очевидні помилки під час виконання, та «Вразливості» (Vulnerabilities) (рис. 3.31). Особлива увага приділяється пошуку вразливостей (наприклад, потенційних ін'єкцій або небезпечної обробки JWT-токенів [18]), оскільки VTT-платформа оперує персональними даними користувачів та використовує бази даних.

Окрім безпосереднього аналізу якості вихідного коду, платформа збирає та візуалізує результати роботи тестових фреймворків. За допомогою вбудованих парсерів Sonar імпортує згенеровані звіти про покриття коду тестами у форматі «Icov», які автоматично формуються після завершення стадії модульного тестування на клієнті та бекенді. Це дозволяє в режимі єдиного інформаційного вікна відстежувати не лише якість написаної логіки, але й відсоток її захищеності тестами. Завдяки налаштуванню бар'єрів якості (Quality

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

Gates), система здатна автоматично блокувати розгортання релізу (Deployment), якщо поточний коміт містить критичні вразливості або знижує загальне покриття коду тестами нижче допустимої норми. Такий підхід трансформує процес QA з епізодичних перевірок у безперервний автоматизований контроль.

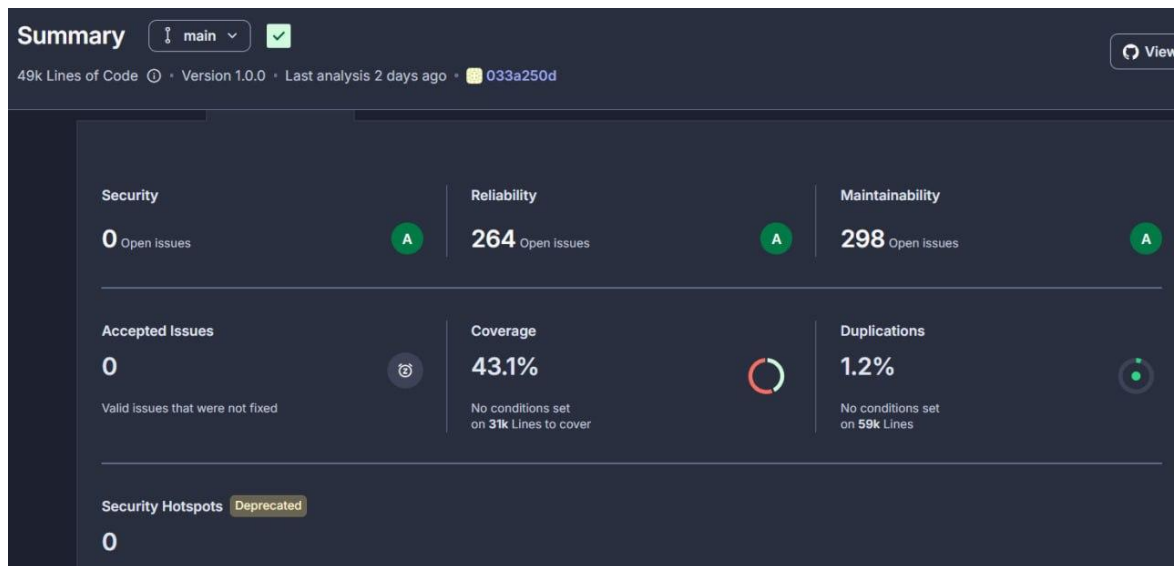


Рис. 3.31. Звіт автоматизованого сканування якості коду та безпеки архітектури на платформі Sonar

Оскільки платформа побудована за архітектурою модульного моноліту, вона поєднує як механізми реального часу (WebSockets), так і розгалужене REST API [27], яке відповідає за організаційний контур: управління профілями, кампаніями, сесіями та авторизацією. Важливим етапом тестування та супроводу такої архітектури є забезпечення актуальної та точної документації для всіх мережевих маршрутів. Ручне тестування цих маршрутів є необхідним кроком перед їхньою інтеграцією в React-застосунок.

Для вирішення цього завдання у проєкті впроваджено галузевий стандарт OpenAPI [21] за допомогою екосистеми інструментів Swagger (зокрема пакетів `swagger-jsdoc` та `swagger-ui-express`). Такий підхід дозволив автоматично генерувати модульну документацію для всіх підсистем моноліту на основі структурованих YAML-файлів, розбитих по предметних областях (маршрути сесій, кампаній, чатів, VTT тощо).

Головною перевагою використання Swagger у контексті забезпечення якості є те, що він виступає не лише статичним довідником, але й повноцінним

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

середовищем для функціонального тестування. Через графічний веб-інтерфейс Swagger UI розробник має змогу формувати HTTP-запити до серверних ендпоінтів безпосередньо в браузері, передавати тіло запиту (body), змінні параметри маршрутів та миттєво отримувати відповіді бази даних у форматі JSON. Окрім валідації структури даних, Swagger значно спрощує тестування систем безпеки (рис. 3.32). Його інтерфейс підтримує вбудовану конфігурацію авторизації на основі файлів cookie (cookieAuth). Це дозволяє безпечно тестувати ендпоінти від імені конкретного користувача (наприклад, перевіряти права доступу Майстра чи Гравця) без необхідності налаштовувати сторонні програми-клієнти на кшталт Postman. Використання єдиної задокументованої схеми запитів (Single Source of Truth) мінімізує ризик розбіжностей під час інтеграції модулів між різними розробниками та суттєво прискорює процес загального тестування системи.

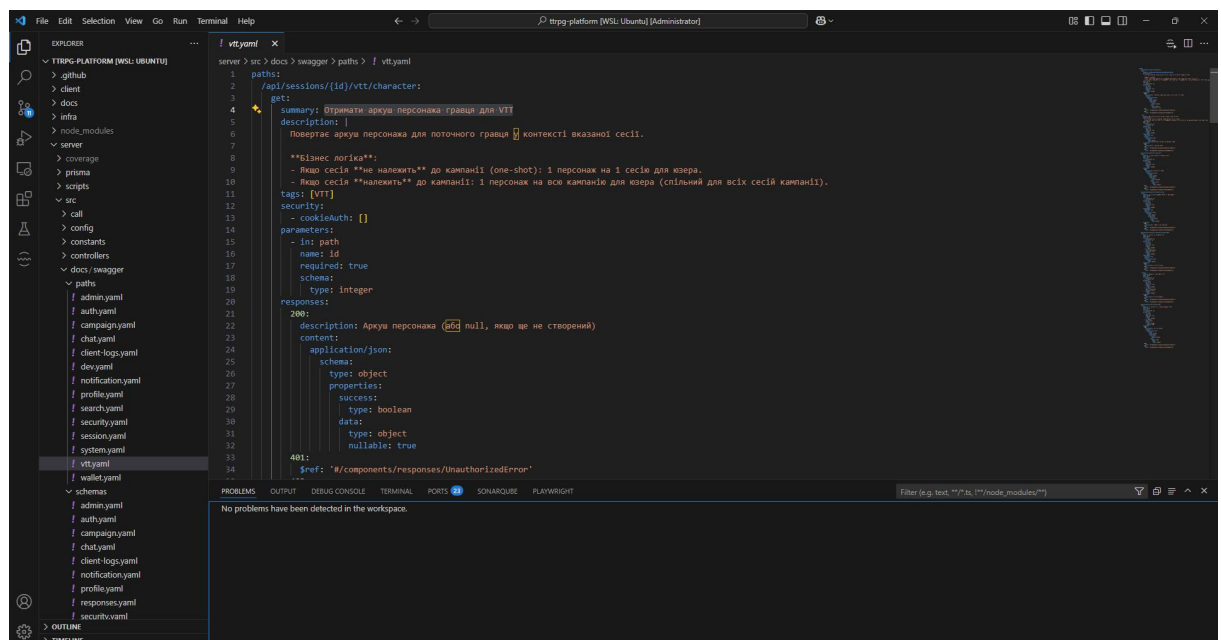


Рис. 3.32. Структура директорій та фрагмент коду специфікації OpenAPI для документування маршрутів платформи

Для практичної реалізації описаного підходу та прискорення процесу функціонального тестування, безпосередньо у веб-інтерфейсі Swagger UI було імплементовано спеціалізований інструмент для розробників. На рисунку 3.33 наведено результат виконання запиту у цьому графічному середовищі, що наочно демонструє інтерактивну взаємодію з API платформи.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

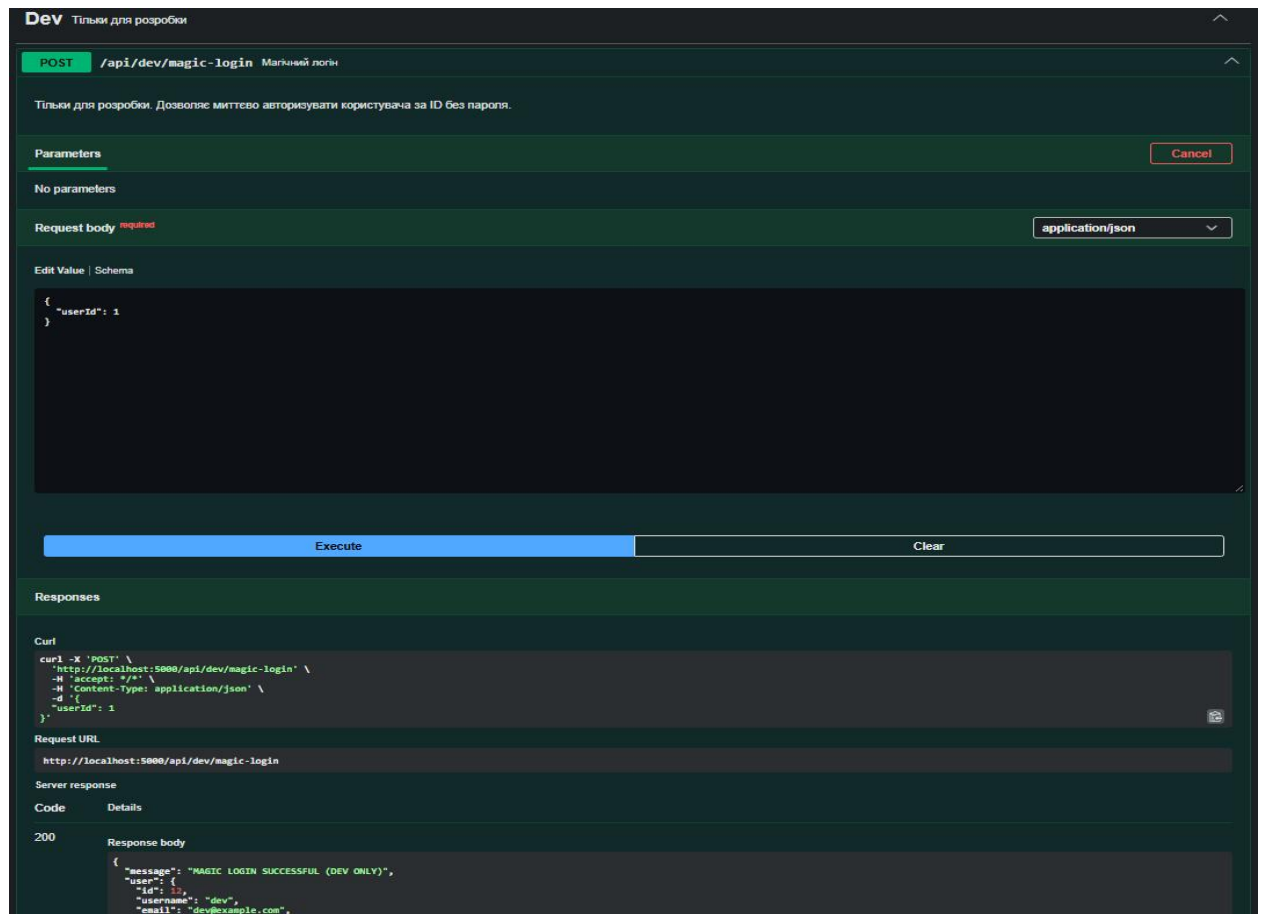


Рис. 3.33. Виконання тестового запиту в інтерфейсі Swagger UI

Як видно з рисунка 3.33, тестування системи суттєво оптимізовано завдяки впровадженню так званого «магічного» маршруту розробника (POST /api/dev/magic-login). Цей ендпоінт дозволяє обійти складну бізнес-логіку: замість проходження повного циклу реєстрації та підтвердження електронної пошти, інженер може одним кліком згенерувати тестового користувача й автоматично встановити необхідні сесійні JWT-токени безпосередньо у свій браузер.

Важливою архітектурною деталлю є суворі ізоляція цього маршруту на рівні серверного контролера. Ендпоінт захищений програмним блокуванням і примусово повертає помилку доступу (статус 404), якщо сервер працює у продуктовому середовищі (NODE_ENV !== 'development') або якщо системний прапорець ENABLE_DEV_AUTH деактивовано. Такий підхід гарантує абсолютну безпеку бази даних платформи при публічному розгортанні, одночасно забезпечуючи максимальний рівень комфорту під час локальної розробки та налагодження (Developer Experience).

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи детально описано процеси розгортання, організації користувацького інтерфейсу та комплексного тестування створеного візуального модуля віртуального ігрового столу (VTT). Продемонстровано практичне втілення спроектованої архітектури в умовах реального навантаження, що підтвердило життєздатність обраного технологічного стека та готовність платформи до безперебійного використання ігровими спільнотами.

Під час налаштування параметрів платформи обґрунтовано використання технологій Docker та Docker Compose, що забезпечило абсолютну ідентичність середовища розробки у підсистемі WSL із цільовим сервером. Налаштовано Nginx-контейнер для підтримки довготривалих WebSocket-з'єднань, а також організовано маршрутизацію портів і розгорнуто TURN-сервери для стабільного WebRTC-зв'язку між гравцями.

В огляді структури інтерфейсу продемонстровано переваги концепції «плаваючих віджетів» (Floating UI). Цей підхід забезпечує ергономічність, дозволяючи Майстру гри керувати сценами та трекером ініціативи, не перекриваючи тактичну карту. Інтерфейс суворо розмежований за рольовими правами: гравці мають доступ лише до власних аркушів, віртуальної лінійки, інструментів малювання та бару кидків 3D-кубиків, результати яких фіксуються у спільному журналі.

Коректність та безпека візуального модуля підтверджені розробкою багаторівневої стратегії тестування (Quality Assurance). Превентивний контроль кодової бази забезпечено аналізатором ESLint, а глибоку інспекцію технічного боргу виконано на платформі SonarQube у межах конвеєрів GitLab CI. Надійність серверної частини доведено серією модульних тестів (Node.js Test Runner). Крім того, функціональність REST-інфраструктури успішно верифіковано в інтерактивному середовищі Swagger UI за допомогою спеціалізованого «магічного» маршруту розробника, що суттєво оптимізувало перевірку безпеки системи.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну науково-прикладну задачу проєктування та програмної реалізації візуального модуля (віртуального ігрового столу) у складі вебплатформи для дистанційного проведення настільних рольових ігор. У результаті виконання роботи повністю досягнуто поставленої мети та отримано низку вагомих науково-практичних результатів.

На основі глибокого аналізу предметної області та комерційних аналогів (Roll20, Foundry VTT, Owlbear Rodeo) виявлено їхні системні недоліки. Зокрема, ідентифіковано проблеми щодо низької продуктивності графічного рендерингу великих карт, нестачі інструментів автоматизації рутини та перевантаженості інтерфейсу. За результатами дослідження сформовано перелік вимог до проєктуваної підсистеми, що дозволило уникнути типових архітектурних помилок.

Відповідно до визначених вимог обґрунтовано вибір гібридної клієнт-серверної архітектури за шаблоном модульного моноліту. Для забезпечення максимальної інтерактивності клієнтського застосунку (SPA) обрано сучасну екосистему на базі бібліотеки React та стейт-менеджера Zustand. Це дозволило уникнути надлишкових перемальовувань DOM-дерева під час динамічних подій. Для серверної частини застосовано подієво-орієнтоване середовище Node.js із використанням надійної реляційної СУБД PostgreSQL для збереження персистентних даних.

Ключовим практичним здобутком стала розробка інноваційної підсистеми відображення графіки на базі апаратно-прискорених рушіїв. Використання технологій WebGL через абстракції Pixi.js для 2D-рендерингу багатошарової тактичної карти, сітки та інструментів малювання у поєднанні з рушієм Three.js для 3D-симуляції фізики кидків кубиків кардинально змінило підхід до обробки графіки. Це рішення дозволило перенести основне обчислювальне навантаження з сервера на відеоадаптери клієнтських пристроїв, що гарантує плавну зміну кадрів навіть за великої кількості активних токенів.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		93

Паралельно з удосконаленням візуальної частини розроблено оптимізований механізм просторової синхронізації ігрового стану. Інтеграція повнодуплексних каналів зв'язку на базі WebSocket забезпечила мінімальні мережеві затримки під час переміщення ігрових фішок, оновлення бойового трекера ініціативи та багатокористувацької взаємодії. Серверний маршрутизатор подій оптимізовано для трансляції виключно тих змін, що стосуються локального контексту поточної сесії, що суттєво знизило мережеве навантаження.

Важливим етапом роботи стало проєктування ергономічного інтерфейсу платформи за концепцією "плаваючих віджетів" (Floating UI). Цей підхід надав користувачам унікальну можливість вільно конфігурувати власний робочий простір. До системи успішно інтегровано комплекс допоміжних механік: динамічні аркуші персонажів, інструменти вимірювання дистанцій (геометрична лінійка), а також інтегрований модуль зв'язку на базі WebRTC для забезпечення повноцінного живого спілкування.

Надійність, безпеку та стабільність розробленої архітектури підтверджено практичним впровадженням комплексної стратегії тестування (QA). Програмний код пройшов перевірку інструментами статичного аналізу (ESLint) та глибоку інспекцію на наявність вразливостей і технічного боргу на платформі Sonar. Коректність бізнес-логіки доведено серією ізольованих модульних тестів за допомогою Node.js Test Runner, а взаємодія з базою даних повноцінно задокументована та протестована в інтерактивному середовищі Swagger UI.

Таким чином, у ході виконання роботи створено стабільний, масштабований та високопродуктивний програмний продукт. Розроблений модуль повністю усуває ключові недоліки існуючих рішень і готовий до практичного застосування ігровими спільнотами, забезпечуючи максимально комфортний досвід проведення дистанційних рольових ігор.

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
2. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003. 560 p.
3. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.
4. React - A JavaScript library for building user interfaces [Електронний ресурс]. - Режим доступу: <https://react.dev/> (дата звернення: 10.05.2026).
5. Zustand - Bear necessities for state management in React [Електронний ресурс]. - Режим доступу: <https://zustand-demo.pmnd.rs/> (дата звернення: 11.05.2026).
6. TanStack Query: Powerful asynchronous state management [Електронний ресурс]. - Режим доступу: <https://tanstack.com/query/latest> (дата звернення: 12.05.2026).
7. PixiJS - The HTML5 Creation Engine [Електронний ресурс]. - Режим доступу: <https://pixijs.com/> (дата звернення: 14.05.2026).
8. Three.js - JavaScript 3D Library [Електронний ресурс]. - Режим доступу: <https://threejs.org/> (дата звернення: 15.05.2026).
9. WebGL API - MDN Web Docs [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (дата звернення: 15.05.2026).
10. Node.js Documentation [Електронний ресурс]. - Режим доступу: <https://nodejs.org/en/docs/> (дата звернення: 16.05.2026).
11. Express - Node.js web application framework [Електронний ресурс]. - Режим доступу: <https://expressjs.com/> (дата звернення: 16.05.2026).
12. The WebSocket API (WebSockets) [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 17.05.2026).

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

13. WebRTC API - MDN Web Docs [Электронный ресурс]. - Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API (дата звернення: 18.05.2026).
14. Mediasoup - Cutting Edge WebRTC Video Conferencing [Электронный ресурс]. - Режим доступа: <https://mediasoup.org/> (дата звернення: 19.05.2026).
15. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс]. - Режим доступа: <https://www.postgresql.org/> (дата звернення: 20.05.2026).
16. Prisma - Next-generation Node.js and TypeScript ORM [Электронный ресурс]. - Режим доступа: <https://www.prisma.io/> (дата звернення: 21.05.2026).
17. Redis - The open source, in-memory data store [Электронный ресурс]. - Режим доступа: <https://redis.io/> (дата звернення: 22.05.2026).
18. JSON Web Tokens (JWT) [Электронный ресурс]. - Режим доступа: <https://jwt.io/> (дата звернення: 23.05.2026).
19. Docker Documentation [Электронный ресурс]. - Режим доступа: <https://docs.docker.com/> (дата звернення: 24.05.2026).
20. NGINX Reverse Proxy [Электронный ресурс]. - Режим доступа: <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/> (дата звернення: 25.05.2026).
21. OpenAPI Specification [Электронный ресурс]. - Режим доступа: <https://swagger.io/specification/> (дата звернення: 26.05.2026).
22. SonarQube - Static Code Analysis [Электронный ресурс]. - Режим доступа: <https://docs.sonarqube.org/> (дата звернення: 27.05.2026).
23. ESLint - Find and fix problems in your JavaScript code [Электронный ресурс]. - Режим доступа: <https://eslint.org/> (дата звернення: 28.05.2026).
24. Playwright - Fast and reliable end-to-end testing [Электронный ресурс]. - Режим доступа: <https://playwright.dev/> (дата звернення: 29.05.2026).

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

25. Tailwind CSS - Rapidly build modern websites [Електронний ресурс]. - Режим доступу: <https://tailwindcss.com/> (дата звернення: 30.05.2026).
26. Vite - Next Generation Frontend Tooling [Електронний ресурс]. - Режим доступу: <https://vitejs.dev/> (дата звернення: 01.06.2026).
27. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. dissertation. University of California, Irvine, 2000. 162 p.
28. Dungeons & Dragons Official Website [Електронний ресурс]. - Режим доступу: <https://dnd.wizards.com/> (дата звернення: 02.06.2026).
29. Roll20: Online virtual tabletop for pen and paper RPGs [Електронний ресурс]. - Режим доступу: <https://roll20.net/> (дата звернення: 03.06.2026).
30. Foundry Virtual Tabletop [Електронний ресурс]. - Режим доступу: <https://foundryvtt.com/> (дата звернення: 03.06.2026).
31. Owlbear Rodeo: A Virtual Tabletop for Tabletop RPGs [Електронний ресурс]. - Режим доступу: <https://www.owlbear.rodeo/> (дата звернення: 04.06.2026).
32. Single-page application (SPA) architecture patterns [Електронний ресурс]. - Режим доступу: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (дата звернення: 05.06.2026).
33. 3D Dice Roller for Three.js (dice-box-threejs) [Електронний ресурс]. - Режим доступу: <https://github.com/3d-dice/dice-box-threejs> (дата звернення: 05.06.2026).
34. Pixi React - Write PixiJS applications using React declarative style [Електронний ресурс]. - Режим доступу: <https://pixijs.io/pixi-react/> (дата звернення: 06.06.2026).

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		97

ДОДАТКИ

		Бордюг Д. В.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Талавер О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		98

Лістинг вихідного коду

Фрагмент схеми реляційної бази даних (schema.prisma):

```

``prisma
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
enum UserRole {
  USER
  ADMIN
}
model User {
  id          Int          @id
  @default(autoincrement())
  email       String       @unique
  username    String       @unique
  password    String
  role        UserRole     @default(USER)
  createdAt   DateTime     @default(now())
  updatedAt   DateTime     @updatedAt
  // —— VTT Relations ——
  vttRoomsOpened VttRoom[]
  @relation("VttRoomOpenedBy")
  vttCharacterSheets VttCharacterSheet[] @relation("VttCharacterOwner")
  vttGmCreatures    VttGmCreature[]
  @relation("VttGmCreatureOwner")
  vttDiceRolls      VttDiceRoll[]      @relation("VttDiceRollInitiator")
  vttLayerItems      VttLayerItem[]     @relation("VttLayerItemAuthor")
}
model Campaign {
  id          Int          @id @default(autoincrement())

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    title                String
    status                CampaignStatus    @default(ACTIVE)
    ownerId              Int
    owner                User                @relation("OwnerCampaigns", fields:
[ownerId], references: [id])
    sessions              Session[]
    // — VTT Relations —
    vttScenes             VttScene[]        @relation("CampaignVttScenes")
    vttCharacterSheets    VttCharacterSheet[] @relation("VttCharacterCampaign")
  }
  model Session {
    id                    Int                @id @default(autoincrement())
    title                 String
    date                  DateTime
    duration              Int                @default(180)
    status                SessionStatus     @default(PLANNED)
    campaignId            Int?
    campaign              Campaign?         @relation(fields: [campaignId], references:
[id], onDelete: SetNull)
    ownerId              Int
    owner                User                @relation("OwnedSessions", fields:
[ownerId], references: [id])
    // — VTT Relations —
    vttRoom              VttRoom?
    vttCharacterSheets    VttCharacterSheet[] @relation("VttCharacterSession")
    vttGmCreatures        VttGmCreature[]    @relation("VttGmCreatureSession")
  }
  ...

```

Менеджер глобального стану віртуального столу (useBattlefieldStore.js):

```

````javascript
import { create } from 'zustand';
import { persist } from 'zustand/middleware';

export function findNewSceneId(prevScenes, nextScenes) {
 const prevKeys = Object.keys(prevScenes || {});

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const nextKeys = Object.keys(nextScenes || {});
if (prevKeys.length === 0) return null;
if (nextKeys.length <= prevKeys.length) return null;

return nextKeys.find((id) => !prevKeys.includes(id)) ?? null;
}

const useBattlefieldStore = create(
 persist(
 (set) => ({
 gridSize: 64,
 activeSceneId: null,
 gmViewSceneId: null,
 scenes: {},
 tokens: [],
 backgroundImage: null,
 mapWidth: 2048,
 mapHeight: 2048,
 setVttState: (state) => set((prev) => {
 let gmViewSceneId = prev.gmViewSceneId;
 const activeSceneId = state.activeSceneId ?? null;
 const hasScenes = state.scenes && Object.keys(state.scenes).length > 0;
 if (gmViewSceneId && hasScenes && !state.scenes[gmViewSceneId]) {
 gmViewSceneId = activeSceneId;
 }
 if (!gmViewSceneId && activeSceneId) {
 gmViewSceneId = activeSceneId;
 }
 if (state.scenes && prev.scenes) {
 const newId = findNewSceneId(prev.scenes, state.scenes);
 if (newId) {
 gmViewSceneId = newId;
 }
 }
 return {
 activeSceneId,
 scenes: state.scenes || prev.scenes,

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				101
Змн.	Арк.	№ док.ум.	Підпис	Дата		

```

 gmViewSceneId,
 backgroundImage: state.backgroundUrl ?? prev.backgroundUrl,
 mapWidth: state.mapWidth ?? prev.mapWidth,
 mapHeight: state.mapHeight ?? prev.mapHeight,
 };
}),
previewSceneItem: (sceneId, itemId, updates) => set((state) => {
 if (!state.scenes?.[sceneId]) return state;
 const scene = state.scenes[sceneId];
 if (!scene.layers) return state;
 let layerIndex = -1;
 let itemIndex = -1;
 for (let i = 0; i < scene.layers.length; i++) {
 const items = scene.layers[i].items;
 if (items) {
 itemIndex = items.findIndex(item => item.id === itemId);
 if (itemIndex !== -1) {
 layerIndex = i;
 break;
 }
 }
 }
 if (layerIndex === -1) return state;
 const layer = scene.layers[layerIndex];
 const item = layer.items[itemIndex];
 const newLayers = [...scene.layers];
 newLayers[layerIndex] = {
 ...layer,
 items: [
 ...layer.items.slice(0, itemIndex),
 { ...item, ...updates },
 ...layer.items.slice(itemIndex + 1)
]
 };
 return {
 scenes: {

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				102
Змн.	Арк.	№ док.ум.	Підпис	Дата		

```

 ...state.scenes,
 [sceneId]: {
 ...scene,
 layers: newLayers
 }
 }
};

)),
setGmViewSceneId: (sceneId) => set({ gmViewSceneId: sceneId }),
setGridSize: (size) => set({ gridSize: size }),
drawingTool: null,
setDrawingTool: (tool) => set({ drawingTool: tool }),
drawingColor: 'ffffff',
setDrawingColor: (color) => set({ drawingColor: color }),
drawingThickness: 4,
setDrawingThickness: (thickness) => set({ drawingThickness: thickness }),
drawPreviews: {},
setDrawPreview: (userId, previewData) => set((state) => ({
 drawPreviews: {
 ...state.drawPreviews,
 [userId]: previewData
 }
}))),
clearAllPreviews: () => set({ drawPreviews: {} }),
rulerTool: null,
rulerConfig: { coneAngle: 60, distance: 15, radius: 20, width: 20, height: 20 },
localRuler: null,
remoteRulers: {},
setRulerTool: (tool) => set({ rulerTool: tool }),
setLocalRuler: (ruler) => set({ localRuler: ruler }),
setRemoteRuler: (userId, ruler) => set((state) => {
 if (!ruler) {
 const next = { ...state.remoteRulers };
 delete next[userId];
 return { remoteRulers: next };
 }
}

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

```

 return { remoteRulers: { ...state.remoteRulers, [userId]: ruler } } };
 }},
 }},
 {
 name: 'vtt-battlefield-storage',
 partialize: (state) => ({ gmViewSceneId: state.gmViewSceneId }),
 }
)
);
export default useBattlefieldStore;
...

```

Серверний обробник WebSocket-подій віртуального столу (ws-vtt.handler.js):

```

````javascript
const { AppError, ERROR_CODES } = require('../constants/errors');
const { vttStateManager } = require('../vtt/vtt-state.manager');
const sessionService = require('../services/session.service');
const { rollDice } = require('../vtt/dice-engine');
const { sendEvent } = require('../ws-utils');
const vttSceneService = require('../services/vtt-scene.service');
const { logger } = require('../lib/logger');
const autoSaveIntervals = new Map();
const AUTO_SAVE_INTERVAL_MS = 5 * 60 * 1000; // 5 хв
function parseSessionId(value) {
  const parsed = Number.parseInt(value, 10);
  if (!Number.isInteger(parsed) || parsed <= 0) {
    throw new AppError(ERROR_CODES.VALIDATION_FAILED, 'sessionId повинен
бути позитивним числом');
  }
  return parsed;
}
function getVttRoom(sessionId) {
  return `vtt:${sessionId}`;
}
async function handleVttOpen(socket, payload, roomManager) {
  const sessionId = parseSessionId(payload.sessionId);

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				104
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const userId = socket.user?.id;
if (!userId) {
    throw new AppError(ERROR_CODES.SECURITY_ACCESS_DENIED, 'Необхідна
авторизація');
}
const sessionPage = await sessionService.getSessionPageById(sessionId, userId);
if (!sessionPage.actions.canOpenVtt) {
    throw new AppError(ERROR_CODES.SECURITY_ACCESS_DENIED, 'Тільки GM
може відкрити Ігровий стіл');
}
const currentState = vttStateManager.getVttState(sessionId);
const hasScenes = Object.keys(currentState.scenes || {}).length > 0;
if (!hasScenes) {
    try {
        const campaignId = sessionPage.campaignId || null;
        const dbState = await vttSceneService.loadVttState(sessionId, campaignId);
        if (Object.keys(dbState.scenes).length > 0) {
            const room = vttStateManager._ensureRoom(sessionId);
            room.scenes = dbState.scenes;
            room.activeSceneId = dbState.activeSceneId;
            room.initiative = dbState.initiative;
            logger.info({ sessionId, sceneCount: Object.keys(dbState.scenes).length }, '[VTT]
Hydrated from DB');
        }
    } catch (err) {
        logger.error({ err, sessionId }, '[VTT] Failed to load state from DB, starting fresh');
    }
}
vttStateManager.openVtt(sessionId, userId);
startAutoSave(sessionId, sessionPage.campaignId || null);
sendEvent(socket, 'vtt:opened', { sessionId, isOpen: true });
roomManager.broadcastExcept(getVttRoom(sessionId), {
    type: 'vtt:opened',
    sessionId,
    isOpen: true,
}, socket);

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				105
Змн.	Арк.	№ докум.	Підпис	Дата		

```

}

async function handleVttTokenMove(socket, payload, roomManager, eventType) {
  const sessionId = parseSessionId(payload.sessionId);
  const { sceneId, tokenId, x, y } = payload;
  if (!tokenId) return;
  if (sceneId && eventType === 'vtt:token_drop') {
    vttStateManager.updateToken(sessionId, sceneId, tokenId, { x, y });
  }
  roomManager.broadcastExcept(getVttRoom(sessionId), {
    type: eventType,
    sessionId,
    sceneId,
    tokenId,
    x,
    y,
  }, socket);}

async function handleVttDiceRoll(socket, payload, roomManager) {
  const sessionId = parseSessionId(payload.sessionId);
  const { formula, name, strength, visibility, characterName } = payload;
  const player = characterName || socket.user?.username || 'Гравець';
  const initiatorId = socket.user?.id ? String(socket.user.id) : null;
  const rollResult = {
    ...rollDice(formula, player),
    name,
    strength: strength || 1,
    visibility: visibility || 'PUBLIC',
    initiatorId,
  };
  const entry = vttStateManager.addDiceRoll(sessionId, rollResult);
  // Асинхронне збереження в БД (не блокує WS)
  vttSceneService.saveDiceRoll(
    sessionId,
    entry,
    socket.user?.id ? Number(socket.user.id) : null
  ).catch((err) => logger.error({ err, sessionId }, '[VTT] dice roll DB save failed'));

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				106
Змн.	Арк.	№ докum.	Підпис	Дата		

```

    if (entry.visibility === 'GM_ONLY') {
    const vttState = vttStateManager.getVttState(sessionId);
    const gmId = vttState.openedBy ? String(vttState.openedBy) : null;
    roomManager.broadcastFilter(getVttRoom(sessionId), {
        type: 'vtt:dice:result',
        sessionId,
        roll: entry
    }, (clientSocket) => {
        const clientId = clientSocket.user?.id ? String(clientSocket.user.id) : null;
        return clientId === gmId || clientId === initiatorId;
    });
    } else {
    roomManager.broadcast(getVttRoom(sessionId), {
        type: 'vtt:dice:result',
        sessionId,
        roll: entry
    });
    }
}

function startAutoSave(sessionId, campaignId) {
    const key = String(sessionId);
    if (autoSaveIntervals.has(key)) return;
    const interval = setInterval(async () => {
        try {
            const state = vttStateManager.getVttState(sessionId);
            if (state.isOpen) {
                await vttSceneService.saveVttState(sessionId, campaignId, state);
            }
        } catch (err) {
            logger.error({ err, sessionId }, '[VTT] Auto-save failed');
        }
    }, AUTO_SAVE_INTERVAL_MS);
    autoSaveIntervals.set(key, interval);
}

```

		Бордюг Д. В.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Талавер О. В.				107
Змн.	Арк.	№ док.ум.	Підпис	Дата		