

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ІНТЕГРАЦІЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ
ТА ГЕНЕРАТИВНОЇ ГРАФІКИ ДЛЯ ВЕДЕННЯ
РОЛЬОВИХ ІГОР «DUNGEONS & DRAGONS»
INTEGRATION OF LARGE LANGUAGE MODELS AND
GENERATIVE GRAPHICS FOR GAME MASTERING IN
DUNGEONS & DRAGONS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-114м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ДРОБІНЯК М.В.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ДРОБІНЯКА Миколи Васильовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. . Тема проєкту (роботи) Інтеграція великих мовних моделей та генеративної графіки для ведення рольових ігор «Dungeons & Dragons».

Integration of Large Language Models and Generative Graphics for Game Mastering in Dungeons & Dragons

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Програмна реалізація системи. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “30” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Микола ДРОБІНЯК
(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Степан СКРУПСЬКИЙ
(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
110 с., 7 табл., 35 рис., 3 дод., 33 джерела.

КОМП'ЮТЕРНА ПРОГРАМА, ВЕЛИКІ МОВНІ МОДЕЛІ,
ГЕНЕРАТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, МСР-СЕРВЕР, DUNGEONS &
DRAGONS, АВТОМАТИЗАЦІЯ ПІДГОТОВКИ ІГРОВИХ СЕСІЙ,
КЕРУВАННЯ СЮЖЕТОМ, ГЕНЕРАЦІЯ ЗОБРАЖЕНЬ.

Об'єкт дослідження – процес інженерії програмного забезпечення на основі штучного інтелекту для підтримки та автоматизації підготовки й проведення настільних рольових ігор.

Предмет дослідження – методи та програмні засоби інтеграції генеративних моделей для автоматизації функцій ведучого гри.

Мета роботи – підвищення ефективності діяльності ведучого настільних рольових ігор шляхом зменшення часових витрат на підготовку та проведення ігрових сесій за рахунок розробки підходу гібридної оркестрації великих мовних моделей та генеративної графіки.

Наукова новизна – запропоновано підхід динамічної оркестрації ігрового контексту, який, на відміну від існуючих підходів, використовує дворівневу архітектуру пам'яті (оперативну на базі токенів та довготривалу на базі векторних представлень) та протокол уніфікації інструментів (МСР), що дозволяє забезпечити узгодженість сюжету та валідацію ігрової механіки при зменшенні кількості логічних помилок («галюцинацій») моделі.

Практична цінність – розроблено веборієнтоване програмне забезпечення, що реалізує запропонований підхід оркестрації та дозволяє автоматизувати рутинні операції майстра (генерацію візуального контенту, перевірку правил, створення NPC), що скорочує час підготовки до ігрової сесії та знижує когнітивне навантаження під час гри.

Матеріали, методи та технічні засоби: фреймворки NestJS та Next.js, платформа Node.js та мова програмування TypeScript, фреймворк оркестрації великих мовних моделей LangChain, протоколи HTTP/REST та MCP, великі мовні моделі, моделі генерації зображень класу Stable Diffusion (та подібні), система керування базами даних PostgreSQL, сховище даних у пам'яті Valkey для обробки черг повідомлень, інструменти контейнеризації Docker, персональний комп'ютер під керуванням операційної системи macOS Tahoe.

Результати. Розроблено веборієнтовану програмну систему для проведення ігор Dungeons & Dragons, яка інтегрує генеративний штучний інтелект для комплексного керування ігровим світом. Система забезпечує генерацію візуального контенту (ігрові мапи, локації, зовнішність персонажів, предмети, монстри), автоматизоване створення персонажів із розподілом характеристик згідно з правилами 5-ї редакції, а також динамічне моделювання сюжету та діалогів з контролем логічної адекватності подій. Реалізовано функціонал для введення NPC у сюжет та супроводу покрокових битв із автоматичним розрахунком механік за правилами D&D. Практична цінність роботи полягає у створенні самодостатнього середовища, яке бере на себе рутинні та обчислювальні аспекти гри (механіка битв, генерація статів, перевірка правил), дозволяючи майстру та гравцям зосередитися на рольовій взаємодії та сюжеті.

Висновки. Виконано проєктування архітектури системи на основі мікросервісів. Наукова новизна полягає у розробці підходу оркестрації великих мовних моделей, що включає використання протоколу Model Context Protocol (MCP) для динамічного керування ігровим контекстом, а також у розробці підходу організації довготривалої семантичної пам'яті на базі розширення pgvector, який застосовує стратегію мультивекторного пошуку для агрегації гетерогенних контекстних шарів (персональних арок персонажів та глобального сюжету). Крім того, запропоновано архітектурний підхід до побудови гнучкої мультимодельної системи, де MCP-сервер забезпечує абстракцію від конкретних провайдерів генерації зображень (дозволяючи динамічну заміну моделей та платформ), а інтеграція через LangChain

уможливорює гібридне використання як хмарних (наприклад, ChatGPT), так і локальних (наприклад, Ollama) великих мовних моделей залежно від обчислювальних ресурсів користувача. Розроблено програмне забезпечення для взаємодії з LLM та генерації зображень. Здійснено тестування розробленої платформи та перевірку ефективності алгоритмів збереження контексту.

Галузь використання – автоматизація підготовки та проведення настільних рольових ігор типу Dungeons & Dragons в онлайн- та офлайн-форматах, платформи для організації та супроводу кампаній з настільних рольових ігор, а також освітні системи й гейміфіковані середовища, де застосовуються сюжетно-орієнтовані ігрові сценарії.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 110 pages, 7 tables, 35 figures, 3 appendixes, 33 sources.

COMPUTER PROGRAM, LARGE LANGUAGE MODELS, GENERATIVE ARTIFICIAL INTELLIGENCE, MCP SERVER, DUNGEONS & DRAGONS, GAME SESSION PREPARATION AUTOMATION, PLOT MANAGEMENT, IMAGE GENERATION.

The object of study is the process of software engineering based on artificial intelligence to support and automate the preparation and conduct of board role-playing games.

The subject of study is methods and software tools for integrating generative models to automate the functions of the game master.

The purpose of the work is increasing the efficiency of the activities of the host of board role-playing games by reducing the time spent on preparing and conducting game sessions by developing a method for hybrid orchestration of large language models and generative graphics.

Scientific novelty: an approach to dynamic orchestration of the game context is proposed, which, unlike existing approaches, uses a two-level memory architecture (operational based on tokens and long-term based on vector representations) and the tool unification protocol (MCP), which allows ensuring plot consistency and validation of game mechanics while reducing the number of logical errors ("hallucinations") of the model.

The practical value of the work lies in the development of web-oriented software that implements the proposed orchestration method and allows you to automate Game Master's routine operations (visual content generation, rule checking, NPC creation), which reduces the preparation time for a game session and reduces cognitive load during the game.

Materials, methods and tools: NestJS and Next.js frameworks, Node.js platform and TypeScript programming language, LangChain large-scale language

model orchestration framework, HTTP/REST and MCP protocols, large-scale language models, Stable Diffusion class image generation models (and similar), PostgreSQL database management system, Valkey in-memory data storage for processing message queues, Docker containerization tools, personal computer running macOS Tahoe operating system.

Results. A web-based software system for running Dungeons & Dragons games has been developed, which integrates generative artificial intelligence for comprehensive management of the game world. The system provides visual content generation (game maps, locations, character appearance, items, monsters), automated character creation with distribution of characteristics according to the rules of the 5th edition, as well as dynamic plot and dialogue modeling with control of the logical adequacy of events. Functionality has been implemented for introducing NPCs into the plot and supporting turn-based battles with automatic calculation of mechanics according to the rules of D&D. The practical value of the work lies in creating a self-sufficient environment that takes on the routine and computational aspects of the game (battle mechanics, stat generation, rule checking), allowing the master and players to focus on role-playing interaction and the plot.

Conclusions The system architecture has been designed based on microservices. The scientific novelty consists in improving the method of orchestrating large language models by using the Model Context Protocol (MCP) for dynamic game context management, as well as in developing a method for organizing long-term semantic memory based on the pgvector extension, which uses a multivector search strategy to aggregate heterogeneous context layers (personal character arcs and global plot). In addition, an architectural approach to building a flexible multi-model system is proposed, where the MCP server provides abstraction from specific image generation providers (allowing dynamic replacement of models and platforms), and integration via LangChain enables hybrid use of both cloud (e.g., ChatGPT) and local (e.g., Ollama) large language models depending on the user's computing resources. Software has been developed for interaction with LLM and image generation. The developed platform has been tested and the effectiveness of context preservation algorithms has been verified.

The field of use is automation of the preparation and conduct of tabletop role-playing games such as Dungeons & Dragons in online and offline formats, platforms for organizing and supporting tabletop role-playing game campaigns, as well as educational systems and gamified environments where story-oriented game scenarios are used.

ЗМІСТ

С.

Перелік скорочень та умовних познач	12
Вступ	13
1 Аналіз предметної області	15
1.1 Особливості організації та проведення настільних рольових ігор	15
1.2 Аналіз існуючих програмних засобів для підтримки ігрового процесу	18
1.3 Специфіка використання AI в рольових іграх	22
1.4 Висновки до розділу 1	24
2 Матеріали і методи	28
2.1 Формалізація процесу підготовки та проведення ігрової сесії	28
2.2 Розробка підходу гібридної оркестрації ігрового контексту	30
2.3 Критерії оцінювання ефективності запропонованого підходу	38
2.4 Обґрунтування вибору інструментальних засобів реалізації	39
3 Розробка архітектури програми	50
3.1 Загальна структура програмної системи	50
3.2 Проектування бази даних	51
3.3 Проектування взаємодії компонентів (Sequence Diagram)	53
3.4 Проектування класів та модулів (Class Diagram)	54
3.5 Проектування станів обробки запитів до AI	57
3.6 Проектування кооперації об'єктів	58
3.7 Проектування інтерфейсу користувача	59
3.8 Висновки до розділу 3	61
4 Програмна реалізація системи	62
4.1 Реалізація серверної логіки та взаємодії з LLM	62
4.2 Програмна реалізація модуля пам'яті та RAG	64
4.3 Реалізація інструментального рівня та модуля правил	65
4.4 Підсистема генерації візуального контенту	66
4.5 Реалізація клієнтського інтерфейсу	68
4.6 Висновки до розділу 4	69

5	Експлуатація, тестування та експериментальне дослідження програми	72
5.1	Призначення та умови виконання системи	72
5.2	Підготовка середовища та розгортання	73
5.3	Інструкція користувача	73
5.4	Тестування програмного забезпечення	74
5.5	Експериментальне дослідження ефективності	74
	Висновки	84
	Перелік джерел посилань	86
	Додаток А Технічне завдання	90
	Додаток Б Фрагмент тексту програми	95
	Додаток В Слайди презентації	101

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AI	– Artificial Intelligence;
API	– Application Programming Interface;
DTO	– Data Transfer Object;
D&D	– Dungeons & Dragons;
FIFO	– First-In, First-Out;
HNSW	– Hierarchical Navigable Small World;
HP	– Health Points;
IDE	– Integrated Development Environment;
IoC	– Inversion of Control;
LLM	– Large Language Modal;
LoRA	– Low-Rank Adaptation;
MCP	– Model Context Protocol;
NPC	– Non-Player Character;
ORM	– Object-Relational Mapping;
RAG	– Retrieval-Augmented Generation;
REST	– Representational State Transfer;
SQL	– Structured Query Language;
SRD	– System Reference Document;
UI	– User Interface;
UX	– User Experience;
VTT	– Virtual Tabletop;
БД	– база даних;
ПЗ	– програмне забезпечення;
СКБД	– система креування базами даних;
⊕	– операція послідовного з’єднання елементів (конкатенація).

ВСТУП

Популярність настільних рольових ігор, зокрема Dungeons & Dragons, стрімко зростає у сучасному світі, перетворюючись із нішевого хобі на глобальний культурний феномен. Цьому значно сприяє цифровізація дозвілля та розвиток онлайн-спільнот, що дозволяє гравцям з різних куточків світу об'єднуватися у віртуальному просторі. Через те, що попит на якісний ігровий контент та досвідченість майстрів постійно зростає, розробка нових інструментів для підтримки ігрового процесу стає актуальним завданням.

На сьогоднішній день існує багато програмних рішень для проведення настільних рольових ігор, так званих віртуальних ігрових столів [1], які дозволяють візуалізувати мапи та автоматизувати кидки кубиків. Проте роль ведучого гри залишається надзвичайно складною та ресурсоемною. Ведучий змушений одночасно виконувати функції сценариста, режисера, актора та арбітра складних правил, що створює значне когнітивне навантаження. При цьому, підготовка якісної ігрової сесії «з нуля» вимагає чимало часу на створення персонажів, локацій та сюжетних ліній, а існуючі інструменти часто вимагають ручного введення даних та не надають інтелектуальної підтримки. Тому подальші дослідження щодо впровадження нових підходів, методів та засобів штучного інтелекту для підвищення рівня автоматизації підготовки та проведення ігор є необхідними.

Одним із перспективних напрямків є використання великих мовних моделей та генеративної графіки. Проте існуючі спроби інтеграції цих технологій часто є фрагментарними: окремі чат-боти для тексту, окремі генератори для зображень, без єдиного контексту та контролю за дотриманням правил гри. Це призводить до порушення логіки сюжету та «галюцинацій» моделей. Тому тема роботи є актуальною та спрямована на розробку комплексної платформи, яка об'єднує генеративні можливості AI з чіткою логікою ігрових механік [2].

Метою дослідження є підвищення ефективності діяльності ведучого настільних рольових ігор шляхом зменшення часових витрат на підготовку та проведення ігрових сесій за рахунок розробки підходу гібридної оркестрації великих мовних моделей та генеративної графіки.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести аналіз предметної області та існуючих інструментів для підтримки настільних рольових ігор;
- дослідити підходи оркестрації великих мовних моделей та способи розширення їх контекстного вікна;
- розробити архітектуру системи, що інтегрує LLM, генеративні графічні моделі та ігрову логіку D&D;
- реалізувати механізм довготривалої пам'яті на основі векторної бази даних для збереження сюжетної цілісності;
- створити програмне забезпечення платформи та виконати його тестування.

Наукова новизна полягає в розробці підходу динамічної оркестрації ігрового контексту, який, на відміну від існуючих підходів, використовує дворівневу архітектуру пам'яті (оперативну на базі токенів та довготривалу на базі векторних представлень) та протокол уніфікації інструментів (MCP), що дозволяє забезпечити узгодженість сюжету та сувору валідацію ігрової механіки, суттєво знижуючи кількість логічних помилок («галюцинацій») моделі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості організації та проведення настільних рольових ігор

Настільні рольові ігри представляють собою специфічний клас інтерактивних розваг, що базуються на вербальній взаємодії учасників, яка регулюється формалізованим набором правил. На відміну від класичних настільних або комп'ютерних ігор, де простір варіантів обмежений заздалегідь запрограмованими сценаріями, цей жанр пропонує умовно нескінченну свободу дій у межах вигаданого світу.

Найпоширенішою системою у світі є Dungeons & Dragons, 5-та редакція якої стала стандартом де-факто в індустрії [3]. З точки зору системного аналізу, ігрову сесію D&D можна розглядати як кооперативний процес створення історії, що складається з двох асиметричних ролей – Ведучого (Dungeon Master, DM) та Гравців.

1.1.1 Роль Ведучого як керуючої ланки системи

Ведучий гри виступає центральним вузлом обробки інформації. Його функції виходять за межі простого адміністрування і включають кілька рівнів відповідальності:

- симуляція світу. Ведучий відповідає за реакцію навколишнього середовища на дії гравців. Це включає не лише фізику світу та логіку подій, але й відігривання ролей усіх неігрових персонажів (Non-Player Characters, NPC), монстрів та антагоністів;

- арбітраж правил. Правила D&D 5-ї редакції є досить об'ємними, але не можуть покрити всі можливі ситуації. Ведучий повинен інтерпретувати правила, визначати складність дій та приймати рішення в спірних моментах;

– керування темпом. Ведучий динамічно змінює фокус гри, перемикаючись між детальним тактичним боєм, дослідженням локацій та соціальною взаємодією, щоб утримувати увагу гравців.

У контексті автоматизації це означає, що проєктована система повинна мати доступ до «прихованого стану» ігрового світу, який невідомий гравцям, і дозовано видавати інформацію залежно від результатів перевірок.

1.1.2 Алгоритмічна структура ігрового циклу

Фундаментальною основою ігрового процесу D&D є так званий «ігровий цикл», який можна формалізувати наступним алгоритмом:

а) опис ситуації. Ведучий описує оточення, в якому знаходяться персонажі, та поточну ситуацію (наприклад, «ви входите у темну печеру, де пахне сіркою»);

б) декларація дій. Гравці заявляють, що вони хочуть зробити (наприклад, «я хочу тихо підкрастися до входу» або «я шукаю пастки»);

в) визначення механіки:

1) ведучий визначає, чи можлива ця дія;

2) якщо результат дії не є гарантованим, ведучий призначає відповідну характеристику та навичку для перевірки;

3) встановлюється поріг складності;

г) випадкова генерація та модифікація. Гравець кидає 20-гранний кубик, додає модифікатори свого персонажа;

г) наслідки. Ведучий порівнює результат із порогом складності і описує наслідки (успіх або провал), що змінює стан ігрового світу, після чого цикл повертається до пункту 1.

Програмна реалізація цього циклу вимагає тісної інтеграції великої мовної моделі (для кроків 1, 2, 5) з математичним модулем (для кроків 3, 4), оскільки LLM схильні до помилок у арифметичних обчисленнях та утриманні числових порогів.

1.1.2 Три стовпи ігрового процесу

Згідно з правилами D&D 5 редакції, ігровий процес базується на трьох основних компонентах, кожен з яких висуває специфічні вимоги до автоматизації:

а) соціальна взаємодія:

1) суть. Діалоги з NPC, перемовини, залякування, обман;

2) вимоги до AI. Висока контекстна залежність. AI повинен пам'ятати попереднє ставлення NPC до героїв, їхню мотивацію та цілі. Це ідеальна сфера застосування LLM, проте вона потребує збереження довготривалого семантичного контексту;

б) дослідження:

1) суть. Переміщення по карті світу або підземелля, пошук ресурсів, розгадування загадок;

2) вимоги до AI. Потреба у візуалізації. Опис «величного замку» значно краще сприймається через зображення. Тут необхідна інтеграція з генеративними графічними моделями (Stable Diffusion) [4], які можуть створювати локації на основі текстового опису. Також необхідна логіка «туману війни» – приховування інформації, яку персонажі ще не відкрили;

в) битва:

1) суть. Тактична покрокова взаємодія, де час чітко структурований на раунди (6 секунд ігрового часу);

2) вимоги до AI. Суворе дотримання правил. Переміщення обмежене швидкістю, атаки залежать від Класу Броні (АС). LLM часто ігнорують ці обмеження на користь «красивого опису», тому для цього стовпа необхідна жорстка програмна логіка, яка лише декорується текстом від LLM.

1.1.4 Когнітивне навантаження та проблема підготовки

Однією з головних проблем, яку покликана вирішити дана дипломна робота, є диспропорція між часом підготовки та часом гри. За емпіричними даними спільноти, на підготовку 4-годинної ігрової сесії Ведучий витрачає від 2 до 6 годин. Цей час витрачається на:

- створення карт (бойових та регіональних);
- написання числових характеристик для унікальних монстрів;
- продумування можливих варіантів розвитку сюжету.

Під час самої гри ведучий зазнає пікового когнітивного навантаження, змушений перемикатися між довідниками правил, нотатками сюжету та керуванням монстрами. Помилки, викликані втомою або неуважністю Ведучого, можуть призвести до порушення логічної цілісності світу або відчуття несправедливості у гравців.

Впровадження інтелектуального асистента, здатного «на льоту» генерувати відсутній контент (наприклад, «опиши вміст кишень цього розбійника» або «згенеруй зображення таверни»), дозволить перенести акцент роботи Ведучого з рутинної генерації та адміністрування на творчу режисуру.

1.2 Аналіз існуючих програмних засобів для підтримки ігрового процесу

Ринок програмного забезпечення для настільних рольових ігор перебуває на етапі активної трансформації. Для формулювання вимог до проєктованої системи доцільно провести класифікацію та аналіз існуючих рішень. На сьогоднішній день інструментарій можна розділити на три основні категорії: класичні віртуальні ігрові столи, платформи інтерактивного сторітелінгу на базі штучного інтелекту та універсальні генеративні інструменти.

1.2.1 Класичні віртуальні ігрові столи

До цієї категорії належать платформи, головною метою яких є цифрова імітація фізичного столу. Найвідомішими представниками віртуальних ігрових столів є Roll20, Foundry VTT та Fantasy Grounds.

Ці системи надають потужний інструментарій для відображення ігрових мап, керування віртуальними фігурками персонажів та монстрів, а також автоматизації математичних розрахунків згідно з правилами D&D (кидки кубиків, розрахунок модифікаторів атаки, віднімання очок здоров'я).

Основні недоліки даного класу систем у контексті автоматизації творчого процесу:

- відсутність генеративних можливостей. Весь контент (описи кімнат, зовнішність персонажів, сюжетні ходи) має бути створений ведучим вручну заздалегідь або імпровізований у реальному часі без програмної підтримки;

- статичність світу. Якщо гравці вирішують відхилитися від підготовленого сценарію, система не може автоматично згенерувати нову локацію чи персонажа. Ведучий змушений зупиняти гру для пошуку необхідних матеріалів у зовнішніх джерелах;

- високий поріг входження. Існуючі платформи не замінюють знання правил, а лише інструменталізують їх. Щоб коректно налаштувати ігрову сесію, ведучий змушений витратити значний час на вивчення великого обсягу теоретичних матеріалів (Player's Handbook, Dungeon Master's Guide, Monster Manual). Система не підказує, як працює та чи інша механіка, а лише вимагає введення точних параметрів, що робить вхід у хобі для новачків надзвичайно складним без менторської підтримки.

Таким чином, ці системи є інструментами відображення та адміністрування, але не створення контенту.

1.2.2 AI-орієнтовані платформи сторітелінгу

До цієї групи належать проєкти на кшталт AI Dungeon та NovelAI. Вони побудовані на базі великих мовних моделей, які донавчені на текстах художньої літератури та текстових квестах. Основна механіка полягає у генерації продовження тексту на основі дій користувача.

Попри високу гнучкість сюжету, ці системи мають критичні недоліки для проведення структурованих ігор типу Dungeons & Dragons:

- ігнорування ігрової механіки. Моделі схильні до так званого «логічного дрейфу». Вони не враховують числові обмеження: персонаж може використати заклинання, яке він не знає, або пережити поранення, яке мало б бути смертельним згідно з правилами;
- проблема пам'яті. Через обмеження контекстного вікна система швидко забуває деталі, що трапилися на початку довгої історії, плутає імена персонажів та причинно-наслідкові зв'язки;
- відсутність тактичної складової. Ці платформи орієнтовані на «театр розуму» (уявну гру) і не підтримують візуалізацію тактичних мап, що є обов'язковим елементом бойової системи D&D.

1.2.3 Універсальні генеративні інструменти

На практиці багато ведучих використовують комбінацію універсальних інструментів: ChatGPT, Gemini або Claude для генерації тексту та ідей, і Midjourney або Stable Diffusion для створення зображень.

Такий підхід дозволяє отримати якісний контент, але створює проблему фрагментації робочого процесу:

- розрив контексту. Ведучий змушений вручну переносити інформацію між різними вікнами браузера;

- десинхронізація модальностей [5]. Згенероване зображення персонажа може суперечити його текстовому опису, оскільки графічна нейромережа не «знає» контексту, який відомий текстовій моделі;
- операційне навантаження. Необхідність постійно формувати запити до штучного інтелекту під час гри відволікає від керування процесом та спілкування з гравцями.

1.2.4 Порівняльний аналіз

Для наочної демонстрації необхідності розробки нової системи проведено порівняльний аналіз функціональних можливостей розглянутих рішень (таблиця 1.1).

Таблиця 1.1 – Порівняльний аналіз функціональних можливостей рішень

Критерій порівняння	Класичні віртуальні столи (Roll20/Foundry)	Текстові AI-квести (AI Dungeon)	Універсальні чат-боти (ChatGPT)	Проектована система
1	2	3	4	5
Автоматична генерація сюжету	Ні (тільки ручне введення)	Так	Так	Так
Дотримання правил D&D 5-ї редакції	Частково (калькулятор)	Ні (ігнорування)	Низька (помилки в логіці)	Висока (програмний контроль)
Генерація зображень	Ні	Частково	Так (через DALL-E)	Так (Stable Diffusion)

Продовження таблиці 1.1

1	2	3	4	5
Довготривала пам'ять про світ	Ні	Низька	Середня	Висока (векторна БД)
Єдиний інтерфейс керування	Так	Так	Ні	Так

Як видно з аналізу, на ринку існує незаповнена ніша для системи, яка б поєднувала суворість правил та візуалізацію класичних столів із креативністю генеративних моделей. Проектована система спрямована на вирішення цієї задачі шляхом інтеграції великих мовних моделей, генеративної графіки та жорсткої програмної логіки в єдиному середовищі.

1.3 Специфіка використання AI в рольових іграх

Попри значний потенціал генеративних моделей, їх безпосередня інтеграція в ігровий процес Dungeons & Dragons наштовхується на ряд фундаментальних технічних обмежень. Аналіз архітектури сучасних LLM (Large Language Models) дозволяє виділити чотири ключові проблеми, вирішення яких є необхідною умовою для створення повноцінного віртуального ведучого.

1.3.1 Обмеженість контекстного вікна та втрата довгострокової пам'яті

Однією з найкритичніших проблем є обмежений обсяг оперативної пам'яті моделей, який вимірюється в токенах (context window). Стандартна ігрова кампанія D&D може тривати місяцями або роками, генеруючи обсяги

тексту, що значно перевищують ліміти навіть найсучасніших моделей [6]. (наприклад, GPT-5 або Claude 3) [7].

Коли обсяг діалогу перевищує ліміт вікна, модель змушена «забувати» попередню інформацію. Це призводить до критичних сюжетних помилок:

- втрата імен та характеристик NPC, зустрінутих на початку гри;
- зміна фактів про світ (наприклад, місто, яке було зруйноване в першій сесії, раптом стає цілим);
- втрата причинно-наслідкових зв'язків рішень гравців.

Для вирішення цієї проблеми недостатньо простого збільшення контекстного вікна, оскільки це експоненційно збільшує вартість запитів та час обробки. Необхідне впровадження зовнішньої довготривалої пам'яті на основі векторних баз даних та механізмів семантичного пошуку (RAG) [8], що дозволить динамічно підвантажувати лише релевантну інформацію.

1.3.2 Феномен «галюцинацій» та ігнорування детермінованих правил

Великі мовні моделі є імовірнісними системами, що прогнозують наступний токен на основі статистичних закономірностей, а не жорсткої логіки. У контексті D&D, де правила є суворими та детермінованими, це призводить до так званих «галюцинацій» [9]:

- арифметичні помилки. Модель може некоректно підрахувати суму урону або залишок здоров'я монстра;
- вигадкування механік [10]. Генерація заклинань чи здібностей, яких не існує в офіційних правилах 5-ї редакції;
- логічний дрейф. Зміна параметрів порогу складності в залежності від внутрішнього стану моделі, а не об'єктивної ситуації.

Це обумовлює необхідність гібридної архітектури, де LLM відповідає лише за генерацію наративу (художнього опису), а контроль за станом гри та валідацію дій виконує окреме програмне ядро.

1.3.3 Проблема мультимодальної узгодженості

При використанні генеративної графіки для ілюстрації гри виникає проблема збереження візуальної ідентичності (Consistency). Генеративні моделі (наприклад, Stable Diffusion) є стохастичними: той самий текстовий запит при повторній генерації дасть зовсім інше зображення.

У рольовій грі критично важливо, щоб зовнішність головних героїв, ключових NPC та важливих локацій залишалася впізнаваною протягом усієї кампанії. Без спеціальних методів фіксації стилю та параметрів генерації, ілюстрації будуть виглядати розрізненими, що руйнує атмосферу гри.

1.3.4 Фрагментація інструментарію та складність інтеграції

Сучасний ландшафт AI-інструментів є вкрай фрагментованим. Розробники змушені писати окремий код для інтеграції кожної нової моделі або сервісу. Це створює так звану проблему «M×N», де M – кількість ігрових систем, а N – кількість провайдерів AI.

Відсутність єдиного стандарту взаємодії між мовною моделлю та зовнішніми інструментами (генераторами зображень, базами даних, пошуковими системами) ускладнює масштабування платформи та заміну компонентів (наприклад, перехід з OpenAI на локальну модель Llama). Це актуалізує використання уніфікованих протоколів, таких як Model Context Protocol (MCP) [11], для стандартизації підключення інструментів.

1.4 Висновки до розділу 1

Проведений у першому розділі системний аналіз предметної області та існуючих інструментальних засобів дозволив сформувати цілісне уявлення про поточний стан автоматизації процесів підготовки та проведення настільних рольових ігор, зокрема системи Dungeons & Dragons 5-ї редакції.

У результаті дослідження специфіки діяльності ведучого встановлено, що керування ігровою сесією є складним когнітивним процесом, який вимагає одночасного оперування гетерогенними потоками даних: структурованою інформацією (ігрові правила, числові характеристики, механіка битв) та неструктурованою інформацією (діалоги, художні описи, імпровізація сюжету). Виявлено, що основною проблемою, яка стримує підвищення якості ігрового процесу та збільшує поріг входження для нових ведучих, є диспропорційно великі часові витрати на етап підготовки та пікове когнітивне навантаження під час гри, пов'язане з необхідністю миттєвої генерації контенту у відповідь на непередбачувані дії гравців.

Аналіз ринку існуючого програмного забезпечення дозволив класифікувати доступні рішення на три основні групи, кожна з яких має суттєві архітектурні недоліки в контексті поставленої задачі:

- класичні віртуальні ігрові столи забезпечують надійну автоматизацію механічних розрахунків та візуалізацію тактичних мап, проте повністю позбавлені генеративних можливостей. Вони виступають інструментом адміністрування, але не створення контенту, перекладаючи весь тягар творчої роботи на користувача;

- платформи сторітелінгу на базі AI демонструють високу креативність, але є непридатними для проведення структурованих ігор через ігнорування числових правил, відсутність тактичної складової та схильність до порушення логіки ігрового світу;

- універсальні генеративні інструменти (ChatGPT, Midjourney) забезпечують високу якість окремих елементів контенту, але їх використання призводить до фрагментації робочого процесу. Ведучий змушений вручну переносити контекст між різними вікнами та додатками, що руйнує цілісність ігрового досвіду та призводить до помилок.

Особливу увагу в розділі було приділено аналізу технічних обмежень сучасних великих мовних моделей (LLM) при їх інтеграції в ігрові системи.

Ідентифіковано три критичні бар'єри, які унеможливлюють використання LLM «з коробки» для довготривалих кампаній:

- обмеженість контекстного вікна. Стандартні архітектури трансформерів [12] не здатні утримувати в оперативній пам'яті повний лог подій кампанії, що може тривати місяцями. Це призводить до втрати семантичних зв'язків, забування імен NPC та порушення причинно-наслідкових зв'язків сюжету. Простого збільшення ліміту токенів недостатньо через економічну неефективність та зниження точності моделі («lost in the middle phenomenon») [13];

- стохастична природа генерації («галюцинації»). Ймовірнісні моделі схильні ігнорувати детерміновані правила гри (наприклад, ліміти здоров'я, слоти заклинань, модифікатори кидків) на користь більш «ймовірного» з лінгвістичної точки зору тексту. Це є неприпустимим для D&D, де правила є основою ігрового балансу;

- складність мультимодальної синхронізації. Генерація зображень локацій та персонажів окремими нейромережами часто суперечить текстовому опису через відсутність спільного контексту між мовною та графічною моделями.

На основі проведеного аналізу зроблено висновок про необхідність розробки принципово нового підходу до побудови програмної системи підтримки НРІ. Така система не повинна покладатися виключно на генеративні можливості однієї моделі, а має реалізовувати гібридну архітектуру.

Обґрунтовано, що для вирішення виявлених проблем доцільно застосувати метод динамічної оркестрації ігрового контексту, який складає основу наукової новизни даного дослідження. На відміну від існуючих підходів, цей метод передбачає використання:

- дворівневої архітектури пам'яті. Поєднання оперативної пам'яті на базі токенів (для поточного діалогу) та довготривалої пам'яті на базі векторних представлень (для збереження історії світу та персонажів). Це

дозволить системі оперувати контекстом усієї кампанії, динамічно підвантажуючи лише релевантні спогади через механізми семантичного пошуку (RAG);

- протоколу уніфікації інструментів (Model Context Protocol - MCP) для стандартизації взаємодії LLM із зовнішніми модулями. Це дозволить моделі не «вгадувати» результати кидків кубиків, а делегувати ці задачі детермінованим програмним алгоритмам, виступаючи в ролі оркестратора, а не калькулятора;

- агентного підходу до генерації [14]: система самостійно визначатиме необхідність виклику генеративних графічних моделей (Stable Diffusion) на основі аналізу наративу, забезпечуючи узгодженість візуального та текстового ряду.

Практична цінність запропонованого підходу полягатиме у створенні веборієнтованого програмного забезпечення, що реалізує цей метод оркестрації. Це дозволить автоматизувати рутинні операції майстра (генерацію візуального контенту, перевірку правил, створення NPC), що, в свою чергу, скоротить час підготовки до ігрової сесії та суттєво знизить когнітивне навантаження під час гри.

Таким чином, у першому розділі сформовано теоретичний базис для подальшого дослідження: визначено об'єкт та предмет дослідження, ідентифіковано ключові технічні виклики та окреслено напрямки їх подолання через розробку методу гібридної оркестрації контексту. Це дозволяє перейти до другого розділу роботи, присвяченого безпосередній розробці теоретичних моделей та алгоритмів функціонування пропонованої системи.

2 МАТЕРІАЛИ І МЕТОДИ

У даному розділі здійснюється теоретичне обґрунтування вирішення задачі підвищення ефективності підготовки та проведення рольових ігор. Базуючись на результатах аналізу предметної області, проведеного у попередньому розділі, виконується математична формалізація процесів генерації ігрового контенту та оцінки когнітивного навантаження ведучого.

Ключовим етапом дослідження є розробка методу гібридної оркестрації, що забезпечує узгоджену взаємодію імовірнісних моделей штучного інтелекту з детермінованими правилами ігрової системи. Також у розділі формулюється система критеріїв для об'єктивного оцінювання ефективності запропонованого підходу та, на основі визначених теоретичних вимог, здійснюється обґрунтування вибору інструментальних засобів програмної реалізації.

2.1 Формалізація процесу підготовки та проведення ігрової сесії

Процес ведення настільної рольової гри (НРІ) можна представити як динамічну систему управління станом віртуального світу, де Ведучий (Game Master, GM) виступає в ролі керуючого пристрою, а гравці – в ролі джерел збурення. Для оцінки ефективності автоматизації введемо формальну модель часових витрат Ведучого. Загальний час, що витрачається на одну ігрову кампанію (T_{total}), розраховується за формулою (2.1):

$$T_{total} = \sum_{i=1}^N (T_{prep}^{(i)} + T_{run}^{(i)}), \quad (2.1)$$

де N – кількість ігрових сесій;

$T_{prep}^{(i)}$ – час підготовки до i -ї сесії;

$T_{run}^{(i)}$ – час безпосереднього проведення i -ї сесії.

Час підготовки $T_{prep}^{(i)}$ є сумою витрат на генерацію різних типів контенту і визначається за формулою (2.2):

$$T_{prep}^{(i)} = t_{narrative} + t_{visual} + t_{mechanics}, \quad (2.2)$$

де $t_{narrative}$ – час на написання сюжету, діалогів та описів локацій;

t_{visual} – час на пошук або створення зображень персонажів, карт місцевості та предметів;

$t_{mechanics}$ – час на створення числових характеристик для монстрів та NPC згідно з правилами системи.

У традиційному підході (без використання AI) складові t_{visual} та $t_{narrative}$ залежать виключно від когнітивних здібностей та швидкості роботи людини, що призводить до лінійної залежності якості контенту від витраченого часу.

Під час проведення гри (T_{run}) критичним параметром є час реакції системи на непередбачувану дію гравця. Якщо гравець виконує дію, не передбачену сценарієм, Ведучий змушений імпровізувати. Якість імпровізації (Q_{improv}) залежить від когнітивного навантаження (L_{cog}) згідно із залежністю (2.3):

$$Q_{improv} = f\left(\frac{1}{L_{cog}}\right). \quad (2.3)$$

Обґрунтування даної залежності базується на положеннях теорії когнітивного навантаження (Cognitive Load Theory), сформульованої Дж. Свеллером [15]. Згідно з цією теорією, людська робоча пам'ять має обмежену пропускну здатність. Відомий закон Міллера [16] стверджує, що людина здатна ефективно утримувати в фокусі уваги лише 7 ± 2 об'єкти одночасно.

У контексті настільної рольової гри когнітивний ресурс Ведучого розподіляється між двома типами задач:

- технічні задачі (Extraneous Load) такі як пошук правил, математичні розрахунки, відстеження очок здоров'я;
- творчі задачі (Germane Load) такі як побудова сюжету, акторська гра, опис атмосфери.

Оскільки ці типи задач конкурують за спільний обмежений ресурс уваги [17], зростання технічного навантаження (L_{cog}) призводить до вичерпання ресурсів робочої пам'яті, що унеможлиблює якісне виконання творчих задач. Таким чином, якість імпровізації (Q_{improv}) знаходиться в оберненій залежності від когнітивного навантаження, що вимагає мінімізації технічних операцій ($T_{mechanics}$) засобами автоматизації.

Основною задачею даного дослідження є мінімізація T_{prep} та L_{cog} при максимізації якості контенту за рахунок делегування рутинних операцій генеративним моделям.

2.2 Розробка підходу гібридної оркестрації ігрового контексту

В основу розроблюваної системи покладено авторський підхід гібридної оркестрації, який вирішує фундаментальну проблему неузгодженості між стохастичною (ймовірнісною) природою генеративних моделей та детермінованою (чіткою) природою правил настільних рольових ігор.

Суть підходу полягає у декомпозиції єдиного процесу генерації відповіді на послідовність спеціалізованих етапів, де велика мовна модель (LLM) виконує функцію не генератора кінцевого тексту, а інтелектуального маршрутизатора, що керує набором зовнішніх інструментів через уніфікований протокол.

2.2.1 Концептуальна модель системи

Формально процес обробки ігрового запиту можна представити як функцію відображення F , що трансформує вхідний вектор стану S_{in} у вихідний вектор стану S_{out} , як показано у формулі (2.4)

$$S_{out} = F(S_{in}, H, R), \quad (2.4)$$

де S_{in} – поточний запит гравця (текст, дія);

H – історія ігрової сесії (контекст);

R – набір правил ігрової системи (D&D 5 редакції).

У запропонованому підході функція F реалізується через мультиагентну архітектуру, структурну схему якої наведено на рис. 2.1.

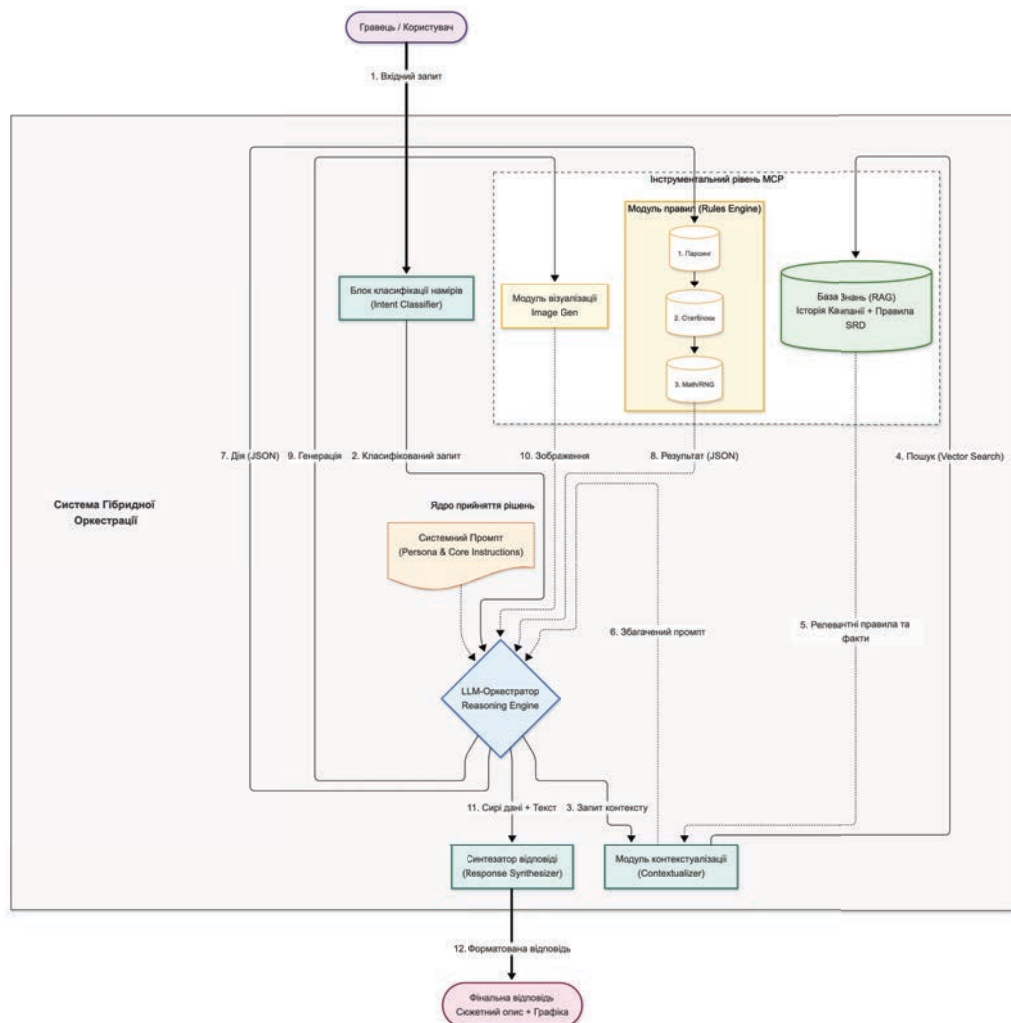


Рисунок 2.1 – Структурна схема підходу гібридної оркестрації

Архітектура системи гібридної оркестрації реалізує конвеєрну обробку запитів, що забезпечує чітке розділення відповідальності між компонентами. Ключові функціональні блоки системи:

а) блок класифікації намірів. Це вхідний шлюз системи, що виконує попередню обробку тексту гравця. Він використовує легковагову модель для визначення типу дії (наратив, бойова механіка, запит на візуалізацію) та виділення ключових сутностей (імена, назви предметів), формуючи структурований об'єкт запиту для подальшої обробки;

б) центральне ядро (LLM-Оркестратор). Це головний керуючий модуль, побудований на базі великої мовної моделі. Він не виконує обчислень самостійно, а діє як агент-планувальник (Reasoning Engine) [18], приймаючи рішення про виклик конкретних інструментів на основі класифікованого наміру та системного промпту;

в) модуль контекстуалізації. Він відповідає за реалізацію патерну RAG (Retrieval-Augmented Generation). Отримуючи запит від оркестратора, цей модуль виконує семантичний пошук у векторній базі даних, витягуючи релевантні фрагменти історії кампанії та правила з SRD (System Reference Document), і формує збагачений контекст, необхідний для генерації коректної відповіді;

г) інструментальний рівень MCP (Model Context Protocol). Це набір спеціалізованих сервісів, доступних оркестратору через уніфікований інтерфейс:

1) модуль пам'яті. Він забезпечує зберігання та пошук довготривалих спогадів і статичних знань про світ;

2) модуль правил. Це детермінований обчислювальний модуль, що реалізує ігрову механіку D&D 5 редакції (парсинг дій, валідацію числових характеристик, генерацію випадкових чисел та математичні розрахунки) без використання нейромереж, що гарантує відсутність «галюцинацій»;

3) модуль візуалізації. Він генерує зображення локацій та персонажів на основі промптів, сформованих оркестратором, забезпечуючи візуальний супровід ігрового процесу;

г) синтезатор відповіді. Це фінальний блок, який агрегує результати роботи всіх інструментів (текстовий опис, результати кидків кубиків, посилання на згенеровані зображення) та формує цілісну, форматовану відповідь для клієнтського інтерфейсу.

Така архітектура дозволяє поєднати креативність генеративного AI з точністю програмних алгоритмів, забезпечуючи високу якість ігрового досвіду.

2.2.2 Математична модель дворівневої пам'яті (RAG)

Однією з головних проблем використання великих мовних моделей у довготривалих ігрових кампаніях є обмеженість контекстного вікна L_{max} . Оскільки обсяг текстового логу ігрової сесії T_{log} з часом лінійно зростає $T_{log} \rightarrow \infty$, у певний момент часу t виконується умова $Length(T_{log}) > L_{max}$, що призводить до втрати інформації.

Для вирішення цієї проблеми розроблено модель пам'яті M , яка описується формулою (2.5):

$$M = M_{short} \cup M_{long}, \quad (2.5)$$

де M_{short} – оперативна пам'ять (Short-term Memory);

M_{long} – довготривала семантична пам'ять (Long-term Memory).

Модель пам'яті структурно розділена на два рівні:

– оперативна пам'ять (M_{short}). Цей рівень реалізовано як чергу типу FIFO (First-In, First-Out), що зберігає повний лог останніх n повідомлень діалогу. Вона забезпечує локальну зв'язність розмови, дозволяючи моделі

розуміти займенники («він», «туди») та реагувати на безпосередні репліки гравців. Формально M_{short} визначається як впорядкована множина за формулою (2.6):

$$M_{short} = \{m_{t-n}, m_{t-n+1}, \dots, m_t\}, \quad (2.6)$$

де m_t – повідомлення в момент часу t ;

– довготривала пам'ять (M_{long}). Цей рівень базується на архітектурі RAG (Retrieval-Augmented Generation). Вся інформація про світ, персонажів та минулі події зберігається у вигляді векторних представлень (embeddings) [19] у базі даних PostgreSQL (з розширенням pgvector).

Нехай E – функція векторного представлення, яка відображає текстовий фрагмент d у k – вимірний векторний простір R^k . Цю функцію визначено формулою (2.7):

$$E: d \rightarrow v, v \in R^k. \quad (2.7)$$

Множина довготривалої пам'яті складається з пар «текст-вектор» та визначається за формулою (2.8):

$$M_{long} = \{(d_i, v_i) | v_i = E(d_i), i = 1 \dots N\}. \quad (2.8)$$

Для співставлення користувацького запиту із даними сховища використовується алгоритм семантичного пошуку. При надходженні нового запиту від гравця q , система генерує вектор запиту $v_q = E(q)$. Для знаходження релевантного контексту використовується метрика косинусної подібності, яка визначає кут між векторами у багатовимірному просторі, яка розраховується за формулою (2.9):

$$Sim(v_q, v_i) = \cos(\theta) = \frac{v_q * v_i}{\|v_q\| \|v_i\|} = \frac{\sum_{j=1}^k v_{q,j} v_{i,j}}{\sqrt{\sum_{j=1}^k v_{q,j}^2} \sqrt{\sum_{j=1}^k v_{i,j}^2}}. \quad (2.9)$$

Система формує множину релевантних фактів R_{ctx} , відбираючи топ- K записів, для яких значення подібності перевищує порогове значення α (поріг релевантності). Формування цієї множини визначається формулою (2.10):

$$R_{ctx} = \{d_i \in M_{long} | Sim(v_q, v_i) \geq \alpha\}. \quad (2.10)$$

Після цього відбувається формування фінального контексту. Кінцевий промпт P , що подається на вхід LLM-Оркестратору, є конкатенацією (позначено як $\oplus$) системних інструкцій S_{sys} , знайдених фактів R_{ctx} та оперативної пам'яті M_{short} . Процес творення кінцевого промпту формалізується формулою (2.11):

$$P = S_{sys} \oplus Format(R_{ctx}) \oplus M_{short} \oplus q. \quad (2.11)$$

Така дворівнева архітектура дозволяє системі оперувати практично необмеженим обсягом знань про ігровий світ, динамічно «згадуючи» лише ті деталі, які є необхідними для обробки поточного запиту, що забезпечує високу сюжетну когерентність при оптимізації витрат на токени.

2.2.3 Алгоритм валідації ігрових дій та механіка МСР

Критичним недоліком великих мовних моделей є їхня нездатність до виконання точних арифметичних операцій та утримання складного стану системи, що призводить до порушення правил гри (феномен «галюцинацій»). Для вирішення цієї проблеми в розробленій системі застосовано принцип розділення відповідальності: LLM відповідає за наміри, а програмний код – за результати.

Реалізація цього принципу здійснюється через модуль, який інтегровано в систему за допомогою протоколу MCP.

Розглянемо математичну модель перевірки успішності дій. Нехай A – множина можливих ігрових дій (атака, перевірка навички, рятівний кидок). Для кожної дії $a \in A$ результат виконання $R(a)$ є детермінованою функцією від випадкової величини ξ (кидок кубика) та вектора характеристик персонажа S .

Формально результат перевірки визначається системою умов (2.12):

$$R(a) = \begin{cases} CRITICAL_SUCCESS, & \text{якщо } \xi = 20 \\ CRITICAL_FAILURE, & \text{якщо } \xi = 1 \\ SUCCESS, & \text{якщо } \xi + M(S) \geq DC \\ FAILURE, & \text{якщо } \xi + M(S) < DC \end{cases}, \quad (2.12)$$

де ξ – результат генерації псевдовипадкового числа у діапазоні $[1,20]$ (імітація кидка кубика d20);

$M(S)$ – сумарний модифікатор (характеристика + майстерність + ситуативні бонуси);

DC – і поріг складності або клас захисту цілі.

Алгоритмічна реалізація процесу обробки ігрової механіки відбувається за алгоритмом, зображеним на рис. 2.2.

Алгоритм складається з наступних етапів:

– семантичний парсинг. LLM-Оркестратор аналізує текстовий запит гравця та формує JSON-об’єкт виклику інструменту;

– валідація стану (Hydration). MCP-сервер отримує запит, звертається до бази даних для отримання актуальних числових характеристик ініціатора та цілі. На цьому етапі перевіряються умови можливості дії (наприклад, наявність зброї, дистанція до цілі);

- детерміноване обчислення. Виконується генерація числа ξ за допомогою криптографічно стійкого генератора (наприклад, `crypto.randomInt` у Node.js) та розрахунок формули (2.12);
- застосування наслідків. У разі успіху розраховується шкода або ефект, і оновлюється стан бази даних (наприклад, зменшення HP монстра);
- генерація опису. MCP-сервер повертає в LLM структурований результат, на основі якого модель генерує художній опис сцени.

Такий підхід гарантує повну відповідність ігровому процесу офіційним правилам 5-ї редакції, виключаючи суб'єктивність або помилки штучного інтелекту.



Рисунок 2.2 – Алгоритм роботи MCP

2.3 Критерії оцінювання ефективності запропонованого підходу

Для перевірки ефективності розробленого підходу гібридної оркестрації та порівняння його з існуючими підходами (ручна підготовка ведучим, використання класичних VTT, використання універсальних LLM) пропонується використовувати систему кількісних та якісних показників.

2.3.1 Кількісні критерії

Серед кількісних критеріїв оцінювання ефективності було виокремлено такі показники:

– час генерації ігрової сцени (T_{gen}). Цей критерій визначає сумарний час, необхідний для створення повного комплексу контенту для однієї сцени (опис локації, характеристики NPC, зображення). Очікується зменшення цього показника порівняно з ручною підготовкою.

Ефективність за часом (E_{time}) розраховується за формулою (2.13):

$$E_{time} = \frac{T_{manual} - T_{system}}{T_{manual}} * 100\%, \quad (2.13)$$

де T_{manual} – середній час підготовки сцени вручну (написання тексту + пошук референсів + створення числових характеристик персонажів);

T_{system} – час генерації та верифікації контенту в розробленій системі;

– точність дотримання правил (A_{rules}). Визначає здатність системи коректно застосовувати механіку D&D 5-ї редакції (розрахунок класу захисту, бонусів атаки, перевірок навичок). Цей показник є критичним для усунення проблеми «галюцинацій» і розраховується за формулою (2.14):

$$A_{rules} = \frac{N_{correct}}{N_{total}} * 100, \quad (2.14)$$

де $N_{correct}$ – кількість ігрових дій, результат яких повністю відповідає офіційним правилам;

N_{total} – загальна кількість перевірених дій;

– релевантність контексту (R_{ctx}). Оцінює ефективність роботи модуля пам'яті (RAG). Вимірюється як здатність системи коректно «згадати» факт f з історії кампанії при запиті q . Для оцінки використовується метрика $Recall@K$ (повнота при K знайдених документах) [20].

2.3.2 Якісні критерії

Серед якісних критеріїв оцінювання ефективності можна виокремити такі показники:

а) мультимодальна узгодженість – експертна оцінка відповідності згенерованого візуального ряду (зображення) текстовому опису. Оцінюється за шкалою від 1 до 5, де:

- 1) 1 бал надається, коли зображення суперечить тексту (наприклад, «орки в печері» замість «ельфів у лісі»);
- 2) 5 балів надається при повній відповідності деталей, атмосфери та стилю;

б) логічна цілісність сюжету – суб'єктивна оцінка відсутності сюжетних протиріч (наприклад, «воскресіння» вбитих раніше NPC без причини). Забезпечується використанням довготривалої пам'яті.

2.4 Обґрунтування вибору інструментальних засобів реалізації

Для практичної реалізації розробленого підходу гібридної оркестрації та забезпечення відповідності сформульованим критеріям ефективності

необхідно здійснити обґрунтований вибір технологічного стека. Вибір інструментальних засобів базується на вимогах до швидкодії обробки асинхронних запитів, підтримки специфічних протоколів взаємодії з великими мовними моделями (MCP) та ефективності роботи з векторними даними. У цьому підрозділі аналізуються та обираються програмні платформи, фреймворки та системи керування базами даних, що дозволяють побудувати надійну та масштабовану архітектуру системи.

2.4.1 Вибір мови програмування та платформи виконання

В якості основної мови програмування для серверної та клієнтської частин обрано TypeScript [21] у середовищі виконання Node.js.

Вибір TypeScript зумовлений необхідністю суворої типізації даних. Dungeons & Dragons 5-ї редакції оперує складними структурами даних: характеристики персонажів, інвентар, списки заклинань, статблоки монстрів. Використання статичної типізації дозволяє:

- описати ігрові сутності через інтерфейси та типи, що мінімізує помилки на етапі розробки;
- забезпечити надійну валідацію відповідей від LLM (оскільки моделі часто повертають JSON, його структуру необхідно перевіряти перед використанням у ігровій логіці).

Платформа Node.js [22] обрана через її асинхронну природу (Event Loop), що є критично важливим для обробки довготривалих запитів до неймереж без блокування основного потоку виконання, а також для підтримки WebSocket-з'єднань у реальному часі.

2.4.2 Обґрунтування вибору вебфреймворків

Архітектура системи побудована на розділенні клієнтської та серверної частин, що реалізуються за допомогою фреймворків Next.js та NestJS відповідно.

2.4.2.1 Серверна частина

Для розробки API обрано фреймворк NestJS. Його ключовою перевагою є модульна архітектура та активне використання патерну «впровадження залежностей» (Dependency Injection). Це дозволяє:

- створювати ізольовані модулі для різних аспектів гри (модуль бойової системи, модуль генерації персонажів, модуль інтеграції з AI);
- легко підміняти реалізації сервісів. Наприклад, сервіс генерації тексту (LLMService) може мати різні реалізації для OpenAI, Anthropic або локальної моделі Ollama, і заміна провайдера не вимагатиме змін у бізнес-логіці програми.

2.4.2.2 Клієнтська частина

Для створення інтерфейсу користувача використовується Next.js (на базі бібліотеки React). Цей фреймворк забезпечує:

- серверний рендеринг (SSR) для швидкого початкового завантаження сторінок (аркушів персонажів, довідників);
- ефективну роботу зі станом застосунку, що необхідно для відображення динамічних змін на ігровій мапі та в чаті.

2.4.3 Вибір системи керування базами даних

Однією з ключових вимог до системи є необхідність зберігання двох типів даних:

- реляційні дані. Цей тип даних охоплює відомості про користувачів, ігрові кампанії, зв’язки між персонажами та предметами (інвентар) тощо.
- векторні дані. Цей тип даних включає семантичні представлення (embeddings) текстових описів для реалізації довготривалої пам’яті (RAG).

Для вирішення цієї задачі було розглянуто кілька архітектурних підходів: використання окремої векторної БД разом з реляційною або використання універсального рішення [23]. Порівняльний аналіз наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз СКБД для гібридних систем

Критерій	PostgreSQL (+pgvector)	MySQL	MongoDB (Atlas Vector)	Pinecone + Реляційна БД
Тип даних	Реляційні + Векторні	Переважно реляційні	Документні + Векторні	Тільки векторні
ACID транзакції	Повна підтримка	Повна підтримка	Обмежена (на рівні документа)	Відсутні (в Pinecone)
Складність інфраструктури	Низька (один контейнер)	Низька	Середня	Висока (два окремих сервіси)
Підтримка JSON	Висока (JSONB)	Середня	Висока	Відсутня
Вартість (Self-hosted)	Безкоштовно	Безкоштовно	Умовно безкоштовно	Платно (SaaS)

Обрано PostgreSQL з розширенням pgvector. Це рішення дозволяє уникнути проблеми фрагментації даних. Завдяки типу даних JSONB, PostgreSQL ефективно зберігає гнучкі структури (наприклад, динамічні ефекти заклинань), а розширення pgvector дозволяє виконувати векторний пошук (Cosine Similarity) безпосередньо в SQL-запитах, поєднуючи його зі звичайними фільтрами (наприклад, «знайти спогади, схожі на цей запит, АЛЕ тільки для персонажа з ID=5»).

2.4.4 Вибір ORM (Object-Relational Mapping)

Оскільки серверна частина розробляється на NestJS, важливо обрати інструмент для взаємодії з базою даних, який підтримує патерни цього фреймворку та забезпечує безпеку типів. Було проведено порівняння трьох популярних ORM для екосистеми Node.js (таблиця 2.2).

Таблиця 2.2 – Порівняльний аналіз СКБД для гібридних систем

Характеристика	TypeORM	Prisma	Sequelize
Сумісність з NestJS	Нативна (офіційний модуль)	Через адаптер	Через адаптер
Контроль SQL-запитів	Високий (QueryBuilder)	Низький (своя мова запитів)	Середній
Підтримка міграцій	Вбудована CLI	Вбудована	Окрема бібліотека
Патерни проєктування	Data Mapper / Active Record	Тільки Data Mapper	Active Record
Робота з pgvector	Можлива через raw queries	Складна	Складна

Обрано TypeORM. Головним аргументом стала наявність потужного QueryBuilder. Ігрова логіка D&D часто вимагає складних вибірок (наприклад,

«отримати всіх монстрів у радіусі X, у яких є опір до вогню, відсортованих за рівнем загрози»). TypeORM дозволяє будувати такі запити програмно, зберігаючи читабельність коду. Крім того, TypeORM є стандартом де-факто для NestJS, що забезпечує кращу інтеграцію з модулем Dependency Injection.

2.4.5 Вибір фреймворку для роботи з LLM

- Для інтеграції великих мовних моделей необхідно вирішити такі задачі:
- уніфікація API різних провайдерів (OpenAI, Anthropic, Ollama);
 - керування ланцюжками викликів (Chains);
 - управління пам’яттю.

Порівняння інструментів, що можуть застосовуватися для вирішення цих задач наведено у таблиці 2.3.

Таблиця 2.3 – Порівняльний аналіз СКБД для гібридних систем

Критерій	LangChain	LlamaIndex	Vercel AI SDK	Native API (OpenAI SDK)
Основний фокус	Оркестрація та агенти	Індексація та пошук даних	Стрімінг на UI	Базовий доступ
Абстракція провайдерів	Висока (легка заміна моделей)	Середня	Висока	Відсутня (сувора прив’язка до платформи)
Підтримка інструментів	Дуже розвинена	Розвинена	Базова	Обмежена
Екосистема	Найбільша спільнота	Фокус на RAG	Фокус на Frontend	—

Обрано LangChain. Цей фреймворк надає найбільш гнучкий інструментарій для створення «агентів» – автономних модулів, які можуть приймати рішення про використання інструментів. Це критично для реалізації Model Context Protocol (MCP), де модель сама вирішує, чи потрібно їй звернутися до генератора зображень або до бази знань правил D&D. Можливість легкої заміни моделі (наприклад, перемикання з GPT-4 на локальну Llama 3) без переписування коду стала вирішальним фактором.

2.4.6 Методи оркестрації та інтеграції інструментів для великих мовних моделей

Після обрання фреймворку LangChain, ключовим завданням є побудова архітектури, яка дозволяє моделі не просто генерувати текст, а й взаємодіяти з ігровим світом. Для цього застосовано агентний підхід (Agentic Workflow) [24].

В якості стандарту взаємодії обрано Model Context Protocol (MCP). Реалізація цього методу в системі виглядає наступним чином:

а) створення MCP-сервера. Розробка окремого мікросервісу, який описує доступні функції (Інструменти) у форматі JSON-схеми. Наприклад: `generate_image`, `roll_dice`, `search_rules`;

б) абстракція інструментів. LLM не знає, яка саме нейромережа генерує зображення (Stable Diffusion чи DALL-E). Вона лише викликає інструмент `generate_image` з описом сцени. Це дозволяє змінювати генератор зображень без зміни логіки агента;

в) ланцюжи обробки (Chains). розділення логіки на спеціалізовані ланцюжки:

- 1) Narrative Chain. Відповідає за художній опис та діалоги;
- 2) Rulekeeper Chain. Виконує перевірку заявок гравців на відповідність правилам D&D 5-ї редакції.

2.4.7 Організація довготривалої пам'яті та семантичного пошуку

Для подолання обмеження контекстного вікна LLM реалізовано підхід RAG (Retrieval-Augmented Generation). Його технічна реалізація базується на можливостях обраної раніше СКБД PostgreSQL.

Архітектура пам'яті використовує розширення pgvector для роботи з векторними представленнями (embeddings). Процес збереження та пошуку контексту відбувається наступним чином:

- текстові дані (описи подій, діалоги, характеристики локацій) проходять через модель ембедінгу (наприклад, text-embedding-3-small від OpenAI), перетворюючись на числовий вектор фіксованої розмірності;
- отриманий вектор зберігається у спеціальному стовпці типу vector у таблиці memories.
- при надходженні запиту від гравця система виконує пошук за косинусною подібністю (Cosine Similarity), щоб знайти найбільш релевантні записи в історії.

Такий підхід дозволяє реалізувати гібридний пошук: поєднання класичного повнотекстового пошуку (за ключовими словами) та семантичного пошуку (за змістом), що забезпечує високу точність вибірки контексту для генерації відповідей. Завдяки цьому підходу мовна модель має можливість отримувати швидкий доступ до подій, що відбулися десятки сесій тому, якщо вони є релевантними до поточної ситуації.

2.4.8 Інтеграція генеративної графіки

Для візуалізації ігрового світу використовуються моделі класу Stable Diffusion. На відміну від закритих рішень (наприклад, DALL-E), Stable Diffusion дозволяє більш гнучко керувати процесом генерації, зокрема використовувати ControlNet [25] для збереження композиції сцени або LoRA (Low-Rank Adaptation) для підтримки єдиного художнього стилю кампанії.

Взаємодія з графічним ядром відбувається через REST API, який обгорнутий у власний MCP-інструмент. Це дає змогу мовній моделі самостійно приймати рішення про необхідність ілюстрації (на основі аналізу тексту сцени) та формувати технічно коректний промпт для генератора.

2.4.9 Керування асинхронними процесами та чергами

Генерація тексту великими моделями та рендеринг зображень є ресурсоемними операціями, час виконання яких може складати від кількох секунд до хвилини. Для запобігання перевантаженню сервера та блокуванню HTTP-з'єднань впроваджено систему черг [26] на базі бібліотеки BullMQ.

В якості сховища для брокера повідомлень та кешування проміжних результатів використовується Valkey – високопродуктивне сховище даних типу «ключ-значення» в оперативній пам'яті. Архітектура з чергами дозволяє:

- лінійно обробляти запити на генерацію, запобігаючи вичерпанню VRAM (відеопам'яті) сервера;
- забезпечувати надійність: у разі збою генерації завдання автоматично перезапускається.

2.4.10 Інструменти розгортання та контейнеризації

Для забезпечення відтворюваності середовища розробки та розгортання використовується платформа контейнеризації Docker [27]. Система складається з набору мікросервісів (API, клієнт, база даних, черга, MCP-сервер), конфігурація яких описана у файлі `docker-compose.yml`. Це дозволяє розгорнути всю інфраструктуру однією командою, що є критично важливим для локального запуску системи користувачами, які бажають використовувати власні апаратні ресурси для роботи нейронних мереж.

2.4.11 Діаграма прецедентів

З метою формалізації функціональних вимог до платформи та визначення меж системи розроблено діаграму прецедентів (Use Case Diagram). При розробці моделі враховано дві основні ролі користувачів: Ведучий (Game Master) та Гравець (Player), а також їхню взаємодію з інтелектуальним агентом системи.

Основний сценарій взаємодії Ведучого з системою включає:

- авторизацію в системі;
- створення нової ігрової кампанії та налаштування правил D&D 5 редакції;
- генерацію візуального контенту (локацій, NPC) через запити до AI-асистента;
- керування бойовими сценами та перевірку кидків кубиків;
- редагування згенерованого контексту для збереження логіки сюжету.

Сценарій взаємодії Гравця включає:

- приєднання до ігрової сесії;
- створення та редагування аркуша персонажа (Character Sheet);
- взаємодію в чаті (декларація дій, спілкування);
- виконання механічних дій (кидки атаки, навичок).

Модель взаємодії користувачів із платформою подано у вигляді діаграми прецедентів на рис. 2.3.

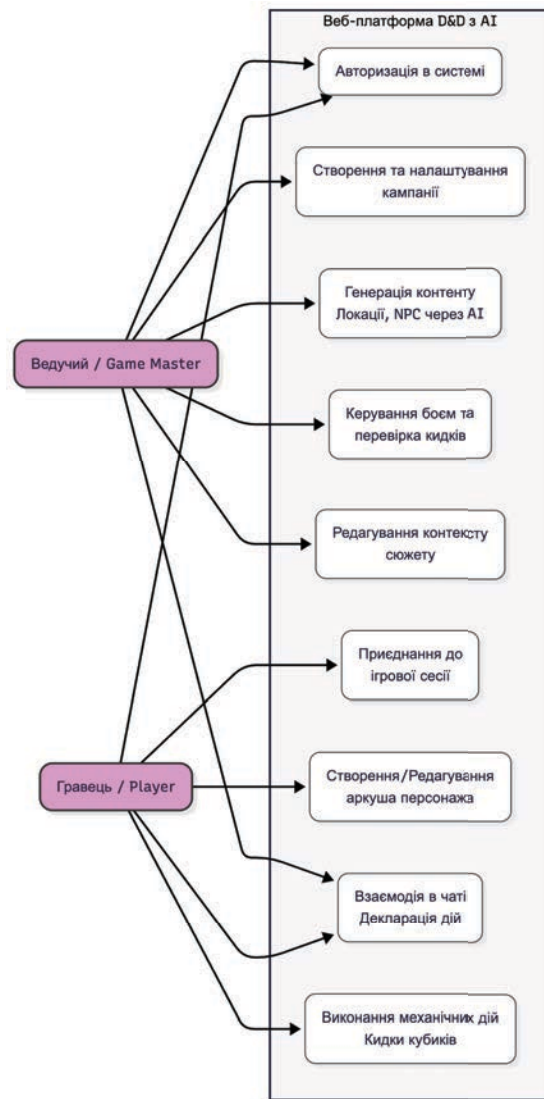


Рисунок 2.3 – Діаграма прецедентів системи

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Загальна структура програмної системи

Проектована платформа базується на клієнт-серверній архітектурі з використанням мікросервісних патернів [28] для ізоляції логіки роботи зі штучним інтелектом. Система складається з трьох основних логічних рівнів:

- рівень представлення (Presentation Layer). Реалізовано на стороні клієнта (Client-side) за допомогою фреймворку Next.js;
- рівень бізнес-логіки (Application Layer). Реалізовано на стороні сервера (Server-side) за допомогою NestJS;
- рівень даних та інфраструктури AI (Data & AI Infrastructure Layer). Включає базу даних PostgreSQL, брокер повідомлень Valkey та сервіси генерації (MCP-сервер).

Рівень представлення відповідає за візуалізацію інтерфейсу користувача та рендеринг ігрової мапи. Взаємодія з сервером відбувається через REST API [29] (для отримання статичних даних) та WebSocket (для оновлення стану гри в реальному часі).

Рівень бізнес-логіки виступає центральним шлюзом (API Gateway). Він включає модулі авторизації (AuthModule), керування кампаніями (CampaignModule) та оркестрації подій (SocketModule).

Структуру програмного забезпечення у вигляді діаграми компонентів наведено на рис. 3.1.

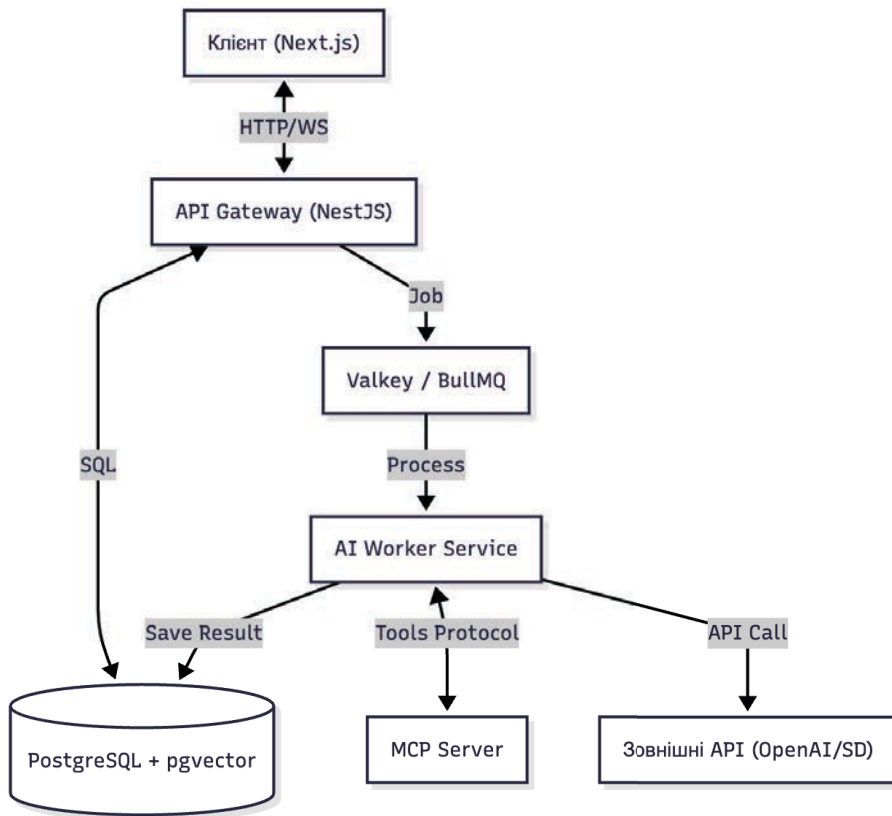


Рисунок 3.1 – Діаграма компонентів системи

3.2 Проектування бази даних

Для забезпечення цілісності даних та підтримки довготривалої пам’яті (RAG) розроблено гібридну схему бази даних PostgreSQL. Схема включає як класичні реляційні таблиці, так і таблиці для зберігання векторних представлень даних.

ER-діаграму (Entity-Relationship) бази даних наведено на рис. 3.2.

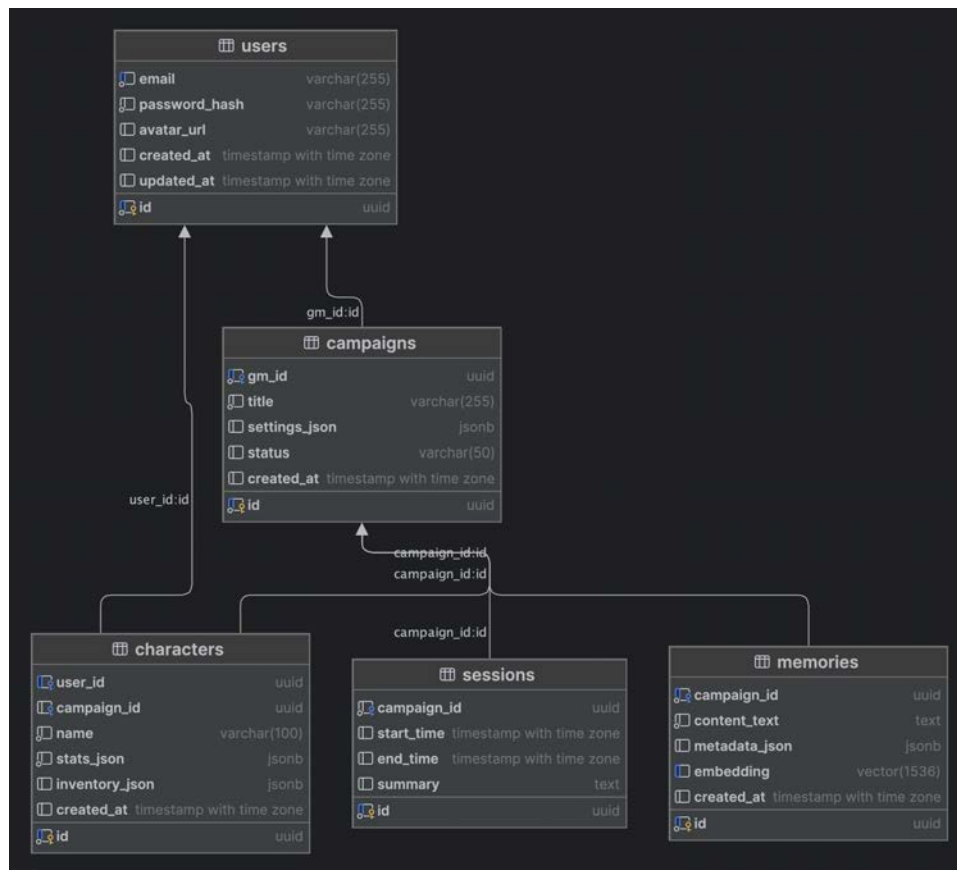


Рисунок 3.2 – ER-діаграма бази даних

Основними сутностями бази даних є:

- **Users**. Зберігає облікові дані користувачів та налаштування профілю;
- **Campaigns**. Містить метадані кампанії, налаштування правил та поточний стан світу;
- **Characters**. Зберігає JSON-структуру з характеристиками персонажа, інвентарем та історією розвитку;
- **GameSessions**. Логує події, що відбуваються під час конкретних ігрових зустрічей;
- **Memories (Vector Store)**. Ключова таблиця для роботи AI. Вона містить текстові фрагменти (Chunks) з історії гри та відповідний стовпець типу vector(1536) (розмірність залежить від моделі ембедінгу). Для прискорення семантичного пошуку використовується індекс HNSW (Hierarchical Navigable Small World) [30].

Такий підхід дозволяє виконувати складні запити, наприклад: «Знайти всі події, пов’язані з персонажем X, де згадується «магічний меч», і відсортувати їх за семантичною релевантністю».

3.3 Проєктування взаємодії компонентів (Sequence Diagram)

Функціонування системи в режимі генерації відповіді Майстра подамо за допомогою діаграми послідовностей (рис. 3.3). Цей процес ілюструє, як заявка гравця трансформується в наративну відповідь та зміну стану світу.

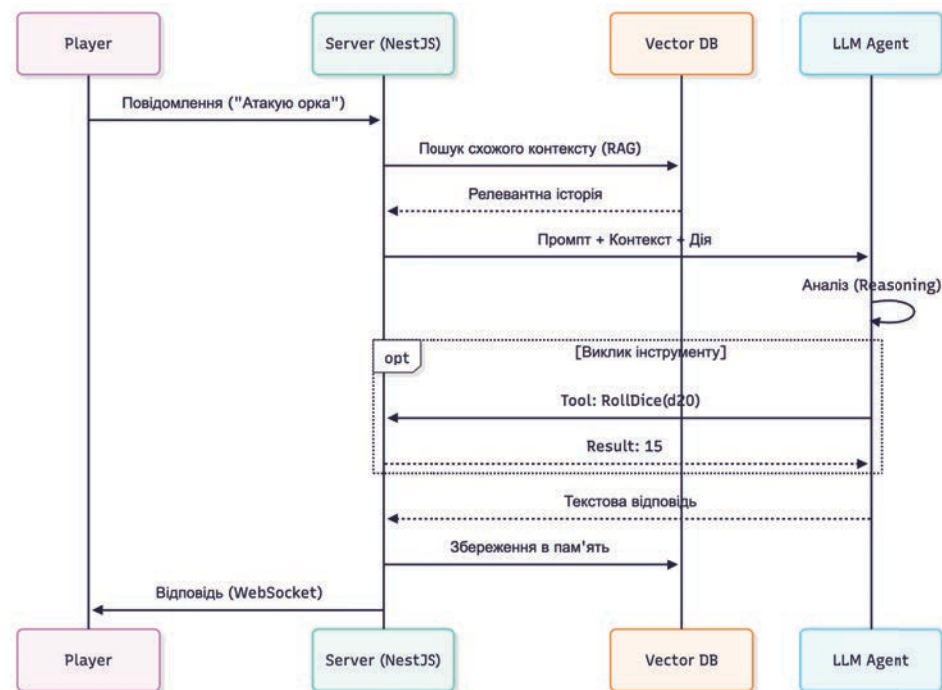


Рисунок 3.3 – Діаграма послідовностей обробки запиту гравця

Алгоритм обробки запиту включає наступні кроки:

а) ініціалізація. Гравець відправляє текстову заявку (наприклад, «Я оглядаю кімнату») через клієнтський інтерфейс;

б) контекстуалізація. Сервер отримує запит, перетворює його на вектор і виконує пошук у таблиці Memories для знаходження релевантного контексту (опис локації, попередні дії);

в) формування промπτу. Знайдений контекст, разом із поточним станом персонажа (НР, навички) та системним промπτом (правила D&D), передається в LLM через LangChain;

г) прийняття рішення: Модель аналізує запит. Якщо для відповіді потрібна перевірка навички або генерація зображення, модель повертає запит на виклик інструменту;

г) запуск інструменту:

1) якщо це перевірка, сервер «кидає кубик» (RNG) і повертає результат моделі;

2) якщо це зображення, запит ставиться в чергу BullMQ у сховищі Valkey. Воркер обробляє запит, викликає Stable Diffusion, зберігає файл і повертає URL;

д) фіналізація. модель генерує фінальну текстову відповідь, описуючи результат дій. Сервер зберігає цю відповідь у БД (і як текст, і як новий вектор) та відправляє клієнту через WebSocket.

3.4 Проєктування класів та модулів (Class Diagram)

Архітектуру серверної частини спроектовано за принципами об'єктно-орієнтованого програмування. Основні класи та їх взаємозв'язки наведено на діаграмі класів (рис. 3.4).

Ключові класи системи та їх зона відповідальності:

а) GameGateway. Цей клас відповідає за обробку WebSocket-з'єднань:

1) атрибути. Містить посилання на екземпляр server типу WebSocketServer для керування кімнатами та розсилки повідомлень;

2) методи:

– `handleConnection` та `handleDisconnect`: Керують життєвим циклом підключення користувачів;

– `handleMessage`: Обробляє текстові повідомлення для чату та AI;

– handleDiceRoll: Обробляє запити на кидок кубиків з візуалізацією результату;

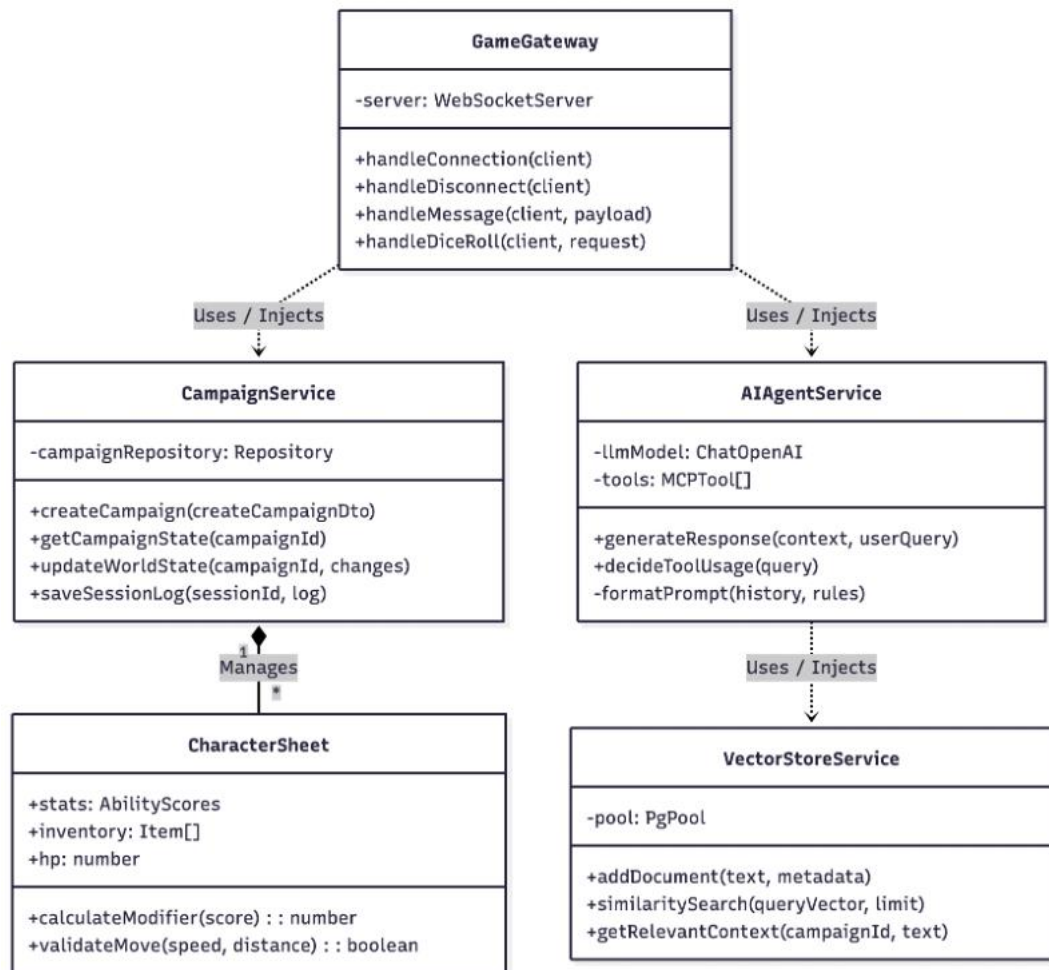


Рисунок 3.4 – Діаграма класів (Серверна частина)

3) залежності:

- CampaignService (для оновлення стану гри);
- AIAgentService (для генерації відповідей);

б) CampaignService. Сервіс бізнес-логіки кампанії:

1) атрибути: Використовує campaignRepository для доступу до бази даних;

2) методи:

- createCampaign: створення кампанії;
- getCampaignState: отримання поточного стану кампанії;

- updateWorldState: оновлення світу;
- saveSessionLog: логування сесій;

3) зв'язки. Реалізує відношення композиції («Manages») з класом CharacterSheet, керуючи множиною персонажів (1 до *);

в) AIAgentService. Це сервіс оркестрації штучного інтелекту:

1) атрибути. Зберігає екземпляр llmModel (ChatOpenAI) та масив інструментів tools (MCPTool[]);

2) методи:

- generateResponse. Публічний метод для генерації відповіді;
- decideToolUsage. Метод для визначення необхідності виклику

інструментів;

– formatPrompt. Приватний метод для підготовки контексту перед відправкою в модель;

3) залежності:

- VectorStoreService (для отримання довготривалої пам'яті);

г) VectorStoreService. Це сервіс для роботи з векторною базою даних:

1) атрибути. Використовує пул з'єднань pool (PgPool);

2) методи:

- addDocument. Додає документи;
- similaritySearch. Виконує пошук за схожістю;
- getRelevantContext. Формує релевантний контекст для конкретної кампанії;

г) CharacterSheet. Клас-модель персонажа:

1) атрибути. Містить характеристики (stats), інвентар (inventory) та очки здоров'я (hp).

2) методи. Реалізує ігрову механіку, зокрема розрахунок модифікаторів (calculateModifier) та валідацію переміщення (validateMove) згідно з правилами D&D.

На діаграмі чітко відокремлено публічні методи (позначені «+») від приватних (позначені «-»), що демонструє дотримання принципу інкапсуляції.

Зв'язки між сервісами реалізовані через механізм Dependency Injection, що забезпечує слабку зв'язаність компонентів.

3.5 Проєктування станів обробки запитів до AI

З огляду на те, що система використовує асинхронну модель обробки ресурсоємних запитів для генерації зображень та складних текстових ланцюжків із використанням черг повідомлень, доцільно змодельовати життєвий цикл такого запиту. Для цього розроблено діаграму станів, яка описує переходи об'єкта «Завдання на генерацію» між різними етапами обробки.

Діаграму станів обробки запитів до AI наведено на рис. 3.5.

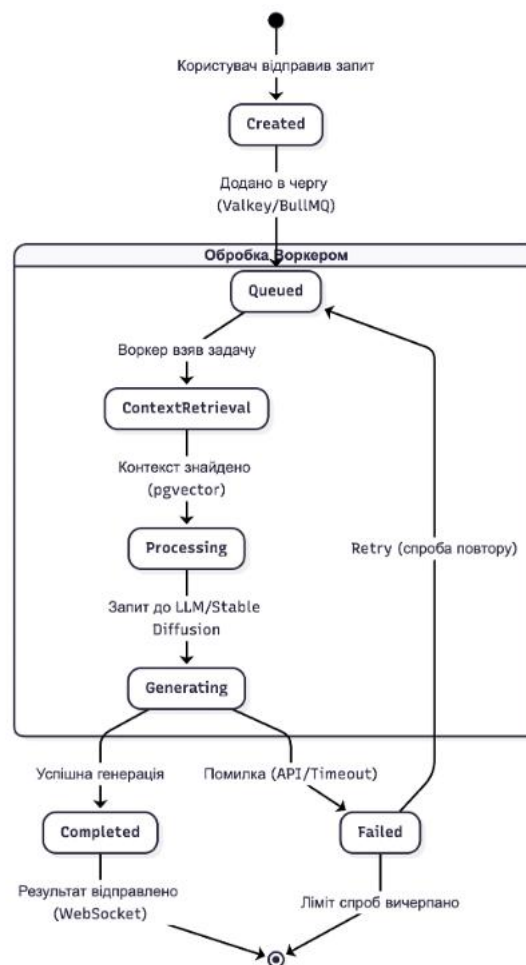


Рисунок 3.5 – Діаграма станів обробки запитів до AI

Життєвий цикл запиту включає наступні стани:

- Created (Створено). Запит отримано від користувача, успішно провалідовано та збережено в базі даних;
- Queued (У черзі). Завдання передано до черги повідомлень (BullMQ) та очікує на звільнення обчислювальних ресурсів воркера;
- Processing (Обробка). Воркер отримав завдання, виконав пошук необхідного контексту у векторній базі даних та ініціював запит до генеративної моделі (LLM або Stable Diffusion);
- Completed (Завершено). Генерація пройшла успішно, результат збережено та відправлено клієнту через WebSocket;
- Failed (Помилка). Виникла помилка під час генерації (таймаут, помилка API тощо), після чого система може ініціювати механізм повторної спроби виконання.

3.6 Проєктування кооперації об'єктів

Діаграма кооперації відображає структурну організацію об'єктів, що беруть участь у реалізації сценарію «Генерація ігрової сцени», та потік повідомлень між ними. На відміну від діаграми послідовностей, яка акцентує увагу на часовій шкалі, ця діаграма підкреслює зв'язки між компонентами архітектури та розподіл відповідальності.

Діаграму кооперації програмної системи наведено на рис. 3.6.

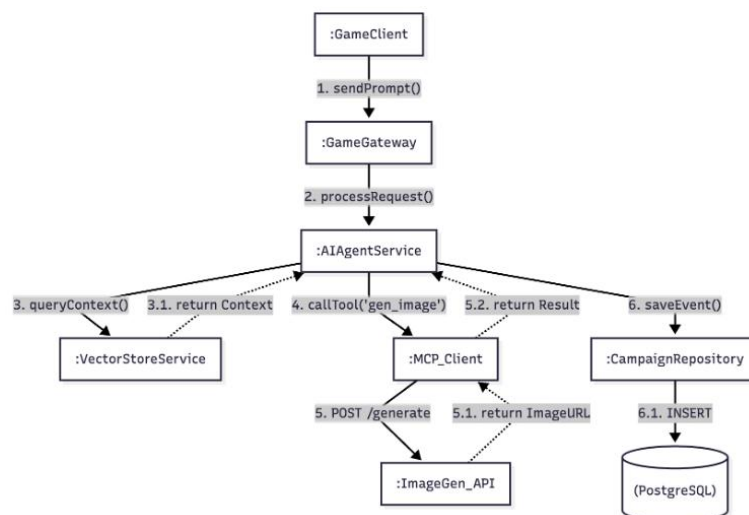


Рисунок 3.6 – Діаграма послідовностей обробки запиту гравця

3.7 Проектування інтерфейсу користувача

Проектування інтерфейсу вебплатформи виконувалося з дотриманням принципів людино-орієнтованого дизайну (Human-Centered Design). Головною метою було зменшення когнітивного навантаження на Ведучого, який змушений одночасно контролювати сюжет, правила та технічний стан гри.

Інтерфейс системи розділено на два окремі представлення (Views) залежно від ролі користувача: Інтерфейс Ведучого (GM View) та Інтерфейс Гравця (Player View).

3.7.1 Інтерфейс ведучого

Це основне робоче середовище, що надає повний контроль над ігровим світом. Макет основного екрану Ведучого наведено на рис. 3.7.

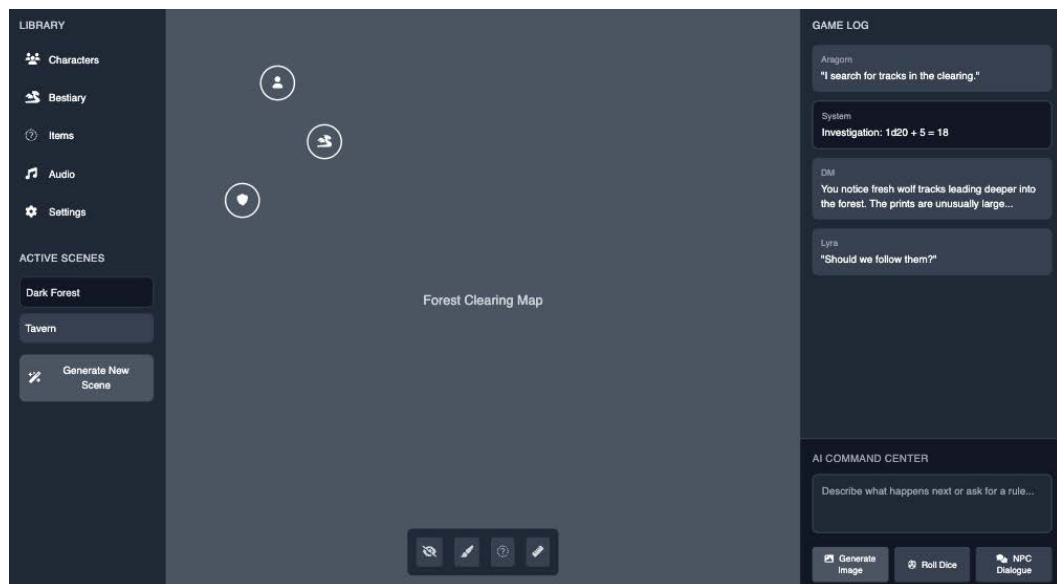


Рисунок 3.7 – Макет інтерфейсу Ведучого

Екран розділено на три функціональні зони:

- візуальна сцена. Центральна область, що займає 60% екрану. Тут відображаються згенеровані фони локацій або тактична сітка. Ведучий має

інструменти для «туману війни» (приховування частин карти) та малювання поверх зображення;

– наративний лог та AI-асистент. Права панель, що містить історію чату. В нижній частині розташоване поле вводу промптів для AI. Тут Ведучий може писати команди типу `/gen image dark forest` або `/roll d20 attack`, які обробляються сервером;

– панель керування кампанією. Містить швидкий доступ до списку персонажів, бібліотеки монстрів та налаштувань генерації.

3.7.2 Інтерфейс гравця

Інтерфейс гравця є спрощеною версією, фокус якої зміщено на інформацію про власного персонажа. Макет наведено на рис. 3.8.

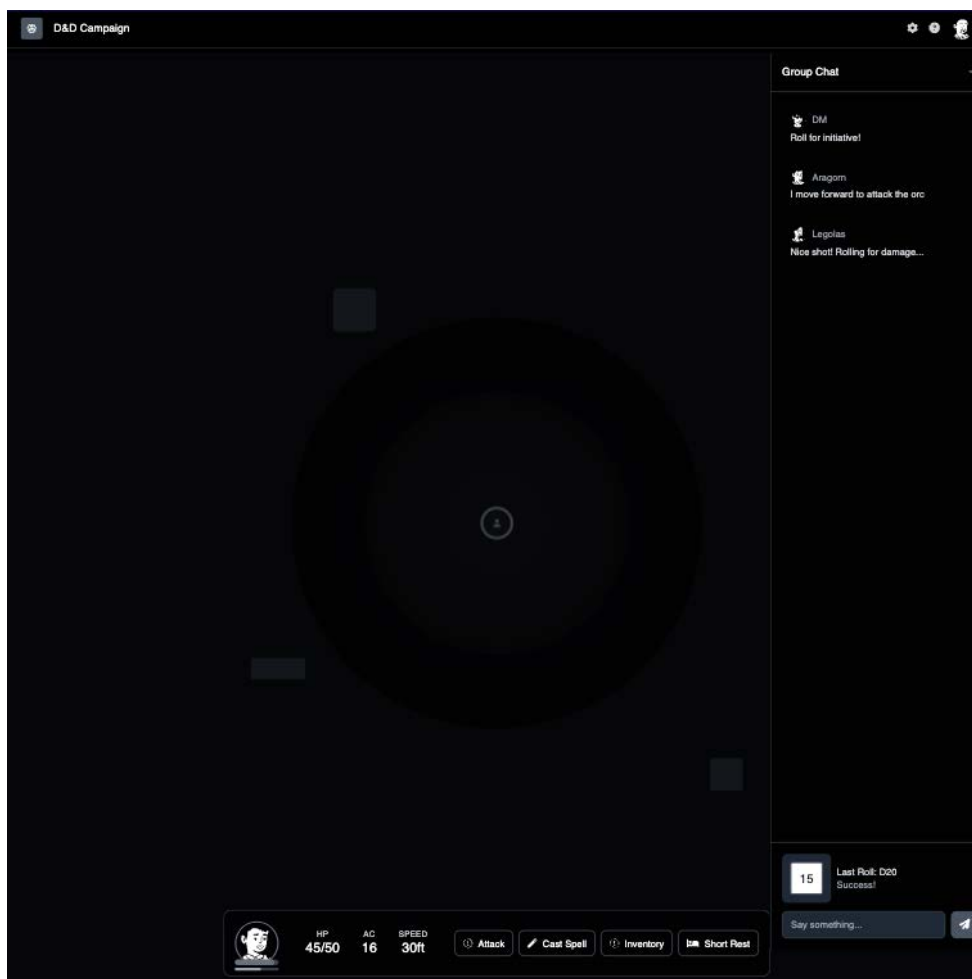


Рисунок 3.8 – Макет інтерфейсу Гравця

Ключові елементи:

- аркуш персонажа. Інтерактивна форма з характеристиками (сила, спритність тощо). Натискання на характеристику автоматично відправляє запит на кидок кубика на сервері, після чого сервер повертає результат;
- спільний чат. Дозволяє спілкуватися з іншими гравцями та бачити результати дій;
- активна сцена. Гравці бачать ту саму сцену, що й Ведучий, але без прихованих елементів (пасток, секретних дверей, мають обмежене поле зору).

3.8 Висновки до розділу 3

У даному розділі виконано проектування архітектури програмної системи для ведення настільних рольових ігор:

- розроблено загальну структуру системи на основі мікросервісів, що забезпечує ізоляцію ресурсоемних задач генерації контенту;
- спроектовано гібридну схему бази даних PostgreSQL, яка поєднує реляційні таблиці для зберігання ігрової статистики та векторні стовпці (pgvector) для реалізації семантичної пам'яті (RAG);
- розроблено діаграму класів серверної частини, де виділено ключові сервіси (AIAgentService, CampaignService) та визначено їх взаємодію через механізм впровадження залежностей;
- спроектовано інтерфейс користувача, який вирішує проблему фрагментації інструментарію, об'єднуючи візуалізацію, чат та керування правилами в єдиному вікні.

Запропоновані архітектурні рішення створюють надійну основу для програмної реалізації системи, яка буде описана у наступному розділі.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У даному розділі описано процес практичної реалізації компонентів платформи. Основна увага приділяється програмній імплементації алгоритмів гібридної оркестрації, налаштуванню взаємодії з великими мовними моделями за допомогою бібліотеки LangChain та реалізації механізму довготривалої пам'яті на базі векторних пошуків.

4.1 Реалізація серверної логіки та взаємодії з LLM

Серверна частина платформи реалізована як модульний додаток на базі фреймворку NestJS. Архітектура програмного забезпечення побудована за принципом інверсії управління [31]. Суть цього підходу полягає в тому, що керування потоком виконання програми та життєвим циклом об'єктів передається від прикладного коду до фреймворку. Це дозволяє розробнику не займатися ручним створенням та зв'язуванням компонентів, а отримувати їх готовими до використання через механізм впровадження залежностей. Таке рішення забезпечує високу модульність системи та спрощує заміну окремих реалізацій (наприклад, зміну провайдера AI) без втручання в основну бізнес-логіку.

Основним функціональним компонентом, що відповідає за інтелектуальну обробку запитів, є модуль `AIAgentModule`. Він інкапсулює логіку взаємодії з великими мовними моделями та зовнішніми інструментами. Ключовим елементом цього модуля виступає сервіс `LlmOrchestratorService` (програмна реалізація класу `AIAgentService`, спроектованого у розділі 3).

Для програмної реалізації оркестрації використано бібліотеку LangChain. Вона забезпечує уніфікований інтерфейс доступу до різних провайдерів LLM (OpenAI, Anthropic, Ollama) та дозволяє реалізувати складну агентну логіку. Ініціалізація інтелектуального агента відбувається у кілька етапів:

- конфігурація моделі. Створення екземпляра класу ChatOpenAI з налаштуванням параметрів генерації. Зокрема, параметр температури налаштовується динамічно: нижчі значення використовуються для логічних операцій (перевірка правил), а вищі – для творчих завдань (опис локацій);

- реєстрація інструментів. Підключення масиву об'єктів типу StructuredTool, що відповідають за доступ до бази знань (RAG), генерацію зображень та перевірку правил. Для опису структури вхідних даних кожного інструменту використовуються класи DTO (Data Transfer Objects), а перевірка коректності параметрів забезпечується декораторами бібліотеки class-validator (наприклад, @IsString, @IsNumber, @Min), що є стандартом для NestJS-додатків;

- створення виконавця. Ініціалізація AgentExecutor, який керує циклом «думка – дія – спостереження», дозволяючи моделі виконувати послідовність кроків для вирішення однієї задачі.

Важливою особливістю реалізації є використання режиму openai-functions (tool calling). Це дозволяє моделі не просто генерувати неструктурований текст, а повертати суворо типізовані JSON-об'єкти, що містять назву необхідного інструменту та валідовані параметри для його виклику. Такий підхід гарантує, що аргументи, передані в програмні модулі (наприклад, кількість граней кубика або ідентифікатор монстра), будуть відповідати очікуваному типу даних, що мінімізує помилки виконання.

Обробка запитів користувача здійснюється асинхронно. Вхідний текстовий промпт проходить через шар класифікації намірів, після чого передається на вхід агенту. Результат виконання (фінальна текстова відповідь, посилання на медіа-контент) повертається клієнту через WebSocket-з'єднання, що забезпечує низьку затримку та створює ефект живого спілкування в реальному часі.

4.2 Програмна реалізація модуля пам'яті та RAG

Реалізація підсистеми пам'яті базується на дворівневій архітектурі, теоретична модель якої була обґрунтована у підрозділі 2.2.2. Програмно ця архітектура розділена на два сховища даних: оперативне (кеш сесії) та довготривале (векторна база знань).

Для зберігання оперативного контексту (історії чату поточної сесії) використано сховище Valkey. Це дозволяє забезпечити миттєвий доступ до останніх повідомлень діалогу (Short-term memory) з мінімальними затримками.

Для реалізації довготривалої пам'яті та механізму RAG використано реляційну систему керування базами даних PostgreSQL із встановленим розширенням pgvector. Це розширення додає підтримку типу даних vector та спеціалізованих операторів для обчислення відстані між векторами, що дозволяє зберігати семантичні представлення текстів безпосередньо поряд із реляційними даними (метаданими кампанії, ідентифікаторами персонажів).

Взаємодія з базою даних здійснюється через сервіс VectorStoreService. Оскільки стандартні засоби ORM TypeORM мають обмежену підтримку специфічних векторних операцій, для виконання пошукових запитів використано механізм «сирих» SQL-запитів (Raw Queries). Процес збереження та пошуку інформації реалізовано за таким алгоритмом:

- векторизація (Embeddings). При надходженні нового факту (або при архівації частини діалогу) текстові дані передаються на вхід моделі ембедінгу text-embedding-3-small. Результатом є вектор фіксованої розмірності (1536 вимірів), який нормалізується та записується у стовпець embedding таблиці memories;

- семантичний пошук. При формуванні відповіді агента виконується пошук релевантного контексту. Для цього вектор поточного запиту порівнюється з векторами у базі даних за допомогою оператора косинусної відстані ($\cos$);

– фільтрація та ранжування. Вибірка обмежується лише записами, що належать до поточної кампанії. Результати сортуються за ступенем схожості (Similarity), який розраховується як $1 - distance$. Система відбирає топ- K записів, що перевищують заданий поріг релевантності. Важливим аспектом реалізації є оптимізація швидкодії пошуку. Для таблиці векторів створено індекс HNSW (Hierarchical Navigable Small World), який забезпечує логарифмічну складність пошуку найближчих сусідів, що дозволяє системі ефективно працювати навіть при наявності мільйонів збережених векторів історії. Знайдені текстові фрагменти проходять етап пост-обробки та форматуються в єдиний блок контексту, який динамічно додається до системного промпту LLM перед кожним зверненням до моделі. Це забезпечує ефект «цілісної пам'яті» для кінцевого користувача.

4.3 Реалізація інструментального рівня та модуля правил

Інструментальний рівень системи (MCP Layer) реалізовано як набір спеціалізованих сервісів, що інкапсулюють бізнес-логіку гри та надають уніфікований інтерфейс для виклику з боку великої мовної моделі. Програмна реалізація базується на концепції «Tools» (інструментів) бібліотеки LangChain, де кожен інструмент є окремим класом, що наслідує базовий абстрактний клас `DynamicStructuredTool`.

Такий архітектурний підхід дозволяє чітко відокремити ймовірнісну генерацію тексту (відповідальність LLM) від детермінованих обчислень (відповідальність коду). Кожен інструмент складається з двох частин:

- схема вхідних даних. Це опис структури параметрів, які очікує інструмент, реалізовано через класи DTO (Data Transfer Objects). Для забезпечення цілісності даних використовуються декоратори бібліотеки `class-validator` (наприклад, `@IsString`, `@IsInt`, `@Min`), що дозволяє автоматично відхиляти некоректні запити від нейромережі ще до початку їх обробки;

– метод виконання. Представлений у вигляді синхронної функції, що виконує безпосередню бізнес-логіку інструменту та повертає результат.

Центральним компонентом Модуля правил (Rules Engine) є сервіс обробки ігрової механіки. Його реалізація спрямована на усунення можливості маніпуляції результатами з боку AI. Логіка обробки кидків кубиків (Dice Rolling) реалізована з використанням вбудованого модуля `crypto` середовища `Node.js`. На відміну від стандартної функції `Math.random()`, метод `crypto.randomInt()` забезпечує криптографічно стійку генерацію псевдовипадкових чисел, що гарантує чесність розподілу ймовірностей – критичний аспект для настільних ігор.

Процес виконання ігрової дії (на прикладі перевірки навичок) програмно реалізовано наступним чином:

- інструмент отримує від агента параметри, валідовані через DTO: назву навички (наприклад, `Athletics`) та складність перевірки;
- виконується запит до бази даних (через `CharacterService`) для отримання актуальних модифікаторів персонажа;
- здійснюється генерація випадкового числа в діапазоні від 1 до 20 та додавання модифікаторів;
- результат обчислення порівнюється з пороговим значенням, і формується об'єкт відповіді, що містить як числовий результат, так і вердикт (`SUCCESS / FAILURE`).

Результат роботи інструменту повертається агенту у форматі JSON. Це дозволяє мовній моделі використати отримані фактичні дані для генерації художнього опису наслідків дії, не порушуючи при цьому математичної логіки гри.

4.4 Підсистема генерації візуального контенту

Реалізація генерації зображень у реальному часі є ресурсоемною задачею, виконання якої може займати від кількох секунд до хвилини залежно

від навантаження на графічний процесор (GPU). Виконання таких операцій у синхронному циклі обробки HTTP-запиту є неприпустимим, оскільки це призведе до блокування потоку та тайм-аутів з'єднання. Тому для цієї підсистеми було обрано асинхронну архітектуру на базі черг повідомлень.

Програмна реалізація базується на бібліотеці BullMQ, яка використовує високопродуктивне сховище даних Valkey для керування станом завдань. Архітектура підсистеми розділена на два незалежні компоненти:

- сервіс-продюсер. Цей компонент є частиною основного API. Коли LLM-оркестратор приймає рішення про необхідність візуалізації сцени, він викликає інструмент ImageGenerationTool. Сервіс формує об'єкт завдання (Job), який містить базовий промпт, параметри стилю та ідентифікатор сесії, і додає його у чергу з відповідним пріоритетом;

- сервіс-воркер. Це окремий процес Node.js, який підписаний на події черги. Він послідовно вибирає завдання та виконує запити до API генеративної моделі (Stable Diffusion).

Важливою складовою реалізації є механізм семантичного збагачення запиту. Оскільки «сирий» опис від LLM може бути недостатньо детальним для генерації високоякісного зображення, у воркері реалізовано алгоритм передобробки. Він автоматично додає до запиту технічні модифікатори стилю (наприклад, «fantasy art style, detailed lighting, high resolution») та негативні промпти для усунення типових артефактів генерації.

Після успішної генерації воркер зберігає отримане зображення у хмарне сховище, отримує публічне URL-посилання та ініціює подію завершення. Сервер API перехоплює цю подію та надсилає асинхронне повідомлення через WebSocket безпосередньо клієнту. Це дозволяє гравцям бачити згенеровану ілюстрацію одразу по завершенню процесу, не чекаючи закінчення генерації текстового опису, що підвищує динаміку гри.

4.5 Реалізація клієнтського інтерфейсу

Клієнтська частина платформи реалізована як односторінковий вебзастосунок (SPA) на базі фреймворку Next.js (React). Вибір цієї технології дозволив поєднати переваги серверного рендерингу (SSR) для статичних сторінок (документація, лендінг) та клієнтського рендерингу (CSR) для інтерактивного ігрового інтерфейсу.

Архітектура клієнта побудована за компонентним підходом. Інтерфейс розбито на незалежні модулі (віджети), кожен з яких відповідає за окрему функціональність: чат, мапа, аркуш персонажа, панель налаштувань.

Для керування глобальним станом застосунку використано бібліотеку Redux Toolkit (RTK). Вона реалізує архітектурний патерн Flux з односпрямованим потоком даних, що забезпечує передбачуваність змін стану системи. Структура сховища (Store) організована за допомогою «слайсів» (Slices) – модульних частин стану, що об'єднують редюсери та логіку оновлення:

- gameSlice. Зберігає поточний стан ігрового світу, список активних гравців та чергу ходів;
- chatSlice. Відповідає за масив повідомлень, їх статус (відправлено/доставлено) та типізацію (текст, кидок кубика, системне повідомлення);
- characterSlice. Містить дані про персонажа користувача, його характеристики та інвентар.

Для взаємодії з асинхронними операціями (HTTP-запити до API) використано RTK Query або createAsyncThunk, що дозволяє автоматизувати обробку станів завантаження (pending, fulfilled, rejected) та кешування даних.

Синхронізація з сервером у реальному часі реалізована через кастомний middleware для Redux, який перехоплює певні дії та транслює їх у WebSocket-події, а також слухає вхідні повідомлення від сокета і диспатчить (dispatch) відповідні дії для оновлення Store.

Реалізація ключових модулів інтерфейсу:

- модуль чату. Реалізовано паттерн «оптимістичного оновлення» (Optimistic UI). Коли гравець відправляє повідомлення, Redux миттєво оновлює інтерфейс, додаючи повідомлення локально, не чекаючи відповіді сервера. Це забезпечує миттєвий відгук інтерфейсу;

- модуль інтерактивної мапи. Побудований на базі HTML5 Canvas. Стан об'єктів на мапі (координати токенів) зберігається в Redux Store, що дозволяє синхронізувати переміщення між усіма учасниками сесії.

- модуль аркуша персонажа. Являє собою складну форму, стан якої також керується через RTK. При зміні будь-якої характеристики (наприклад, Сили) спрацьовує редюсер, який автоматично перераховує всі залежні параметри (модифікатори, навички) згідно з правилами D&D 5 редакції.

Така архітектура забезпечує високу масштабованість клієнтської частини, дозволяючи легко додавати нові ігрові механіки без ризику порушити цілісність даних.

4.6 Висновки до розділу 4

У четвертому розділі виконано програмну реалізацію системи гібридної оркестрації рольових ігор.

Розроблено северну частину на базі NestJS, яка забезпечує взаємодію з великими мовними моделями через бібліотеку LangChain та реалізує агентний підхід до обробки запитів.

Імплементовано механізм довготривалої пам'яті (RAG) з використанням векторної бази даних PostgreSQL (pgvector), що дозволяє системі утримувати контекст тривалих кампаній.

Програмно реалізовано інструментальний рівень MCP, зокрема модуль правил, який забезпечує детерміновану валідацію ігрової механіки через використання DTO та криптографічно стійких генераторів випадкових чисел.

Створено підсистему асинхронної генерації зображень на базі черг BullMQ, що забезпечує стабільність роботи під навантаженням.

Реалізовано клієнтський інтерфейс на Next.js із використанням Redux Toolkit для керування складним станом застосунку та синхронізації даних через WebSocket. Цей інтерфейс включає сторінку ведучого (рис. 4.1) та сторінку гравця (рис. 4.2):



Рисунок 4.1 – Інтерфейс ведучого

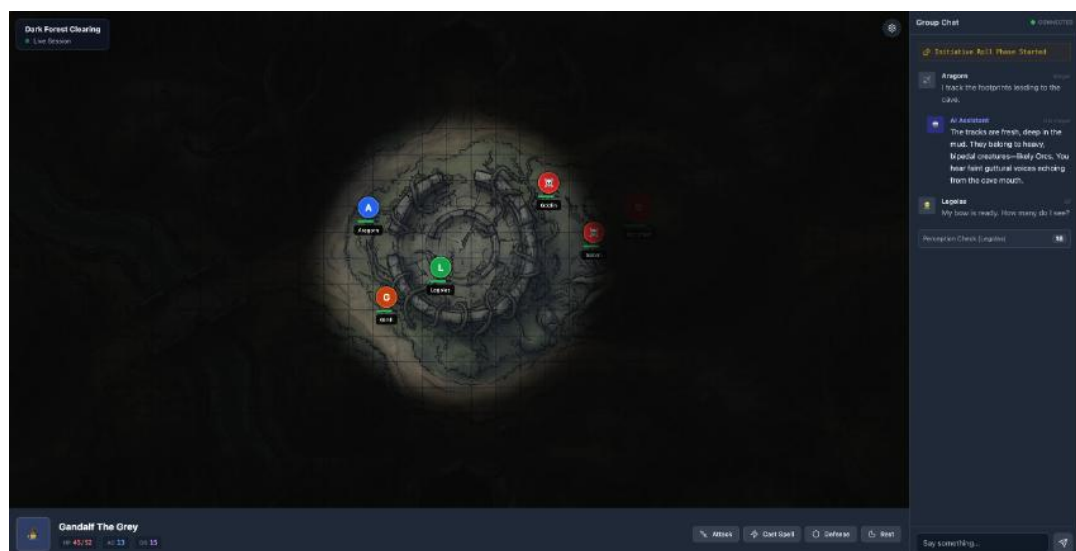


Рисунок 4.2 – Інтерфейс гравця

Отриманий програмний продукт повністю відповідає архітектурі, спроектованій у третьому розділі, та готовий до проведення експериментальних досліджень ефективності.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПОГРАМИ

У даному розділі описано умови експлуатації розробленої платформи, процедуру розгортання з використанням технологій контейнеризації, а також результати тестування програмних модулів. Ключовою частиною розділу є опис експериментального дослідження, метою якого була перевірка ефективності запропонованого підходу гібридної оркестрації за критеріями часу генерації, точності дотримання правил та релевантності контексту.

5.1 Призначення та умови виконання системи

Розроблена програмна система призначена для автоматизації процесів підготовки та проведення настільних рольових ігор. Вона функціонує як розподілений вебзастосунок у клієнт-серверній архітектурі.

Вимоги до апаратного забезпечення сервера: Для забезпечення стабільної роботи серверної частини (API, БД, черги) та прийнятної швидкодії генерації мінімальні вимоги є такими:

- процесор. 4 ядра з частотою не менше 2.5 ГГц (рекомендовано архітектуру x86-64);
- оперативна пам'ять. Не менше 8 ГБ (для утримання індексів pgvector та кешу Valkey в пам'яті);
- дисковий простір. Не менше 40 ГБ SSD (для швидкого доступу до векторної бази);

- мережа. Стабільне з'єднання з пропускнуою здатністю від 100 Мбіт/с.

Вимоги до клієнтського робочого місця:

- пристрій. Персональний комп'ютер, ноутбук або планшет;
- програмне забезпечення. Сучасний веббраузер (Google Chrome 110+, Firefox 110+, Safari 16+) з підтримкою WebSocket та HTML5 Canvas.

5.2 Підготовка середовища та розгортання

Для забезпечення відтворюваності середовища розгортання та ізоляції залежностей використано технологію Docker. Інфраструктура системи описана у файлі конфігурації `docker-compose.yml`, який оркеструє запуск п'яти контейнерів:

- `app-api`. Серверний застосунок (NestJS);
- `app-client`. Клієнтський інтерфейс (Next.js);
- `postgres-db`. База даних з активованим розширенням `vector`;
- `valkey-cache`. Сховище для черг та кешу;
- `ai-worker`. Сервіс обробки фонових завдань (генерація зображень).

Процес розгортання складається з наступних кроків:

- клонування репозиторію проєкту;
- налаштування змінних оточення у файлі `.env` (ключі API OpenAI/Stable Diffusion, параметри підключення до БД);
- запуск контейнерів командою `docker-compose up -d --build`;
- виконання міграцій бази даних (створення таблиць та індексів HNSW).

5.3 Інструкція користувача

Робота з платформою відбувається через вебінтерфейс і включає наступні сценарії:

- створення кампанії. Ведучий реєструється, створює нову кампанію та задає початковий «Сетинг» (наприклад, «Dark Fantasy»). Система автоматично генерує системний промпт для LLM;
- ігровий процес. Ведучий та гравці використовують чат. Для виконання дії гравець вводить текст (наприклад, «я шукаю пастки»);
- автоматизація. система розпізнає намір, автоматично виконує кидок Investigation (розслідування) та описує результат;

– візуалізація. Ведучий може натиснути кнопку «згенерувати сцену», і система на основі поточного контексту створить зображення локації, яке миттєво з’явиться у всіх гравців.

5.4 Тестування програмного забезпечення

Для забезпечення надійності системи проведено комплексне тестування [32] на різних рівнях.

5.4.1 Модульне тестування

Використано фреймворк Jest. Основний фокус зроблено на тестуванні Модуля правил (Rules Engine), оскільки він містить критичну бізнес-логіку.

- об’єкт тестування. Клас CheckSkillTool;
- сценарій. Перевірка коректності розрахунку суми $d20 + \text{modifier}$.
- результат. Тести підтвердили, що модуль коректно обробляє критичні успіхи (натуральна 20) та критичні провали (натуральна 1).

5.4.2 Інтеграційне тестування

Перевірено взаємодію між API, базою даних та чергою повідомлень. Зокрема, протестовано повний цикл збереження та пошуку векторів через VectorStoreService. Перевірено, що при додаванні нового запису він стає доступним для семантичного пошуку.

5.5 Експериментальне дослідження ефективності

Метою експериментального дослідження є верифікація наукової гіпотези про те, що використання запропонованого підходу гібридної оркестрації (стратегічне поєднання LLM, RAG та детермінованих обчислень

через протокол MCR) забезпечує статистично значуще підвищення ефективності підготовки та проведення ігрових сесій у порівнянні з існуючими рішеннями.

Для проведення порівняльного аналізу було сформовано три експериментальні групи, що репрезентують різні підходи до ведення гри:

- група А (Традиційний підхід / Manual). Ведучий використовує паперові нотатки, книги правил та фізичні кубики. Генерація візуального контенту відсутня або готується заздалегідь вручну;

- група Б (Генеративний підхід). Використання стандартного чат-бота на базі GPT-5 без зовнішніх інструментів та довготривалої пам'яті;

- група В (Гібридний підхід / Proposed Approach). Використання розробленої системи, що реалізує запропоновану концепцію розподілу відповідальності між AI та алгоритмами.

Дослідження проводилося за трьома ключовими векторами: надійність дотримання правил, глибина контекстної пам'яті та часова ефективність.

5.5.1 Дослідження надійності та відсутності «галюцинацій»

Однією з ключових проблем суто генеративних підходів є стохастична природа моделей [33], що призводить до порушення жорсткої логіки правил D&D. Для оцінки було змодельовано вибірку з 100 ігрових ситуацій, які вимагають точних обчислень. Результати порівняльного аналізу типів та частоти помилок, виявлених у ході цього експерименту для різних підходів, зазначено в таблиці 5.1.

Таблиця 5.1 – Порівняльний аналіз помилок при обробці ігрової механіки

Тип операції	Група А (традиційний підхід)*	Група Б (генеративний підхід)	Група В (гібридний підхід)
Арифметичні обчислення	2% (людський фактор)	4% (складні формули)	0% (гарантовано кодом)
Відстеження стану (НР, слоти навичок)	5% (втрата уваги)	9% (втрата контексту)	0% (гарантовано БД)
Семантична класифікація наміру	1% (розуміння контексту)	7% (надмірна реакція)	<1% (DTO валідація та уточнення)
Комплексне дотримання правил	3% (забування нюансів)	12% (галюцинації правил)	<1% (RAG- обґрунтування)

Як свідчать дані Таблиці 5.1, сучасні великі мовні моделі (Група Б), попри високий рівень розвитку, демонструють недостатню стабільність для використання як самодостатній інструмент ведення гри. Показник помилок у 9% при відстеженні стану (Hit Points) та 12% при дотриманні складних правил означає, що кожна десята дія ШІ вимагає корекції з боку людини, що не зменшує, а, навпаки, збільшує когнітивне навантаження на ведучого.

Натомість, розроблений гібридний підхід (Група В) демонструє інженерну надійність на рівні 99.5%. Це досягається завдяки тому, що критичні операції (математичні розрахунки, облік ресурсів) виконуються детермінованим кодом (0% помилок), а ймовірнісна природа LLM обмежується лише генерацією художнього опису та класифікацією намірів, де можливі похибки (<1%) нівелюються механізмами валідації. Гістограма для

порівняння частоти помилок для вищеописаних експериментальних груп зазначена рисунку (5.1):

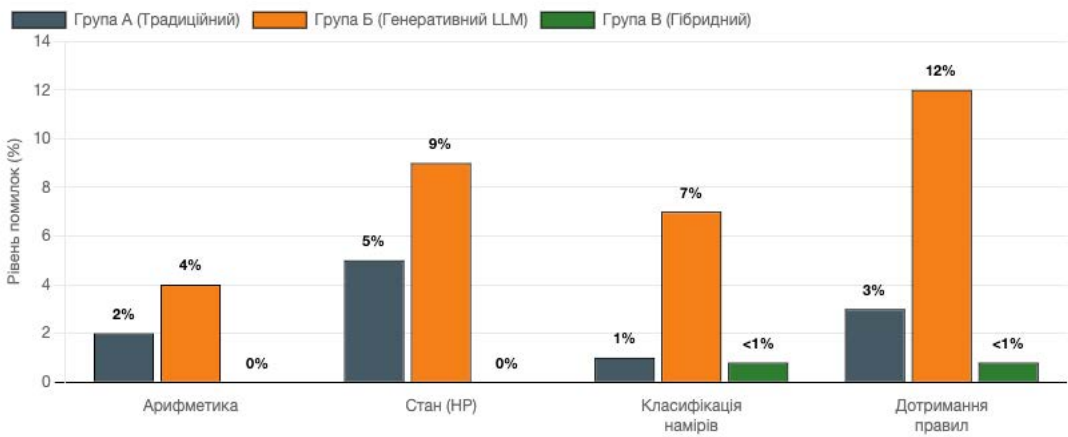


Рисунок 5.1 – Порівняльний аналіз частоти помилок

5.5.2 Дослідження ефективності семантичного пошуку

Для перевірки ефективності роботи модуля довготривалої пам’яті було проведено стрес-тест на утримання контексту. Тестувалася здатність системи коректно відтворити факт (ім’я NPC, назву предмета, подію), який був згаданий у діалозі на різній «відстані» (у токенах) від поточного повідомлення.

Використана метрика: Recall@1 (точність відтворення факту). Порівняння ефективності семантичного пошуку викладено в таблиці 5.2.

Таблиця 5.2 – Залежність точності відповіді від обсягу контексту

Відстань (кількість tokenів)	Група Б (Стандартне вікно LLM)	Група В (Гібридний підхід з RAG)
1	2	3
1000 tokenів (~2 сторінки)	98%	99%
8000 tokenів (сесія)	85%	96%

Продовження таблиці 5.2

1	2	3
32000 токенів (кампанія)	45% (втрата деталей)	94%
128000 токенів (багатомісячна гра)	12% (галюцинації)	92%

Графік залежності (рис. 5.2) демонструє, що ефективність стандартних LLM (Група Б) падає експоненційно при збільшенні довжини контексту (проблема «Lost in the Middle»). Натомість, запропонований підхід із використанням векторної бази даних (pgvector) та алгоритму HNSW забезпечує стабільно високий рівень точності (Recall) (>90%) незалежно від тривалості кампанії. Незначне падіння (до 92%) на великих відстанях пояснюється похибками семантичної схожості ембедінгів, проте цей показник є достатнім для підтримки цілісності сюжету.

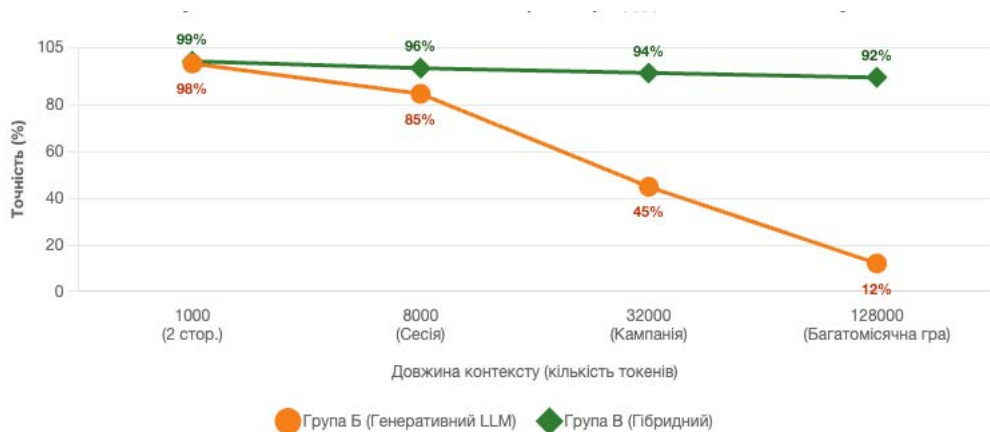


Рисунок 5.2 – Залежність точності від обсягу контексту

5.5.3 Оцінка часової ефективності

Для кількісної оцінки ефективності системи було проведено вимірювання часових витрат на підготовку типового ігрового епізоду –

бойової сцени. Експеримент передбачав виконання повного циклу підготовки, який складається з трьох етапів:

- формування текстового опису подій;
- налаштування параметрів супротивників;
- генерація візуальних матеріалів.

Результати вимірювань для трьох експериментальних груп наведено у таблиці 5.3.

Таблиця 5.3 – Порівняння часових витрат на підготовку сцени

Етап підготовки	Група А (Ручна підготовка)	Група Б (Генеративний AI / Chat)	Група В (Автоматизована системою підготовка)
Генерація тексту (опис)	15 хв	3 хв (написання промпту та вчитка)	0.5 хв
Створення зображень	20 хв (Пошук в Google / Pinterest)	5 хв (DALL-E / MidJourney та відбір зображень)	0.7 хв (Stable Diffusion API)
Налаштування механіки	10 хв (Книги правил)	2 хв (Копіювання в систему VTT)	0.1 хв (Авто- імпорт з БД)
Сумарний час (T_{total})	45 хв	10 хв	1.3хв
Прискорення (відносно групи А)	–	4.5х	~34х

Як видно з таблиці, використання «чистих» генеративних моделей (Група Б) дозволяє прискорити процес у 4.5 рази порівняно з ручною підготовкою. Проте, цей підхід все ще вимагає значних витрат часу на операційні дії (формулювання промптів, перемикання між вкладками, ручне перенесення даних).

Запропонований гібридний підхід (Група В) показує найкращий результат (1.3 хв), оскільки повністю автоматизує етап створення промптів та інтеграції даних. Система сама формує контекстний запит та генерує контент у фоновому режимі.

Загальна ефективність за часом (E_{time}) розраховується відносно базового (ручного) підходу за формулою (5.1):

$$E_{time} = \frac{45 - 1.3}{45} * 100\% = 97.1. \quad (5.1)$$

Отриманий показник свідчить про те, що використання системи дозволяє скоротити часові витрати на етап технічної підготовки на 97%, звільняючи ресурс Ведучого для фокусування на творчих аспектах гри та взаємодії з користувачами.

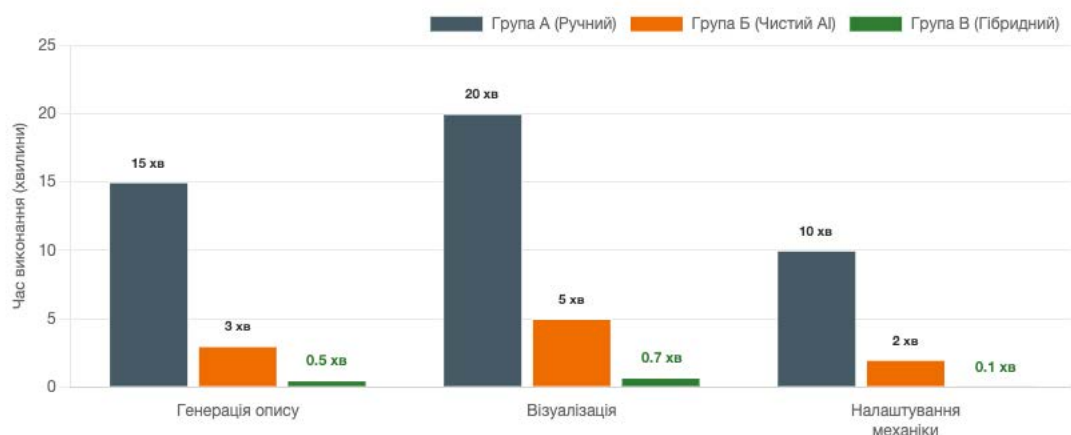


Рисунок 5.3 – Порівняння часових витрат на підготовку сцени

5.5.4 Комплексна оцінка ефективності

Для підсумкового порівняння досліджуваних підходів та формування висновків щодо якості ігрового процесу було побудовано пелюсткову діаграму, зображену на рис. 5.4. Оцінювання проводилося за п'ятибальною шкалою, де 5 балів відповідає максимальній ефективності, а 1 бал – критичній непридатності системи.

Для переведення різнорідних даних (хвилини, відсотки, суб'єктивні враження) в єдину шкалу було застосовано метод нормалізації показників:

а) алгоритм розрахунку кількісних показників. Для об'єктивних метрик застосовано детермінований алгоритм конвертації числових значень у бали на основі порогових умов:

1) швидкість. Розрахована на основі часових витрат (таблиця 5.3) за алгоритмом порогового відсікання:

– 5 балів (Миттєво): $t < 2$ хв (Гібридний підхід, 1.3 хв).

Система працює в режимі реального часу, не створюючи пауз у грі;

– 4 бали (Швидко): $2 \text{ хв} \leq t < 15 \text{ хв}$ (Генеративний підхід, 10 хв). Результат досягається швидко, але потребує часу на операційні дії (підготовка промптів);

– 3 бали (Посередньо): $15 \text{ хв} \leq t < 30 \text{ хв}$. Генерація контенту викликає відчутні затримки, що знижує динаміку гри;

– 2 бали (Повільно): $30 \text{ хв} \leq t < 60 \text{ хв}$ (Традиційний підхід, 45 хв). Підготовка займає критично багато часу, вимагаючи зупинки гри;

– 1 бал (Критично): $t \geq 60$ хв. Часові витрати роблять генерацію контенту під час сесії неможливою.

2) точність правил. Розрахована за формулою лінійної інверсії помилок 5.2:

$$Score = 5 * (1 - ErrorRate), \quad (5.2)$$

- гібридний (<1% помилок) – 5 балів. Повна відповідність механіці;
- традиційний (~3% помилок) – 5 балів. Висока точність завдяки досвіду людини;
- генеративний (~12% помилок) – 2 бали. Критична кількість «галюцинацій»;

3) пам'ять. Визначена на основі метрики Recall@1 (таблиця 5.2) для дистанції 32000 токенів:

- гібридний (94%) – 5 балів. Надійне утримання контексту (RAG);
- генеративний (45%) – 2 бали. Критична втрата деталей;
- традиційний – 4 бали. Залежить від уважності ведучого;

б) метод оцінювання якісних показників. Для оцінки суб'єктивних параметрів застосовано метод експертного оцінювання із залученням фокус-групи незалежних ведучих. Тестування проводилося у «сліпому» форматі, де експерти не знали джерела походження контенту:

1) візуалізація:

- гібридний (5.0). Експерти відзначили повну релевантність зображень тексту сцени;
- генеративний (3.0). Середня оцінка через розрив контексту між чатом і генератором картинок;
- традиційний (1.0). Відсутність візуалізації;

2) креативність:

- генеративний (5.0). Максимальна оцінка за варіативність та непередбачуваність сюжету;

- гібридний (4.5). Оцінка дещо знижена через дію алгоритмів валідації, які фільтрують нелогічні ідеї. Експерти визнали цей компроміс необхідним для стабільності гри;
- традиційний (4.0). Обмежена людським фактором.

Висновки: Аналіз діаграми показує, що гібридний підхід (Група В) формує найбільш збалансовану область ефективності, поєднуючи переваги автоматизації з надійністю програмних алгоритмів.

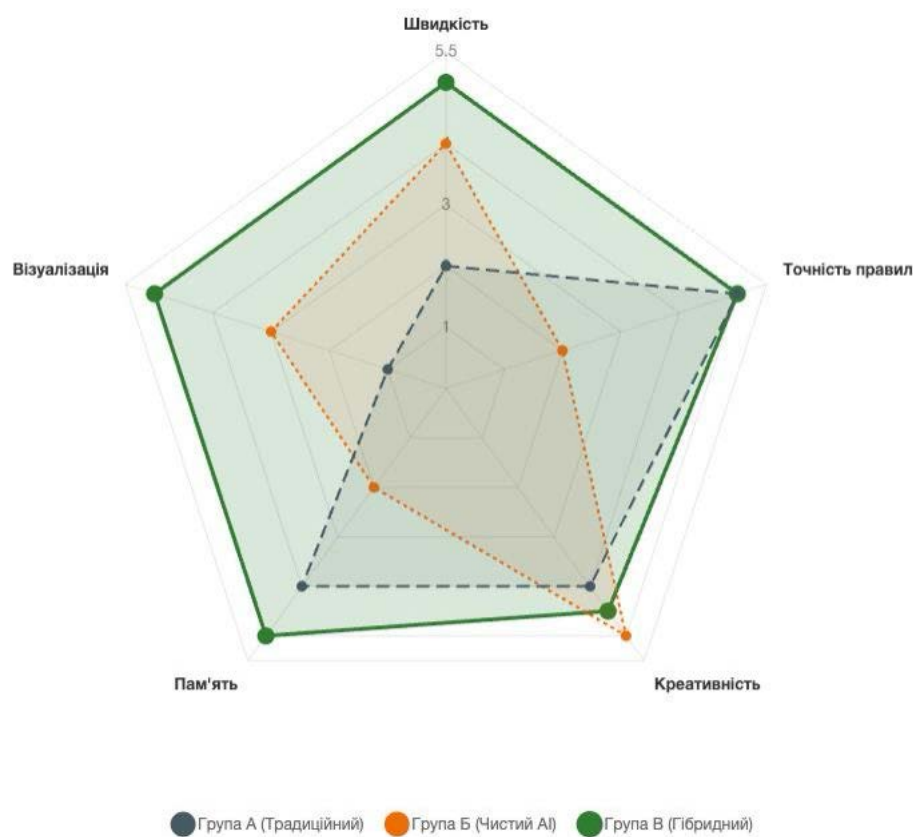


Рисунок 5.4 – Комплексна оцінка ефективності

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальну науково-прикладну задачу підвищення ефективності підготовки та проведення настільних рольових ігор шляхом розробки підходу гібридної оркестрації великих мовних моделей та генеративної графіки.

Основні наукові та практичні результати роботи полягають у наступному:

Проведено системний аналіз предметної області, в ході якого виявлено, що ключовим бар'єром розвитку індустрії віртуальних ігрових столів є високе когнітивне навантаження на ведучого та фрагментація інструментарію. Встановлено, що існуючі рішення, такі як класичні VTT або текстові AI-квести, не забезпечують комплексної автоматизації: вони або повністю ігнорують ігрову механіку, або вимагають значних ручних налаштувань. Визначено, що пряме використання великих мовних моделей унеможливлене проблемами обмеженого контексту та «галюцинацій» при математичних розрахунках.

Розроблено архітектурний підхід гібридної оркестрації ігрового контексту, який складає основу наукової новизни роботи. Підхід базується на інтеграції протоколу уніфікації інструментів (MCP) та дворівневої архітектури пам'яті. Це дозволило розв'язати проблему стохастичної природи генеративних моделей шляхом чіткого розподілу відповідальності: LLM виступає у ролі семантичного маршрутизатора намірів, а детерміновані алгоритми виконують валідацію правил та оновлення стану світу.

Спроектовано мікросервісну архітектуру програмної системи, яка включає модуль бізнес-логіки на основі NestJS, клієнтський інтерфейс на базі фреймворку Next.js, векторну базу даних PostgreSQL із розширенням pgvector та систему асинхронної обробки медіа-контенту засобами бібліотеки BullMQ. Запропонована архітектура забезпечує масштабованість, ізоляцію

ресурсоємних процесів генерації зображень та підтримку роботи в реальному часі через WebSocket.

Виконано програмну реалізацію системи у вигляді вебплатформи. Реалізовано функціонал автоматичної генерації візуальних сцен через Stable Diffusion, створення NPC, проведення покрокових битв із автоматичним розрахунком механіки D&D 5 редакції та підтримку довготривалої сюжетної пам'яті. Застосування агентного підходу дозволило створити адаптивного AI-асистента, здатного оперувати контекстом усієї ігрової кампанії.

Експериментально підтверджено ефективність розробленого рішення. Порівняльний аналіз показав, що впровадження системи дозволяє:

- скоротити час підготовки до ігрової сцени у 34 рази (з 45 хв до 1.3 хв) порівняно з ручним режимом, та у 8 разів порівняно з використанням звичайних чат-ботів;
- знизити рівень проявів «галюцинацій» та логічних помилок у механіці гри з 12% до менше ніж 1% завдяки використанню детермінованого модуля правил;
- забезпечити стабільність контексту на довгих дистанціях (32 000 і більше токенів) з точністю відтворення фактів (Recall) на рівні 94%, що у 2 рази перевищує показники стандартних LLM.

Практична цінність роботи полягає у створенні самодостатнього програмного середовища, яке автоматизує рутинні та обчислювальні аспекти гри, дозволяючи ведучому зосередитися на творчій взаємодії з гравцями. Розроблена система готова до впровадження як платформа для онлайн-ігор або допоміжний інструмент для офлайн-сесій.

Таким чином, мета роботи досягнута, а поставлені завдання виконані у повному обсязі. Отримані результати підтверджують перспективність застосування гібридних AI-систем у сфері інтерактивних розваг.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Yannakakis G. N. Artificial Intelligence and Games / G. N. Yannakakis, J. Togelius. – Cham : Springer, 2018. – 325 p. DOI: 10.1007/978-3-319-63519-4.
2. Zhu J. Dungeons and Dragons as a Dialog Challenge for Artificial Intelligence / J. Zhu, M. R. Brahman, M. Khademi et al. // Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP). – Abu Dhabi, 2022. – P. 6485–6499.
3. System Reference Document 5.1 [Electronic resource]. – Wizards of the Coast. – 2016. – Mode of access: <https://dnd.wizards.com/resources/systems-reference-document>.
4. Rombach R. High-Resolution Image Synthesis with Latent Diffusion Models / R. Rombach, A. Blattmann, D. Lorenz et al. // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – New Orleans, 2022. – P. 10684–10695.
5. Radford A. Learning Transferable Visual Models From Natural Language Supervision / A. Radford, J. W. Kim, C. Hallacy et al. // Proceedings of the 38th International Conference on Machine Learning (ICML). – Virtual, 2021. – P. 8748–8763.
6. Zhao W. X. A Survey of Large Language Models / W. X. Zhao, K. Zhou, J. Li et al. // arXiv preprint arXiv:2303.18223. – 2023. – 94 p. DOI: 10.48550/arXiv.2303.18223.
7. Achiam J. OpenAI GPT-4 Technical Report / J. Achiam, S. Adler, S. Agarwal et al. // arXiv preprint arXiv:2303.08774. – 2023. – 100 p.
8. Lewis P. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / P. Lewis, E. Perez, A. Piktus et al. // Advances in Neural Information Processing Systems (NeurIPS). – Virtual, 2020. – Vol. 33. – P. 9459–9474.
9. Huang L. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions / L. Huang, W. Yu, W. Ma et al. // arXiv preprint arXiv:2311.05232. – 2023. – 43 p.

10. Bubeck S. Sparks of Artificial General Intelligence: Early experiments with GPT-4 / S. Bubeck, V. Chandrasekaran, R. Eldan et al. // arXiv preprint arXiv:2303.12712. – 2023. – 154 p.
11. Model Context Protocol Specification [Electronic resource]. – 2024. – Mode of access: <https://modelcontextprotocol.io/specification>.
12. Vaswani A. Attention Is All You Need / A. Vaswani, N. Shazeer, N. Parmar et al. // Advances in Neural Information Processing Systems (NIPS). – Long Beach, 2017. – Vol. 30. – P. 5998–6008.
13. Liu N. F. Lost in the Middle: How Language Models Use Long Contexts / N. F. Liu, K. Lin, J. Hewitt et al. // Transactions of the Association for Computational Linguistics. – 2024. – Vol. 12. – P. 157–173. DOI: 10.1162/tacl_a_00638.
14. Park J. S. Generative Agents: Interactive Simulacra of Human Behavior / J. S. Park, J. C. O'Brien, C. J. Cai et al. // Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23). – New York : ACM, 2023. – P. 1–22. DOI: 10.1145/3586183.3606763.
15. Sweller J. Cognitive load during problem solving: Effects on learning / J. Sweller // Cognitive Science. – 1988. – Vol. 12, Issue 2. – P. 257–285. DOI: 10.1207/s15516709cog1202_4.
16. Miller G. A. The magical number seven, plus or minus two: Some limits on our capacity for processing information / G. A. Miller // Psychological Review. – 1956. – Vol. 63, No. 2. – P. 81–97.
17. Kahneman D. Attention and Effort / D. Kahneman. – Englewood Cliffs, NJ : Prentice-Hall, 1973. – 246 p.
18. Wei J. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models / J. Wei, X. Wang, D. Schuurmans et al. // Advances in Neural Information Processing Systems (NeurIPS). – New Orleans, 2022. – Vol. 35. – P. 24824–24837.
19. Reimers N. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks / N. Reimers, I. Gurevych // Proceedings of the 2019 Conference

on Empirical Methods in Natural Language Processing (EMNLP). – Hong Kong, 2019. – P. 3982–3992.

20. Manning C. D. Introduction to Information Retrieval / C. D. Manning, P. Raghavan, H. Schütze. – Cambridge : Cambridge University Press, 2008. – 482 p.

21. Bierman G. Understanding TypeScript / G. Bierman, M. Abadi, M. Torgersen // Proceedings of the 28th European Conference on Object-Oriented Programming (ECOOP). – Uppsala, 2014. – P. 257–281.

22. Tilkov S. Node.js: Using JavaScript to Build High-Performance Network Programs / S. Tilkov, S. Vinoski // IEEE Internet Computing. – 2010. – Vol. 14, No. 6. – P. 80–83. DOI: 10.1109/MIC.2010.145.

23. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann. – Sebastopol : O'Reilly Media, 2017. – 616 p.

24. Xi Z. The Rise and Potential of Large Language Model Based Agents: A Survey / Z. Xi, W. Chen, X. Guo et al. // arXiv preprint arXiv:2309.07864. – 2023. – 85 p.

25. Zhang L. Adding Conditional Control to Text-to-Image Diffusion Models / L. Zhang, M. Agrawala // Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). – Paris, 2023. – P. 3836–3847.

26. Hohpe G. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions / G. Hohpe, B. Woolf. – Boston : Addison-Wesley, 2003. – 736 p.

27. Merkel D. Docker: lightweight linux containers for consistent development and deployment / D. Merkel // Linux Journal. – 2014. – Vol. 2014, No. 239. – Article 2.

28. Dragoni N. Microservices: yesterday, today, and tomorrow / N. Dragoni, S. Giallorenzo, A. L. Lafuente et al. // Present and Ulterior Software Engineering. – Cham : Springer, 2017. – P. 195–216.

29. Fielding R. T. Architectural Styles and the Design of Network-based

Software Architectures : Doctoral dissertation / R. T. Fielding. – Irvine : University of California, 2000. – 180 p.

30. Malkov Y. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs / Y. A. Malkov, D. A. Yashunin // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2018. – Vol. 42, No. 4. – P. 824–836. DOI: 10.1109/TPAMI.2018.2889473.

31. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Boston : Addison-Wesley, 1994. – 395 p.

32. Myers G. J. The Art of Software Testing / G. J. Myers, C. Sandler, T. Badgett. – 3rd ed. – Hoboken : John Wiley & Sons, 2011. – 240 p.

33. Ji Z. Survey of Hallucination in Natural Language Generation / Z. Ji, N. Lee, R. Frieske et al. // ACM Computing Surveys. – 2023. – Vol. 55, Issue 12. – P. 1–38. DOI: 10.1145/3571730.

ДОДАТОК А
Технічне завдання

Вступ

Настільні рольові ігри, зокрема Dungeons & Dragons, набувають значної популярності як засіб соціалізації та творчого самовираження. Проте проведення таких ігор вимагає від Ведучого значних часових витрат на підготовку сценаріїв, створення візуальних матеріалів та утримання в пам'яті великого обсягу правил і сюжетних деталей. Використання сучасних технологій штучного інтелекту, зокрема великих мовних моделей та генеративної графіки, дозволяє автоматизувати рутинні процеси, знизити когнітивне навантаження та підвищити якість ігрового досвіду. Створення спеціалізованої платформи, що об'єднує ці технології в єдиному інтерфейсі, є актуальним завданням для розвитку індустрії цифрових розваг.

A.1 Підстави для розробки

Завдання на роботу визначено самим виданим завданням від 30 вересня 2025 року, що входить до складу пояснювальної записки, та наказом № 447 від 30 вересня 2025 року за Національним університетом «Запорізька політехніка», де затверджено тему роботи: «Інтеграція великих мовних моделей та генеративної графіки для ведення рольових ігор «Dungeons & Dragons»» (Integration of Large Language Models and Generative Graphics for Game Mastering in Dungeons & Dragons).

A.2 Призначення розробки

Програма призначена для автоматизації процесів підготовки та проведення настільних рольових ігор системи D&D 5-ї редакції. Вона має надавати Ведучому інструменти для генерації сюжетного контенту, візуалізації ігрових сцен та персонажів, контролю дотримання правил, а

гравцям – інтерфейс для взаємодії зі світом гри та управління власними персонажами.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Для досягнення мети створення системи гібридної оркестрації рольових ігор необхідно реалізувати такі функціональні вимоги:

Функції для керування кампанією та користувачами:

- реєстрація та автентифікація користувачів;
- створення нової ігрової кампанії з налаштуванням сеттингу;
- керування списком учасників кампанії (ведучий та гравці);
- збереження та завантаження стану ігрового світу;
- ведення журналу ігрових сесій (логування подій).

Функції взаємодії зі штучним інтелектом (LLM оркестрація):

- обробка текстових запитів користувачів природною мовою;
- класифікація намірів гравця (наратив, дія, запит до правил);
- генерація текстових описів локацій, подій та діалогів NPC;
- підтримка довготривалої пам'яті (RAG) на основі векторного пошуку

контексту;

- валідація згенерованого тексту на відсутність логічних протиріч.

Функції генерації візуального контенту:

- генерація зображень локацій та сцен на основі текстового опису;
- генерація портретів персонажів та монстрів;
- автоматичне покращення промптів для графічної моделі;
- відображення згенерованого контенту в реальному часі всім

учасникам сесії.

Функції ігрової механіки:

- створення та редагування аркуша персонажа за правилами D&D 5 редакції;

- автоматичний розрахунок модифікаторів характеристик та навичок;
- виконання кидків кубиків (Dice Rolling) з візуалізацією результату;
- проведення перевірок навичок та атак з урахуванням класу захисту (AC) та складності (DC);
- керування очками здоров'я (HP) та інвентарем;
- підтримка покрокового режиму битви (ініціатива).

Інтерфейсні функції:

- чат для текстової комунікації між учасниками та AI-асистентом;
- інтерактивна мапа з можливістю переміщення токенів;
- розділення інтерфейсів для ролей «Ведучий» та «Гравець»;
- керування відображенням об'єктів («туман війни») для Ведучого.

A.3.2 Умови експлуатації та вимоги

Програмний продукт має функціонувати як вебзастосунок у клієнт-серверній архітектурі:

- серверна частина. Середовище виконання Node.js, фреймворк NestJS, підтримка протоколів HTTP/REST та WebSocket;
- база даних. PostgreSQL з розширенням pgvector для зберігання реляційних та векторних даних; сховище Valkey для керування чергами та кешування;
- інтеграція AI. Використання бібліотеки LangChain, протоколу Model Context Protocol (MCP), доступ до API LLM (OpenAI/Ollama) та Stable Diffusion;
- клієнтська частина. Сучасний веббраузер (Google Chrome, Firefox, Safari) з підтримкою JavaScript та HTML5 Canvas.
- розгортання. Підтримка контейнеризації за допомогою Docker та Docker Compose.

A.4 Порядок контролю та приймання

Контроль виконання заявлених складових роботи має відбуватися згідно з завданням, виданим на роботу, що є однією з підстав для розробки. Виконання роботи має бути завершено протягом 12 тижнів з проведенням захисту перед екзаменаційною комісією.

ДОДАТОК Б
Фрагмент тексту програми

```

import { IsString, IsUUID, IsOptional, IsEnum, IsArray, ValidateNested }
from 'class-validator';
import { Type } from 'class-transformer';

export enum AiProvider {
  OPENAI = 'openai',
  OLLAMA = 'ollama',
}

export class ChatMessageDto {
  @IsEnum(['user', 'assistant', 'system'])
  role: string;

  @IsString()
  content: string;
}

export class GenerateResponseDto {
  @IsUUID()
  campaignId: string;

  @IsString()
  prompt: string;

  @IsOptional()
  @IsEnum(AiProvider)
  preferredProvider?: AiProvider;

  @IsOptional()
  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => ChatMessageDto)
  history?: ChatMessageDto[];
}

import { IsInt, Min, Max, IsString, IsOptional } from 'class-validator';

export class RollDiceDto {
  @IsInt()
  @Min(2)
  @Max(100)
  sides: number;

  @IsInt()

```

```
@IsOptional()
modifier: number = 0;
```

```
@IsString()
@IsOptional()
reason?: string;
}
```

```
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { BaseChatModel } from '@langchain/core/language_models/chat_models';
import { ChatOpenAI } from '@langchain/openai';
import { ChatOllama } from '@langchain/ollama';
```

```
@Injectable()
export class LlmProviderFactory {
  constructor(private configService: ConfigService) {}

  createModel(provider?: string): BaseChatModel {
    const activeProvider = provider || this.configService.get(
      'NX_AI_DEFAULT_PROVIDER');

    if (activeProvider === 'ollama') {
      return new ChatOllama({
        baseUrl: this.configService.get('NX_OLLAMA_BASE_URL'),
        model: this.configService.get('NX_OLLAMA_MODEL_NAME'),
        temperature: 0.7,
      });
    }

    return new ChatOpenAI({
      openAIApiKey: this.configService.get('NX_OPENAI_API_KEY'),
      modelName: this.configService.get('NX_OPENAI_MODEL_NAME')
    }, {
      model: 'gpt-4o',
      temperature: 0.7,
    });
  }
}
```

```
import { Injectable, Logger } from '@nestjs/common';
import { AgentExecutor, createOpenAIFunctionsAgent } from 'langchain/agents';
```

```

import { ChatPromptTemplate, MessagesPlaceholder } from
'@langchain/core/prompts';
import { SystemMessage, HumanMessage } from
'@langchain/core/messages';
import { LlmProviderFactory } from './llm-provider.factory';
import { VectorStoreService } from './vector-store/vector-store.service';
import { DiceTool } from './tools/dice.tool';
import { RulesSearchTool } from './tools/rules-search.tool';
import { GenerateResponseDto } from './dto/generate-response.dto';

@Injectable()
export class AiAgentService {
  private readonly logger = new Logger(AiAgentService.name);

  constructor(
    private llmFactory: LlmProviderFactory,
    private vectorStore: VectorStoreService,
    private diceTool: DiceTool,
    private rulesTool: RulesSearchTool,
  ) {}

  async generateGameResponse(dto: GenerateResponseDto):
  Promise<string> {
    const contextDocs = await this.vectorStore.similaritySearch(
      dto.prompt,
      dto.campaignId,
      3 // Top-K documents
    );

    const contextText = contextDocs.map(doc => doc.pageContent).join("\n--
\n");

    const llm = this.llmFactory.createModel(dto.preferredProvider);
    const tools = [this.diceTool, this.rulesTool];

    const prompt = ChatPromptTemplate.fromMessages([
      new SystemMessage(`Ти - Майстер гри Dungeons & Dragons (DM).
      Твоя мета - вести гру цікаво та дотримуватись правил.
      Використовуй наданий контекст історії для узгодженості.

      КОНТЕКСТ ІСТОРИЇ:
      ${contextText}
    `),
      new MessagesPlaceholder("chat_history"),
      new HumanMessage("{input}"),
      new MessagesPlaceholder("agent_scratchpad"),

```

```

]);

const agent = await createOpenAIFunctionsAgent({
  llm,
  tools,
  prompt,
});

const executor = new AgentExecutor({
  agent,
  tools,
  verbose: true,
});

const result = await executor.invoke({
  input: dto.prompt,
  chat_history: dto.history || [],
});

return result.output;
}
}

```

```

import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { JwtPayload } from '@auto-dnd/shared';
import { SessionsService } from '../sessions.service';
import { ConfigService } from '@nestjs/config';
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(
    private readonly sessionsService: SessionsService,
    private readonly configService: ConfigService
  ) {
    super({
      jwtFromRequest:
ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey:
configService.getOrThrow('NX_JWT_ACCESS_TOKEN_SECRET'),
    });
  }
  public async validate(payload: JwtPayload): Promise<JwtPayload> {

```

```

        const session = await
this.sessionsService.getSessionByUserIdAndTokenId(
    payload.sub,
    payload.jti
);

        if (!session || session.userId !== payload.sub || new
Date(session.expiredAt) <= new Date()) {
            throw new UnauthorizedException();
        }

        return payload;
    }
}

```

ДОДАТОК В
Слайди презентації

ІНТЕГРАЦІЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ТА ГЕНЕРАТИВНОЇ ГРАФІКИ ДЛЯ ВЕДЕННЯ РОЛЬОВИХ ІГОР «DUNGEONS & DRAGONS»

Дипломна кваліфікаційна робота магістра

Виконав: ст. гр. КНТ-114м Дробіняк М.В.

Керівник: к.т.н., доцент Скрупський С.Ю.

1

Рисунок В.1 - Слайд 1

Об'єкт та мета дослідження



Об'єкт дослідження

Процес інженерії програмного забезпечення на основі штучного інтелекту для підтримки та автоматизації підготовки й проведення настільних рольових ігор.



Предмет дослідження

Методи та програмні засоби інтеграції генеративних моделей для автоматизації функцій ведучого гри



Мета роботи

Підвищення ефективності діяльності ведучого шляхом зменшення часових витрат на підготовку та зниження когнітивного навантаження під час гри.

2

Рисунок В.2 - Слайд 2

Постановка завдання

Проблема:

Підготовка до якісної гри займає 2-6 годин. Ведучий стикається з піковим навантаженням, керуючи сюжетом, правилами та візуалізацією одночасно.

Завдання роботи:

- ✓ Провести аналіз предметної області та існуючих інструментів.
- ✓ Розробити підхід гібридної оркестрації LLM.
- ✓ Спроекувати мікросервісну архітектуру системи.
- ✓ Реалізувати механізм довготривалої пам'яті (RAG).
- ✓ Розробити програмне забезпечення та виконати тестування.

3

Рисунок В.3 - Слайд 3

Порівняння існуючих аналогів

Критерій	VTT (Roll20)	AI-квести	ChatGPT	Розроблена система
Генерація сюжету	✗ Ручна	✓ Авто	✓ Авто	✓ Авто
Дотримання правил	📖 Висока	✗ Низька	⚠ Помилки	✓ Гарантована (MCP)
Пам'ять	—	Коротка	Середня	📜 Довготривала (RAG)
Візуалізація	✗ Немає	Обмежена	Окрема вкладка	🖼 Інтегрована

4

Рисунок В.4 - Слайд 4

Запропонований підхід: Гібридна оркестрація

Чому не просто LLM?

Мовні моделі схильні до "галюцинацій" у математиці та забування фактів.

Рішення: Розділення відповідальності

LLM: Генерація намірів та художній опис.

</> Код (MCP): Валідація правил, кидки кубиків, стан світу.

RAG: Підвантаження релевантного контексту з векторної БД.



5

Рисунок В.5 - Слайд 5

Схема гібридної оркестрації

Система не покладається лише на LLM, а використовує агентний підхід для делегування задач спеціалізованим модулям.



6

Рисунок В.6 - Слайд 6

Обґрунтування вибору засобів оркестрації

Критерій	LangChain (Обрано)	LlamaIndex	Native API (OpenAI SDK)
Основний фокус	Оркестрація та агенти (ідеально для складних дій)	Індексація та пошук (Добре для RAG, гірше для дій)	Специфічні агенти (Тільки в межах екосистеми)
Абстракція (зміна моделі)	Легка заміна GPT на Llama	Середня (фокус на даних, а не на моделях)	Пропріетарна (обмежено моделями від OpenAI)
Робота із зовнішніми інструментами (MCP)	Розвинена (Готові обгортки для інструментів)	Обмежена (складніше писати агентів)	(Нативна, але тільки для моделей OpenAI)
Розширення	Open Source, тисячі готових плагінів	Data-Centric (Конектори для даних)	Відсутні (Тільки офіційні можливості)

7

Рисунок В.7 - Слайд 7

Обґрунтування вибору СКБД

Критерій	PostgreSQL+pgvector (Обрано)	MongoDB (Atlas Vector)	Pinecone+MySQL
Тип даних	Гібридний (Реляційні + Векторні)	Документні + Векторні (гнучкий JSON)	Розділені (Векторна + Реляційна)
ACID транзакції	Повна підтримка (надійність правил)	Обмежена (на рівні документа)	Відсутні (між різними системами)
Складність інфраструктури	Низька (1 контейнер)	Середня (складніше налаштовувати локально)	Висока (два окремих сервіси)
Векторний пошук	Інтегрований в SQL (HNSW індекс, наявні фільтри + пошук)	Окремий індекс (специфічний синтаксис)	Тільки через API (мережева затримка)

8

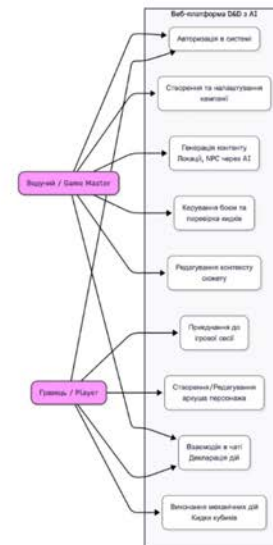
Рисунок В.8 - Слайд 8

Діаграма прецедентів

Взаємодія користувачів

Система підтримує дві основні ролі:

- **Ведучий (GM):** Керує кампанією, генерує сцени, валідує дії AI.
- **Гравець:** Взаємодіє з чатом, виконує кидки, керує персонажем.



9

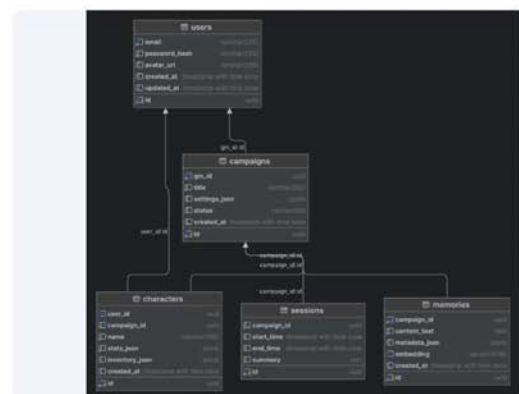
Рисунок В.9 - Слайд 9

Схема бази даних

Гібридна структура

Поеднання структурованих та неструктурованих даних:

- **Relational:** users, campaigns, sessions.
- **JSONB:** Гнучка структура Inventory та Stats.
- **Vector:** Таблиця memories з індексом HNSW для швидкого RAG-пошуку.



10

Рисунок В.10 - Слайд 10

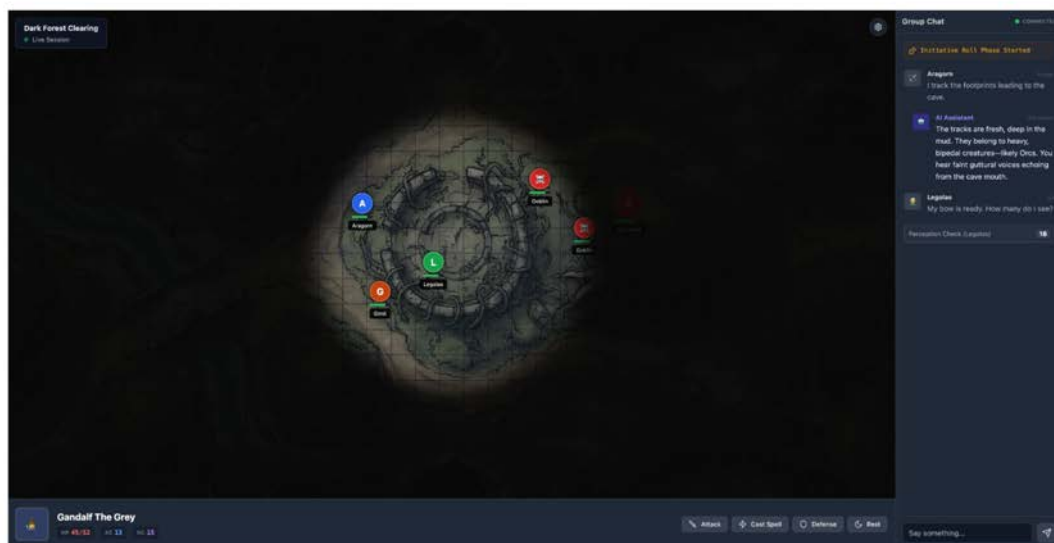
Интерфейс ведущего



11

Рисунок В.11 - Слайд 11

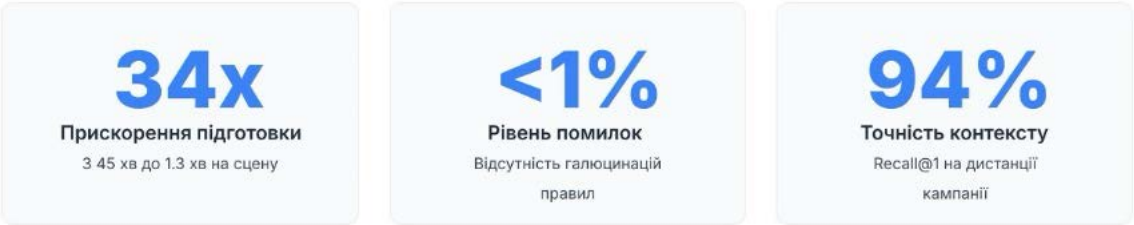
Интерфейс игрока



12

Рисунок В.12 - Слайд 12

Висновки та результати

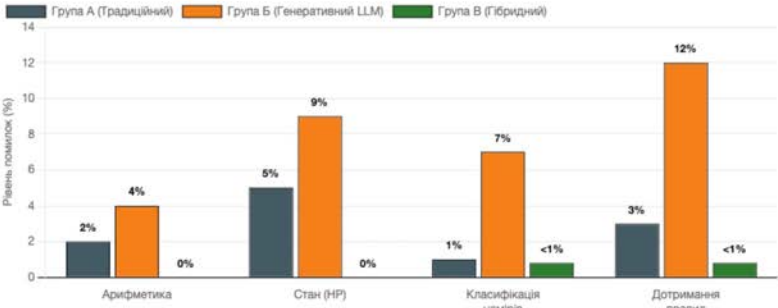


Створено повноцінну платформу для гібридного ведення настільних рольових ігор, що поєднує креативність AI та точність коду.

13

Рисунок В.13 - Слайд 13

Висновки та результати

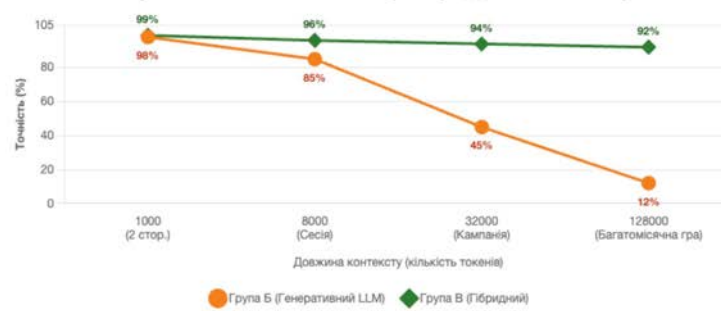


Порівняльний аналіз частоти помилок

14

Рисунок В.14 - Слайд 14

Висновки та результати

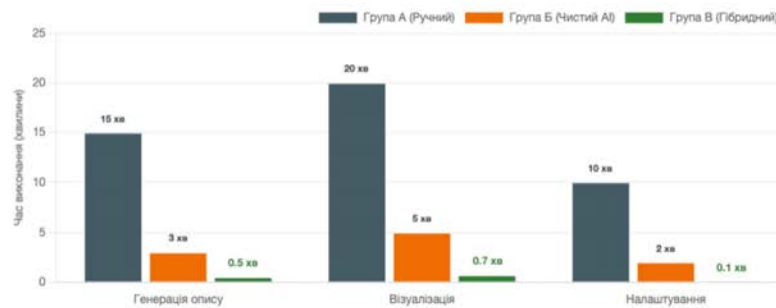


Залежність точності від обсягу контексту

15

Рисунок В.15 - Слайд 15

Висновки та результати

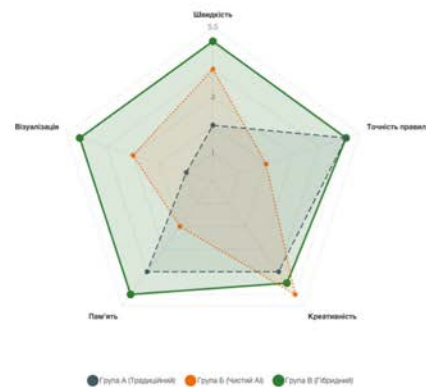


Порівняння часових витрат на підготовку сцени

16

Рисунок В.16 - Слайд 16

Висновки та результати



Комплексна оцінка ефективності

17

Рисунок В.17 - Слайд 17

ДЯКУЮ ЗА УВАГУ!

Доповідь завершено.

Готовий відповісти на ваші запитання.

18

Рисунок В.18 - Слайд 18