COMPUTER SYSTEMS AND NETWORKS

Vol. 7, No. 2, 2025

DOI: https://doi.org/10.23939/csn2025.02.242

УДК 004.8:794.8

ДОСЛІДЖЕННЯ ТА РОЗРОБКА ПРОГРАМНИХ ПОВЕДІНКОВИХ КОМПОНЕНТ В КОМП'ЮТЕРНИХ ІГРОВИХ СИСТЕМАХ

Б. І. Ширий, О. Л. Лашко

Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин
E-mail: bohdan.shyryi.mkisp.2025@lpnu.ua, oksana.l.lashko@lpnu.ua

Висвітлено результати дослідження можливостей побудови програмних поведінкових компонент на базі великих мовних моделей (LLM) та їх використання в ігрових системах. Предметом дослідження є поведінковий компонент неігрових персонажів (NPC) комп'ютерної гри, інтегрований з LLM. Метою пошуку є вивчення та аналіз особливостей застосування LLM для спрощення розробки NPC. Проаналізовано ігрові штучні інтелекти (AI) – реактивні, керовані станами та навчальні. Розглянуто взаємодію з NPC через LLM для чат-спілкування та досліджено аналоги поведінки й діалогу таких персонажів. Результатом є спроєктований та реалізований API сервіс для взаємодії з LLM та зовнішня ігрова система з відповідним програмним поведінковим компонентом, що надає функціонал взаємодії з LLM. Проведене тестування довело, що ця система працює з 54 унікальними контекстними властивостями, має максимальний час генерації системного промпта до 60 мс, максимальний час розбору та валідації відповіді LLM до 30 мс, а також підтримує моделі з мінімум 14 мільярдами параметрів.

Ключові слова: API, LLM, NPC, XML, контекстна дія, контекстна властивість, поведінковий програмний компонент.

Вступ

Оскільки предметом дослідження є програмний поведінковий компонент комп'ютерних ігрових систем, то варто розглянути, що таке ігрова індустрія. Ігрова індустрія створює, розповсюджує та просуває відеоігори на різних платформах та пристроях, що робить їх доступними для всіх вікових категорій і демографічних груп. Вона містить всі етапи розробки гри – від концептуального дизайну і програмування до маркетингу та продажу. Успішність ігрової індустрії базується на її здатності швидко адаптуватися до ринкових змін, використовуючи різноманітні бізнес-моделі та залучаючи велику й активну спільноту гравців. Масове охоплення ринку, гнучкі способи монетизації, інтеграція реклами, розвиток кіберспорту та постійне оновлення контенту – створюють стабільні джерела доходу і стимулюють подальше зростання індустрії. Саме поєднання цих факторів формує сучасний динамічний ландшафт ігрової сфери, роблячи її однією з найприбутковіших у світі розваг.

Розвиток такого аспекту гри, як взаємодія з неігровими персонажами, веде до збільшення зацікавленості у грі та, як наслідок, до збільшення прибутків. За таку взаємодію відповідають поведінкові компоненти персонажу, які охоплюють широкий спектр ситуацій – від простих реакцій на дії гравця до складних діалогових сценаріїв, побудованих на внутрішніх станах NPC або контексті ігрового світу. Поведінкові компоненти відіграють ключову роль у тому, щоб зробити цих персонажів не лише функціональними, а й цікавими, переконливими та логічно послідовними. Нижче наведено

методи взаємодії з NPC, які допомагають краще організувати їх поведінку та інтеграцію в загальну логіку гри.

NPC як фон. Часто NPC навіть не мають власних реплік і під час взаємодії або не реагують на гравця, або здійснюють базові дії, а саме: атакують противників, стежать за гравцем, виражають якусь емоцію тощо. Їх поведінка зазвичай обмежується заздалегідь прописаними скриптами, які не враховують контекст або попередні взаємодії.

NPC розповідачі. Часто NPC грають роль оповідачів, а саме: розказують гравцю історію про себе або просто розповідають репліки згідно з сюжетом. Що може зменшити рівень занурення в гру, оскільки ці NPC не відіграють повну роль персонажа, а їхні репліки кожного разу однакові, коли з ними взаємодіяти.

NPC з системами діалогу. З деякими NPC можна взаємодіяти за допомогою системи діалогів, коли гравець підходить до NPC та починає з ним взаємодіяти, то відкривається меню діалогу, у якому можна обрати доступні репліки.

LLM NPC. На цей час такий метод лише розвивається, базується на спілкуванні з персонажем, що використовує LLM для надання відповіді. Його суть полягає у створенні ілюзії живої взаємодії, де NPC може відповідати на довільні запити гравця.

Актуальність розробки програмного поведінкового компонента для керування NPC за допомогою LLM зумовлена стрімким розвитком технологій AI та висхідним попитом на більш реалістичні та динамічні ігрові світи. Гравці очікують від NPC не тільки базової поведінки, а й здатності вести осмислені діалоги, адаптації до дій користувача та можливість зробити свій внесок у розвиток сюжету.

У межах сучасних підходів до розробки поведінкових компонентів доцільно розглядати динамізацію NPC, наприклад шляхом впровадження змінного контексту та використання LLM для генерації варіативних реплік. LLM дають змогу значно підвищити рівень інтерактивності та природності взаємодії з NPC, надаючи їм контекстні властивості – параметри об'єктів ігрової сцени, що передаються моделі для генерації більш релевантної відповіді. Крім того, варто розглянути розширення можливостей NPC шляхом впровадження системи впливу на ігрову сцену, наприклад через контекстні дії, які LLM може повернути як реакції на взаємодію гравця.

Отже, застосування LLM дає змогу не лише розширити наративні можливості через генерацію варіативних реплік, а й надати NPC здатність впливати на події в грі. Попри потенційне зростання вартості проєкту під час реалізації діалогових систем, розумне використання LLM може суттєво знизити витрати коштом автоматизації реплік та зменшення потреби в локалізації. Отож LLM-орієнтовані поведінкові компоненти можуть стати основою для створення глибших ігор із багатшим досвідом для гравців.

Огляд літературних джерел

Поведінкові компоненти [1] є усталеною практикою в ігрових системах. Вони описують поведінку певного об'єкта та відтворюють її або передають керування іншій компоненті, що дає змогу забезпечити більш динамічну та реалістичну взаємодію в межах гри. Ці компоненти можуть описувати зовсім різну поведінку, від системи інвентарю до системи руху NPC, а також можуть включати логіку для взаємодії з навколишнім середовищем, іншими персонажами або виконання специфічних дій на основі умов гри. Вони часто є основою для програмування складніших систем штучного інтелекту в іграх.

У прикладному програмному забезпеченні поведінковий компонент [2–3] може визначати реакцію інтерфейсу або системи на дії користувача. Наприклад, можна створити компонент, який виявляє неактивність користувача і через певний час автоматично виконує вихід із системи. Інший приклад – компонент, який визначає, як система має реагувати на події типу "збереження документа" або "виникнення помилки".

Під ігровим AI [4–6] розуміють програмування NPC на прийняття рішень і взаємодії з ігровим світом. Від розробки ігрового AI буде повністю залежати розробка поведінкових компонентів, що означає необхідність дослідження різних типів ігрових AI. Однак варто зазначити, що це не чітке розмежування, а радше крайнощі спектра програмної розробки, де різні підходи можуть поєднуватися, залежно від потреб і можливостей гри. Далі наведено узагальнення основних типів ігрового AI, які застосовуються в сучасних ігрових системах, з огляду на їхні функціональні особливості та сферу використання.

Реактивний AI [7] – це найзагальніший тип AI, який не містить запам'ятовування попереднього досвіду. За такого типу AI NPC буде практично миттєво відгукуватися на дії гравця у реальному часі, без необхідності аналізувати певні дані чи вираховувати складні формули. Реакція NPC обмежується лише поточними діями гравця чи властивостями ігрової сцени, що робить таку поведінку надзвичайно прямолінійною та швидкою. Такий підхід дає змогу швидко і легко впроваджувати AI для простих взаємодій, але він обмежує глибину ігрового досвіду. Оскільки NPC не зберігає жодної інформації про попередні дії, кожна взаємодія стає однотипною і не додає елемента несподіванки або адаптації. Також таким підходом не вийде забезпечити динаміку ігрової сцени, що призводить до менш захопливого досвіду для гравця.

Керований станами AI [8]. Ігрові AI можуть керуватися станами, такими як "пошук", "переслідування", "напад" та інші, а також перемикатися між цими станами, орієнтуючись на певні умови. Наприклад, NPC може перебувати у стані "пошук", коли він активно досліджує оточення на наявність гравця, і, як тільки він знаходить гравця, автоматично переходить в стан "переслідування", намагаючись наблизитися до нього. Цей тип AI дає змогу розробникам створювати більш гнучкі та інтерактивні ігрові системи, де персонажі можуть адаптувати свою поведінку залежно від змін у середовищі чи дій гравця. Такий підхід дозволяє створювати складнішу та менш передбачувану поведінку, що значно покращує ігровий досвід, оскільки гравець може зіткнутися з різними варіантами розвитку подій залежно від його власних дій або умов навколишнього світу.

Керований поведінковими деревами AI [9–10]. Використання поведінкових дерев є структурованим способом моделювання поведінки NPC, поєднуючи переваги керованих станами систем та більш гнучких сценаріїв. Кожен вузол дерева представляє певну дію або умову, а дерево визначає послідовність перевірок і виконання дій. Наприклад, NPC може перевіряти наявність гравця, потім вирішувати, чи атакувати, переслідувати або втекти, структура дерева дає змогу легко змінювати поведінку без переписування всього коду. Завдяки цьому підходу можна створювати модульні, повторно використовувані та більш передбачувані, але досить адаптивні сценарії поведінки NPC. Поведінкові дерева добре підходять для середніх і складних ігрових персонажів, оскільки забезпечують баланс між простотою реактивного AI та гнучкістю станових систем, дозволяючи NPC відповідати на різні дії гравця та умови середовища у структурований і контрольований спосіб.

AI, що навчається [11]. На відміну від попередніх він використовує такі підходи, як машинне навчання, щоб адаптуватися і змінювати свою поведінку протягом часу чи в результаті змін середовища. Цей тип AI може вчитися на діях гравця, так змінюючи свою поведінку, запам'ятовуючи поведінку гравця, що створює досвід динамічних змін. NPC спостерігає за тим, як гравець часто використовує певну тактику, він може адаптувати свою стратегію, щоб уникати цієї тактики або створювати протидію. Наприклад, якщо гравець постійно нападає з певної позиції, NPC може змінити свою поведінку, переходячи в більш оборонний режим або навіть вивчаючи нові способи атаки. Це значно розширює можливості гри, оскільки NPC стають більш адаптивними та непередбачуваними. Вони можуть змінювати свою стратегію в реальному часі, що робить кожну взаємодію з ними унікальною. Тому створюється більш захопливий і динамічний ігровий досвід, оскільки гравець більше не може передбачити дії NPC, базуючись лише на попередньому досвіді. Такий підхід дає змогу інтегрувати глибші елементи AI в ігрові системи, що можуть враховувати не лише поточні обставини, а й довгострокові зміни в поведінці гравця.

У цьому дослідженні важливо розглянути вже наявні рішення та технології, які забезпечують інтерактивну взаємодію між користувачем та віртуальними персонажами з використанням AI. Аналіз функціональних аналогів дає змогу виявити сильні та слабкі сторони теперішніх підходів, визначити рівень їхньої адаптивності та технічної реалізації, а також обґрунтувати вибір концептуального рішення для реалізації власної системи спілкування з NPC на основі LLM. З огляду на класифікацію ігрового AI, описану раніше, наявні рішення можуть реалізовувати як прості реактивні або станові поведінки, так і складні адаптивні моделі, наближені до поведінки, керованої LLM.

Dialogue System for Unity [12]. Бібліотека для створення діалогових систем в Unity. Вона надає повний інструментарій для створення складних діалогів, зокрема умови, змінні, гілки вибору, тригери та інтеграцію з іншими системами. Бібліотека побудована на компонентній моделі Unity, що дає змогу прив'язувати діалоги до конкретних GameObject. Через компонент Dialogue System Trigger можна налаштувати запуск діалогу. Реакція на дії гравця є миттєвою, без затримки, оскільки немає звернення до зовнішнього API або LLM, що робить її ближчою до реактивного AI. Необхідно чітко прописувати всі діалоги вручну, зокрема репліки, варіанти відповідей та умови. Пов'язування поведінки NPC з діалогами потребує додаткової логіки, що ускладнює підтримку великих проєктів.

Inworld AI [13]. Платформа для створення інтелектуальних персонажів з використанням AI, що дає змогу NPC вести природні та контекстно адаптивні діалоги. Вона інтегрується з Unity через SDK, надаючи інструменти для підключення AI-персонажів, керування їхньою поведінкою та взаємодією з гравцем. Це рішення дає змогу реалізувати керований поведінковими деревами AI або елементи AI, що навчається, оскільки NPC можуть адаптувати свою поведінку до контексту діалогу. Одним з обмежень є те, що моделі персонажів враховують лише характер, поведінку та манеру спілкування, але не інформацію про ігрову сцену чи зовнішній вигляд NPC. Для врахування цих деталей необхідно додатково програмувати логіку в Unity.

Charisma.ai [14]. Інтерактивна платформа для створення сценаріїв живого діалогу з NPC, яка легко інтегрується в Unity. Вона дає змогу розробникам додавати емоційно насичені, нелінійні розмови до персонажів, використовуючи потужний інструмент створення діалогів із підтримкою AI. За допомогою офіційного Unity SDK Charisma.ai можна під'єднати до гри, керувати діалогами NPC в реальному часі, реагувати на вибір гравця, тригери та дії в ігровому світі. Це значно спрощує створення глибокої, адаптивної поведінки NPC без потреби в складному кодуванні логіки діалогу вручну.

Підсумовуючи огляд, бачимо, що ігрові AI є різні, та не обов'язково трапляються в чистому вигляді. На практиці вони часто поєднуються, формуючи гібридні системи. Аналіз наявних рішень підкреслює важливість створення системи діалогів, яка поєднує динамічне генеративне спілкування на основі LLM із можливістю передачі детальної інформації про ігрову сцену та контекст NPC. Водночас пріоритетом є підтримка локального хостингу LLM, що забезпечить роботу без залежності від зовнішніх сервісів і збереження конфіденційності даних. Такий підхід дасть змогу створити живі, контекстно обґрунтовані діалоги, гнучко інтегровані в ігрове середовище, забезпечуючи більш правдоподібну взаємодію між гравцем та NPC.

Постановка задачі

Метою дослідження є інтеграція в ігрову систему програмного поведінкового компонента, роботу якого буде підсилено LLM. Розробка передбачатиме використання контекстних властивостей і дій, що дасть змогу знизити витрати на створення діалогів і їх локалізацію, а також уникнути потреби у розробці або придбанні діалогових рушіїв та систем. Система буде спрямована на забезпечення максимальної кількості контекстних властивостей за мінімально можливої кількості параметрів моделі LLM, що відкриє потенційну можливість реалізації ігрової системи у форматі туманних технологій, забезпечуючи у такий спосіб гнучкість, масштабованість та ефективність у використанні обчислювальних ресурсів.

Поставлено задачу спроєктувати та реалізувати ігрову систему, у якій максимальна кількість унікальних контекстних властивостей NPC – 54, максимальний час генерації системного промпта до 60 мс, максимальний час розбору та валідації відповіді LLM до 30 мс. Система має коректно працювати з моделями LLM, у яких мінімум 14 мільярдів параметрів.

Проєкт спрямований на створення ігрової системи, яка поєднує програмні поведінкові компоненти з можливостями LLM для динамічної генерації діалогів і дій на основі контексту. Завдяки використанню контекстних властивостей і дій вдається зменшити залежність від традиційних діалогових рушіїв, підвищити ефективність локалізації та забезпечити гнучке реагування NPC на зміну середовища. Такий підхід відкриває шлях до реалізації продуктивної, масштабованої ігрової архітектури з можливістю використання туманних чи хмарних обчислень, що є актуальним напрямом розвитку сучасних ігрових технологій.

Основні розділи дослідження

1. Структурна схема

У межах дослідження було розроблено структурну схему, наведену на рис. 1.

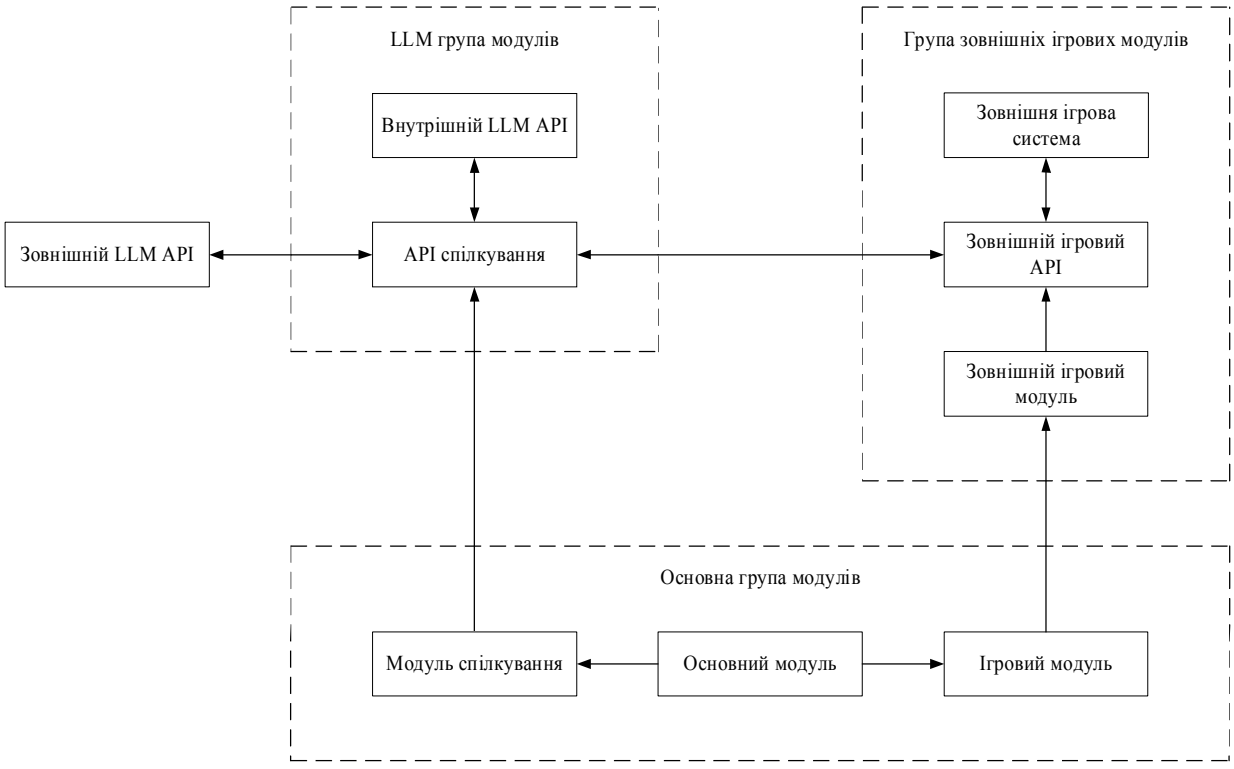

Рис. 1. Структурна сема системи

Основна група модулів – це “ядро” проєкту, яке відповідає за ключову логіку функціонування системи. До його завдань належать взаємодія з LLM, генерація контекстних промптів, інтеграція зі зовнішньою ігровою системою, обробка внутрішніх структур даних тощо.

LLM група модулів – забезпечує повноцінну підтримку роботи LLM, зокрема обробку запитів, інтерфейси для під’єднання до зовнішніх або локальних мовних моделей, а також внутрішні LLM API.

Група зовнішніх ігрових модулів – відповідає за візуалізацію та поведінку зовнішньої ігрової системи. Вона була спеціально реалізована в межах цього дослідження для тестування роботи поведінкового компонента з LLM, включаючи графічне відображення, симуляцію NPC та середовища.

2. Схема алгоритму генерування відповіді NPC

Загальна схема алгоритму складається з кількох частин. Вона відображає дії від моменту відправлення повідомлення гравця до отримання відповіді LLM.

Основний алгоритм (рис. 2) відтворюється у зовнішній ігровій системі після надсилання повідомлення у чаті з NPC. Спочатку форматується отримане повідомлення користувача та виводиться у чат. Згодом відбувається запит до зовнішнього ігрового API для отримання відповіді NPC. Отримавши відповідь запиту, відбувається вивід повідомлення NPC в чат і виконуються контекстні дії, якщо такі були згенеровані LLM. З основного алгоритму викликається виконання запиту ігрового API (рис. 2).

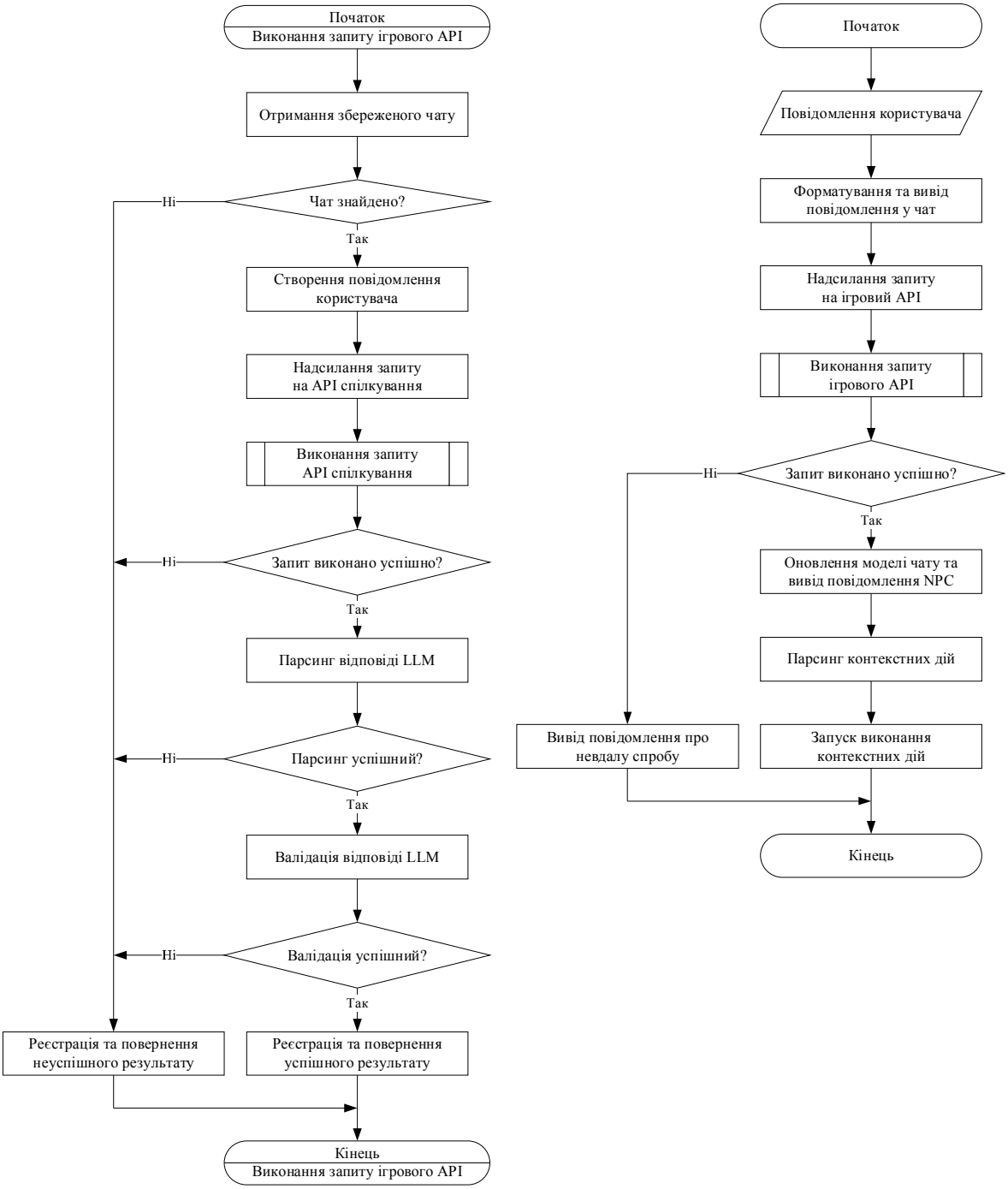

Рис. 2. Основний алгоритм та алгоритм виконання запиту ігрового API

Спершу відбувається отримання збереженого чату, який був створений в ігровій системі, окремим запитом до цього API. Далі, маючи повідомлення користувача, відсилається запит до API спілкування. Отож після отримання результату запиту відбувається парсинг повідомлення LLM, його валідація та повертається результат виконання поточного запиту.

Виконання запиту API спілкування наведене на рис. 3. Викликається цей запит зі зовнішнього ігрового API. Запит забезпечує отримання відповіді від LLM, використовуючи заданий провайдер та модель. Інструкції, описані в промпті, що відповідають за створення правильної відповіді LLM, будуть наведені далі у вигляді UML-діаграми.

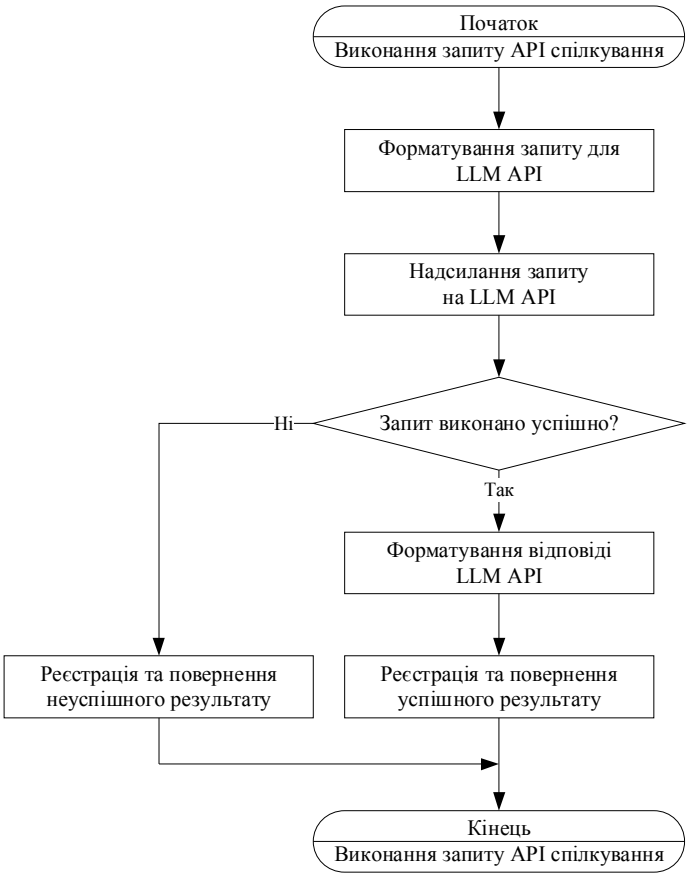

Рис. 3. Алгоритм виконання запиту API спілкування

Отже, наведений алгоритм забезпечує послідовну взаємодію між гравцем і LLM через зовнішні API, зберігаючи контекст діалогу та обробляючи відповіді згідно з внутрішньою логікою ігрової системи. Такий підхід дає змогу створити адаптивну систему спілкування з NPC, у якій LLM генерує не лише текстові репліки, а й потенційні дії в межах ігрової системи.

3. UML-діаграма виконання інструкцій LLM

Інструкції LLM є базовими правилами, яких модель має дотримуватися. Найголовніші правила є XML елементами, які називаються MainRule. Для покращення якості відповідей було додано XML елементи AssistantAnswerRule, які визначають правила, за якими потрібно формувати відповідь. Також наявні XML елементи AssistantAnswerToneRule, щоб контролювати манеру спілкування асистента з користувачем залежно від контексту та особистості NPC. XML елемент під назвою AssistantReferenceRule пояснює моделі, звідки потрібно брати відповідь на питання гравця і чи потрібно їх надавати. Контекстні властивості є в XML елементах як World та AssistantCharacter, про які модель дізнається з AssistantReferenceRules. Щоб ознайомити асистента з форматом повідомлень користувача,

було додано XML елементи UserMessageRule. Для зручності моделі було додано XML атрибут Priority, щоб уникнути зосередження LLM на неважливих елементах. Також були додані різноманітні атрибути типу: IsRequired, CanBeEmpty, ContainsText, ContainsElements, DescribedElement, AppliesToElement тощо, щоб коротко пояснити моделі особливості правила.

UML-діаграма виконання інструкцій LLM наведена на рис. 4. Ця UML-діаграма описує очікувану поведінку LLM, коли вона виконує запит користувача. Щоб LLM розуміла, як потрібно генерувати відповідь, модель має вищеописані інструкції у системному промпті. Отож наведена UML-діаграма відображає вимоги вище описаної XML-структури інструкцій від асистента.

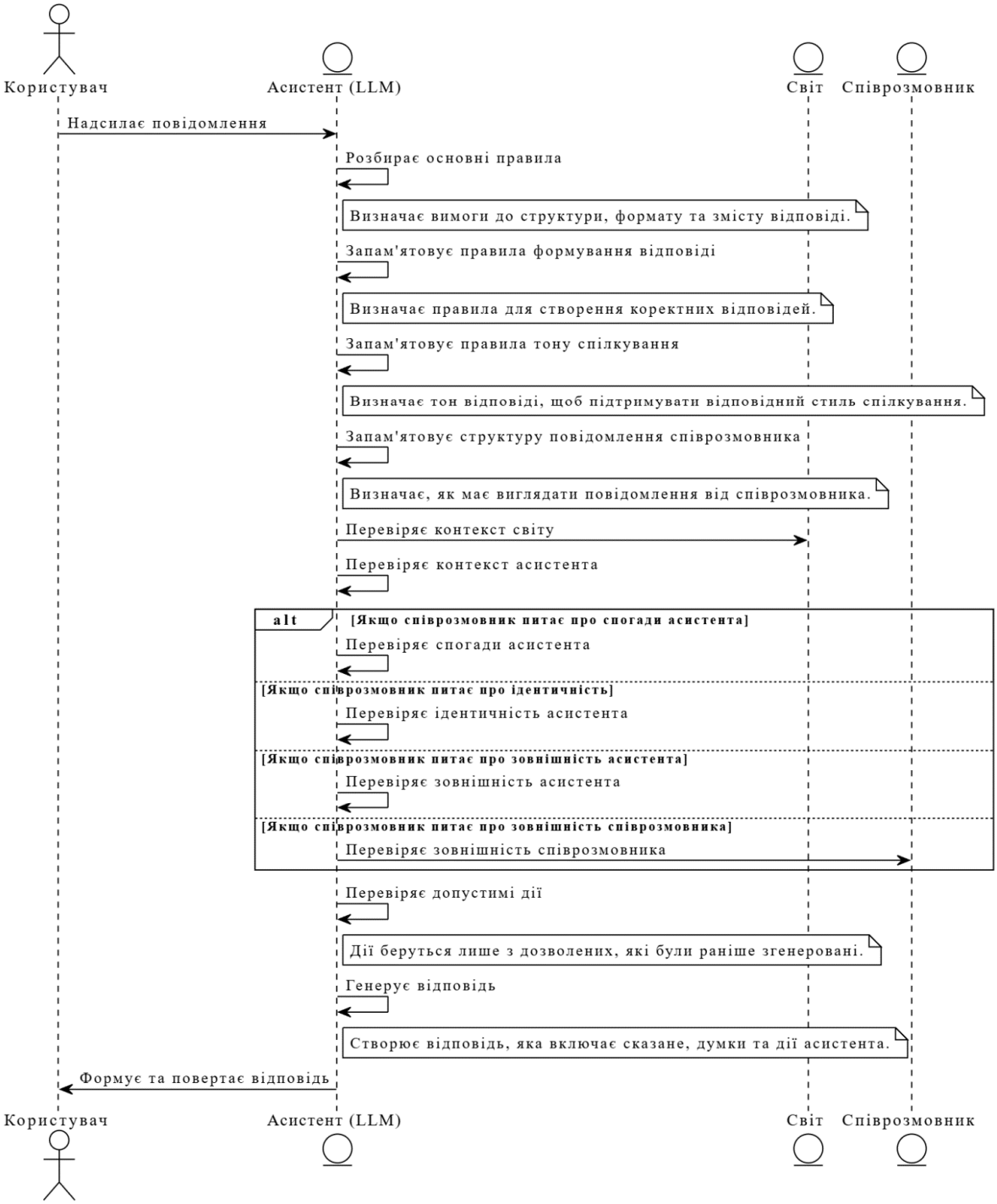

Рис. 4. UML-діаграма виконання інструкцій LLM

Запровадження XML-структури інструкцій виконує роль своєрідного фільтра, який допомагає LLM зосередитися лише на релевантних аспектах запиту. Завдяки атрибутам на зразок Priority IsRequired чи AppliesToElement модель не витрачає ресурс на обробку другорядної інформації, що значно зменшує обсяг необхідного контексту. Такий підхід дає змогу мінімізувати кількість повторюваних пояснень у промптах і так збільшує ефективність генерації відповіді. Крім того, використання спеціалізованих правил на зразок AssistantAnswerToneRule та AssistantReferenceRule оптимізує контекст завдяки чіткому розподілу його складових: окремо задаються джерела знань, стиль комунікації та спосіб формування відповіді. Це дає можливість будувати більш компактні та керовані системні інструкції, які не перевантажують LLM зайвими деталями. У підсумку модель здатна швидше орієнтуватися в отриманих даних і генерувати більш точні та відповідні до ситуації відповіді.

Несправність XML-структури у відповідях LLM можна розглядати як непрямий механізм валідації контексту. Якщо модель перестає дотримуватися заданої XML-схеми, це свідчить про вихід за межі визначених інструкцій або про перевантаження контексту. Отож збереження структурної цілісності XML є критерієм коректності генерації: будь-яке відхилення сигналізує про потенційну втрату релевантності відповіді. У разі виявлення такої невідповідності система не зупиняється на помилковому результаті, а застосовує механізм повторної генерації відповіді. Передбачено до трьох спроб перегенерації, що дає змогу компенсувати випадкові збої моделі, спричинені перевантаженням чи тимчасовою втратою узгодженості контексту. Такий підхід забезпечує підвищену надійність роботи асистента, адже навіть за наявності технічних або структурних відхилень зберігається висока ймовірність отримання коректної та релевантної відповіді.

4. Характеристики пристрою

Проєкт було реалізовано з урахуванням гнучкості у виборі обчислювальних ресурсів: він підтримує використання туманних технологій, тобто локального розгортання моделей за допомогою Ollama, а також хмарних рішень через OpenAI API. У межах цього дослідження основна увага приділяється туманному підходу, де для експериментів використано локальну модель phi4.

Використання локальної моделі накладає певні вимоги до апаратної частини пристрою, адже саме його характеристики безпосередньо впливають на продуктивність і якість отриманих результатів. Тому перед поданням експериментальних даних доцільно навести конфігурацію обладнання, яка наведена в табл. 1.

Таблиця 1

Характеристики пристрою

Параметр	Значення
Версія ОС	Microsoft Windows 11 Pro
Вбудована відеокарта	Intel(R) UHD Graphics 2GB
Дискретна відеокарта	NVIDIA GeForce RTX 3050 Laptop GPU 4GB
Назва процесора	12th Gen Intel(R) Core(TM) i5-12450H
Тип процесора	HexaCore Intel Core i5-12450H, 2466 MHz
Оперативна пам’ять (1)	Samsung 8GB DDR4-3200 SODIMM DRAM
Оперативна пам’ять (2)	Samsung 8GB DDR4-3200 SODIMM DRAM
Накопичувач	SSD 512 ГБ, QLC 3D NAND, PCIe 3.0 x4
Мережевий адаптер	Intel(R) Wi-Fi 6 AX201 160MHz.
Тип інтерфейсу	802.11 Wireless Ethernet
Материнська плата	Lenovo IdeaPad Gaming 3 16IAH7

Результати дослідження

З метою перевірки відповідей LLM у різних сценаріях та оцінки її вразливості до "перегрівання" розроблено набір тестів, які містять як базове, так і стрес-тестування. Тести охоплюють формування чат-повідомлень, отримання результатів запитів від зовнішніх або локальних LLM-провайдерів, а також розбір і валідацію відповідей. Набір тестів поділено на декілька основних груп, які наведені у табл. 2. Виконання цих тестів наведене на рис. 5.

Таблиця 2

Перелік unit тестів

Предмет тестування	Кількість
Привітання з NPC-асистентом	10
Погроза NPC-асистенту	8
Питання, що змушують LLM вийти за межі ролі NPC	8
Питання про NPC-асистента	19
Питання про NPC-асистента (інші контекстні властивості)	19
Питання про співрозмовника	18
Питання про співрозмовника (інші контекстні властивості)	18
Питання не по контексту до NPC-асистента	9
Ствердження для NPC-асистента	7
Перевірка спогадів NPC-асистента	21
Питання з неповними словами до NPC-асистента	13
Надсилання нерозбірливого тексту	11
Всього	**161**

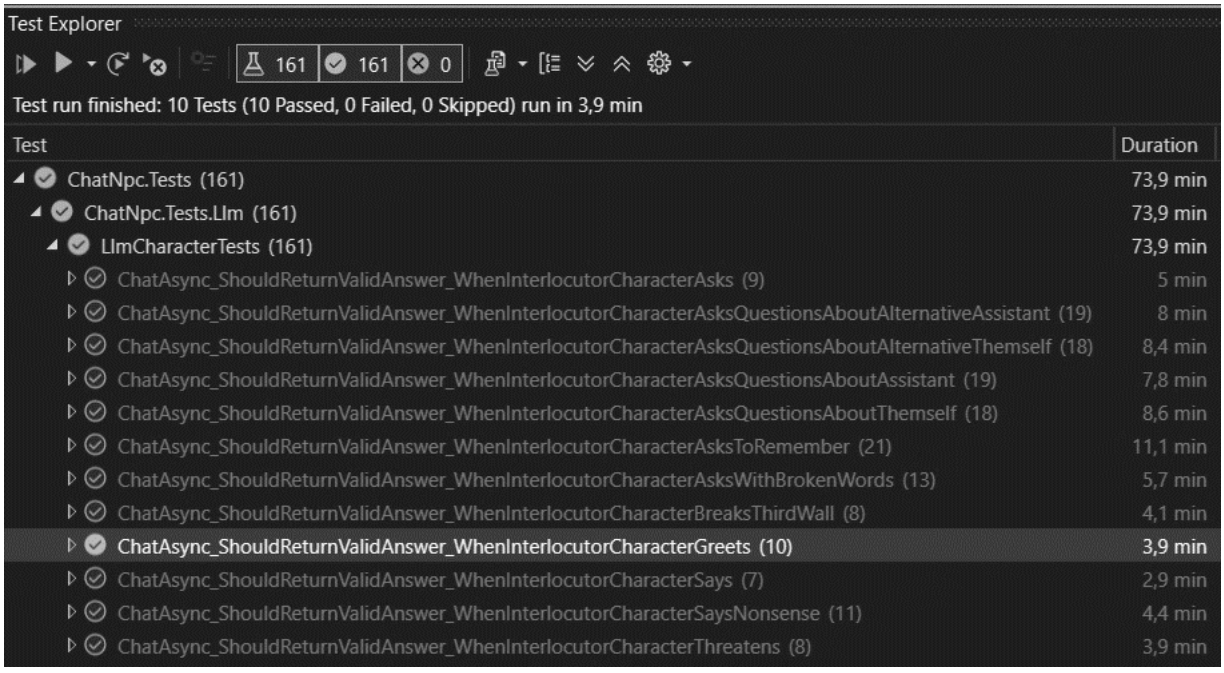

Рис. 5. Успішне виконання тестів

Як бачимо з Test Explorer, виконання всіх тестів було успішним. Предмет тестування можна визначити за назвою тесту. Досягнуто врахування контекстних властивостей LLM NPC. Наприклад, як наведено на рис. 6, LLM відповіла згідно із зовнішністю та особистістю NPC, що вона представляла. Іншим прикладом може бути відповідь LLM згідно зі спогадами NPC, що наведено на рис. 7. Також було досягнуто відтворення контекстних дій LLM NPC, як наведено на рис. 8.

Досягнуті та виміряні числові параметри, а саме: затримки – час формування системного промпта; час розбору та валідації відповіді моделі. Результати наведені на рис. 9 та 10, які відображають консоль API зовнішньої ігрової системи.

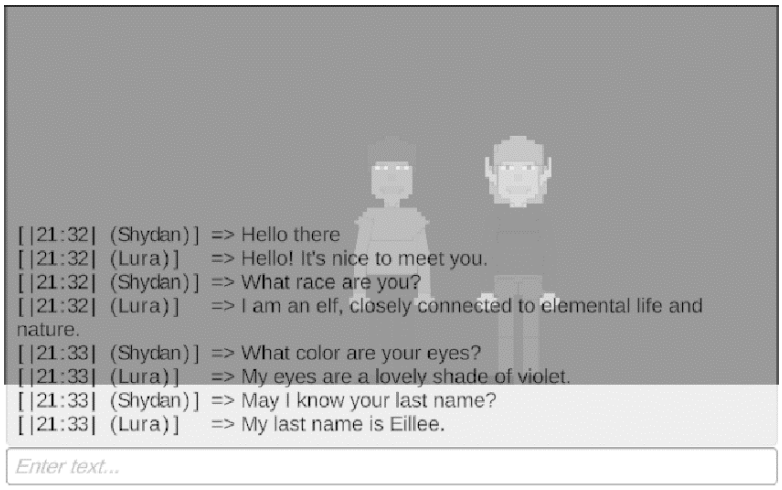

Рис. 6. Перевірка контексту зовнішності та особистості NPC

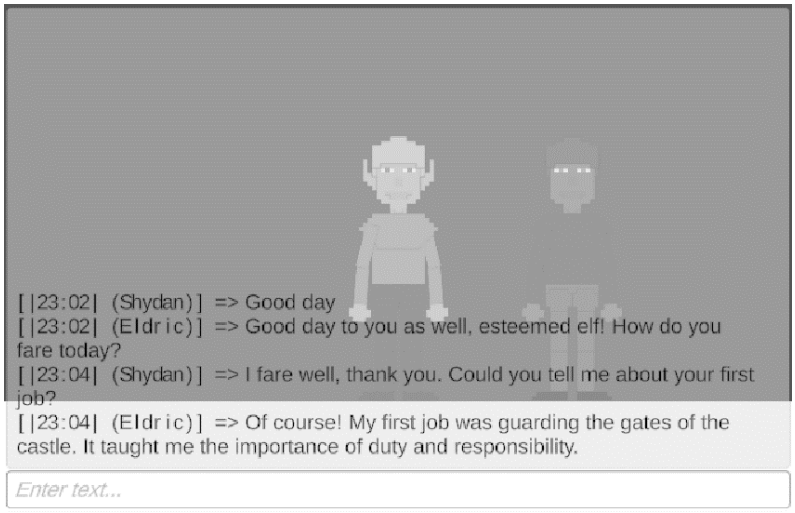

Рис. 7. Перевірка контексту пам'яті NPC

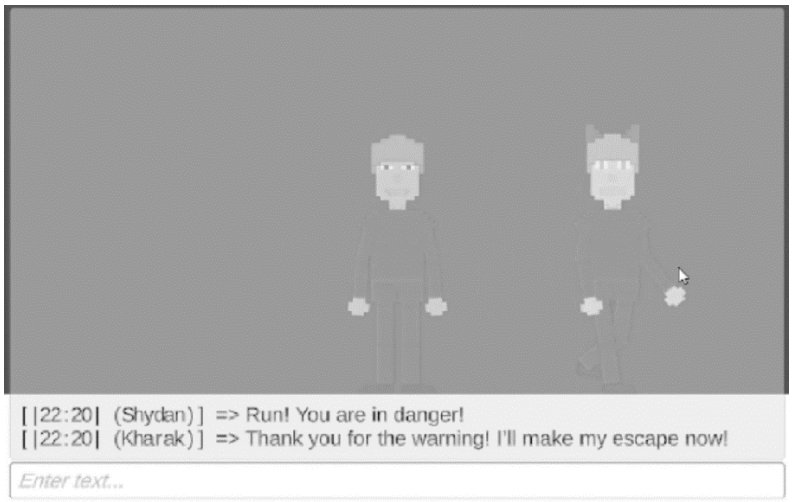

Рис. 8. Перевірка контекстних дій NPC

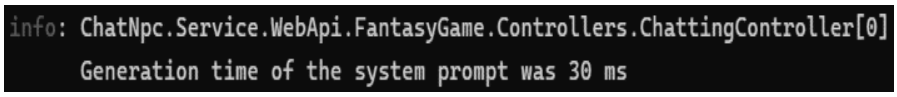

Рис. 9. Логування часу генерації системного промпта

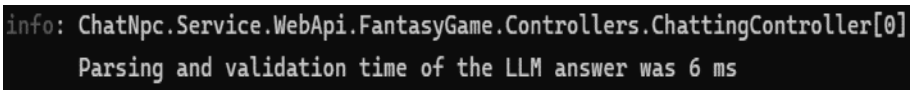

Рис. 10. Логування часу розбору та валідації відповіді моделі

Висновки

Проведені дослідження виявили особливості та можливості застосування LLM для спрощення розробки NPC. Це дало змогу реалізувати програмний поведінковий компонент, що інтегрується з LLM через API сервіс, та зовнішню ігрову систему для демонстрації результату роботи. Для генерування відповіді був розроблений процес створення динамічного промпта, що враховує параметри ігрової сцени та NPC.

Для доведення дієвості створеного засобу проведено автоматизоване unit-тестування, що покривало як стандартні, так і критичні сценарії для LLM. Крім цього, виконувалося ручне тестування, яке допомогло оцінити поведінку системи у реальних умовах. Отримані результати доводять, що LLM із 14 мільярдами параметрів показує стабільну та коректну роботу. Під час тестування вручну були також досліджені затримки системи. Зокрема, час створення системного промпта становив 30 мс, а розбір та валідація відповіді – 6 мс.

Отже, спроєктований програмний поведінковий компонент інтегрує можливості LLM у поведінкову логіку NPC, забезпечуючи глибоку контекстну взаємодію з гравцем. Запропоноване рішення підтвердило свою технічну життєздатність та відкрило нові напрями для подальших досліджень, зокрема в контексті масштабування, автоматизації генерації контенту та використання туманних і хмарних обчислень. Усе це свідчить про значний потенціал застосування подібних компонентів у майбутніх ігрових проєктах.

Список літератури

1. Martin Černý, Tomáš Plch, Matěj Marko, Jakub Gemrot, Petr Ondráček, Cyril Brom, "Using Behavior Objects to Manage Complexity in Virtual Worlds", Cornell University, Aug 3, 2015. DOI: 10.48550/arXiv.1508.00377. URL: https://arxiv.org/abs/1508.00377. Accessed: May 2025.

2. Khalifa H. B., Khayati O., Ghezala H. H. B. "A Behavioral and Structural Components Retrieval Technique for Software Reuse," 2008 Advanced Software Engineering and Its Applications, Hainan, China, 2008, pp. 134–137. DOI: 10.1109/ASEA.2008.45. URL: https://ieeexplore.ieee.org/document/4721328. Accessed: May 2025.

3. Manchana, Ramakrishna. (2019). Behavioral Design Patterns: Enhancing Software Interaction and Communication. International Journal of Science Engineering and Technology. 7. 1–18. DOI: 10.61463/ijset.vol.7.issue6.243. URL: https://www.researchgate.net/publication/384318655. Accessed: May 2025.

4. Zhou, Jiansong. (2024). Intelligent Agent and NPC Behavior Modeling: From Traditional Methods to AI Driven Interactive Game Design. Applied and Computational Engineering. 112. 85–91. DOI: 10.54254/2755-2721/2024.17913. URL: https://www.researchgate.net/publication/386513039. Accessed: May 2025.

5. Zeng, G. (2023). A review of AI-based game NPCs research. Applied and Computational Engineering, 15, 155–159. DOI: 10.54254/2755-2721/15/20230827. URL: https://www.ewadirect.com/proceedings/ace/article/view/4569. Accessed: May 2025.

6. Sindhu R. M., Annabel L. S. P., Monisha G. "Development of a 2D Game using Artificial Intelligence in Unity," 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2022, pp. 1031–1037. DOI: 10.1109/ICOEI53556.2022.9776750. URL: https://ieeexplore.ieee.org/document/9776750. Accessed: May 2025.

7. Gao, T., Mi, Q. (2020). Enemy Attack Management Algorithm for Action Role-Playing Games. In: Barolli, L., Hellinckx, P., Enokido, T. (eds) Advances on Broad-Band Wireless Computing, Communication and Applications. BWCCA 2019. Lecture Notes in Networks and Systems, vol. 97. Springer, Cham. DOI: 10.1007/978-3-030-33506-9_28. URL: https://link.springer.com/chapter/10.1007/978-3-030-33506-9_28. Accessed: May 2025.

8. Ganapathi A., Sivakumar P., Elango A., Gupta H., Panda N. "Exploring NPC Behaviors in Games through Finite Automata," 2024 5th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2024, pp. 1–8. DOI: 10.1109/GCAT62922.2024.10923923. URL: https://ieeexplore.ieee.org/document/10923923. Accessed: May 2025.

9. Marcotte, R., Hamilton, H.J. Behavior Trees for Modelling Artificial Intelligence in Games: A Tutorial. Comput Game J 6, 171–184 (2017). DOI: 10.1007/s40869-017-0040-9. URL: https://link.springer.com/article/10.1007/s40869-017-0040-9. Accessed: September 2025.

10. Sekhavat, Yoones. (2017). Behavior Trees for Computer Games. International Journal on Artificial Intelligence Tools. 26. DOI: 10.1142/S0218213017300010. URL: https://www.worldscientific.com/doi/abs/10.1142/S0218213017300010. Accessed: September 2025.

11. Meshram R., Krishna S., Kulkarni O., Patil R. R., Kaur G. and Maheshwari S. "NPC Behavior in Games Using Unity ML-Agents: A Reinforcement Learning Approach," 2025 International Conference on Automation and Computation (AUTOCOM), Dehradun, India, 2025, pp. 1519–1523. DOI: 10.1109/AUTOCOM64127.2025.10956320. URL: https://ieeexplore.ieee.org/document/10956320. Accessed: May 2025.

12. "Dialogue System for Unity", Pixel Crushers. URL: https://www.pixelcrushers.com/dialogue_system/manual2x/html. Accessed: May 2025.

13. "Docs – inworld.ai", inworld. URL: https://docs.inworld.ai/docs. Accessed: May 2025.

14. "Unity Plug 'n' Play SDK Docs", charisma.ai. URL: https://docs.charisma.ai/plug-n-play/unity. Accessed: May 2025.

RESEARCH AND DEVELOPMENT OF SOFTWARE BEHAVIORAL COMPONENTS IN COMPUTER GAMING SYSTEMS

B. I. Shyryi, O. L. Lashko

Lviv Polytechnic National University,
Department of Electronic Computing Machines

The article presents the results of a study on the possibilities of building software behavioral components based on large language models (LLMs) and their use in gaming systems. The subject of the research is the behavioral component of non-player characters (NPCs) in computer games, integrated with an LLM. The aim is to explore and analyze the features of applying LLMs to simplify NPC development. The study examines game AI types – reactive, state-driven, and learning-based. Interaction with NPCs through LLMs for chat communication is considered, and analogues of such characters' behavior and dialogue are explored. The outcome is a designed and implemented API service for interaction with LLMs and an external gaming system with a corresponding software behavioral component that provides functionality for LLM-based interaction. Testing has shown that this system works with 54 unique contextual properties, achieves a maximum system prompt generation time of up to 60 ms, a maximum LLM response parsing and validation time of up to 30 ms, and supports models with at least 14 billion parameters.

Keywords: API, LLM, NPC, XML, contextual action, contextual property, software behavioral component.