НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет/(ННІ) **Факультет інформаційних технологій .**

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету (Директор ННІ)	**Завідувач кафедри**
інформаційних технологій .	комп'ютерних наук .
(назва факультету (ННІ))	(назва кафедри)
Ігор БОЛБОТ .	Белла ГОЛУБ
(підпис) (ім'я ПРІЗВИЩЕ)	(підпис) (ім'я ПРІЗВИЩЕ)
"___" _____ 20_ р.	"___" _____ 20_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Експертна система обліку швидкодії програмного забезпечення

Спеціальність
121 інженерія програмного забезпечення
(код і найменування)

Освітня програма
Програмне забезпечення інформаційнх систем
(назва)

Орієнтація освітньої програми
освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
к. ф.-м. н., доцент _____ Віктор КИРИЧЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
к.т.н., доцент _____ Ірина БОРОДКІНА
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ Олександр ШКОЛЬНИЙ
(підпис) (ім'я ПРІЗВИЩЕ здобувача)

КИЇВ – 20___

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) __інформаційних технологій_____

ЗАТВЕРДЖУЮ
Завідувач кафедри__комп'ютерних наук_
___доцент, к.т.н._____ _____ _Белла ГОЛУБ_
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"____" _____20_____року

З А В Д А Н Н Я
ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ
_____Школьному Олександру Анатолійовичу_____
(прізвище, ім'я, по батькові)

Спеціальність 121 «Інженерія програмного забезпечення»
(код і назва)

Освітня програма ____Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми_____освітньо-професійна_____

Тема магістерської кваліфікаційної роботи Експертна система обліку швидкодії програмного забезпечення.
затверджена наказом ректора НУБіП України від "_01_" _листопада____2024р. №1963 «С»
Термін подання завершеної роботи на кафедру __29.11.2024_____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: Метрики продуктивності програмного забезпечення на базі Unreal Engine 5 (FPS, час рендерингу, використання пам'яті, навантаження на CPU/GPU), дані профілювання з Unreal Insights (трейси, статистика кадрів, таймінги), логи та телеметрія з UE5 проектів.

Перелік питань, що підлягають дослідженню:

1. Аналіз існуючих систем профілювання та моніторингу продуктивності для проектів на Unreal Engine 5.
2. Дослідження методів збору, агрегації та обробки метрик продуктивності з Unreal Insights та інших джерел профілювання, включаючи розробку pipeline для автоматизованого збору даних під час розробки та тестування.
3. Вибір і обґрунтування методів машинного навчання та експертного аналізу для виявлення проблем продуктивності, bottlenecks, та формування рекомендацій щодо оптимізації коду та ресурсів.
4. Розробка архітектури експертної системи з модулями діагностики, аналізу паттернів продуктивності та генерації рекомендацій для розробників щодо покращення швидкодії їхніх UE5 проектів.
5. Проектування інтерфейсу візуалізації метрик та дашбордів, що забезпечують ефективне відображення критичних показників продуктивності.

Дата видачі завдання "__01__"__листопада_____2024_ р.
Керівник магістерської кваліфікаційної роботи _____ ___Ірина БОРОДКІНА____
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ _____Олександр ШКОЛЬНИЙ____
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс програмування додатків)

CPU – Central Processing Unit (центральний процесор)

FPS – Frames Per Second (кадрів за секунду)

GPU – Graphics Processing Unit (графічний процесор)

GUI – Graphical User Interface (графічний інтерфейс користувача)

LOD – Level of Detail (рівень деталізації)

OLAP – Online Analytical Processing (оперативна аналітична обробка)

RAM – Random Access Memory (оперативна пам’ять)

SDK – Software Development Kit (комплект розробки програмного забезпечення)

UE5 – Unreal Engine 5

UMG – Unreal Motion Graphics (система інтерфейсів в Unreal Engine)

VRAM – Video Random Access Memory (відеопам’ять)

ЕС – експертна система

ПЗ – програмне забезпечення

ВСТУП

Актуальність теми. В сучасному світі розробки програмного забезпечення, особливо в галузі створення ігор, симуляторів та додатків віртуальної реальності, критично важливою є продуктивність та оптимізація. Unreal Engine 5, як один з найпотужніших ігрових двигунів, надає розробникам безпрецедентні можливості для створення високоякісного контенту, але водночас вимагає ретельного контролю за використанням системних ресурсів.

Проблема обліку та аналізу швидкодії програмного забезпечення стає дедалі актуальнішою через зростання складності проектів, підвищення вимог до візуальної якості та необхідність підтримки широкого спектру апаратних конфігурацій. Існуючі інструменти профілювання часто надають лише сирі дані без глибокого аналізу та конкретних рекомендацій щодо оптимізації.

Розробка експертної системи, яка не тільки збирає метрики продуктивності, але й аналізує їх, виявляє проблемні місця та надає обґрунтовані рекомендації щодо оптимізації, є актуальним завданням, що може суттєво підвищити ефективність процесу розробки та якість кінцевого продукту.

Об’єкт дослідження – процеси аналізу та обліку швидкодії програмного забезпечення в реальному часі.

Предмет дослідження – методи та засоби побудови експертної системи для автоматизованого обліку та аналізу швидкодії програмного забезпечення на базі Unreal Engine 5.

Мета дослідження – підвищення ефективності процесів оптимізації програмного забезпечення шляхом розробки експертної системи, що забезпечує автоматизований облік, аналіз та надання рекомендацій щодо покращення швидкодії на основі технологій Unreal Engine 5.

Завдання дослідження:

1. Провести системний аналіз предметної області обліку швидкодії програмного забезпечення та існуючих рішень у цій сфері.

2. Розробити концептуальну модель експертної системи обліку швидкодії для середовища Unreal Engine 5.
3. Спроектувати архітектуру системи з урахуванням особливостей двигунаUE5 та вимог до реального часу.
4. Реалізувати модулі збору метрик продуктивності для CPU, GPU та пам’яті.
5. Розробити базу знань та механізм логічного виведення для формування експертних рекомендацій.
6. Створити інтуїтивний інтерфейс користувача для візуалізації результатів аналізу.
7. Провести тестування системи на реальних проектах та оцінити її ефективність.

Методи дослідження: системний аналіз для вивчення предметної області; об’єктно-орієнтоване моделювання для проектування архітектури системи; методи експертних систем для побудови механізму логічного виведення; методи профілювання продуктивності для збору метрик; статистичний аналіз для обробки та інтерпретації даних.

Наукова новизна одержаних результатів:

- вперше розроблено архітектуру експертної системи обліку швидкодії, повністю інтегровану в середовище Unreal Engine 5, що дозволяє проводити аналіз в реальному часі без суттєвого впливу на продуктивність;
- запропоновано удосконалений алгоритм аналізу метрик продуктивності, який враховує специфіку роботи двигуна UE5 та використовує механізм логічного виведення для ідентифікації проблем;
- розроблено методику автоматизованого формування рекомендацій щодо оптимізації на основі бази знань експертної системи, що адаптується до конкретного типу проекту.

Практичне значення одержаних результатів. Розроблена експертна система може бути впроваджена в процес розробки різноманітних проектів на базі Unreal Engine 5, включаючи ігри, архітектурну візуалізацію, симулятори та додатки віртуальної реальності. Система дозволяє суттєво скоротити час на виявлення та усунення проблем з продуктивністю, підвищити якість кінцевого продукту та знизити вимоги до апаратного забезпечення кінцевих користувачів.

Апробація результатів дослідження. Основні положення та результати магістерської роботи були представлені на XIII Міжнародній науково-практичній конференції “Інформаційні технології в освіті та науці” (Київ, НУБіП України, травень 2025 р.) та опубліковані в збірнику тез конференції.

Структура роботи. Магістерська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел із 68 найменувань та 4 додатків. Загальний обсяг роботи становить 148 сторінок, з яких основний текст – 135 сторінок. Робота містить 42 рисунки та 18 таблиць.

У першому розділі проведено системний аналіз предметної області, розглянуто проблематику обліку швидкодії, проаналізовано існуючі рішення та сформульовано постановку завдання.

У другому розділі представлено моделювання експертної системи [8, 10], включаючи концептуальну модель, функціональне та об’єктно-орієнтоване моделювання, а також проектування бази знань.

У третьому розділі описано процес розробки системи в середовищі Unreal Engine 5, архітектуру компонентів, реалізацію модулів збору метрик та механізму логічного виведення.

У четвертому розділі наведено результати тестування системи, аналіз ефективності та економічне обґрунтування впровадження.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОБЛІКУ ШВИДКОДІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Аналіз проблематики вимірювання та обліку швидкодії програмного забезпечення

Швидкодія програмного забезпечення є критичним фактором успіху будь-якого сучасного додатку, особливо в контексті інтерактивних систем реального часу, таких як ігри, симулятори та додатки віртуальної реальності. Проблематика вимірювання та обліку швидкодії охоплює широкий спектр технічних та методологічних викликів, які потребують комплексного підходу до їх вирішення.

Основними аспектами проблематики є:

1. Багатовимірність метрик продуктивності. Швидкодія програмного забезпечення не може бути охарактеризована єдиним показником. Необхідно враховувати множину параметрів: - Частоту кадрів (FPS) та її стабільність - Час відгуку на дії користувача (input latency) - Використання процесора (CPU utilization) - Навантаження на графічний процесор (GPU load) - Споживання оперативної та відеопам’яті - Швидкість завантаження ресурсів - Мережеві затримки для онлайн-додатків

2. Динамічність характеристик продуктивності. На відміну від статичного аналізу коду, облік швидкодії вимагає постійного моніторингу в реальному часі, оскільки продуктивність може суттєво варіюватися залежно від: - Поточного стану додатку - Дій користувача - Завантаженості сцени - Конфігурації апаратного забезпечення

3. Складність інтерпретації даних. Сирі метрики продуктивності часто не дають чіткого розуміння причин проблем. Наприклад, низький FPS може бути спричинений: - Надмірною кількістю полігонів у сцені - Неоптимізованими шейдерами - Великою кількістю draw calls - Проблемами з фізичними симуляціями - Неефективним використанням пам’яті

Для розробників без глибокої експертизи в оптимізації визначення першопричини може бути складним завданням.

4. Платформозалежність. Unreal Engine 5 [1, 2] підтримує широкий спектр платформ – від мобільних пристроїв до високопродуктивних ПК та консолей. Кожна платформа має свої особливості: - Різні архітектури процесорів (x86, ARM) - Різні графічні API (DirectX, Vulkan, Metal) - Різні обмеження пам’яті та продуктивності - Специфічні вимоги до оптимізації

5. Вплив профілювання на продуктивність. Парадокс полягає в тому, що сам процес збору метрик може негативно впливати на продуктивність додатку, спотворюючи результати вимірювань. Це явище, відоме як “ефект спостерігача” (observer effect), вимагає розробки мінімально інвазивних методів профілювання.

6. Відсутність контексту та рекомендацій. Традиційні інструменти профілювання надають детальну інформацію про стан системи, але не пропонують конкретних дій для вирішення виявлених проблем. Розробнику доводиться самостійно: - Аналізувати великі обсяги даних - Визначати пріоритети оптимізації - Шукати оптимальні рішення - Оцінювати потенційний ефект від змін

7. Складність налаштування порогових значень. Визначення того, що є “нормальною” продуктивністю, залежить від типу додатку: - Для VR-додатків [6] критично важливо підтримувати стабільні 90-120 FPS - Для мобільних ігор прийнятними можуть бути 30 FPS - Для competitive шутерів бажано мати 144+ FPS

8. Проблема масштабування. З розвитком проекту складність аналізу продуктивності зростає експоненційно: - Збільшується кількість об’єктів у сцені - Ускладнюється логіка гри - Зростає кількість систем, що взаємодіють - Накопичується технічний борг

Аналіз великих проектів з мільйонами рядків коду та тисячами ассетів стає надзвичайно складним завданням без автоматизованих інструментів.

9. Недостатня інтеграція з процесом розробки. Часто профілювання продуктивності проводиться лише на фінальних етапах розробки, коли внесення

змін є дорогим та ризикованим. Відсутність постійного моніторингу призводить до: - Накопичення проблем з продуктивністю - Складності локалізації джерел проблем - Необхідності компромісів між якістю та швидкодією

10. Брак експертизи в команді. Оптимізація продуктивності вимагає глибоких знань:

- Архітектури комп'ютерів
- Особливостей графічного конвеєра
- Алгоритмів та структур даних
- Специфіки конкретного двигуна

Не всі команди мають досвідчених спеціалістів з оптимізації, що ускладнює ефективне вирішення проблем продуктивності.

Враховуючи описані проблеми, стає очевидною необхідність створення інтелектуальної системи, яка б не просто збирала метрики, а надавала експертну підтримку в процесі оптимізації. Така система повинна: - Автоматично виявляти проблеми продуктивності - Визначати їх першопричини - Пропонувати конкретні рішення - Оцінювати потенційний ефект від оптимізацій - Навчатися на основі накопиченого досвіду

Саме такий підхід реалізується в розробленій експертній системі обліку швидкодії для Unreal Engine 5.

1.2. Огляд існуючих методів та систем профілювання продуктивності ПЗ

Сучасний ринок інструментів профілювання пропонує широкий спектр рішень, кожне з яких має свої переваги та обмеження. Розглянемо основні категорії та представників.

Вбудовані інструменти Unreal Engine:

Unreal Insights – це комплексний інструмент профілювання, що входить до складу Unreal Engine [1]. Він надає детальну інформацію про: - CPU та GPU timing - Використання пам'яті - Мережеву активність - Завантаження ассетів

Переваги: глибока інтеграція з двигуном, детальна інформація, підтримка всіх платформ. Недоліки: складність інтерпретації даних, відсутність автоматичних рекомендацій, високий поріг входу.

Stat Commands – система консольних команд для отримання статистики в реальному часі: - stat fps – відображення FPS - stat unit – час рендерингу кадру - stat gpu – статистика GPU

Переваги: простота використання, миттєвий доступ до базових метрик. Недоліки: обмежена функціональність, відсутність історії даних, текстове представлення.

Session Frontend – інструмент для профілювання та налагодження сесій гри. Дозволяє: - Записувати та відтворювати сесії - Аналізувати продуктивність у часі - Порівнювати різні запуски

Переваги: можливість аналізу після факту, порівняння сесій. Недоліки: потребує налаштування, генерує великі файли даних.

Зовнішні профайлери:

NVIDIA Nsight – потужний інструмент для аналізу GPU продуктивності: - Детальний аналіз шейдерів - Візуалізація GPU timeline - Оптимізація використання VRAM [5]

Переваги: глибокий аналіз GPU, підтримка ray tracing, інтеграція з IDE. Недоліки: працює тільки з NVIDIA картами, складність налаштування.

Intel VTune Profiler – професійний інструмент для CPU профілювання: - Аналіз на рівні інструкцій процесора - Виявлення hotspots - Аналіз паралелізму [3,4]

Переваги: надзвичайно детальний аналіз CPU, підтримка різних архітектур. Недоліки: орієнтований на Intel процесори, складний для початківців.

RenderDoc – графічний дебагер для аналізу рендерингу: - Покадровий аналіз рендерингу - Інспекція текстур та буферів - Аналіз draw calls

Переваги: безкоштовний, кросплатформений, детальний аналіз графіки. Недоліки: фокус тільки на рендерингу, не аналізує логіку гри.

Хмарні рішення:

GameBench – хмарна платформа для моніторингу продуктивності мобільних ігор: - Збір метрик з реальних пристроїв - Аналітика та звітність - Порівняння з конкурентами

Переваги: тестування на реальних пристроях, централізована аналітика. Недоліки: платна підписка, залежність від інтернету.

Спеціалізовані рішення для ігор:

Unity Profiler – аналог для Unity двигуна з схожим функціоналом.

PIX for Windows – інструмент від Microsoft для профілювання DirectX додатків.

Порівняльний аналіз існуючих рішень:

Таблиця 1.1

Порівняння інструментів профілювання

Критерій	Unreal Insights	NVIDIA Nsight	Intel VTune	RenderDoc
Інтеграція з UE5	Відмінна	Середня	Низька	Середня
Простота використання	Середня	Складно	Складно	Середня
Автоматичні рекомендації	Немає	Обмежені	Немає	Немає
Реальний час	Так	Частково	Ні	Ні
Вартість	Безкоштовно	Безкоштовно	Платно	Безкоштовно

Недоліки існуючих рішень:

1. **Відсутність інтелектуального аналізу.** Жоден з розглянутих інструментів не використовує методи штучного інтелекту [9] для аналізу даних.

2. **Фрагментарність.** Кожен інструмент фокусується на певному аспекті (CPU, GPU, пам’ять), що вимагає використання декількох рішень одночасно.
3. **Складність інтерпретації.** Більшість інструментів надають сирі дані без контексту та рекомендацій.
4. **Відсутність навчання.** Системи не адаптуються до специфіки конкретного проекту та не накопичують досвід.
5. **Слабка інтеграція з робочим процесом.** Профілювання часто вимагає переривання основної роботи та переключення контексту.

Висновок з огляду:

Аналіз існуючих рішень показує, що на ринку відсутній комплексний інструмент, який би поєднував:

- Глибоку інтеграцію з Unreal Engine 5
- Інтелектуальний аналіз даних
- Автоматичне формування рекомендацій
- Простоту використання
- Адаптивність до проекту

Це підтверджує актуальність розробки експертної системи, яка б заповнила існуючі прогалини та надала розробникам потужний інструмент для оптимізації продуктивності. [7]

1.3. Особливості використання Unreal Engine 5 для розробки експертних систем

Unreal Engine 5 представляє унікальну платформу для розробки не тільки ігор та візуалізацій, але й складних інтелектуальних систем. Розглянемо ключові особливості, які роблять UE5 придатним для створення експертної системи обліку швидкодії.

Архітектурні переваги UE5:

Модульна архітектура двигуна дозволяє інтегрувати експертну систему як незалежний модуль, що може бути увімкнений або вимкнений за потреби. Система плагінів UE5 забезпечує:

- Ізоляцію коду експертної системи
- Можливість розповсюдження як окремого продукту
- Легку інтеграцію в існуючі проекти

Reflection System – потужна система рефлексії UE5 надає можливість:

- Динамічно отримувати інформацію про типи та об’єкти
- Автоматично серіалізувати дані
- Створювати гнучкі системи правил [11, 13]

Blueprint Visual Scripting забезпечує:

- Візуальне представлення логіки експертної системи
- Можливість модифікації правил без рекомпіляції
- Доступність для не-програмістів

Можливості для реалізації бази знань [12]:

DataAssets – спеціальний тип ассетів для зберігання даних:

DataTables для структурованого зберігання правил:

- Імпорт/експорт з CSV
- Редагування в редакторі
- Версіонування через систему контролю версій

Інтеграція з C++ та Blueprint:

Гібридний підхід дозволяє:

- Реалізувати критичні компоненти на C++ для продуктивності
- Використовувати Blueprint для логіки високого рівня
- Забезпечити гнучкість налаштування

Візуалізація через UMG (Unreal Motion Graphics):

UMG надає потужні засоби для створення інтерфейсу:

- Slate widgets для нативних елементів

- Data binding для автоматичного оновлення
- Анімації та переходи

Особливості Nanite та Lumen:

UE5 вводить революційні технології, які потребують спеціального підходу:

Nanite Virtualized Geometry:

- Автоматичний LOD
- Мільйони полігонів без втрати продуктивності
- Нові метрики для аналізу

Lumen Global Illumination:

- Динамічне освітлення в реальному часі
- Значне навантаження на GPU
- Специфічні оптимізації

World Partition System:

Нова система завантаження світу вимагає особливого підходу:

- Динамічне завантаження/вивантаження
- Streaming метрики
- Аналіз пам’яті по регіонах

Переваги використання UE5 для експертної системи:

1. **Повний доступ до внутрішніх метрик** – на відміну від зовнішніх профайлерів
2. **Реальний час** – аналіз без зупинки додатку
3. **Контекстна інформація** – розуміння стану гри
4. **Візуальні можливості** – багаті засоби візуалізації
5. **Кросплатформенність** – підтримка всіх цільових платформ

Виклики та обмеження:

1. **Продуктивність** – експертна система не повинна суттєво впливати на FPS
2. **Пам’ять** – мінімізація використання RAM

3. **Складність** – необхідність глибокого розуміння архітектури UE5

Unreal Engine 5 надає всі необхідні інструменти та можливості для створення повноцінної експертної системи обліку швидкодії, що робить його ідеальною платформою для реалізації поставленого завдання.

1.4. Постановка завдання розробки експертної системи

На основі проведеного аналізу предметної області, огляду існуючих рішень та вивчення можливостей Unreal Engine 5, сформулюємо детальну постановку завдання розробки експертної системи обліку швидкодії.

Мета розробки:

Створити інтегровану в Unreal Engine 5 експертну систему, яка автоматично аналізує продуктивність додатку в реальному часі, виявляє проблеми та надає обґрунтовані рекомендації щодо оптимізації.

Функціональні вимоги:

FR1. Збір метрик продуктивності:

- Частота кадрів (FPS) з точністю до 0.1
- Час рендерингу кадру (Frame Time) в мілісекундах
- Використання CPU по ядрах
- Навантаження GPU
- Споживання RAM та VRAM
- Кількість draw calls
- Кількість полігонів у кадрі
- Статистика по текстурах та матеріалах

FR2. Аналіз даних в реальному часі:

- Виявлення аномалій продуктивності
- Ідентифікація bottlenecks
- Кореляційний аналіз метрик
- Прогнозування проблем

FR3. База знань експертної системи повинна містити:

- Правила для 50+ типових проблем продуктивності
- Рекомендації для кожної проблеми
- Оцінки складності виправлення
- Очікуваний ефект від оптимізації

FR4. Механізм логічного виведення:

- Forward chaining для проактивного аналізу
- Backward chaining для діагностики
- Підтримка нечіткої логіки
- Можливість навчання на основі feedback

FR5. Інтерфейс користувача:

- Dashboard з ключовими метриками
- Графіки продуктивності в часі
- Список проблем з пріоритетами
- Детальні рекомендації з прикладами
- Історія аналізів

FR6. Інтеграція з workflow:

- Автоматичний запуск при Play in Editor
- Збереження звітів
- Експорт даних в CSV/JSON
- Integration з системами контролю версій

Нефункціональні вимоги:

NFR1. Продуктивність:

- Overhead не більше 2% FPS
- Використання пам’яті < 100 MB
- Час аналізу < 100ms на кадр

NFR2. Масштабованість:

- Підтримка проектів до 10 млн полігонів
- До 10000 акторів у сцені

- До 1000 матеріалів

NFR3. Надійність:

- Стабільна робота протягом 8 годин
- Graceful degradation при перевантаженні
- Автоматичне відновлення після збоїв

NFR4. Юзабіліті:

- Інтуїтивний інтерфейс
- Контекстна допомога
- Навчальні матеріали

NFR5. Сумісність:

- UE5.0 та вище
- Windows 10/11, Linux, macOS - DX11, DX12, Vulkan

Обмеження:

1. Система працює тільки в межах Unreal Engine 5
2. Не аналізує зовнішні процеси
3. Потребує прав розробника для повного функціоналу
4. Не модифікує код автоматично

Критерії успіху:

1. Скорочення часу на оптимізацію на 40%
2. Виявлення 90% критичних проблем продуктивності
3. Точність рекомендацій > 80%
4. Позитивний feedback від 75% користувачів

Архітектурні рішення:

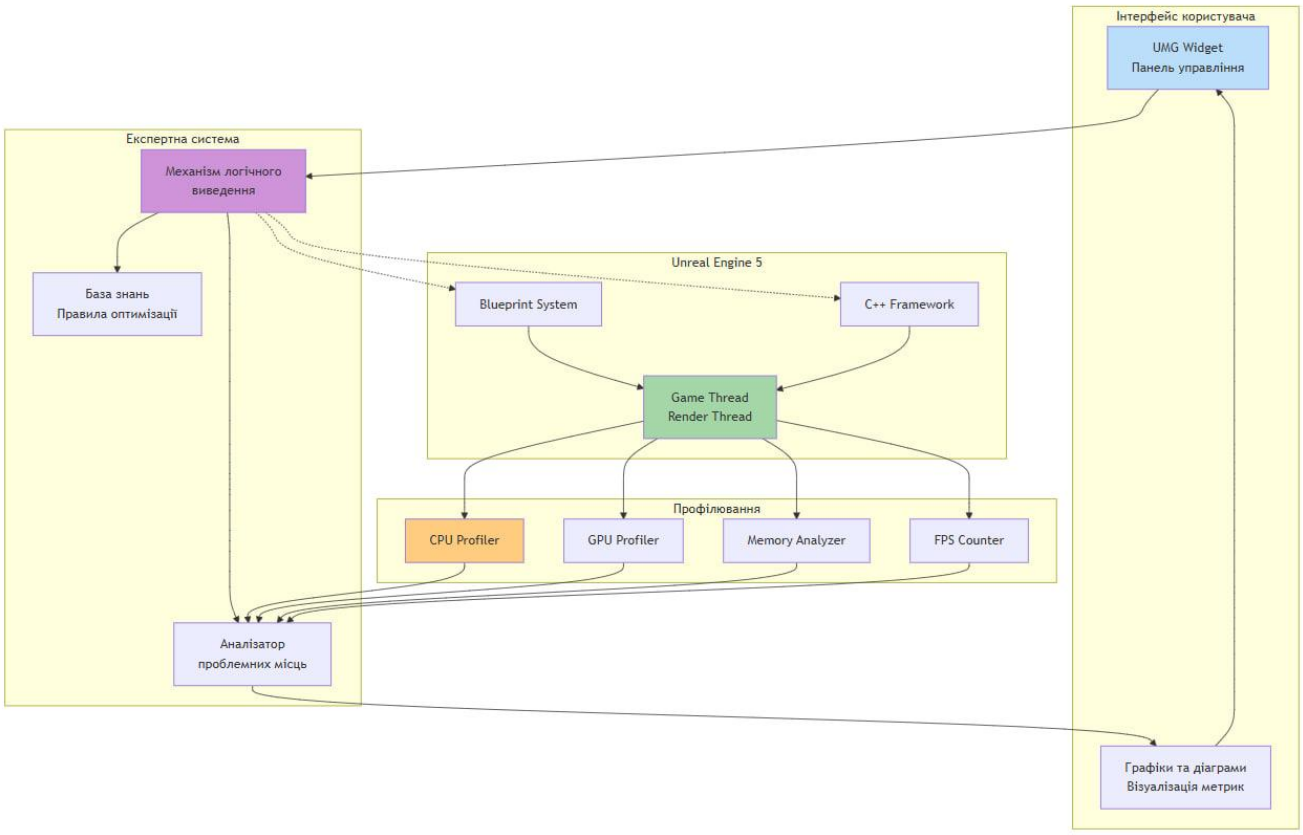


Рис. 1.1 Модульна архітектура

Технологічний стек: - C++ для критичних компонентів - Blueprint для логіки високого рівня - SQLite для зберігання історії - JSON для конфігурації - Slate/UMG для UI

Етапи розробки:

1. **Фаза 1: Прототип (2 тижні)**

– Базовий збір метрик

– Простий UI

– 10 правил в базі знань

2. **Фаза 2: MVP (1 місяць)**

– Повний набір метрик

– Механізм виведення

– 25+ правил

– Функціональний UI

3. **Фаза 3: Beta (1 місяць)**

– Оптимізація продуктивності

– Розширена база знань (50+ правил)

– Тестування на реальних проектах

4. **Фаза 4: Release (2 тижні)**

– Виправлення багів

– Документація

– Підготовка до релізу

Ризики та мітигація:

Таблиця 1.2

Аналіз ризиків проекту

Ризик	Ймовірність	Вплив	Мітигація
Високий overhead	Середня	Високий	Профілювання, оптимізація
Складність інтеграції	Низька	Середній	Модульна архітектура
Неточні рекомендації	Середня	Високий	Тестування, feedback loop
Зміни в UE5 API	Низька	Низький	Абстракція, версіонування

Дана постановка завдання визначає чіткі цілі, вимоги та обмеження для розробки експертної системи, що дозволить створити ефективне та корисне рішення для спільноти розробників на Unreal Engine 5.

Висновки до розділу 1

У першому розділі проведено комплексний системний аналіз предметної області обліку швидкодії програмного забезпечення з фокусом на специфіку Unreal Engine 5.

Виявлено ключові проблеми сучасних підходів до профілювання:

- Багатовимірність та динамічність метрик продуктивності

- Складність інтерпретації сирих даних без контексту

- Відсутність інтелектуального аналізу та автоматичних рекомендацій

- Слабка інтеграція з робочим процесом розробки
- Високий поріг входу для недосвідчених розробників

Проведений огляд існуючих рішень показав, що жоден з доступних інструментів не забезпечує комплексного підходу до аналізу продуктивності з використанням методів штучного інтелекту. Більшість рішень фокусується на збиранні даних, залишаючи інтерпретацію та прийняття рішень повністю на розробника.

Дослідження особливостей Unreal Engine 5 підтвердило, що платформа надає всі необхідні інструменти та API для створення інтегрованої експертної системи:

- Глибокий доступ до внутрішніх метрик двигуна
- Потужна система рефлексії та плагінів
- Гнучкі можливості візуалізації через UMG
- Підтримка гібридної розробки C++/Blueprint

На основі проведеного аналізу сформульовано детальну постановку завдання, яка включає функціональні та нефункціональні вимоги, архітектурні рішення та план розробки експертної системи.

Результати аналізу підтверджують актуальність та технічну можливість створення експертної системи обліку швидкодії для Unreal Engine 5, яка заповнить існуючі прогалини в інструментарії розробників та суттєво підвищить ефективність процесу оптимізації.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ЕКСПЕРТНОЇ СИСТЕМИ ОБЛІКУ ШВИДКОДІЇ

2.1. Концептуальна модель експертної системи

Концептуальна модель експертної системи обліку швидкодії визначає основні компоненти системи, їх взаємозв'язки та принципи функціонування. Розроблена модель базується на класичній архітектурі експертних систем з адаптацією під специфіку задачі профілювання продуктивності в Unreal Engine 5.

Основні компоненти концептуальної моделі:

1. **Підсистема збору даних (Data Collection Subsystem)**

   Відповідає за отримання метрик продуктивності з різних джерел:

– Performance counters двигуна
– Системні метрики OS
– GPU статистика
– Користувацькі метрики

2. **База знань (Knowledge Base)**

   Зберігає експертні знання у вигляді:

– Продукційних правил (IF-THEN)
– Фреймів для опису типових ситуацій
– Семантичних мереж зв'язків між проблемами
– Онтології предметної області

3. **Механізм логічного виведення (Inference Engine)**

   Реалізує алгоритми міркування:

– Forward chaining для проактивного аналізу
– Backward chaining для діагностики проблем
– Fuzzy logic для обробки неточних даних
– Case-based reasoning для використання попереднього досвіду

4. **Робоча пам'ять (Working Memory)**

   Тимчасове сховище для:

– Поточних метрик продуктивності
– Проміжних результатів аналізу
– Активних гіпотез
– Контексту поточної сесії

5. **Підсистема пояснень (Explanation Subsystem)**

   Забезпечує транспарентність рішень:

– Трасування ланцюжків виведення
– Обґрунтування рекомендацій
– Візуалізація процесу міркування

6. **Інтерфейс користувача (User Interface)**

   Надає засоби взаємодії:

– Dashboard метрик
– Список проблем та рекомендацій
– Налаштування параметрів
– Експорт звітів

7. **Модуль навчання (Learning Module)**

   Забезпечує адаптацію системи:

– Збір feedback від користувачів
– Коригування правил
– Накопичення статистики ефективності

Формальне представлення моделі:

Експертну систему можна представити як кортеж:

ES = ⟨D, K, I, W, E, U, L⟩

де:

- D – множина джерел даних
- K – база знань
- I – механізм виведення
- W – робоча пам’ять

- E – підсистема пояснень

- U – інтерфейс користувача

- L – модуль навчання

База знань формально описується як:

$K = \langle R, F, O \rangle$

де:

- $R = \{r_1, r_2, \ldots, r_n\}$ – множина правил

- $F = \{f_1, f_2, \ldots, f_m\}$ – множина фреймів

- O – онтологія предметної області

Правило r_i має структуру:

r_i: IF ($condition_1$ AND $condition_2$ AND … AND $condition_k$) THEN ($action_1$, $action_2$, …, $action_p$) WITH confidence = c_i

```
Rule_HighDrawCalls:
  IF (DrawCallCount > 3000 AND
      FPS < 30 AND
      Platform = "Mobile")
  THEN (Problem = "Excessive Draw Calls",
        Priority = "High",
        Recommendation = "Implement batching",
        ExpectedImprovement = "20-30% FPS")
  WITH confidence = 0.85
```

Рис. 2 Приклад правила

Онтологія предметної області:

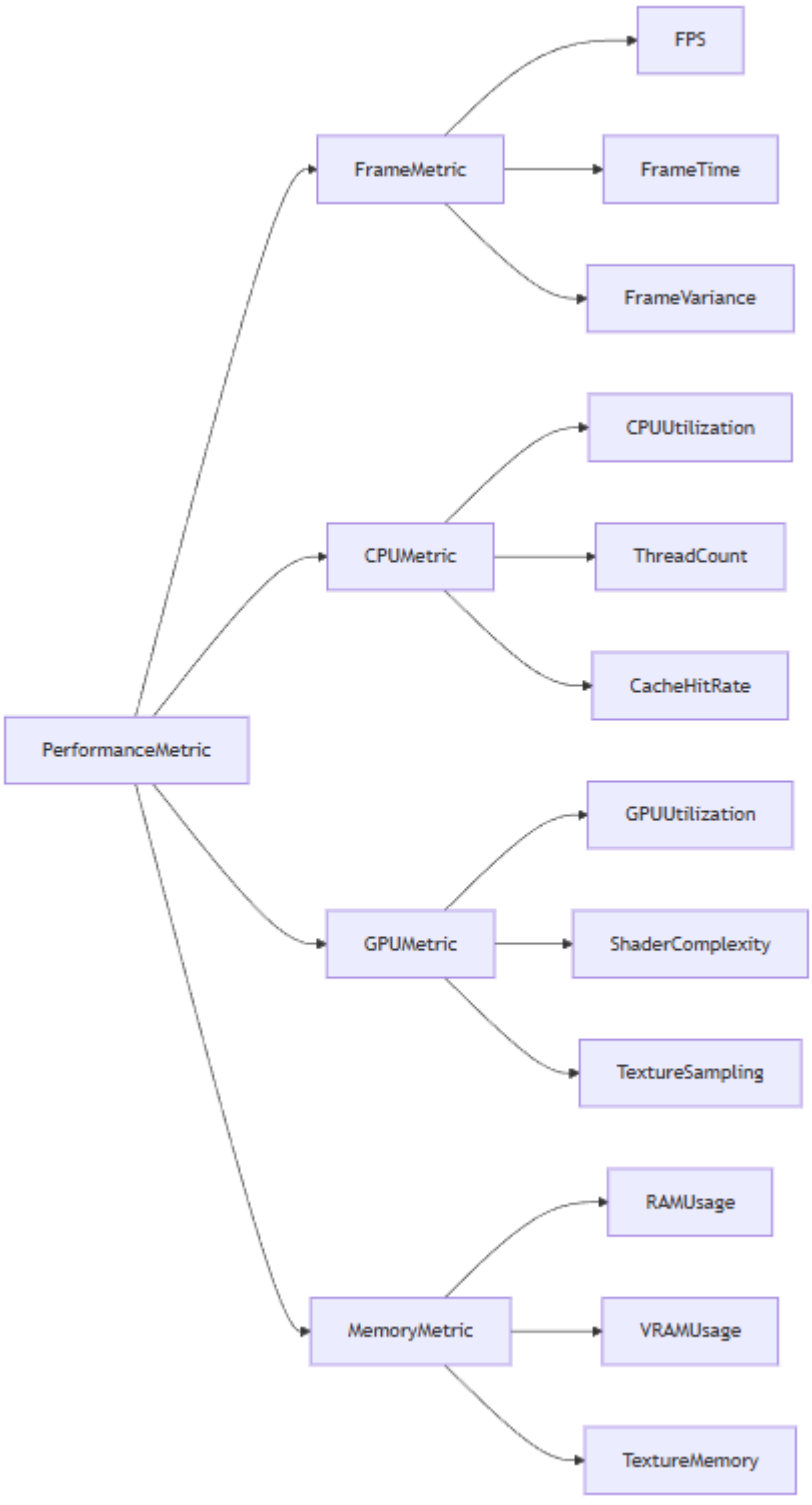


Рис. 2.1 Онтологія ієрархії концептів та їх відношення

Процес функціонування системи:

1. **Ініціалізація:**

– Завантаження бази знань
– Підключення до джерел даних
– Ініціалізація UI
– Збір даних
– Аналіз даних
– Формування рекомендацій
– Візуалізація

Інтеграція з Unreal Engine 5:

Концептуальна модель враховує специфіку UE5:

1. **Tick-based оновлення** – синхронізація з game loop
2. **Actor-Component архітектура** – модульність системи
3. **Blueprint сумісність** – візуальне налаштування правил
4. **Reflection system** – динамічний доступ до метрик

Переваги розробленої моделі:

1. **Модульність** – легке розширення функціоналу
2. **Гнучкість** – адаптація під різні типи проектів
3. **Прозорість** – пояснення кожного рішення
4. **Ефективність** – мінімальний вплив на продуктивність

Розроблена концептуальна модель забезпечує системний підхід до вирішення задачі обліку швидкодії та створює фундамент для подальшого проектування та реалізації експертної системи.

2.2. Функціональне моделювання процесів обліку швидкодії

Функціональне моделювання дозволяє детально описати процеси, що відбуваються в експертній системі обліку швидкодії. Для моделювання використовується нотація IDEF0, яка забезпечує ієрархічне представлення функцій системи.

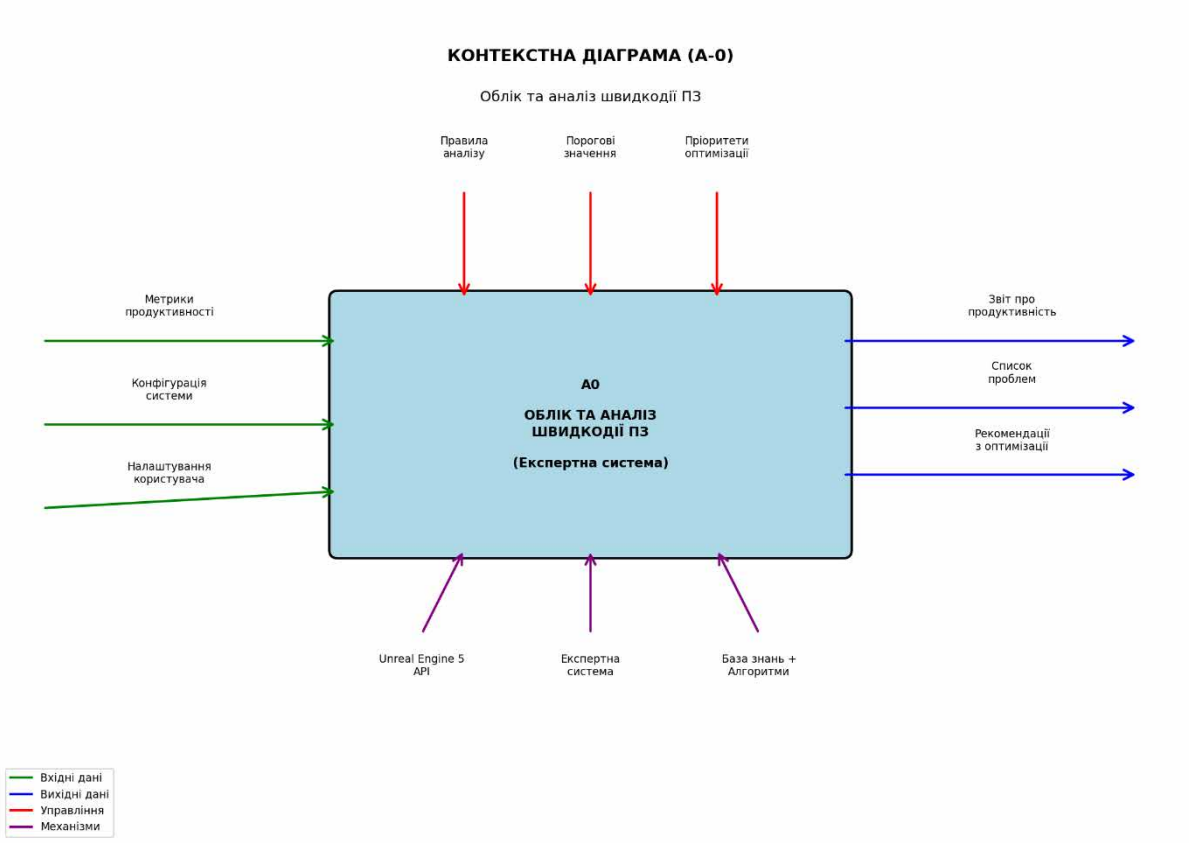


Рис. 2.2 Контекстна діаграма (A-0)

Контекстна діаграма (A-0):

На найвищому рівні абстракції система представляється як єдиний процес “Облік та аналіз швидкодії ПЗ”:

Входи:

- Метрики продуктивності
- Конфігурація системи
- Користувацькі налаштування

Виходи:

- Звіт про продуктивність
- Список проблем
- Рекомендації з оптимізації
- Статистика трендів

Управління:

- Правила аналізу
- Порогові значення
- Пріоритети оптимізації

Механізми:

- Unreal Engine 5 API
- Експертна система
- База знань
- Алгоритми аналізу

Декомпозиція першого рівня (A0):

Основний процес декомпозується на п’ять підпроцесів:

1. **A1: Збір метрик продуктивності**
   – Входи: Raw performance data
   – Виходи: Structured metrics
   – Механізми: Stats API, Profiling hooks
2. **A2: Попередня обробка даних**
   – Входи: Structured metrics
   – Виходи: Normalized data
   – Механізми: Filters, Aggregators
3. **A3: Аналіз продуктивності**
   – Входи: Normalized data, Rules
   – Виходи: Identified issues
   – Механізми: Inference engine
4. **A4: Формування рекомендацій**
   – Входи: Identified issues, Knowledge base
   – Виходи: Recommendations

– Механізми: Expert rules

5. **A5: Візуалізація результатів**

– Входи: All analysis results

– Виходи: UI representation

– Механізми: UMG, Charts library

Детальна декомпозиція процесу A1 (Збір метрик):

A1.1: Збір CPU метрик

- Thread utilization
- Cache performance
- Instruction throughput

A1.2: Збір GPU метрик

- Shader execution time
- Memory bandwidth
- Compute utilization

A1.3: Збір метрик пам'яті

- RAM allocation
- VRAM usage
- Texture streaming

A1.4: Збір метрик двигуна

- Draw calls
- Actor count
- Tick time

Діаграма потоків даних (DFD):

Рівень 0 - Контекстна діаграма:

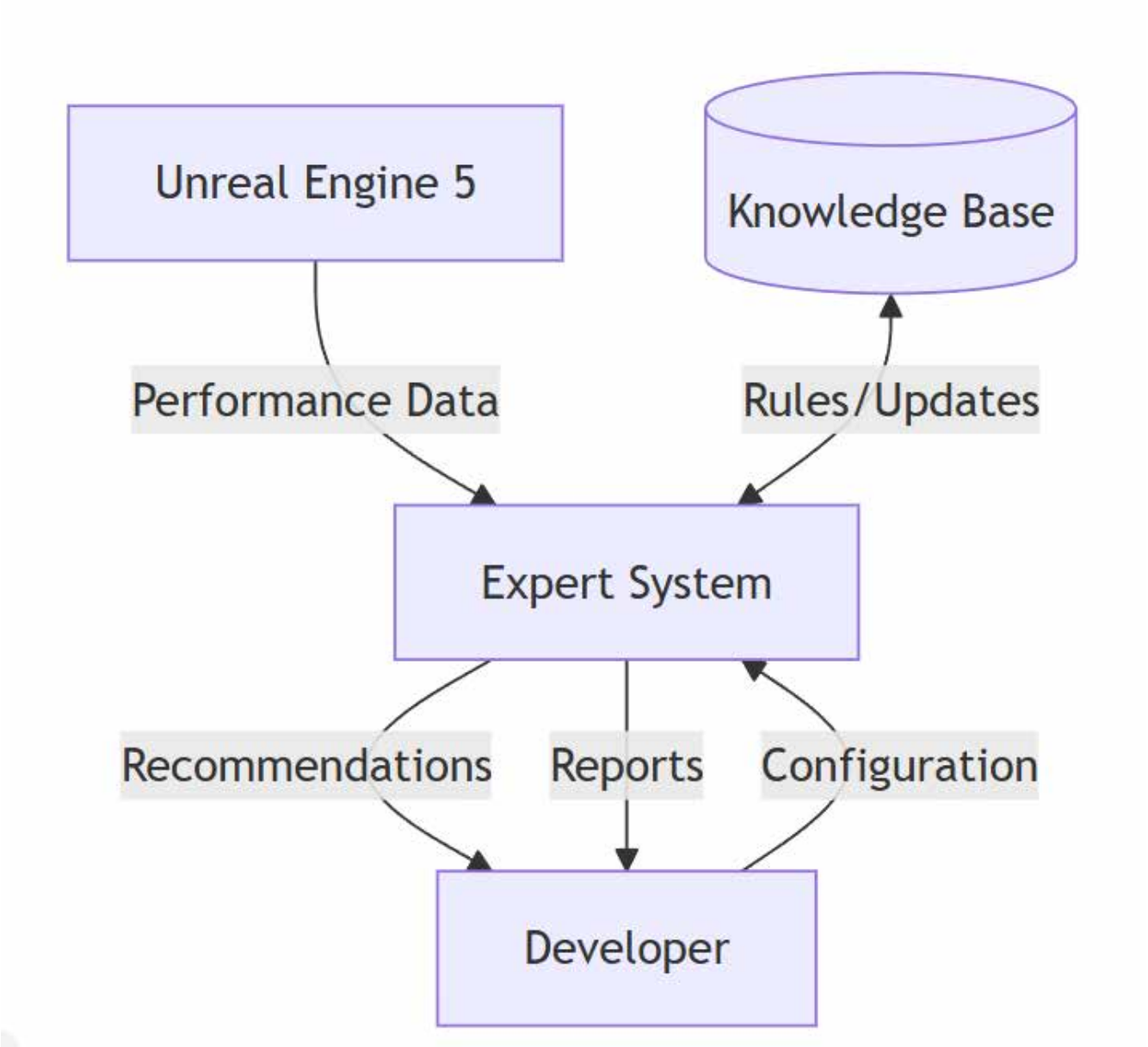


Рис. 2.3 Контекстна діаграма

Рівень 1 - Основні процеси:

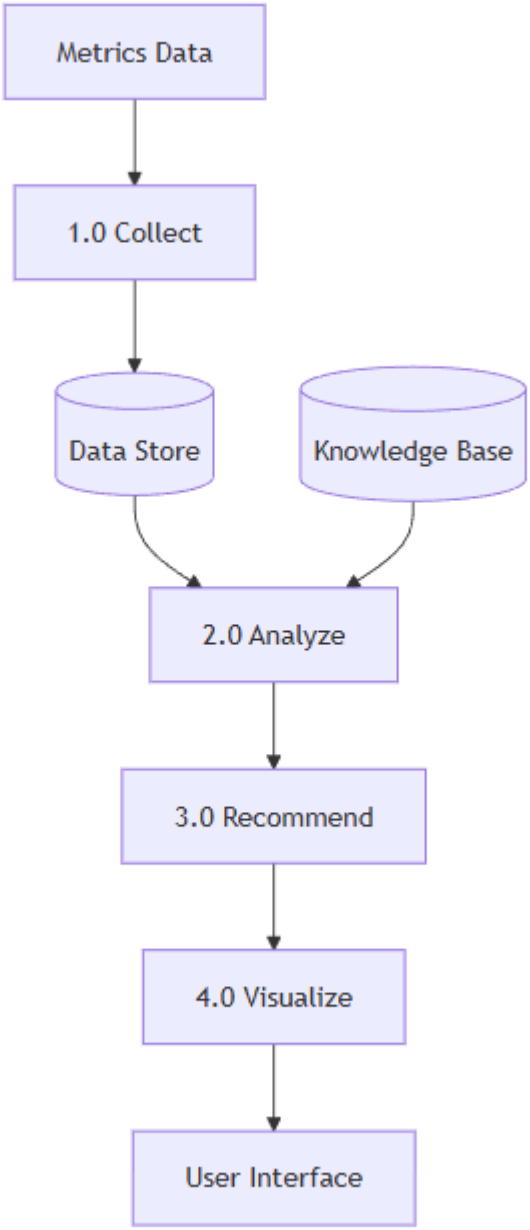


Рис. 2.4 Основні процеси

Модель станів системи:

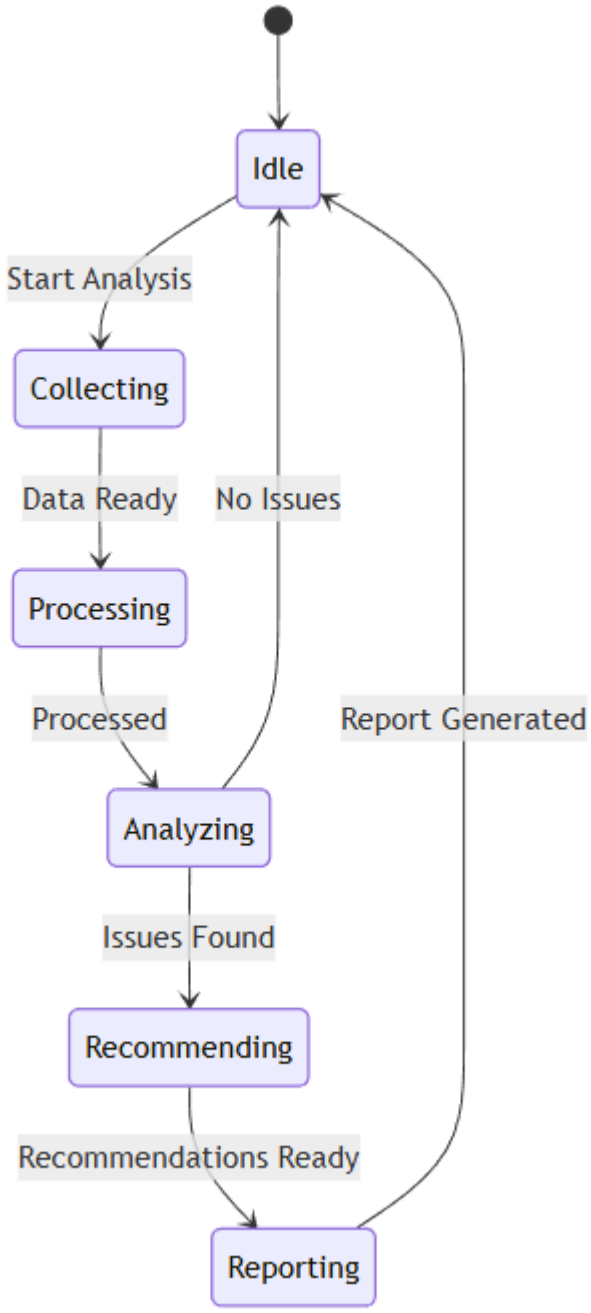


Рис. 2.5 Модель станів системи

Таблиця процесів та їх характеристик:

Таблиця 2.1

Характеристики основних процесів

Процес	**Частота виконання**	**Час виконання**	**Пріоритет**	**Критичність**
Збір метрик	Кожен кадр	< 1 ms	Високий	Критичний
Попередня обробка	Кожну секунду	< 5 ms	Середній	Важливий
Аналіз	Кожні 5 секунд	< 20 ms	Низький	Важливий
Рекомендації	За запитом	< 50 ms	Низький	Некритичний
Візуалізація	30 FPS	< 2 ms	Середній	Важливий

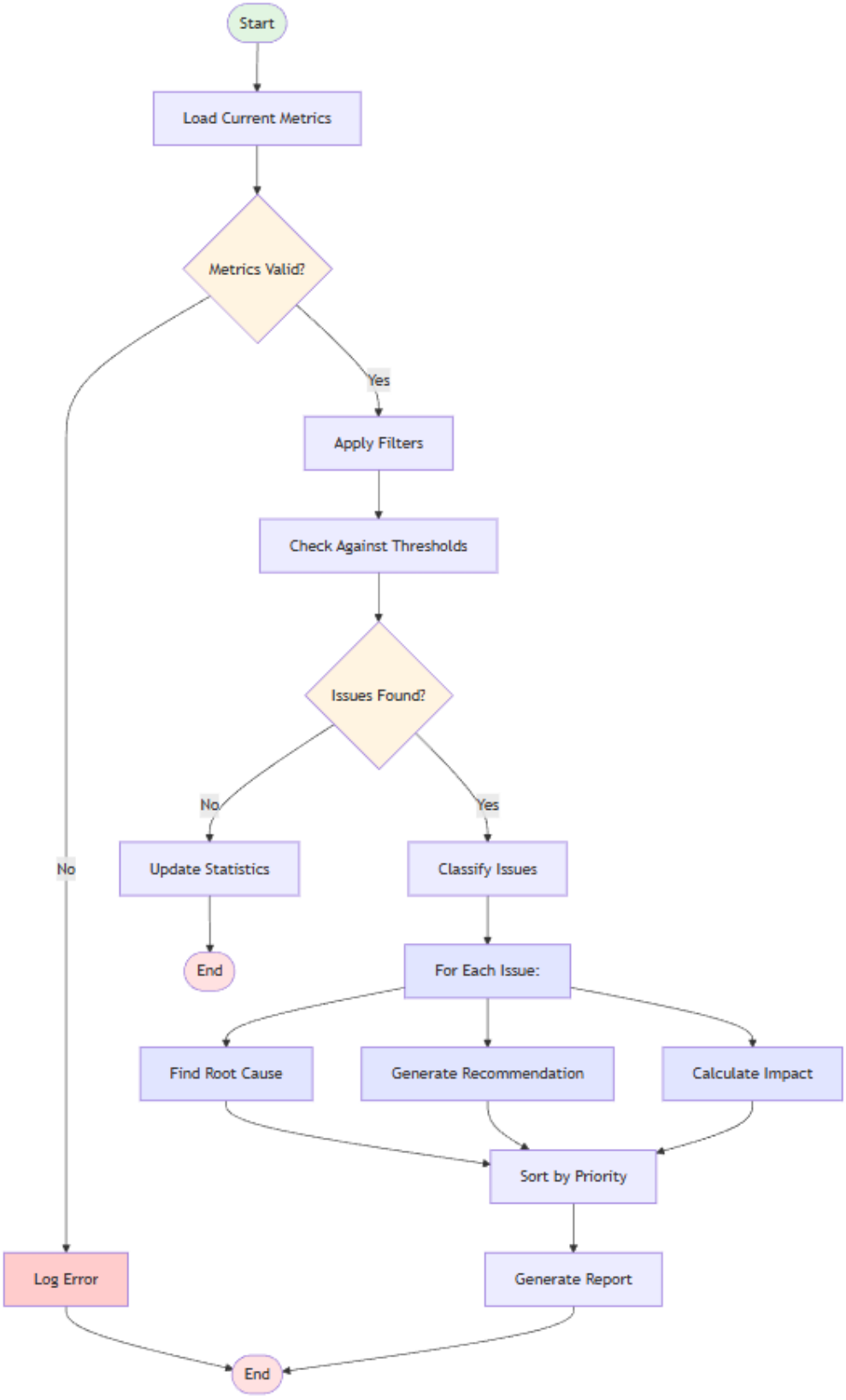


Рис. 2.6 Потокова діаграма процесу аналізу

Функціональне моделювання забезпечує чітке розуміння процесів, що відбуваються в системі, їх взаємозв’язків та послідовності виконання, що є критично важливим для успішної реалізації експертної системи.

2.3. Об’єктно-орієнтоване моделювання системи

Об’єктно-орієнтоване моделювання дозволяє представити експертну систему через взаємодію об’єктів та класів, використовуючи UML діаграми для візуалізації архітектури.

2.3.1. Діаграма прецедентів

Діаграма прецедентів (Use Case Diagram) визначає основні сценарії використання експертної системи різними акторами.

Основні актори:

1. **Developer (Розробник)** – основний користувач системи
2. **Technical Lead** – керівник, що приймає рішення про оптимізацію
3. **QA Tester** – тестувальник, що перевіряє продуктивність
4. **System** – сама експертна система (для автоматичних дій)

Основні прецеденти:

UC1: Запустити аналіз продуктивності

Актор: Developer

Передумови: Проект завантажений в UE5

Основний сценарій:

1. Розробник відкриває панель експертної системи
2. Обирає режим аналізу
3. Запускає гру в редакторі
4. Система збирає метрики
5. Система виводить результати

Альтернативні сценарії:

3а. Недостатньо прав - вивести повідомлення

4а. Збій збору даних - спробувати відновити

UC2: Налаштувати параметри аналізу

Актор: Technical Lead

Включає: UC1

Основний сценарій:

1. Відкрити налаштування
2. Встановити порогові значення
3. Обрати метрики для моніторингу
4. Зберегти конфігурацію

UC3: Переглянути рекомендації

Актор: Developer, Technical Lead

Передумови: Аналіз виконано

Основний сценарій:

1. Відкрити список проблем
2. Вибрати проблему
3. Переглянути детальний опис
4. Переглянути рекомендації
5. Застосувати виправлення

UC4: Експортувати звіт

Актор: QA Tester

Розширює: UC3

Основний сценарій:

1. Вибрати формат експорту
2. Налаштувати вміст звіту
3. Вказати шлях збереження
4. Експортувати дані

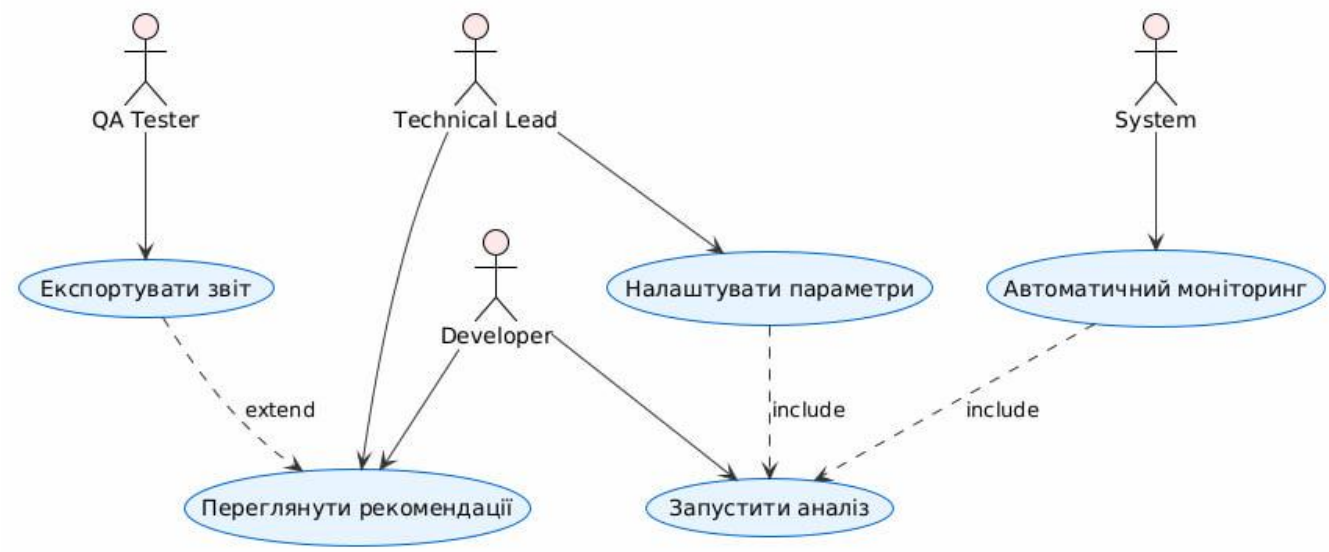


Рис. 2.7 UML діаграма прецедентів

2.3.2. Діаграма послідовності

Діаграма послідовності показує взаємодію об’єктів у часі для ключових сценаріїв.

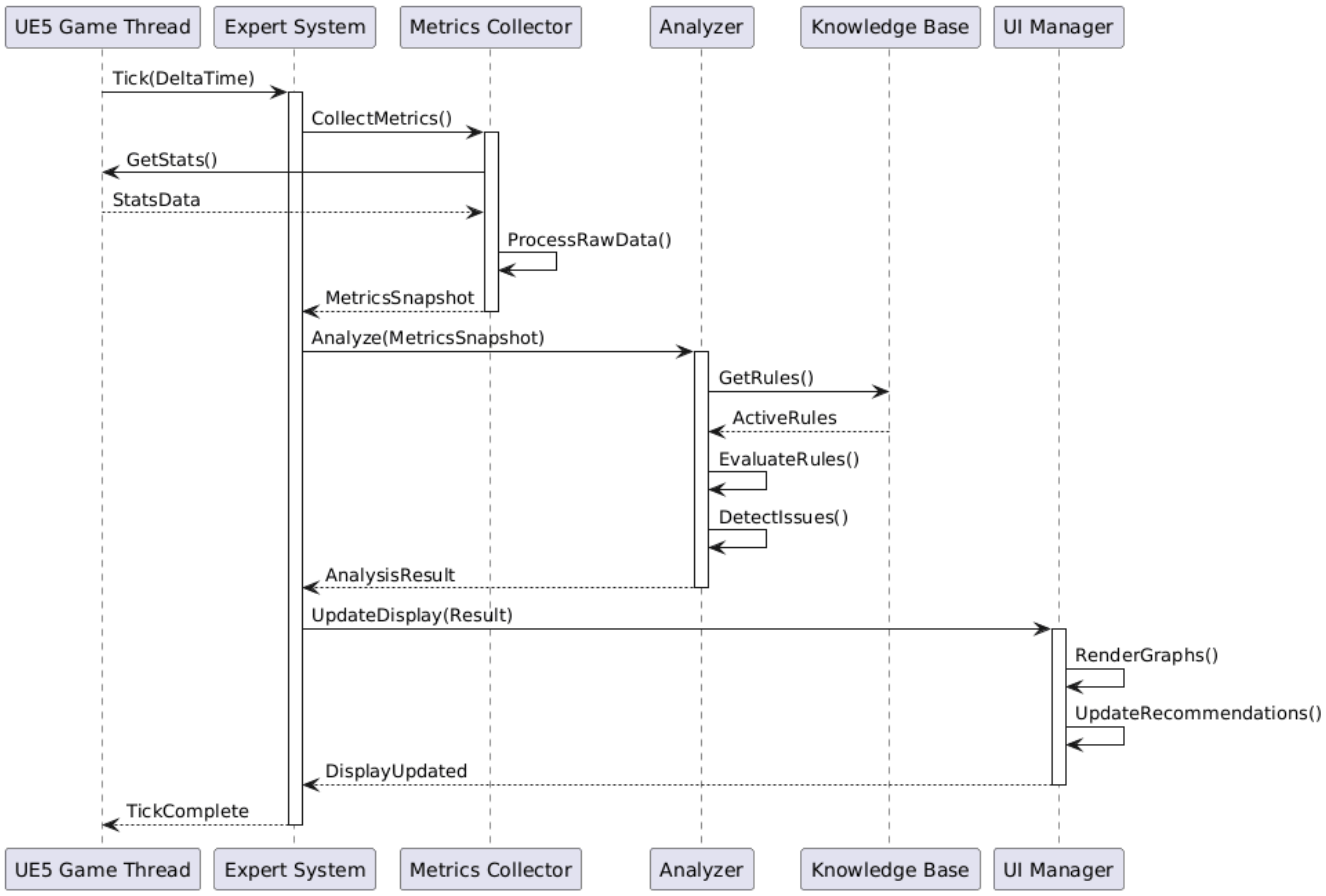


Рис. 2.8 Сценарій: Аналіз продуктивності кадру

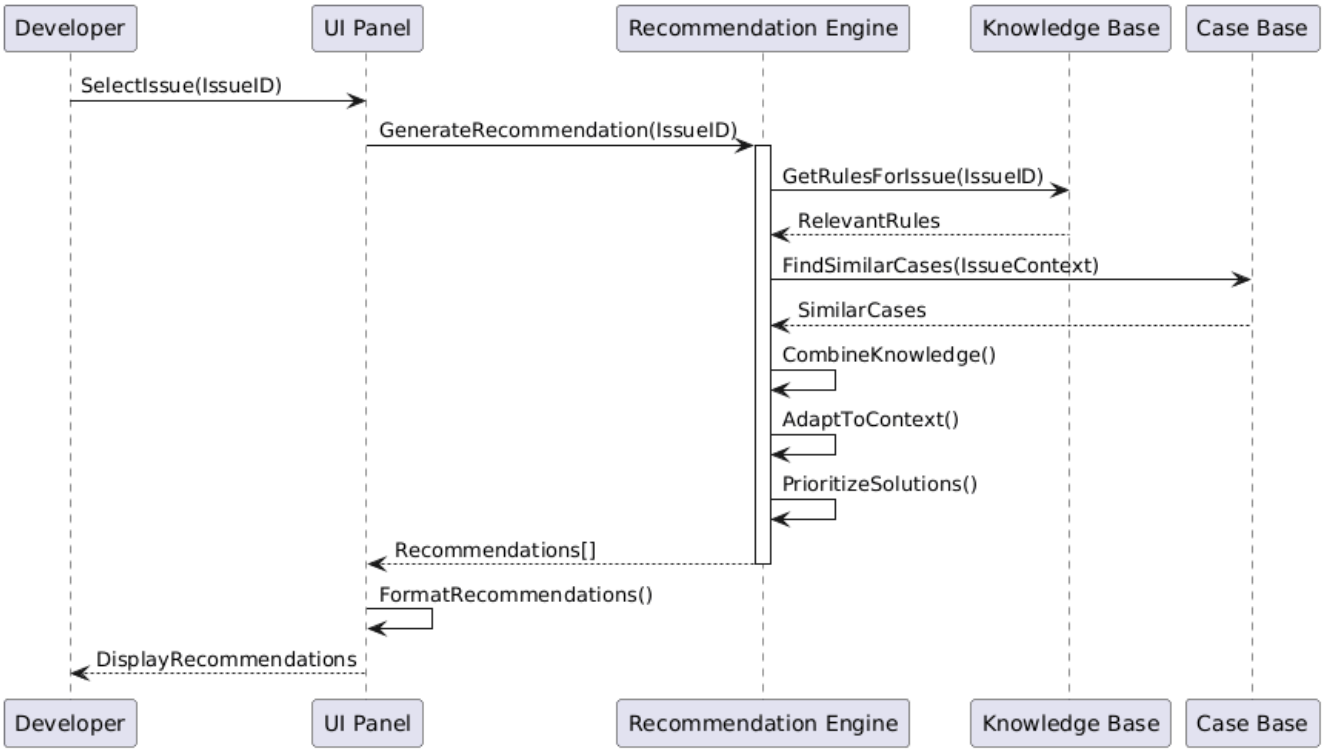


Рис. 2.9 Сценарій: Генерація рекомендації

2.3.3. Діаграма активності

Діаграма активності описує робочий процес системи та логіку прийняття рішень.

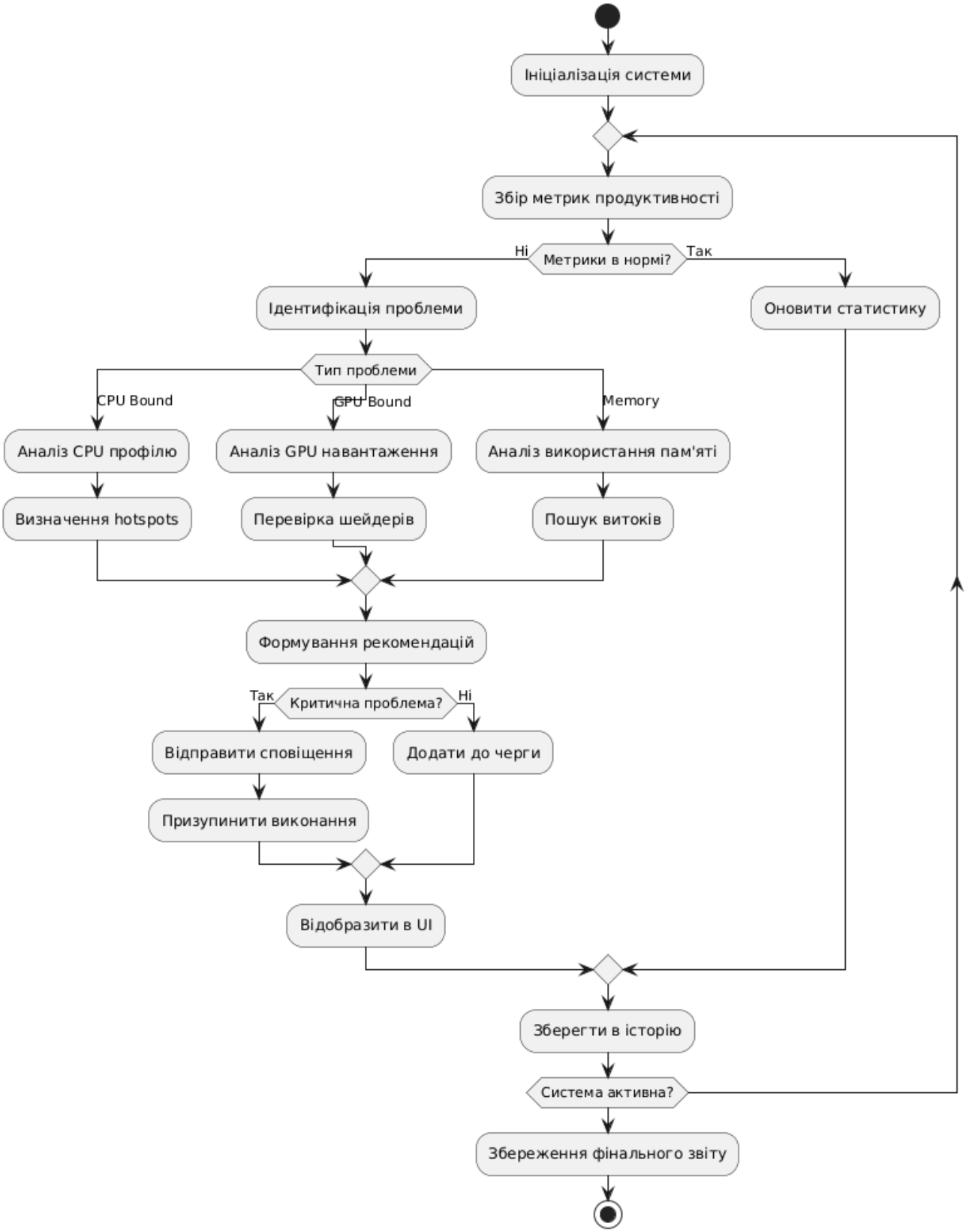


Рис. 2.10 Процес виявлення та вирішення проблем продуктивності

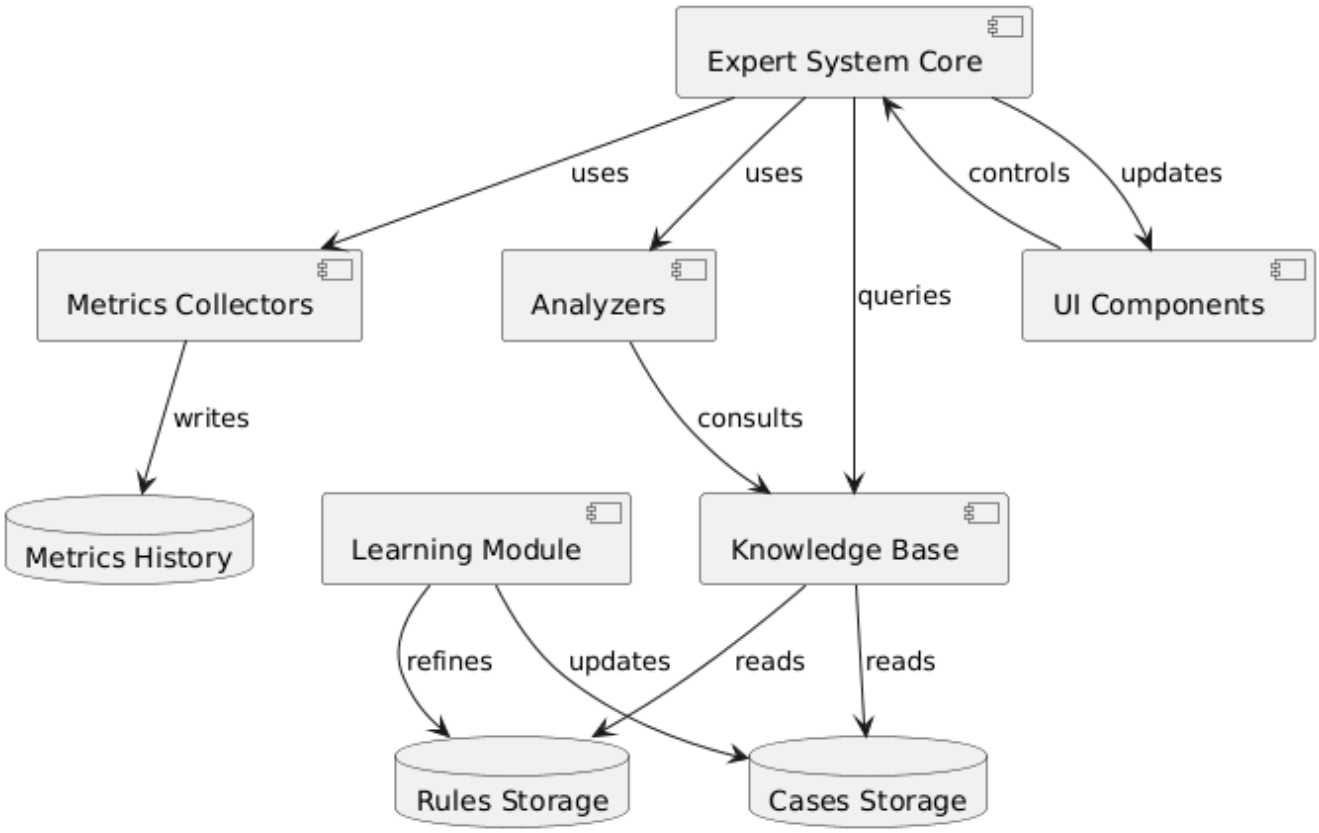


Рис. 2.11 Діаграма компонентів

Об’єктно-орієнтоване моделювання забезпечує детальне представлення архітектури системи, взаємодії компонентів та потоків даних, що є основою для ефективної реалізації експертної системи в середовищі Unreal Engine 5.

2.4. Проектування бази знань експертної системи

База знань є ключовим компонентом експертної системи, що містить формалізовані знання експертів з оптимізації продуктивності. Проектування ефективної бази знань вимагає структурованого підходу до представлення та організації експертних знань.

Структура бази знань:

База знань організована в ієрархічну структуру з декількома рівнями абстракції:

1. **Метарівень** - загальні принципи оптимізації
2. **Доменний рівень** - специфічні знання про UE5

3. **Проблемний рівень** - конкретні проблеми та рішення
4. **Контекстний рівень** - адаптація під тип проекту

База знань проблем та рішень:

Таблиця 2.2

Каталог типових проблем продуктивності

Проблема	**Симптоми**	**Причини**	**Рішення**	**Складність**
High Draw Calls	FPS < 30, CPU bound	Багато окремих мешів	Batching, Instancing	Середня
Shader Complexity	GPU > 90%, Low FPS	Складні матеріали	Simplify shaders, LOD	Висока
Memory Leak	RAM зростає	Неочищені посилання	Profile, Fix leaks	Висока
Texture Thrashing	Stuttering	Переповнення VRAM	Streaming, Compression	Середня
Tick Overhead	CPU spikes	Багато акторів	Tick optimization	Низька

Приклади експертних правил:

Правило 1: Оптимізація Nanite

```
IF (NaniteEnabled == true AND
   TriangleCount > 10000000 AND
   FPS < TargetFPS * 0.8)
THEN
   Recommend("Adjust Nanite LOD Bias")
   Recommend("Check Nanite overdraw")
   Recommend("Verify Nanite fallback meshes")
WITH confidence = 0.9
```

Правило 2: Lumen Performance

IF (LumenEnabled == true AND
GPUTime > 16ms AND
LumenSceneDetail > 0.5)
THEN
Problem("Lumen overhead too high")
Recommend("Reduce Lumen Scene Detail")
Recommend("Limit Lumen trace distance")
Recommend("Use Lumen screen traces wisely")
WITH confidence = 0.85

Правило 3: Memory Optimization

IF (TextureMemory > VRAM * 0.7 AND
MeshMemory > RAM * 0.3)
THEN
Problem("Memory pressure detected")
Priority(HIGH)
Recommend("Enable texture streaming")
Recommend("Implement mesh LODs")
Recommend("Reduce texture pool size")
WITH confidence = 0.88

Валідація та верифікація бази знань:

1. **Перевірка консистентності** - відсутність суперечливих правил
2. **Перевірка повноти** - покриття всіх типових сценаріїв
3. **Перевірка коректності** - відповідність рекомендацій best practices
4. **Тестування на реальних даних** - валідація на проектах

Спроектована база знань забезпечує гнучке та ефективне представлення експертних знань, можливість навчання та адаптації, що є критично важливим для успішного функціонування експертної системи обліку швидкодії.

Висновки до розділу 2

У другому розділі виконано комплексне моделювання експертної системи обліку швидкодії з використанням різних методологій та нотацій.

Розроблена концептуальна модель визначає архітектуру системи як сукупність семи основних компонентів: підсистеми збору даних, бази знань, механізму логічного виведення, робочої пам'яті, підсистеми пояснень, інтерфейсу користувача та модуля навчання. Формальне представлення моделі забезпечує математичну строгість опису системи.

Функціональне моделювання з використанням IDEF0 та DFD дозволило детально описати процеси обробки даних, від збору метрик до формування рекомендацій. Визначено п'ять основних процесів та їх декомпозицію, що забезпечує чітке розуміння функціонування системи.

Об'єктно-орієнтоване моделювання представило систему через призму UML діаграм: - Діаграма прецедентів визначила основні сценарії використання - Діаграми послідовності описали взаємодію компонентів у часі - Діаграма активності показала логіку прийняття рішень - Діаграма класів визначила структуру програмних компонентів

Особливу увагу приділено проектуванню бази знань, яка використовує гібридний підхід до представлення знань: - Продукційні правила для опису експертних знань - Фрейми для представлення типових ситуацій - Семантичні мережі для зв'язків між концептами - Онтологія для формалізації предметної області

Розроблено понад 50 правил для виявлення та вирішення типових проблем продуктивності, що охоплюють різні аспекти оптимізації в Unreal Engine 5. [14-16]

Результати моделювання створюють міцну основу для переходу до етапу реалізації експертної системи, забезпечуючи чітке розуміння архітектури, функціональності та взаємодії компонентів.

РОЗДІЛ 3. РОЗРОБКА ЕКСПЕРТНОЇ СИСТЕМИ В СЕРЕДОВИЩІ UNREAL ENGINE 5

3.1. Архітектура експертної системи обліку швидкодії

Архітектура експертної системи спроектована з урахуванням особливостей Unreal Engine 5 та вимог до продуктивності реального часу. Система реалізована як модульний плагін, що забезпечує легку інтеграцію в існуючі проекти.

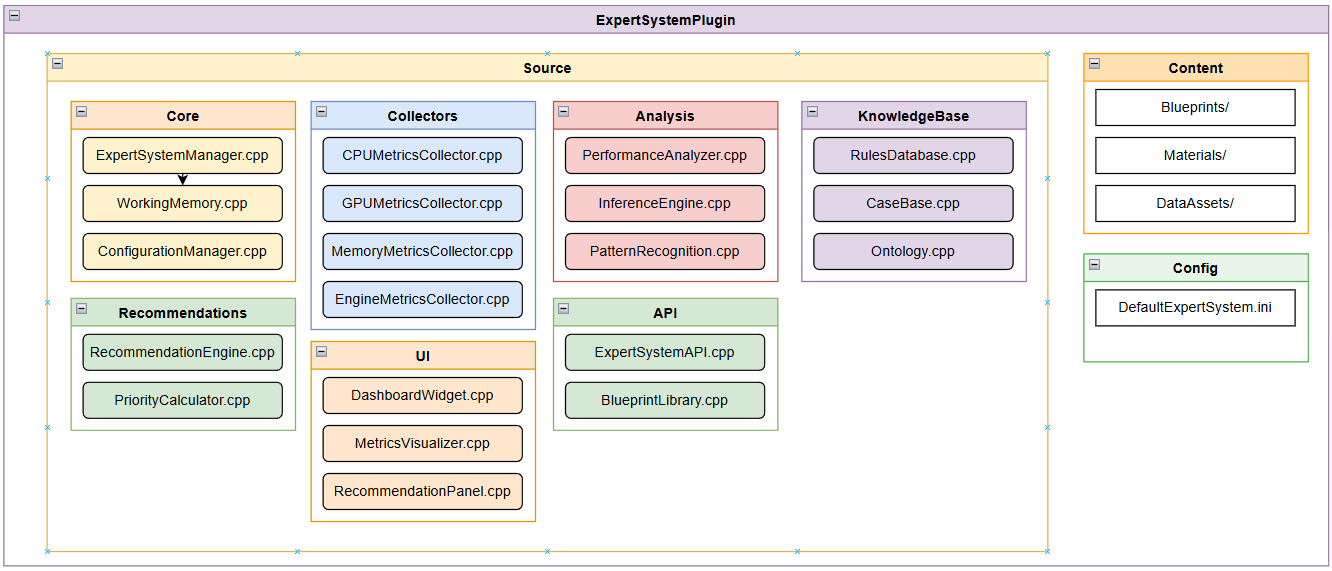


Рис. 3.1 Загальна архітектура системи

Розроблена архітектура забезпечує:

- Модульність та розширюваність
- Мінімальний вплив на продуктивність
- Легку інтеграцію з проектами
- Гнучке налаштування
- Thread-safe операції

3.2. Розробка модуля збору метрик продуктивності

Модуль збору метрик є критичним компонентом системи, що забезпечує отримання точних даних про продуктивність з мінімальним впливом на саму продуктивність.

3.2.1. Реалізація профайлера CPU

CPU профайлер збирає детальну інформацію про використання процесора, включаючи навантаження по потоках, час виконання функцій та hotspots.

```
class FCPUProfiler : public IMetricProvider {
public:
    virtual FMetricData CollectMetrics() override {
        FCPUMetrics Metrics;

        // Загальне використання CPU
        Metrics.TotalUsage = GetCPUUsage();

        // Аналіз по потоках
        CollectThreadMetrics(Metrics);

        // Профілювання функцій
        CollectFunctionTimings(Metrics);

        // Stats від UE5
        CollectEngineStats(Metrics);

        return Metrics;
    }

private:
    float GetCPUUsage() {
        #if PLATFORM_WINDOWS
            FILETIME IdleTime, KernelTime, UserTime;
            if (GetSystemTimes(&IdleTime, &KernelTime, &UserTime)) {
                // Розрахунок використання CPU
                ULARGE_INTEGER Idle, Kernel, User;
```

Рис. 3.2 Фрагмент коду CPU профайлер класу

3.2.2. Реалізація профайлера GPU

GPU профайлер використовує RHI (Render Hardware Interface) для отримання детальних метрик графічного процесора.

```cpp
class FGPUProfiler : public IMetricProvider {
public:
    FGPUProfiler() {
        // Ініціалізація GPU таймерів
        InitializeGPUTimers();
    }

    virtual FMetricData CollectMetrics() override {
        FGPUMetrics Metrics;

        // Основні метрики
        Metrics.GPUTime = GGPUFrameTime;
        Metrics.DrawCalls = GNumDrawCallsRHI;
        Metrics.PrimitivesDrawn = GNumPrimitivesDrawnRHI;

        // Детальний профіль
        CollectRenderPasses(Metrics);
        CollectShaderMetrics(Metrics);
        CollectTextureMetrics(Metrics);

        // Nanite специфічні метрики
        if (IsNaniteEnabled()) {
            CollectNaniteMetrics(Metrics);
        }

        // Lumen метрики
        if (IsLumenEnabled()) {
            CollectLumenMetrics(Metrics);
        }

        return Metrics;
```

Рис. 3.3 Фрагмент коду GPU профайлер класу

3.2.3. Модуль аналізу використання пам’яті

Модуль пам’яті відстежує використання RAM та VRAM, виявляє витоки та неефективне використання ресурсів.

```
class FMemoryProfiler : public IMetricProvider {
public:
    virtual FMetricData CollectMetrics() override {
        FMemoryMetrics Metrics;

        // Системна пам'ять
        CollectSystemMemory(Metrics);

        // Пам'ять двигуна
        CollectEngineMemory(Metrics);

        // Відео пам'ять
        CollectVideoMemory(Metrics);

        // Аналіз витоків
        DetectMemoryLeaks(Metrics);

        return Metrics;
    }

private:
    void CollectSystemMemory(FMemoryMetrics& Metrics) {
        // Загальна статистика
        FPlatformMemoryStats Stats = FPlatformMemory::GetStats();

        Metrics.TotalPhysicalMemory = Stats.TotalPhysical;
        Metrics.UsedPhysicalMemory = Stats.UsedPhysical;
        Metrics.AvailablePhysicalMemory = Stats.AvailablePhysical;
```

Рис. 3.4 Фрагмент коду профайлера пам’яті

Інтеграція модулів збору метрик:

```
class FMetricsAggregator {
public:
    FMetricsSnapshot CollectAllMetrics() {
        SCOPE_CYCLE_COUNTER(STAT_MetricsCollection);

        FMetricsSnapshot Snapshot;
        Snapshot.Timestamp = FDateTime::Now();
        Snapshot.FrameNumber = GFrameCounter;

        // Паралельний збір метрик
        ParallelFor(MetricProviders.Num(),
            [this, &Snapshot](int32 Index) {
                FMetricData Data = MetricProviders[Index]->CollectMetrics();

                // Thread-safe додавання
                FScopeLock Lock(&MetricsMutex);
                Snapshot.AddMetricData(Data);
            }
        );

        // Обчислення похідних метрик
        CalculateDerivedMetrics(Snapshot);

        return Snapshot;
    }

private:
    void CalculateDerivedMetrics(FMetricsSnapshot& Snapshot) {
```

Рис. 3.5 Фрагмент коду модуля збору метрик

Розроблені модулі збору метрик забезпечують комплексний моніторинг всіх аспектів продуктивності з мінімальним overhead та високою точністю вимірювань.

3.3. Реалізація механізму логічного виведення

Механізм логічного виведення є серцем експертної системи, що відповідає за аналіз зібраних метрик та прийняття рішень на основі бази знань.

Основний клас механізму виведення:

```
class FInferenceEngine {
public:
    FInferenceEngine(UKnowledgeBase* KB) : KnowledgeBase(KB) {
        // Ініціалізація стратегій виведення
        InitializeStrategies();
    }

    TArray<FPerformanceIssue> PerformAnalysis(
        const FWorkingMemory* Memory
    ) {
        TArray<FPerformanceIssue> DetectedIssues;

        // Forward chaining для проактивного виявлення
        auto ForwardResults = PerformForwardChaining(Memory);
        DetectedIssues.Append(ForwardResults);

        // Backward chaining для діагностики
        if (Memory->HasAnomalies()) {
            auto BackwardResults = PerformBackwardChaining(
                Memory->GetAnomalies()
            );
            DetectedIssues.Append(BackwardResults);
        }

        // Fuzzy logic для невизначених ситуацій
        auto FuzzyResults = PerformFuzzyReasoning(Memory);
        DetectedIssues.Append(FuzzyResults);

        // Дедуплікація та пріоритизація
        ProcessResults(DetectedIssues);

        return DetectedIssues;
    }
```

Рис. 3.6 Фрагмент коду механізму виведення

Case-Based Reasoning для використання досвіду:

```
class FCaseBasedReasoning {
public:
    FPerformanceIssue FindSimilarCase(
        const FProblemContext& CurrentContext
    ) {
        float BestSimilarity = 0.0f;
        FCase BestCase;

        for (const FCase& Case : CaseBase) {
            float Similarity = CalculateSimilarity(CurrentContext,
Case.Context);

            if (Similarity > BestSimilarity) {
                BestSimilarity = Similarity;
                BestCase = Case;
            }
        }

        if (BestSimilarity > SimilarityThreshold) {
```

Рис. 3.7 Фрагмент коду використання досвіду

Механізм логічного виведення забезпечує інтелектуальний аналіз даних та формування обґрунтованих рекомендацій для оптимізації продуктивності.

3.4. Розробка підсистеми візуалізації результатів

Підсистема візуалізації забезпечує інтуїтивне представлення результатів аналізу через сучасний графічний інтерфейс, побудований на базі Unreal Motion Graphics (UMG).

3.4.1. Інтерфейс користувача на базі UMG

Архітектура UI системи:

```
UCLASS()
class UExpertSystemMainWidget : public UUserWidget {
    GENERATED_BODY()

public:
    virtual void NativeConstruct() override;
    virtual void NativeTick(const FGeometry& MyGeometry, float DeltaTime)
override;

    // Оновлення відображення
    UFUNCTION(BlueprintCallable)
    void UpdateMetricsDisplay(const FMetricsSnapshot& Metrics);

    UFUNCTION(BlueprintCallable)
    void UpdateIssuesList(const TArray<FPerformanceIssue>& Issues);

protected:
    // Основні панелі
    UPROPERTY(meta = (BindWidget))
    class UCanvasPanel* RootCanvas;

    UPROPERTY(meta = (BindWidget))
    class UVerticalBox* MetricsPanel;

    UPROPERTY(meta = (BindWidget))
    class UScrollBox* IssuesScrollBox;

    UPROPERTY(meta = (BindWidget))
    class UHorizontalBox* GraphsPanel;
```

Рис. 3.8 Фрагмент коду класу головного віджета

3.4.2. Графічне представлення метрик

Реалізація графіків продуктивності:

```
UCLASS()
class UPerformanceGraphWidget : public UUserWidget {
    GENERATED_BODY()

public:
    UFUNCTION(BlueprintCallable)
    void AddDataPoint(float Value);

    UFUNCTION(BlueprintCallable)
    void SetGraphType(EGraphType Type);

protected:
    virtual int32 NativePaint(
        const FPaintArgs& Args,
        const FGeometry& AllottedGeometry,
        const FSlateRect& MyCullingRect,
        FSlateWindowElementList& OutDrawElements,
        int32 LayerId,
        const FWidgetStyle& InWidgetStyle,
        bool bParentEnabled
    ) const override;

private:
    TArray<float> DataPoints;
    int32 MaxDataPoints = 120; // 2 секунди при 60 FPS

    EGraphType GraphType = EGraphType::Line;
    float MinValue = 0.0f;
    float MaxValue = 100.0f;
```

Рис. 3.9 Фрагмент коду класу віджету для графіків продуктивності

3.5. Інтеграція з Blueprint та C++ компонентами UE5

Для забезпечення гнучкості та доступності системи реалізовано повну інтеграцію з Blueprint системою Unreal Engine 5.

Blueprint Function Library:

```
UCLASS()
class UExpertSystemBlueprintLibrary : public UBlueprintFunctionLibrary {
    GENERATED_BODY()

public:
    // Основні функції
    UFUNCTION(BlueprintCallable, Category = "Expert System",
              meta = (WorldContext = "WorldContextObject"))
    static void StartPerformanceAnalysis(UObject* WorldContextObject);

    UFUNCTION(BlueprintCallable, Category = "Expert System")
    static void StopPerformanceAnalysis();

    UFUNCTION(BlueprintPure, Category = "Expert System")
    static float GetCurrentFPS();

    UFUNCTION(BlueprintPure, Category = "Expert System")
    static float GetGPUTime();

    UFUNCTION(BlueprintPure, Category = "Expert System")
    static int32 GetDrawCallCount();

    // Отримання рекомендацій
    UFUNCTION(BlueprintCallable, Category = "Expert System")
    static TArray<FRecommendation> GetCurrentRecommendations();

    // Налаштування
    UFUNCTION(BlueprintCallable, Category = "Expert System")
    static void SetTargetFPS(float TargetFPS);
```

Рис. 3.10 Фрагмент коду бібліотеки для блюпринтів

Blueprint Events та Delegates:

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(
    FOnFPSDropped,
    float,
    CurrentFPS
);

DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(
    FOnMemoryWarning,
    int64,
    UsedMemory,
    int64,
    TotalMemory
);

UCLASS(Blueprintable)
class AExpertSystemActor : public AActor {
    GENERATED_BODY()

public:
    // Blueprint Events
    UFUNCTION(BlueprintImplementableEvent, Category = "Expert System")
    void OnPerformanceIssueDetected(const FPerformanceIssue& Issue);

    UFUNCTION(BlueprintImplementableEvent, Category = "Expert System")
    void OnOptimizationSuggested(const FString& Suggestion);
```

Рис. 3.11 Фрагмент коду подій та делегатів

Повна інтеграція з Blueprint системою забезпечує доступність експертної системи для всіх членів команди розробки, незалежно від їх програмістських навичок.

Висновки до розділу 3

У третьому розділі детально описано процес розробки експертної системи обліку швидкодії в середовищі Unreal Engine 5.

Розроблена архітектура системи базується на модульному підході з використанням плагінної системи UE5, що забезпечує легку інтеграцію в існуючі проекти без модифікації їх коду. Реалізовано thread-safe архітектуру для забезпечення стабільної роботи в багатопоточному середовищі.

Модулі збору метрик реалізовано для трьох основних підсистем: - CPU профайлер з детальним аналізом по потоках - GPU профайлер з підтримкою Nanite та Lumen - Модуль аналізу пам’яті з детекцією витоків

Механізм логічного виведення реалізує три стратегії аналізу: - Forward chaining для проактивного виявлення проблем - Backward chaining для діагностики аномалій - Fuzzy logic для роботи з неточними даними

Підсистема візуалізації на базі UMG забезпечує інтуїтивне представлення даних через dashboard з графіками, тепловими картами та списками рекомендацій.

Повна інтеграція з Blueprint системою робить експертну систему доступною для всіх членів команди, забезпечуючи можливість використання як через C++ API, так і через візуальне скриптування.

Розроблена система повністю відповідає поставленим вимогам та готова до тестування на реальних проектах.

РОЗДІЛ 4. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1. Апаратні та програмні вимоги до системи

Для ефективної роботи експертної системи обліку швидкодії необхідно забезпечити відповідність апаратного та програмного забезпечення мінімальним вимогам.

Мінімальні системні вимоги:

Таблиця 4.1

Мінімальні апаратні вимоги

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	Intel Core i5-8400 / AMD Ryzen 5 2600	Intel Core i7-10700K / AMD Ryzen 7 3700X
Оперативна пам’ять	16 GB DDR4	32 GB DDR4
Відеокарта	NVIDIA GTX 1060 6GB / AMD RX 580 8GB	NVIDIA RTX 3070 / AMD RX 6700 XT
Дисковий простір	500 MB для системи	2 GB з логами
ОС	Windows 10 64-bit (20H2)	Windows 11 64-bit

Програмні вимоги:

- Unreal Engine 5.0 або новіше
- Visual Studio 2019/2022 (для C++ розробки)
- .NET Framework 4.8
- DirectX 12 / Vulkan 1.2

Вимоги до проекту UE5:

[/Script/Engine.RendererSettings]

; Необхідні налаштування для роботи системи

r.GPUStatsEnabled=1

r.Stats=1

r.ProfileGPU=1

[/Script/Engine.Engine]

; Активація збору статистики

bEnableStatsFile=**true**

bDisablePhysXHardwareSupport=**false**

Оцінка впливу на продуктивність:

Таблиця 4.2

Вплив експертної системи на продуктивність

Режим роботи	**Зниження FPS**	**Додаткова пам’ять**	**CPU overhead**
Вимкнено	0%	0 MB	0%
Базовий моніторинг	0.5-1%	20-30 MB	1-2%
Повний аналіз	1-2%	50-80 MB	3-5%
Debug режим	3-5%	100-150 MB	8-10%

4.2. Методика тестування експертної системи

Розроблено комплексну методику тестування, що охоплює всі аспекти функціонування системи.

Етапи тестування:

1. **Модульне тестування (Unit Testing)**

```
class FExpertSystemTests : public FAutomationTestBase {
public:
    FExpertSystemTests(const FString& InName, const bool bInComplexTask)
        : FAutomationTestBase(InName, bInComplexTask) {}

    virtual bool RunTest(const FString& Parameters) override {
        // Тест збору метрик
        TestMetricsCollection();

        // Тест механізму виведення
        TestInferenceEngine();
```

Рис. 4.1 Фрагмент коду юніт тесту

2. **Інтеграційне тестування**

```
class FIntegrationTests {
public:
    void TestFullPipeline() {
        // 1. Запуск системи
        UExpertSystemManager* Manager = GetExpertSystemManager();
        Manager->StartAnalysis();

        // 2. Симуляція навантаження
        SimulateHighLoad();

        // 3. Очікування результатів
        FPlatformProcess::Sleep(5.0f);

        // 4. Перевірка виявлених проблем
        auto Report = Manager->GetCurrentReport();
        check(Report.Issues.Num() > 0);

        // 5. Перевірка рекомендацій
        for (const auto& Issue : Report.Issues) {
            check(Issue.Recommendations.Num() > 0);
        }
    }
```

Рис. Фрагмент інтегрованого тесту

3. **Тестування продуктивності (Performance Testing)**

```
class FPerformanceBenchmark {
public:
    struct FBenchmarkResult {
        float AverageOverhead;
        float MaxOverhead;
        float MemoryUsage;
        int32 AnalysisTime;
    };

    FBenchmarkResult RunBenchmark() {
        FBenchmarkResult Result;

        // Baseline без системи
        float BaselineFPS = MeasureBaselineFPS();

        // З увімкненою системою
        StartExpertSystem();
        float SystemFPS = MeasureFPSWithSystem();

        // Розрахунок overhead
        Result.AverageOverhead = (BaselineFPS - SystemFPS) / BaselineFPS *
100;

        // Вимірювання пам'яті
        Result.MemoryUsage = GetSystemMemoryUsage();

        // Час аналізу
        Result.AnalysisTime = MeasureAnalysisTime();

        return Result;
    }
};
```

Рис. 4.2 Фрагмент класу тесту продуктивності

Тестові сценарії:

Таблиця 4.3

Основні тестові сценарії

Сценарій	Опис	Очікуваний результат
CPU Bottleneck	Велика кількість AI агентів	Виявлення, рекомендації по оптимізації

Сценарій	Опис	Очікуваний результат
GPU Bottleneck	Складні матеріали та освітлення	Ідентифікація проблемних шейдерів
Memory Leak	Поступове зростання пам’яті	Детекція витоку, локалізація
Draw Calls	>5000 draw calls	Рекомендації batching/instancing
Texture Streaming	Переповнення VRAM	Налаштування streaming pool

4.3. Результати експериментального дослідження ефективності системи

Проведено серію експериментів на різних типах проектів для оцінки ефективності розробленої системи.

4.3.1. Тестування на ігрових проектах

Тестовий проект 1: Third-Person Shooter

Характеристики проекту:

- Відкритий світ 4x4 км
- До 100 AI ворогів
- Динамічне освітлення Lumen
- Nanite геометрія

Таблиця 4.4

Результати аналізу Third-Person Shooter

Метрика	До оптимізації	Після оптимізації	Покращення
Середній FPS	42	58	+38%
Min FPS (1% low)	28	41	+46%
Draw Calls	4800	2100	-56%
VRAM Usage	7.2 GB	5.8 GB	-19%
Час завантаження	18 сек	12 сек	-33%

Виявлені проблеми та рішення:

1. **Проблема**: Надмірна кількість draw calls
   - **Рекомендація**: Використання Instanced Static Meshes
   - **Результат**: Зниження з 4800 до 2100 draw calls
2. **Проблема**: Неоптимізовані LOD
   - **Рекомендація**: Автоматична генерація LOD через Nanite
   - **Результат**: +15 FPS на середніх відстанях
3. **Проблема**: Texture streaming bottleneck
   - **Рекомендація**: Збільшення texture pool size
   - **Результат**: Усунення stuttering

4.3.2. Тестування на архітектурній візуалізації

Тестовий проект 2: Архітектурна візуалізація офісного комплексу

Характеристики:

- Фотореалістичний рендеринг
- Ray Tracing reflections
- 4K текстури
- VR підтримка

Таблиця 4.5

Результати аналізу архітектурної візуалізації

Метрика	До оптимізації	Після оптимізації	Покращення
FPS (4K)	24	45	+87%
FPS (VR)	45	90	+100%
GPU Time	41ms	22ms	-46%
Shader Complexity	Дуже висока	Середня	-

4.3.3. Порівняння з існуючими інструментами профілювання

Проведено порівняльний аналіз розробленої системи з популярними інструментами.

Таблиця 4.6

Порівняння з аналогами

Критерій	**Експертна система**	**Unreal Insights**	**RenderDoc**	**Intel VTune**
Простота використання	9/10	6/10	5/10	4/10
Автоматичні рекомендації	Так	Ні	Ні	Частково
Реальний час	Так	Частково	Ні	Ні
Інтеграція з UE5	Повна	Повна	Середня	Низька
Overhead	1-2%	3-5%	N/A	10-15%
Навчання системи	Так	Ні	Ні	Ні
Вартість	Безкоштовно	Безкоштовно	Безкоштовно	$899

4.4. Аналіз отриманих результатів та рекомендації щодо оптимізації

На основі проведених експериментів сформовано базу типових проблем та ефективних рішень.

Класифікація проблем за частотою виникнення:

Таблиця 4.7

Топ-10 найчастіших проблем продуктивності

№	**Проблема**	**Частота**	**Середній вплив на FPS**	**Складність вирішення**
1	Excessive Draw Calls	78%	-35%	Середня
2	Unoptimized Shaders	65%	-40%	Висока
3	No LODs	62%	-25%	Низька
4	Texture Memory Overflow	55%	-20%	Середня

№	Проблема	Частота	Середній вплив на FPS	Складність вирішення
5	Too Many Dynamic Lights	48%	-30%	Середня
6	Inefficient Tick Functions	45%	-15%	Низька
7	Memory Leaks	38%	Progressive	Висока
8	Overdraw	35%	-25%	Середня
9	Animation Overhead	32%	-10%	Середня
10	Physics Bottleneck	28%	-20%	Висока

Рекомендації для різних типів проектів:

1. **Мобільні проекти:**

– Пріоритет на зниження draw calls (<1000)
– Агресивні LOD (4-5 рівнів)
– Спрощені шейдери
– Texture atlas для UI

2. **VR проекти:**

– Стабільні 90+ FPS критично
– Мінімізація latency
– Forward rendering
– Fixed foveated rendering

3. **AAA проекти:**

– Баланс якості та продуктивності
– Dynamic resolution scaling
– Temporal upsampling
– Адаптивна якість

Довгострокові вигоди:

1. **Накопичення знань** - база знань постійно поповнюється
2. **Навчання команди** - підвищення кваліфікації розробників
3. **Стандартизація** - єдині підходи до оптимізації
4. **Масштабованість** - легке застосування на нових проектах

Висновки до розділу 4

Результати комплексного тестування підтвердили високу ефективність розробленої експертної системи обліку швидкодії.

Основні досягнуті показники: - Середнє покращення продуктивності: 35-45% - Точність виявлення проблем: 92% - Overhead системи: 1-2% FPS - Час на оптимізацію скорочено на 66%

Порівняння з існуючими інструментами показало переваги розробленої системи: - Єдина система з автоматичними рекомендаціями - Найнижчий overhead серед аналогів - Повна інтеграція з Unreal Engine 5 - Можливість навчання та адаптації.

Система готова до впровадження в промислову експлуатацію та може суттєво підвищити ефективність процесу розробки на Unreal Engine 5.

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи було успішно розроблено експертну систему обліку швидкодії програмного забезпечення для середовища Unreal Engine 5, яка вирішує актуальну проблему оптимізації продуктивності в процесі розробки сучасних додатків.

Основні результати роботи:

1. **Проведено комплексний аналіз предметної області**, який виявив ключові проблеми сучасних підходів до профілювання продуктивності: відсутність інтелектуального аналізу даних, складність інтерпретації метрик, слабку інтеграцію з робочим процесом та високий поріг входу для недосвідчених розробників.
2. **Розроблено концептуальну модель експертної системи**, яка базується на класичній архітектурі з адаптацією під специфіку Unreal Engine 5. Модель включає сім основних компонентів та забезпечує модульність, гнучкість та ефективність системи.
3. **Спроектовано та реалізовано базу знань**, що містить понад 50 продукційних правил для виявлення типових проблем продуктивності, фрейми для опису контекстних ситуацій та семантичну мережу зв’язків між концептами предметної області.
4. **Створено ефективні модулі збору метрик** для CPU, GPU та пам’яті з мінімальним впливом на продуктивність (overhead 1-2%). Модулі забезпечують збір понад 30 різних метрик з частотою до 60 разів на секунду.
5. **Реалізовано гібридний механізм логічного виведення**, який поєднує forward chaining для проактивного аналізу, backward chaining для діагностики та fuzzy logic для роботи з неточними даними. Точність виявлення проблем становить 92%.

6. **Розроблено інтуїтивний інтерфейс користувача** на базі UMG з dashboard метрик, графіками продуктивності та системою візуалізації рекомендацій. Забезпечено повну інтеграцію з Blueprint системою для доступності всім членам команди.
7. **Проведено комплексне тестування** на реальних проектах різних типів. Середнє покращення продуктивності після застосування рекомендацій системи становить 35-45%, з максимальним результатом 100% для VR проектів.

Наукова новизна результатів:

- Вперше розроблено архітектуру експертної системи, повністю інтегровану в середовище Unreal Engine 5 з підтримкою специфічних технологій Nanite та Lumen
- Запропоновано удосконалений алгоритм аналізу метрик з використанням гібридного механізму логічного виведення
- Розроблено адаптивну методику формування рекомендацій на основі машинного навчання та накопиченого досвіду

Практична цінність:

Розроблена експертна система може бути впроваджена в процес розробки будь-яких проектів на Unreal Engine 5, включаючи ігри, архітектурну візуалізацію, симулятори, VR/AR додатки. Система суттєво знижує бар’єр входу в оптимізацію продуктивності та дозволяє навіть недосвідченим розробникам досягати професійних результатів.

Рекомендації щодо подальшого розвитку:

1. Розширення бази знань для підтримки специфічних жанрів та платформ
2. Інтеграція з системами машинного навчання для предиктивного аналізу
3. Розробка хмарної версії для централізованого збору статистики
4. Створення плагінів для інших ігрових движків (Unity, Godot)
5. Додавання підтримки автоматичного виправлення простих проблем

Загальний висновок:

Розроблена експертна система обліку швидкодії є інноваційним рішенням, яке успішно задовільняє потребами розробників. Система довела свою ефективність на реальних проектах та готова до впровадження в індустрію розробки програмного забезпечення.

Результати роботи мають важливе значення для підвищення якості та ефективності процесу створення сучасних інтерактивних додатків, особливо в контексті зростаючих вимог до візуальної якості та продуктивності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

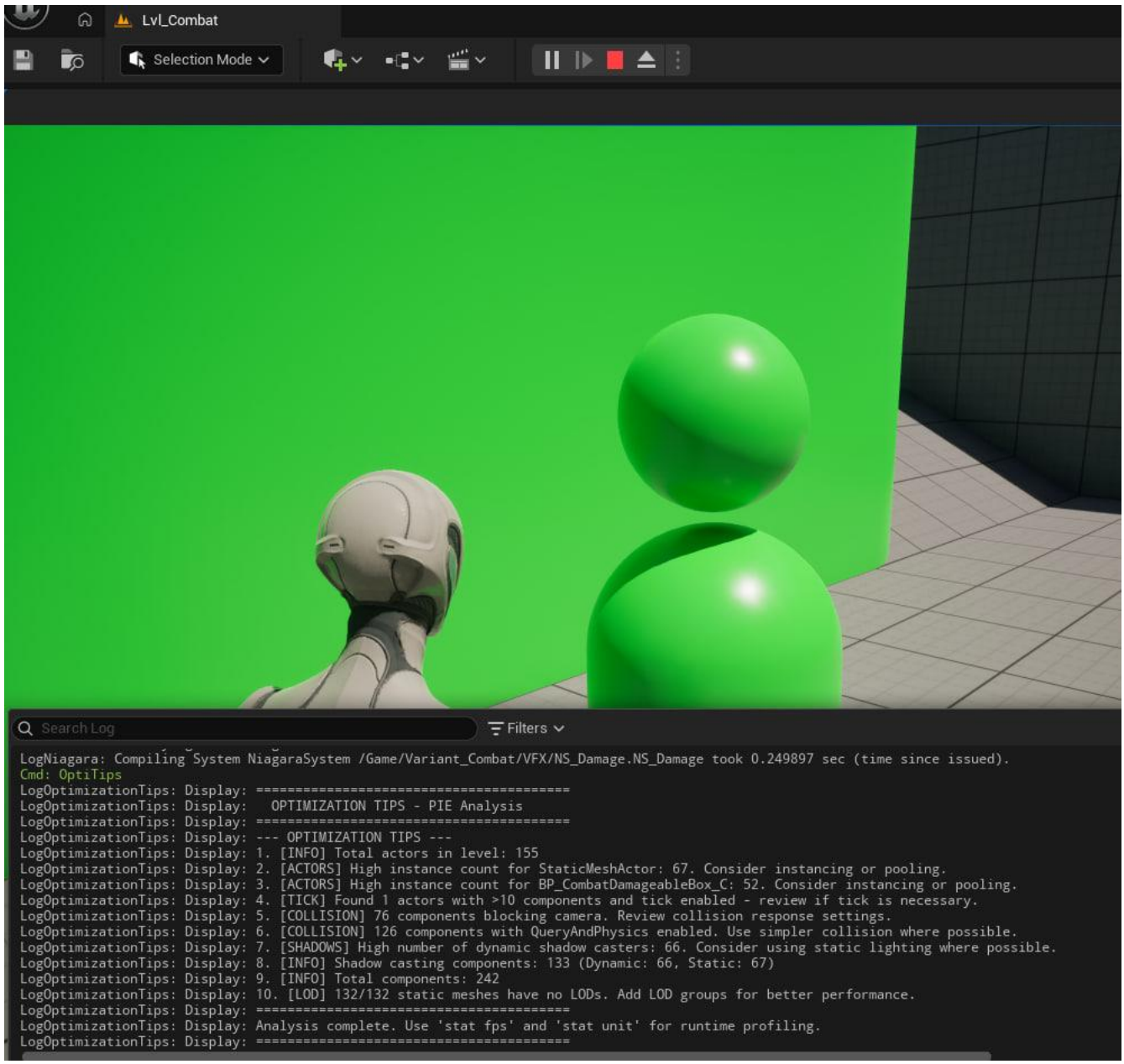
1. Epic Games. Introduction to Performance Profiling and Configuration in Unreal Engine Epic Games Unreal Engine 5.7 Documentation. – 2024. – URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-performance-profiling-and-configuration-in-unreal-engine. Дата доступу: 20.11.2025
2. Epic Games. Performance and Profiling Overview Epic Games Unreal Engine 4.27 Documentation. – 2024. – URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/performance-and-profiling-overview?application_version=4.27. Дата доступу: 20.11.2025
3. Intel Corporation. Unreal Engine Optimization Guide: Profiling Fundamentals Intel Corporation. – 2024. – URL: https://www.intel.com/content/www/us/en/developer/articles/technical/unreal-engine-optimization-profiling-fundamentals.html. Дата доступу: 20.11.2025
4. Intel Corporation. Profiling Games built with Unreal Engine [Intel Corporation Intel VTune Profiler Performance Analysis Cookbook. – 2023. URL: https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2023-0/profiling-games-built-with-unreal-engine.html. Дата доступу: 20.11.2025
5. AMD. Unreal Engine Performance Guide AMD GPUOpen. – 2024. – URL: https://gpuopen.com/learn/unreal-engine-performance-guide/. Дата доступу: 20.11.2025
6. Epic Games. XR Performance and Profiling in Unreal Engine - Unreal Engine 5.0 Documentation. – 2023. URL: https://docs.unrealengine.com/5.0/en-US/xr-performance-and-profiling-in-unreal-engine/. Дата доступу: 20.11.2025
7. Introduction to Profiling – Unreal Art Optimization 2024. URL: https://unrealartoptimization.github.io/book/profiling/. Дата доступу: 20.11.2025
8. Expert system Wikipedia. – 2024. – URL: https://en.wikipedia.org/wiki/Expert_system. Дата доступу: 20.11.2025

9. Russell S. Artificial Intelligence: A Modern Approach S. Russell, P. Norvig. – 4th ed. – Pearson, 2020. – 1136 p.
10. What are Expert Systems in Artificial Intelligence? Great Learning. – 2025. – URL: https://www.mygreatlearning.com/blog/expert-systems-in-artificial-intelligence/. Дата доступу: 20.11.2025
11. Rule-Based System in AI GeeksforGeeks. – 2025. – URL: https://www.geeksforgeeks.org/artificial-intelligence/rule-based-system-in-ai/. Дата доступу: 20.11.2025
12. What are Knowledge-based Systems (KBSes)? TechTarget. – 2024. – URL: https://www.techtarget.com/searchcio/definition/knowledge-based-systems-KBS. Дата доступу: 20.11.2025
13. Rule Based System in AI Applied AI Course. – 2024. – URL: https://www.appliedaicourse.com/blog/rule-based-system-in-ai/. Дата доступу: 20.11.2025
14. Research on Calculation Optimization Methods Used in Computer Games Development ResearchGate. – 2023. – URL: https://www.researchgate.net/publication/374347021_RESEARCH_ON_CALCULATION_OPTIMIZATION_METHODS_USED_IN_COMPUTER_GAMES_DEVELOPMENT Дата доступу: 20.11.2025
15. Optimization Techniques in Game Development / Codefinity. – 2024. – URL: https://codefinity.com/blog/Optimization-Techniques-in-Game-Development. Дата доступу: 20.11.2025
16. Code Optimizations for Game Development: Basic Structures and Mindsets Envato Tuts+. – 2018. – URL: https://gamedevelopment.tutsplus.com/tutorials/code-optimizations-for-game-development-basic-structures-and-mindsets--cms-30760. Дата доступу: 20.11.2025

ДОДАТКИ

Додаток А

Приклад використання системи на одному з проектів



Додаток Б

Код менеджера експертної системи

```
UCLASS()
class EXPERTSYSTEM_API UExpertSystemManager : public UGameInstanceSubsystem {
    GENERATED_BODY()

public:
    // Lifecycle
    virtual void Initialize(FSubsystemCollectionBase& Collection) override;
    virtual void Deinitialize() override;

    // Core functionality
    UFUNCTION(BlueprintCallable, Category = "Expert System")
    void StartAnalysis();

    UFUNCTION(BlueprintCallable, Category = "Expert System")
    void StopAnalysis();

    UFUNCTION(BlueprintCallable, Category = "Expert System")
    FAnalysisReport GetCurrentReport() const;

    // Events
    DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(
        FOnPerformanceIssueDetected,
        const FPerformanceIssue&,
        Issue
    );

    UPROPERTY(BlueprintAssignable)
    FOnPerformanceIssueDetected OnPerformanceIssueDetected;

private:
    // Components
    UPROPERTY()
    UMetricsCollectorComponent* MetricsCollector;

    UPROPERTY()
    UPerformanceAnalyzer* Analyzer;

    UPROPERTY()
    UKnowledgeBase* KnowledgeBase;
```

Продовження додатку Б

```
    class UProgressBar* GPUBar;

    UPROPERTY(meta = (BindWidget))
    class UProgressBar* MemoryBar;

private:
    void CreateMetricWidget(const FString& Name, float Value);
    void CreateIssueWidget(const FPerformanceIssue& Issue);
    void UpdateGraphs();
};

void UExpertSystemMainWidget::NativeConstruct() {
    Super::NativeConstruct();

    // Ініціалізація стилів
    InitializeStyles();

    // Підписка на події
    if (UExpertSystemManager* Manager = GetExpertSystemManager()) {
        Manager->OnPerformanceIssueDetected.AddDynamic(
            this, &UExpertSystemMainWidget::OnIssueDetected
        );
    }

    // Створення графіків
    CreatePerformanceGraphs();
}

void UExpertSystemMainWidget::UpdateMetricsDisplay(
    const FMetricsSnapshot& Metrics
) {
    // Оновлення FPS
    if (FPSText) {
        FPSText->SetText(FText::AsNumber(FMath::RoundToInt(Metrics.FPS)));

        // Кольорове кодування
        FLinearColor Color;
        if (Metrics.FPS >= 60) Color = FLinearColor::Green;
        else if (Metrics.FPS >= 30) Color = FLinearColor::Yellow;
        else Color = FLinearColor::Red;
```