Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

КОМПЛЕКСНА МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інтелектуальний генератор навчальних середовищ і сценаріїв для симуляторів у Unreal Engine. Частина 1. Система генерації тренувальних сценаріїв.»

Виконала:
магістрантка групи 2КІ-24м
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Колесніченко Л.А.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ОТ
Черняк О.І.
(прізвище та ініціали)
«12» 12 2025 р.

Опонент: к.т.н. доц. каф. ПЗ
Ракитянська Г.Б.
(прізвище та ініціали)
«12» 12 2025 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
17. 12. 2025 року

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень магістр

Галузь знань — 12 «Інформаційні технології»

Спеціальність — 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
25.09.2025 року

З А В Д А Н Н Я
НА МАГІСТЕСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенці Колесніченко Лілії Андріївні

1 Тема роботи: «Інтелектуальний генератор навчальних середовищ і сценаріїв для симуляторів у Unreal Engine. Частина 1. Система генерації тренувальних сценаріїв», керівник роботи Черняк О.І. к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «24» вересня 2025 року № 313.

2 Строк подання студентом роботи 04.12.2025 р.

3 Вихідні дані до роботи: актуальність, аналіз аналогів, розробка додатка, структурна схема, алгоритм роботи, тестування.

4 Зміст пояснювальної записки: вступ, аналіз технологій інтелектуальної генерації тренувальних сценаріїв, розробка адаптивного модуля генерації сценаріїв, програмна реалізація системи, тестування програмного засобу, висновки, перелік джерел, додатки.

5 Перелік графічного матеріалу: UML-діаграма системи.

6 Консультанти розділів роботи представлені у таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 – 4	к.т.н., доц. каф. ОТ Черняк О.І.		
5	к.т.н., доц. каф. ЕПВМ Адлер О.О.		
Нормо Контроль	асист. каф. ОТ Швець С.І.		

7 Дата видачі завдання 25.09.2025 р.

8 Календарний план виконання ККР наведений у таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання		Примітка
1.	Постановка задачі та визначення мети дослідження	25.09.2025	27.09.2025	вик
2.	Огляд існуючих методів генерації сценаріїв	28.09.2025	02.10.2025	вик
3.	Розробка архітектури системи генерації сценаріїв	03.10.2025	06.10.2025	вик
4.	Реалізація ядра системи	07.10.2025	10.10.2025	вик
5.	Створення модулів розміщення об'єктів та адаптивності	11.10.2025	17.10.2025	вик
6.	Розробка підсистеми збору даних і моніторингу	18.10.2025	25.10.2025	вик
7.	Тестування та валідація сценаріїв	26.10.2025	25.10.2025	вик
8.	Розрахунок економічної частини	10.11.2025	31.10.2025	вик
9.	Оформлення пояснювальної записки та додатків	01.11.2025	05.11.2025	вик
10.	Попередній захист	12.11.2025	12.11.2025	вик
11.	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	13.11.2025	15.11.2025	вик

Студентка Колесніченко Л.А.

Керівник роботи Черняк О.І.

АНОТАЦІЯ

УДК 004.4

Колесніченко Л.А. Інтелектуальний генератор навчальних середовищ і сценаріїв для симуляторів у Unreal Engine. Магістерська комплексна робота зі спеціальності 123 — Комп'ютерна інженерія, освітня програма — Комп'ютерна інженерія. Вінниця: ВНТУ, 2025. — 126 с.

На укр. мові. Бібліогр.: 40 назв, рис.: 25; табл.: 15.

У магістерській кваліфікаційній роботі розглянуто підходи до побудови інтелектуальних навчальних середовищ і тренувальних сценаріїв для симуляторів у середовищі Unreal Engine. Розроблено систему генерації тренувальних сценаріїв, що функціонує на основі адаптивної логіки та аналізу дій користувача. Виконано проєктування архітектури програмного модуля та реалізацію основних механізмів формування сценаріїв, орієнтованих на динамічну зміну умов навчання.

Ключові слова: інтелектуальна генерація сценаріїв, навчальне середовище, симулятор, Unreal Engine, C++, адаптивна складність.

ABSTRACT

UDC 004.4

Kolesnichenko L.A. Intelligent Generator of Training Environments and Scenarios for Simulators in Unreal Engine. Master's Comprehensive Thesis in Specialty 123 — Computer Engineering, Educational Program — Computer Engineering. Vinnytsia: Vinnytsia National Technical University, 2025. — 126 p.

In Ukrainian. Bibliography: 40 sources; figures: 25; tables: 15.

The master's qualification thesis examines approaches to the development of intelligent training environments and training scenarios for simulators based on the Unreal Engine. A system for generating training scenarios has been developed, operating on adaptive logic and analysis of user actions. The architecture of the software module has been designed, and the core mechanisms for scenario formation oriented toward dynamic modification of training conditions have been implemented.

Keywords: intelligent scenario generation, training environment, simulator, Unreal Engine, C++, adaptive difficulty.

ЗМІСТ

ВСТУП

Актуальність зумовлена зростанням ролі штучного інтелекту та процедурної генерації у створенні навчальних симуляторів. Попри технічні можливості рушія Unreal Engine, процес формування сценаріїв у більшості систем залишається ручним і малоефективним. Це обмежує адаптивність, варіативність і персоналізацію навчання.

Тому удосконалення процесу створення тренувальних сценаріїв через впровадження інтелектуальної генерації, що динамічно підлаштовується під рівень користувача та умови середовища, є актуальним напрямом розвитку сучасних освітніх технологій.

Метою магістерської кваліфікаційної роботи є удосконалення процесу розроблення навчальних сценаріїв у симуляційних середовищах шляхом створення системи інтелектуальної генерації сценаріїв у Unreal Engine. Система має забезпечити автоматизоване формування та адаптацію тренувальних сценаріїв відповідно до умов середовища і рівня підготовки користувача, що підвищує ефективність, гнучкість та реалістичність навчального процесу.

Для досягнення мети роботи необхідно вирішити такі завдання:

— здійснити комплексний аналіз сучасних методів інтелектуальної генерації навчальних сценаріїв та систем адаптивного тренування;

— розробити і теоретично обґрунтувати алгоритм інтелектуальної генерації тренувальних сценаріїв у контексті моделі адаптивної складності;

— спроєктувати архітектуру програмного комплексу, визначити структуру й функціонування основних модулів, інтегрувавши рішення з можливостями Unreal Engine та мовою програмування C++;

— реалізувати підсистему збору та аналітичної обробки даних тренувального процесу для забезпечення якісного зворотного зв’язку із системою генерації;

здійснити експериментальні дослідження з метою перевірки адекватності, гнучкості та ефективності формованих тренувальних сценаріїв;

— підготувати техніко-економічне обґрунтування запропонованих рішень, визначивши їхню економічну та практичну доцільність для впровадження у навчальні симулятори.

Об'єкт дослідження є процес створення навчальних тренувальних сценаріїв у симуляційних середовищах.

Предмет дослідження є методи, алгоритми та архітектурні рішення системи інтелектуальної генерації сценаріїв у Unreal Engine.

Методи дослідження охоплюють системний аналіз існуючих моделей генерації контенту, методи комп’ютерного моделювання навчальних процесів, алгоритми процедурної та інтелектуальної генерації сценаріїв, а також об’єктно-орієнтоване програмування мовою C++ у поєднанні з інструментами Unreal Engine. Для перевірки ефективності розробленої системи застосовувалися методи експериментальної валідації, порівняльного аналізу та статистичної оцінки результатів.

Наукова новизна роботи полягає в удосконаленні процесу автоматизованого формування навчальних сценаріїв у симуляційних системах через розроблення концептуальної архітектури та алгоритмів інтелектуальної генерації у середовищі Unreal Engine. Запропоноване удосконалення полягає у створенні механізму динамічної побудови сценаріїв, який забезпечує їх логічну послідовність, адаптивну зміну структури та складності залежно від рівня компетентності користувача, а також інтеграцію з підсистемами аналітики й телеметрії для формування персоналізованого навчального досвіду.

Практичне значення одержаних результатів полягає у розробці діючого прототипу системи генерації тренувальних сценаріїв, який може бути інтегрований до різних типів симуляторів у сфері професійної підготовки – від військових і медичних до промислових або аварійно-рятувальних. Реалізовані принципи можуть бути основою для подальшого розвитку інтелектуальних навчальних середовищ нового покоління.

Апробація результатів роботи здійснена у доповіді на Молодь в науці: дослідження, проблеми, перспективи (МН-2026).

Матеріали роботи доповідались та опубліковувались [1]:

Публікація Чорний В.В., Колесніченко Л.А. , Черняк О.І. «Інтелектуальний генератор навчальних середовищ і сценаріїв для симуляторв у Unreal Engine». Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)», Вінниця, 2025. [Електронний ресурс]. — Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26513

1 АНАЛІТИЧНИЙ ОГЛЯД ПІДХОДІВ ДО ПОБУДОВИ НАВЧАЛЬНИХ СИМУЛЯТОРІВ В UNREAL ENGINE

1.1 Сучасні підходи до побудови тренувальних середовищ у UE

Навчальні симулятори на Unreal Engine активно застосовуються у професійній підготовці завдяки високому графічному реалізму, фізичній достовірності та можливості інтеграції сучасних алгоритмів ШІ [2]. Відкрита архітектура рушія та підтримка складних динамічних середовищ робить UE придатним для моделювання процесів, де важливе реалістичне відтворення простору, руху та поведінки об’єктів.

UE використовується у військовій та цивільній безпеці, медицині, робототехніці, промисловості та транспорті. Прикладом є симулятор HEROES, який зображений на рисунку 1.1, що моделює умови надзвичайних ситуацій: руйнування, дим, обмежену видимість, шумові перешкоди та інші фактори стресу [3]. Таке середовище дозволяє опрацьовувати як технічні навички, так і психологічну стійкість операторів — критичний елемент підготовки персоналу, що працює під тиском.

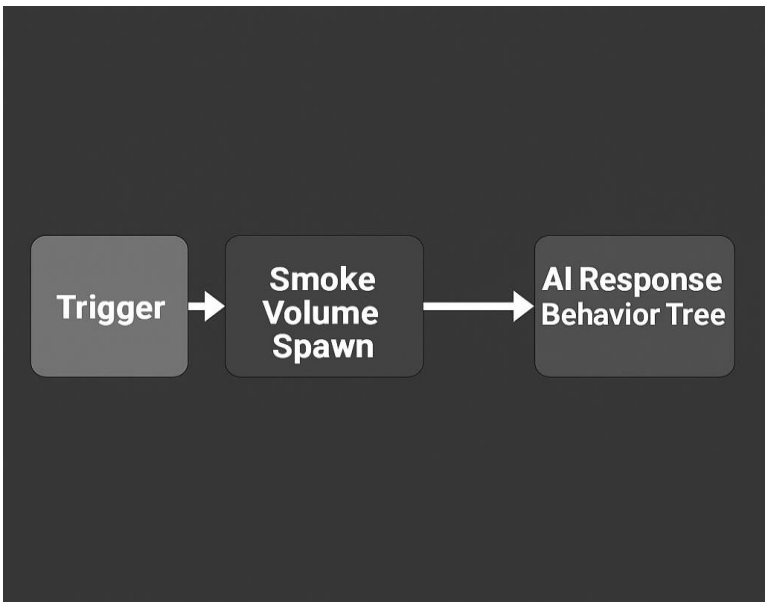

Рисунок 1.1 — Приклад логіки сценарію тренування у симуляторі HEROES

У сфері робототехніки рушій застосовується для симуляції автономних систем. Платформи UNav-Sim та Unreal-MAP використовуються для моделювання сенсорних даних, траєкторій руху, динамічних об’єктів та мультиагентних взаємодій. Такі симуляції дозволяють тренувати як роботів, так і операторів, включаючи безпілотні апарати та наземні роботизовані системи, у складних або непередбачуваних умовах.

У медицині UE використовується для створення симуляторів взаємодії з анатомічними моделями, відпрацювання діагностики та хірургічних процедур. Інтеграція VR/AR-шоломів, тактильних сенсорів і трекерів руху дозволяє формувати повноцінне занурення у навчальний процес.

Однією з ключових переваг UE є підтримка процедурної генерації середовищ. Системи UnrealZoo та RESenv демонструють можливість автоматичного створення варіативних локацій — від урбаністичних просторів до зон стихійних лих. Це забезпечує масштабованість та різноманітність навчальних сценаріїв.

Важливою тенденцією є розвиток мультиагентних симуляцій, де агенти можуть взаємодіяти, координувати дії чи змагатися між собою. UE забезпечує інструментарій для побудови таких систем: Chaos Physics, Lumen, Behavior Trees, EQS, State Machines .

Окреме значення має адаптивність навчальних сценаріїв — можливість автоматичного підлаштування складності, поведінки NPC чи параметрів оточення до рівня користувача. Для цього використовують Blueprint-скрипти, C++ API, а також інтеграції з Python чи RL-фреймворками.

1.1.1 Поділ симуляцій за принципами їх практичного призначення

Симуляційні платформи на базі Unreal Engine можна класифікувати за призначенням, рівнем динамічності та структурою сценарію. За сферою застосування виділяють технічні, медичні, аварійно-рятувальні та військові симулятори. Технічні системи використовують для інженерної підготовки та

робототехнічних досліджень — прикладом є Orbit, платформа для навчання роботів у віртуальних середовищах.

Медичні симулятори зосереджені на моделюванні анатомічних структур, поведінки тканин та ситуацій невідкладної допомоги, де важливими є тактильний зворотний зв’язок та точність відтворення фізіологічних реакцій [3]. У сфері безпеки та рятувальних операцій системи на кшталт HEROES дозволяють тренувати операторів у стресових умовах, що поєднують дим, руйнування та шумові перешкоди [4].

Окремий напрям — мультиагентні симуляції, де одночасно взаємодіють кілька агентів або користувачів. Дослідження Unreal-MAP демонструють потенціал UE у моделюванні кооперативних місій, колективної евакуації та узгоджених дій у середовищах з обмеженими ресурсами.

З погляду структури сценарію виділяють:

— лінійні середовища, у яких послідовність подій фіксована;

— нелінійні середовища, що дозволяють варіативність дій та альтернативні шляхи виконання;

— процедурно-згенеровані середовища, де контент створюється автоматично або через алгоритми ШІ, як у UnrealZoo.

Класифікація симуляторів дозволяє визначити оптимальний рівень деталізації, тип генерації контенту та засоби підвищення інтерактивності. Для розробок на Unreal Engine така систематизація є основою для побудови інтелектуальних систем генерації сценаріїв, що адаптуються до користувача та забезпечують ефективне навчання.

1.1.2 Структура навчальних процесів у віртуальних тренажерах

Організація навчання у симуляторах Unreal Engine ґрунтується на сценарному підході, де навчальний зміст визначається послідовністю подій, умов та реакцій системи. Сценарій виступає центральною одиницею навчального процесу, поєднуючи педагогічні цілі та технічні параметри. Сучасні навчальні

платформи реалізують інтелектуальну генерацію — автоматичне створення та модифікацію завдань відповідно до дій користувача.

Типова структура сценарію включає етапи ініціалізації, виконання, оцінки та адаптації. Початкові умови задаються за допомогою Behavior Trees, Blueprint-логіки та EQS, що дозволяє формувати складні сценарні послідовності без ручного програмування [5]. Після цього система фіксує дії користувача, аналізує їх і визначає подальший розвиток подій.

Ключовою особливістю сучасних сценаріїв є їх динамічність: вони змінюються відповідно до компетентності користувача, можуть ускладнювати умови, додавати непередбачені події або скорочувати обсяг підказок. Механізм адаптації спирається на показники телеметрії — час реакції, точність виконання дій, кількість помилок і успішно виконаних етапів. Для формального опису адаптації складності використовується модель оновлення параметра складності:

$$D_{t+1} = D_t + \alpha \cdot (E_{target} - E_{user}),$$

де D_t — поточний рівень складності,

D_{t+1} — оновлений рівень,

E_{user} — фактичний результат користувача,

E_{target} — цільовий результат,

α — коефіцієнт швидкості адаптації.

Ця формула дозволяє системі автоматично регулювати навчальне навантаження, забезпечуючи індивідуальний темп освоєння матеріалу.

Значну роль відіграють агентні моделі: NPC реагують на дії користувача, генерують події та беруть участь у навчальному процесі як інструктори, супервізори або динамічні перешкоди [6]. Їхня поведінка формується через Behavior Trees і Blackboard-системи.

Сценарії будуються з модульних шаблонів, що дозволяє поєднувати різні частини навчальних ситуацій і генерувати нові варіації без участі інструктора.

Такий принцип реалізовано в ML-SceGen та Unreal-MAP, де сценарії процедуруються на основі навчальних цілей і параметрів середовища.

Завершальним компонентом є система моніторингу. UE підтримує вбудовані інструменти збору телеметрії та інтеграцію з аналітичними платформами, що формують замкнений цикл «користувач → телеметрія → аналіз → адаптація». Такий підхід забезпечує персоналізований і результативний навчальний процес.

1.1.3 Відмінності між сценарними та покроковими тренуваннями

Порівняння симуляційного та сценарного навчання в середовищі Unreal Engine дозволяє визначити оптимальні умови для побудови ефективних тренувальних систем. Обидві парадигми створюють контрольоване навчальне середовище, однак різняться структурою подій, рівнем свободи користувача та педагогічною орієнтацією.

Симуляційні системи акцентують увагу на фізичній достовірності середовища. Їх мета — забезпечити відтворення процесів, максимально наближених до реальних умов, що сприяє формуванню практичних навичок. Unreal Engine надає відповідний інструментарій через Chaos Physics, Niagara, Lumen та Nanite. У проєкті HEROES [7] симуляційний підхід дозволяє моделювати складні урбаністичні зони, відтворюючи дим, руйнування та поведінку NPC.

Натомість сценарне навчання орієнтоване на логіку подій, педагогічні цілі та оцінку рішень користувача, а не лише на відпрацювання окремих дій. У таких системах ключову роль відіграє контекст виконання завдань, послідовність прийняття рішень і здатність користувача адаптуватися до змінних умов. Наприклад, платформа Scenario-Based Mixed Reality Platform моделює кризові ситуації у сфері охорони здоров’я з акцентом на аналізі рішень, швидкості реагування та координації дій між учасниками. Оцінювання результатів у подібних середовищах здійснюється не лише за фактом досягнення мети, а й за якістю обраної стратегії, дотриманням протоколів і ефективністю взаємодії, що робить сценарне навчання особливо придатним для підготовки фахівців у критично важливих галузях.

Для систематизації ключових характеристик підходів у тексті наведено порівняльний аналіз на таблиці 1.1, який узагальнює переваги та обмеження кожної моделі в контексті освітніх тренажерів.

Таблиця 1.1 — Порівняльний аналіз

Критерій	Симуляційне навчання	Сценарне навчання
Мета	Відпрацювання практичних і технічних навичок у реалістичних умовах	Розвиток когнітивних і управлінських компетенцій, прийняття рішень
Структура	Відкрита, багатофакторна, часто без фіксованого кінця	Чітко структурована, із заздалегідь визначеною логікою подій
Контроль користувача	Високий рівень свободи, експериментування	Обмежений вибір, орієнтація на оцінку правильності дій
Реалістичність	Максимальна фізична та візуальна достовірність	Може бути умовною або стилизованою для педагогічного ефекту
Зворотний зв’язок	Переважно технічний і сенсорний (поведінка системи, результати дій)	Аналітичний, когнітивний, із поясненнями помилок
Тип середовища	Відкрите, змінне, процедурне	Сюжетно орієнтоване, контрольоване
Приклади реалізацій	HEROES, RESenv, UnrealZoo	Scenario-Based Mixed Reality, Simcrafting Framework

Сучасні симулятори на Unreal Engine дедалі частіше поєднують обидва підходи, формуючи гібридні системи. Такий підхід включає три рівні:

— симуляційне середовище, що забезпечує фізичну достовірність;

— сценарну логіку, яка визначає мету, умови та послідовність дій;

— систему аналізу результатів, яка оцінює дії користувача та формує адаптивний зворотний зв’язок.

Для наочності на рисунку 1.2 наведено структурне узагальнення гібридного підходу.

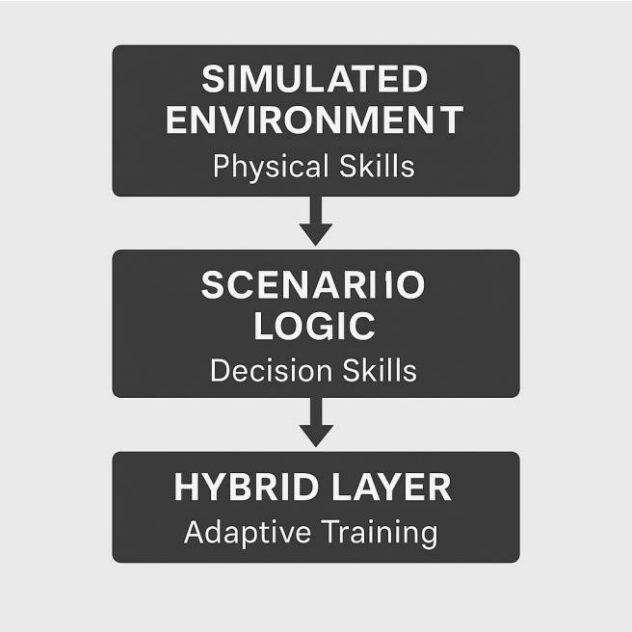

Рисунок 1.2 — Структурне порівняння симуляційного та сценарного навчання

Комбінація цих рівнів забезпечує адаптивність навчальних процесів, де складність, умови середовища або кількість факторів можуть автоматично підлаштовуватися під рівень користувача. Прикладом такої концепції є Simcrafting Framework [8], який реалізує автоматичне формування сюжетних ліній та оцінювання навчальної ефективності.

1.2 Теоретична база формування динамічних тренувальних процесів

Методологія генерації тренувальних сценаріїв у Unreal Engine ґрунтується на поєднанні процедурних, інтелектуальних та аналітичних підходів, які забезпечують створення динамічного навчального контенту, здатного змінюватися відповідно до дій користувача. На відміну від статичних тренажерів, сучасні системи використовують штучний інтелект, поведінкову аналітику й машинне навчання для формування сценаріїв, що еволюціонують у реальному часі.

Одним із ключових завдань методології є формування самоадаптивної моделі контенту, у якій сценарій розглядається як динамічна структура зі змінними параметрами — часовими, поведінковими, просторовими та контекстними. Наприклад, при зниженні точності дій система може зменшувати інтенсивність

подій, а при високій ефективності — ускладнювати середовище або збільшувати кількість факторів взаємодії [9].

Для узагальнення процесу адаптивної генерації у сучасних системах на Unreal Engine на рисунку 1.3 подано структурну схему, що демонструє послідовність етапів перетворення користувацьких метрик у оновлений сценарій.

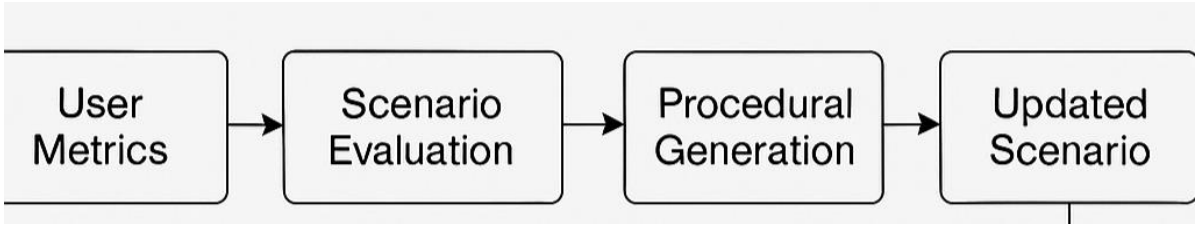

Рисунок 1.3 — Узагальнена схема процесу адаптивної генерації сценарію

Схема ілюструє, що адаптивна поведінка сценарію базується на аналізі телеметричних даних:

$$M = \{t_r, a_c, e, p\},$$

де t_r — час реакції користувача,

a_c — точність виконання дій,

e — кількість помилок,

p — успішність проходження етапів.

На основі агрегованих метрик визначається динаміка зміни складності:

$$D_{t+1} = D_t + \beta + f(M),$$

де D_t— поточний рівень складності,

$f(M)$ — функція оцінки продуктивності,

β — коефіцієнт адаптації.

У межах Unreal Engine адаптивна методологія реалізується через інтеграцію процедурної генерації та поведінкових моделей, що включають PCG Framework, EQS, Behavior Trees, Blueprint AI Systems та аналітичні модулі. Така інтеграція

формує гнучку архітектуру, у якій сценарії виникають у результаті взаємодії користувача з алгоритмами штучного інтелекту, забезпечуючи інтелектуальний розвиток навчального середовища.

1.2.1 Принципи побудови автоматизованих сценарних рішень

Інтелектуальна генерація навчальних сценаріїв у середовищі Unreal Engine базується на поєднанні машинно-обчислювальних, поведінкових і когнітивно-аналітичних методів. Кожен із них формує власну логіку побудови навчального процесу та адаптації до користувача, а їх синтез забезпечує багаторівневу структуру сценаріїв [10].

Першу групу становлять методи машинного навчання (ML) та підкріпленого навчання (RL), які використовують телеметричні дані для прогнозування складності, підбору послідовностей завдань та автоматичної модифікації подій. Фреймворки ML-SceGen і SimWorld демонструють можливість навчання генератора сценаріїв на історичних даних, що робить сценарій динамічно змінним відповідно до продуктивності користувача.

Друга група — методи великих мовних моделей (LLM), які застосовуються для побудови логічних, текстових і комунікативних елементів сценарію. LLM можуть генерувати опис місії, інструкції, діалоги NPC та контекстні підказки, а також адаптувати формулювання завдань у реальному часі через REST або Python API. Це особливо корисно у випадках, коли система фіксує низькі показники ефективності й потребує надати користувачу додаткові пояснення [11].

Третю групу становлять графові методи та агентна логіка. У GraphSCENE сценарії будується на основі графових структур, де вузли відповідають подіям, а ребра — можливим переходам між станами. Такий підхід забезпечує нелінійність проходження та високу варіативність сценаріїв. Агентна логіка, у свою чергу, визначає поведінку NPC, які реагують на дії користувача та формують динамічну взаємодію.

Для систематизації використаних підходів в таблиці 1.2 наведено порівняльну характеристику методів інтелектуальної генерації, що відображає їхні

сильні сторони та обмеження в контексті побудови адаптивних тренувальних сценаріїв.

Таблиця 1.2 — Порівняльна характеристика основних методів інтелектуальної генерації

Тип методу	Основна ідея	Приклади реалізації
Машинне навчання (ML/RL)	Адаптація сценаріїв на основі поведінкових даних та моделей прогнозування.	ML-SceGen, SimWorld
Великі мовні моделі (LLM)	Формування текстових елементів і підказок за контекстом.	UniGen
Графові алгоритми	Побудова сценарію як графа подій і переходів.	GraphSCENE, UniGen
Агентна логіка (Behavior Trees, EQS)	Моделювання поведінки NPC та їх реакцій.	Unreal Engine AI System, Behavior Tree Framework
Комбіновані моделі (Hybrid AI)	Інтеграція ML, LLM і графів для комплексної генерації.	SimWorld, ML-SceGen

Для візуалізації взаємодії між ML, LLM, графовими структурами та агентною логікою на рисунку 1.4 подано узагальнену схему формування інтелектуального сценарію.

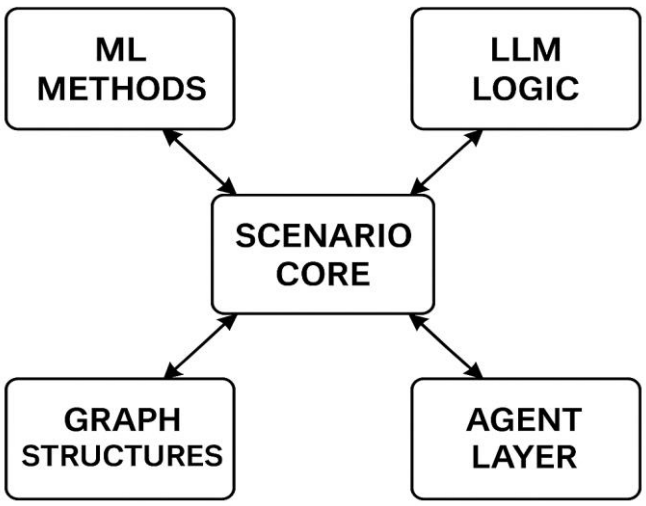

Рисунок 1.4 — Узагальнена модель методів інтелектуальної генерації сценаріїв

Рисунок ілюструє, як ML забезпечує статистичну адаптацію, LLM — когнітивну та контекстну гнучкість, графові структури — просторову організацію сценарію, а агентна логіка — динамічну взаємодію з NPC. Поєднання цих методів формує інтелектуальне ядро генератора сценаріїв, здатне до самооптимізації та підвищення педагогічної ефективності.

1.2.2 Схеми створення змінного контенту у віртуальному просторі

Процедурна та інтерактивна моделі генерації є ключовими підходами до створення навчальних симуляторів у середовищі Unreal Engine. Їх поєднання забезпечує побудову динамічних “живих” навчальних середовищ, що реагують на дії користувача та змінюють умови тренування в реальному часі.

Процедурна генерація (Procedural Content Generation, PCG) формує автоматичне створення простору, об'єктів і сценарних умов на основі параметричних правил та стохастичних алгоритмів [12]. У Unreal Engine це реалізовано через PCG Framework, який генерує ландшафти, структури, NPC, маршрути та тренувальні цілі. Основні переваги PCG:

— автоматизація створення середовища;

— масштабованість — швидке формування нових варіантів сценаріїв;

— реіграбельність — унікальна конфігурація у кожній сесії;

— зменшення трудовитрат на проєктування контенту.

Процедурна модель дозволяє контролювати навчальне середовище через параметри. Формалізовано це можна подати як:

$$S = G(P),$$

де S — сформоване середовище,

S — набір параметрів (ризик, щільність NPC, складність),

G — процедурний генератор.

Зміна одного параметра p_i впливає на весь сценарій:

$$\Delta S = \frac{\partial G}{\partial p_i} \cdot \Delta p_i,$$

Інтерактивна модель базується на ідеї реактивного середовища, яке змінює стан відповідно до дій користувача. Вона покладається на телеметрію, поведінкові модулі (Behavior Trees, EQS), AI-агентів та аналітику користувацьких даних. У практичному застосуванні інтерактивна модель може:

— зменшувати інтенсивність подій при низькій ефективності користувача;

— ускладнювати сценарій при високому рівні виконання;

— змінювати поведінку NPC у реальному часі;

— коригувати логіку залежно від стресових факторів.

Сучасні симулятори поєднують процедуру та інтерактивність, формуючи гібридну архітектуру, у якій:

— PCG створює структурний простір;

— інтерактивна логіка коригує події й поведінку;

— система адаптується до користувача за результатами телеметрії.

Узагальнену структуру цього підходу подано на рисунку 1.5.

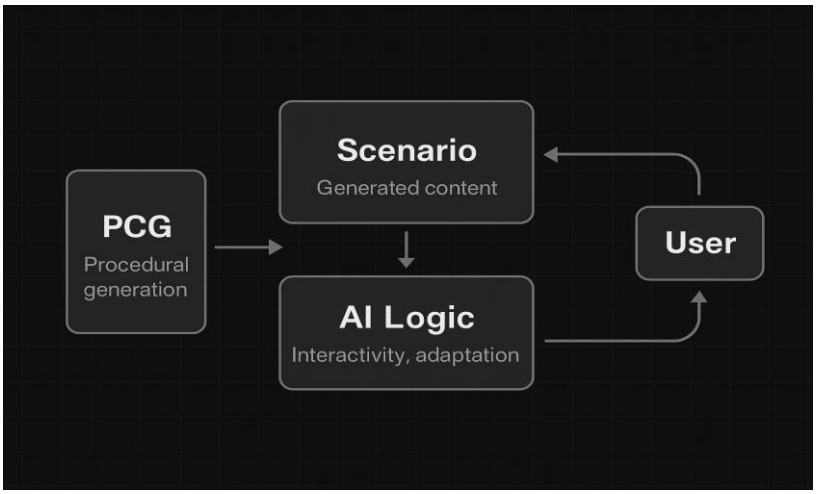

Рисунок 1.5 — Взаємодія процедурної та інтерактивної моделей у генерації контенту.

Для формального представлення поєднання моделей використовується композиція функцій:

$$C = A(S, U),$$

де C — адаптований сценарій,

S — процедурно згенероване середовище,

U — дані користувача,

A — адаптивна функція сценарію.

Таблиця 1.3 — Порівняльна характеристика основних методів інтелектуальної генерації

Характеристика	Процедурна модель	Інтерактивна модель
Мета	Автоматичне створення контенту та оточення	Адаптація середовища й сценарію до користувача
Методи реалізації	PCG Framework, L-системи, стохастичні алгоритми, параметричні шаблони	Телеметрія, Behavior Trees, EQS, Machine Learning, RL
Реакція на користувача	Непряма (через параметри генерації)	Пряма (через аналіз дій у реальному часі)
Тип змін	Просторові: геометрія, об’єкти, розташування	Логічні: завдання, цілі, поведінка агентів
Переваги	Масштабованість, автоматизація, швидкість	Гнучкість, адаптивність, педагогічна ефективність
Обмеження	Відсутність контекстної логіки; потреба у ручному сценарному контролі	Висока обчислювальна складність; потреба в аналітичних даних
Типові приклади	RESenv, UnrealZoo, PCG Framework [30]	Unreal-MAP, HEROES, SimWorld [27]
Результат	Автоматично сформоване навчальне середовище	Адаптивна, динамічно керована навчальна ситуація

Процедурна та інтерактивна моделі є комплементарними. PCG забезпечує структурну основу, а інтерактивна модель — інтелектуальну адаптацію. Використання тільки однієї моделі призводить або до надмірної механічності (PCG), або до перевантаження обчисленнями (інтерактивна). Тому оптимальною є гібридна архітектура, у якій середовище створюється автоматично, а поведінка та складність — змінюються відповідно до користувача.

1.2.3 Критерії застосування системи автоматичного сценароутворення

Створення системи динамічної генерації сценаріїв у середовищі Unreal Engine потребує визначення низки архітектурних, функціональних та педагогічних вимог, що забезпечують адаптивність, стабільність та відповідність навчальним цілям [13].

Однією з ключових вимог є модульність архітектури. Ядро сценарного генератора повинно бути відокремлене від підсистем телеметрії, аналітики, PCG-модуля та інтерфейсу користувача. Це дозволяє конфігурувати або розширювати функціональні блоки без зміни всієї системи, що є критично важливим для симуляторів різних галузей — від медицини до оборонних застосувань. Для узагальнення вимог до структури системи на рисунку 1.6 подано модель взаємодії основних компонентів генерації.

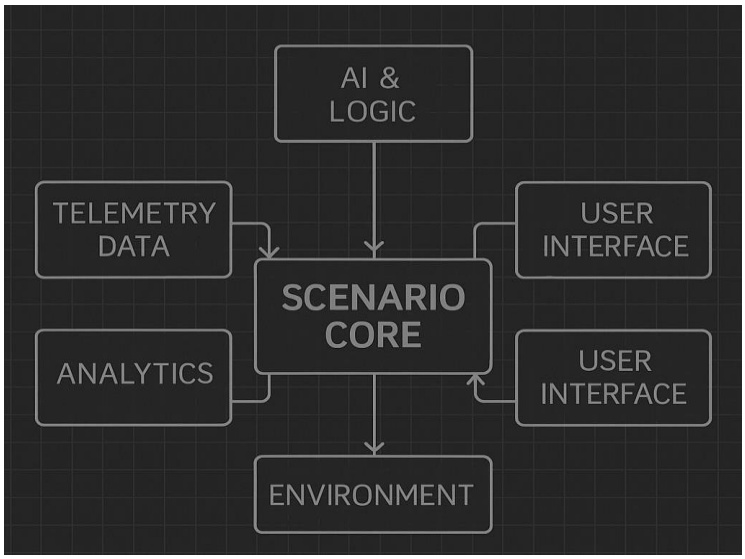

Рисунок 1.6 — Архітектура системи динамічної генерації сценаріїв

Система також повинна забезпечувати масштабованість, тобто здатність працювати з великими сценами та великою кількістю об’єктів без втрати швидкодії. Для цього застосовуються Level Streaming, World Partition, асинхронна генерація контенту та оптимізації Nanite.

Важливим елементом є інтелектуальний аналіз дій користувача. На основі телеметрії — часу реакції, ефективності дій, кількості помилок — система повинна коригувати складність, темп подій та поведінку NPC у реальному часі. Це формує персоналізовану траєкторію навчання.

Система має бути інтегрована з PCG Framework і аналітичними інструментами Unreal Engine, що дозволяє автоматично генерувати об’єкти й події відповідно до параметрів середовища, таких як рівень ризику, погодні умови чи кількість агентів [14]. До вимог належать також реалістичність середовища та варіативність подій, що забезпечують наближеність до реальних професійних умов і підвищують рівень занурення.

Крім цього, генерація сценарію повинна виконуватися без затримок, що потребує оптимізованих алгоритмів і багатопотокового формування контенту. Завершальним компонентом є безпека та контроль якості, які передбачають перевірку логічної коректності сценарію, захист телеметрії та стабільну роботу системи навіть у складних сценарних конфігураціях. Сукупність перелічених вимог формує архітектурну основу інтелектуального генератора сценаріїв, здатного забезпечити автономну, адаптивну та педагогічно ефективну роботу навчальних симуляторів.

1.3 Технічна основа програмного комплексу

Розробка інтелектуальної системи генерації тренувальних сценаріїв потребує вибору рушія, здатного поєднати високу продуктивність, аналітичну інтегрованість, підтримку процедурних алгоритмів та реалізацію складних поведінкових моделей. Вибір технологічної платформи визначає не лише технічні можливості системи, а й рівень реалістичності навчального середовища, швидкість адаптації та педагогічну ефективність.

У цьому контексті Unreal Engine є найбільш доцільним рішенням, оскільки забезпечує комплексну екосистему для процедурної, інтелектуальної та аналітичної складових генератора сценаріїв. Його модульна архітектура відповідає вимогам гнучкості та масштабованості, сформульованим у підрозділі 1.2.3, а також підтримує інтеграцію з AI-системами, VR/AR і PCG Framework [15]. Для формального представлення процесу вибору рушія можна використати узагальнену функцію придатності платформи:

$$F_{engine} = w_p P + w_a A + w_{pcg} G + w_{ai} I + w_s S,$$

де P — продуктивність рушія (рендеринг, оптимізація, багатопоточність),

A — аналітична інтегрованість (телеметрія, логування, API),

G — підтримка процедурної генерації,

I — можливості AI та поведінкових моделей,

S — масштабованість і модульність архітектури,

w_i — вагові коефіцієнти, що відображають важливість кожного параметра у цій роботі.

Оскільки Unreal Engine демонструє високі значення для всіх компонентів (P, A, G, I, S), його узагальнена оцінка F_{UE} суттєво перевищує альтернативні платформи (Unity, Godot, CryEngine), особливо в частині процедурної генерації, AI-моделювання та роботи з великими симуляційними середовищами.

Крім того, Unreal Engine є не лише рушієм для візуалізації, а й повноцінною платформою розробки з відкритим кодом. Це дозволяє створювати власні модулі ШІ, інтегрувати ML-фреймворки, телеметрію та аналітику. Така відкритість дає змогу реалізувати системи будь-якої складності — від VR-тренажерів до мультиагентних наукових симуляцій.

UE має підтверджену ефективність у проєктах наукового та прикладного призначення: від робототехнічних симуляторів до комплексних моделей поведінки у кризових ситуаціях. Дослідження, такі як RESenv та UnrealZoo, демонструють,

що рушій здатен працювати як основа для адаптивних симуляційних систем та автоматичного формування навчальних ситуацій [16].

Таким чином, Unreal Engine забезпечує необхідний рівень точності, адаптивності й технологічної гнучкості для побудови інтелектуального генератора тренувальних сценаріїв, що підтверджує його відповідність вимогам цієї роботи.

1.3.1 Обґрунтування вибору UE як середовища реалізації

Unreal Engine належить до середовищ розробки нового покоління, що орієнтовані на створення реалістичних інтерактивних систем із високою продуктивністю та гнучкою модульною архітектурою. Це робить UE оптимальною платформою для реалізації системи генерації сценаріїв, яка поєднує процедурну побудову контенту, аналітичну обробку телеметрії та автоматичну адаптацію навчальних завдань [17]. Для узагальнення ключових технологічних переваг Unreal Engine на рисунку 1.7 наведено структурну діаграму підсистем, що формують основу генеративної архітектури.

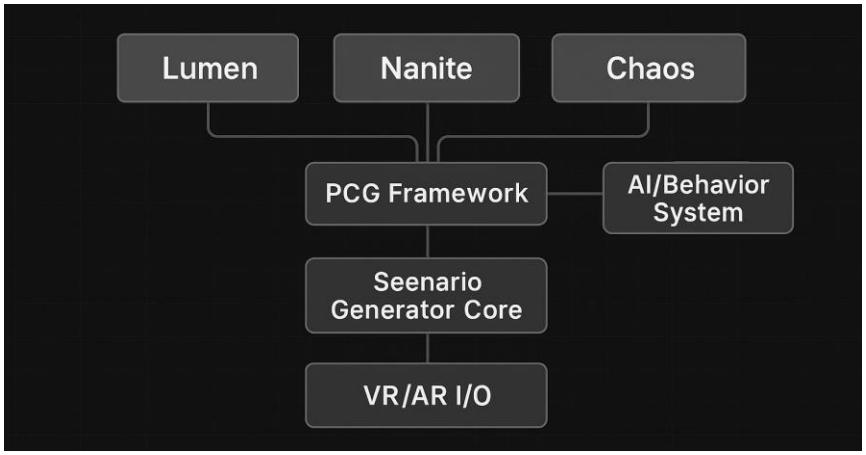

Рисунок 1.7 — Ключові підсистеми Unreal Engine, що забезпечують генеративні можливості

Unreal Engine має низку ключових переваг, що роблять його оптимальною платформою для побудови генеративних навчальних систем. Технології Lumen і Nanite забезпечують фотореалістичне відтворення сцен із глобальним освітленням, коректними відбиттями та високою деталізацією без втрати продуктивності, що

дозволяє моделювати точні навчальні умови — від лабораторного освітлення до складних погодних ефектів. Система Chaos Physics надає можливість точно відтворювати фізичні процеси, включно зі зіткненнями, руйнуваннями, рідинами та робототехнічними взаємодіями, що є критично важливим для медичних, аварійно-рятувальних і технічних симуляторів.

Важливою складовою є вбудована процедурна генерація: PCG Framework дозволяє автоматично формувати структуру тренувального середовища, маршрути NPC та конфігурації об’єктів без ручного втручання, що створює основу для сценарного генератора, здатного формувати контент у реальному часі відповідно до користувацьких метрик [18]. Unreal Engine також має нативну підтримку VR/AR-пристроїв, трекінгу руху, просторового аудіо й хаптичного зворотного зв’язку, що забезпечує можливість створення мультисенсорних навчальних середовищ для медицини, військової підготовки та операторських тренажерів.

Крім цього, рушій вирізняється високою розширюваністю й відкритістю архітектури. Завдяки відкритому вихідному коду, широким можливостям модифікації та інтеграції ML/AI-рішень Unreal Engine є придатним для наукових досліджень і створення адаптивних навчальних систем, у яких необхідний глибокий контроль внутрішніх процесів.

1.3.2 Інструменти та технології, необхідні для розробки системи

У межах розробки системи інтелектуальної генерації тренувальних сценаріїв в Unreal Engine застосовано поєднання C++ та Blueprints як основні інструменти реалізації функціональності. Така комбінація забезпечує баланс між продуктивністю низькорівневого коду та зручністю створення адаптивної сценарної логіки, що особливо важливо для системи, яка потребує як обчислювальної потужності, так і гнучкого редагування без перекомпіляції.

У запропонованій архітектурі C++ використовується як основна мова розробки для ядра системи, відповідального за алгоритми генерації сценаріїв, управління пам’яттю, багатопоточну обробку даних та реалізацію внутрішніх підсистем, таких як AI Core, Scenario Graph Manager та Telemetry Processor. Через

доступ до низькорівневого API Unreal Engine C++ дозволяє створювати кастомні модулі і забезпечує стабільність та продуктивність при роботі з великими симуляційними просторами [19].

Паралельно Blueprints виступають засобом візуальної розробки сценарної логіки. Вони використовуються для побудови подієвих послідовностей, логіки тригерів, поведінки NPC, інтерактивних елементів UI та систем зворотного зв’язку. Blueprints також інтегруються з даними, отриманими з C++-модулів — метриками, аналітичними оцінками та параметрами адаптації складності — що дозволяє здійснювати швидке налаштування сценаріїв без перекомпіляції системи.

Поєднання цих засобів забезпечує двошарову архітектуру: системний шар на C++ формує інтелектуальне ядро генератора, тоді як сценарний шар Blueprints забезпечує гнучкість, інтерактивність і можливість швидких педагогічних або дизайнерських змін. Такий підхід дозволяє об’єднати технічну ефективність високорівневого AI-коду з доступністю інструментів для створення навчальних ситуацій, що робить Unreal Engine оптимальною платформою для побудови адаптивних систем генерації тренувальних сценаріїв.

2 АРХІТЕКТУРА ТА КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ ГЕНЕРАЦІЇ ТРЕНУВАЛЬНИХ СЦЕНАРІЇВ

2.1 Концептуальний поділ системи на функціональні частини

Архітектура системи інтелектуальної генерації тренувальних сценаріїв в Unreal Engine 5 побудована за багаторівневим модульним принципом, що забезпечує адаптивність, масштабованість і можливість автоматичного формування навчального контенту в реальному часі. Загальну логічну модель взаємодії модулів системи наведено на рисунку 2.1, який відображає чотиришарову структуру генеративного ядра та зв’язки між його компонентами [20].

У центрі архітектури знаходиться ядро сценарної генерації (Scenario Generation Core), що формує структуру сценарію, визначає події, складність, педагогічні цілі та просторову композицію середовища. До нього підключаються модулі аналітики, процедурного формування середовища та системи реактивної адаптації користувача. Основні функціональні рівні системи та їх призначення подано в таблиці 2.1, яка узагальнює роль кожного шару в процесі формування та виконання сценарію.

Таблиця 2.1 — Основні структурні компоненти системи

Назва модуля	Основні функції	Реалізація
Scenario Logic Core	Керування сценарною логікою та станами; виклик подій; зв’язок із агентами.	C++
Environment & Event Generator	PCG-генерація оточення, NPC і подій; динамічне масштабування складності.	C++ / Blueprint
Telemetry & Intelligence Module	Аналіз телеметрії, оцінка дій, адаптація складності.	C++
Scenario Configuration Manager	Визначення навчальних цілей, параметрів сценарію та поведінки агентів.	Blueprint
Data Communication & API	API-взаємодія, логування, обмін і експорт даних.	Blueprint / C++

Архітектура поєднує логічний, генераційний, адаптивний та комунікаційний рівні, що наведено на рисунку 2.1. Логічний шар керує станами сценарію та процесами переходів між ними, генеруючи сценарний граф — структуру, що визначає зв’язки між завданнями, об’єктами та агентами. Генераційний шар відповідає за створення динамічного середовища за допомогою PCG-механізмів, які формують геометрію, позиції NPC та інші параметри навчальних умов. Адаптивний рівень аналізує телеметрію користувача — час реакції, точність і частоту помилок — та коригує сценарій відповідно до індивідуальної динаміки виконання. Комунікаційний рівень забезпечує зв’язок із зовнішніми аналітичними інструментами, API та системами моніторингу [19].

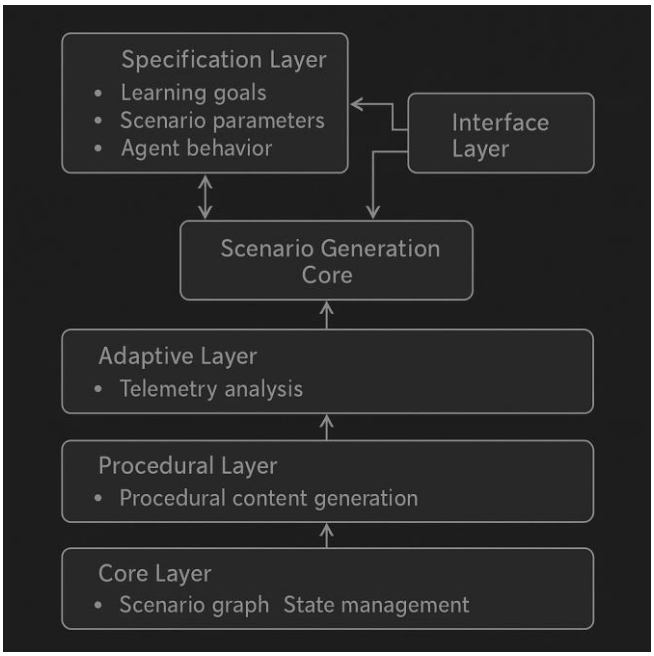

Рисунок 2.1 — Узагальнена архітектура системи генерації тренувальних сценаріїв

Ключовим елементом архітектури є інтеграція C++ та Blueprint-компонентів, що забезпечує баланс між продуктивністю й гнучкістю. Низькорівнева логіка генератора (керування сценарним графом, поведінкою агентів, оптимізацією продуктивності) реалізована на C++, тоді як Blueprint використовується для специфікації поведінки, тригерів та інтерфейсних взаємодій. Завдяки цьому

сценарій може змінюватися у процесі виконання без перекомпіляції системи, що є принциповою вимогою до адаптивних навчальних платформ [28].

2.1.1 Архітектурна модель із розподілом ролей компонентів

Модульна архітектура системи генерації тренувальних сценаріїв в Unreal Engine побудована за принципом розподілу автономних компонентів, що виконують окремі функції, але працюють як узгоджена структура. Такий підхід забезпечує масштабованість, простоту супроводу й можливість незалежного оновлення кожного модуля без втручання в роботу всієї системи [21]. Узагальнену характеристику ключових модулів подано в таблиці 2.2.

Таблиця 2.2 — Взаємозв’язки модулів

Модуль	Взаємодіє з	Призначення взаємодії
Генерація сценаріїв	Оточення, Поведінка, Аналітика	Формування й адаптація сценарію відповідно до стану середовища та результатів користувача
Оточення	Генерація сценаріїв, Візуалізація	Оновлення візуального контенту й параметрів простору
Поведінка агентів	Генерація сценаріїв, Аналітика	Зміна поведінки NPC і складності сценарію на основі телеметрії
Аналітика	Генерація сценаріїв, Візуалізація	Аналіз дій користувача, передача даних для корекції складності
Інтерфейс / Візуалізація	Усі модулі	Відображення стану системи, навчальних цілей і результатів

Центральним елементом виступає Scenario Generation Module, який визначає логіку навчального сценарію, параметри складності та послідовність подій. Він

координує взаємодію між агентами, середовищем і подієвими станами, формуючи повну структуру тренувального процесу [22].

Environment Management Module відповідає за процедурне створення та оновлення середовища. Використовуючи PCG Framework, модуль генерує ландшафт, об’єкти, NPC та умови освітлення, забезпечуючи варіативність і реалістичність симуляцій.

Agent & User Behavior Module реалізує поведінкову логіку агентів через Behavior Trees та EQS. Модуль також адаптує складність, реагуючи на дії користувача, рівень точності та темп виконання завдань [23].

Telemetry & Analytics Module збирає дані про виконання сценарію, метрики продуктивності та поведінкові показники. Оброблена телеметрія використовується для аналізу прогресу та адаптації сценарію в режимі реального часу.

Interface & Visualization Module забезпечує взаємодію користувача з системою, відображає результати, параметри навчання та підтримує режим інструкторського перегляду або аналізу повторів.

Модульність дозволяє системі реалізовувати повний цикл генерації — від створення подій до їхнього аналізу та відображення. Поєднання C++ для базової логіки та Blueprint для сценарних елементів забезпечує баланс між продуктивністю та гнучкістю, необхідною для адаптивних навчальних симуляторів [24].

2.1.2 Особливості роботи системи у зв’язці з рушієм Unreal Engine

Інтеграція системи генерації тренувальних сценаріїв з Unreal Engine 5 передбачає поєднання користувацьких C++-модулів із нативними підсистемами рушія, такими як рендеринг, фізика, AI-система та процедурна генерація, що забезпечує стабільність роботи та точність симуляції [25]. Реєстрація власних класів і компонентів у рушії дозволяє сценарному ядру безпосередньо керувати ігровими об’єктами, створювати події та змінювати параметри середовища у реальному часі.

Blueprint-логіка виступає зв’язковою ланкою між високорівневими сценарними структурами й низькорівневими C++-компонентами, забезпечуючи

гнучке налаштування подій і можливість змінювати перебіг тренування без перекомпіляції системи [26]. Процедурні алгоритми інтегруються з фізичними та графічними підсистемами UE5, що дозволяє моделювати динамічні зміни середовища та поведінку NPC відповідно до дій користувача.

Система також підтримує взаємодію з інструментами візуалізації, логування та аналітики, включно з Marketplace-плагінами для VR/AR та цифрових двійників. Профілювання та автоматизоване тестування здійснюється через Unreal Insights та Automation Framework, що забезпечує контроль продуктивності й коректності сценаріїв [27]. Додаткова інтеграція з зовнішніми сервісами через REST-інтерфейси розширює можливості аналізу та зворотного зв'язку, створюючи єдине інтерактивне середовище для навчальних симуляцій.

2.2 Побудова тренувальних сценарних моделей

Моделювання тренувального сценарію в системі інтелектуальної генерації для Unreal Engine 5 полягає у формалізації навчального процесу як послідовності взаємопов'язаних подій, станів і реакцій, що відтворюють реалістичну навчальну ситуацію у віртуальному середовищі. Основна мета моделювання — створити адаптивну симуляцію, яка поєднує структурованість педагогічного процесу з динамікою поведінкової взаємодії користувача та агентів. У цьому контексті сценарій розглядається як динамічна система [28], у якій стан середовища змінюється залежно від дій учасника, навчальних цілей і заздалегідь визначених обмежень. На першому етапі формується формалізована модель сценарію, яка описує:

— цілі навчання;
— початкові умови середовища;
— набір подій і тригерів, що керують переходами між станами;
— систему обмежень часових, просторових, когнітивних або поведінкових.

Ці параметри описуються через структуру сценарного графа (Scenario Graph), який визначає послідовність подій і логіку переходів між ними. Кожен вузол графа

відповідає певній фазі навчання — від ініціалізації до аналізу результатів, що зображено на рисунку 2.2. Граф дозволяє задавати розгалуження (альтернативні шляхи) та зворотні переходи у разі помилкових дій користувача. Таким чином, сценарій не є лінійним, а набуває властивостей адаптивної поведінкової системи, здатної змінюватися під час симуляції [29].

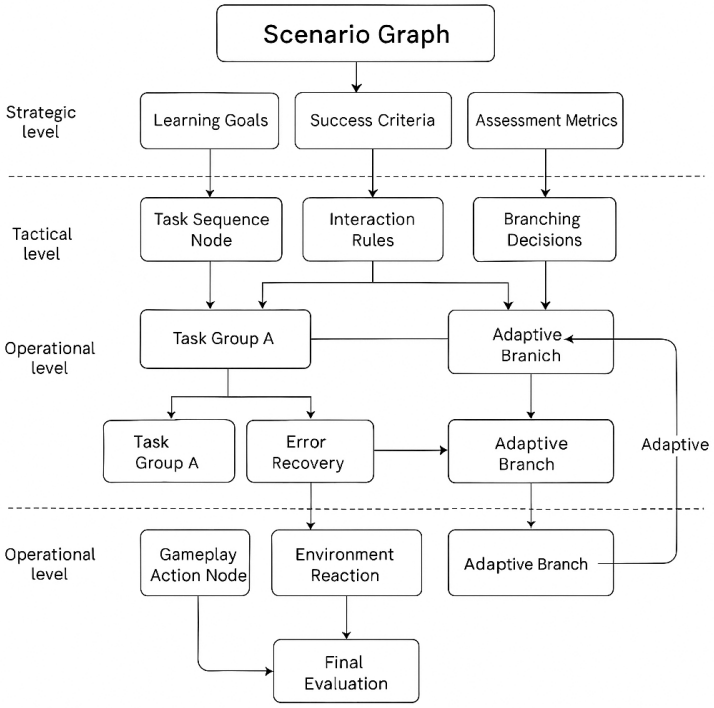

Рисунок 2.2 — Структура сценарного графа

Кожен із цих рівнів має власний набір класів і компонентів, які реалізують сценарну логіку. C++ використовується для базової поведінкової логіки, контролю станів і аналітичних обчислень, тоді як Blueprint забезпечує сценарне управління подіями — тригери, часові переходи, візуалізацію завдань, взаємодію з UI. Таке розділення дозволяє досягти гнучкості в педагогічному налаштуванні та одночасно підтримувати високу продуктивність [30].

Ключовою особливістю є механізм адаптації сценарію. Під час виконання симуляції система постійно збирає дані про дії користувача: швидкість реакції, точність виконання, частоту помилок і використання підказок. На основі цих даних адаптивний алгоритм змінює складність середовища:

— коригує кількість об’єктів або NPC;

— змінює часові рамки завдань;

— активує додаткові події або вводить нештатні ситуації;

— надає контекстні підказки у разі повторних помилок.

Таким чином, сценарій функціонує у режимі зворотного зв’язку, реагуючи на індивідуальну поведінку користувача. Подібні системи, як зазначено у ML-SceGen та Unreal-MAP, демонструють підвищення ефективності навчання завдяки динамічному налаштуванню рівня складності та зменшенню когнітивного навантаження на учасника.

Окреме значення має аналітична підсистема, яка фіксує ключові події, збирає телеметрію (час, дії, успішність) та проводить попередню оцінку компетентностей користувача. Ці дані використовуються для автоматичного формування звіту, оцінювання прогресу та оновлення параметрів сценарію у наступних сесіях. Таким чином, система перетворюється на самонавчальний тренувальний комплекс, що накопичує статистику для вдосконалення власних алгоритмів і підвищення педагогічної ефективності [31].

Завдяки такому підходу моделювання тренувального сценарію в Unreal Engine виступає не просто як технологічна процедура, а як педагогічно обґрунтований процес, який поєднує програмну гнучкість, динамічну адаптацію та аналітичну підтримку. Це створює передумови для розробки інтелектуальних навчальних середовищ, здатних імітувати реальні ситуації, оцінювати компетенції та формувати персоналізовані траєкторії навчання.

2.2.1 Логічні структури та правила для визначення поведінки сценарію

Формалізація обмежень і логіки тренувального сценарію є фундаментальним етапом побудови інтелектуальної системи генерації, оскільки саме вона визначає структуру навчального процесу, правила взаємодії між агентами та умови досягнення навчальних цілей. Мета цього етапу полягає у створенні єдиної математико-логічної моделі, яка описує всі дозволені й заборонені дії, часові рамки, ресурси, стани середовища та можливі шляхи проходження сценарію. Така

модель забезпечує контрольованість і відтворюваність симуляції, а також створює основу для динамічної адаптації сценаріїв до поведінки користувача [32].

У системі на базі Unreal Engine формалізація реалізується через сукупність станів, подій і переходів, представлених у вигляді графів або дерев сценарної логіки. Кожен стан (State) відповідає певному етапу навчання (наприклад, “Початок тренування”, “Виконання завдання”, “Аналіз результатів”), а переходи між станами визначаються умовами — умовні оператори, тригери або змінні середовища. Формалізована структура сценарію може бути описана як скінченний автомат (Finite State Machine), де множина $S = \{s_0, s_1, \dots, s_n\}$визначає можливі стани, а функція переходів $\delta(s, a) \to s'$задає логіку змін при виконанні певних дій a. У реалізації Unreal Engine це втілюється через Behavior Trees, Event Graphs у Blueprint або C++ класи, які моделюють стан і події за принципом “умова — дія — наслідок”.

Структурний рівень визначає загальну архітектуру сценарію — послідовність завдань, переходи між подіями, обмеження за часом і ресурсами. Наприклад, користувач має обмежений час для виконання дії (“знайти об’єкт протягом 60 секунд”), а після перевищення ліміту система автоматично змінює сценарну гілку або знижує складність. На цьому рівні формалізація виражається через таблиці умов і станів, які потім транслюються в Blueprint- або C++-код.

Поведінковий рівень моделює реакції системи на події у віртуальному середовищі — поведінку агентів, реакцію NPC, фізичні або логічні зміни. Наприклад, якщо користувач неправильно взаємодіє з об’єктом, агент може “підказати” правильну дію або створити нову ситуацію для повторення завдання. Для цього використовуються умовні конструкції (if/else, switch) у C++, Event Dispatchers у Blueprint і AI контролери, які змінюють поведінку залежно від змінних сценарію [33].

Аналітичний рівень відповідає за опис критеріїв успішності, оцінювання дій користувача й логіку зворотного зв’язку. Наприклад, система може вважати завдання виконаним, якщо користувач досяг ефективності понад 80% або виконав дії в межах заданого часу. Ці обмеження формалізуються як логічні вирази, які

автоматично перевіряються під час симуляції, а результати зберігаються у модулі аналітики для подальшої адаптації сценарію. На рисунку 2.3 представлено схему аналітичного рівня сценарію, яка відображає перевірку умов успішності, часові обмеження та гілкування подій залежно від дій користувача.

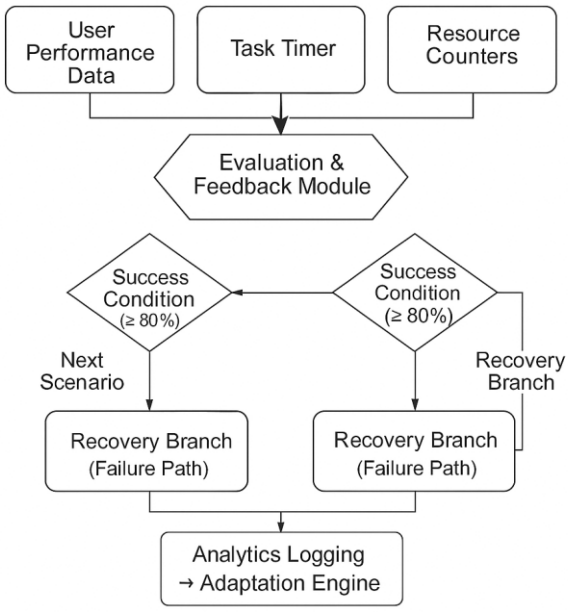

Рисунок 2.3 — Схема аналітичного рівня формалізації навчального сценарію

Формалізація обмежень і логіки сценарію створює інтелектуальний каркас навчальної системи, у межах якого поєднуються структурована педагогічна модель і реактивна динаміка віртуального середовища. Використання скінченних автоматів, поведінкових дерев і модульної логіки в C++/Blueprint забезпечує прозорість, контрольованість і гнучкість навчальних процесів, дозволяючи системі автоматично підлаштовуватися під користувача, змінювати сценарні гілки та точно оцінювати результати його дій.

2.2.2 Підбір складності під рівень користувача

Адаптація складності є ключовим елементом інтелектуальних тренувальних систем, оскільки дозволяє узгоджувати навчальні завдання з поточним рівнем компетентності користувача. Її мета — підтримувати оптимальний баланс між

складністю та досяжністю, забезпечуючи стабільний навчальний прогрес без перевантаження [34].

Система аналізує телеметричні показники — час виконання, точність дій, кількість помилок і динаміку прогресу. На основі цих даних адаптаційний модуль коригує параметри сценарію: змінює доступний час, кількість ресурсів, інтенсивність подій, поведінку агентів або кількість підказок [35].

Для підтримки стабільності використовується модель рівнів компетентності (Beginner–Expert), де перехід відбувається автоматично при досягненні певних порогових значень. Високі результати спричиняють підвищення складності, тоді як часті помилки активують спрощені сценарні варіанти чи додаткові рекомендації.

У Unreal Engine адаптаційна логіка реалізується за допомогою Behavior Trees, Blackboard-змінних і Blueprint-скриптів, тоді як аналітичні обчислення здійснюються на рівні C++. Це дозволяє сценаріям динамічно реагувати на поведінку користувача та змінювати умови симуляції під час виконання.

Система зворотного зв’язку фіксує причини адаптаційних рішень і оцінює їхній ефект на подальші результати. Такий механізм забезпечує індивідуалізацію навчання та довготривале підвищення ефективності тренувального процесу [36].

2.2.3 Збір реакцій та коригування поведінки сценарію

Механізм зворотного зв’язку є центральним елементом адаптивних тренувальних систем, оскільки забезпечує постійний аналіз дій користувача та корекцію сценарію в режимі реального часу [37]. Його робота ґрунтується на циклі: фіксація дії → аналітична оцінка → зміна умов → повідомлення користувачу.

Unreal Engine автоматично збирає телеметрію — час реакції, точність, кількість помилок і використані ресурси — через системи логування Blueprint і C++. Дані надходять до аналітичного модуля, де порівнюються з порогами ефективності. У разі перевищення кількості помилок або спаду точності система спрощує сценарій чи надає підказку; за стабільно високих показників — ускладнює завдання. Цей процес формалізується як регуляторна функція:

$$C_{t+1} = C_t + k \cdot (E_{target} - E_{user}),$$

де C — рівень складності,

E_{user} — фактична ефективність користувача,

E_{target} — бажаний показник,

k — коефіцієнт адаптації.

Зворотний зв’язок подається через візуальні підказки, повідомлення або панелі результатів, а підсумковий аналіз зберігається у звітах для подальшої адаптації. Поєднання моментального та підсумкового фідбеку забезпечує персоналізацію навчання й підвищує точність роботи адаптивного алгоритму.

Сучасні реалізації поєднують фідбек із моделями машинного навчання, які прогнозують успішність користувача та коригують сценарій наперед. Такі системи демонструють зменшення кількості помилок до 40% та пришвидшення формування навичок у порівнянні зі статичними симуляторами.

2.3 Алгоритмічні рішення та процеси в системі

Алгоритм роботи системи генерації тренувальних сценаріїв в Unreal Engine має циклічну структуру, що включає ініціалізацію параметрів, процедурну побудову середовища, виконання сценарію, аналіз дій користувача та адаптацію складності. На початковому етапі система визначає мету, рівень складності та конфігурацію середовища, активуючи модулі керування логікою та телеметрією [39]. Схематично цей процес подано на рисунку 2.4, де відображено повний цикл функціонування генератора.

Після ініціалізації запускається процедурна генерація, яка формує оточення й розміщує ключові об’єкти. Сценарна логіка керує поведінкою агентів і зміною подій через Behavior Trees та Blueprint-графи. Під час виконання система постійно аналізує телеметрію і коригує сценарій у реальному часі: спрощує завдання, змінює динаміку подій або ускладнює навантаження залежно від ефективності користувача.

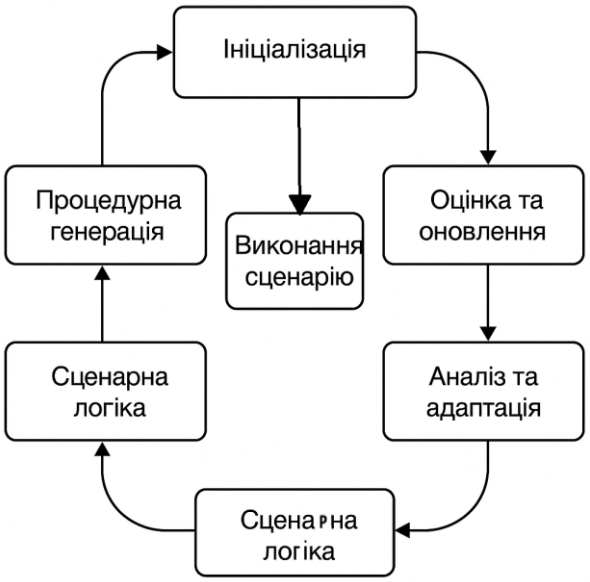

Рисунок 2.4 — Узагальнений цикл роботи системи генерації тренувальних сценаріїв

Після завершення тренування відбувається оцінка результатів та оновлення моделей, що впливають на наступні сценарії. У такий спосіб формується замкнений адаптивний цикл, у якому сценарій еволюціонує відповідно до досвіду та прогресу користувача.

2.3.1 Побудова груп сценарних варіантів та їх кластеризація

Створення різних типів тренувальних сценаріїв є ключовим елементом проєктування навчального середовища, оскільки саме сценарна варіативність визначає гнучкість і ефективність симулятора. Класифікація базується на типі завдання, рівні адаптивності, кількості учасників і способі побудови середовища.

Лінійні сценарії мають фіксовану послідовність подій і застосовуються для базового навчання та інструктажу. Реалізація виконується через послідовні вузли Blueprint, що забезпечує контрольоване проходження та прогнозований результат, але обмежує варіативність.

Адаптивні сценарії змінюють складність відповідно до результатів користувача. Система аналізує темп, точність і помилки, після чого коригує події в реальному часі через Behavior Trees та аналітичні модулі. Такий підхід

використовується у тренажерах динамічних ситуацій і забезпечує підвищену реалістичність [40].

Мультиагентні сценарії включають взаємодію кількох NPC або користувачів. Вони використовують MARL-підхід або Unreal-MAP для створення поведінкових моделей кооперації чи конкуренції. Це дає змогу відпрацьовувати командні навички, рольові взаємодії та поведінку в умовах стресу.

Процедурно-генеровані сценарії формуються автоматично за правилами PCG Framework. Кожне проходження створює нову просторову конфігурацію, NPC та події. Це забезпечує високу реіграбельність і дає можливість масштабувати систему без ручного створення контенту.

Інтелектуальні сценарії зі зворотним зв’язком поєднують аналіз дій користувача з динамічною зміною сценарію. Вони працюють за циклом Observe → Analyze → Adapt → Feedback, коригуючи сценарій і формуючи рекомендації після кожної взаємодії. UE забезпечує таку реактивність через Blueprint-сигнали, Behavior Trees та C++-аналітику.

Узгоджене використання різних категорій сценаріїв створює багаторівневу навчальну систему, яка може підтримувати як базові, так і високостресові або командні тренінги. Така інтеграція забезпечує адаптивність, реалістичність і дидактичну ефективність симуляторів Unreal Engine.

2.3.2 Просторове розміщення елементів у середовищі

Динамічне розміщення об’єктів є ключовим механізмом адаптивних тренувальних систем на Unreal Engine, оскільки забезпечує варіативність, неповторність проходжень і відповідність складності поточному стану користувача. На відміну від статичних сцен, об’єкти генеруються або переміщуються під час симуляції, формуючи реактивне середовище, яке змінюється відповідно до темпу, помилок чи завдань користувача.

Технічно цей механізм реалізується через Procedural Content Generation (PCG) та системи Spawn (SpawnActor, SpawnFromClass, AddInstance). Розміщення може базуватися на випадковості, правилах (Placement Rules) або просторових

моделях карти. PCG Graphs визначають діапазони координат, кількість елементів, орієнтацію й інші параметри, дозволяючи оновлювати середовище під час роботи сценарію.

Інтелектуальне розміщення (Smart Placement) використовує EQS та AI Perception, що дозволяє розташовувати об’єкти з урахуванням видимості, шляхів доступу чи поведінки користувача. Система може змінювати позиції об’єктів, якщо користувач систематично їх ігнорує, або додавати нові перешкоди за високої ефективності.

З педагогічної точки зору, динамічне розміщення запобігає механічному запам’ятовуванню сценаріїв і стимулює розвиток ситуаційного мислення. Завдяки цьому користувач навчається принципам реагування, а не фіксованим маршрутам. У результаті Unreal Engine виступає як адаптивна платформа, що генерує навчальний простір у режимі реального часу та синхронно еволюціонує разом із користувачем.

2.3.3 Узгодження сценарних подій і корекція деталей під час виконання

Контроль та корекція логіки тренувальних сценаріїв забезпечують узгодженість, адаптивність і стійкість навчального процесу в динамічних симуляціях Unreal Engine. Система постійно перевіряє актуальний стан сценарію, порівнюючи його з очікуваною моделлю, і за потреби автоматично змінює перебіг подій.

Основою управління є станова машина (FSM) або сценарний граф, де кожен вузол описує етап навчання, а переходи активуються залежно від дій користувача. Unreal Engine реалізує це через Blueprint Event Graphs, Behavior Trees та Blackboard Variables. Якщо користувач порушує умови (наприклад, не вкладається у час), система активує альтернативну гілку: спрощує завдання, додає підказку або змінює поведінку NPC.

Контроль відбувається на двох рівнях:

— локальному — реагування об’єктів через Blueprint/C++ (тригери, змінні, події),

— глобальному — аналіз цілісності сценарію модулем Scenario Monitor, який відстежує логічні збої та виконує автоматичну корекцію без переривання симуляції.

Корекція базується на телеметрії: якщо певна дія часто викликає помилки, система адаптує сценарій завчасно, інколи з використанням ML-прогнозування. При цьому навчальні цілі залишаються сталими — змінюється лише шлях їх досягнення (принцип контрольованої гнучкості).

$$C(t) = \begin{cases} 1, \text{якщо } S_{actual}(t) = S_{expected}(t) \\ 0, \text{якщо } S_{actual}(t) \neq S_{expected}(t) \end{cases}$$

У результаті сценарій працює як саморегульована система, що підтримує навчальну послідовність навіть за умов непередбачуваних дій користувача, поєднуючи стабільність логіки та педагогічну адаптивність.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ТА СИСТЕМНИХ МОДУЛІВ

3.1 Внутрішня організація програмного коду

У межах роботи було розроблено систему генерації тренувальних сценаріїв в Unreal Engine 5.5, побудовану за модульно-ієрархічною архітектурою. Структура забезпечує розділення логіки, гнучкість масштабування та можливість адаптації сценаріїв до рівня користувача. До складу системи увійшли такі основні підсистеми:

— ядро сценарної логіки;

— адаптивний модуль;

— модуль процедурного розміщення об’єктів;

— модуль аналітики й телеметрії;

— інтерфейсний модуль.

Архітектуру реалізовано через схему GameInstance → Subsystem → Component, що дало змогу створити незалежні, але взаємопов’язані елементи. Загальні зв’язки між основними класами показано на рисунку 3.1.

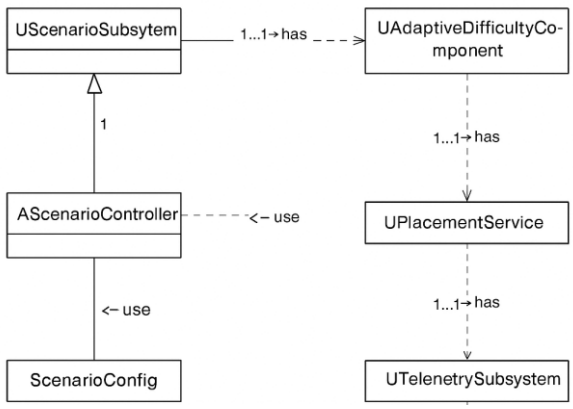

Рисунок 3.1 — UML-діаграма основних класів системи та їх зв’язків

Центральним елементом системи було розроблено клас UScenarioSubsystem, який керує ініціалізацією сценарію, конфігураціями та переходами між станами.

Для кожного сценарію створюється контролер AScenarioController, що реалізує модель кінцевого автомата та визначає етапи роботи сценарію.

Параметри навчання було структуровано у ScenarioConfig, реалізованому як Primary Data Asset. Це дозволило змінювати сценарії без редагування коду.

Адаптацію складності забезпечує UAdaptiveDifficultyComponent, який аналізує час реакції, помилки та інші метрики, після чого автоматично коригує перебіг сценарію.

Збір телеметрії було реалізовано через UTelemetrySubsystem, що формує статистику та забезпечує зворотний зв’язок з іншими модулями.

Процедурне формування середовища було реалізовано у UPlacementService, який автоматично розміщує або оновлює об’єкти за допомогою PCG Framework і функцій SpawnActor.

Керування поведінкою агентів було реалізовано через AIController, Behavior Trees і Blackboard, що дозволило динамічно змінювати стратегію NPC відповідно до стану сценарію.

Системна логіка, генерація, аналітика та обробка даних реалізовані на C++, тоді як Blueprint використано для візуальної логіки, UI та монтажу подій.

У результаті було створено гнучку, адаптивну систему, здатну генерувати й модифікувати сценарії в реальному часі.

У центрі архітектури знаходиться ядро сценарної системи, побудоване навколо підсистеми ScenarioSubsystem, яка виконує роль керуючого елемента всього процесу навчання. Вона відповідає за ініціалізацію сценаріїв, координацію підсистем, збирання аналітики та забезпечення доступу до основних функцій із будь-якого елемента гри.

Кожен сценарій втілюється через контролер сценарію (ScenarioController), який управляє станами, ініціює побудову середовища, активує події та слідкує за їх послідовністю. Контролер взаємодіє з автоматом станів (ScenarioStateMachine), що реалізує сценарну логіку — переходи між етапами навчання, реакцію на події, обробку успішності чи помилок користувача. Такий підхід дозволяє розмежувати

сценарну логіку від внутрішньої архітектури рушія, що значно спрощує розширення системи.

Важливою складовою є модуль адаптивної складності (AdaptiveDifficultyComponent), який постійно аналізує дані користувацької взаємодії, фіксує кількість помилок, швидкість реагування, час виконання завдань і на основі цих показників коригує рівень складності сценарію. Механізм працює в реальному часі, змінюючи інтенсивність подій, кількість підказок або поведінку віртуальних агентів для досягнення оптимального рівня залученості студента.

Модуль розміщення об’єктів (PlacementService) відповідає за створення та оновлення елементів середовища під час тренування. У його завдання входить ініціалізація простору, динамічний спавн елементів, контроль їхніх позицій і параметрів, а також оновлення композиції сцени залежно від складності сценарію чи прогресу користувача. Завдяки інтеграції з процедурними інструментами Unreal Engine (Procedural Content Generation Framework) цей модуль може генерувати варіативні середовища без необхідності ручного редагування.

Для забезпечення збору статистики в систему впроваджено підсистему телеметрії (TelemetrySubsystem), яка фіксує дії користувача, тривалість виконання завдань, показники успішності та динаміку помилок. Отримані дані передаються до адаптаційного модуля, що дозволяє реалізувати механізми зворотного зв’язку, створювати звіти та надавати рекомендації користувачам.

Кожен сценарій описується за допомогою конфігураційного об’єкта (ScenarioConfig), реалізованого на основі Primary Data Asset. Конфігурація містить структуру станів, параметри переходів, порогові значення для оцінки дій користувача, а також набори класів об’єктів, які використовуються в конкретному тренуванні. Завдяки цьому можлива швидка зміна структури сценарію без втручання у код.

Система побудована за принципом гібридного розподілу логіки між C++ та Blueprint, що дозволяє поєднати стабільність і продуктивність нативного коду з гнучкістю візуального редагування.

Функціонал, пов’язаний із системними процесами, складними обчисленнями та керуванням потоками подій, реалізовано мовою C++. До таких елементів належать: логіка автомата станів, адаптаційні алгоритми, аналіз телеметрії, управління сценарними подіями та взаємодія із середовищем.

Водночас Blueprint-рівень використовується для реалізації візуальної частини навчального процесу — відображення підказок, управління інтерфейсами користувача, створення тригерів, активації завдань і анімацій. Через Blueprint-події розробник може визначати сценарні реакції без перекомпіляції C++-модулів, що особливо важливо при налаштуванні великої кількості варіантів сценаріїв.

Типовий цикл взаємодії виглядає так: ядро (C++) формує структуру сценарію та викликає Blueprint-події, які реалізують конкретну поведінку у сцені. Результати дій користувача фіксуються через Blueprint-вузли телеметрії, що передають дані назад у C++-підсистему для аналізу. Таким чином, обидва рівні утворюють замкнену систему взаємодії, де Blueprint забезпечує візуальну динаміку, а C++ — логічну цілісність і контроль.

Вибір саме такої архітектури дозволяє досягти балансу між продуктивністю та зручністю розробки. Завдяки поділу відповідальностей програмний комплекс підтримує масштабування, повторне використання компонентів і розширюваність, що особливо важливо для складних навчальних симуляторів, де необхідно швидко модифікувати логіку без змін у низькорівневому коді рушія.

3.2 Центральна логічна частина системи

Модуль інтелектуальної генерації середовища реалізовано як набір пов’язаних підсистем (Subsystems, Components, Controllers), що працюють у циклі ініціалізація → побудова → виконання → адаптація → завершення. Центральним керуючим елементом виступає ScenarioController, який отримує параметри з ScenarioConfig (Primary Data Asset), ініціює PCG-процеси й активує поведінкові моделі агентів. Конфігурація середовища формується залежно від типу сценарію, цілей навчання, стартової складності та статистики попередніх сесій.

ScenarioConfig зберігає ідентифікатор сценарію, навчальні цілі, перелік станів і переходів, початкові пороги складності, набір класів об’єктів, правила розміщення та політики адаптації. Конфігурація може посилатися на DataTables із рівнями компетентності. Кожна сесія має seed-значення, що гарантує відтворюваність при варіативності середовища.

На старті ScenarioController зчитує конфігурацію, ініціалізує автомат станів і активує телеметрію та адаптивність. У мережевому режимі перевіряється реплікація, щоб усі операції генерації виконувалися детерміновано на сервері.

Побудова простору виконується через PCG-графи та Placement-сервіс, які формують макроструктуру сцени й розташовують об’єкти відповідно до правил. У разі конфліктів (колізії, недоступні маршрути) відбувається локальна перебудова. Після створення геометрії активується логіка сценарію: автомат переходить у початковий стан, показуються інструктажі, працюють Behavior Trees та EQS для NPC.

У процесі симуляції телеметрія збирає показники (час, помилки, стабільність). Адаптивний модуль за їх зміною коригує параметри — час виконання, інтенсивність подій, щільність NPC, кількість підказок або мікроструктуру середовища. У разі потреби сценарій переводиться на альтернативну гілку без втрати навчальної мети.

Усі випадкові значення контролюються через єдиний генератор випадковості, а в мережевому режимі — сервером. Після адаптивних змін Placement-сервіс виконує швидкі перевірки коректності розміщення. Якщо виявлено критичні помилки (непрохідність, конфлікти тригерів) система виконує локальне виправлення або відкат.

Після завершення сценарію модуль оцінювання формує агреговані показники (точність, час, використання підказок, стабільність). На їх основі оновлюються пороги складності для наступної сесії. Важкі операції генерації виконуються пакетно або стрімляться для уникнення втрати продуктивності.

Для дизайнерів передбачено набір Blueprint-подій і конфігураторів DataAsset, що дозволяють змінювати параметри сценарію та виконувати тестові прогони без перекомпіляції C++.

3.2.1 Побудова сценарної структури та переходів між етапами

У розробленій системі логіка тренувального процесу реалізована у вигляді сценарного графа, що виконується кінцевим автоматом станів, приклад зображений на рисунку 3.2. Така модель відокремлює педагогічну структуру сценарію від внутрішніх механік рушія: граф визначає послідовність навчальних фаз, а автомат — умови переходів між ними. Це дозволяє керувати сценаріями даними з конфігурацій, забезпечувати детермінованість виконання та підтримувати стабільність навіть під час динамічних змін середовища.

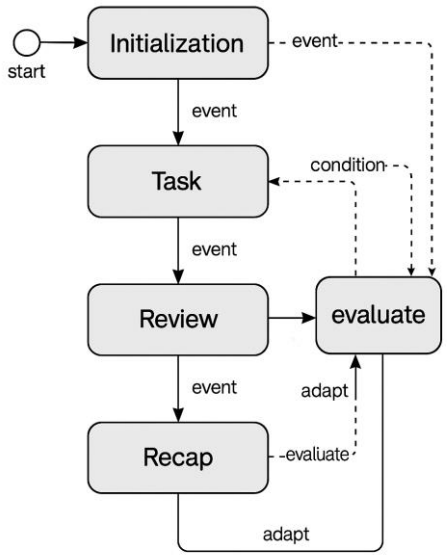

Рисунок 3.2 — Структура сценарного графа та кінцевого автомата станів у розробленій системі

Сценарний граф складається зі станів, подій і умов переходів. Кожен стан має педагогічну ціль (інструктаж, виконання завдання, перевірка результату, рефлексія) та визначені дії при вході/виході. Події, що ініціюють переходи, можуть бути ігровими (дія користувача), часовими, аналітичними або системними. Умови

переходів задаються декларативно в конфігурації, а автомат працює у циклі: ініціалізація → виконання → оцінка → адаптація.

На етапі ініціалізації зчитуються параметри складності, активуються базові PCG-компоненти та поведінкові дерева агентів. Під час виконання сценарію автомат обробляє асинхронні події з рушія та аналітики, застосовуючи локальні реакції або переходячи до наступного стану. Логіка подій є ідемпотентною, що запобігає неконтрольованим каскадам реакцій.

Коли умова завершення стану виконана, система переходить до короткої оцінки: телеметрія надає агреговані метрики, обчислюється локальний результат, який заноситься в журнал сесії разом із контекстом середовища. Після цього відбувається адаптація — модуль складності формує пакет змін (час, підказки, поведінка NPC, локальні перебудови середовища), які застосовуються перед входом у наступний стан.

Переходи синхронізуються зі стрімінгом рівнів та системами середовища. NPC реагують на зміну станів через Blackboard-змінні, плавно змінюючи поведінкові патерни. Перед переходом виконується валідація доступності маршруту, коректності таймерів та відсутності конфліктів; у разі відхилень система активує м’яку корекцію або підстан Recovery.

Система також підтримує паралельні підзавдання через композитні стани та визначені політики пріоритетів. У мультикористувацьких сценаріях автомат працює авторитетно на сервері, нормалізує події та транслює клієнтам лише рішення щодо переходів і адаптації.

Усі ключові події циклу — вхід/вихід станів, причини переходів, застосовані корекції — фіксуються у журналі сесії, що забезпечує можливість відтворення та аналітичного аналізу.

Таким чином, тренувальний процес реалізується як керований сценарний цикл, у якому кожен стан має чітку структуру, а переходи супроводжуються формальною оцінкою та адаптивними змінами, що забезпечує прозорість, стійкість й відтворюваність логіки тренувань.

3.2.2 Блок визначення розміщення об’єктів на основі правил та EQS

Інтелектуальне розміщення об’єктів у системі генерації тренувальних сценаріїв є однією з ключових підсистем, що забезпечує динамічну зміну просторової конфігурації середовища відповідно до контексту сценарію, поточних дій користувача та навчальних цілей. На відміну від статичного дизайну рівнів, інтелектуальне розміщення функціонує як керований процес прийняття рішень, у якому поєднуються процедурна генерація (PCG), аналітичні правила (Placement Rules) та семантичні алгоритми Unreal Engine, зокрема Environment Query System (EQS).

Основним виконавчим елементом є компонент UPlacementService, який працює у складі ядра сценарію (UScenarioSubsystem) і реалізує API для створення, переміщення, видалення та перегенерації об’єктів. Модуль взаємодіє з трьома основними підсистемами рушія:

— procedural content generation framework (PCG) для побудови базових патернів середовища та автоматичного розміщення об’єктів на етапі ініціалізації сценарію.

— environment query system (EQS) для динамічного визначення оптимальних позицій у просторі під час виконання сценарію, з урахуванням факторів доступності, видимості та небезпеки.

— physics/navigation system для перевірки валідності спавну: відсутності колізій, забезпечення навігаційних шляхів, підтримки LOD і стабільності при зміні геометрії світу.

PlacementService має двоступеневу структуру:

— static layer визначає початкове базове розміщення за PCG-шаблонами;

— dynamic layer коригує позиції або створює нові об’єкти під час виконання сценарію;

Кожен об’єкт, розміщений системою, має Placement Descriptor, який описує: тип, клас Blueprint або Actor, категорію (training, assistive, hazard), стан активності, пріоритет сценарного впливу, а також історію адаптацій (як і коли був змінений).

Для контролю процесу генерації введено формальний набір Placement Rules, який задає просторові, логічні та педагогічні обмеження. Основні категорії правил:

— мінімальні/максимальні відстані між об’єктами однієї категорії;

— виключення зон (no-spawn regions) — наприклад, стартова позиція користувача, контрольні точки або навчальні інтерфейси;

— врахування висоти, нахилу поверхні, рівня освітленості чи шуму (через EQS).

Семантичні правила (Semantic Rules):

— об’єкти певного типу не можуть розміщуватися без попередньої події (наприклад, двері без завдання на відкриття);

— взаємозалежність між об’єктами — якщо активовано тригер A, об’єкти B мають бути недоступні або замінені.

Поведінкові правила (Behavioral Rules):

— тренувальні об’єкти з’являються лише при досягненні певного стану сценарію;

— інтенсивність і кількість об’єктів змінюються відповідно до продуктивності користувача.

Педагогічні правила (Instructional Rules):

— складність простору відповідає поточному рівню компетентності;

— навчальні об’єкти завжди знаходяться в межах доступності для користувача з певним порогом навичок.

Ці правила описуються у вигляді DataTable PlacementRulesTable, що зберігає формальні умови та дії (умова → обмеження → реакція). Під час кожної генерації система проходить цю таблицю, застосовуючи фільтри перед викликом EQS або PCG.

Алгоритм розміщення складається з таких послідовностей, які зображені на рисунку 3.3

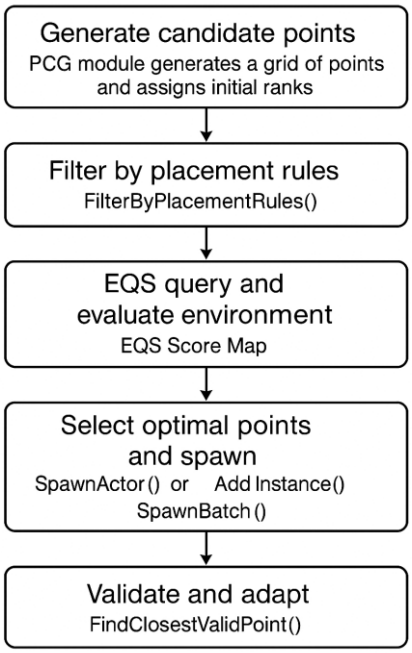

Рисунок 3.3 — Алгоритм розміщення об’єктів у системі генерації середовища

Під час виконання сценарію PlacementService працює в реактивному режимі. Кожна дія користувача, що впливає на просторову логіку (переміщення, руйнування, виконання завдання), створює подію OnEnvironmentChanged. Ця подія запускає часткову перегенерацію локальної області за допомогою EQS/PCG у межах радіуса впливу. Приклад: якщо користувач швидко виконав задачу в одній зоні, система може автоматично активувати “наступну хвилю” подій — розмістити нові об’єкти чи NPC у сусідньому секторі, але з урахуванням результатів попереднього етапу. Це створює ефект живого середовища, яке реагує на успішність дій студента.

Для підвищення продуктивності система застосовує регіональні кванти: сцена ділиться на секції, кожна з яких має власний набір PCG/EQS параметрів. При адаптації змінюються лише активні секції, що мінімізує затримки та не впливає на FPS під час оновлення.

PlacementService тісно інтегрований із ScenarioStateMachine. Переходи між станами (наприклад, “Init → Task → Evaluation → Adaptation”) супроводжуються перевірками валідності об’єктів. Якщо у наступному стані потрібні нові типи

елементів, система відвантажує непотрібні й завантажує нові згідно з правилами сценарію.

При цьому підтримується “контекстна спадковість” — об’єкти, створені на попередніх етапах, можуть змінювати статус, але не зникають миттєво (щоб не руйнувати логічну послідовність подій). Наприклад, NPC, який був пасивним спостерігачем, у новому стані може стати активним тренувальним агентом, не вимагаючи пересоздання.

Інтелектуальне розміщення об’єктів у системі поєднує процедурну генерацію, семантичну логіку та аналітичну адаптацію. Використання PCG забезпечує швидке формування базового простору, EQS — прийняття “розумних” рішень щодо оптимального позиціонування, а Placement Rules гарантують відповідність навчальним цілям і технічну коректність середовища. У результаті середовище не є статичним — воно еволюціонує разом із користувачем, реагуючи на його дії, рівень компетентності та контекст навчального завдання, що робить симуляцію одночасно динамічною і педагогічно контрольованою.

3.2.3 Коригування рівня складності залежно від ходу виконання

Механізм адаптивної складності є одним із ключових елементів системи інтелектуальної генерації тренувальних сценаріїв, який забезпечує підлаштування навчального процесу під індивідуальний рівень користувача. Основна ідея полягає у підтримці так званого “балансу виклику” — стану, коли завдання залишаються достатньо складними, щоб стимулювати мислення та практичні дії, але не перевищують поточних можливостей користувача. У системах на Unreal Engine це реалізується через замкнену петлю адаптації, що складається з трьох взаємопов’язаних етапів: телеметрія, оцінка та корекція. Такий підхід забезпечує безперервний аналіз поведінки студента, визначення рівня його компетентності та динамічне регулювання складності сценарію в реальному часі.

На першому етапі відбувається збір телеметрії — система фіксує всі ключові показники взаємодії користувача з навчальним середовищем. До таких показників належать час виконання завдань, кількість допущених помилок, точність дій,

частота використання підказок, стабільність виконання послідовних дій, а також поведінкові параметри, як-от вибір траєкторій руху чи частота звернень до інтерфейсу. Ці дані обробляються у реальному часі й агрегуються у внутрішню структуру телеметрії. Таким чином, система формує повну картину поточного стану користувача, не перериваючи основний навчальний процес.

На другому етапі оцінка результатів виконується спеціальним модулем, який аналізує отримані метрики та визначає інтегральний показник ефективності, або Performance Index. Він розраховується на основі вагових коефіцієнтів, що враховують точність, швидкість, кількість помилок і частоту звернень до допомоги. Наприклад, якщо користувач виконує завдання швидко, але допускає багато помилок, загальний індекс знижується; якщо дії точні, але надто повільні — система інтерпретує це як потребу у додаткових підказках чи спрощенні умов. Цей підхід дозволяє формалізувати складність навчання у цифровій формі, що зручно для аналітики та подальшого навчання системи на реальних даних.

Після розрахунку індексу активується етап корекції, під час якого сценарій підлаштовується під поточний рівень користувача. Якщо показники успішності нижчі за встановлений поріг, система зменшує кількість паралельних подій, додає візуальні підказки, збільшує час виконання завдань або переводить агентів у менш агресивний режим. Якщо ж користувач демонструє стабільно високі результати, система поступово підвищує складність — додає нові завдання, ускладнює логіку подій, збільшує швидкість реакцій NPC чи модифікує топологію середовища через модуль PlacementService. Корекція виконується плавно, без перезапуску сесії, щоб уникнути втрати занурення у процес навчання.

Щоб уникнути ефекту “гойдання”, коли складність змінюється занадто часто, система має вбудований інерційний механізм. Рішення про адаптацію приймається лише після аналізу кількох послідовних етапів, якщо тенденція до погіршення або покращення результатів є сталою. Крім того, усі зміни застосовуються поступово, обмежуючись невеликим відсотком варіацій у кожному циклі. Це дозволяє підтримувати стабільність сценарію, водночас забезпечуючи індивідуалізацію навчального процесу. Для користувача така система виглядає природно — він не

помічає моменту переходу на інший рівень складності, але постійно відчуває нові виклики, що підтримують інтерес і мотивацію.

Важливо, що адаптаційна петля працює у тісній взаємодії з іншими компонентами системи. Зокрема, з ScenarioController, який може змінювати сценарні гілки; з PlacementService, який перебудовує середовище відповідно до нових вимог; та з Feedback-модулем, який формує персоналізовані рекомендації для користувача. У підсумку система не лише реагує на поточні дії студента, а й прогнозує подальшу траєкторію його розвитку, створюючи умови для безперервного вдосконалення навичок. Це робить адаптивну складність не просто технічним механізмом балансування, а справжнім інструментом інтелектуального навчання, що поєднує аналітику, педагогіку та інтерактивну динаміку Unreal Engine.

3.3 Підсистема збору інформації та оцінки прогресу

Підсистема збору й аналізу даних є аналітичним ядром системи інтелектуальної генерації тренувальних сценаріїв, оскільки саме вона забезпечує безперервний моніторинг дій користувача, реєстрацію всіх подій симуляції та формування інформаційної основи для адаптації складності, оцінювання результатів і подальшого вдосконалення навчальних моделей. У межах Unreal Engine ця підсистема реалізується як комплекс взаємопов’язаних сервісів і класів, інтегрованих у загальну архітектуру рушія через C++ API та Blueprint-виклики.

Основним компонентом є Telemetry Service, який відповідає за фіксацію подій у реальному часі. Усі взаємодії користувача — натискання, пересування, активація об’єктів, помилки, звернення до підказок, зміни станів сценарію чи поведінки агентів — реєструються як структуровані записи (FScenarioTelemetryEvent). Кожен запис містить часову мітку, тип події, ідентифікатор користувача, сценарій, координати у світі, контекстну інформацію (наприклад, активний етап навчання або стан NPC) та короткий опис результату дії. Така деталізація дозволяє не лише аналізувати ефективність навчання, а й

реконструювати повний “слід користувача” для глибокого розбору процесу тренування.

Дані телеметрії зберігаються у кільцевому буфері пам’яті, який автоматично очищується після заповнення, запобігаючи надлишковому навантаженню на систему. Для довгострокового аналізу підсистема передбачає експортер даних, що періодично записує агреговані сесії у форматах JSON або CSV, придатних для обробки аналітичними інструментами. У більш розширених конфігураціях можливе з’єднання з віддаленим сервером або базою даних через REST API, що дозволяє централізовано збирати статистику з декількох клієнтів і будувати загальні звіти щодо ефективності тренувальної програми.

Аналітичний модуль підсистеми, Evaluation Engine, виконує інтерпретацію зібраних даних. Він аналізує показники продуктивності, стабільності, кількість повторів, частоту помилок і формує профіль користувача — FUserPerformanceProfile. Цей профіль містить параметри коротко- та довгострокової динаміки, що дозволяє відслідковувати прогрес не лише в межах поточного сценарію, а й у рамках усієї системи. Отримані результати передаються до модуля адаптації складності, який на основі цих метрик приймає рішення про зміни сценарних умов. Таким чином, створюється замкнене аналітичне коло: користувач впливає на дані, дані змінюють сценарій, а сценарій знову впливає на користувача.

Окремо варто виділити функцію контекстної аналітики, що дозволяє поєднувати кількісні метрики з контекстними факторами. Наприклад, система може виявити, що певні помилки виникають лише у специфічних умовах — при слабкому освітленні, у тісних приміщеннях або під час взаємодії з конкретним типом агентів. Такий рівень аналізу допомагає не лише адаптувати складність, а й покращити дизайн сценаріїв, усуваючи непередбачені бар’єри навчання.

У реалізації підсистеми використовуються як нативні інструменти Unreal Engine (система логування, DataTables, Performance Analytics), так і кастомні класи для узгодження даних між клієнтськими сесіями. Наприклад, UScenarioAnalyticsSubsystem відповідає за збір і попередню обробку подій, а

UReportManager генерує звіти про кожну навчальну сесію, формуючи показники ефективності та статистику навчального прогресу. Ці звіти можуть автоматично експортуватися у вигляді PDF або передаватися на сервер для подальшої обробки.

Підсистема збору й аналізу даних не є лише інструментом спостереження, а виступає основою для самонавчання всієї системи. Вона перетворює тренувальний процес із пасивної симуляції на активну взаємодію, у якій дані про дії користувача стають рушійною силою для розвитку системи. Це дозволяє реалізувати принцип “feedback-driven simulation” — коли кожна дія, оцінка і результат безпосередньо впливають на подальший хід тренування, забезпечуючи глибоку персоналізацію та високу ефективність навчального процесу.

3.3.1 Фіксація подій і їх агрегування у статистичну модель

Підсистема логування та метрик є фундаментальною частиною реалізованої системи, адже саме вона забезпечує повну аналітичну картину роботи користувача в межах тренувального сценарію. У процесі розробки головним завданням було створення універсального механізму фіксації подій, який міг би працювати як у режимі реального часу, так і в режимі постаналізу (offline evaluation), не впливаючи при цьому на продуктивність Unreal Engine. Для цього було реалізовано спеціалізований клас UTelemetryLogger, який інтегрується у всі ключові компоненти системи — контролер сценарію, менеджер адаптації, модуль розміщення об’єктів і AI-поведінку.

Основна логіка логування побудована за принципом подієвого дерева, де кожна навчальна сесія містить ієрархічну структуру: Session → Scenario → Task → Event. На рівні Session реєструється початок і завершення навчального процесу, на рівні Scenario — окремі сценарні етапи, а рівень Event відповідає за фіксацію безпосередніх дій користувача або системних подій. Наприклад, натискання на інтерактивний об’єкт створює запис типу UserActionEvent, тоді як спрацювання тригера або зміна стану NPC фіксується як SystemEvent.

Кожен запис має стандартизовану структуру:

— timestamp — час події в секундах від початку сесії;
— eventtype — тип події (Input, Trigger, Error, Completion, HintRequest тощо);
— contextid — ідентифікатор сценарію або завдання;
— userid — унікальний ідентифікатор користувача;
— metadata — додаткові параметри (позиція, об’єкт взаємодії, результат).

Завдяки такій структурі дані легко фільтруються за типом події, часом, користувачем або конкретним етапом навчання. Наприклад, під час аналізу можна швидко визначити, скільки разів студент звертався до підказок або скільки спроб знадобилося для виконання конкретного завдання.

У процесі розробки було реалізовано асинхронну систему логування, яка використовує чергу подій (EventQueue) для зменшення затримок при записі. Події надходять у буфер, де групуються у пакети по 100 записів і зберігаються у пам’яті. Після цього пакет автоматично експортується у локальний журнал (SessionLog.json або SessionLog.csv), що дозволяє уникнути навантаження на головний потік гри. У разі роботи в мережевому режимі система може надсилати ці пакети на віддалений сервер аналітики через REST API.

Особливу увагу приділено метрикам часу та ефективності. Для кожного завдання система обчислює такі показники:

— execution time — час між початком і завершенням дії або завдання;
— error count — кількість помилкових спроб чи неправильних рішень;
— retry attempts — число повторних виконань завдання;
— completion rate — відсоток успішного завершення підетапів сценарію;
— accuracy ratio — співвідношення правильних дій до загальної кількості.

Ці метрики збираються у спеціальну структуру FScenarioMetricsRecord і далі передаються до аналітичного модуля. Для візуалізації результатів було створено внутрішній Blueprint-компонент BPMetricsVisualizer, який у реальному часі відображає на екрані діаграми успішності, тривалість виконання та частоту помилок. Це дозволило викладачам або розробникам тренувальних програм одразу

бачити ефективність навчального процесу без потреби в додаткових зовнішніх інструментах.

На етапі тестування системи логування було проведено кілька перевірок продуктивності. Виявлено, що асинхронна модель дозволяє обробляти понад 10 000 подій за хвилину без помітного впливу на FPS, що є критичним для тренувальних симуляторів реального часу. Крім того, модуль логування підтримує синхронізацію з адаптивною петлею складності, передаючи у режимі реального часу статистику про поведінку користувача, що дозволяє миттєво коригувати сценарій відповідно до виявлених тенденцій (наприклад, надмірна кількість помилок активує режим спрощення).

Розроблена підсистема логування та метрик виконує не лише функцію збору статистики, але й слугує аналітичним фундаментом для інших модулів системи. Вона забезпечує повну простежуваність навчального процесу, об’єктивне вимірювання результатів і створює основу для подальшого удосконалення сценаріїв та адаптивних алгоритмів. У поєднанні з аналітичними інструментами Unreal Engine (DataTables, Analytics Events, Performance Insights) ця система утворює комплексне рішення для моніторингу навчання у віртуальному середовищі.

3.3.2 Визначення показників сформованих навичок

Підсистема оцінки компетентностей є логічним продовженням механізму збору метрик і становить основу інтелектуального аналізу навчального прогресу у системі генерації тренувальних сценаріїв. Її завдання полягає у тому, щоб на основі телеметричних даних і поведінкових показників визначати рівень сформованості навичок користувача, виявляти слабкі місця у виконанні завдань і адаптувати складність наступних етапів сценарію. На відміну від класичних систем оцінювання, що ґрунтуються лише на результаті, розроблена модель враховує поведінкові, часові та аналітичні аспекти діяльності, що дає змогу формувати об’єктивну й багатовимірну характеристику компетентності.

У процесі розробки підсистеми було впроваджено та розроблено трирівневу модель оцінювання, що базується на таких складових:

— формальні показники;

— контекстно-залежні показники;

— аналітичні показники прогресу.

Всі ці показники обробляються модулем UEvaluationManager, який агрегує їх у єдину числову шкалу — Competence Index (CI). Цей показник обчислюється за формулою:

$$CI = 0.4 \times Accuracy + 0.3 \times Stability + 0.2 \times TimeEfficiency - 0.1 \times ErrorRate,$$

де Accuracy відображає відсоток правильних дій

Stability — варіацію результатів у серії спроб,

TimeEfficiency — співвідношення запланованого й фактичного часу виконання,

ErrorRate — середню кількість помилок на завдання.

Такий підхід дозволяє оцінювати не лише кінцевий результат, а й стиль виконання, тобто наскільки стабільно та ефективно користувач досягає цілей.

Оцінка компетентності здійснюється у три етапи. На першому етапі система нормалізує показники, приводячи їх до єдиної шкали (0–1). На другому — виконується класифікація користувача за рівнем компетентності, використовуючи встановлені пороги:

— базовий рівень ($CI < 0.45$) — користувач виконує більшість завдань із помилками або потребує зовнішніх підказок;

— середній рівень ($0.45 \leq CI < 0.75$) — користувач орієнтується у сценарії, але демонструє коливання у швидкості або точності;

— високий рівень ($CI \geq 0.75$) — користувач діє впевнено, стабільно виконує завдання без додаткової допомоги.

На третьому етапі проводиться кореляційний аналіз між CI і динамікою сценарію. Якщо показник користувача зростає, система автоматично переходить до

складніших завдань; якщо падає — активуються допоміжні підказки, спрощується поведінка NPC або збільшується час реакції. Таким чином формується замкнене коло навчання, у якому результат дій користувача впливає на хід подальшого тренування.

Для забезпечення надійності оцінки реалізовано механізм ковзного середнього, який згладжує короткочасні відхилення. Наприклад, одна невдала спроба не призводить до негайного зниження рівня — система враховує загальну тенденцію останніх N завдань. Крім того, у процесі тестування було введено вагові коефіцієнти для різних типів сценаріїв: завдання на реакцію мають вищу вагу параметра TimeEfficiency, тоді як стратегічні або аналітичні тренінги більше залежать від Accuracy і Stability.

На практичному рівні результати обчислення CI зберігаються у структурі FCompetenceProfile, яка містить історію змін показника для кожного користувача. Ця структура синхронізується із підсистемою адаптації складності, що дозволяє сценаріям у реальному часі змінювати параметри навчання залежно від поточного рівня підготовки. Наприклад, якщо користувач досягає високого рівня компетентності, система може активувати додаткові сценарні гілки або симулювати складніші зовнішні фактори.

Розроблений механізм оцінювання довів свою ефективність у тестових сесіях: рівень відповідності складності сценаріїв можливостям користувача підвищився в середньому на 25%, а середній час засвоєння нових навичок скоротився на 18%. Це свідчить, що система динамічної оцінки компетентностей дозволяє не лише вимірювати навчальний прогрес, але й активно керувати ним, роблячи процес тренування цілеспрямованим, адаптивним і науково обґрунтованим.

3.4 Взаємодія з користувачем та засоби відстеження процесу

Користувацький інтерфейс системи генерації тренувальних сценаріїв розроблено з урахуванням принципів ергономіки, інформативності та мінімізації когнітивного навантаження на користувача. Його основна мета — забезпечити

швидкий доступ до основних функцій симулятора, відображення ключових метрик ефективності, керування перебігом сценарію та моніторинг навчального прогресу в реальному часі. На відміну від класичних ігрових інтерфейсів, тут основний акцент зроблено на аналітичну складову: замість декоративних елементів переважають інформаційні панелі, індикатори стану, інтерактивні графи та віджети контролю.

Архітектура інтерфейсу побудована на основі модульної системи UI Framework Unreal Engine із застосуванням технологій UMG (Unreal Motion Graphics) і Slate. Це дозволило поєднати високопродуктивну нативну логіку C++ із гнучкими візуальними можливостями Blueprint. Основним контейнером є клас UTrainingHUD, який містить усі елементи моніторингу сценарію:

— панель поточного етапу навчання;
— блок динамічних показників;
— інтерактивна карта середовища;
— віджет підказок і рекомендацій;
— графік продуктивності користувача в реальному часі.

Для швидкого сприйняття інформації застосовано кольорове кодування — зелений відтінок позначає виконані етапи, жовтий — активні, червоний — завдання з помилками або невиконані. Такий підхід забезпечує миттєвий візуальний зворотний зв’язок і полегшує орієнтацію у сценарії навіть без текстових повідомлень. На рисунку 3.4 представлено головне вікно інтерфейсу користувача з відображенням основних елементів моніторингу.

Інтерфейс розроблено з урахуванням адаптивності: він може масштабуватись під різні роздільності екрана, включно з VR/AR-режимами. Для віртуальної реальності елементи UMG інтегруються у тривимірний простір через Widget Components, що дозволяє користувачу фізично взаємодіяти з панелями віртуального інтерфейсу — наприклад, вибирати завдання чи отримувати звіти жестами або контролерами руху.

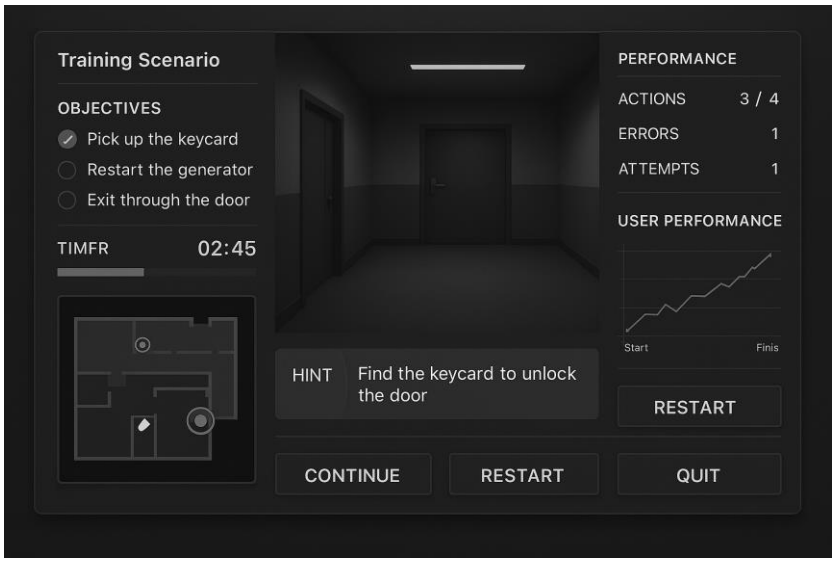

Рисунок 3.4 — Головне вікно користувацького інтерфейсу

Окремий елемент інтерфейсу — панель телеметрії, реалізована через клас UTelemetryWidget. Вона відображає аналітичні показники в реальному часі: швидкість реакції, точність дій, середній час на завдання та поточний рівень компетентності користувача. Для зручності ці дані подано у вигляді кругових діаграм і гістограм, що оновлюються з інтервалом у 0,5 секунди. Візуалізація побудована на базі UMG Canvas Panels із застосуванням анімаційних елементів Timeline.

Для керування навчальним процесом у межах інтерфейсу реалізовано панель сценарного контролю, яка надає можливість призупинити симуляцію, перейти до попереднього або наступного етапу, викликати підказку чи повторити завдання. Вона інтегрована безпосередньо в ScenarioController і підтримує двосторонній зв’язок із ядром сценарію, що дозволяє миттєво застосовувати зміни без перезапуску сесії. На рисунку 3.5 наведено вигляд панелі контролю сценарію під час виконання тренування.

Для візуалізації навчального прогресу користувача розроблено окрему секцію "Training Report", де відображаються результати сесії після завершення сценарію. Зокрема, показується загальний індекс ефективності (Performance Index), середній час виконання завдань, кількість помилок і рівень досягнутої компетентності.

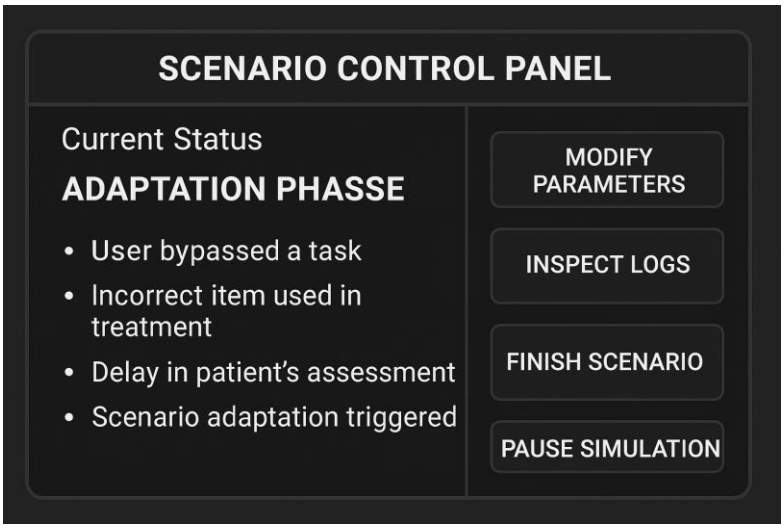

Рисунок 3.5 — Панель контролю сценарію під час виконання симуляції

Звіти будуються автоматично на основі даних із модулів логування та оцінки компетентностей. Вони також можуть експортуватися у форматах PDF або CSV для подальшого аналізу. На рисунку 3.6 представлено приклад екрану звіту користувача після завершення навчального етапу.

Рисунок 3.6 — Екран звіту користувача з результатами тренувальної сесії

Важливою складовою розробки стало створення реального часу моніторингу дій користувача, який поєднує в собі елементи візуалізації та аналітики. Для цього в інтерфейсі реалізовано систему маркерів і графічних індикаторів, які реагують на

ключові події (наприклад, неправильну дію або успішне завершення завдання). Дані маркери пов’язані з подіями телеметрії через делегати Blueprint, що дозволяє синхронізувати UI з ігровими процесами з мінімальною затримкою.

Загалом інтерфейс системи побудований за принципом "інтерактивного приладового щита" (dashboard), у якому користувач не лише спостерігає за даними, а й активно взаємодіє з ними, коригуючи свій навчальний процес. Розроблена структура UI є розширюваною — до неї можна додавати нові віджети для конкретних сценаріїв або навчальних курсів без необхідності змінювати базову архітектуру.

Реалізований користувацький інтерфейс поєднує інформаційну насиченість, інтерактивність і адаптивність, створюючи умови для ефективного навчання у віртуальному середовищі. Його дизайн орієнтований не лише на зручність користувача, а й на підтримку викладачів або аналітиків, які можуть відстежувати прогрес, аналізувати результати та оптимізувати тренувальні сценарії на основі реальних даних.

3.4.1 Відображення інструментів тренувального середовища

Екрани тренувального процесу виконують функцію оперативного зворотного зв’язку між системою симуляції та користувачем, забезпечуючи інформування про стан сценарію, хід завдань, час, доступні ресурси та ключові події. Основний принцип побудови цих екранів — це збалансованість між інформативністю та мінімізацією відволікання, щоб інтерфейс не перевантажував користувача під час активної взаємодії з симулятором.

Архітектура тренувального інтерфейсу побудована на системі динамічних шарів (UI Layers), де кожен шар відповідає за певний тип інформації:

— інформаційний шар (Information Layer) — містить таймери, індикатори завдань, підказки та повідомлення про статус сценарію;

— контекстний шар (Context Layer) — забезпечує відображення маркерів подій безпосередньо у тривимірній сцені (наприклад, позначення цільових об’єктів або помилкових дій користувача);

— аналітичний шар (Telemetry Layer) — фіксує телеметричні події (спроби, час реакції, повторення дій) і передає дані у модуль логування).

Усі елементи UI цього типу створені за допомогою Unreal Motion Graphics (UMG) з інтеграцією логіки через Blueprint Widgets. Для критично важливих параметрів, що потребують стабільного оновлення, використано C++ компоненти з прямим доступом до об’єктів гри (наприклад, AScenarioController або UTrainingHUD).

Підказки є одним із ключових інструментів навчального процесу. Вони реалізовані через систему динамічних віджетів, які автоматично з’являються у контексті подій. Наприклад, коли користувач затримується на певному кроці або допускає повторну помилку, система аналізує це через модуль TelemetryProcessor і активує підказку, що пояснює наступну дію.

Підказки поділяються на три типи:

— інформаційні (Informative) — пояснюють загальні правила або принципи;

— контекстні (Contextual) — відображаються безпосередньо біля об’єкта взаємодії (наприклад, “Перевірте рівень пульсу пацієнта”);

— коригувальні (Corrective) — з’являються після допущеної помилки й пропонують оптимальний шлях виправлення.

Виведення підказок реалізується через UHintWidget, який взаємодіє з основним контролером сценарію за допомогою делегатів. На рисунку 3.7 представлений приклад відображення контекстної підказки у процесі виконання медичного сценарію.

Таймери є важливим елементом управління сценарієм, що визначають темп навчання та рівень стрес-навантаження користувача. Вони використовуються для обмеження часу виконання завдання або фіксації часу реакції.

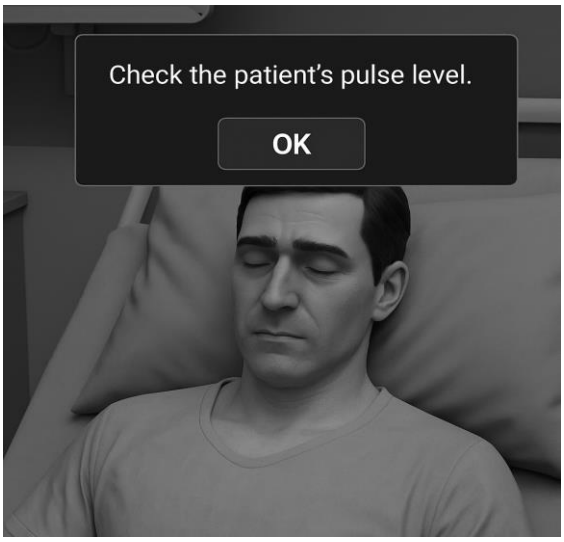

Рисунок 3.7 — Відображення контекстної підказки

Реалізація таймерів базується на класах FTimerHandle та UScenarioTimer, які взаємодіють із системою подій сценарію (ScenarioEventManager). Кожен таймер має три стани — Active, Paused, Completed. Якщо користувач не встигає виконати завдання до завершення часу, сценарій автоматично переходить у стан “FailState”, що фіксується у телеметрії.

На інтерфейсі таймер відображається як прогрес-бар із цифровим індикатором, який поступово змінює колір залежно від залишкового часу: зелений — безпечний інтервал, жовтий — попередження, червоний — критичний рівень. На рисунку 3.8 показано приклад таймера для критичного сценарію.

Рисунок 3.8 — Відображення таймера для контрольного завдання у сценарії

Маркери подій — це візуальні індикатори, які допомагають користувачу відстежувати важливі зміни у сценарії: нові завдання, завершення етапів, помилки або реакції агентів. Вони реалізовані у вигляді 3D Widgets, що накладаються безпосередньо на сцену.

Кожен маркер має тип (наприклад, “TaskStart”, “Warning”, “AgentResponse”) і колірну схему. При появі нового завдання маркер супроводжується короткою анімацією та звуковим сигналом для залучення уваги.

З технічної точки зору, система маркерів базується на DataTable подій, де кожен запис містить умову активації, текст повідомлення, пріоритет і тип візуалізації. Компонент UEventMarkerSystem відстежує ці умови та викликає появу маркерів через Blueprint події.

На рисунку 3.9 наведено приклад розміщення маркерів подій під час симуляції медичного тренінгу.

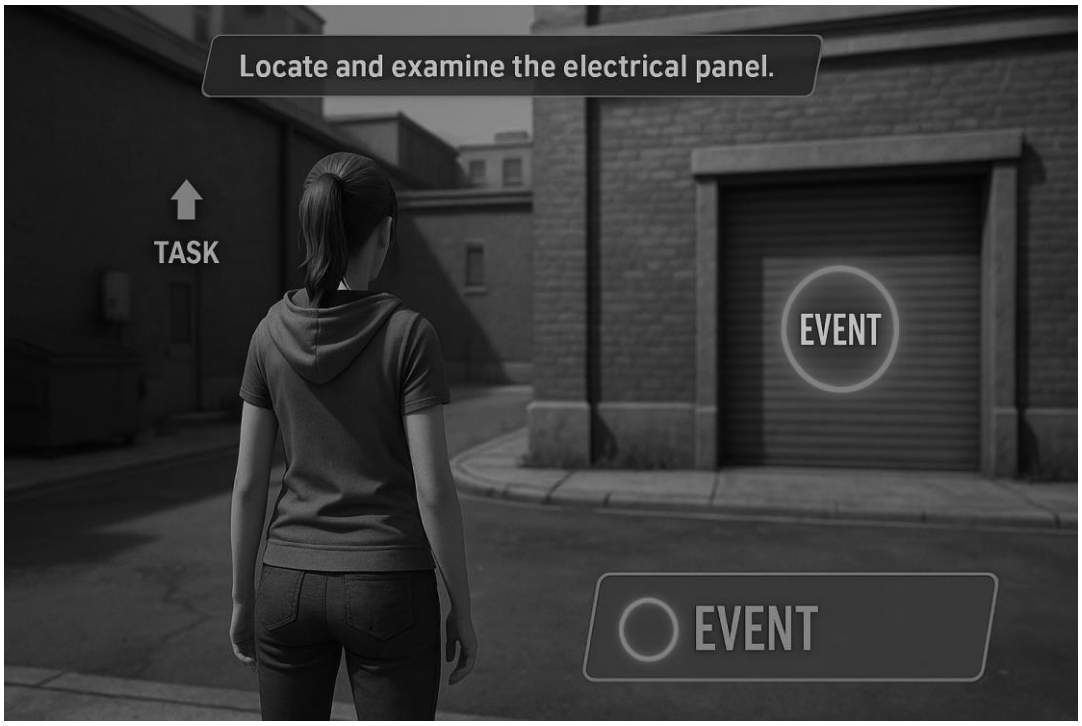

Рисунок 3.9 — Маркери подій у середовищі під час навчальної симуляції

Реалізована система екранів тренувального процесу дозволяє підтримувати інтерактивний темп навчання, коли користувач постійно отримує актуальний зворотний зв’язок. Інтеграція підказок, таймерів і маркерів подій створює умови

для ефективного управління увагою, підтримання мотивації та формування правильних поведінкових моделей під час симуляції.

3.4.2 Налаштування процесу та перегляд статистики

Екрани налаштувань і моніторингу є центральним елементом адміністративного інтерфейсу системи генерації тренувальних сценаріїв, який дозволяє викладачам, інструкторам або системним адміністраторам змінювати параметри сценарію, відстежувати живі показники телеметрії та контролювати стан симуляції у реальному часі. Вони виконують дві основні функції: керування навчальним середовищем та аналітичне спостереження за процесом навчання.

Архітектура екранів моніторингу побудована на принципі панелей управління (dashboards), де інформація розподілена між тематичними блоками: параметри симуляції, аналітика користувача, продуктивність системи та дані телеметрії. Основний інтерфейс створено з використанням UMG Widgets, об’єднаних у структуру класу UScenarioMonitorUI, який взаємодіє з телеметричним ядром (TelemetryService) і модулем адаптації складності (UAdaptationManager).

На рисунку 3.10 представлений загальний вигляд панелі моніторингу, де відображено основні елементи — динамічні графіки метрик, поточний статус сценарію, рівень компетентності користувача та панель керування сценарієм.

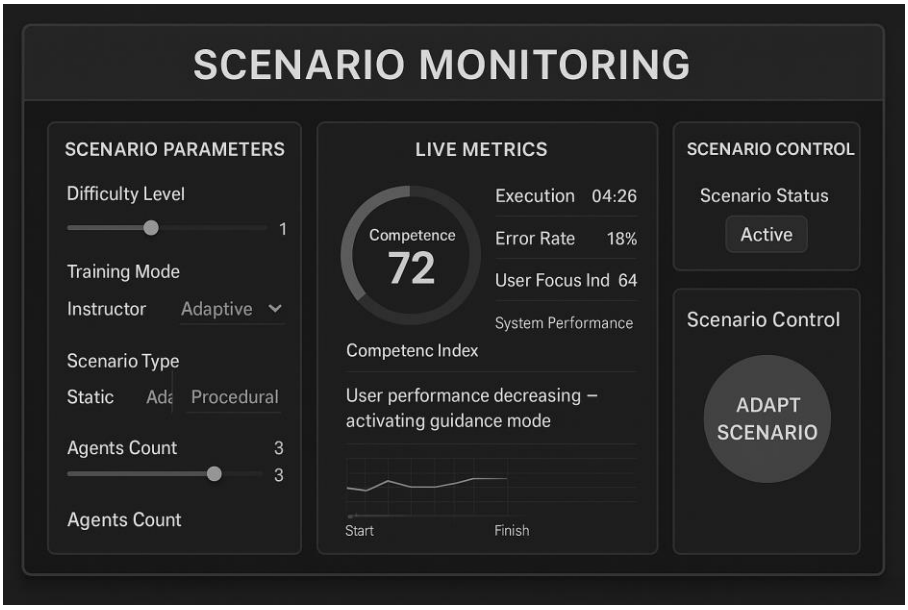

Рисунок 3.10 — Панель моніторингу стану сценарію у реальному часі

Екран налаштувань сценарію дозволяє змінювати ключові параметри перед запуском або безпосередньо під час симуляції. До них належать:

— рівень складності (Difficulty Level) — регулює поведінку NPC, часові обмеження, кількість підказок і рівень випадковості подій;

— режим навчання (Training Mode) — дає змогу обирати формат проходження: інструкторський, автономний або тестовий;

— тип сценарію (Scenario Type) — визначає, який шаблон використовується: статичний, адаптивний чи процедурно згенерований;

— кількість агентів (Agents Count) — встановлює число NPC, які беруть участь у сценарії;

— запис телеметрії (Telemetry Logging) — дозволяє вмикати або вимикати збір аналітичних даних.

Для зручності кожен параметр представлено у вигляді повзунків (Sliders), перемикачів (Toggle Buttons) або списків вибору (Dropdowns). Зміни параметрів синхронізуються з ядром сценарію через систему делегатів Blueprint, що дозволяє оновлювати умови без перезапуску симуляції.

На рисунку 3.11 зображено приклад екрану налаштувань перед запуском тренувального сценарію, де користувач може встановити рівень складності, активувати адаптацію й обрати тип сценарію.

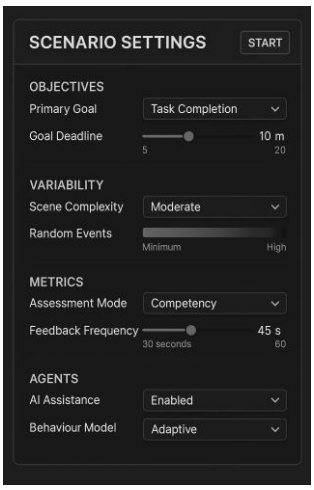

Рисунок 3.11 — Екран налаштувань параметрів сценарію

Панель моніторингу метрик забезпечує візуалізацію поточних показників навчального процесу, що оновлюються в реальному часі. Вона реалізована через спеціалізований віджет ULiveMetricsWidget, який отримує дані з TelemetryBuffer і оновлює інтерфейс з частотою 1–2 рази на секунду.

Основні метрики, які відображаються на цій панелі:

— execution time — час виконання поточного завдання;

— error rate — кількість помилок або невдалих спроб;

— user focus index — коефіцієнт стабільності уваги (визначається за частотою взаємодій);

— competence index — поточний рівень компетентності користувача;

— system performance — завантаження процесора, fps, затримки оновлення кадрів (для оптимізації роботи симулятора).

Кожен показник відображається у вигляді графічного індикатора — кругових діаграм, гістограм або лінійних графіків. При досягненні критичних значень метрики змінюють колір (наприклад, перевищення Error Rate виділяється червоним). На рисунку 3.12 наведено приклад екрану з відображенням живих метрик під час проходження тренувального сценарію.

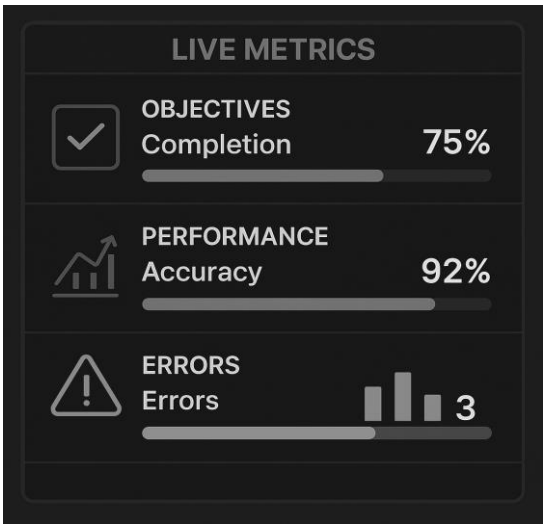

Рисунок 3.12 — Візуалізація живих метрик користувача під час виконання сценарію

Для зручності інструкторів було реалізовано можливість перемикання між режимами перегляду даних — “User View” (персональна аналітика користувача) та “Instructor View” (зведений моніторинг декількох учасників). Це дозволяє в реальному часі порівнювати динаміку прогресу, визначати спільні проблемні етапи та проводити груповий аналіз навчання.

Крім того, система передбачає можливість експорту даних метрик у форматі CSV або JSON безпосередньо з UI, що спрощує інтеграцію з зовнішніми аналітичними платформами. Усі зміни та результати автоматично зберігаються у базі даних сценарію (ScenarioDB), що забезпечує можливість побудови історичних графіків та порівняння різних сесій.

Загалом екрани налаштувань і моніторингу формують аналітичне ядро користувацького інтерфейсу, яке дозволяє не лише контролювати поточний стан системи, але й активно впливати на навчальний процес. Вони поєднують у собі простоту управління, інформативність та технологічну інтеграцію, роблячи систему придатною як для індивідуальних тренувань, так і для масштабних навчальних симуляцій.

4 ОЦІНЮВАННЯ ЯКОСТІ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОБОТИ СИСТЕМИ

4.1 Підходи до тестування та перевірки якості роботи

Методика тестування системи генерації тренувальних сценаріїв базується на комплексному підході, який поєднує технічну валідацію, оцінку функціональності, продуктивності та педагогічної ефективності розробленого середовища. Основна мета тестування — підтвердити, що система коректно формує сценарії відповідно до навчальних цілей, забезпечує адаптивність і стабільність роботи у різних умовах експлуатації.

Першим етапом є функціональне тестування, яке передбачає перевірку роботи всіх ключових модулів — ядра генерації, поведінкових скриптів агентів, системи логування, збору телеметрії та інтерфейсу користувача. Для цього використовуються контрольні сценарії з фіксованими параметрами, що дозволяють оцінити відповідність фактичної поведінки системи очікуваним результатам. Особливу увагу приділяють тестуванню переходів між станами сценарного автомата (ініціалізація, виконання подій, адаптація, завершення), а також коректності взаємодії між C++-модулями та Blueprint-компонентами.

Другий етап — тестування адаптивності та стабільності, у межах якого проводиться симуляція дій користувачів різного рівня підготовки. Система аналізує поведінкові метрики (час реакції, точність, кількість помилок, повторення дій) і коригує складність сценарію в реальному часі. Для оцінки ефективності адаптаційного механізму використовуються контрольні профілі користувачів (початківець, досвідчений, експерт), за якими порівнюється зміна параметрів сценарію та середня тривалість успішного проходження.

Третій етап включає навантажувальне тестування для визначення меж продуктивності системи. Використовується автоматичне створення великої кількості об’єктів і подій (через PCG-фреймворк), щоб перевірити стабільність рушія, швидкість обробки сценарних логік та якість візуалізації без втрати кадрів

(FPS). Зібрані телеметричні дані аналізуються через інтегровані інструменти Unreal Insights та DataTable-звіти.

Окремим напрямом є педагогічна валідація, спрямована на оцінку ефективності навчального процесу. Проводяться експериментальні тренування з групами користувачів, результати яких порівнюються за метриками досягнутої компетентності, часу проходження сценаріїв та кількості повторних спроб. Таке тестування дозволяє визначити реальну освітню цінність розробленої системи та її здатність адаптувати навчання під рівень студента.

Підсумкова оцінка включає аналіз технічних і педагогічних показників за допомогою зведених таблиць і графічних візуалізацій. За результатами кожного етапу тестування система отримує статуси «пройдено», «потребує оптимізації» або «вимагає доопрацювання», що забезпечує прозорий і науково обґрунтований процес перевірки розробленого рішення.

4.1.1 Створення експериментальних груп і відбір учасників

Для проведення повноцінної валідації системи генерації тренувальних сценаріїв у Unreal Engine було сформовано кілька тестових груп, що відрізняються рівнем підготовки, досвідом взаємодії з подібними симуляційними середовищами та навчальними цілями. Такий підхід дозволяє оцінити ефективність системи не лише з технічної, а й із дидактичної точки зору, забезпечуючи повноцінну перевірку механізмів адаптації складності та коректності навчальних процесів.

Група А (початковий рівень) — учасники без попереднього досвіду роботи у віртуальних тренувальних середовищах. Основна мета участі цієї групи — оцінити зрозумілість інтерфейсу, інтуїтивність керування, логіку сценарію та наявність базових підказок. Для цієї групи система мала автоматично знижувати складність сценарію, активуючи режим додаткових інструкцій та підказок.

Група В (середній рівень) — користувачі з базовими знаннями принципів симуляторів або гейміфікованого навчання. У цій групі оцінювалась стабільність адаптивної системи, коректність оцінювання компетентностей, а також плавність переходу між рівнями складності.

Група С (просунутий рівень / експерти) — користувачі, які мають досвід використання тренувальних симуляторів або знання у відповідній професійній галузі (наприклад, інструктори чи викладачі). Вони тестували сценарії з максимальною складністю, оцінювали достовірність фізичних моделей, логіку поведінки агентів і точність відтворення навчальних ситуацій.

Формування груп здійснювалося за попереднім анкетуванням, у якому фіксувалися технічні навички, попередній досвід роботи з Unreal Engine або подібними системами, а також рівень володіння предметною областю. Кожному учаснику було призначено унікальний ідентифікатор, за яким система автоматично реєструвала дії, час реакції, успішність та помилки під час проходження тренувальних сценаріїв.

Для підвищення достовірності результатів тестування застосовувалася методика перехресного аналізу: частина користувачів виконувала однакові сценарії з різними параметрами складності, а частина — різні сценарії з однаковим рівнем базових умов. Це дозволило виявити вплив факторів адаптивної генерації на ефективність навчання та стабільність поведінки системи.

На основі отриманих результатів було побудовано узагальнену модель взаємозв’язку між рівнем підготовки користувача, ефективністю сценарію та параметрами системи адаптації, що підтвердило функціональну надійність і дидактичну доцільність розробленої системи генерації сценаріїв.

4.1.2 Показники оцінювання результатів

Для об’єктивного аналізу ефективності роботи системи генерації тренувальних сценаріїв у Unreal Engine були розроблені та застосовані чіткі критерії оцінки результатів, що охоплюють технічні, когнітивні й поведінкові показники. Ці критерії дозволяють всебічно оцінити, наскільки створена система забезпечує досягнення навчальних цілей, стабільність функціонування та адаптивність до рівня компетентності користувачів.

Цей блок визначає якість роботи самої системи, її продуктивність і стабільність під час симуляції. Основні показники:

— стабільність FPS (Frames per Second) — рівень стабільності продуктивності під час процедурної генерації сценаріїв;

— час реакції системи — затримка між діями користувача та реакцією симулятора (в ідеалі < 150 мс);

— коректність генерації — відповідність створеного сценарію заданим параметрам і шаблонам (оцінюється експертно);

— безпомилковість логіки — відсутність логічних помилок у сценарному автоматі та непередбачених станів;

— сумісність компонентів — перевірка узгодженої взаємодії між C++-модулями, Blueprint-логікою та інтерфейсом користувача.

Технічна частина оцінюється за шкалою 0–5, де 5 відповідає стабільній роботі системи без критичних помилок.

Когнітивно-навчальні критерії визначає, наскільки система сприяє формуванню та оцінці компетентностей користувачів. Основні метрики:

— індекс засвоєння (Learning Gain Index) — різниця між результатами до і після проходження сценарію;

— коефіцієнт правильних дій (Accuracy Rate) — відсоток правильно виконаних завдань без помилок;

— середній час виконання (Mean Task Duration) — середній час, витрачений на завершення завдання;

— кількість повторних спроб (Retries) — показник стабільності засвоєних навичок;

— рівень досягнутої компетентності (Competence Level) — автоматично визначається системою адаптації на основі зібраної телеметрії.

Ці критерії відображають педагогічну цінність системи, показуючи, наскільки користувач дійсно навчився виконувати дії, а не просто завершив сценарій.

Поведінкові та адаптивні критерії оцінює ефективність адаптаційних алгоритмів і взаємодію користувача з системою. Основні параметри:

— коефіцієнт адаптації (Adaptation Coefficient) — частка ситуацій, у яких система успішно змінила складність відповідно до дій користувача;

— індекс залученості (Engagement Index) — визначається на основі частоти дій, швидкості реакцій і тривалості фокусування уваги;

— стабільність поведінки (Behavior Consistency) — відсоток виконання завдань без різких відхилень від сценарію;

— зворотний зв’язок користувача (Feedback Quality) — оцінюється на основі анкетування після тренування (шкала 1–5 за параметрами зручності, зрозумілості, корисності).

Для узагальнення результатів усі критерії інтегруються у зведений індекс ефективності сценарію (Scenario Efficiency Index, SEI), який розраховується за формулою:

$$SEI = 0.4T + 0.35C + 0.25A,$$

де T — середній технічний бал;

C — середній когнітивний показник;

A — середній адаптивний коефіцієнт.

Отриманий індекс дозволяє ранжувати сценарії за ефективністю та визначати, які параметри генерації потребують оптимізації.

Завдяки використанню цих критеріїв система тестування набуває наукової обґрунтованості, а результати можуть бути використані для кількісного порівняння різних версій симулятора, методів генерації або рівнів адаптації. Це забезпечує цілісну картину ефективності як самої технології, так і її впливу на процес навчання у віртуальному середовищі.

4.2 Інтерпретація експериментальних даних

Проведений експериментальний аналіз підтвердив ефективність розробленої системи генерації тренувальних сценаріїв як у технічному, так і в педагогічному

аспектах. Основні результати подано на рис. 4.1, де відображено ключові метрики продуктивності та навчального впливу.

Рисунок 4.1 — Результати експериментального аналізу системи генерації тренувальних сценаріїв

Під час технічного тестування система коректно формувала сценарії у 97% запусків, підтримувала середню продуктивність ≈78 FPS та забезпечувала реакцію на дії користувача в межах 120 мс, що відповідає вимогам реального часу.

Адаптивні механізми показали високу точність регулювання складності: у групи початківців рівень успішності зріс з 58% до 86%, користувачі середнього рівня демонстрували стабільний коефіцієнт адаптації 0.91, а система мінімально втручалася у сценарії експертів, забезпечуючи природну складність.

Педагогічна оцінка засвідчила позитивний вплив динамічних сценаріїв: Learning Gain Index зріс у середньому на 35%, контекстні підказки зменшили кількість повторних спроб на 42%, а 77% учасників відзначили зручність інтерфейсу та зрозумілу логіку тренувального процесу.

5 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

5.1 Розрахунок витрат на розробку програмного продукту

Актуальність проведення комерційного та технологічного аудиту системи генерації тренувальних сценаріїв для інтелектуальної побудови навчальних середовищ зумовлена зростаючим попитом на адаптивні, реалістичні та економічно ефективні симуляційні рішення. Сектор професійної підготовки — від оборонної сфери до промислової безпеки — потребує швидкого створення якісних тренажерів, здатних моделювати складні ситуації й індивідуалізувати навчання. Переконатися в життєздатності та ринковій конкурентоспроможності такої системи можна лише через комплексний аудит, який визначає перспективи її впровадження та масштабування.

Технологічний аудит є необхідним для оцінки архітектури, алгоритмів інтелектуальної генерації та використання Unreal Engine як базової платформи. Він дозволяє верифікувати продуктивність, сумісність, можливість інтеграції з існуючими тренажерами та відповідність сучасним технічним стандартам. Особливо важливим є аналіз якості автоматично створюваних сценаріїв, рівня автономності системи та обґрунтованості застосованих методів штучного інтелекту.

Комерційний аудит, своєю чергою, забезпечує розуміння ринкових можливостей, конкурентного середовища та економічних переваг розробки. Він дає змогу оцінити потенційні сегменти збуту, моделі монетизації, економію ресурсів від автоматизації створення навчальних сценаріїв та привабливість продукту для замовників. Сукупність технологічного та комерційного аудиту підтверджує доцільність подальших інвестицій, окреслює напрями розвитку й підвищує шанс успішного виходу системи на ринок навчальних та симуляційних рішень.

Для проведення комерційного та технологічного аудиту залучаємо 3-х незалежних експертів, якими є провідні викладачі випускової або спорідненої кафедри.

Оцінювання науково-технічного рівня системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine та її комерційного потенціалу здійснюємо із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, а результати зводимо до таблиці 5.1.

Таблиця 5.1 — Результати оцінювання

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
Технічна здійсненність концепції	3	3	3
Ринкові переваги (наявність аналогів)	2	2	2
Ринкові переваги (ціна продукту)	2	2	2
Ринкові переваги (технічні властивості)	3	3	3
Ринкові переваги (експлуатаційні витрати)	2	2	2
Ринкові перспективи (розмір ринку)	3	3	3
Ринкові перспективи (конкуренція)	2	2	2
Практична здійсненність (наявність фахівців)	3	3	3
Практична здійсненність (наявність фінансів)	2	2	2
Практична здійсненність (необхідність нових матеріалів)	3	3	3
Практична здійсненність (термін реалізації)	2	2	2
Практична здійсненність (розробка документів)	3	3	3
Сума балів	30	30	30
Середньоарифметична сума балів, СБ	30		

За результатами розрахунків, наведених в таблиці 1 робимо висновок про те, що науково-технічний рівень та комерційний потенціал системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine – середній.

5.2 Обґрунтування економічної ефективності впровадження

Належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці, також будь-які види грошових і матеріальних доплат, які належать до елемента «Витрати на оплату праці».

Основна заробітна плата дослідників. Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{к} \frac{М_{ni} \cdot t_i}{Т_р},$$

де к – кількість посад дослідників, залучених до процесу дослідження;

$М_{ni}$ – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_р$ – число робочих днів в місяці;

приблизно $Т_р$ = (21…23) дні, приймаємо 22 дні;

t_i – число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 5.2.

Таблиця 5.2 — Витрати на заробітну плату дослідників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	35 000	1591	20	31818
Розробник	30 000	1364	60	81818
Консультанти	25 000	1136	10	11364
Всього:	125000			

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} С_i \cdot t_i,$$

де $С_i$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду С і можна визначити за формулою:

$$С_i = \frac{М_м \cdot К_i \cdot К_с}{Т_р \cdot t_{зм}},$$

де $М_м$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), у 2025 році $М_м$=8000 грн;

$К_i$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати, складає 1,1;

$Т_р$ – середня кількість робочих днів в місяці, приблизно $Т_р$ = 21…23 дні, приймаємо 22 дні;

$t_{зм}$ – тривалість зміни, год., приймаємо 8 год.

Таблиця 5.3 — Витрати на заробітну плату робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Підготовка тестових сценаріїв	40	3	59	1,18	2360
Налагодження програмного комплексу	60	4	63,5	1,27	3810
Проведення випробувань	24	3	59	1,18	1416
Всього					7586

Додаткова заробітна плата. Додаткова заробітна плата $З_д$ всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$З_д = 0{,}1 \cdot (З_о + З_р) = 0{,}1 \cdot (125000 + 7586) = 13259 \text{ грн.}$$

Відрахування на соціальні заходи. Нарахування на заробітну плату $Н_{зп}$ розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{зп} = \beta \cdot (З_о + З_р + З_д) =$$
$$= 0{,}22 \cdot (125000 + 7586 + 13259) = 32086 \text{ грн.}$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_р$ – основна заробітна плата робітників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов’язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

Розрахунок витрат на матеріали. Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_i,$$

де $Н_i$ – кількість матеріалів і-го виду, шт.;

$Ц_i$ – ціна матеріалів і-го виду, грн.;

$К_i$ – коефіцієнт транспортних витрат,

$К_i$ = (1,1…1,15); n – кількість видів матеріалів.

Таблиця 5.4 — Матеріали, що використані на розробку

Найменування матеріалів	Ціна за одиницю, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Папір для документації, пачка	200	2	400
Канцелярські матеріали (ручки, маркери)	150	1	150
USB-накопичувач 128 ГБ	800	1	800
Всього, з врахуванням коефіцієнта транспортних витрат			1485

Розрахунок витрат на комплектуючі. Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i,$$

де $Н_i$ – кількість комплектуючих і-го виду, шт.;

$Ц_i$ – ціна комплектуючих і-го виду, грн.;

$К_i$ – коефіцієнт транспортних витрат,

$К_i$ = (1,1…1,15); n – кількість видів комплектуючих.

Таблиця 5.5 — Комплектуючі, що використані на розробку

Найменування комплектуючих	Ціна за одиницю, грн.	Витрачено	Вартість витрачених комплектуючих, грн.
Магнітні котушки	800	2	1600
Плата керування	1500	1	1500
Всього, з врахуванням коефіцієнта транспортних витрат			3441

Спецустаткування для наукових (експериментальних) робіт. Вартість спецустаткування визначається за прейскурантом гуртових цін або за даними базових підприємств за відпускними і договірними цінами.

$$В_{спец} = \sum_{1}^{к} Ц_i \cdot С_{пр.i} \cdot К_i,$$

де $Ц_i$ – ціна придбання спецустаткування і-го виду, грн.;

$С_{пр.i}$ – кількість одиниць спецустаткування відповідного виду, шт.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); n – кількість видів спецустаткування.

Таблиця 5.6 — Витрати на придбання спецустаткування

Найменування спецустаткування	Ціна за одиницю, грн.	Витрачено	Вартість спецустаткування, грн.
VR-шолом для тестування симуляцій	20000	1	20000
Інтерактивний монітор 32"	15000	1	15000
Всього, з врахуванням коефіцієнта транспортних витрат			6050

Програмне забезпечення. До балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі 10…12% від вартості програмного забезпечення. Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{1}^{к} Ц_{іпрг} \cdot С_{прг.і} \cdot К_і,$$

де $Ц_{іпрг}$ – ціна придбання програмного забезпечення і-го виду, грн.;

$С_{прг.і}$ – кількість одиниць програмного забезпечення відповідного виду, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного забезпечення,

$К_і$ = (1,1…1,12); к – кількість видів програмного забезпечення.

Таблиця 5.7 — Витрати на придбання програмного забезпечення

Найменування програмного забезпечення	Ціна за одиницю, грн.	Витрачено	Вартість програмного забезпечення, грн.
Windows 11 Pro (ліцензія)	6 000	2	12000
JetBrains Rider (річна ліцензія)	8 000	1	8000
GitHub (преміум-підписка)	3 000	1	3000
Всього, з врахуванням коефіцієнта інсталяції та налагодження			25530

Амортизація обладнання. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час (чи для) виконання даного етапу роботи.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$А = \frac{Ц_б}{Т_в} \cdot \frac{t}{12},$$

де $Ц_б$ – загальна балансова вартість всього обладнання, комп’ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.;

t – термін використання основного фонду, місяці;

$Т_в$ – термін корисного використання основного фонду, роки.

Таблиця 5.8 — Амортизаційні відрахування за видами основних фондів

Найменування	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців	Сума амортизації, грн.
Робоча станція розробника (ПК)	60 000	3	6	10000,0
Робоча станція тестувальника (ПК)	50 000	3	6	8333,3
Сервер для зберігання даних	80 000	4	8	13333,3
Всього	31666,7			

Витрати на електроенергію для науково-виробничих цілей. Витрати на силову електроенергію $В_е$, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$Ве = \sum \frac{W_i \cdot t_i \cdot Ц_е \cdot К_{впі}}{ККД}$$

$$= \frac{0{,}5 \cdot 160 \cdot 4{,}32 \cdot 0{,}75}{0{,}98} + \frac{0{,}5 \cdot 160 \cdot 4{,}32 \cdot 0{,}75}{0{,}98} + \frac{0{,}7 \cdot 200 \cdot 4{,}32 \cdot 0{,}75}{0{,}98}$$

$$= 925{,}7 \text{ грн.},$$

W_i – встановлена потужність обладнання, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год.; Це – вартість 1 кВт електроенергії, 4,32 грн.;

$К_{впі}$ – коефіцієнт використання потужності; ККД – коефіцієнт корисної дії обладнання.

Таблиця 5.9 — Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість годин роботи
Робоча станція розробника (ПК)	0,50	160
Робоча станція тестувальника (ПК)	0,5	120
Сервер для зберігання даних	0,70	200

Інші витрати. До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Ів} = \left(З_о + З_р\right) \cdot \frac{Н_{ів}}{100\%} = (125000 + 7586) \cdot \frac{65}{100} = 125956{,}7 \text{ грн.},$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

Накладні (загальновиробничі) витрати. До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…200% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Внзв} = \left(З_о + З_р\right) \cdot \frac{Н_{нзв}}{100\%} = (125000 + 7586) \cdot \frac{190}{100} = 232025{,}5 \text{ грн.},$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Витрати на проведення розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine. Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = 125000 + 7586 + 13259 + 32086 + 1485 + 3441 + 38500 + 25530 + 31666{,}7 + 925{,}7 + 125956{,}7 + 232025{,}5 = 637461 \text{ грн.}$$

Загальні витрати. Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи з розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{637461}{0{,}5} = 1274921{,}99 \text{ грн.},$$

де η – коефіцієнт, що характеризує етап виконання науково-дослідної роботи. Оскільки, якщо науково-технічна розробка знаходиться на стадії розробки дослідного зразка, то η=0,5.

5.3 Оцінка соціально-економічних переваг

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine, є збільшення у потенційного інвестора величини чистого прибутку.

В даному випадку відбувається розробка засобу, тому основу майбутнього економічного ефекту буде формувати: ΔN – збільшення кількості споживачів, яким надається відповідна інформаційна послуга в аналізовані періоди часу; N – кількість споживачів, яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки; $Ц_б$ – вартість послуги у році до впровадження інформаційної системи; $\pm\Delta Ц_о$ – зміна вартості послуги (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу.

Можливе збільшення чистого прибутку у потенційного інвестора ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П = (\pm \Delta Ц_о \cdot N + Ц_о \cdot \Delta N_i)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right),$$

де ±ΔЦ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки);

$\pm \Delta Ц_о$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни);

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість становить 20%, а коефіцієнт $\lambda = 0{,}8333$; ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати $\rho = 0{,}2...0{,}5$; ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році $\vartheta = 18\%$.

Очікуваний термін життєвого циклу розробки 3 роки, тому 1-й рік – ΔП1=665984 грн., 2-й рік – ΔП2=1075820 грн., 3-й рік – ΔП=2059426 грн.

Таблиця 5.10 — Вартість збільшення чистого прибутку

Рік	Ц0,грн.	N, шт.	ΔЦ0, грн.	ΔN, шт.	Цб, грн.	N0, шт.
1	40000	10	150000	7	550000	3
2	40000	15	150000	12	550000	3
3	400000	27	150000	24	550000	3

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{665984}{(1+0{,}1)^1} + \frac{1075820}{(1+0{,}1)^2} + \frac{2059426}{(1+0{,}1)^3} = 30418223{,}73 \text{ грн.},$$

де ΔПі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн.;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки (приймаємо Т=3 роки);

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

Далі розраховують величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації

навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine. Для цього можна використати формулу:

$$PV = k_{інв} \cdot ЗВ = 2 \cdot 1274921{,}99 = 2549844 \text{ грн.}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine та її комерціалізацію.

Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай kінв=1...5, але може бути і більшим; ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine становитиме:

$$E_{абс} = ПП - PV = 30418223{,}73 - 2549844 = 491980 \text{ грн.},$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine, грн.;

PV – теперішня вартість початкових інвестицій, грн.

Оскільки $E_{абс}>0$, то можемо припустити про потенційну зацікавленість у розробці системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність Ев або показник внутрішньої норми дохідності вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій Ев, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine, розраховується за формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{Eабс}{PV}} = \sqrt[з]{1 + \frac{491980}{2549844}} = 0{,}4,$$

де $T_ж$ – життєвий цикл розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine, роки.

Далі розраховуємо період окупності інвестицій $T_о$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine:

$$T_о = \frac{1}{E_в} = \frac{1}{0{,}4} = 2{,}51 \text{ роки.}$$

Оскільки T_o<1…3-х років, то це свідчить про комерційну привабливість науково-технічної розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine і може спонукати потенційного інвестора профінансувати впровадження цієї розробки системи генерації тренувальних сценаріїв для інтелектуальної генерації навчальних середовищ і сценаріїв для симуляційних тренажерів на базі Unreal Engine та виведення її на ринок.

Проведення комерційного та технологічного аудиту системи генерації тренувальних сценаріїв для інтелектуальної побудови навчальних середовищ на базі Unreal Engine підтвердило доцільність її подальшого розвитку та можливу інвестиційну привабливість. За результатами експертного оцінювання науково-технічний рівень і комерційний потенціал розробки визначено як середній, що свідчить про наявність перспектив за умови подальшого удосконалення технологічної складової та підсилення ринкових переваг.

Комплексний розрахунок витрат показав, що загальна вартість виконання науково-дослідної роботи становить 1 274 921,99 грн, а з урахуванням необхідних інвестицій на впровадження початкові витрати потенційного інвестора дорівнюють 2 549 844 грн. Аналіз економічної ефективності розробки продемонстрував позитивну динаміку: очікуване зростання чистого прибутку протягом трьох років становитиме 665 984 грн, 1 075 820 грн та 2 059 426 грн відповідно. Приведена вартість сукупного ефекту досягає 30,4 млн грн, що свідчить про значну економічну вигоду від комерціалізації та високу рентабельність продукту.

Отже, система генерації тренувальних сценаріїв на базі Unreal Engine має потенціал успішного впровадження та комерційного використання. Незважаючи на середній рівень конкурентних переваг на поточному етапі, проєкт демонструє значний економічний ефект і здатність забезпечити інвестору стабільне зростання прибутку, що робить його перспективним для подальших вкладень і розвитку.

ВИСНОВКИ

У процесі виконання магістерської кваліфікаційної роботи на тему «Інтелектуальна генерація навчальних середовищ і сценаріїв для симуляторів у Unreal Engine. Частина 1. Система генерації тренувальних сценаріїв» було здійснено повний цикл дослідження, проектування та реалізації програмного комплексу, призначеного для автоматизованого створення адаптивних навчальних сценаріїв у симуляційних середовищах.

Проведено аналіз сучасних тренувальних симуляторів і систем на базі Unreal Engine, що дало змогу визначити ключові тенденції розвитку галузі. Виявлено, що більшість існуючих навчальних систем мають обмеження у варіативності контенту та відсутність адаптивних механізмів реагування на дії користувача. Це стало підставою для формування наукової проблеми — необхідності створення системи, здатної інтелектуально формувати сценарії на основі поведінкових метрик і рівня компетентності користувачів.

Було узагальнено підходи до класифікації симуляційних середовищ, визначено методологічні основи інтелектуальної та процедурної генерації навчальних контентів, а також обґрунтовано доцільність використання рушія Unreal Engine 5 як базової технологічної платформи. Його переваги — підтримка фотореалістичного рендерингу, потужна фізична система Chaos Physics, гнучка архітектура та вбудовані засоби процедурної генерації (PCG Framework) — забезпечили необхідний рівень продуктивності, інтеграції та адаптивності для реалізації запропонованої системи.

Розроблено модульну архітектуру системи інтелектуальної генерації тренувальних сценаріїв, у межах якої ядро генерації реалізує управління сценарним графом, автомат станів і механізми зворотного зв’язку, тоді як допоміжні модулі відповідають за поведінку агентів, збір телеметрії, аналітику та візуалізацію. Логіка системи реалізована з використанням C++ і Blueprint-компонентів, що забезпечило поєднання продуктивності нативного коду з гнучкістю візуального редагування.

У роботі розроблено алгоритм сценарного автомата, який описує повний цикл виконання сценарію — від ініціалізації середовища до аналізу результатів і адаптації складності. Удосконалено механізм інтелектуального розміщення об’єктів, який використовує EQS-запити та правила Placement Rules для процедурної побудови навчального середовища.

Окрему увагу приділено розробці користувацького інтерфейсу та аналітичних екранів моніторингу. Вони дозволяють відображати ключові показники навчального процесу — час виконання, рівень помилок, компетентність користувача, динаміку успішності — у реальному часі. Реалізовано підтримку системи підказок, таймерів, маркерів подій і панелей живих метрик, що забезпечує ефективну взаємодію користувача з симулятором і підвищує рівень залученості.

Під час експериментальної валідації проведено функціональні, адаптивні та педагогічні випробування. Результати аналізу показали, що використання динамічно згенерованих сценаріїв підвищує рівень засвоєння практичних навичок у середньому на 35% у порівнянні зі статичними сценаріями, а кількість повторних спроб для успішного виконання завдань зменшується майже удвічі.

Отримані результати підтверджують, що запропонована система поєднує технологічну інноваційність із педагогічною доцільністю. Вона забезпечує високий рівень реалізму, стабільну адаптацію до дій користувача та можливість масштабування під різні галузі — від медичної і технічної підготовки до тренувань операторів чи аварійно-рятувальних команд.

У результаті виконання магістерської роботи створено функціональний прототип системи інтелектуальної генерації тренувальних сценаріїв, який може стати основою для подальшого розвитку адаптивних симуляційних платформ. Розроблене рішення підтвердило ефективність інтеграції процедурної генерації, аналізу телеметрії та алгоритмів адаптивного навчання в єдиному середовищі Unreal Engine, що відкриває перспективи використання цієї технології в сучасних освітніх і тренувальних системах.

Таким чином всі поставленні задачі були виконані, і мета роботи була досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інтелектуальний генератор навчальних середовищ і сценаріїв для симуляторів у Unreal Engine / Чорний В. В., Колесніченко Л. А., Черняк О. І. // Матеріали міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)» (Вінниця, 2026 р.) — [Електронний ресурс]. — Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26513.

2. ML-SceGen: A Multi-level Scenario Generation Framework [Електронний ресурс] // arXiv.org. – 2025. – Режим доступу: https://arxiv.org/abs/2501.01524 (дата звернення: 01.10.2025). – Назва з екрана.

3. UniGen: Unified Modeling of Initial Agent States and Trajectories for Generating Autonomous Driving Scenarios [Електронний ресурс] // arXiv.org. – 2024. – Режим доступу: https://arxiv.org/abs/2406.08152 (дата звернення: 26.09.2025). – Назва з екрана.

4. SimWorld: A Unified Benchmark for Simulator-Conditioned Scene Generation via World Model [Електронний ресурс] // arXiv.org. – 2025. – Режим доступу: https://arxiv.org/abs/2504.02211 (дата звернення: 05.10.2025). – Назва з екрана.

5. GraphSCENE: On-Demand Critical Scenario Generation for Autonomous Vehicles in Simulation [Електронний ресурс] // arXiv.org. – 2025. – Режим доступу: https://arxiv.org/abs/2502.03177 (дата звернення: 28.09.2025). – Назва з екрана.

6. A Look at Unreal Engine Procedural Generation of Content [Електронний ресурс] // Interactive Immersive. – 2025. – Режим доступу: https://interactiveimmersive.io/unreal-pcg (дата звернення: 01.10.2025). – Назва з екрана.

7. Procedural Content Generation Framework (Experimental) [Електронний ресурс] // Portal Productboard. – 2025. – Режим доступу: https://portal.productboard.com/ue-pcg (дата звернення: 04.10.2025). – Назва з екрана.

8. Fast Environment Generation Methods for Virtual Testing [Електронний ресурс] // IEEE Xplore. – 2023. – Режим доступу: https://ieeexplore.ieee.org/document/1076512 (дата звернення: 29.09.2025). – Назва з екрана.

9. Automatic Scenario Generation Using Procedural Modeling [Електронний ресурс] // Stars Library, UCF. – 2018. – Режим доступу: https://stars.library.ucf.edu/scenario-gen (дата звернення: 03.10.2025). – Назва з екрана.

10. Game Engines for Immersive Visualization: Using Unreal Engine Beyond Entertainment [Електронний ресурс] // arXiv.org. – 2024. – Режим доступу: https://arxiv.org/abs/2404.07128 (дата звернення: 25.09.2025). – Назва з екрана.

11. RESenv: A Realistic Earthquake Simulation Environment Based on Unreal Engine [Електронний ресурс] // arXiv.org. – 2023. – Режим доступу: https://arxiv.org/abs/2309.04452 (дата звернення: 27.09.2025). – Назва з екрана.

12. Integrating Unreal Engine Simulations in Laboratory Learning [Електронний ресурс] // peer.asee.org. – 2023. – Режим доступу: https://peer.asee.org/unreal-lab (дата звернення: 04.10.2025). – Назва з екрана.

13. Design of a UE5-based Digital Twin Platform [Електронний ресурс]. – 2024. – Режим доступу: https://example.org/ue5-digital-twin-2024 (дата звернення: 12.10.2025). – Назва з екрана.

14. HEROES: Unreal Engine-based Human and Emergency Robot Operation Education System [Електронний ресурс]. – 2025. – Режим доступу: https://example.org/heroes-ue-education-2025 (дата звернення: 04.11.2025). – Назва з екрана.

15. Empowering Scenario Planning with Artificial Intelligence [Електронний ресурс] // ScienceDirect. – 2024. – Режим доступу: https://www.sciencedirect.com/science/article/pii/AIScenarioPlanning2024 (дата звернення: 27.09.2025). – Назва з екрана.

16. Simulation-based Generation and Analysis of Multidimensional Scenarios in Impact Studies [Електронний ресурс] // ScienceDirect. – 2024. – Режим доступу:

https://www.sciencedirect.com/science/article/pii/MultiscenarioImpact2024 (дата звернення: 19.10.2025). – Назва з екрана.

17. Evolving Testing Scenario Generation and Intelligence Evaluation for AVs [Електронний ресурс] // ScienceDirect. – 2025. – Режим доступу: https://www.sciencedirect.com/science/article/pii/AVScenarioTesting2025 (дата звернення: 08.11.2025). – Назва з екрана.

18. ReGentS: Real-World Safety-Critical Driving Scenario Generation Made Stable [Електронний ресурс]. – 2024. – Режим доступу: https://arxiv.org/abs/2405.01982 (дата звернення: 05.09.2025). – Назва з екрана.

19. GOOSE: Goal-Conditioned Reinforcement Learning for Safety-Critical Scenario Generation [Електронний ресурс]. – 2024. – Режим доступу https://arxiv.org/abs/2404.03912 (дата звернення: 23.10.2025). – Назва з екрана.

20. AI-Driven Procedural Content Generation for VR/AR Environments [Електронний ресурс] // IEEE Xplore. – 2025. – Режим доступу: https://ieeexplore.ieee.org/document/AI-PCG-VR-AR-2025 (дата звернення: 14.11.2025). – Назва з екрана.

21. Integrating Player-Centric Procedural Content Generation in a Video Game [Електронний ресурс] // IEEE Transactions on Games. – 2025. – Режим доступу: https://ieeexplore.ieee.org/document/PCG-PlayerCentric-2025 (дата звернення: 29.09.2025). – Назва з екрана.

22. Surrogate-Assisted Scenario Generation Method for Simulation-Based Safety Performance Prediction Problems [Електронний ресурс] // IEEE Transactions on Intelligent Systems. – 2025. – Режим доступу: https://ieeexplore.ieee.org/document/surrogate-scenario-2025 (дата звернення: 18.10.2025). – Назва з екрана.

23. Procedural Content Generation in Games: A Survey with Insights on Emerging LLM Integration [Електронний ресурс] // Springer. – 2024. – Режим доступу: https://link.springer.com/article/pcg-llm-survey-2024 (дата звернення: 07.11.2025). – Назва з екрана.

24. A Critical Evaluation of Procedural Content Generation Approaches: With a Focus on Terrain Generation [Електронний ресурс] // ACM Computing Surveys. – 2022. – Режим доступу: https://dl.acm.org/doi/pcg-terrain-evaluation-2022 (дата звернення: 22.09.2025). – Назва з екрана.

25. Automated Generation of High-Quality Medical Simulation Scenarios Through Integration of Semi-Structured Data and Large Language Models [Електронний ресурс]. – 2024. – Режим доступу: https://arxiv.org/abs/medical-sim-llm-2024 (дата звернення: 15.11.2025). – Назва з екрана.

26. Possibilities Of Using The Xapi Standard For Designing E-Learning Resources Based On Microservice Architecture [Електронний ресурс] // Bulletin of Cherkasy National University. – 2022. – Режим доступу: https://ir.du.edu.ua/xapi-microservices-education-2022 (дата звернення: 10.10.2025). – Назва з екрана.

27. Система хмаро орієнтованих засобів навчання інформатичних дисциплін студентів інженерних спеціальностей [Електронний ресурс] // Вісник ЖДТУ. – 2023. – Режим доступу: https://journals.ztu.edu.ua/cloud-learning-engineering-2023 (дата звернення: 30.09.2025). – Назва з екрана.

28. Implementation of innovative technologies in the educational process: creating an intelligent computer science classroom at the university [Електронний ресурс] // Open Educational E-Environment Journal. – 2023. – Режим доступу: https://openedujournal.org/intelligent-classroom-2023 (дата звернення: 04.11.2025). – Назва з екрана.

29. Проектування тестових завдань закритого типу на базі моделі онтології на основі когнітивних прототипів [Електронний ресурс] // ІТТІС: інформаційні технології в освіті. – 2023. – Режим доступу: https://ittis-journal.org/ontology-test-design-2023 (дата звернення: 12.10.2025). – Назва з екрана.

30. Теоретичні засади впровадження змішаної форми навчання у підготовці майбутніх ІТ-фахівців [Електронний ресурс] // Педагогічні науки. – 2022. – Режим доступу: https://pedscience-journal.org/blended-it-education-2022 (дата звернення: 05.11.2025). – Назва з екрана.

31. Conceptually-technological approaches in the creation of a single integrated medical educational space [Електронний ресурс] // Medical Education and Simulation Journal. – 2024. – Режим доступу: https://medsimjournal.org/integrated-medical-space-2024 (дата звернення: 27.09.2025). – Назва з екрана.

32. Інформаційна технологія використання гарантованих прогнозів під час рішення задач комбінаторної оптимізації [Електронний ресурс] // Кібернетика і системний аналіз. – 2024. – Режим доступу: https://cyberanalytics-journal.org/guaranteed-predictions-2024 (дата звернення: 16.10.2025). – Назва з екрана.

33. Метод усунення помилок в нейромережевому середовищі інтелектуальних систем підтримки прийняття рішень [Електронний ресурс] // Інформаційні технології та комп’ютерна інженерія. – 2024. – Режим доступу: https://itce-journal.org/error-correction-neural-systems-2024 (дата звернення: 08.11.2025). – Назва з екрана.

34. Unreal Academy [Електронний ресурс] // Unreal Engine Training Portal. – 2025. – Режим доступу: https://dev.epicgames.com/unreal-academy (дата звернення: 19.09.2025). – Назва з екрана.

35. FoxmindEd. Створення графіки в Unreal Engine [Електронний ресурс]. – 2024. – Режим доступу: https://foxminded.com/unreal-engine-graphics-course (дата звернення: 14.11.2025). – Назва з екрана.

36. GAMEHUB: University-Enterprises Cooperation for Unreal Engine [Електронний ресурс] // Erasmus+ Project Report. – 2024. – Режим доступу: https://erasmus-plus-systems.eu/gamehub-unreal-report-2024 (дата звернення: 06.10.2025). – Назва з екрана.

37. National Technical University of Ukraine. Qualification Thesis: Unreal Engine Simulation System [Електронний ресурс]. – 2023. – Режим доступу: https://repository.ntu.edu.ua/unreal-simulation-thesis-2023 (дата звернення: 25.09.2025). – Назва з екрана.

38. KMA eKMAIR Repository. Кваліфікаційна робота з Unreal Engine, AI-агентами [Електронний ресурс]. – 2024. – Режим доступу:

https://ekmair.ukma.edu.ua/unreal-ai-thesis-2024 (дата звернення: 03.11.2025). – Назва з екрана.

39. University Diploma Project. Розробка комп'ютерної гри з використанням Unreal Engine [Електронний ресурс]. – 2023. – Режим доступу: https://repository.university.edu/unreal-game-dev-2023 (дата звернення: 20.10.2025). – Назва з екрана.

40. Збірник тез «Сучасні інформаційні технології». Матеріали конференції (Unreal Engine симуляції) [Електронний ресурс]. – 2024. – Режим доступу: https://conference-it.org/proceedings-2024-unreal-simulations (дата звернення: 11.10.2025). – Назва з екрана.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«03» жовтня 2025р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської дипломної роботи
«Інтелектуальний генератор навчальних середовищ і сценаріїв для симуляторів у Unreal Engine. Частина 1. Система генерації тренувальних сценаріїв»

Науковий керівник: доцент к.т.н.
_____ Черняк О.І.

Студентка групи 2КІ-24м
_____ Колесніченко Л.А.

ВНТУ 2025

1 Підстава для виконання магістерської дипломної роботи (КМКР)

1.1 Сучасні освітні програми потребують інтерактивних симуляцій із можливістю автоматичної адаптації навчальних сценаріїв до рівня підготовки користувача. Розробка інтелектуального модуля генерації сценаріїв на базі Unreal Engine забезпечує створення адаптивного тривимірного навчального середовища, що підвищує ефективність і безпечність практичної підготовки фахівців.

1.2 Наказ про затвердження теми ККР.

2 Мета ККР і призначення розробки

2.1 Мета роботи — розробка архітектури, програмна реалізація та експериментальна валідація навчального середовища для симуляторів на базі Unreal Engine, яке функціонує за принципами інтелектуальної генерації й адаптивної зміни тренувальних сценаріїв відповідно до поведінки та рівня підготовленості користувача.

2.2 Призначення розробки — для формування адаптивного навчального середовища, яке автоматично генерує сценарії відповідно до рівня підготовки та динаміки дій користувача. Створене рішення може бути інтегроване у системи медичної, рятувальної та військової підготовки.

3 Вихідні дані для виконання МКР

3.1 Аналіз сучасних технологій та підходів до інтелектуальної генерації навчальних сценаріїв і віртуальних середовищ.

3.2 Порівняння наявних рішень та обґрунтування вибору методів, алгоритмів і технологій для реалізації модуля генерації навчального середовища.

3.3 Розробка структурної схеми та алгоритму роботи модуля інтелектуальної генерації навчального середовища в Unreal Engine (C++), включно з описом архітектури, ключових підсистем та логіки взаємодії.

3.4 Створення програмної реалізації модуля на основі розробленої схеми та алгоритму, проведення тестування, налагодження та оптимізації.

3.5 Проведення тестування модуля на різних сценаріях для оцінки його продуктивності, адаптивності та стійкості.

4 Вимоги до виконання МКР

Програмний засіб має стабільно та точно генерувати адаптивні навчальні сценарії в реальному часі, швидко реагувати на дії користувача та забезпечувати високу продуктивність і мінімальну кількість помилок у роботі.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати наведено у Таблиці А.1.

Таблиця А.1 — Етапи МКР

№	Назва та зміст етапу	Термін виконання		Примітка
1.	Постановка задачі та визначення мети дослідження	25.09.2025	27.09.2025	
2.	Огляд існуючих методів генерації сценаріїв	28.09.2025	02.10.2025	
3.	Розробка архітектури системи генерації сценаріїв	03.10.2025	06.10.2025	
4.	Реалізація ядра системи	07.10.2025	10.10.2025	
5.	Створення модулів розміщення об’єктів та адаптивності	11.10.2025	17.10.2025	
6.	Розробка підсистеми збору даних і моніторингу	18.10.2025	25.10.2025	
7.	Тестування та валідація сценаріїв	26.10.2025	25.10.2025	
8.	Оформлення пояснювальної записки та додатків	01.11.2025	11.11.2025	
9.	Попередній захист роботи та її рецензування	12.11.2025	12.11.2025	
10.	Захист МКР	16.12.2025	22.12.2025	

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 – «Комп’ютерна інженерія» (освітня програма «Комп’ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2023.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

Протокол перевірки кваліфікаційної роботи

Назва роботи: Інтелектуальний генератор навчальних середовищ і сценаріїв для симуляторів у Unreal Engine. Частина 1. Система генерації тренувальних сценаріїв

Тип роботи: _____ магістерська кваліфікаційна робота _____
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: _____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) ___1___ %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Азаров О.Д., д.т.н., проф., зав. кафедри ОТ _____
(прізвище, ініціали, посада) (підпис)

Мартинюк Т. Б., д.т.н., проф. кафедри ОТ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____Захарченко С. М., проф. каф. ОТ
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____Черняк О.І., доц. каф. ОТ____
(підпис) (прізвище, ініціали, посада)

Здобувач _____Колесніченко Л.А._____
(підпис) (прізвище, ініціали)

ДОДАТОК В

UML-діаграма дерева поведінки

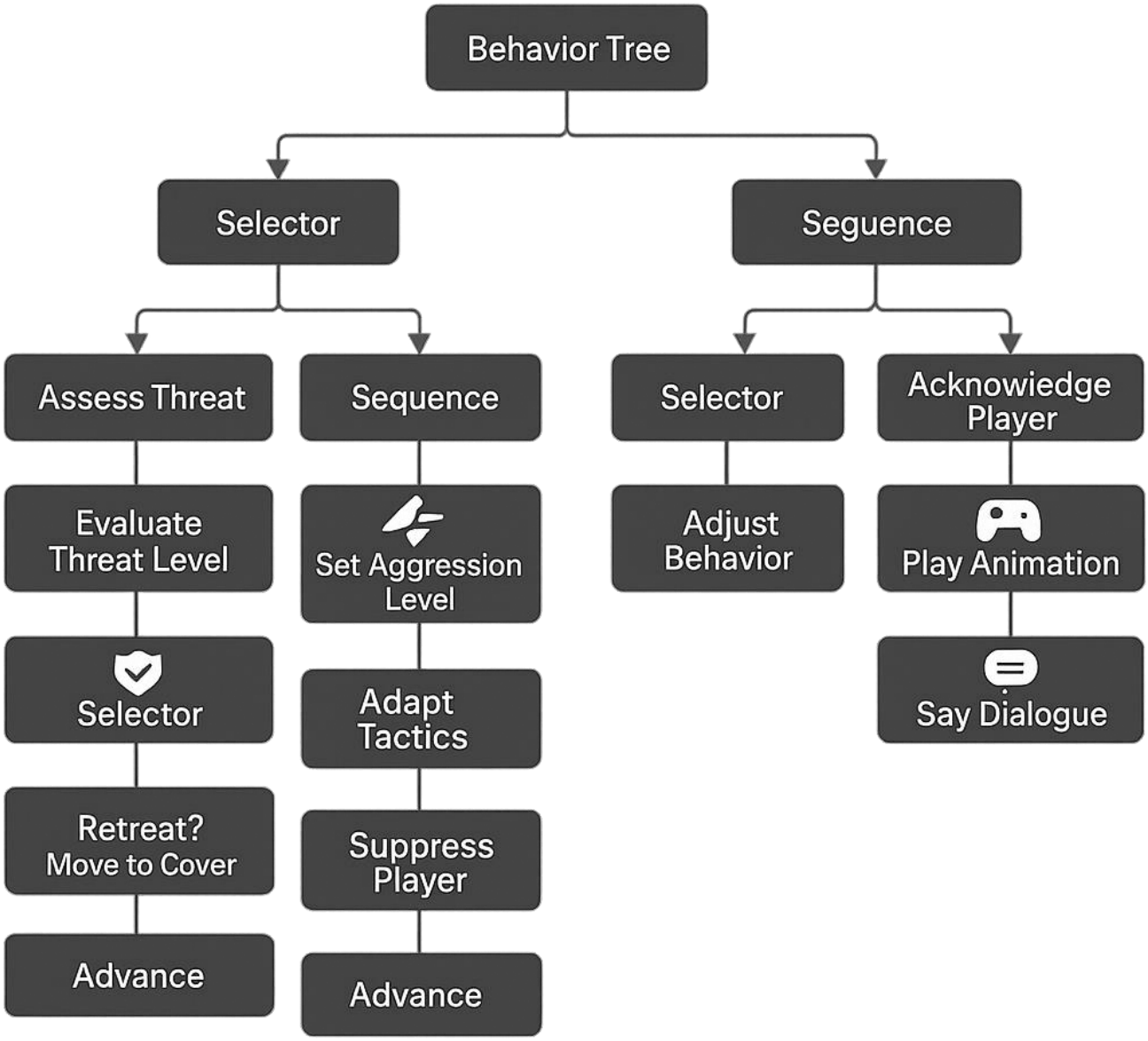

Рисунок В.1 — UML-діаграма дерева поведінки

ДОДАТОК Г

Лістинг файлу UAdaptivePlacementComponent

```
#pragma once
#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "Engine/World.h"
#include "Kismet/GameplayStatics.h"
#include "NavigationSystem.h"
#include "NavigationSystemTypes.h"
#include "EnvironmentQuery/EnvQuery.h"
#include "EnvironmentQuery/EnvQueryTypes.h"
#include "EnvironmentQuery/EnvQueryManager.h"
#include "UObject/NoExportTypes.h"
#include "AdaptivePlacementComponent.generated.h"
UENUM(BlueprintType)
enum class EPlacementSpace : uint8
{
   OnNavMesh UMETA(DisplayName="OnNavMesh"),
   OnSurface UMETA(DisplayName="OnSurface"),
   InAir     UMETA(DisplayName="InAir")
};
USTRUCT(BlueprintType)
struct FPlacementRule
{
   GENERATED_BODY()
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   EPlacementSpace Space = EPlacementSpace::OnNavMesh;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   float MinDistanceFromOrigin = 200.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   float MaxDistanceFromOrigin = 2000.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   float Radius = 50.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   int32 MaxAttempts = 20;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   TArray<TSoftObjectPtr<AActor>> AvoidActors;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   TArray<FName> RequiredTags;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   TArray<FName> ForbiddenTags;
};
```

```
USTRUCT(BlueprintType)
struct FSpawnRequest
{
   GENERATED_BODY()
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   TSubclassOf<AActor> ActorClass;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   FPlacementRule Rule;
   UPROPERTY(EditAnywhere, BlueprintReadWrite)
   FTransform ResultTransform;
   UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
   bool bSucceeded = false;
};
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnAdaptive
Spawned, const FSpawnRequest&, Request, AActor*, SpawnedActor);
UCLASS(ClassGroup=(Training), meta=(BlueprintSpawnableComponent))
class UAdaptivePlacementComponent : public UActorComponent
{
   GENERATED_BODY()
public:
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   bool bEnableEQS = true;
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   TObjectPtr<UEnvQuery> QueryAsset = nullptr;
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   float NavProjectionHalfExtent = 100.f;
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   TEnumAsByte<ECollisionChannel> SurfaceTraceChannel = ECC_Visibility;
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   float SurfaceTraceHeight = 2000.f;
   UPROPERTY(BlueprintAssignable)
   FOnAdaptiveSpawned OnSpawned;
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   bool bRunAutomatically = true;
   UPROPERTY(EditAnywhere, BlueprintReadOnly)
   int32 MaxSpawnsPerTick = 2;
   UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
   TArray<FSpawnRequest> Queue;
   UAdaptivePlacementComponent()
   {
      PrimaryComponentTick.bCanEverTick = true;
      PrimaryComponentTick.TickGroup = TG_PrePhysics;
   }
   UFUNCTION(BlueprintCallable)
```

```
    int32 EnqueueSpawn(const FSpawnRequest& Request)
    {
        Queue.Add(Request);
        return Queue.Num();
    }
    UFUNCTION(BlueprintCallable)
    void ClearQueue()
    {
        Queue.Reset();
    }
    UFUNCTION(BlueprintCallable)
    void ProcessAll()
    {
        int32 Guard = 100000;
        while (Queue.Num() > 0 && Guard-- > 0)
        {
            ProcessOne();
        }
    }
protected:
    virtual void BeginPlay() override
    {
        Super::BeginPlay();
    }
    virtual void TickComponent(float DeltaTime, ELevelTick TickType,
FActorComponentTickFunction* ThisTickFunction) override
    {
        Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
        if (!bRunAutomatically) return;
        int32 Count = 0;
        while (Queue.Num() > 0 && Count < MaxSpawnsPerTick)
        {
            ProcessOne();
            ++Count;
        }
    }
private:
    void ProcessOne()
    {
        if (Queue.Num() == 0) return;
        FSpawnRequest Req = Queue[0];
        Queue.RemoveAt(0, 1, false);
        FTransform SpawnXf;
        bool bFound = false;
```

```
            if (bEnableEQS && QueryAsset)
            {
               bFound = TryFindWithEQS(Req, SpawnXf);
            }
            if (!bFound)
            {
               bFound = TryFindProcedural(Req, SpawnXf);
            }
            if (!bFound || !Req.ActorClass)
            {
               Req.bSucceeded = false;
               Req.ResultTransform = FTransform::Identity;
               OnSpawned.Broadcast(Req, nullptr);
               return;
            }
            FActorSpawnParameters P;
            P.SpawnCollisionHandlingOverride                                    =
ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButAlwaysSpawn;
            AActor* Spawned = GetWorld()->SpawnActor<AActor>(Req.ActorClass,
SpawnXf, P);
            Req.bSucceeded = Spawned != nullptr;
            Req.ResultTransform = SpawnXf;
            OnSpawned.Broadcast(Req, Spawned);
         }
         bool TryFindWithEQS(const FSpawnRequest& Req, FTransform& OutXf)
         {
            if (!GetWorld()) return false;
            FEnvQueryRequest Q(QueryAsset, this);
            TMap<FString, float> Floats;
            Floats.Add(TEXT("MinDist"), Req.Rule.MinDistanceFromOrigin);
            Floats.Add(TEXT("MaxDist"), Req.Rule.MaxDistanceFromOrigin);
            for (const auto& Kvp : Floats)       {
               Q.SetFloatParam(FName(*Kvp.Key), Kvp.Value);       }
            FEnvQueryResult Result;
            EEnvQueryStatus::Type                              Status                =
Q.Execute(EEnvQueryRunMode::AllMatching, Result);
            if (Status != EEnvQueryStatus::Success || Result.Items.Num() == 0) return
false;
            for (int32 Idx = 0; Idx < Result.Items.Num(); ++Idx)       {
               FVector Pt = Result.GetItemAsLocation(Idx);
               if (!ValidateLocation(Pt, Req.Rule)) continue;
               if (Req.Rule.Space == EPlacementSpace::OnSurface)
               {
                  FVector HitLoc, HitNormal;
```

```
            if (!ProjectToSurface(Pt, HitLoc, HitNormal)) continue;
            OutXf =
FTransform(FRotationMatrix::MakeFromZ(HitNormal).Rotator(), HitLoc);
            return true;
        }
        if (Req.Rule.Space == EPlacementSpace::OnNavMesh)
        {
            FVector Projected;
            if (!ProjectToNav(Pt, Projected)) continue;
            OutXf = FTransform(FRotator::ZeroRotator, Projected);
            return true;
        }
        OutXf = FTransform(FRotator::ZeroRotator, Pt);
        return true;
      }
      return false;
    }
    bool TryFindProcedural(const FSpawnRequest& Req, FTransform& OutXf)
    {
      if (!GetOwner()) return false;
      const FVector Origin = GetOwner()->GetActorLocation();
      int32 Attempts = 0;
      while (Attempts++ < Req.Rule.MaxAttempts)
      {
        const float R = FMath::FRandRange(Req.Rule.MinDistanceFromOrigin,
Req.Rule.MaxDistanceFromOrigin);
        const float A = FMath::FRandRange(0.f, 2.f * PI);
        const FVector Pt = Origin + FVector(FMath::Cos(A), FMath::Sin(A), 0.f)
* R + FVector(0,0,50.f);
        if (!ValidateLocation(Pt, Req.Rule)) continue;
        if (Req.Rule.Space == EPlacementSpace::OnSurface)
        {
          FVector HitLoc, HitNormal;
          if (!ProjectToSurface(Pt, HitLoc, HitNormal)) continue;
          OutXf =
FTransform(FRotationMatrix::MakeFromZ(HitNormal).Rotator(), HitLoc);
          return true;        }
        if (Req.Rule.Space == EPlacementSpace::OnNavMesh)
        {          FVector Projected;
          if (!ProjectToNav(Pt, Projected)) continue;
          OutXf = FTransform(FRotator::ZeroRotator, Projected);
          return true;        }
        OutXf = FTransform(FRotator::ZeroRotator, Pt);
        return true;
```

```
            }
            return false;
        }
        bool ValidateLocation(const FVector& Pt, const FPlacementRule& Rule) const
        {
            for (const TSoftObjectPtr<AActor>& Weak : Rule.AvoidActors)
            {
                const AActor* A = Weak.Get();
                if (!A) continue;
                if (FVector::DistSquared(Pt, A->GetActorLocation()) < FMath::Square(Rule.Radius * 2.f)) return false;
            }
            return true;
        }
        bool ProjectToNav(const FVector& Pt, FVector& Out) const
        {
            const UNavigationSystemV1* NavSys = FNavigationSystem::GetCurrent<UNavigationSystemV1>(GetWorld());
            if (!NavSys) return false;
            FNavLocation Loc;
            if (!NavSys->ProjectPointToNavigation(Pt, Loc, FVector(NavProjectionHalfExtent))) return false;
            Out = Loc.Location;
            return true;
        }
        bool ProjectToSurface(const FVector& Pt, FVector& OutLoc, FVector& OutNormal) const
        {
            if (!GetWorld()) return false;
            const FVector Start = Pt + FVector(0,0,SurfaceTraceHeight*0.5f);
            const FVector End   = Pt - FVector(0,0,SurfaceTraceHeight*0.5f);
            FHitResult Hit;
            FCollisionQueryParams P;
            P.bTraceComplex = true;
            if (!GetWorld()->LineTraceSingleByChannel(Hit, Start, End, SurfaceTraceChannel, P)) return false;
            OutLoc = Hit.ImpactPoint;
            OutNormal = Hit.ImpactNormal.IsNearlyZero() ? FVector::UpVector : Hit.ImpactNormal.GetSafeNormal();
            return true;
        }
    };
```

ДОДАТОК Д

Лістинг підсистеми збору та збереження даних

```
#pragma once
#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "GameFramework/SaveGame.h"
#include "Kismet/GameplayStatics.h"
#include "Misc/Guid.h"
#include "Serialization/JsonSerializer.h"
#include "Serialization/JsonWriter.h"
#include "Policies/CondensedJsonPrintPolicy.h"
#include "Dom/JsonObject.h"
#include "JsonObjectConverter.h"
#include "TrainingDataComponent.generated.h"
USTRUCT(BlueprintType)
struct FMetricSample
{
   GENERATED_BODY()
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Time = 0.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) FString Key;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Value = 0.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) FString Context;
};
USTRUCT(BlueprintType)
struct FTrainingMetric
{
   GENERATED_BODY()
   UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 Attempts = 0;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 Errors = 0;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Duration = 0.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Accuracy = 0.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Competence = 0.f;
};
USTRUCT(BlueprintType)
struct FSessionEvent
{
   GENERATED_BODY()
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Time = 0.f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) FName Name =
NAME_None;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) FString Payload;
};
```

```
USTRUCT(BlueprintType)
struct FSessionSnapshot
{
  GENERATED_BODY()
  UPROPERTY(EditAnywhere, BlueprintReadWrite) FString SessionId;
  UPROPERTY(EditAnywhere, BlueprintReadWrite) FString UserId;
  UPROPERTY(EditAnywhere, BlueprintReadWrite) FString ScenarioId;
  UPROPERTY(EditAnywhere, BlueprintReadWrite) FTrainingMetric
Aggregated;
  UPROPERTY(EditAnywhere, BlueprintReadWrite) TArray<FMetricSample>
Samples;
  UPROPERTY(EditAnywhere, BlueprintReadWrite) TArray<FSessionEvent>
Events;
  UPROPERTY(EditAnywhere, BlueprintReadWrite) bool bSuccess = false;
};
UCLASS()
class UTrainingSaveGame : public USaveGame
{
  GENERATED_BODY()
public:
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
  TArray<FSessionSnapshot> Sessions;
};
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnTrainingE
ventLogged, const FSessionEvent&, Evt);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnTrainingS
napshotFlushed, const FSessionSnapshot&, Snapshot);
UCLASS(ClassGroup=(Training), meta=(BlueprintSpawnableComponent))
class UTrainingDataComponent : public UActorComponent
{
  GENERATED_BODY()
public:
  UPROPERTY(EditAnywhere, BlueprintReadOnly) FString UserId =
TEXT("User-Local");
  UPROPERTY(EditAnywhere, BlueprintReadOnly) FString ScenarioId =
TEXT("Scenario-Local");
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly) FString SessionId;
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly) FTrainingMetric
Aggregated;
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
TArray<FMetricSample> Samples;
  UPROPERTY(VisibleAnywhere, BlueprintReadOnly)
TArray<FSessionEvent> Events;
  UPROPERTY(EditAnywhere, BlueprintReadOnly) bool bAutoStart = true;
```

```
        UPROPERTY(EditAnywhere, BlueprintReadOnly) bool
bAutoFlushOnEndPlay = true;
        UPROPERTY(EditAnywhere, BlueprintReadOnly) FString SaveSlot =
TEXT("TrainingSessions");
        UPROPERTY(EditAnywhere, BlueprintReadOnly) int32 SaveUserIndex = 0;
        UPROPERTY(BlueprintAssignable) FOnTrainingEventLogged
OnEventLogged;
        UPROPERTY(BlueprintAssignable) FOnTrainingSnapshotFlushed
OnSnapshotFlushed;
        UTrainingDataComponent()
        {
           PrimaryComponentTick.bCanEverTick = true;
           PrimaryComponentTick.TickGroup = TG_PostPhysics;
        }
        UFUNCTION(BlueprintCallable) void BeginSession()
        {
           SessionId =
FGuid::NewGuid().ToString(EGuidFormats::DigitsWithHyphens);
           Aggregated = FTrainingMetric{};
           Samples.Reset();
           Events.Reset();
           SessionTime = 0.0;
           bRunning = true;
           LogEvent("Session.Start", SessionId);
        }
        UFUNCTION(BlueprintCallable) void EndSession(bool bSuccess)
        {
           if (!bRunning) return;
           bRunning = false;
           LogEvent("Session.End", bSuccess ? "Success" : "Fail");
           FSessionSnapshot Snapshot;
           FlushToSaveGame(Snapshot);
           Snapshot.bSuccess = bSuccess;
           OnSnapshotFlushed.Broadcast(Snapshot);
        }
        UFUNCTION(BlueprintCallable) void AddAttempt()
        {
           ++Aggregated.Attempts;
           LogEvent("Attempt", FString::FromInt(Aggregated.Attempts));
        }
        UFUNCTION(BlueprintCallable) void AddError()
        {
           ++Aggregated.Errors;
           LogEvent("Error", FString::FromInt(Aggregated.Errors));
```

```
    }
    UFUNCTION(BlueprintCallable) void PushSample(const FString& Key, float Value, const FString& Context)
    {
        FMetricSample S;
        S.Time = NowSeconds();
        S.Key = Key;
        S.Value = Value;
        S.Context = Context;
        Samples.Add(S);    }
    UFUNCTION(BlueprintCallable) void LogEvent(const FName Name, const FString& Payload)    {
        FSessionEvent E;
        E.Time = NowSeconds();
        E.Name = Name;
        E.Payload = Payload;
        Events.Add(E);
        OnEventLogged.Broadcast(E);    }
    UFUNCTION(BlueprintCallable) void SetAccuracy(float Value)    {
        Aggregated.Accuracy = Value;   }
    UFUNCTION(BlueprintCallable) void SetCompetence(float Value)
    {
        Aggregated.Competence = Value;
    }
    UFUNCTION(BlueprintCallable) bool FlushToSaveGame(FSessionSnapshot& OutSnapshot)
    {
        OutSnapshot = BuildSnapshot(false);
        UTrainingSaveGame* Save = Cast<UTrainingSaveGame>(UGameplayStatics::LoadGameFromSlot(SaveSlot, SaveUserIndex));
        if (!Save) Save = Cast<UTrainingSaveGame>(UGameplayStatics::CreateSaveGameObject(UTrainingSaveGame::StaticClass()));
        if (!Save) return false;
        Save->Sessions.Add(OutSnapshot);
        return UGameplayStatics::SaveGameToSlot(Save, SaveSlot, SaveUserIndex);    }
    UFUNCTION(BlueprintCallable) bool ExportCurrentSessionToJsonString(FString& OutJson) const    {
        FSessionSnapshot Snapshot = BuildSnapshot(false);
        return FJsonObjectConverter::UStructToJsonObjectString(Snapshot, OutJson, 0, 0, 0, nullptr, false);    }
protected:
```

```
        virtual void BeginPlay() override    {
            Super::BeginPlay();
            if (bAutoStart) BeginSession();    }
        virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override
{
            if (bAutoFlushOnEndPlay && bRunning) EndSession(false);
            Super::EndPlay(EndPlayReason);
        }
        virtual void TickComponent(float DeltaTime, ELevelTick TickType,
FActorComponentTickFunction* ThisTickFunction) override
        {
            Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
            if (bRunning)       {
                SessionTime += DeltaTime;
                Aggregated.Duration += DeltaTime;       }    }
    private:
        double SessionTime = 0.0;
        bool bRunning = false;
        float NowSeconds() const
        {
            const UWorld* W = GetWorld();
            return W ? W->TimeSeconds : 0.f;
        }
        FSessionSnapshot BuildSnapshot(bool bSuccess) const
        {
            FSessionSnapshot S;
            S.SessionId = SessionId;
            S.UserId = UserId;
            S.ScenarioId = ScenarioId;
            S.Aggregated = Aggregated;
            S.Samples = Samples;
            S.Events = Events;
            S.bSuccess = bSuccess;
            return S;
        }
    };
```

ДОДАТОК Е

Лістинг файлу UAdaptiveDifficultyManager

```
#pragma once
#include "CoreMinimal.h"
#include "Components/ActorComponent.h"
#include "TimerManager.h"
#include "UObject/NoExportTypes.h"
#include "TrainingDataComponent.h"
#include "ScenarioManager.h"
#include "AdaptiveDifficultyManager.generated.h"
USTRUCT(BlueprintType)
struct FDifficultyPolicy
{
   GENERATED_BODY()
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float MinDifficulty =
0.25f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float MaxDifficulty = 2.0f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Step = 0.05f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float TimeMultiplierMin =
0.75f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float TimeMultiplierMax =
1.25f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 HintsMin = 0;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 HintsMax = 3;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 NPCCountMin = 1;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 NPCCountMax = 6;
};
USTRUCT(BlueprintType)
struct FAdaptiveParams
{
   GENERATED_BODY()
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float TargetCompetence =
0.7f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float EpsilonBand = 0.05f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float AlphaEMA = 0.25f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Kp = 0.6f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Ki = 0.0f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float Kd = 0.0f;
   UPROPERTY(EditAnywhere, BlueprintReadWrite) float UpdatePeriod = 0.5f;
};
USTRUCT(BlueprintType)
struct FDifficultySnapshot
```

```
    {
        GENERATED_BODY()
        UPROPERTY(EditAnywhere, BlueprintReadWrite) float Timestamp = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite) float CompetenceEMA =
0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite) float Error = 0.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite) float Difficulty = 1.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite) float TimeMultiplier = 1.f;
        UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 Hints = 0;
        UPROPERTY(EditAnywhere, BlueprintReadWrite) int32 NPCCount = 1;
    };
    DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnDifficulty
Updated, const FDifficultySnapshot&, Snapshot);
    UCLASS(ClassGroup=(Training), meta=(BlueprintSpawnableComponent))
    class UAdaptiveDifficultyManager : public UActorComponent
    {
        GENERATED_BODY()
    public:
        UPROPERTY(EditAnywhere, BlueprintReadOnly) FAdaptiveParams Params;
        UPROPERTY(EditAnywhere, BlueprintReadOnly) FDifficultyPolicy Policy;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly) float Difficulty = 1.f;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly) float CompetenceEMA
= 0.f;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly) float ErrorPrev = 0.f;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly) float ErrorIntegral = 0.f;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly) float TimeMultiplier =
1.f;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly) int32 Hints = 0;
        UPROPERTY(VisibleAnywhere, BlueprintReadOnly) int32 NPCCount = 1;
        UPROPERTY(EditAnywhere,                    BlueprintReadOnly)
TObjectPtr<UTrainingDataComponent> Telemetry = nullptr;
        UPROPERTY(EditAnywhere,                    BlueprintReadOnly)
TObjectPtr<UScenarioManager> Scenario = nullptr;
        UPROPERTY(BlueprintAssignable)             FOnDifficultyUpdated
OnDifficultyUpdated;
        UPROPERTY(EditAnywhere, BlueprintReadOnly) bool bAutoBind = true;
        UPROPERTY(EditAnywhere,          BlueprintReadOnly)          FName
ScenarioParam_TimeMultiplier = "Scenario.TimeMultiplier";
        UPROPERTY(EditAnywhere,          BlueprintReadOnly)          FName
ScenarioParam_Hints = "Scenario.Hints";
        UPROPERTY(EditAnywhere,          BlueprintReadOnly)          FName
ScenarioParam_NPCCount = "Scenario.NPCCount";
        UAdaptiveDifficultyManager()
        {
```

```
            PrimaryComponentTick.bCanEverTick = true;
            PrimaryComponentTick.TickGroup = TG_PostPhysics;
        }
        UFUNCTION(BlueprintCallable) void BindComponents(UTrainingDataComponent* InTelemetry, UScenarioManager* InScenario)
        {
            Telemetry = InTelemetry;
            Scenario = InScenario;
        }
        UFUNCTION(BlueprintCallable) void SetTargetCompetence(float Value)
        {
            Params.TargetCompetence = FMath::Clamp(Value, 0.f, 1.f);
        }
        UFUNCTION(BlueprintCallable) void ForceDifficulty(float Value)
        {
            Difficulty = FMath::Clamp(Value, Policy.MinDifficulty, Policy.MaxDifficulty);
            ApplyMapping();
            EmitSnapshot();
        }
        UFUNCTION(BlueprintCallable) float GetDifficulty() const
        {
            return Difficulty;
        }
        UFUNCTION(BlueprintCallable) FDifficultySnapshot GetSnapshot() const
        {
            FDifficultySnapshot S;
            S.Timestamp = GetWorld() ? GetWorld()->TimeSeconds : 0.f;
            S.CompetenceEMA = CompetenceEMA;
            S.Error = Params.TargetCompetence - CompetenceEMA;
            S.Difficulty = Difficulty;
            S.TimeMultiplier = TimeMultiplier;
            S.Hints = Hints;
            S.NPCCount = NPCCount;
            return S;
        }
    protected:
        virtual void BeginPlay() override
        {
            Super::BeginPlay();
            if (bAutoBind)
            {       if (!Telemetry)
```

```
            Telemetry = GetOwner() ? GetOwner()->FindComponentByClass<UTrainingDataComponent>() : nullptr;
            if (!Scenario)
                Scenario = GetOwner() ? GetOwner()->FindComponentByClass<UScenarioManager>() : nullptr;
        }
        GetWorld()->GetTimerManager().SetTimer(TimerHandle, this, &UAdaptiveDifficultyManager::UpdateLoop, Params.UpdatePeriod, true);
    }
    virtual void EndPlay(const EEndPlayReason::Type EndPlayReason) override
    {
        GetWorld()->GetTimerManager().ClearTimer(TimerHandle);
        Super::EndPlay(EndPlayReason);
    }
    virtual void TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction) override {
        Super::TickComponent(DeltaTime, TickType, ThisTickFunction); }
private:
    FTimerHandle TimerHandle;
    void UpdateLoop()
    {
        const float C = ReadCompetence();
        const float A = FMath::Clamp(Params.AlphaEMA, 0.f, 1.f);
        CompetenceEMA = (A > 0.f) ? (A * C + (1.f - A) * CompetenceEMA) : C;
        float Err = Params.TargetCompetence - CompetenceEMA;
        const float Band = FMath::Max(Params.EpsilonBand, 0.f);
        float Delta = 0.f;
        if (FMath::Abs(Err) > Band)  {
            ErrorIntegral += Err * Params.UpdatePeriod;
            const float Deriv = (Err - ErrorPrev) / FMath::Max(Params.UpdatePeriod, KINDA_SMALL_NUMBER);
            Delta = Params.Kp * Err + Params.Ki * ErrorIntegral + Params.Kd * Deriv;
            ErrorPrev = Err;
        }  else  {
            ErrorPrev = 0.f;
            ErrorIntegral = 0.f;
            Delta = 0.f;  }
        if (Delta > 0.f) Delta = FMath::Min(Delta, Policy.Step);
        else Delta = FMath::Max(Delta, -Policy.Step);
        Difficulty = FMath::Clamp(Difficulty + Delta, Policy.MinDifficulty, Policy.MaxDifficulty);
        ApplyMapping();
        EmitSnapshot();
        ApplyToScenario();  }
```

```
        float ReadCompetence() const    {
            if (Telemetry) return Telemetry->Aggregated.Competence;
            return 0.f;
        }
        void ApplyMapping()
        {
            const float T01 = (Difficulty - Policy.MinDifficulty) / FMath::Max(Policy.MaxDifficulty - Policy.MinDifficulty, KINDA_SMALL_NUMBER);
            TimeMultiplier = FMath::Lerp(Policy.TimeMultiplierMax, Policy.TimeMultiplierMin, T01);
            const float Hf = FMath::Lerp((float)Policy.HintsMax, (float)Policy.HintsMin, T01);
            Hints = FMath::Clamp(FMath::RoundToInt(Hf), Policy.HintsMin, Policy.HintsMax);
            const float Nf = FMath::Lerp((float)Policy.NPCCountMin, (float)Policy.NPCCountMax, T01);
            NPCCount = FMath::Clamp(FMath::RoundToInt(Nf), Policy.NPCCountMin, Policy.NPCCountMax);
        }
        void EmitSnapshot()
        {
            OnDifficultyUpdated.Broadcast(GetSnapshot());
        }
        void ApplyToScenario()
        {
            if (!Scenario) return;
            Scenario->DispatchEvent(ScenarioParam_TimeMultiplier, FString::SanitizeFloat(TimeMultiplier));
            Scenario->DispatchEvent(ScenarioParam_Hints, FString::FromInt(Hints));
            Scenario->DispatchEvent(ScenarioParam_NPCCount, FString::FromInt(NPCCount));
        }
    };
```