Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА RPG-ГРИ В UNREAL ENGINE 5 ІЗ ЗАСТОСУВАННЯМ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ МОДЕЛЮВАННЯ ПОВЕДІНКИ НЕКЕРОВАНИХ ПЕРСОНАЖІВ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-3
Літвінова О. О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Любченко В. А.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Літвінової Олесі Олександрівни
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка RPG-гри в Unreal Engine 5 із застосуванням засобів штучного інтелекту для моделювання поведінки некерованих персонажів

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 29 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, документація і довідкові матеріали Unreal Engine 5, офіційна документація засобів штучного інтелекту, дослідження в галузі моделювання поведінки NPC у відеоіграх.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних підходів до реалізації штучного інтелекту для NPC у відеоіграх.
2. Дослідження можливостей Unreal Engine 5 для створення інтелектуальної поведінки.
3. Розробка шаблонної структури дерева поведінки NPC.
4. Побудова простої моделі навчального агента.
5. Оцінка ефективності запропонованої моделі та можливостей її масштабування.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Дерево поведінки, постановка задачі, тренувальне середовище, графік навчання, схема пошуку точки.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка гри	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Любченко В. А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 71 с., 2 табл., 32 рис., 44 джерела.

ШТУЧНИЙ ІНТЕЛЕКТ, НЕКЕРОВАНИЙ ПЕРСОНАЖ, ДЕРЕВО ПОВЕДІНКИ, НЕЙРОМЕРЕЖІ, UNREAL ENGINE 5.

Об’єктом роботи є некеровані персонажі або NPC, що функціонують в ігровому середовищі.

Метою роботи є дослідження сучасних методів створення реалістичної поведінки NPC за допомогою засобів штучного інтелекту в Unreal Engine 5.

У роботі розглянуто еволюцію підходів до розробки поведінки NPC, від скриптів і скінченних автоматів до гнучких систем на основі Behavior Tree та Environmental Query System. Поведінка персонажів аналізується з точки зору адаптивності, динамічності взаємодії з гравцем та ефективності реагування на змінні умови ігрового середовища. Особливу увагу приділено можливості інтеграції методів машинного навчання для створення самонавчальних агентів, здатних адаптуватися до ігрового процесу на основі досвіду.

У результаті роботи сформовано уявлення про ключові інструменти реалізації інтелектуальної поведінки NPC та окреслено перспективи впровадження нейромережевих підходів у сучасній ігровій індустрії.

ARTIFICIAL INTELLIGENCE, NON-PLAYABLE CHARATER, BEHAVIOR TREE, NEURAL NETVORKS, UNREAL ENGINE 5.

The object of the work is non-playable characters or NPCs functioning in a game environment.

The aim of the work is to explore modern methods for creating realistic NPC behavior using artificial intelligence tools in Unreal Engine 5.

The work examines the evolution of approaches to NPC behavior development, from scripts and finite state machines to flexible systems based on Behavior Tree and the Environmental Query System. NPC behavior is analyzed in terms of adaptability, dynamic interaction with the player, and responsiveness to changing game conditions. Special attention is given to the potential integration of machine learning methods to create self-learning agents capable of adapting to gameplay based on experience.

As a result, the work outlines the core tools for implementing intelligent NPC behavior and highlights the prospects of applying neural network approaches in modern game development.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence (штучний інтелект)

ШІ – штучний інтелект

NPC – Non-Playable Character (некерований персонаж); ігровий персонаж, поведінка якого контролюється системою, а не гравцем

BT – Behavior Tree (дерево поведінки); структура для формалізації логіки дій NPC залежно від умов у грі

EQS – Environmental Query System (система запитів до середовища); компонент Unreal Engine, що дозволяє NPC аналізувати навколишнє середовище

UE5 – Unreal Engine 5; ігровий рушій

NavMesh – Navigation Mesh (навігаційна сітка); структура, яка дозволяє NPC пересуватися у просторі гри з урахуванням перешкод

RPG – Role-Playing Game (рольова гра); жанр відеоігор, у якому гравець керує персонажем або групою персонажів

AAA – Triple-A (AAA-рівень); позначення відеоігор з великим бюджетом, високою якістю та масштабною розробкою

GAS – Gameplay Ability System (система ігрових здібностей); фреймворк Unreal Engine для створення здібностей

ВСТУП

У сучасній ігровій індустрії одним із ключових завдань є розробка реалістичної поведінки віртуальних персонажів, які взаємодіють із гравцем та навколишнім середовищем. Некеровані персонажі – це персонажі, що функціонують на основі заданих алгоритмів та приймають рішення відповідно до програмних сценаріїв.

Перші методи розробки поведінки NPC ґрунтувалися на використанні скриптових систем та скінченних автоматів. Такі підходи мали низку суттєвих обмежень, зокрема відсутність адаптивності до змінних умов гри, лінійність дій та обмежена взаємодія з гравцем. Через ці недоліки постала необхідність впровадження більш гнучких систем штучного інтелекту.

Одним із найбільш поширених механізмів управління поведінкою NPC є система Behavior Tree. Вона дозволяє формалізувати процес прийняття рішень шляхом побудови дерева логіки, що визначає послідовність дій персонажа залежно від зовнішніх факторів. Проте цей підхід, попри його структурованість, має обмеження, пов’язані з фіксованими правилами, які не враховують динамічні зміни в ігровому середовищі.

Для підвищення рівня автономності NPC використовується система запитів до середовища (Environmental Query System, EQS), що дозволяє персонажам здійснювати аналіз оточення та приймати рішення на основі отриманих даних. Ця система є особливо ефективною в складних сценаріях, де необхідний адаптивний пошук оптимальних точок пересування, укриття або взаємодії. Застосування EQS сприяє створенню більш природної поведінки NPC, роблячи їхні дії менш передбачуваними для гравця. EQS дозволяє враховувати динамічні зміни в ігровому середовищі, що робить поведінку персонажів більш гнучкою та реалістичною.

Крім того, перспективним напрямом у розробці NPC є інтеграція машинного навчання, яке в майбутньому може забезпечити можливість навчання персонажів на основі їхньої взаємодії з гравцем та ігровим

середовищем. Незважаючи на те, що наразі такий підхід не є широко впровадженим через високу складність реалізації та значні обчислювальні витрати, у майбутньому він може стати ключовим інструментом для створення більш динамічних і реалістичних ігрових всесвітів.

Інтеграція алгоритмів штучного інтелекту відкриває нові можливості у розробці NPC, дозволяючи персонажам змінювати свою поведінку на основі отриманого досвіду. Використання нейромережевих моделей та методів підкріпленого навчання дає змогу не лише підвищити рівень реалістичності, але й забезпечити поступову адаптацію персонажів до нових умов гри.

Впровадження інтелектуальних NPC значно підвищує рівень занурення гравця у віртуальний світ, забезпечуючи складніші та більш інтерактивні сценарії розвитку подій. У перспективі розвиток адаптивних агентів штучного інтелекту дозволить створювати динамічні системи, що змінюються у відповідь на дії гравця, надаючи кожному користувачеві унікальний досвід.

Таким чином, використання штучного інтелекту відіграє ключову роль у модернізації сучасних відеоігор, сприяючи переходу від статичних механік до самонавчальних систем, що суттєво розширюють можливості інтерактивності та реалістичності ігрового процесу.

1 ЗАГАЛЬНІ ПРИНЦИПИ ПОБУДОВИ ІНТЕЛЕКТУАЛЬНОЇ ПОВЕДІНКИ НЕКЕРОВАНИХ ПЕРСОНАЖІВ

1.1 Поняття та роль NPC у сучасних відеоіграх

У контексті сучасної індустрії відеоігор некеровані персонажі (Non-Playable Characters, NPC) є ключовим елементом, що забезпечує насиченість і реалістичність ігрового світу. NPC – це цифрові агенти, які функціонують за межами прямого контролю гравця та діють згідно з попередньо визначеними алгоритмами або гнучкими системами штучного інтелекту [1]. Вони можуть виконувати як прості завдання, наприклад озвучувати репліки на тригер, так і складні ролі, такі як супутники, що адаптують свою поведінку до дій гравця або змін навколишнього середовища.

На початкових стадіях еволюції відеоігор NPC виконували роль статичних елементів, переважно слугуючи лише частиною оточення або джерелом елементарної інформації. У класичних RPG такого типу персонажі зазвичай повторювали одну й ту ж репліку під час кожної взаємодії. Цей підхід ґрунтувався на спрощеній логіці і жорстко залежав від сценарного пропису. Проте з поступовим удосконаленням комп’ютерної інженерії, алгоритмів штучного інтелекту та підходів до геймдизайну функціональність і складність NPC значно розширилися. Вони набули здатності до умовного прийняття рішень, виконання патрульних маршрутів, реагування на звуки і візуальні стимули, дотримання власних щоденних розкладів, запам’ятовування подій минулого тощо.

У структурі відеогри NPC можуть виконувати функцію як допоміжних, так і ключових персонажів. Допоміжні NPC сприяють атмосфері гри, створюють ефект населеного світу, наповнюють міста мешканцями, грають роль продавців або клієнтів. Їхня поведінка зазвичай керована простими правилами. Натомість основні NPC можуть бути інтегровані в сюжет і навіть впливати на хід подій. Прикладами можуть бути союзники в бойових місіях,

антагоністи з власною мотивацією, або ті хто дає гравцеві завдання та змінює до нього ставлення залежно від попередніх рішень.

Однією з ключових особливостей сучасних NPC є їхня здатність взаємодіяти з навколишнім середовищем. У багатьох іграх NPC можуть розуміти топографію карти, уникати перешкод, використовувати укриття, змінювати місце дислокації залежно від часу доби чи загроз.

Окремо слід виділити роль NPC у побудові наративу. Завдяки детально прописаним сценаріям і емоційній анімації, системам діалогів та варіативності реакцій NPC стають посередниками емоційної взаємодії гравця з ігровим світом. У проєктах із відкритим світом, таких як The Witcher 3, Red Dead Redemption 2 чи Cyberpunk 2077 NPC не лише «реагують» на гравця, а й мають власні щоденні рутини (рис. 1.1), унікальні діалоги (рис. 1.2), історії та поведінку, що змінюється залежно від ігрового контексту [2].

Сучасні підходи до реалізації NPC все більше орієнтовані на персоналізації досвіду. Ігри можуть генерувати різні типи NPC з унікальними наборами параметрів, що змінюються в залежності від дій гравця. Сюди входить система моралі або репутації, зв’язків між персонажами. У багатьох RPG взаємини з NPC можуть мати довгострокові наслідки, які впливають на епілог гри або хід подій.

Рисунок 1.1 – Приклад розпорядку дня NPC у грі Red Dead Redemption 2

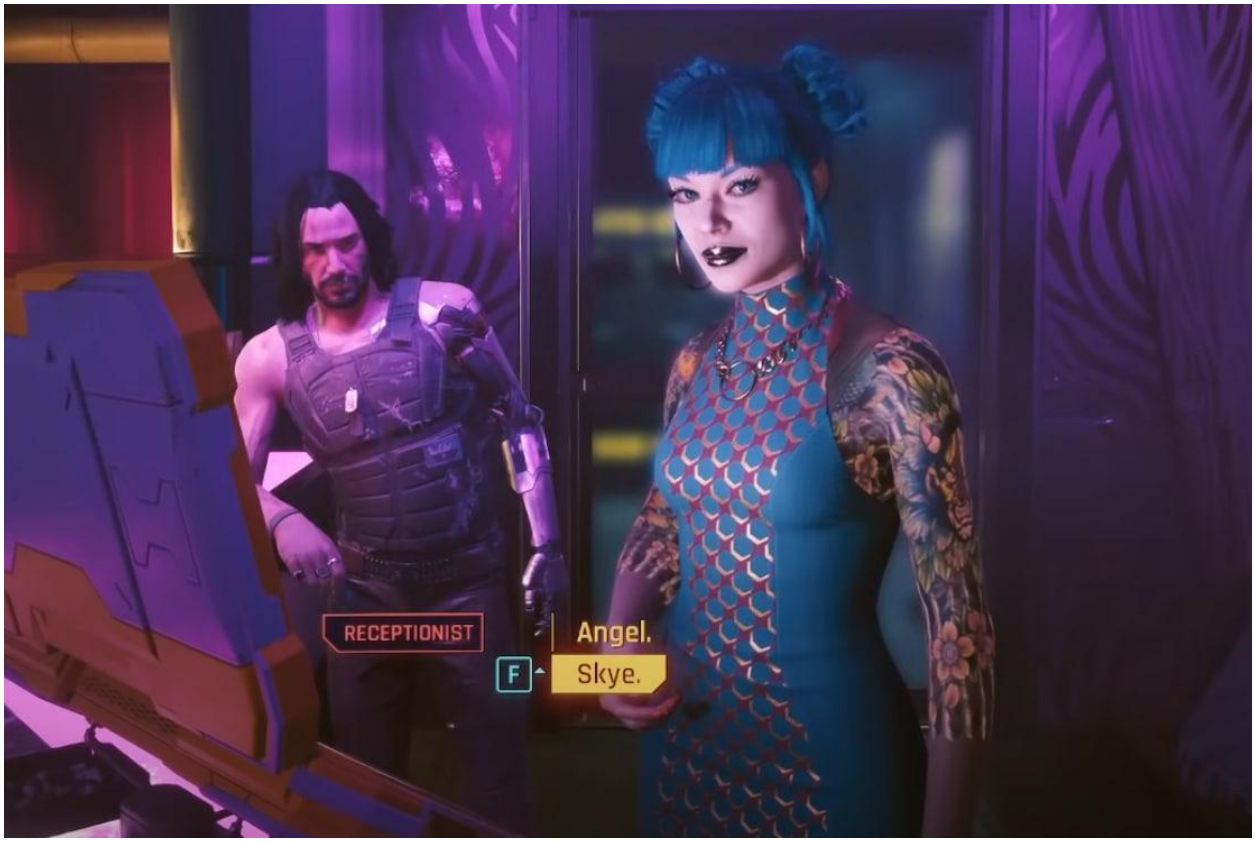

Рисунок 1.2 – Приклад вибору відповідей у діалогах у грі Cyberpunk 2077

Не менш важливою є роль NPC у бойових системах. У шутерах, стелс-іграх або тактичних RPG вони можуть діяти як противники з індивідуальними шаблонами поведінки. Наприклад ворог може втікати, якщо його сили менші або викликати підкріплення при усвідомленнні переваги гравця. Завдяки алгоритмам адаптивної поведінки такі NPC здатні змінювати тактику залежно від ігрової ситуації.

Розвиток навчальних агентів та впровадження методів машинного навчання відкриває нову еру для NPC. У майбутньому, вони зможуть не просто реагувати на ситуації, а й самостійно вивчати ігровий світ, розробляти нові стратегії, співпрацювати або конкурувати з гравцем. Вже сьогодні деякі проєкти експериментують із навчанням з підкріпленням, що дозволяє NPC розвивати свої навички на основі досвіду.

1.2 Історичний розвиток підходів до створення штучного інтелекту некерованого персонажа

Розвиток штучного інтелекту для некерованих персонажів тісно пов’язаний з загальною еволюцією ігрової індустрії. З найпростіших форм алгоритмів у класичних аркадах і до сучасних адаптивних систем з елементами машинного навчання. Кожен етап демонструє поступовий перехід від статичної запрограмованої поведінки до гнучкої взаємодії між NPC та гравцем.

Перші NPC у відеоіграх з’явилися ще в 1970-х роках, коли обчислювальні ресурси були надзвичайно обмеженими. Їх поведінка задавалась через прості скрипти, які жорстко закодовані в програмному коді. Такі персонажі виконували фіксовані дії у відповідь на конкретні тригери. Наприклад, в Pac-Man 1980 року (рис. 1.3) привиди мали заздалегідь визначені патерни руху, які імітували «переслідування» гравця, але на справді не адаптувались до ситуації [3].

Рисунок 1.3 – Гра Pac-Man 1980 року

Подібні підходи були ефективні у рамках технічних обмежень, однак все ж таки не дозволяли створити ілюзію інтелекту чи розумінню світу. Поведінка NPC була передбачуваною, легко піддавалась запам’ятовуанню, що знижувало загальну глибину геймплею.

У 1990-х роках з появою тривимірних ігор і складніших механік почали активно використовуватись Finite State Machines – скінченні автомати станів. У такій моделі NPC перемикався між обмеженим набором поведінкових станів (патрулює → атакує → тікає), залежно від умов у грі. Кожен стан мав визначені умови входу, дії та умови виходу в дію. Finite State Machines стали стандартом для багатьох жанрів особливо у шутерах та RPG. Їх реалізація була відносно простою, але разом зі зростанням складності ігор, FSM виявили свої обмеження, вони погано масштабуються, потребують значної кількості переходів при ускладненні логіки та не дозволяють NPC поводитись «розумно» у непередбачуваних ситуаціях.

Для подолання обмежень FSM у 2000-х роках почали широко впроваджуватись дерева рішень, які згодом трансформувались у Behavior Trees, тобто дерева повідінки – структури, що організовують поведінку NPC у вигляді ієрархії перевірок умов та відповідних дій. Цей підхід значно полегшив керування складною логікою, а також зробив поведінку NPC більш модульною та розширюваною. Behavior Trees швидко стали індустріальним стандартом у розробці ігор, особливо після їх інтеграції в рушії на кшталт Unreal Engine та Unity. Серед відомих прикладів Halo 2 де Behavior Trees використовувались для координації тактичної поведінки ворогів, або Far Cry де система дозволяла створювати складні сценарії переслідування, пошуку укриттів та групової тактики.

Основною перевагою Behavior Trees є можливість комбінувати окремі елементи поведінки, швидко налагоджувати логіку і гнучко адаптувати зміни у грі. Проте навіть ці дерева мають свої межі, вони залишаються передбачуваними, і не здатні самостійно змінювати логіку без втручання розробника.

У 2010-х роках із розвитком машинного навчання почали проводитися перші експерименти з використанням reinforcement learning або навчання з підкріпленням для створення агентів, які здатні самостійно навчатися на основі досвіду. Такий підхід дозволяє NPC адаптувати поведінку в реальному часі, не покладаючись на заздалегідь визначені правила.

Прикладом є гра Montezuma’s Revenge, у якій агенти навчалися проходити складні рівні через багаторазові спроби. Гарним прикладом є система штучного інтелекту у Alien: Isolation, де поведінка головного противника не є заздалегідь запрограмованою, а змінюється динамічно аналізуючи попередні дії гравця [4].

Проте такі підходи поки що рідко використовуються у великих комерційних проєктах через обчислювальні складнощі та необхідність довготривалого навчання. Вони застосовуються переважно в наукових або експериментальних розробках і залишаються перспективним напрямом для досліджень.

1.3 Основні підходи до організації поведінки NPC

1.3.1 Лінійно-залежна поведінка

У процесі моделювання поведінки некерованих персонажів застосовуються різні підходи до побудови логіки прийняття рішень. Структура поведінки NPC у більшості випадків відповідає одному з типових патернів: лінійно-залежній, станоорієнтованій або ієрархічній, незалежно від технологічного рівня чи ігрового рушія. Ці підходи формують основу для створення простих і складних сценаріїв контекстно-залежної поведінки в сучасних іграх.

Лінійно-залежний тип поведінки базується на простих правилах: певна подія викликає фіксовану реакцію. У випадку коли гравець підходить до

персонажа, то той починає говорити. У такій моделі немає змінних станів або гнучкої логіки, лише послідовність дій які тісно залежать від умов.

Цей підхід зручно використовувати для створення сценарних сцен або персонажів, що виконують одноразові дії. Але така система не може адаптуватися до контексту. Якщо зміниться ситуація NPC не змінить свою поведінку. Як наслідок, цей тип використовується лише у другорядних взаємодіях або в іграх із лінійним перебігом подій.

1.3.2 Станоорієнтована поведінка

Станоорієнтована модель має вже більш структурований підхід який дозволяє NPC перебувати у визначених «станах», між якими можливі переходи залежно від певних умов. Кожен стан має власну поведінку, а перехід між ними ініціюється зовнішніми чи внутрішніми подіями, це може бути втрата здоров’я, поява ворога, досягнення цілі.

Таку модель зручно використовувати коли поведінку персонажа можна розкласти на зрозумілі етапи: «чекає», «виявив загрозу», «атакує», «втікає», «повертається на позицію». Для реалізації такої поведінки часто використовуються таблиці переходів або діаграми станів, що дозволяє візуалізувати логіку й ефективно її налагоджувати.

Основною перевагою такого підходу є його логічність і контрольованість. Поведінка NPC чітко визначається умовами переходу між станами. Проте зі збільшенням кількості можливих ситуацій, система стає дуже громіздкою. Додавання нових станів і зв’язків між ними призведе до ускладнення підтримки коду та зменшення гнучкості. Крім того NPC у такій системі залишаються досить пасивними, вони не передбачають наслідків, а лише реагують на зовнішні стимули.

1.3.3 Ієрархічна та модульна поведінка

Ієрархічний підхід дозволяє організувати поведінку NPC у вигляді дерева або багаторівневої структури, де високорівневі завдання поділяються на підзадачі. Така поведінка часто реалізується через так звані дерева поведінки або Behavior Trees.

Ієрархічна модель дозволяє NPC мати декілька активних гілок логіки, це відрізняє її від станів які зазвичай є взаємовиключнними. Прикладом є персонаж, який може одночасно уникати перешкод, слідкувати за гравцем і оцінювати відстань до цілі. Залежно від результатів кожного рівня аналізу поведінка формується динамічно.

Перевагами такого підходу є в першу чергу гнучкість, масштабованість і повторне використання логіки. Поведінкові гілки можна створювати окремо, а потім комбінувати між різними NPC, тобто створювати унікальні комбінації без переписування всього з самого нуля. Плюсом є той факт, що така структура дозволяє легко змінювати або розширювати поведінку після додавання нових умов або дії без переробки всієї системи.

Проте ієрархічні моделі вимагають чіткої архітектури та планування. Неправильно побудоване дерево може призвести до конфліктів, зациклення ігрової логіки або неочікуваних результатів. Також сильно зростає навантаження на продуктивність, особливо якщо в грі одночасно активні десятки NPC із великою кількістю логічних перевірок.

1.4 Системи сприйняття як основа адаптивної поведінки NPC

Під час побудови інтелектуальної поведінки некерованих персонажів важливу роль відіграють системи сприйняття. Саме завдяки ним NPC отримують уявлення про те, що відбувається навколо і приймають рішення на основі зовнішніх або внутрішніх стимулів. Сприйняття – це міст між

ігровим середовищем і логікою поведінки персонажа. Без цього компоненту будь-яка модель, не важливо чи то скрипт, станова машина або дерево поведінки, буде діяти в ізоляції, без реакції на зміну контексту.

Система сприйняття в NPC імітує здатність бачити, чути, відчувати та аналізувати сигнали з оточення. Система може охоплювати прості умови виявлення гравця або складну модель, що враховує поле зору, звукову гучність, рівень освітлення, швидкість об’єкта, кількість перешкод тощо, в залежності від її складності. Чим точніше NPC може «відчути» ситуацію, тим правдоподобнішою та реалістичнішою буде його поведінка.

Найбільш поширеними каналами сприйняття є зір і слух (рис. 1.4). Зорове сприйняття дозволяє персонажам орієнтуватися у просторі, виявляти ворогів або союзників, розпізнавати об’єкти, а також відстежувати дії гравця. Слух забезпечує реакцію на події, які можуть виходити за межі поля зору, це може бути шум кроків, постріл або кинутий об’єкт [5]. У деяких випадках, в залежності від механік гри до моделі можуть додаватися інші канали, такі як відчуття вібрації, температури або змін у магічному полі.

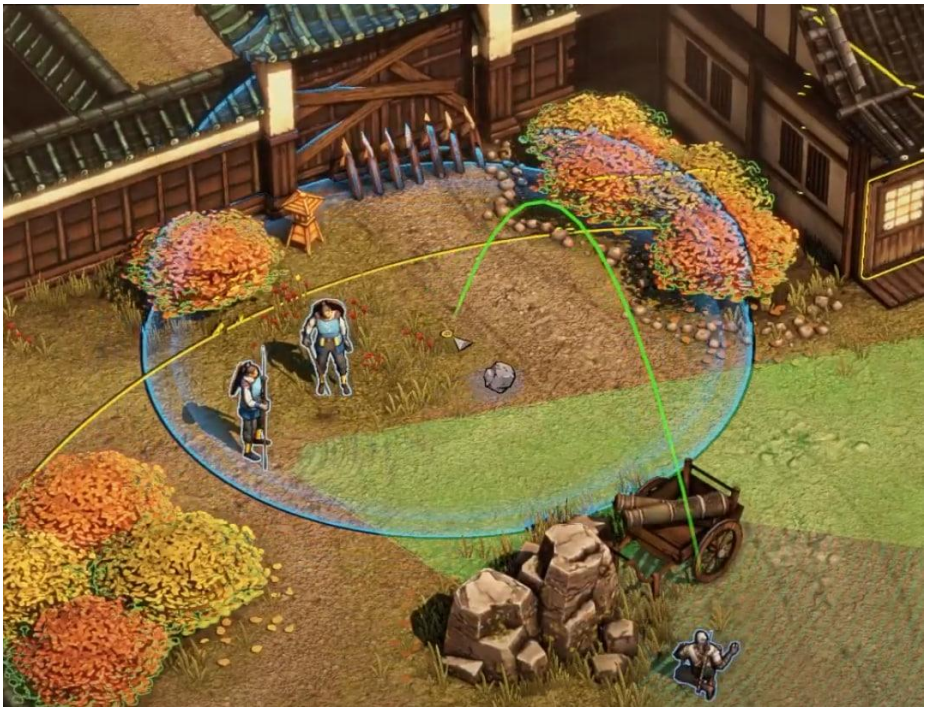

Рисунок 1.4 – Приклад зору (зелений колір) і звуку (синій колір) NPC у грі Shadow Tactics: Blades of the Shogun

Обмеженність є важливою характеристикою будь-якої системи сприйняття. NPC не може знати все що відбувається у грі, він повинен діяти лише на основі доступної йому інформації через ті сами канали. Це дозволяє досягти більшої правдоподібності і, що не менш важливо формувати виклики для гравця, які ґрунтуються на уникненні виявлення. Якщо NPC миттєво реагує на присутність гравця без будь-якої затримки така поведінка здається ненатуральною.

Іншим важливим аспектом є пам’ять NPC. Деякі системи розроблені таким чином, що NPC не просто реагує на моментальний стимул, а зберігає інформацію про нього протягом певного часу, який задає розробник. Наприклад якщо гравець зник із поля зору, NPC не забуває про нього одразу, а намагається вирушити до останнього відомого місця або продовжує пошук, якщо гравець не був знайден. Це дозволяє побудувати логіку для переслідування, дослідження або повернення до звичної поведінки після невдалого виявлення.

Також система сприйняття дозволяє враховувати контекст навколишнього середовища. NPC може не реагувати на гравця, якщо його позначить як союзника або не вважає загрозою. У цьому випадку важливим елементом є не просто наявність події, а її інтерпретація.

Роль сприйняття в поведінці NPC є однією з фундаментальних. Саме воно визначає на що звертає увагу персонаж, коли реагує на зміну ситуації та які сигнали вважає значущими. Навіть найскладніші моделі поведінки втрачають сенс без адекватного сприйняття, адже прийняття рішень NPC має спиратися на поточній, а не на повній інформації.

1.5 Постановка задачі

Відеоігри нового покоління ставлять високі вимоги до якості поведінки неігрових персонажів. Замість заздалегідь визначених сценаріїв, гравці

очікують від NPC природної та гнучкої реакції на події в грі. Це створює потребу у впровадженні адаптивних систем штучного інтелекту, здатних аналізувати поточну ситуацію, приймати самостійні рішення та коригувати свою поведінку залежно від змін середовища і дій гравця.

Об’єктом роботи є некеровані персонажі або NPC, що функціонують в ігровому середовищі. Предметом – механізми штучного інтелекту, що визначають їхню поведінку: дерево поведінки, система сприйняття, навігація, система запитів до середовища та навчальні агенти.

Метою роботи є дослідження сучасних методів створення реалістичної поведінки NPC за допомогою засобів штучного інтелекту в Unreal Engine 5.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести огляд сучасних підходів до створення штучного інтелекту NPC;

– проаналізувати можливості Unreal Engine 5 для реалізації інтелектуальної поведінки NPC, включаючи компоненти AIController, Blackboard, Behavior Tree, NavMesh та EQS;

– побудувати шаблон дерева поведінки NPC, що забезпечує реакцію на дії гравця, зміну стану середовища, пересування та бойові сценарії;

– розробити логіку взаємодії з середовищем за допомогою Environmental Query System;

– реалізувати базову систему навчання агентів;

– створити тестову карту гри в Unreal Engine 5 з інтегрованими NPC і перевірити коректність роботи системи ШІ в реальних ігрових умовах.

Особливу увагу буде приділено структурі архітектури ШІ-агентів, їхньому реагуванню на події у грі, а також можливості гнучкого налаштування поведінки для різних типів персонажів без переписування логіки. Очікуваним результатом є створення функціонального прототипу системи NPC, що може бути легко масштабована і використовуватись у повноцінному RPG-проєкті.

2 ОГЛЯД ОСНОВНИХ МЕТОДІВ СТОВРЕННЯ ШТУЧНОГО ІНТЕЛЕКТУ NPC У ВІРТУАЛЬНОМУ СЕРЕДОВИЩІ

2.1 Аналіз рушія для створення штучного ітелекту у віртуальному середовищі

Вибір рушія для реалізації поведінки неігрових персонажів є важливим стратегічним рішенням, яке визначає можливості, обмеження, темпи розробки та якість кінцевого продукту. Для ефективної реалізації інтелектуальної поведінки персонажів у кваліфікаційній роботі було обрано Unreal Engine 5 – сучасну платформу, що поєднує гнучкий інструментарій, високу продуктивність і підтримку складних систем штучного інтелекту. На відміну від альтернативних середовищ, UE5 надає не лише візуальні засоби розробки, а й потужні вбудовані механізми управління AI-логікою, що дозволяє створювати адаптивну та натурально поведінку NPC навіть у складних ігрових середовищах.

Однією з причин вибору UE5 стало те, що рушій пропонує вже інтегровану архітектуру, яка орієнтована на створення NPC з підтримкою складної поведінки без потреби у використанні сторонніх бібліотеках. Завдяки чітко сформованій структурі взаємодії між основними компонентами, такими як Behavior Tree, Blackboard, AIController, система навігації та система сприйняття розробник отримує можливість не лише швидко створити базову логіку, а й з часом нарощувати її складність. Таке архітектурне рішення дозволяє зберігати цілісність системи й уникати дублювання або надмірної жорсткості при масштабуванні функціональності NPC.

Перевагою UE5 також є зручна взаємодія між візуальним та програмним підходами. Розробник може будувати логіку поведінки використовуючи Blueprints, або через C++, а також легко поєднувати обидва підходи в рамках одного проєкту [6–8]. Це особливо важливо на етапах

прототипування і тестування, коли пластичність і швидкість ітерацій мають критичне значення. Значно спрощує процес налагодження, діагностики й оптимізації те, що додатково рушій дозволяє візуалізувати роботу логіки NPC у режимі реального часу. Завдяки цьому поведінка NPC не залишається «чорною скринькою», а стає прозорою для аналізу і її вдосконалення.

UE5 забезпечує високий рівень інтеграції компонентів, що відповідають за сприйняття оточення, побудову маршруту, обробку ситуацій та прийняття рішень. Поведінка NPC у сучасних іграх значною мірою залежить від здатності реагувати на зміну середовища. Unreal Engine 5 надає змогу враховувати ці фактори у єдиній системі, де кожен елемент взаємодіє з іншими без необхідності ручного «зшивання» логіки. Така взаємодія дозволяє створювати NPC, які не лише демонструють базову правдиву поведінку, а й можуть адаптуватися до складних сценаріїв.

Значною перевагою UE5 є також активна підтримка та стабільна екосистема. Велика кількість документації, навчальних матеріалів, подкастів і прикладів дозволяє ефективно засвоювати інструментарій та використовувати найкращі практики. Багато сучасних проєктів, в основному відомимі AAA-ігри, реалізовані саме на Unreal Engine. Це підтверджує відповідність рушія реальним вимогам ігрової індустрії, а отже також доцільність його використання в навчальних дослідженнях. Відомі приклади, такі як Fortnite, Hogwarts Legacy або Senua’s Saga: Hellblade II демонструють потужність UE5 саме у контексті побудови складної логіки NPC та глибокої інтеграції зі світом гри.

2.2 Традиційні підходи до моделювання поведінки NPC

2.2.1 Реалізація логіки NPC за допомогою дерев рішень

У процесі розробки інтелектуальної поведінки некерованих персонажів важливою є не лише загальна ідея їхнього реагування на події в грі, а й

глибоке розуміння технічних інструментів які використовуються для реалізації цієї поведінки. Адже без правильного підходу до побудови архітектури штучного ітелекту навіть найцікавіша задумка може не виправдати очікувань. Ігрові рушії надають розробникам широкий арсенал засобів і механізмів, які дозволяють створювати гнучкі, адаптивні та легко масштабовані системи штучного інтелекту для NPC що можуть ефективно взаємодіяти з оточенням та гравцем у режимі реального часу.

Один із базових підходів, реалізований в Unreal Engine 5 це використання дерев поведінки або Behavior Tree [9] (рис. 2.1). Цей інструмент дозволяє будувати логіку прийняття рішень у вигляді структури, яка ієрархічно розділяє поведінку персонажа на послідовні логічні етапи. Такий підхід забезпечує розробникам високий рівень модульності. Зміна окремих гілок дерева або доповнення новими діями не вимагає глобального переписування існуючої логіки, що значно сильно пришвидшує розробку і полегшує підтримку проєкту. Кожен вузол дерева може відповідати за конкретну перевірку умови або виконання певної дії, а також об’єднання кількох підзадач у комплексні поведінкові блоки.

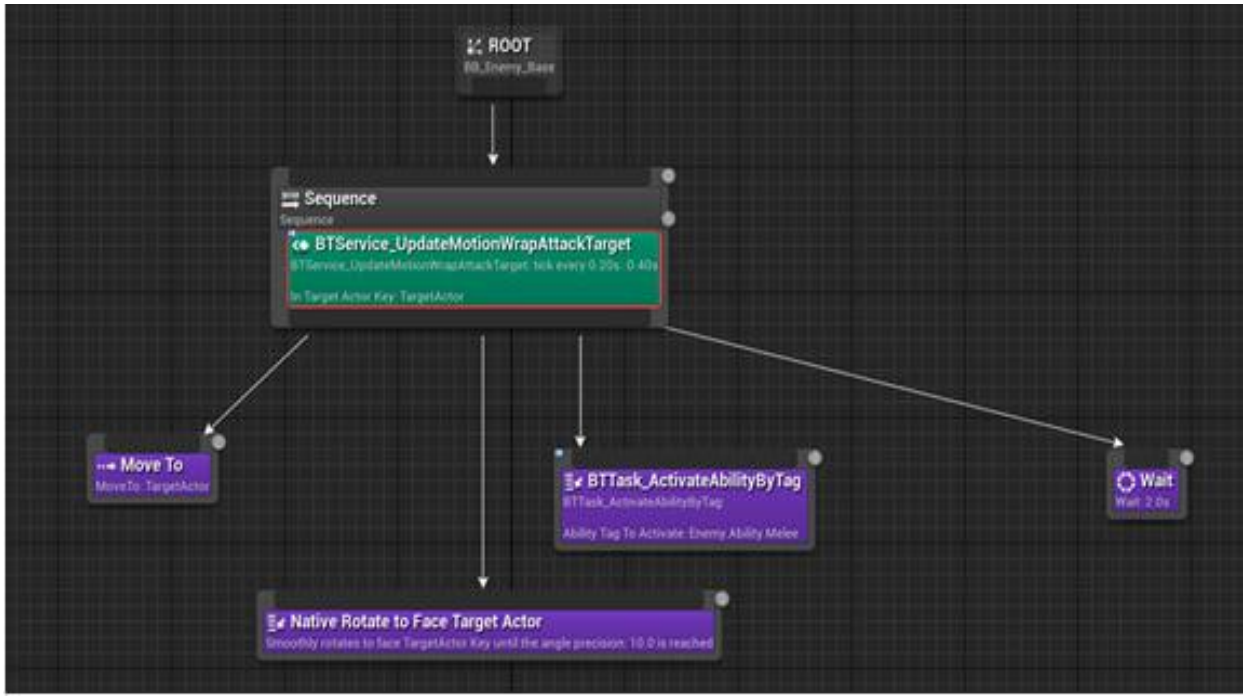

Рисунок 2.1 – Приклад простого дерева рішень

Behavior Tree у UE5 тісно інтегрований із системою Blackboard, яка слугує як центральне сховище даних для NPC. Blackboard виступає у ролі «пам’яті» персонажа, у ній зберігаються всі необхідні змінні. В залежності від жанру може бути поточне місце розташування гравця, рівень загрози, активні завдання або стан тривоги. Завдяки такій структурі даних вузли дерева можуть адаптувати свою поведінку динамічно, відповідно до змін у середовищі гри. Виявивши гравця у полі зору NPC може оновити координати цілі у Blackboard і негайно активувати гілку дерева, що відповідає за переслідування. Замість передачі змінних у кожен вузол дерева вручну, розробник просто вказує, яку інформацію слід зчитувати або оновлювати з Blackboard. Це значно спрощує архітектуру логіки і дозволяє реалізувати адаптивні сценарії поведінки.

Принцип роботи Behavior Tree полягає у послідовному проходженні по логіці. Починаючи з кореневого вузла система поступово спускається до підлеглих гілок, при цьому аналізує умови і виконує відповідні дії. Кожен вузол має свій статус, який може бути «успіх», «невдача» або «виконується». Ці статуси визначають подальший напрямок обробки, в залежності чи варто переходити до іншої гілки, чи потрібно завершити виконання поточної задачі. Серед основних типів вузлів можна виділити селектори або Selector, які перебирають підвузли доти, доки один із них не поверне статус «успіх», а також послідовності або Sequence, що виконують підвузли один за одним до момента першої невдачі. Окрім цього, задачі або Tasks реалізують конкретні дії NPC, від простого руху до складних анімацій чи бойових маневрів.

AI Controller у свою чергу, виконує роль центрального керуючого механізму або «мозком» системи. Це спеціалізований клас, який бере на себе управління NPC, також займається ініціалізацією дерева поведінки, реагує на події, що надходять від систем сприйняття і взаємодіє з іншими частинами рушія. Саме AI Controller відповідає за старт поведінки, обробку змін у середовищі та координацію між Behavior Tree, Perception System, NavMesh і Blackboard. Як приклад, якщо NPC втратив з поля зору гравця, то саме AI

Controller може ініціювати пошук останньої відомої позиції або повернення до патрулювання. Він діє як міст між ігровою логікою та інтелектуальними модулями і забезпечує автономність дій персонажа.

2.2.2 Застосування Environmental Query System для адаптивної поведінки NPC

Іншим важливим підходом є Environmental Query System або система запитів навколишнього середовища. Це інструмент, що дозволяє NPC здійснювати динамічний аналіз навколишнього середовища. Завдяки EQS некерований персонаж може обрати найкращу точку для укриття, проаналізувати можливі шляхи наближення до ворога або визначити місце звідки найбільш вигідно вести атаку [10, 11].

Принцип роботи EQS полягає у формуванні запитів або queries, які містять генератори точок і тести, що фільтрують і оцінюють ці точки. Генератори визначають які точки слід проаналізувати, як приклад усі навколишні локації в радіусі 1000 одиниць, а тести перевіряють ці точки за такими параметрами як відстанню, наявністю прямої видимості, висотою, близькістю до ворога тощо. Кожна точка отримує оцінку, яка визначає її придатність для виконання задачі (рис. 2.2).

Кожен тест може мати вагу, яка вказує на його пріоритетність. Якщо NPC шукає укриття, то тест на наявність прямої видимості з боку ворога може мати більшу вагу ніж відстань до цієї точки. Це дозволяє некерованому персонажу робити більш зважені рішення, при цьому комбінуючи кілька критеріїв при виборі дій. Unreal Engine дозволяє налаштовувати тип перевірки і навіть писати власні тести для специфічних сценаріїв. Схема принципу дії Environmental Query System зображена на рисунку 2.3.

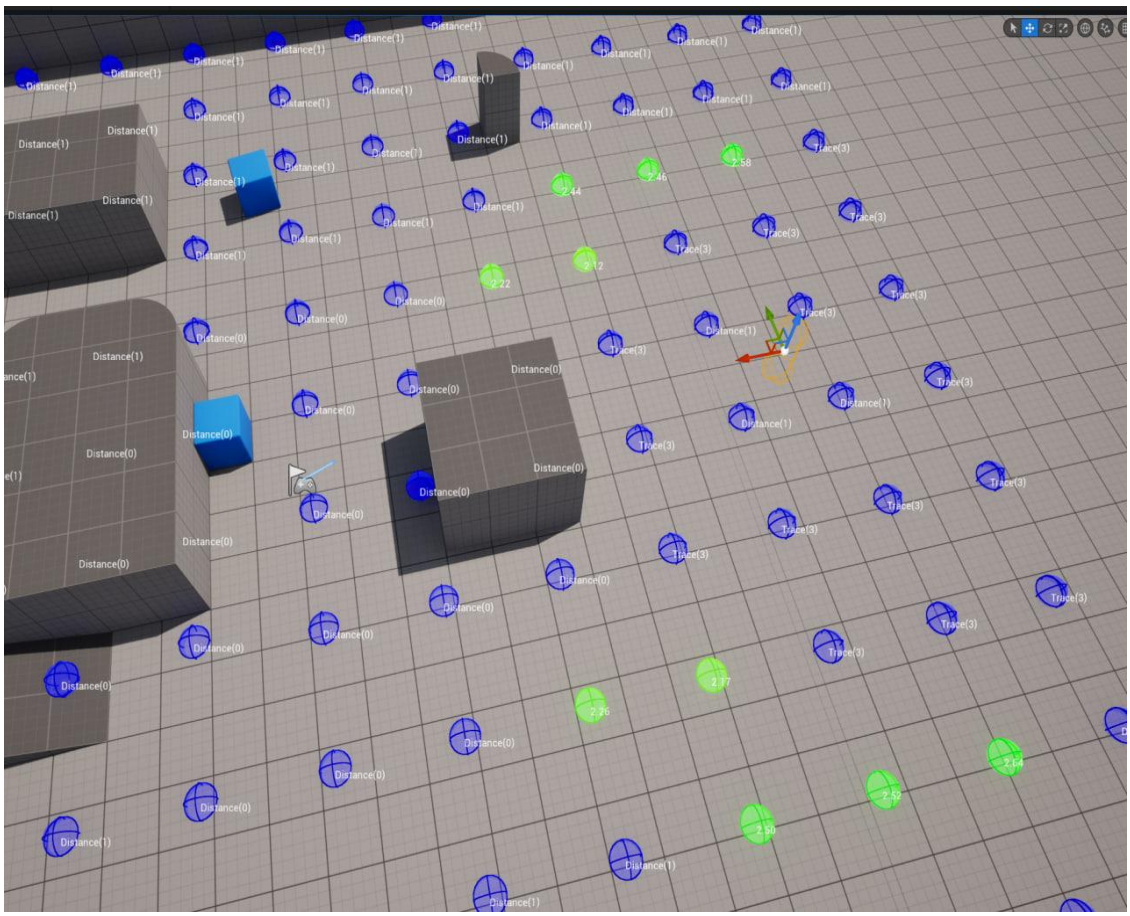

Рисунок 2.2 – Візуалізація EQS

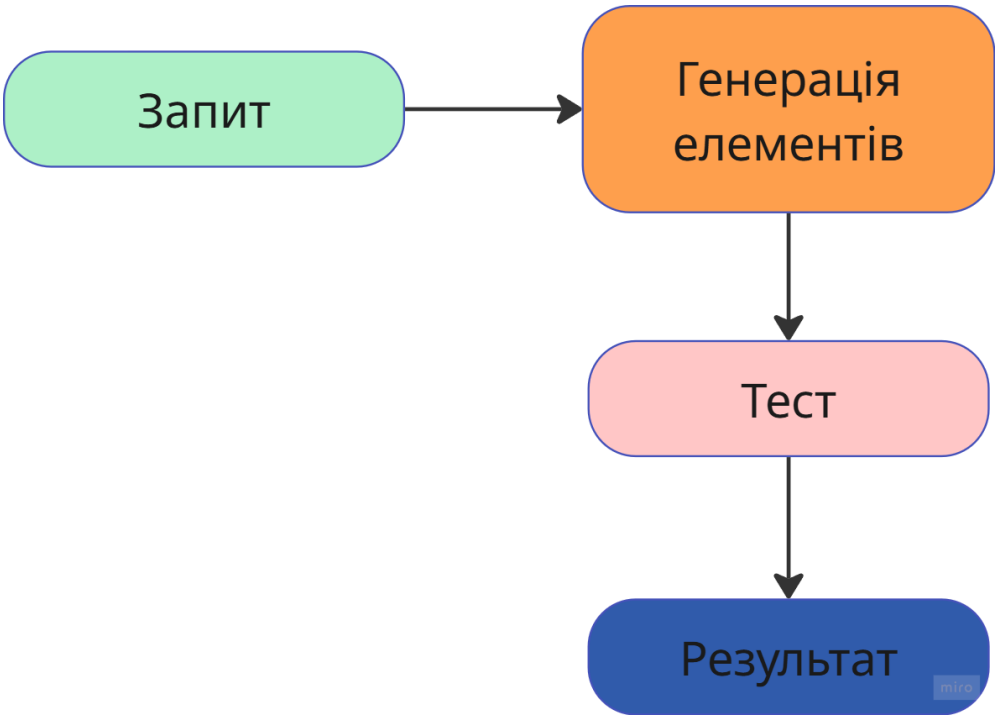

Рисунок 2.3 – Схема принципу дії EQS

Процес налаштування EQS у UE5 є візуальним. Запити створюються як окремі об’єкти EQS Query, які можна приєднати до поведінкових вузлів у Behavior Tree. В одному з вузлів NPC може виконати запит EQS на «найближче безпечне укриття» і результат буде записаний у Blackboard. Далі Behavior Tree використовує це значення, щоб задати ціль для пересування.

EQS також підтримує динамічну зміну параметрів під час гри. Це означає, що значення такі як радіус пошуку чи тип фільтрації, можна задавати змінними з Blackboard або Blueprint, що дозволяє NPC адаптуватися до нових умов (наприклад, шукати укриття лише тоді, коли рівень здоров’я нижчий за 30%).

Комбінація Behavior Tree, Blackboard і EQS створює зручну екосистему для управління NPC. Коли персонаж виявляє гравця, ця подія фіксується у Blackboard, після чого активується гілка Behavior Tree, яка ініціює запит EQS на пошук укриття, після чого NPC використовує NavMesh для переміщення до обраної точки. Уся ця логіка функціонує без потреби жорсткого скриптування, що зменшує ймовірність помилок і підвищує масштабованість проєкту.

2.3 Основні поняття машиного навчання

Машинне навчання є підгалуззю штучного інтелекту, яка дозволяє комп’ютерним системам автоматично покращувати свої характеристики через досвід [10–13]. На відміну від класичних алгоритмів, де кожен крок логіки чітко визначено програмістом, у машинному навчанні модель самостійно формує правила на основі наданих даних.

Узагальнено, процес машинного навчання включає кілька ключових етапів. На першому етапі здійснюється збір даних, тобто спостережень за середовищем, в якому працює агент. Це можуть бути координати об’єктів, рівень здоров’я, наявність перешкод, швидкість переміщення тощо. Далі

будується модель, яка здатна знаходити зв’язки між цими спостереженнями та діями, що приводять до бажаного результату. Наприкінці відбувається оцінка ефективності моделі, вона тестується в нових умовах, де перевіряється, наскільки вдало сформовані правила допомагають досягати мети.

Розрізняють кілька типів машинного навчання залежно від підходу до обробки даних. Якщо модель навчається на основі прикладів, де вже відомий правильний результат, це називається навчанням з учителем. Якщо ж вона працює із «чистими» даними без міток, намагаючись виявити закономірності, то мова йде про навчання без учителя. Існує навчання з підкріпленням, коли агент самостійно відкриває закономірності у процесі взаємодії зі світом, отримуючи винагороду за вдалі дії та штраф за помилки [14–18]. Саме цей підхід найбільше наближений до реальної поведінки у грі та є найбільш перспективним у сфері ігрового ШІ.

Ключовим компонентом багатьох систем машинного навчання є нейронна мережа. Це структура, натхненна принципами функціонування людського мозку, яка складається з рядів взаємопов’язаних штучних нейронів. Кожен із них приймає сигнали, трансформує їх через функцію активації, і передає результат далі. Такі мережі здатні моделювати складні залежності між змінними та застосовуються для розв’язання широкого кола завдань від розпізнавання образів до прийняття рішень у реальному часі. Актуальні тенденції застосування глибинного навчання в завданнях обробки детально проаналізовано у роботі [19], де розглянуто сильні сторони різних моделей та їх ефективність.

Навчання нейронної мережі здійснюється за допомогою спеціального алгоритму зворотнього поширення помилки. Після кожного епізоду взаємодії система порівнює прогнозовані результати з реальними й відповідно коригує внутрішні параметри або ваги зв’язків. Повторюючи цей процес тисячі разів, модель поступово зменшує помилку та покращує точність своїх передбачень.

У сучасних системах використовуються також такі техніки, як стохастичний градієнтний спуск, функції втрат і механізми регуляризації.

На сьогоднішній день машинне навчання дедалі частіше знаходить практичне застосування у сфері комп’ютерного зору, аналізу відео та ігрового ШІ. Наприклад, у роботах [20–23] продемонстровано ефективність використання глибоких згорткових нейронних мереж для класифікації зображень та відео, що підкреслює потенціал цих технологій у задачах розпізнавання об’єктів та прийняття рішень. Додатково, у роботі [24] розглянуто логічні мережі як альтернативу при моделюванні поведінки в умовах обмеженої кількості входів, а в дослідженні [25] виконано порівняння традиційних рекурентних мереж і Long Short-Term Memory у контексті обробки часових послідовностей.

Для ігрових застосунків машинне навчання є особливо цінним завдяки здатності створювати гнучку, непередбачувану поведінку NPC. Агент може поступово вчитися уникати атак, ефективно користуватися укриттями, адаптувати свою тактику до стилю гравця, а також формувати унікальні стратегії, які розробник не програмував явно. Це збагачує геймплей, робить його більш живим та індивідуалізованим.

Окрім нейронних мереж, у практиці моделювання інтелектуальної поведінки застосовуються й альтернативні математичні підходи, зокрема методи оптимізації у нечітких системах. Наприклад, у роботі [26] описано модифікацію методу гілок і меж для знаходження екстремумів функцій належності у нечітких інтелектуальних системах. Актуальні можливості обчислення спеціальних функцій з інтервальними аргументами, поданими в гіперболічній формі, що також можуть бути корисні при моделюванні складних залежностей, представлені у дослідженні [27]. Такі підходи можуть бути корисними у завданнях, де важливо враховувати невизначеність або нечіткість вхідних даних, що трапляється, зокрема, у поведінковому моделюванні NPC в умовах нестачі точних даних.

Загалом, машинне навчання є потужним інструментом, що дозволяє моделювати складну поведінку у динамічних середовищах. Застосування цього підходу в комп’ютерних іграх відкриває нові горизонти у розробці адаптивного ШІ, який здатен не просто реагувати, а й еволюціонувати разом із гравцем.

2.4 Механізм навчання агентів з підкріпленням

Окрім класичних підходів до створення штучного інтелекту в Unreal Engine 5 є можливість використання Learning Agents плагіну, який реалізує механізм навчання з підкріпленням (Reinforcement Learning) [28]. Цей підхід дозволяє агенту самостійно адаптуватися до навколишнього середовища, а також приймати рішення на основі досвіду та змінювати свою поведінку залежно від ситуації.

У Unreal Engine 5 плагін Learning Agents надає готовий набір рішень для навчання агентів, які не просто діють за заздалегідь визначеними правилами, а навчаються оптимальним діям шляхом багаторазової взаємодії зі світом гри. Плагін включає такі компоненти як LearningAgentComponent який є ядром агента, LearningPolicy або нейронна мережа, LearningReward тобто функція винагороди, LearningComponent або реєстрація сенсорних даних та LearningExperience, що є буфером для збереження епізодів.

Агент у грі спостерігає за навколишнім середовищем через системи сприйняття, які можуть передавати координати, швидкість, напрямок, наявність цілі або перешкод. Ці спостереження утворюють вектор стану. На основі цього стану нейронна мережа агента формує дію, яку він вважає найбільш влучною.

Після виконання дії агент отримує винагороду. Якщо дія була корисною, тобто досягнуто мети або уникнено перешкоди, то вона позитивна. Якщо ж навпаки, то штраф. Ці дані накопичуються в пам’яті

досвіду ExperienceBuffer і використовуються для оновлення нейронної мережі під час тренування. Принцип дії навчання з підкріпленням в Unreal Engine 5 зображено на рисунку 2.4.

У плагіні Learning Agents нейронна мережа реалізована як клас ULearningPolicy. Це звичайна нейронна мережа, яка складається з вхідного шару, 2 або 3 прихованих шарів та вихідного шару. Приховані шари зазвичай мають від 64 до 256 нейронів із функцією активації ReLU (випрямлена лінійна функція активації). Rectified Linear Unit – це одна з найпопулярніших функцій активації в нейронних мережах, яка використовується, щоб надати мережі нелінійність, необхідну для навчання складних залежностей між вхідними і вихідними даними.

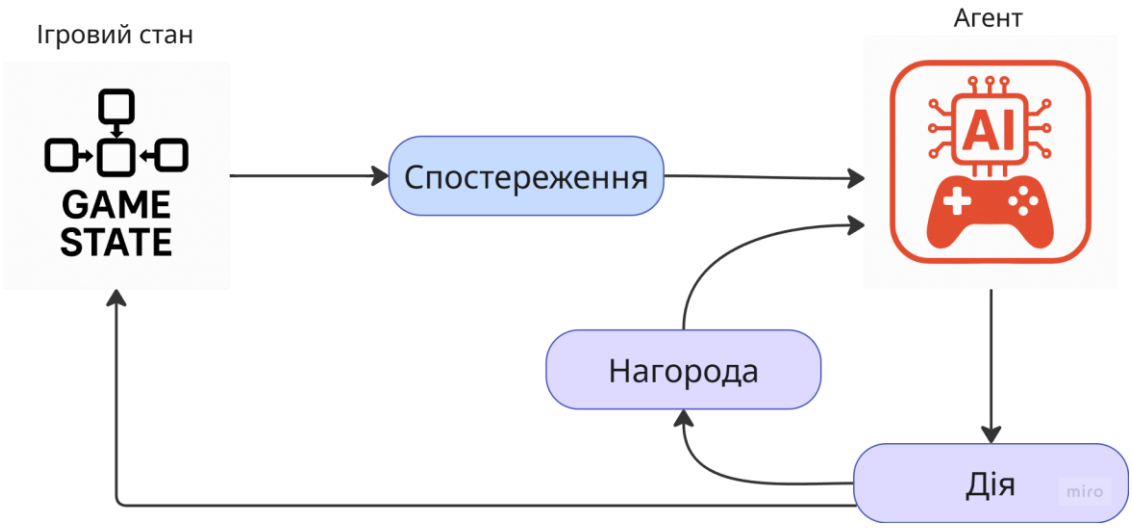

Рисунок 2.4 – Принцип дії навчання з підкріпленням в Unreal Engine 5

Вхідний шар відповідає кількості сенсорних даних, а вихідний кількості доступних дій агента. Вихідні значення інтерпретуються як ймовірності, з яких вибирається дія. Навчання здійснюється методом зворотного поширення помилки з оновленням ваг через алгоритм стохастичного градієнтного спуску.

Оптимізація політики з наближенням (Proximal Policy Optimization) є основним алгоритмом, реалізованим у плагіні Learning Agents для Unreal Engine 5 [29]. Його головною метою є покращення політики агента без різких

змін, які могли б дестабілізувати навчальний процес. У процесі навчання агент накопичує нові епізоди взаємодії зі світом, після чого здійснюється оновлення нейронної мережі. Для цього використовується спеціальна функція втрат, яка порівнює стару і нову політики та обмежує занадто великі оновленн. Завдяки такому підходу розробник досягає стійке і поступове вдосконалення поведінки агента, навіть у складних або динамічних умовах середовища.

Актор-критик (Actor-Critic) – це комбінований підхід до навчання, в основі якого лежить розділення функцій вибору дії та оцінки її якості [29]. У цій архітектурі один компонент Actor відповідає за формування дій, які має виконати агент, а інший Critic оцінює наскільки ці дії були ефективними в контексті досягнення цілі. Взаємодія між цими двома компонентами дозволяє агенту краще враховувати довготривалі наслідки своїх рішень і коригувати політику відповідно до отриманого досвіду.

На першому етапі агент має випадково ініціалізовану мережу і здійснює дії наосліп, тобто здійснює exploration. Він взаємодіє з оточенням, отримує винагороди і поступово накопичує інформацію про те, які дії приводять до кращих результатів. Це дозволяє системі зрозуміти контекст середовища.

Після завершення кожного епізоду зібрані дані зберігаються в буфер досвіду. Потім запускається фаза оновлення політики, де мережа навчається на основі отриманих даних. Для цього обчислюється функція втрат, яка залежить від того, наскільки очікувана винагорода відрізняється від реальної, і мережа оновлює свої ваги.

Починаючи з версії Unreal Engine 5.3, Learning Agents підтримують не лише навчання з підкріпленням, а й навчання на основі прикладу (Imitation Learning), що дозволяє агенту відтворювати поведінку гравця або іншого NPC без необхідності явного визначення функції винагороди (рис. 2.5). У такому підході можна вручну записати дії користувача за допомогою інструменту Learning Recorder, після чого модель тренується шляхом

імітування цієї поведінки. Наприклад, якщо гравець демонструє ефективну тактику ухилення або стрільби, агент зможе її повторити після кількох ітерацій тренування.

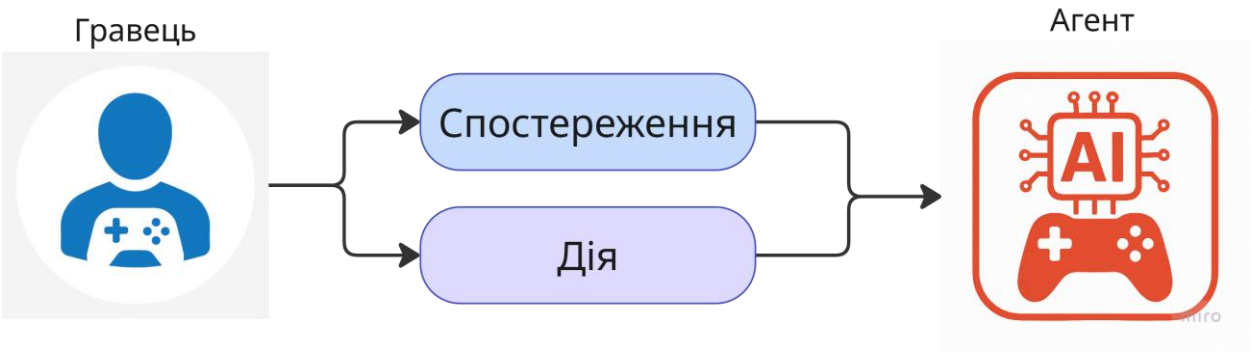

Рисунок 2.5 – Принцип дії навчання на основі прикладу в Unreal Engine 5

Варто також враховувати питання оптимізації обчислювальних ресурсів під час реалізації складних AI-систем, особливо в умовах багатокористувацьких або розподілених середовищ. У роботі [30] підкреслюється важливість підвищення продуктивності серверної частини інформаційних систем, що може бути релевантним і для архітектури ігрових серверів з інтелектуальними NPC.

Ще однією перевагою Learning Agents є можливість поєднання з іншими AI-системами UE5, такими як Behavior Tree або EQS. Behavior Tree може надавати високорівневі задачі, тоді як Learning Agent визначає точну дію на основі тренованої політики. EQS може надавати список можливих точок для укриття чи атаки, а агент вибирати найкращу з них, керуючись попереднім досвідом. Всі ці системи розроблені для кращого досвіду використання і створювання ШІ у грі.

2.5 Використання навігаційної сітки (NavMesh) для забезпечення мобільності NPC

У сучасних іграх загальною практикою є реалізація інтелектуальної поведінки NPC у тривимірному ігровому середовищі, організація навігації, тобто здатність персонажа знаходити шлях до заданої цілі з урахуванням перешкод, змін ландшафту, багаторівневих структур та інших динамічних факторів. В Unreal Engine 5 цей процес реалізується за допомогою механізму Navigation Mesh, навігаційної сітки (рис. 2.6), яка визначає допустимі області для пересування некерованого персонажа [31].

Команда MoveToLocation або MoveToActor ініціює шлях до вказаної цілі, яка може бути або координатою в просторі, або іншим об’єктом. AIController використовує Pathfinding – алгоритм пошуку шляху, зазвичай A* або його варіації, який аналізує NavMesh і формує оптимальний маршрут до цілі з урахуванням усіх допустимих зон [32].

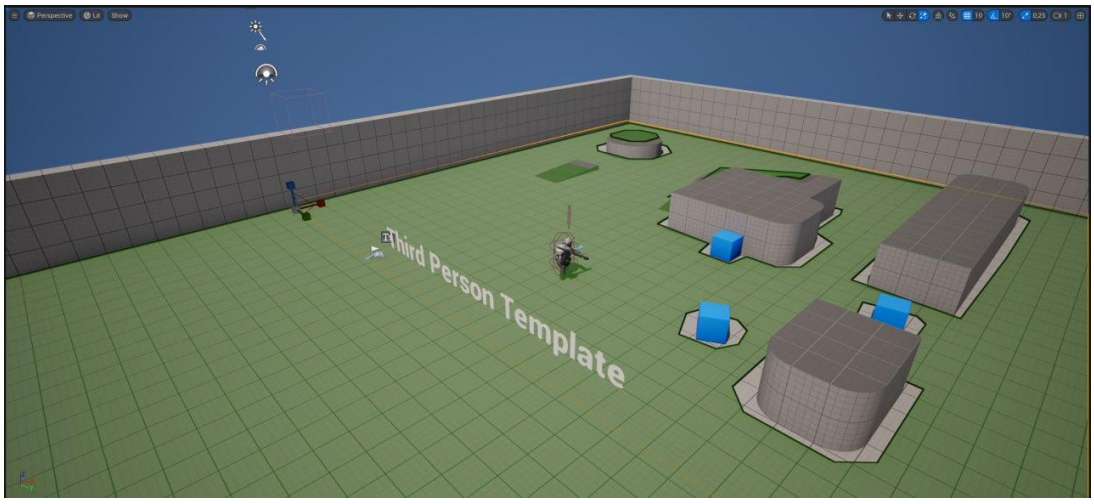

Рисунок 2.6 – Навігаційна сітка у Unreal Enigne 5

Основні етапи виконання AI MoveTo:

Крок 1. NPC визначає ціль Vector або Actor.

Крок 2. Відправляється запит до системи навігації.

Крок 3. Навігаційна система формує шлях по сітці NavMesh.

Крок 4. Рух NPC координується по сегментах маршруту.

Крок 5. При досягненні цілі або в разі помилки виконується відповідна реакція.

Цей процес інтегрується з Behavior Tree через вузли MoveTo або Simple MoveTo, які реагують на значення в Blackboard [33].

Unreal Engine 5 підтримує мультиповерхову навігацію, що є дуже важливою для ігор із вертикальною геометрією, такі як башти, багатоповерхові будівлі, ліфти тощо. NavMesh автоматично генерує окремі «шари» сітки для різних рівнів, якщо між ними є з'єднання у вигляді сходів. Таким чином, NPC може вільно переходити між рівнями і не потребувати окремої логіки маршрутизації. Це особливо корисно у великих локаціях RPG-ігор, де гравець і NPC можуть переміщуватися між декількома поверхами, або між площінами.

Однією з особливостей UE5 є підтримка динамічного оновлення NavMesh у відповідь на зміну ігрової сцени. Така відмінність дає можливість NPC правильно реагувати на появу нових об'єктів або навпаки зникнення перешкод, відкриття або закриття зон. Для цього в UE5 існують NavModifierVolume та NavLinkProxy – інструмент, який дозволяє NPC стрибати, переходити з одного рівня на інший навіть без фізичного з'єднання. Такі зміни особливо важливі в екшенах і стелс-іграх, де ситуація змінюється в реальному часі і зазвичай одразу у декількох гравців у режимі мультиплеєру.

UE5 підтримує Avoidance System – систему уникнення зіткнень між NPC [34]. Хоча NavMesh дозволяє будувати шлях, ця система забезпечує плавне розходження NPC у вузьких коридорах або при перетині маршрутів. Вона реалізується на рівні CharacterMovementComponent, який враховує присутність інших об'єктів і коригує траєкторію руху.

2.6 Використання Gameplay Ability System для розширення поведінки штучного інтелекту

Gameplay Ability System є потужним плагіном Unreal Engine 5, який дозволяє будувати складні системи здібностей, ефектів і механік взаємодії між об'єктами [35]. GAS розроблявся переважно для реалізації здібностей і станів гравців у багатокористувацьких проєктах, проте його можливості надзвичайно ефективні і для створення інтелектуальної поведінки некерованих персонажів.

Перевагою використання GAS для некерованих персонажів є чітке відокремлення здібностей від базової логіки персонажа. У традиційних підходах до створення штучного інтелекту кожна нова дія або сценарій потребували доповнення Behavior Tree або скриптової логіки. Натомість завдяки Gameplay Ability System усі можливі дії NPC оформлюються як окремі Ability-класи, що дозволяє централізовано керувати діями, незалежно від їх складності, а також можна легко створювати комбінації інтеракції із середовищем. Структура взаємодії компонентів Gameplay Ability System зображена на рисунку 2.7.

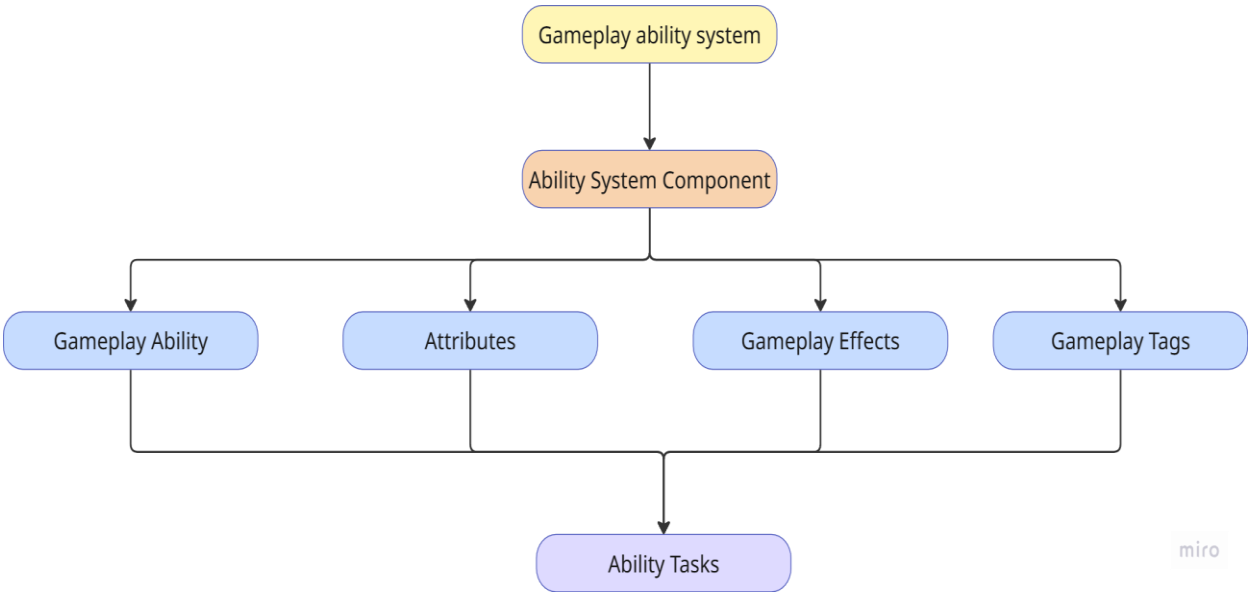

Рисунок 2.7 – Структура взаємодії компонентів Gameplay Ability System

Корисним елементом Gameplay Ability System є концепція Gameplay Effect (геймплейних ефектів). Gameplay Effect – це спеціальний об’єкт, який дозволяє змінювати характеристики або стани персонажа. За допомогою ефектів можна впливати на базові параметри, такі як здоров’я, витривалість, швидкість руху, а також накладати різні статуси. Важливою особливістю є те, що ефекти можуть бути різними по тривалості і по характеристикам. Також для посилення сили статусу при багаторазовому накладанні, ефекти підтримують стеки, тобто механізм накопичення ефектів.

Додатковим механізмом у Gameplay Ability System є Gameplay Tags. Gameplay Tag можна уявити як умовна «мітка», яка описує стан або властивість об’єкта. Теги можуть використовуватися для перевірки умов активації здібностей, накладання ефектів або визначення типу взаємодії між об’єктами. Теги в GAS є ієрархічними, що дозволяє будувати складні логічні залежності. Наявність або відсутність певних тегів визначає доступність дій, вплив ефектів і реакцію на події в грі. Це робить систему надзвичайно гнучкою і придатною для динамічної зміни поведінки NPC залежно від контексту.

Gameplay Ability System також містить в собі Attributes. Це набір базових характеристик персонажа, таких як здоров’я, витривалість, мана, сила атаки тощо. Усі зміни стану NPC, які відбуваються внаслідок застосування ефектів або здібностей, впливають саме на відповідні атрибути. Завдяки цьому GAS забезпечує централізовану і контрольовану систему обробки параметрів персонажа, яка дозволяє в реальному часі реагувати на зміну стану без необхідності вручну відстежувати значення кожної характеристики.

Важливу роль у реалізації інтелектуальної поведінки некерованих персонажів відіграє також інтеграція Gameplay Ability System із Behavior Tree. Завдяки спеціальним вузлам активування здібностей Activate Ability Task NPC може викликати певну здібність залежно від результатів аналізу середовища за допомогою Environmental Query System або змінних

Blackboard. Наприклад, якщо виявлено ворога на близькій відстані, Behavior Tree може ініціювати активацію здібності атаки ближнього бою, якщо здоров’я NPC падає нижче заданого порогу, автоматично активується здібність використання захисної навички.

Додатково, завдяки системі Gameplay Effects стало можливим швидко змінювати характеристики NPC у відповідь на зміну ситуації у грі. При завершенні ефекту всі зміни автоматично скидаються, таким чином немає необхідності ручного керування цими станами в коді.

Застосування Gameplay Ability System у роботі дозволяє також оптимізувати систему ресурсів некерованих персонажів. Усі витрати витривалості, мани чи спеціальних енергетичних шкал управляються через Gameplay Attributes, спеціалізовану систему властивостей, яка інтегрується з GAS і дозволяє точно контролювати стан ресурсів у реальному часі. Це дає змогу NPC приймати рішення про використання здібностей на основі зовнішнього контексту і внутрішніх обмежень.

3 РЕАЛІЗАЦІЯ ПРОТОТИПУ ГРИ ТА МОДЕЛЮВАННЯ ІНТЕЛЕКТУАЛЬНОЇ ПОВЕДІНКИ НЕКЕРОВАНИХ ПЕРСОНАЖІВ

3.1 Розробка архітектури головного ігрового персонажа

На першому етапі створення гри досить важливим завданням стало визначення та побудова архітектури головного ігрового персонажа. Саме він виступає центральною фігурою у взаємодії гравця з віртуальним світом і виконує роль ключової ігрової одиниці. Побудова персонажа потребує ретельного підходу до реалізації базових механік, таких як рух чи атака. Також клас головного персонажа повинен містити інтеграцію його з іншими системами гри, включаючи інтерфейс користувача, систему здібностей, анімаційний каркас, систему введення та бойову логіку [36]. Структурований підхід до реалізації персонажа дозволяє закласти надійну основу для масштабування та подальшого розвитку проєкту.

Головний ігровий персонаж реалізований у класі AESHeroCharacter, який є нащадком базового класу AESBaseCharacter. Такий підхід дозволяє розділити функціональність на загальну для всіх персонажів, та специфічну, характерну лише для головного героя. Наслідування та інкапсуляція дають можливість повторно використовувати код, зменшуючи кількість помилок і прискорюючи розробку.

У класі AESBaseCharacter реалізована базова логіка, необхідна для будь-якого ігрового персонажа. Зокрема, тут ініціалізуються компоненти AttributeSet і MotionWarpingComponent [37]. Перший компонент відповідає за обробку здібностей, які персонаж може активувати під час гри. Другий є набором атрибутів як здоров’я, які є невід’ємною частиною ролевої системи. Компонент MotionWarping відкриває можливості для динамічного коригування анімацій руху персонажа, що особливо корисно під час створення ефектних бойових сцен або складних маневрів.

Головний герой, що описаний у класі AESHeroCharacter розширює функціональність базового класу, додаючи до нього компоненти, специфічні для користувача: HeroCombatComponent, HeroUIComponent, SpringArmComponent і CameraComponent (рис. 3.1). Завдяки SpringArmComponent камера від третьої особи може плавно змінювати позицію та кут огляду, слідуючи за персонажем із заданим зміщенням. Це створює візуально приємний досвід для гравця та дозволяє зберігати контроль над ситуацією в грі.

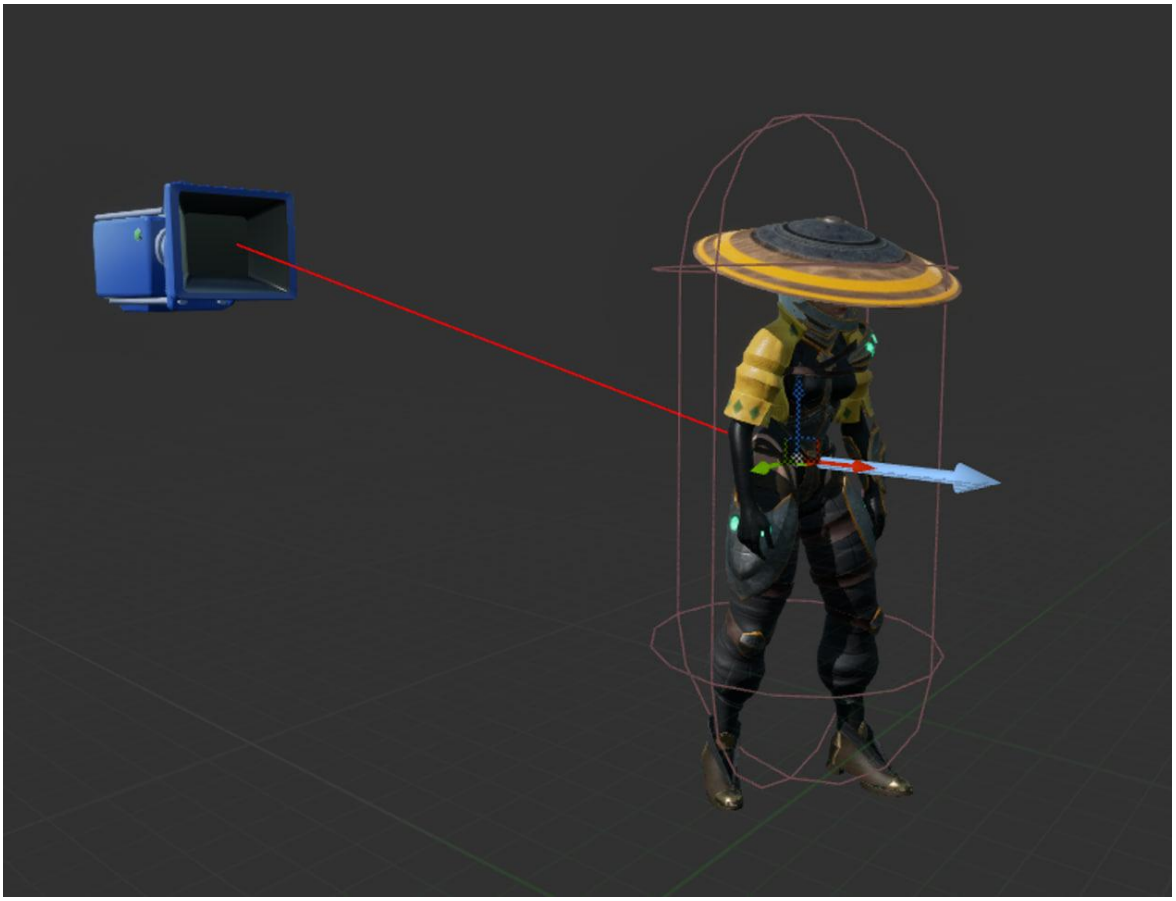

Рисунок 3.1 – Візуалізація CameraComponent та SpringArmComponent та фізичної капсули персонажа

Крім того, інтегрується AbilitySystemComponent, який керує здібностями персонажа за допомогою Gameplay Ability System. Головний персонаж також має власний Combat Component і UI Component [38], які реалізовані як окремі спеціалізовані класи.

Компонент HeroCombatComponent реалізує повний набір функцій для бойових дій. Він дозволяє персонажу атакувати, блокувати, вибирати цілі, а також отримувати та обробляти вхідні пошкодження. Через нього також активуються анімаційні монтажі, що забезпечує точне й синхронне відтворення бойових дій. Компонент працює разом з Ability System, це робить бойову систему більш варіативною та адаптивною до змін у геймплеї.

Паралельно з бойовим модулем HeroUIComponent виконує роль посередника між внутрішнім станом персонажа та гравцем. Компонент відображає інформацію про здоров’я, поточну зброю та інші параметри, що мають критичне значення для прийняття рішень під час гри. Він автоматично оновлюється при зміні значень атрибутів завдяки підписці на події Ability System. Це дозволяє досягти реального часу відображення без зайвого кодування. Ініціалізація атрибутів здійснюється при старті гри шляхом застосування початкових Gameplay Effects, які встановлюють базові значення цих параметрів.

Функції руху, такі як MoveForward, MoveRight, та Jump обробляють введення гравця і додають відповідний рух до персонажа через функцію AddMovementInput. Додаткові дії, такі як атаки або активація здібностей, прив’язані до відповідних кнопок введення через метод SetupPlayerInputComponent [39].

Окремої уваги заслуговує анімаційна система. Animation Blueprint ABP_Hero (рис. 3.2), використаний для головного персонажа, побудований на основі State Machine, яка відповідає за зміну станів залежно від поведінки персонажа [40]. Використання Blend Space дозволяє плавно змінювати анімації між різними швидкостями руху, а Animation Montages застосовуються для бойових дій та інших спеціалізованих станів. Крім того, анімації структуруються за допомогою Animation Layers, тобто кожен шар відповідає за конкретний тип зброї, що дає можливість додавати нові варіанти без переписування всієї системи.

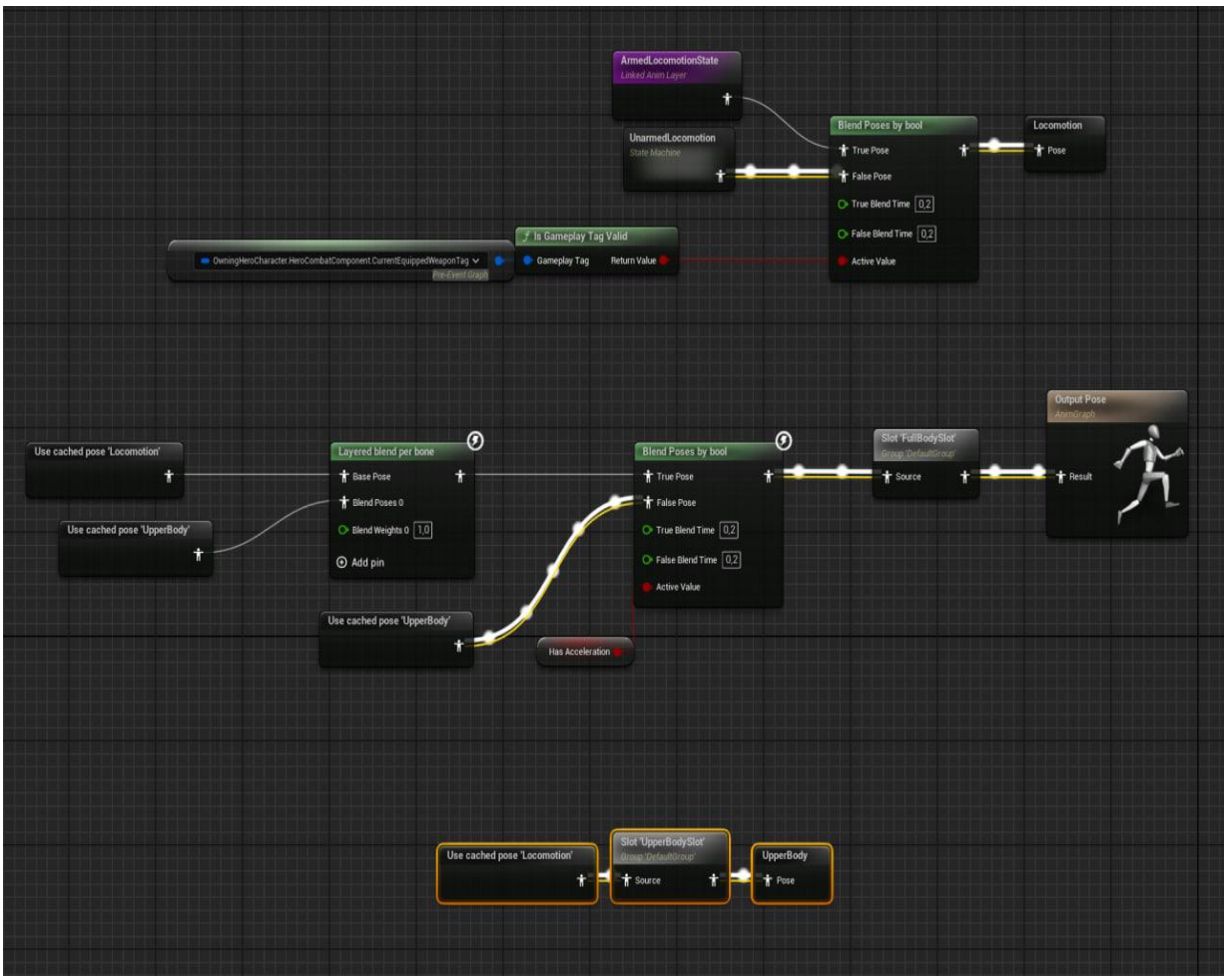

Рисунок 3.2 – Анімаційний блупринт головного персонажа ABP_Hero

Blend Space (рис. 3.3) забезпечує адаптивну зміну анімацій залежно від швидкості пересування [41]. Для спеціалізованих дій, таких як атаки або ухилення, використовуються Animation Montages, які можуть бути викликані через Combat Component або Ability System. Анімаційний Blueprint використовує шари Animation Layers для кожного типу зброї, що є безумовним плюсом. Це рішення дозволяє легко масштабувати гру, додаючи нові типи озброєння без необхідності переписувати всю анімаційну логіку. Кожен шар відповідає за окремий набір анімацій відповідно до активного виду зброї, що забезпечує гнучкість і розширюваність системи анімацій.

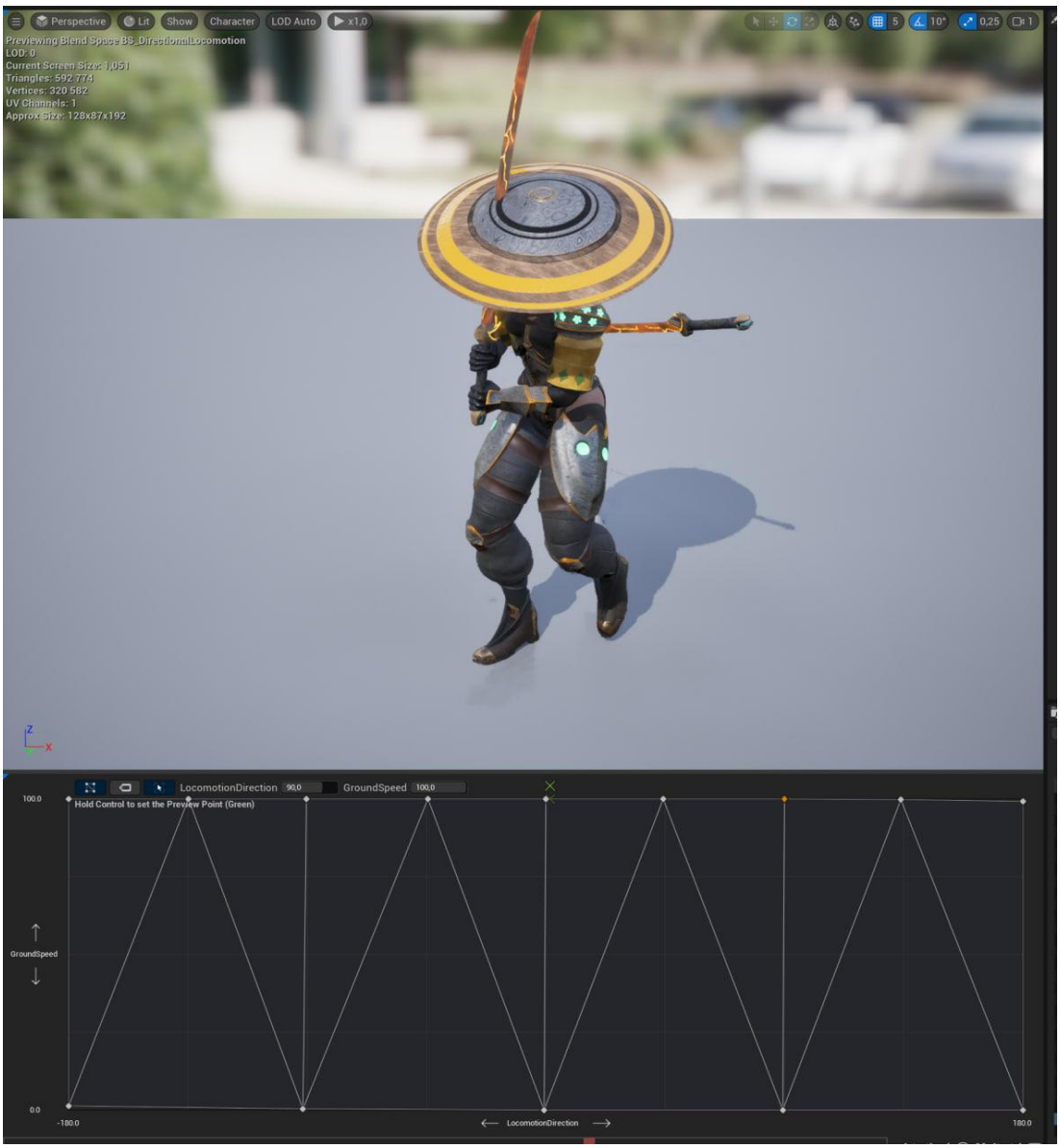

Рисунок 3.3 – Blend Space анімацій переміщення головного персонажа

3.2 Розробка некерований персонажів з використанням класичного підхіду

3.2.1 Реалізація ворогів із ближньою атакою

У процесі створення бойової системи наступним важливим етапом стало впровадження штучного інтелекту для некерованих персонажів, які виконують роль супротивників у грі. Метою цього етапу було не лише забезпечити базову функціональність руху і атаки, а і створити ілюзію

розумної, адаптивної поведінки, яка дозволяє ворогам ефективно реагувати на дії гравця, ухилятися від зіткнень і створювати динамічне бойове середовище.

В рамках реалізації ворогів ближнього бою було створено спеціалізований клас AESEnemyCharacter, який наслідує базову архітектуру AESBaseCharacter, що гарантує сумісність із системою здібностей та інтерфейсом користувача. Цей підхід дозволяє повторно використовувати вже реалізовану логіку, включно з AbilitySystemComponent та AttributeSet, що особливо зручно з огляду на масштабованість гри. Внутрішньо ворог також містить окремі компоненти, такі як EnemyCombatComponent, який відповідає за бойову логіку, та EnemyUIComponent, який керує візуальним відображенням стану ворога. Завдяки WidgetComponent, вбудованому до скелетної сітки, над ворогом динамічно відображається віджет зі здоров’ям.

На рівні логіки управління ворогами використовується власний контролер ESAIController, який інтегрується з навігаційною системою Unreal Engine. Особливістю цього контролера є використання алгоритму DetourCrowdAvoidance, який дозволяє агентам уникати зіткнень одне з одним. Це важливо, оскільки в умовах бою кілька ворогів можуть атакувати гравця одночасно, і їх правильна координація в просторі суттєво підвищує реалізм. Агент, що використовує Crowd Avoidance, не просто прямує до цілі, він здатен обирати обхідні шляхи, динамічно перебудовувати траєкторію руху залежно від навколишніх перешкод і союзників.

Контролер взаємодіє з Behavior Tree, інструментом побудови поведінкової логіки. Через Blackboard зберігається інформація про поточну ціль ворога TargetActor, яка використовується як основна змінна при прийнятті рішень у поведінковому дереві. Blackboard служить місцем зберігання контексту, тобто все, що бачить або знає NPC, зберігається саме у Blackboard.

Виявлення гравця відбувається за допомогою компонента AIPerceptionComponent [42], в якому присутній сенсор

AISense_Sight (рис. 3.4). Це дозволяє ворогам бачити гравця в межах заданого радіусу та поля зору. Як тільки гравець потрапляє до поля зору, інформація про нього заноситься до Blackboard, що й запускає основний цикл прийняття рішень.

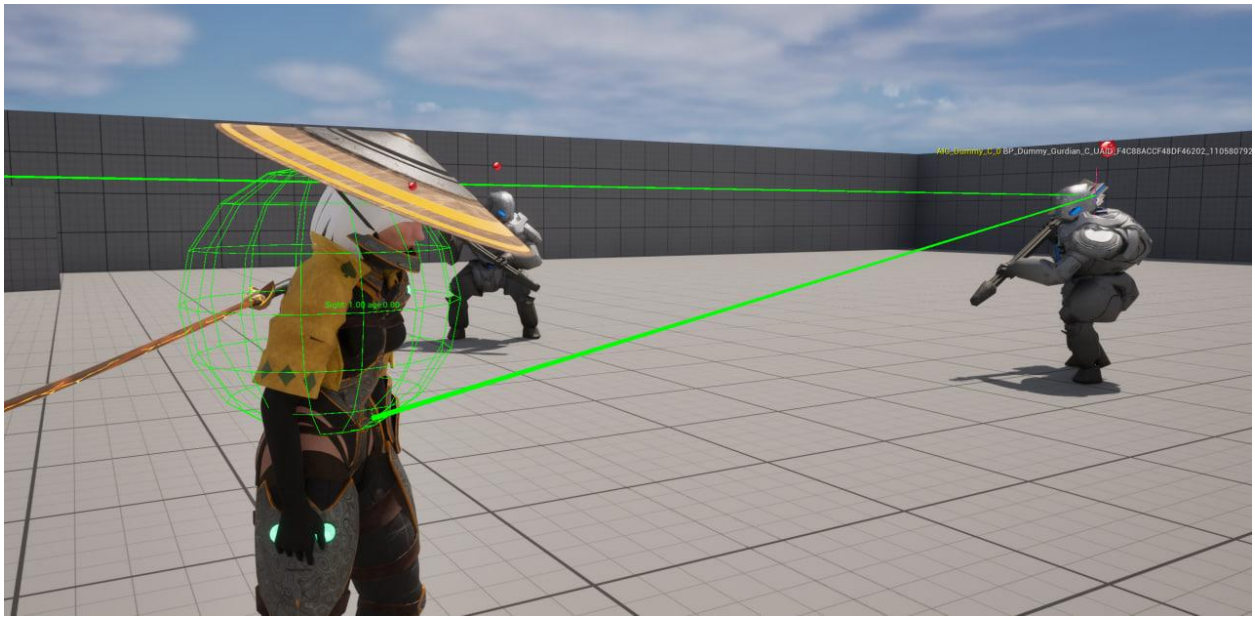

Рисунок 3.4 – Візуалізація «зору» некерованого персонажа

Blackboard служить «пам’яттю» ворога. Основним ключем є TargetActor, тобто ціль яку потрібно переслідувати або атакувати, в залежності від умов. На основі даних у BlackBoard Behavior Tree приймає рішення щодо переміщення та бойових дій.

Behavior Tree (рис. 3.5) складається з кількох основних блоків, які організовують поведінку ворога залежно від ситуації на полі бою. Головний вузол Selector, який по черзі перевіряє кілька варіантів дій, поки одна з них не буде успішною. Першою дією йде захист від атаки. Якщо ворог перебуває під атакою, ворог орієнтується на ворога, наближається, активує здібність захисту і чекає деякий час. Якщо потрібно скасувати дію, ворог просто зупиняє всю свою поведінку на час.

Якщо ворог знаходиться в межах атаки ближнього бою тобто дистанція до цілі менше або дорівнює 150 одиниць, і не триває відкат, некерований

персонаж повертається обличчям до гравця, активує здібність ближньої атаки, чекає деякий час для реалізму анімації, повторно орієнтується на ціль.

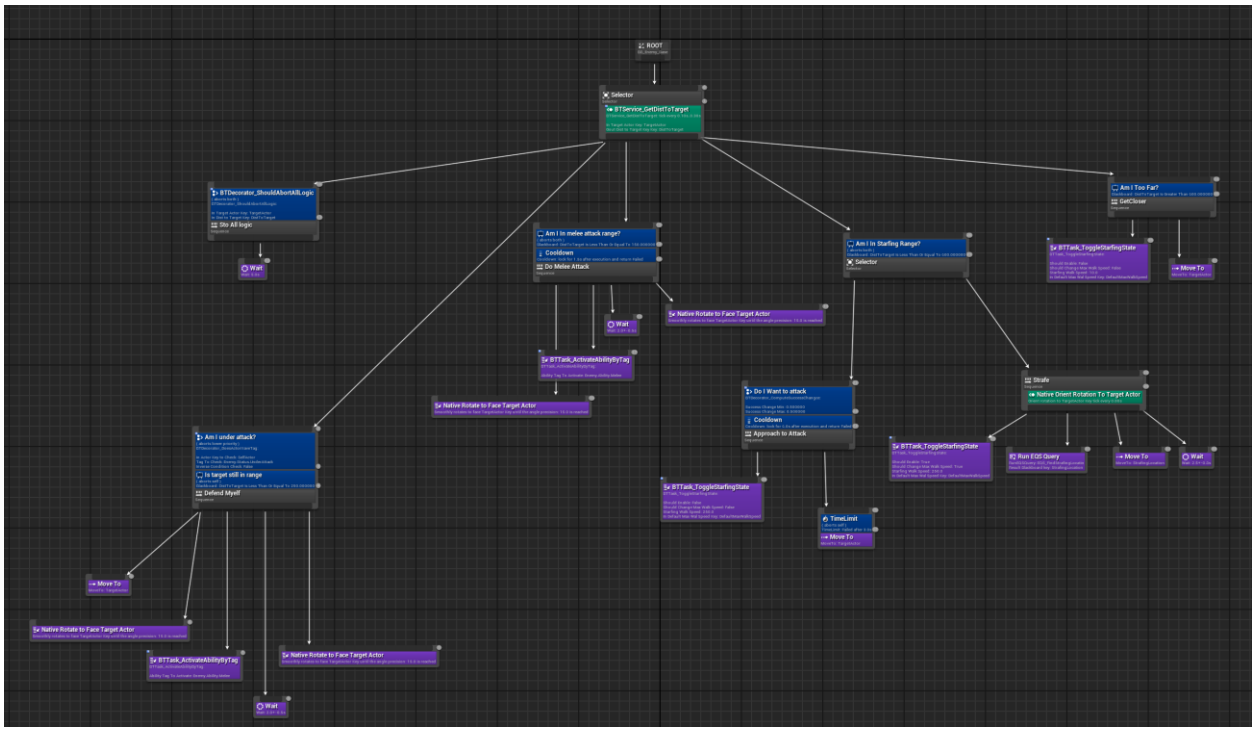

Рисунок 3.5 – Дерево поведінки некерованого персонажа ближнього бою

Якщо ж гравець перебуває на середній дистанції (до 600 одиниць), ворог обирає тактичну поведінку. За допомогою запиту до Environment Query System ворог визначає найбільш вигідну позицію навколо гравця (рис. 3.6). EQS дозволяє динамічно знаходити точки у просторі, що задовольняють таким умовам, як відстань до гравця, наявність укриття, відсутність перешкод. Після отримання результатів запиту NPC переміщується до обраної точки і очікує на нову можливість для атаки.

Детальні механіки:

– BTService_GetDistToTarget постійно оновлює відстань до цілі для прийняття правильних рішень;

– EQS_FindStrafingLocation робить запит у EQS визначає найкращу точку навколо гравця з радіусом близько 1000 одиниць, вибираючи найбільш вигідну позицію;

– Cooldown Decorators запобігають занадто частим атакам та діям, створюючи затримки між спробами.

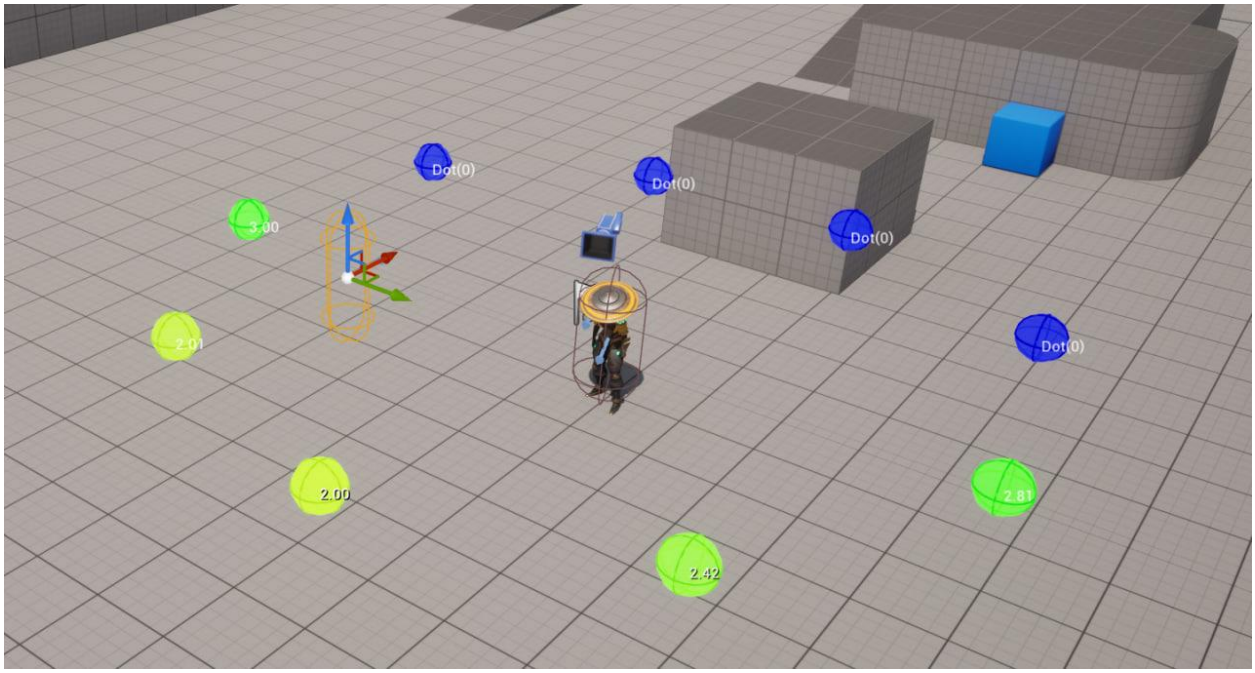

Рисунок 3.6 – Пошук оптимальних позицій через Environment Query System

Ралізація ближнього ворога у грі поєднує в собі одразу декілька сучасних підходів до моделювання поведінки NPC: навігацію, сприйняття, тактичне ухилення, адаптивний вибір дій та бойову взаємодію. Усі ці елементи взаємодіють у межах єдиного архітектурного підходу, побудованого на принципах модульності, повторного використання компонентів та активного використання інструментів Unreal Engine 5.

3.2.2 Реалізація ворогів із атакою на дистанції

Під час реалізації таких ворогів було створено спеціалізоване дерево поведінки (рис. 3.7), яке поєднує в собі декілька сценаріїв реагування на зміну дистанції до гравця, а також алгоритми тактичного переміщення та бойової взаємодії. На верхньому рівні структури Behavior Tree розміщується

вузол Selector, який аналізує ситуацію і послідовно перевіряє всі можливі варіанти поведінки, обираючи той, який наразі є найбільш доречним. Це дозволяє забезпечити гнучкість у поведінці NPC, а також адаптацію до швидко змінюваних обставин на полі бою.

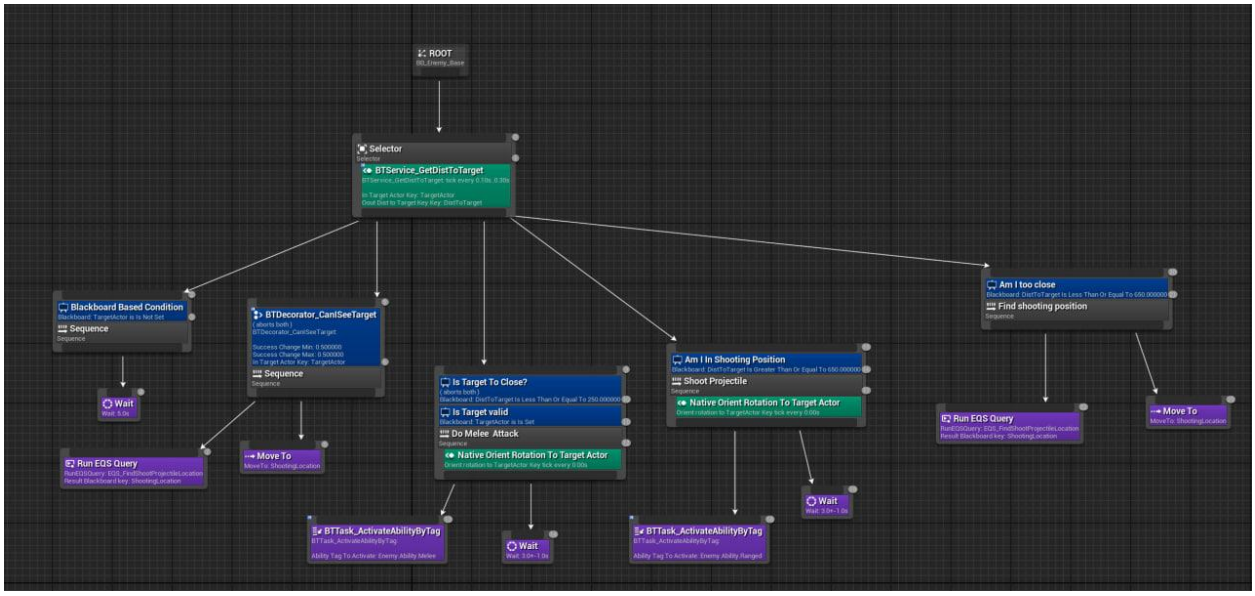

Рисунок 3.7 – Дерево поведінки некерованого персонажа атаки на відстані

Перший блок дерева аналізує наявність видимої цілі. Якщо такої немає, ворог не робить зайвих дій, а переходить у режим очікування. Це дозволяє уникнути надмірної активності NPC у ситуаціях, коли дії були б недоречними або нелогічними. У випадку, коли гравець потрапляє до зони сприйняття, ворог створює запит до Environment Query System (рис. 3.8) для визначення оптимальної позиції для стрільби. Це рішення дає змогу динамічно реагувати на розташування гравця, змінювати бойову тактику та уникати шаблонної поведінки.

Якщо дистанція до гравця перевищує 650 одиниць, некерований персонаж розпізнає, що він перебуває в зоні максимальної ефективності дальнього бою. У цьому випадку NPC фіксує ціль, орієнтується на неї, активує здібність стрільби через систему Gameplay Ability System, після чого запускається пауза, що імітує процес перезарядки або тактичної підготовки

до наступної дії. Такий підхід дозволяє уникнути «спам-атак» і створює відчуття продуманої бойової поведінки, наближеної до людської.

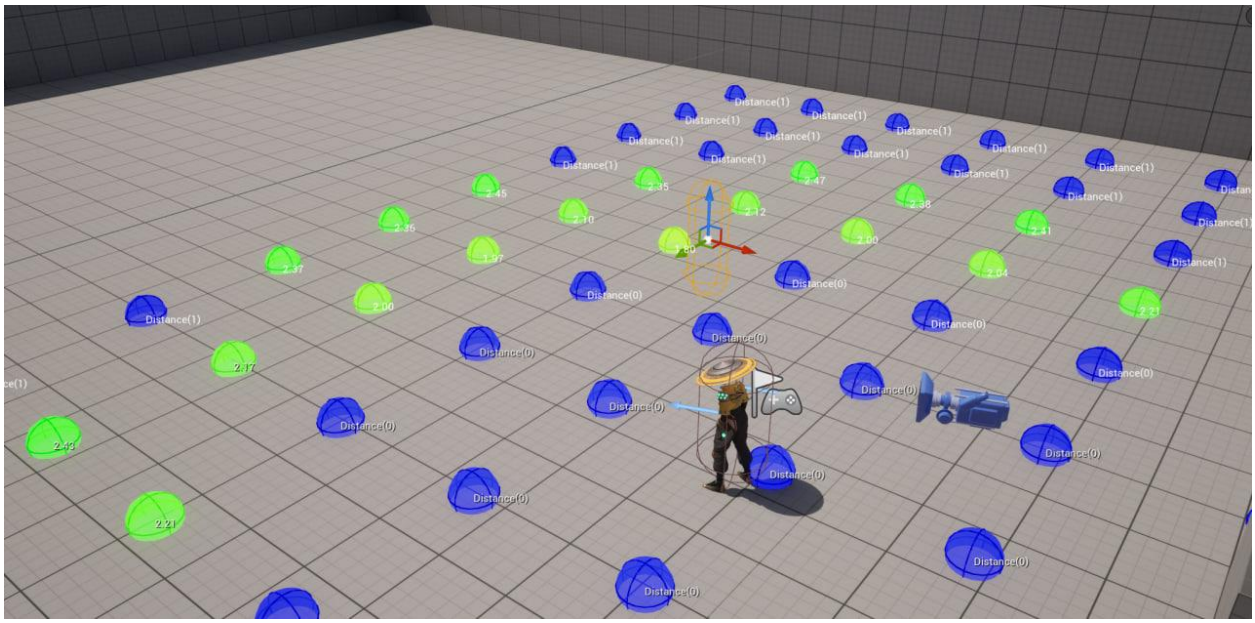

Рисунок 3.8 – Пошук оптимальних позицій через Environment Query System для атак на відстані

Особливу увагу було приділено випадкам середньої дистанції (від 250 до 650 одиниць). У цій зоні NPC не вступає в безпосередній бій, а прагне відновити вигідну позицію. Для цього знову використовується запит до EQS, який обирає найбільш стратегічну точку, з якої можна зберігати дистанцію і при цьому бути готовим до атаки. Завдяки використанню EQS_FindRangedCombatPosition, ворог обирає позицію, яка забезпечує точну оцінку, мінімальні перешкоди та можливість швидкого переміщення в разі небезпеки.

При наближенні гравця, тобто при дистанції менше ніж 250 одиниць, ворог змінює поведінку і застосовує ближню атаку, після чого знову робить паузу. Завдяки цій системі вороги з дистанційною атакою у грі поводяться не шаблонно, а варіативно та більш реалістично, що суттєво підвищує загальну якість бойового геймплею.

3.3 Програмна реалізація навчальних агентів

У межах реалізації системи штучного інтелекту для некерованих персонажів було впроваджено окрему категорію NPC, яка відрізняється принципово новим підходом до поведінки у грі. На відміну від класичних рішень, що базуються на Behavior Tree або жорстко запрограмованих правилах, ці персонажі функціонують на основі навчання з підкріпленням або reinforcement learning, яке забезпечує динамічну та адаптивну поведінку в режимі реального часу. Для реалізації такого підходу було використано плагін Learning Agents доступний у середовищі Unreal Engine 5, що надає набір інструментів для інтеграції машинного навчання без необхідності виходу за межі рушія.

Головною метою було створити інтелектуальних агентів, здатних самостійно формувати поведінкові стратегії на основі взаємодії з оточенням. Ці агенти не мають заздалегідь визначених сценаріїв дій або фіксованої логіки. Їхня поведінка формується в результаті багаторазових ітерацій, під час яких вони спостерігають, експериментують, отримують винагороди або покарання та адаптують свої дії відповідно до отриманого досвіду. Такий підхід дозволяє досягти більш реалістичної та варіативної моделі NPC, що здатна підлаштовуватись під змінні умови гри.

Для дослідження поведінки агентів і формування якісного процесу навчання було реалізовано окреме тренувальне середовище (рис. 3.9). На сцені одночасно присутні 8 NPC з можливістю дальньої атаки, 8 об’єктів-мішеней, які виступають додатковими цілями, головний ігровий персонаж, керований користувачем, який також може бути атакований NPC.

Система винагород є центральним елементом навчання з підкріпленням. Кожен NPC отримує позитивне підкріплення у вигляді +10 балів за успішне знищення цілі (об’єкта або гравця) та негативне підкріплення у вигляді -2.5 балів у разі, якщо був знищений сам гравцем. Такий механізм заохочення стимулює агентів до досягнення цілей, водночас

створюючи ризики, які змушують їх враховувати власну безпеку. Баланс між атакою та обережністю стає головним критерієм ефективної поведінки. Агенти поступово вчаться знаходити оптимальні рішення коли атакувати, коли відступати, як уникати вогню, і які позиції є найбільш вигідними для здійснення атаки.

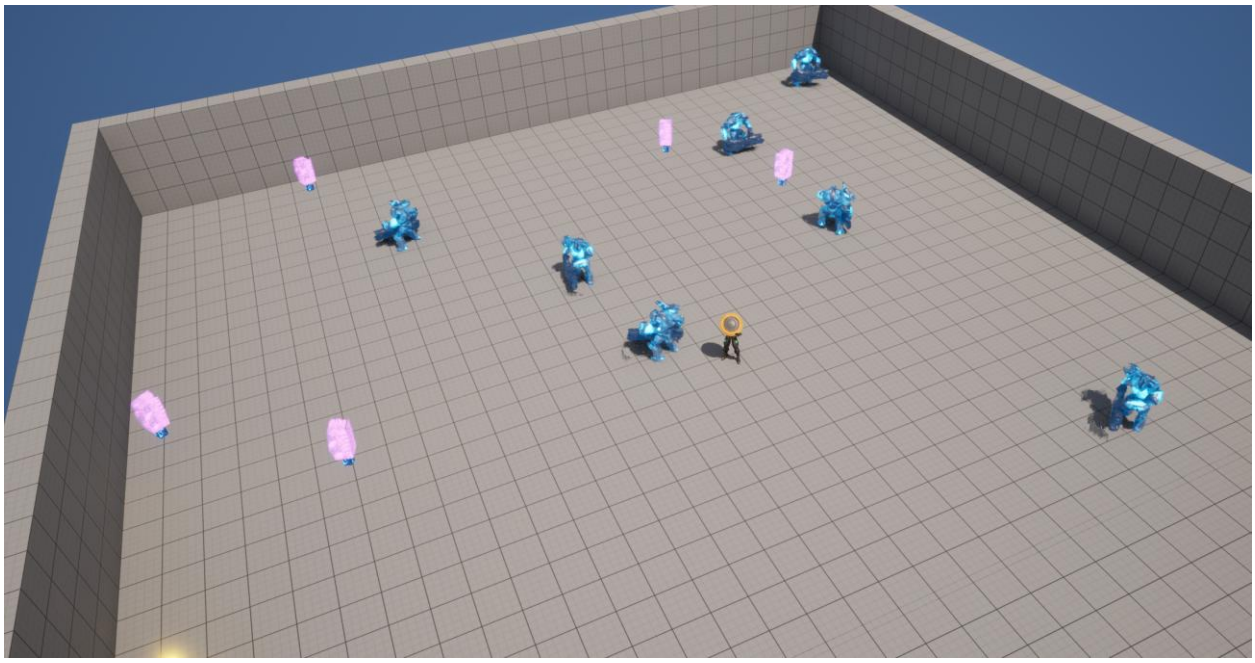

Рисунок 3.9 – Навчальне середовище агентів

Особливістю реалізації є вдосконалена бойова система. Усі NPC озброєні засобами для дальнього бою, які дозволяють запускати снаряди по визначених цілях. Ці снаряди здатні завдати шкоди як мішеням, так і гравцеві, якщо траєкторія польоту була правильно обрана. Водночас самі мішені представляють собою прості об’єкти без штучного інтелекту, куби з базовими візуальними ефектами, які зникають після ураження. Це дозволяє сфокусувати тренування агентів виключно на бойовій поведінці, не відволікаючись на складну симуляцію оточення.

Ще одним елементом, що заслуговує на увагу, є модернізація головного персонажа. Йому була додана можливість атаки на відстані, що також стало новим викликом для Learning Agents. Завдяки використанню Animation Layers, які були реалізовані раніше, додавання нових анімацій

відбувалося без порушення існуючої логіки. Зокрема, шари відповідали за такі фази як підготовка до пострілу, виконання атаки, а також плавні переходи між станами. Це забезпечило органічне розширення бойового функціоналу, без необхідності перебудови анімаційної системи.

Кожен NPC місить компонент LearningAgentComponent, що відповідає за обробку сенсорних даних, генерацію дій, збереження отриманого досвіду та взаємодію з оточенням. Вбудовані механізми плагіна також надають Experience Buffer, структуру, що зберігає дані про попередні епізоди та використовується для вдосконалення моделі поведінки. Цей буфер накопичує вдалі і невдалі приклади, на основі яких відбувається оновлення нейронної мережі, що приймає рішення у майбутніх ітераціях.

Для оцінки ефективності тренування було побудовано графік залежності Average Reward Sum від Iteration (рис. 3.10). Він демонструє зміну середньої суми винагород, яку отримували NPC на кожній з 3840 ітерацій.

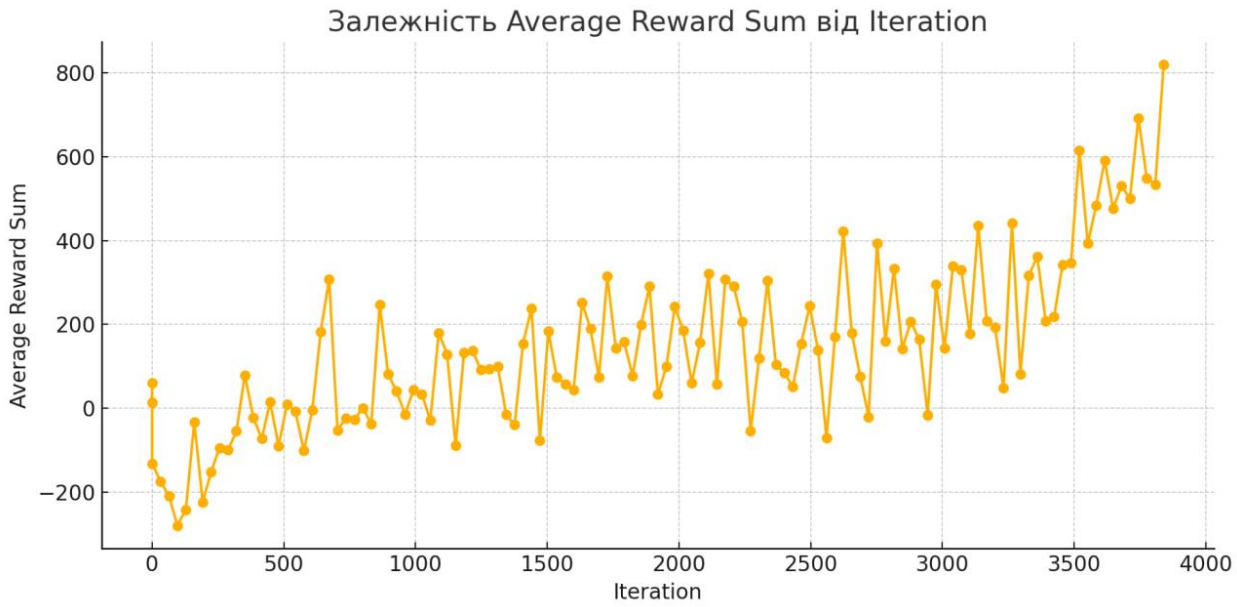

Рисунок 3.10 – Графік залежності середнього значення суми нагороди від ітерації

На початку тренування (ітерації 0–500) спостерігається хаотична поведінка NPC, агенти часто стріляють у випадкові напрямки, не влучають

по цілях і не отримують винагород. Це природно для фази «дослідження», коли нейромережа ще не має сформованої політики.

У діапазоні від 500 до 2500 ітерацій починається етап поступового навчання, агенти все частіше влучають у цілі, вивчають ефективні кути атаки, краще орієнтуються у просторі. Незважаючи на коливання у графіці, видно загальний тренд на зростання ефективності.

Таким чином, у процесі навчання агенти прагнуть максимізувати свою нагороду, активно шукаючи об’єкти для знищення та уникаючи ситуацій, де їх може атакувати головний персонаж. Ця система сприяє виробленню стратегії, за якої вороги намагаються балансувати між агресивними діями і безпекою, оптимізуючи свої переміщення та атаки.

У процесі навчання агенти прагнуть максимізувати свою нагороду, активно шукають об’єкти для знищення та уникають ситуацій, де їх може атакувати головний персонаж. Ця система сприяє виробленню стратегії, за якої вороги намагаються балансувати між агресивними діями і безпекою, оптимізуючи свої переміщення та атаки.

Після 3000-ї ітерації спостерігається різке покращення. NPC демонструють високу точність пострілів і також починають активно використовувати здатність переміщуватися, що дозволяє їм уникати загроз і займати вигідні позиції.

3.4 Тестування розробленої гри

Після завершення реалізації усіх ключових систем гри було проведено комплексне тестування, метою якого стало виявлення помилок, перевірка відповідності функціоналу очікуванням та оцінка стабільності роботи інтегрованих компонентів. Тестування охоплювало основні механіки, пов’язані з головним персонажем, супротивниками, бойовими системами,

інтерфейсом та інтерактивною взаємодією, що узгоджується з підходами до оцінки якості програмного забезпечення, описаними у [43].

У першу чергу було протестовано поведінку головного ігрового персонажа. Анімаційна система, яка реалізована на основі State Machine та Blend Space, забезпечила плавні переходи між станами без візуальних збоїв. Особливу увагу приділено тестуванню анімацій отримання шкоди. Відповідно до напрямку атаки, система запускала різні варіанти анімацій як показано на рисунку 3.11. Це реалізовано шляхом обчислення напрямку влучання відносно вектора вперед персонажа, що дозволило покращити реалізм боїв.

Рисунок 3.11 – Запуск анімації отримання шкоди головним персонажем

Важливим компонентом у системі бою є зменшення рівня здоров’я при отриманні ушкоджень. У ході тестування підтверджено коректну роботу системи атрибутів через Gameplay Ability System. Значення здоров’я головного персонажа зменшується після кожного влучання і ці зміни миттєво передаються у відповідні UI-елементи, що візуалізують стан персонажа на

екрані (рис. 3.12). Це дозволило забезпечити швидкий зворотний зв’язок для гравця, зберігаючи при цьому цілісність інтерфейсу.

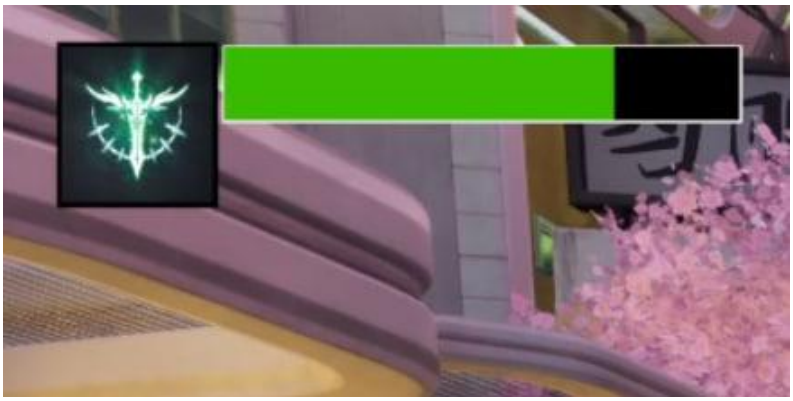

Рисунок 3.12 – Графічний інтерфейс з відображенням поточної зброї і рівня здоров’я

Також було перевірено систему відображення шкоди, що завдається супротивникам. При влучанні гравця по NPC активується спеціальний Gameplay Effect, який запускає візуальний ефект, тимчасову зміну матеріалу, що сигналізує про отримання шкоди (рис. 3.13).

Рисунок 3.13 – Ефект отримання шкоди для некерованого персонажа

У супротивників, як і в головного героя, реалізована система UI-компонента, який відображає шкалу здоров’я (рис. 3.14). Ця шкала оновлюється у режимі реального часу й дозволяє гравцеві орієнтуватися у бойовій ситуації.

Рисунок 3.14 – Компонент з відображенням здоров’я NPC

Окремо тестувалась поведінка ворогів ближнього бою. Було підтверджено, що вони правильно виявляють гравця за допомогою системи зору, а також коректно переміщуються по навігаційній сітці. Завдяки використанню алгоритму Detour Crowd Avoidance NPC не блокують один одного, а динамічно змінюють маршрут, оточуючи гравця (рис. 3.15). При наближенні до цілі супротивники зупиняються на бойовій дистанції та активують ближню атаку через систему здібностей. Анімації атак відтворюються з урахуванням затримки для більшої достовірності, після чого виконується перевірка на влучання та накладається шкода.

Наступним етапом було тестування NPC з атаками на відстані. Ці вороги демонструють складнішу поведінку, за допомогою EQS вони знаходять вигідні позиції для стрільби, тримають рекомендовану дистанцію від гравця, уникають небезпеки зближення (рис. 3.16).

Рисунок 3.15 – Поведінка некерованих персонажів ближньої атаки

Рисунок 3.16 – Дистанційна атака некерованого персонажа

При критичному зближенні використовують ближню атаку (рис. 3.17). Було підтверджено, що візуальні ефекти стрільби активуються синхронно з виконанням здібності.

Рисунок 3.17 – Ближня атака некерованого персонажа

Усі перелічені механіки протестовано в умовах підвищеного навантаження, одночасно на сцені перебувало до 15 NPC, а головний персонаж активно переміщувався, виконував атаки, ухилення та взаємодіяв з UI. Усі компоненти працювали без видимих збоїв або різких змін продуктивності, що підтверджує стабільність реалізованої архітектури та готовність проєкту до подальшої інтеграції контенту.

Особливої уваги під час тестування потребували навчальні агенти, які діють на основі методів підкріпленого навчання. На ранніх етапах симуляції NPC не мали жодного досвіду взаємодії з середовищем, тому демонстрували хаотичну, неефективну поведінку. Це типово для початкової фази навчання,

коли агент виконує випадкові дії з метою зібрати інформацію про наслідки різних сценаріїв. Як видно на рисунку 3.18, агенти хаотично пересуваються, часто стоять на місці або стріляють у випадковому напрямку, не маючи чіткої стратегії ураження цілей.

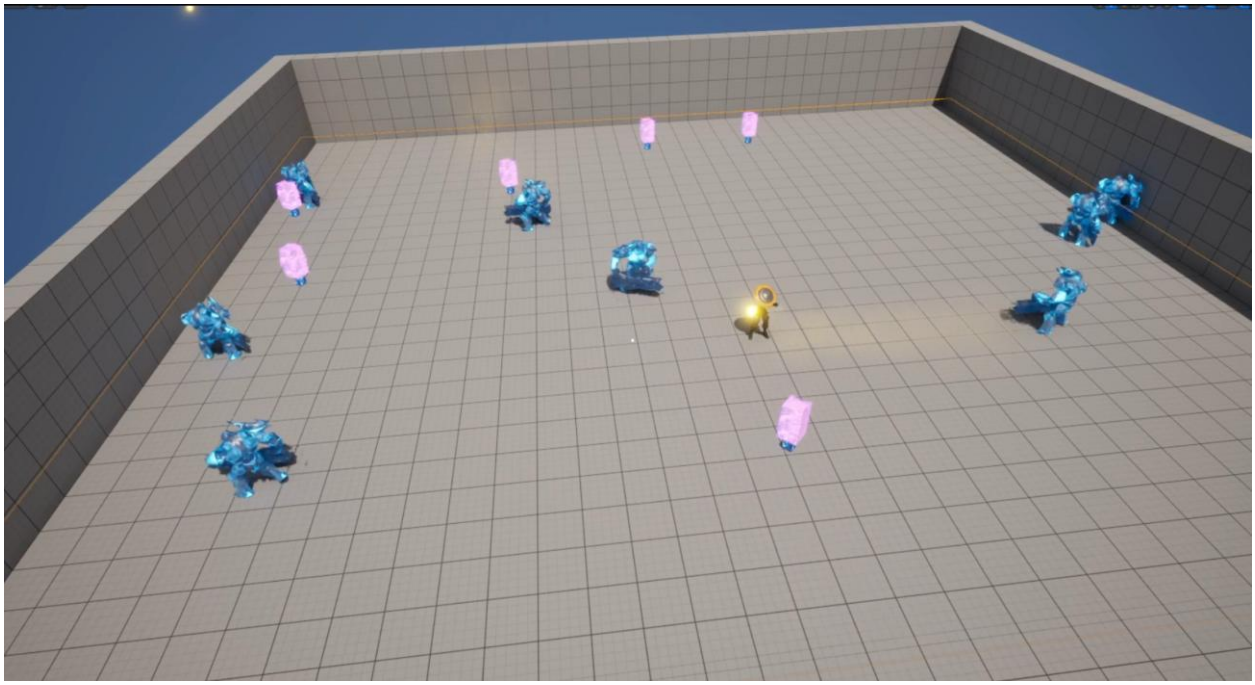

Рисунок 3.18 – Початкові ітерації навчання агентів

Зі збільшенням кількості епізодів NPC поступово накопичували досвід. У діапазоні після 3000-ї ітерації поведінка агентів почала демонструвати явні ознаки впорядкованості. Вони стали точно орієнтуватися в просторі, розпізнавати пріоритетні цілі та виконувати атаки. На рисунку 3.19 спостерігається чітку групу NPC, що стріляють скоординовано і займають стратегічні позиції, розміщуються між гравцем та мішенями. Видно, що кілька агентів об’єднані у «ланцюжки», що дозволяє їм утримувати перевагу у просторі та здійснювати флангові атаки.

Окремо слід зазначити нову поведінкову особливість, яка з’явилась у тренованих агентів, тобто динамічне переміщення під час бою. Якщо на початку NPC стояли на місці та стріляли без урахування ситуації, то після 3000 ітерацій вони почали активно переміщуватися між точками. Це свідчить

про навчання ухиленню та адаптації до нових загроз. Таке переміщення забезпечує виживання агентів і створює складнішу ситуацію для гравця, адже динамічні NPC важче піддаються атаці. Результати підтверджують ефективність впровадженого підходу Learning Agents, демонструючи значне підвищення адаптивності та складності NPC, що суттєво збагачує геймплей.

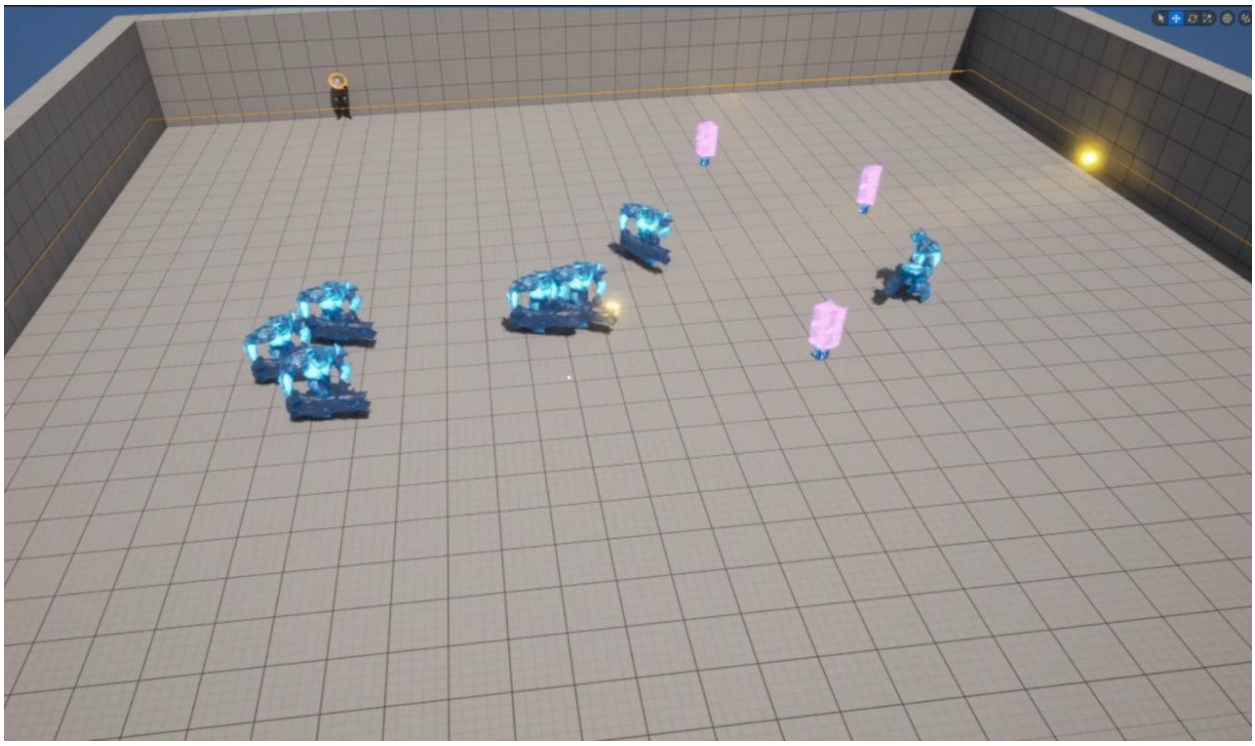

Рисунок 3.19 – Поведінка агентів після 3000-ї ітерації навчання

3.5 Порівняння підходів до розробки штучного інтелекту NPC

У процесі реалізації проєкту було протестовано декілька підходів до створення інтелектуальної поведінки неігрових персонажів. Основну частину системи було реалізовано за допомогою Behavior Tree у поєднанні з Environment Query System. На наступному етапі впроваджено навчальних агентів, які базуються на методах підкріпленого навчання. Обидва підходи мають як спільні риси, так і суттєві відмінності, які впливають як на досвід розробника (табл. 3.1), так і на ігровий досвід користувача (табл. 3.2).

Таблиця 3.1 – Порівняння з точки зору розробника

Критерій	**Behavior Tree + EQS**	**Learning Agents**
Простота реалізації	Наявні шаблони, візуальний редактор	Потребує знань у сфері машинного навчання
Контроль над поведінкою	Повний, але жорстко структурований	Опосередкований, поведінка виникає через навчання
Гнучкість до змін	Обмежена, будь-яка зміна потребує редизайну дерева	Висока, агент сам адаптується до нових умов
Масштабованість	Складна, зі збільшенням логіки дерево ускладнюється	Висока: один агент може працювати в різних умовах
Підтримка та розширення логіки	Потребує ручного втручання	Автоматизована еволюція поведінки під час навчання
Час на реалізацію	Швидкий початковий результат	Більше часу на налаштування і навчання
Повторюваність результату	Передбачуваний, жорстко контрольований результат	Поведінка залежить від контексту, більш варіативна
Використання ресурсів	Мінімальне	Вища обчислювальна складність під час навчання

Хоча впровадження Learning Agents вимагає більше часу, знань та ресурсів, у довгостроковій перспективі воно відкриває набагато ширші можливості. Після етапу навчання агенти можуть бути масштабовані на інші локації, ситуації та рівні складності без необхідності створення нових

сценаріїв вручну. Це дозволяє зекономити час і уникнути помилок, які неминуче виникають у складних Behavior Tree.

Таблиця 3.2 – Порівняння з точки зору гравця

Критерій	**Behavior Tree + EQS**	**Learning Agents**
Рівень виклику для гравця	Середній, шаблонна поведінка NPC	Високий, NPC пристосовуються до дій гравця
Реалістичність поведінки	Обмежена, NPC діють по шаблону	Природна, NPC самі приймають рішення
Здатність до адаптації	Відсутня, не реагують на зміну середовища	Висока, NPC вчаться ухилятись, атакувати з вигідних позицій
Занурення у гру	Середнє, NPC можуть здаватись роботизованими	Глибоке, NPC створюють враження реальних супротивників
Рівень інтелекту NPC	Потребує ручного втручання	Автоматизована еволюція поведінки під час навчання
Враження від гри	Задовільне, але прогнозоване	Динамічне, з ефектом живого середовища

Зі сторони гравця відмінності між підходами відчуваються ще більш виразно. Поведінка NPC, побудована на дереві рішень, хоча й виглядає логічною, досить швидко починає повторюватись. Це знижує рівень занурення та зменшує виклик. Натомість NPC натреновані через підкріплене навчання, здатні змінювати тактики, ухилятись, займати вигідні позиції або навіть уникати конфлікту. Це створює новий тип динамічної взаємодії, який неможливо спрогнозувати на 100% і саме це робить гру цікавішою.

Застосування Learning Agents дозволило досягти такого рівня динаміки поведінки NPC, який не лише перевищує класичні методи, а й піднімає

загальну якість геймплею. У процесі тестування стало очевидним, що гравець відчуває реальний тиск і змушений адаптуватися до нових тактик, замість того, щоб заучувати шаблони дій супротивника. Це особливо важливо для ігор жанру RPG, де взаємодія з супротивником повинна залишатися цікавою протягом десятків годин.

Зі збільшенням кількості агентів у середовищі спостерігається значне прискорення процесу навчання. Це пов'язано з тим, що декілька агентів одночасно генерують більше досвіду, кожен з них взаємодіє з середовищем незалежно, і вся ця інформація може бути використана для оновлення моделі. Таким чином, система отримує багатший та різноманітніший досвід за меншу кількість ітерацій.

На рисунках 3.20 і 3.21 чітко видно цю залежність. Одному агенту потрібно майже 6000 ітерацій, щоб досягти рівня середньої винагороди, який вісім агентів досягають приблизно за 3000 ітерацій. Більше того, при збільшенні кількості агентів до 25, навченість починає проявлятись вже приблизно на 1500-ій ітерації. Це дозволяє значно скоротити час тренування, особливо у великих проєктах із високим навантаженням на систему.

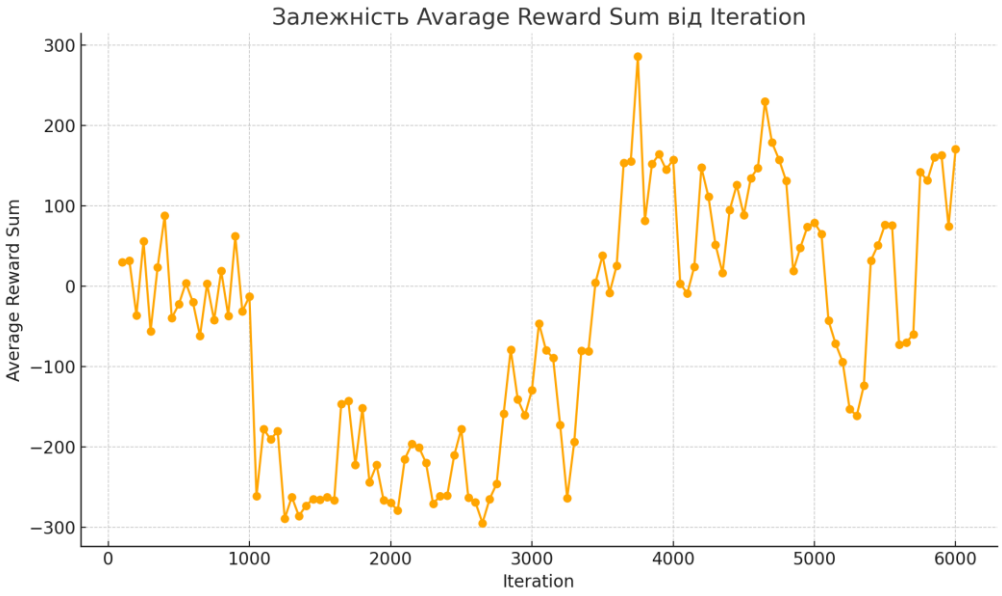

Рисунок 3.20 – Графік залежності середнього значення суми нагороди від ітерації для одного агента

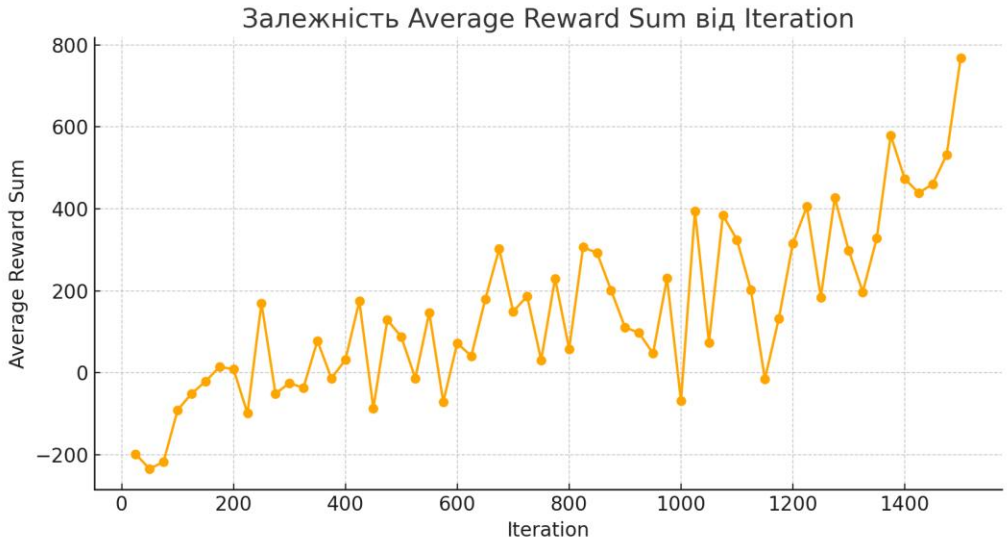

Рисунок 3.21 – Графік залежності середнього значення суми нагороди від ітерації для 25 агентів

Проте така стратегія має і свої обмеження. Збільшення кількості агентів потребує більше ресурсів, зростає навантаження на обчислювальну систему, використання пам’яті та пропускної здатності, особливо якщо всі агенти працюють одночасно у спільному середовищі. Крім того, ефективність не завжди зростає лінійно, після певного порогу збільшення кількості агентів може призводити до конфліктів у середовищі, взаємного блокування або некорисної конкуренції.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований прототип RPG-гри в середовищі Unreal Engine 5 із застосуванням засобів штучного інтелекту для моделювання поведінки некерованих персонажів. Основна увага приділялася створенню гнучкої, адаптивної та масштабованої системи NPC, яка поєднує класичні та сучасні підходи до моделювання поведінк, а саме Behavior Tree та навчання з підкріпленням.

У процесі роботи було здійснено глибокий аналіз еволюції методів моделювання NPC, від скриптових сценаріїв і скінченних автоматів до ієрархічних дерев поведінки та систем адаптивного сприйняття середовища. На основі цього аналізу були виділені основні компоненти, які дозволяють забезпечити природну та динамічну поведінку NPC у складному ігровому середовищі. Було доведено ефективність поєднання Behavior Tree та Blackboard для реалізації логіки високого рівня, а також важливість інтеграції Environmental Query System для ухвалення рішень на основі контексту.

Однією з найважливіших інновацій проєкту стала реалізація навчальних агентів за допомогою плагіну Learning Agents. Було створено окреме тренувальне середовище, в якому NPC самостійно навчалися взаємодіяти з ігровим простором, отримуючи винагороди або штрафи залежно від своїх дій. У результаті проведеного тренування було зафіксовано покращення поведінки агентів, а саме зростання точності атак, ефективне ухилення, переміщення до вигідних позицій, а також здатність до стратегічного розташування під час бою. Це свідчить про доцільність впровадження методів машинного навчання у моделювання ігрових NPC для підвищення глибини геймплею.

Також було протестовано та проаналізовано традиційні підходи реалізації NPC з ближньою та дистанційною атакою. Створені дерева поведінки враховують стан здоров’я, відстань до цілі, наявність перешкод та інші фактори. NPC використовують EQS для пошуку вигідних позицій,

здатні виконувати дії в реальному часі з урахуванням змін у середовищі, і демонструють природну реакцію на дії гравця.

Особливу увагу приділено архітектурі головного персонажа, включаючи інтеграцію системи здібностей, бойових механік, анімацій та UI-компонентів. Це забезпечило цілісність усіх ігрових систем і створило передумови для подальшого розширення функціоналу гри.

Результати роботи опубліковано у вигляді тез доповідей під час Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [44], що підтверджує актуальність і наукову новизну проведених досліджень.

Загалом реалізований проєкт доводить можливість створення складної системи NPC із високим рівнем адаптивності, автономності та реалістичності. Застосування інструментів Unreal Engine 5 у поєднанні з методами навчання з підкріпленням відкриває нові горизонти для інтерактивного дизайну та моделювання поведінки персонажів у відеоіграх. Отримані результати можуть бути використані як основа для подальших наукових досліджень і вдосконалення комерційних проєктів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wikipedia – Non-player character. URL: https://en.wikipedia.org/wiki/Non-player_character (дата звернення: 07.05.2025).

2. YouTube – INSANE REALISM: NPC ROUTINES in Red Dead Redemption 2. URL: https://www.youtube.com/watch?v=eoaP66kp_NA (дата звернення: 09.05.2025).

3. YouTube – AI and Game Design | The History of Artificial Intelligence In Video Games. URL: https://www.youtube.com/watch?v=wh9kpe1Dn8s (дата звернення: 07.05.2025).

4. YouTube – Revisiting the AI of Alien: Isolation | AI and Games #50. URL: https://www.youtube.com/watch?v=P7d5lF6U0eQ (дата звернення: 10.05.2025).

5. YouTube – Why I love Shadow Tactics in 4 minutes. URL: https://www.youtube.com/watch?v=R8zRU7ntOeU (дата звернення: 09.05.2025).

6. Sherif, W. (2015). Learning C++ by Creating Games with Unreal Engine 4: навч. посібник. *Birmingham: Packt Publishing.*

7. Stroustrup, B. (2013). The C++ Programming Language (4th ed.): навч. посібник. *Boston: Addison-Wesley.*

8. Meyers, S. (2005). Effective C++: 55 Specific Ways to Improve Your Programs and Designs: навч. посібник. *Boston: Addison-Wesley.*

9. Unreal Engine Documentation – Behavior Trees. URL: https://docs.unrealengine.com/5.0/en-US/behavior-trees-in-unreal-engine/ (дата звернення: 11.05.2025).

10. Doran, J.P. (2023). Unreal Engine 5 AI Programming with Behavior Trees and EQS: навч. посібник. *Birmingham: Packt Publishing.*

11. Unreal Engine Documentation – EQS. URL: https://docs.unrealengine.com/5.0/en-US/environment-query-system/ (дата звернення: 11.05.2025).

12. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. Cambridge, MA: MIT Press.

13. Chollet, F. (2021). Deep Learning with Python (2nd ed.). Shelter Island, NY: Manning Publications.

14. Lapan, M. (2020). Deep Reinforcement Learning Hands-On (2nd ed.). Birmingham: Packt Publishing.

15. Silver, D. (2015). Reinforcement Learning Course. University College London. URL: https://davidstarsilver.wordpress.com/teaching/ (дата звернення: 01.05.2025).

16. OpenAI. Baselines for Reinforcement Learning. URL: https://github.com/openai/baselines (дата звернення: 12.05.2025).

17. Microsoft Research. Project Malmo: Reinforcement Learning in Minecraft. URL: https://www.microsoft.com/en-us/research/project/project-malmo/ (дата звернення: 17.05.2025).

18. Sutton, R.S., & Barto, A.G. (2018). Reinforcement Learning: An Introduction (2nd ed.). Cambridge, MA: MIT Press.

19. КОБИЛІН, О., ВЕЧІРСЬКА, І., & АФАНАСЬЄВ, А. (2024). Аналіз існуючих моделей глибинного навчання в задачах обробки природної мови. Information Technology: Computer Science, Software Engineering and Cyber Security, (3), 63-76.

20. Tvoroshenko, I., Gorokhovatskyi, V., Kobylin, O., & Tvoroshenko, A. (2023). Application of deep learning methods for recognizing and classifying culinary dishes in images.

21. Tvoroshenko, I., Pomazan, V., Gorokhovatskyi, V., & Kobylin, O. (2023). Application of video data classification models using convolutional neural networks.

22. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Handwritten character recognition models based on convolutional neural networks.

23. Gorokhovatskyi, V., Tvoroshenko, I., Yakovleva, O., Hudáková, M., & Gorokhovatskyi, O. (2024). Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set. IEEE Access.

24. Vechirska, I., Kobylin, O., Prokopiev, S., Vechirska, A., & Kucherenko, M. (2022). Building a logical network for solving the problem of car rental by means algebra of finite predicates. Computer systems and information technologies, (2), 78-87

25. КОБИЛІН, О., ВЕЧІРСЬКА, І., & КРАВЧЕНКО, О. (2024). Порівняння нейронних мереж типу RNN та LSTM. Information Technology: Computer Science, Software Engineering and Cyber Security, (3), 97-107.

26. Tvoroshenko, I. S., & Gorokhovatskyi, V. O. (2019). Modification of the branch and bound method to determine the extremes of membership functions in fuzzy intelligent systems. Telecommunications and Radio Engineering, 78(20).

27. Dubnitskiy, V., Kobylin, A., Kobylin, O., & Kushneruk, Y. (2021). EXCEL–орієнтована процедура для обчислення значень спеціальних функцій з інтервальним аргументом, заданим в гіперболічній формі. Advanced Information Systems, 5(4), 116-123.

28. Unreal Engine Documentation – Learning Agents Plugin. URL: https://docs.unrealengine.com/5.3/en-US/learning-agents-in-unreal-engine/ (дата звернення: 01.05.2025).

29. OpenAI. *Spinning Up in Deep RL: PPO, Actor-Critic Algorithms*. URL: https://spinningup.openai.com/en/latest/ (дата звернення: 01.05.2025).

30. Karakonstantyn, D., & Tvoroshenko, I. (2024). About the issue of optimization the performance of the server part of the information system.

31. Unreal Engine Documentation – NavMesh Navigation. URL: https://docs.unrealengine.com/5.0/en-US/navmesh-in-unreal-engine/ (дата звернення: 09.05.2025).

32. Unreal Engine Documentation – AIController. URL: https://docs.unrealengine.com/5.0/en-US/aicontroller-in-unreal-engine/ (дата звернення: 16.05.2025).

33. Yannakakis, G.N., & Togelius, J. (2018). *Artificial Intelligence and Games*: монографія. Springer.

34. Unreal Engine Documentation – Using Avoidance with the Navigation System in Unreal Engine. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-avoidance-with-the-navigation-system-in-unreal-engine (дата звернення: 14.05.2025).

35. Unreal Engine Documentation – Gameplay Ability System (GAS). URL: https://docs.unrealengine.com/5.0/en-US/gameplay-ability-system-overview/ (дата звернення: 11.05.2025).

36. Schell, J. (2019). The Art of Game Design: A Book of Lenses (3rd ed.): навч. посібник. *Boca Raton: CRC Press.*

37. Unreal Engine Documentation – Motion Warping in Unreal Engine. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/motion-warping-in-unreal-engine (дата звернення: 11.05.2025).

38. Unreal Engine Documentation – UMG UI Designer. URL: https://docs.unrealengine.com/5.0/en-US/unreal-motion-graphics-ui-designer/ (дата звернення: 12.05.2025).

39. Unreal Engine Documentation – Enhanced Input in Unreal Engine. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/enhanced-input-in-unreal-engine (дата звернення: 11.05.2025).

40. Unreal Engine Documentation – Animation Blueprints. URL: https://docs.unrealengine.com/5.0/en-US/animation-blueprints-in-unreal-engine/ (дата звернення: 09.05.2025).

41. Unreal Engine Documentation – Blend Spaces in Unreal Engine. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/blend-spaces-in-unreal-engine (дата звернення: 11.05.2025).

42. Unreal Engine Documentation – AI Perception System. URL: https://docs.unrealengine.com/5.0/en-US/ai-perception-overview-in-unreal-engine/ (дата звернення: 12.05.2025).

43. Tvoroshenko, I. S., & Kuznetsov, M. (2021). About the role of testing in process of mobile application development.

44. Літвінова О.О. (2025) Використання штучного інтелекту для моделювання поведінки NPC у середовищі Unreal Engine 5. *Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–19 квітня 2025 р.)*. Харків: ХНУРЕ, 2025. Т. 7. С. 81-82.