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 ТАКТИЧНОЇ ГРИ ІЗ ЗАСТОСУВАННЯМ ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ АНАЛІЗУ ІГРОВОЇ СЦЕНИ ТА ПРИЙНЯТТЯ РІШЕННЯ ПОВЕДІНКИ ПЕРСОНАЖІВ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-3
Куртенок М. Д.
(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 Куртенку Микиті Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка тактичної гри із застосуванням засобів штучного інтелекту для аналізу ігрової сцени та прийняття рішення поведінки персонажів

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 2 червня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, документація рушія Unity.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд штучного інтелекту.
2. Огляд існуючих ігрових рушіїв.
3. Основи інтеграції нейронної мережі.
4. Розробка тактичних сценаріїв.
5. Реалізація управління гравцем та супротивником.
6. Підключення нейронної мережі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Проблеми ШІ в іграх, вибір нейромережі для проекту, підключення нейромережі, відправка запиту, ситуація з надсиланням пакету даних, переваги Unity.

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	08.04.25-11.04.25	
3	Аналіз літератури з досліджуваної проблеми	12.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-19.04.25	
5	Розробка методу	20.04.25-28.04.25	
6	Програмна реалізація	29.04.25-12.05.25	
7	Оформлення пояснювальної записки	13.05.25-20.05.25	
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	
13	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: 63 с., 2 табл., 26 рис., 31 джерело.

ШТУЧНИЙ ІНТЕЛЕКТ, НЕЙРОННА МЕРЕЖА, АНАЛІЗ ІГРОВОЇ СЦЕНИ, ТАКТИЧНА ГРА, ПОВЕДІНКА ПЕРСОНАЖІВ, UNITY.

Об’єктом роботи є вдосконалення штучного інтелекту в іграх.

Метою роботи є розробка тактичної гри з використанням нейронної мережі для аналізу ігрової сцени та ухвалення рішень поведінки персонажів.

У роботі проаналізовано підходи до моделювання поведінки персонажів за допомогою штучного інтелекту, а також розглянуто структуру та особливості нейронних мереж для ігрових застосувань. Реалізовано гру на рушії Unity, де нейронна мережа аналізує такі параметри, як відстань до противника, рівень здоров’я, час до перезарядки зброї та обирає дію: атака, відступ, займання укриття або інше. Проведено тестування поведінки противника у різних ігрових ситуаціях.

Результати експериментів показали ефективність розробленої системи прийняття рішень у реальному часі для тактичних ігор та її перевагу над звичайними алгоритмами поведінки.

ARTIFICIAL INTELLIGENCE, NEURAL NETWORK, GAME SCENE ANALYSIS, TACTICAL GAME, CHARACTER BEHAVIOR, UNITY.

The object of the work is the improvement of artificial intelligence in games.

The aim of the work is the development of a tactical game using a neural network for analyzing the game scene and making character behavior decisions.

The work analyzes approaches to modeling character behavior using artificial intelligence, as well as examines the structure and features of neural networks for gaming applications. A game was developed using the Unity engine, where the neural network analyzes parameters such as distance to the enemy, health level, weapon reload time, and selects an action: attack, retreat, taking cover, or other. The behavior of the opponent was tested in various gameplay situations.

The results of the experiments demonstrated the effectiveness of the developed real–time decision–making system for tactical games and its advantage over conventional behavior algorithms.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 6
Вступ 7
1 Аналіз сучасного підходу до створення ШІ в іграх 8
1.1 Огляд штучного інтелетку 8
1.2 Інтеграція нейронних мереж у ігровий ШІ 9
1.3 Приклади створених ігр з нейромережами 11
1.4 Майбутнє ігор з вдосконаленим ШІ 13
1.5 Огляд існуючих ігрових рушіїв. 15
1.6 Постановка задачі 17
2 Концептуальна модель ігрової поведінки NPC з використанням нейронної мережі 18
2.1 Основи інтеграції нейронної мережі в Unity 18
2.2 Формування запитів для аналізу стану ігрового світу 20
2.3 Вибір нейронної мережі для інтеграції 22
2.4 Проблеми та виклики при інтеграції нейронної мережі в Unity 25
2.5 Управління танком гравцем і NPC 27
2.6 Розробка тактичних сценаріїв для поведінки супротивника 30
3 Програмна реалізація системи адаптивної поведінки супротивника в грі . 34
3.1 Вибір середовища розробки 34
3.2 Архітектура проекту та структура сцени 36
3.3 Реалізація управління гравцем 38
3.4 Реалізація прицілювання та стрільби гравця 40
3.5 Створення логіки супротивника 42
3.6 Взаємодія з ChatGPT 44
3.7 Система здоров’я та анімація індикатора 47
3.8 Пошук укриттів та взаємодія з ними 49
3.9 Формування запиту до нейромережі 51
3.10 Реалізація тактик поведінки супротивника 53
3.11 Тестування 56
Висновки 60
Перелік джерел посилання 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект

NPC – Non–Playable Character (неігровий персонаж)

API – Application Programming Interface (інтерфейс прикладного програмування)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

Unity – програмне середовище для розробки 2D та 3D ігор

NavMesh – навігаційна сітка, що дозволяє AI переміщуватись по сцені з урахуванням перешкод

ChatGPT – велика мовна модель від компанії OpenAI

FPS – Frames Per Second (кількість кадрів за секунду)

HP – Health Points (очки здоров’я персонажа)

Rigidbody2D – компонент фізики у Unity для 2D–об’єктів

Transform – компонент, що визначає положення, обертання і масштаб об’єкта

Cover – укриття, яке можна використати для захисту на полі бою

ВСТУП

Штучний інтелект – одна з найдинамічніших і найперспективніших галузей сучасної інформатики. Його принципи та технології вже давно вийшли за межі суто наукових досліджень і знаходять широке застосування в реальному житті: в медицині, логістиці, економіці, автоматизації виробництва, обробці природної мови тощо. Однією з найбільш цікавих сфер впровадження є індустрія комп'ютерних ігор, яка стрімко розвивається як у технічному, так і в концептуальному напрямках.

Поведінка персонажів у відеоіграх, які не контролюються гравцем, традиційно реалізується за допомогою простих алгоритмів: умовних переходів, фінітних автоматів або дерев поведінки. Ці підходи дозволяють побудувати елементарну логіку реагування, однак не забезпечують гнучкості, варіативності та адаптації до непередбачених ситуацій. У результаті, ігрові супротивники стають передбачуваними, а сама гра – менш захопливою, особливо для досвідченого гравця.

Розвиток мовних моделей на базі нейронних мереж, таких як ChatGPT, відкриває нові можливості для створення адаптивного ігрового ШІ. Такі моделі здатні аналізувати текстову інформацію, виявляти логічні зв'язки, реагувати на зміну контексту та приймати рішення, що ґрунтуються на поточній ситуації. Це дозволяє сформувати новий підхід до побудови штучного інтелекту у відеоіграх – не як заздалегідь запрограмовану схему дій, а як систему, здатну самостійно обирати тактику в реальному часі.

Актуальність даної роботи полягає у впровадженні нейронної мережі в логіку поведінки супротивника у комп'ютерній грі. На відміну від класичних ігрових моделей, де вся поведінка неігрових персонажів (NPC) будується на фіксованих правилах, запропонований підхід дозволяє персонажу аналізувати ситуацію – з урахуванням свого здоров'я, дистанції до противника, наявності укриттів, часу до перезарядки зброї тощо – і на цій основі приймати обґрунтоване рішення.

1 АНАЛІЗ СУЧАСНОГО ПІДХОДУ ДО СТВОРЕННЯ ШІ В ІГРАХ

1.1 Огляд штучного інтелетку

Найбільш поширеними підходами до створення ігрового ШІ стали фінітні автомати (Finite State Machines) та дерева рішень (Behavior Trees). Фінітні автомати дозволяють персонажу перемикатися між наборами дій залежно від певних умов, наприклад: патрулювання, виявлення гравця, переслідування, атака [1-2]. Дерева рішень забезпечують ієрархічну структуру вибору дій, яка допомагає будувати більш складну, але все ж обмежену модель поведінки [3].

Попри свою ефективність у реалізації базових сценаріїв, такі системи мають низку суттєвих недоліків [4]. Основним з них є передбачуваність дій NPC. Досвідчені гравці досить швидко розуміють логіку роботи ШІ: вони знають, на якій відстані супротивник виявить їх, коли почне атаку, як відреагує на втрату здоров’я. Це знижує рівень виклику, зменшує емоційне залучення та робить геймплей менш захопливим.

Іншою проблемою є неспроможність ШІ адекватно реагувати на ситуації, що не були передбачені розробниками. Наприклад, якщо гравець діє нестандартно, NPC часто виявляються безпорадними: застрягають у кутках мапи, не реагують на загрози або виконують безглузді дії.

Ще одним важливим аспектом є обмеженість варіантів взаємодії. Більшість NPC здатні вести лише шаблонні діалоги з обмеженим набором реплік, що істотно знижує глибину ігрового світу та порушує відчуття реалістичності (рис. 1.1).

Таким чином, хоча традиційні підходи до ігрового ШІ й дозволяють створити функціональні ігрові світи, вони не забезпечують належного рівня занурення та варіативності, яких очікують сучасні гравці.

Рисунок 1.1 – Вікно діалогу у грі Fallout 1

1.2 Інтеграція нейронних мереж у ігровий ШІ

Актуальність роботи полягає у тому, що з появою та розвитком потужних нейронних мереж відкрилися нові можливості для революціонізації ігрового ШІ. Технології на кшталт ChatGPT, DeepSeek, Copilot, Gemini та інших мовних моделей довели здатність машин генерувати осмислені, логічні та гнучкі відповіді у реальному часі. Інтеграція таких систем у відеоігри здатна докорінно змінити підхід до створення NPC та ігрових світів.

Одним з найперспективніших напрямків є застосування великих мовних моделей для управління діалогами. Замість фіксованих гілок розмов, гравець зможе вільно ставити будь–які питання, а персонаж відповідатиме, базуючись на власному «досвіді» та знаннях про світ гри. Це дозволить створити унікальні сюжетні лінії та події, які виникатимуть без прямого сценарного втручання.

Інша можливість – використання нейронних мереж для прийняття тактичних рішень. На основі аналізу змінних, таких як рівень здоров’я, боєздатність, розташування укриттів, дистанція до ворога тощо, NPC зможе адаптувати свою поведінку залежно від конкретної ситуації. Це наблизить ігрову поведінку до реальної логіки людини, зробивши ігри значно більш захопливими.

Уявімо собі гру жанру RPG, де кожен мешканець міста має власну особистість, звички та навіть пам’ять про попередні взаємодії з гравцем. З кожним NPC можна буде вести діалог на будь–яку тему, отримувати унікальну інформацію та впливати на їхню поведінку в залежності від власних дій. Це відкриє шлях до створення ігрових світів безпрецедентної глибини та реалістичності. Цю механіку частково реалізували у Fallout 1, але там це, звісно, не є повноцінним діалогом з NPC, скоріше просто пошук по ключовим словам (рис. 1.2).

Рисунок 1.2 – Симуляція реального діалогу

Крім того, подібна інтеграція сприятиме значному збільшенню реіграбельності. Оскільки взаємодія з NPC буде варіюватися в залежності від кожної нової гри, кожне проходження стане унікальним досвідом.

Однак на шляху до такої інтеграції стоїть низка технічних викликів. Перш за все, це обмеження обчислювальних ресурсів, необхідних для роботи складних моделей у реальному часі, а також потреба у безперебійному доступі до великих обсягів пам’яті. Також постають питання балансу гри: наскільки доречно вводити повністю автономних персонажів, здатних на непередбачувані дії?

Попри ці труднощі, загальна тенденція розвитку ШІ у відеоіграх очевидна. З кожним роком інтеграція нейронних мереж стає все більш реалістичною перспективою [5], і майбутнє ігрової індустрії обіцяє бути неймовірно захопливим.

1.3 Приклади створених ігр з нейромережами

Одна з них – це AI2U: With You ‘Til The End, що вийшла у Steam [6]. Гра має незвичайну концепцію: гравець опиняється в замкненому просторі разом із дівчиною, яка не хоче його відпускати. Її поведінка змінюється залежно від того, як саме гравець з нею спілкується і головна особливість у тому, що ця дівчина керується не простими репліками з діалогового дерева, а справжньою мовною нейромережею ChatGPT (рис. 1.3). В результаті вона може підтримувати розмову на різні теми, реагувати на емоційний тон голосу або тексту, а також пам’ятати, що вже було сказано.

Усе це робить гру дуже незвичною: гравець більше не читає шаблонні фрази, а веде живий діалог, у якому ніколи не знаєш наперед, що буде далі.

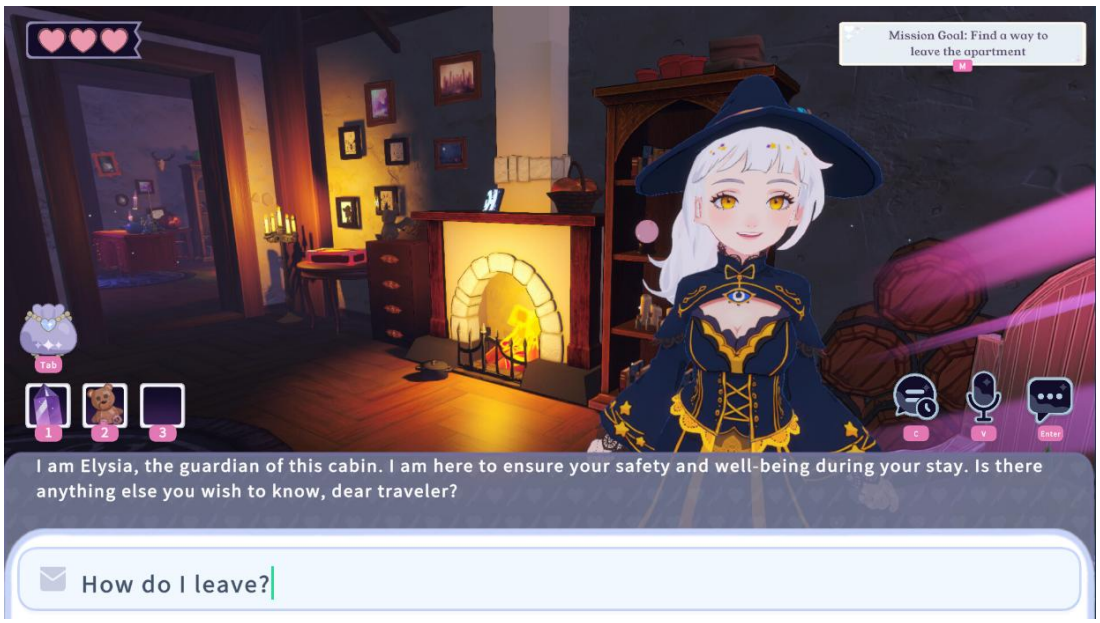

Рисунок 1.3 – Розмова з ШІ у AI2U

Такі моменти створюють новий рівень емоційного напруження. Наприклад, персонаж може образитись, змінити ставлення до гравця або навіть почати брехати – і все це залежить від того, як саме проходить спілкування.

Іншим прикладом є MIR5: The Legend of Mir (рис. 1.4) – корейська онлайн–гра, в якій було реалізовано першого у світі боса, що навчається під час боїв. Замість того, щоб діяти за заздалегідь визначеним сценарієм, цей бос аналізує, як гравці діяли під час попередніх сутичок, і з кожного разу змінює свою тактику [7]. Якщо раніше його легко перемагали певною комбінацією ударів, то наступного разу він намагатиметься її уникнути.

Цей підхід реалізовано завдяки використанню спеціальної технології від NVIDIA, яка дозволяє вбудовувати у гру адаптивну поведінку. Для гравця це означає, що бій ніколи не буде однаковим, і кожна спроба вимагатиме нового підходу [8]. Це також стимулює гравців мислити гнучко, а не просто повторювати одні й ті самі дії.

Рисунок 1.4 – Бос, керуємий нейромережою

Обидва приклади показують, що нейронні мережі в іграх вже не є фантастикою. Навпаки, вони починають змінювати уявлення про те, якими можуть бути персонажі. Гравець перестає бути єдиним джерелом динаміки в грі – тепер і сам світ гри може підлаштовуватись під гравця, створюючи більш глибокий, цікавий і живий досвід.

1.4 Майбутнє ігор з вдосконаленим ШІ

Окрім покращення поведінки NPC, впровадження нейронних мереж може відіграти важливу роль у формуванні може не реалістичного, але достовірного ігрового світу. Успішне занурення гравця у гру значною мірою залежить не лише від графіки чи динаміки, а насамперед від того, наскільки світ гри здається справжнім, послідовним і живим [9]. Тут важливо зробити відмінність між реалістичністю та достовірністю.

Реалістичність означає відповідність реальному світу – з його фізичними законами, поведінкою людей, зовнішністю об’єктів. Натомість достовірність – це внутрішня логіка вигаданого світу, його послідовність, цілісність і

здатність викликати в гравця відчуття, що цей світ міг би існувати, навіть якщо в ньому діють інші закони.

Наприклад, у всесвіті «Зоряних війн» не дотримуються звичні для нашого світу фізичні закони – тут є гіперпростір, світлові мечі, розумні інопланетні раси, Сила, кораблі у космосі мають максимальну швидкість і так далі, але завдяки глибоко продуманій міфології, політичній структурі, культурі планет, характерам персонажів і внутрішній узгодженості всіх елементів, цей світ здається достовірним. Саме це спонукає мільйони фанатів по всьому світу не просто грати чи дивитися фільми, а хотіти жити в цьому всесвіті, створювати фанфіки, вигадувати нові персонажі й історії.

На жаль, у більшості сучасних ігор NPC залишаються «порожніми оболонками». Вони повторюють одну і ту саму репліку, не реагують на зміну ситуації, не мають «особистості». Вони не пам’ятають, що гравець зробив 10 хвилин тому, не розуміють контексту і, по суті, є лише декорацією. Це різко знижує відчуття достовірності – гравець більше не відчуває, що він є частиною живого світу.

І тут на сцену виходить ШІ нового покоління – нейронні мережі з мовними та поведінковими можливостями, які дозволяють персонажам не лише реагувати на слова гравця, а й пам’ятати їх, аналізувати, будувати нові висновки. Уявіть собі світ, у якому кожен NPC має свою мотивацію, життєву історію, відношення до гравця, змінне ставлення до інших персонажів. Він реагує на зраду, вдячність, агресію або допомогу.

Саме завдяки таким технологіям ігри майбутнього зможуть знищити бар’єр між гравцем та симуляцією, дозволивши повністю зануритися у події. NPC стануть справжніми співрозмовниками, союзниками, ворогами. Гравець відчуватиме, що він має реальний вплив на світ гри, а не лише обирає серед приблизно 3 заготовлених опцій.

Уявімо RPG, де замість діалогового колеса гравець просто говорить у мікрофон, як у реальному житті або пише текст у вікні чату, а персонаж відповідає, задає зустрічні питання, враховує емоційний тон і контекст.

Наприклад: гравець віддає наказ солдату обороняти місто, а той запитує: «А що буде з моєю родиною? Вони ще в тому кварталі, де вже почались вибухи». Такі моменти формують глибокий емоційний зв’язок, якого нинішні ігри досягнути не можуть.

Все це виводить поняття «гри» на новий рівень – у бік живого, інтерактивного світу, де гравець стає не просто споживачем контенту, а співавтором історії, яка народжується тут і зараз.

1.5 Огляд існуючих ігрових рушіїв.

Перед кожним розробником завжди встає вибір, який саме ігровий рушій використовувати. На сьогоднішній день є декілька варіантів. По-перше, це Unreal Engine, це дуже потужний інструмент, який має велику кількість вже реалізованих рішень для спрощення розробки і він дає неймовірно красиву картинку, але за все доводиться платити, і в даному випадку це погана оптимізація, проблеми з документацією та неможливість реалізовувати нічого, окрім ігор.

По-друге, є Unity – один із найпопулярніших рушіїв серед інді-розробників і студентів. Його головна перевага – це гнучкість, підтримка як 2D, так і 3D графіки, величезна кількість документації, прикладів, плагінів та активна спільнота. Завдяки цьому новачки швидко можуть отримати результат, а досвідчені розробники – налаштувати все під свої потреби. Unity добре підходить для мобільних, десктопних і навіть WebGL-проєктів, що робить його універсальним вибором [10]. І хоча зображення, створені в Unity, часто поступаються Unreal за якістю, оптимізація, швидкість розробки та доступність – це ті фактори, які важко переоцінити. До того ж, у Unity дуже зручна робота з фізикою, а для ШІ – є підтримка навігаційних сіток, таких як NavMesh і сторонніх розширень на кшталт NavMeshPlus.

Ще один рушій, який заслуговує на увагу, – це Godot. Він є повністю безплатним і з відкритим вихідним кодом [11]. Хоча він не настільки потужний у 3D як Unity або Unreal, зате для 2D-проєктів він є дуже зручним і легким у використанні. Величезною перевагою Godot є його простота, а також те, що він займає дуже мало пам’яті та швидко запускається. До того ж, багато речей у ньому реалізовано через сценову систему, яка є зрозумілою навіть для новачків.

Окремо варто згадати про самописні рушії. Їх зазвичай створюють розробники, які мають специфічні вимоги до продуктивності або хочуть повністю контролювати всі процеси [12-13]. Самописні рушії дозволяють оптимізувати гру під конкретну задачу, позбавитися зайвого коду, зекономити на розмірі збірки або досягти незвичних результатів. У професійних командах або на великих проєктах самописні рушії використовуються досить часто – прикладами є рушії Ubisoft, id Tech або CD Projekt RED (рис. 1.5).

Рисунок 1.5 – Приклади популяних та в більшості саморобних рушіїв

1.6 Постановка задачі

Враховуючи обмеження традиційних алгоритмів побудови штучного інтелекту в іграх, зокрема їхню слабку адаптивність, шаблонність поведінки та відсутність гнучкого реагування на зміну ігрових умов, постає завдання створити нового типу супротивника, який прийматиме рішення за допомогою нейронної мережі. Такий підхід дозволить впровадити адаптивну модель поведінки, що буде залежати від актуального стану ігрової сцени та дій гравця.

Об’єктом роботи є вдосконалення штучного інтелекту в іграх. Метою роботи є розробка тактичної гри з використанням нейронної мережі для аналізу ігрової сцени та ухвалення рішень поведінки персонажів.

Під час реалізації системи необхідно забезпечити стислий формат передавання інформації до моделі [14], мінімізувати затримку у прийнятті рішень та гарантувати коректність обробки навіть у випадку неповних чи спотворених даних. Важливо, щоб модель могла швидко обирати одну з заздалегідь визначених тактик — атакувати, відступити, сховатися або утримувати позицію — з урахуванням наявної ситуації на полі бою. Система має бути легко інтегрована у рушій Unity без суттєвих змін основної логіки гри. Також слід передбачити можливість подальшого масштабування: як за рахунок розширення переліку тактик, так і шляхом додавання нових вхідних параметрів.

Для оцінки ефективності роботи моделі має бути реалізований набір тестових ігрових сценаріїв [15], які охоплюють різні ситуації — від прямого зіткнення до маневрової боротьби. Потрібно передбачити збір статистики щодо рішень моделі, часу їх прийняття та результатів у грі. Уся система повинна працювати стабільно, з оптимальним навантаженням на ресурси, у режимі реального часу, без помітних затримок або збоїв.

2 КОНЦЕПТУАЛЬНА МОДЕЛЬ ІГРОВОЇ ПОВЕДІНКИ NPC З ВИКОРИСТАННЯМ НЕЙРОННОЇ МЕРЕЖІ

2.1 Основи інтеграції нейронної мережі в Unity

Інтеграція нейронних мереж у комп’ютерні ігри на базі рушія Unity є перспективним напрямом розвитку ігрового штучного інтелекту. Зважаючи на обмеження ресурсів локальних пристроїв та складність безпосередньої реалізації великих нейронних моделей у середовищі Unity, найпоширенішим рішенням є використання зовнішніх сервісів через мережеві запити. Одним із найбільш популярних рішень у цій сфері є використання API від OpenAI.

Механізм інтеграції нейронної мережі базується на формуванні запиту, що містить опис поточного стану ігрової сцени, передачею цього запиту на сервер через HTTP–протокол та обробкою відповіді, яка надходить у вигляді текстової рекомендації. У середовищі Unity для роботи із запитами зазвичай застосовується клас UnityWebRequest (рис. 2.1), який дозволяє створювати, налаштовувати і надсилати HTTP–повідомлення з відповідними заголовками.

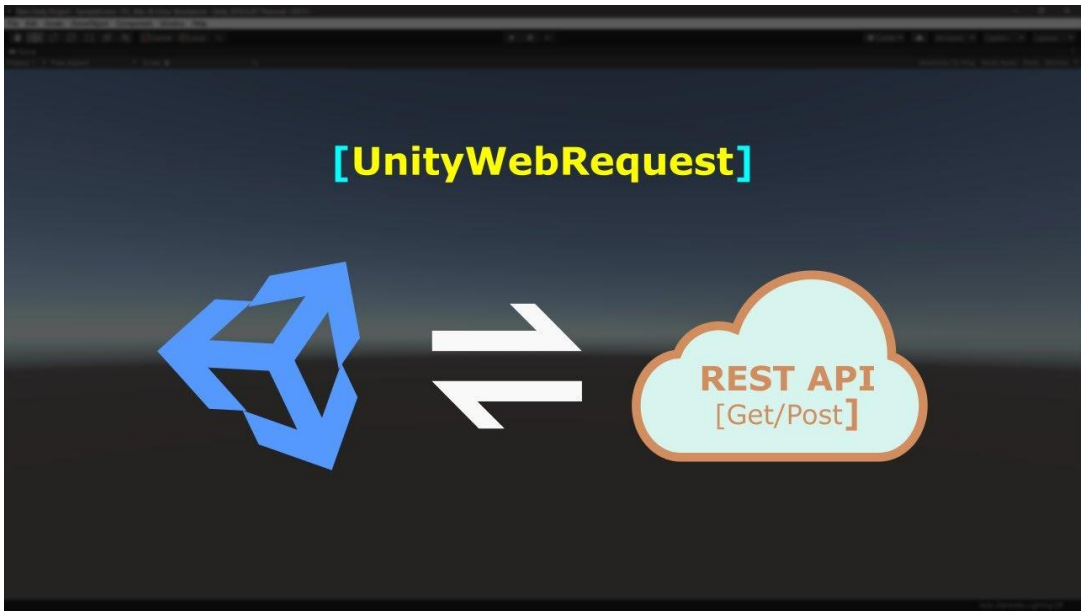

Рисунок 2.1 – Unity Web Request

Для доступу до сервісу для інтеграції нейромережі обов’язковим є наявність API–ключа, який генерується у кабінеті користувача. Ключ повинен бути включений до кожного запиту у спеціальному полі заголовків авторизації. При цьому варто враховувати питання безпеки: ключ не рекомендується зберігати у відкритому вигляді у коді клієнта, щоб уникнути можливого його компрометування.

Щоб забезпечити безперебійну роботу гри під час обробки запитів, процес відправлення і отримання відповідей реалізується асинхронно. У Unity для цього широко застосовуються корутини, що дозволяють виконувати тривалі операції без блокування основного потоку гри. Таким чином, навіть у випадку затримок із відповіддю сервера, геймплей залишається плавним, а супротивник може вчасно реагувати на події.

Важливою складовою інтеграції є правильне формування тексту запиту. У розробленій системі запит містить стислий опис поточної бойової ситуації англійською мовою, включаючи такі параметри як рівень здоров’я обох учасників бою, величину завданої шкоди, час до можливості наступного пострілу, дистанцію до супротивника та найближчого укриття, а також швидкість руху танків. Чітка структура запиту дозволяє мінімізувати ризик неоднозначного трактування даних нейронною мережею та забезпечити високу точність отриманих рекомендацій.

Отримана відповідь аналізується і перетворюється у відповідні ігрові дії. Зазвичай відповідь має форму короткої інструкції на кшталт «Take cover immediately» або «Engage the enemy», яку система розпізнає і реалізує через зміну поведінки супротивника. Коректна обробка відповідей є важливою частиною інтеграції, оскільки дозволяє забезпечити логічну і природну реакцію на зміну умов на полі бою.

Оскільки обмін даними із зовнішнім сервером відбувається через мережу Інтернет, необхідно передбачити механізми обробки затримок та збоїв з’єднання. Для цього у грі реалізується система тайм–аутів: якщо відповідь не

надходить протягом визначеного часу, активується базова стратегія поведінки, що дозволяє уникнути зависання супротивника у критичних ситуаціях.

Також варто враховувати ліміти кількості запитів, які встановлюються політикою компанії для різних тарифних планів. Для оптимізації взаємодії із сервером у грі передбачено контроль частоти звернень: запити надсилаються лише у разі суттєвих змін ігрової ситуації, таких як різке зниження здоров’я або зміна позицій гравців.

З метою підвищення продуктивності і зменшення навантаження на сервер може бути реалізоване кешування відповідей у стандартних ситуаціях. Наприклад, якщо ігрова сцена тривалий час залишається незмінною, система може використовувати раніше отриману відповідь без надсилання нового запиту.

Інтеграція нейронної мережі через Web–запити в середовищі Unity дає можливість значно підвищити гнучкість і реалістичність поведінки супротивників, зробити їх менш передбачуваними для гравця і забезпечити динамічний розвиток подій на полі бою. За умови правильного налаштування обробки запитів і відповідей така система здатна суттєво підвищити якість ігрового процесу і надати гравцям новий рівень взаємодії із супротивниками.

2.2 Формування запитів для аналізу стану ігрового світу

Один з ключових етапів інтеграції нейронної мережі в ігровий процес – це формування текстового запиту, який надсилається на обробку моделі. Саме цей запит забезпечує нейромережу усією необхідною інформацією для аналізу ситуації в бою та вибору подальших дій. Від того, наскільки чітко, лаконічно та структуровано подано дані, залежить якість відповіді та швидкість її обробки.

У розробленій грі кожен запит формується на основі поточного стану ігрової сцени. До запиту включаються ключові параметри, які мають

найбільший вплив на тактичне рішення. Це, зокрема, рівень здоров’я гравця та супротивника, сила потенційного урону з обох боків, час до можливості наступного пострілу, дистанція до ворога, наявність укриття, а також швидкість обох танків. Усі ці дані дозволяють створити достатньо повну картину для ухвалення адекватного рішення.

Структура запиту побудована таким чином, щоб бути максимально зрозумілою для мовної моделі. Текст складається англійською мовою у вигляді переліку коротких тверджень, які формально, але інтуїтивно описують ситуацію. Наприклад, один із запитів може мати такий вигляд: «Your HP: 100, Enemy HP: 200, Your Damage: 150, Enemy Damage: 80, You will be able to shoot in 3s, Enemy will be able to shoot in 10s, Distance to enemy: 20m, Distance to cover: 5m, Your speed: 5 m/s, Enemy speed: 2 m/s.»

Формат подачі інформації є уніфікованим для всіх ситуацій. Це означає, що незалежно від того, які саме дані змінились – здоров’я, відстань, час перезарядки – вся структура запиту залишається сталою. Такий підхід забезпечує послідовність у взаємодії з моделлю та дає змогу уникнути непорозумінь або помилок при обробці запитів.

Кожен із параметрів запиту має фіксоване місце і формулювання. Важливо не тільки подати числове значення, а й правильно пояснити, до якого об’єкта воно відноситься. Наприклад, швидкість чи шкода мають бути чітко розділені для гравця і супротивника, щоб модель не сплутала значення між собою. Також обов’язковим є зазначення одиниць виміру, що допомагає уникнути плутанини у числах і масштабах.

Щоб уникнути зайвого навантаження, у гру вбудований механізм перевірки, чи дійсно варто формувати новий запит. Якщо ситуація на полі бою не змінилася суттєво, наприклад танк стоїть на місці, не стріляє і не отримує урон, запит не генерується повторно. Це дозволяє зменшити кількість звернень до сервера і тим самим уникнути перевищення лімітів API, а також знизити затримки у грі.

Запит формується автоматично кожного разу, коли в грі спрацьовують певні тригери. Такими тригерами можуть бути зміна здоров’я, завершення перезарядки, поява укриття у зоні досяжності або суттєва зміна відстані до супротивника. Завдяки цьому система відстежує лише дійсно значущі зміни, що впливають на рішення.

Формування тексту запиту відбувається в коді гри. На кожному кроці враховується, що модель – це не програма, яка аналізує числові масиви, а саме мовна система, яка краще працює із текстовими описами. Тому особливу увагу приділено стилю формулювання: речення повинні бути простими, без надмірної художності або термінів, які можуть бути інтерпретовані неоднозначно.

У разі, якщо певні параметри тимчасово недоступні (наприклад, супротивника не видно), система не прибирає їх із запиту, а замінює на фіксоване значення. Таким чином структура не порушується, а модель все одно отримує сигнал, що ці дані наразі відсутні або умовні.

Важливою перевагою цього підходу є гнучкість. У разі, якщо у майбутньому буде потреба додати нові параметри, структура запиту легко розширюється. Наприклад, можна буде додати інформацію про тип ландшафту, кількість боєприпасів чи рівень броні, просто додавши нові рядки до запиту у тому ж форматі.

Також потрібно передбачити обробку помилок при формуванні запиту. Якщо будь–яке значення виходить за логічні межі, наприклад час до пострілу стає від’ємним, система автоматично виправляє це значення до нульового або допустимого мінімуму.

2.3 Вибір нейронної мережі для інтеграції

Для вибору найкращої моделі було порівняно топ 5 нейромереж за версією сайта lmarena.ai (табл. 2.1).

Таблиця 2.1 – Найкращі моделі для чату

Місце	Модель	Arena бали
1	Early grok 3	1403
2	Gemini–2.0–Flash–Thinking–Exp–01–21	1385
3	gemini–2.0–pro–exp–02–05	1380
4	chatgpt–4o–latest–20250129	1377
5	deepseek–r1	1363

Для порівняння нейромереж був сформований набір даних з різних тактик, між якими потрібно обрати найкращу, а саме опис карти, танків, та ситуації, коли необхідно прийняти рішення. Наприклад, «**Map**: Middle-sized map with covers in center. **Tank**: You have a medium tank with good speed and damage, but with weak armor, meanwhile enemy has heavy tank, which can`t be damaged in front armor. **Situation**: Your HP: 100, Enemy HP:200, Your Damage 150, Enemy Damage 80, You will be able to shoot in 3s, Enemy will be able to shoot in 10s, Distance to enemy: 20m, Distance to cover 5m, Your speed 5 m/s, Enemy speed 2m/s.».

В роботі оцінена швидкість надання відповіді у секундах та її якість балом від 1 до 5. Для цього була створена таблиця (табл. 2.2).

Таблиця 2.2 – Порівняння нейромереж

Модель	Час на відповідь(с)	Якість відповіді(1–5)
Early grok 3	4,2	4
Gemini–2.0–Flash–Thinking–Exp–01–21	6,49	5
gemini–2.0–pro–exp–02–05	1	4
chatgpt–4o–latest–20250129	0,8	4
deepseek–r1	39	2

Модель Early Grok 3, яка наразі займає перше місце в загальному рейтингу, справді здатна генерувати цікаві й логічні відповіді, проте її

швидкість – 4,2 секунди – є на межі припустимого для швидких дій в ігровому середовищі. Така затримка не критична у покрокових або сюжетних іграх, однак у реальному бою, де танк повинен реагувати миттєво на зміну ситуації, це вже створює проблеми.

Gemini–2.0–Flash–Thinking і Gemini–2.0–Pro – це дві версії великої мовної моделі Google, які дійсно показали високий рівень якості відповідей. Особливо гарно показала себе перша версія, що отримала 5 балів, однак її середній час генерації відповіді склав майже 6,5 секунд. Це робить її практично непридатною для використання в умовах активного бою. Друга версія працює набагато швидше – всього за 1 секунду – при збереженні гарної якості тексту, однак навіть вона поступилася у швидкодії іншій моделі.

Модель chatgpt–4o–latest продемонструвала найкращий баланс. Вона відповіла за 0,8 секунди, що майже непомітно для гравця, а якість відповіді отримала 4 бали з 5. Це не найвищий результат у рейтингу, однак саме з погляду інтеграції в гру ця модель є найбільш придатною. Вона достатньо швидка, щоб працювати у режимі реального часу, і водночас здатна згенерувати тактично логічну відповідь, яка виглядає розумною навіть для досвідченого гравця.

Натомість DeepSeek–r1, хоч і входить до п’ятірки лідерів загального рейтингу, повністю провалився у контексті ігрової задачі. Його відповідь формувалася протягом майже 40 секунд, а результат не відповідав формулюванням запиту і мав дуже низьку інформативність. Це свідчить про те, що модель може бути хорошою для створення великих текстів або сценаріїв, але точно не підходить для задач ухвалення рішень на основі стислої ситуаційної інформації.

Таким чином, під час вибору нейронної мережі важливо орієнтуватися не лише на загальний рейтинг або якість у відриві від контексту, а й враховувати специфіку самої задачі. В умовах тактичної гри головним критерієм є швидкодія моделі, здатність швидко обробити короткий запит і сформувати чітку відповідь, яка не потребує додаткового тлумачення.

Також важливо враховувати стабільність моделі та її здатність до генерації коротких, лаконічних і структурованих відповідей. У випадку з chatgpt–4o ці якості поєдналися найкращим чином. Усі результати тестування підтвердили, що саме ця модель дозволяє отримати придатні для гри рекомендації із мінімальною затримкою, не жертвуючи при цьому змістом чи логікою.

Обраний підхід до порівняння, коли кожній моделі було надано однаковий опис ситуації з боєм, дозволив максимально чесно оцінити їхню ефективність.

2.4 Проблеми та виклики при інтеграції нейронної мережі в Unity

Під час проектування гри з використанням нейронної мережі для керування супротивником виникає низка проблем, які потрібно передбачити ще до початку реалізації. На перший погляд може здатися, що достатньо просто під’єднати модель і надсилати їй дані, але на практиці все значно складніше.

Одна з перших складностей – це відсутність у Unity готових інструментів для зручної роботи з мовними моделями на кшталт GPT. Щоб налаштувати обмін з нейронною мережею, потрібно реалізувати систему HTTP–запитів, сформувати правильний формат даних, налаштувати заголовки, обробку JSON–відповідей та все це запустити без гальмування гри. Усе це має працювати асинхронно, інакше гравець буде відчувати затримки навіть під час звичайної взаємодії з грою.

Ще одна важлива річ – стабільність інтернет–з’єднання. Якщо модель розміщується на сервері, гра повністю залежить від якості зв’язку. Якщо з’єднання зникне, супротивник без запасного плану просто зупиниться. Тому необхідно передбачити поведінку на випадок, якщо модель тимчасово не

доступна. Це можуть бути прості шаблонні дії – рух до укриття, пасивна оборона або просто утримання позиції.

Не менш суттєвою є проблема затримки між надсиланням запиту і отриманням відповіді. Навіть найшвидші моделі типу GPT–4o не завжди повертають результат миттєво. Якщо додати до цього ще затримку в мережі, то може пройти кілька секунд, перш ніж танк–супротивник отримає команду. Це неприпустимо в ситуаціях, де рішення треба приймати негайно. Щоб уникнути такого гальмування, потрібно встановити таймер, який буде слідкувати за часом очікування. Якщо відповідь не приходить за вказаний проміжок, гра просто переходить до однієї з базових поведінок.

Окрему складність становить те, що нейромережа не завжди повертає відповідь у потрібному форматі. Замість чіткої команди на кшталт «Take cover» іноді приходить розпливчасте речення з порадою або кілька варіантів дій. У таких випадках потрібно реалізувати перевірку: чи містить відповідь ключові слова, чи підходить вона до шаблону. Якщо ні – модель отримує повідомлення, що відповідь сформульована неправильно, але повторна відповідь не чекається. Замість цього гра відразу обирає випадкову з кількох заздалегідь підготовлених тактик. Це дозволяє уникнути подвійної затримки і підтримувати динаміку гри.

Ще один виклик – робота з API–ключем. Без нього доступ до нейромережі неможливий, але зберігати його прямо в коді – небезпечно. Якщо хтось дізнається цей ключ, він зможе надіслати тисячі запитів, що призведе до блокування. Тому важливо передбачити безпечне зберігання ключа – наприклад, на сервері або в зашифрованому вигляді.

Також слід враховувати ліміти самої моделі. Наприклад, OpenAI має обмеження на кількість запитів у хвилину та добову кількість символів. Якщо надсилати забагато запитів або занадто довгі тексти, сервіс просто відмовиться їх приймати. Щоб цього уникнути, у грі запит формується тільки тоді, коли справді щось змінилося – наприклад, гравець наблизився, танк отримав пошкодження або з’явилось укриття поблизу.

Тестування роботи з мовною моделлю – окрема складність. Через те, що нейромережа може по–різному реагувати на схожі запити, звичайне автоматичне тестування тут не підходить. Потрібно вручну запускати різні ситуації, спостерігати за поведінкою супротивника і виписувати випадки, коли він реагує неадекватно. Для складних сцен іноді потрібно створювати спеціальні сценарії, які дозволяють точно відтворити ту чи іншу ситуацію.

Ще одна проблема – довжина запиту. Якщо написати коротко, модель може не зрозуміти контекст. Якщо зробити запит занадто детальним, виникне затримка. Потрібно знайти «золоту середину» – передати тільки найважливішу інформацію у зрозумілому вигляді. Наприклад, не просто «здоров'я танка 50», а «Your HP: 50» – англійською, у формі зрозумілій моделі.

Загалом, інтеграція нейронної мережі в Unity – це більше, ніж просто підключити API. Це набір проблем, які потрібно враховувати заздалегідь. Тут важливі і технічні аспекти, і логіка поведінки, і навіть взаємодія з гравцем. Система має працювати плавно навіть тоді, коли сервер недоступний або відповідь надходить із затримкою.

Щоб забезпечити надійність, краще одразу закласти багаторівневу логіку. Перший рівень – це основна поведінка, яку формує нейромережа. Другий – резервна реакція на випадок затримки чи помилки. І третій – найпростіша автономна поведінка, яка запускається, якщо з'єднання повністю втрачено. Такий підхід дозволяє зберегти плавність гри та забезпечити стабільність навіть у нестандартних умовах, таких як відключення світла або тимчасові проблеми з інтернетом.

2.5 Управління танком гравцем і NPC

У проєктованій грі передбачається реалізація двох окремих систем управління – для гравця і для супротивника [16]. Обидві сторони мають

однаковий набір базових дій, але відрізняються способом ухвалення рішень. Якщо гравець керує танком безпосередньо, використовуючи пристрої введення, то дії супротивника планується формувати на основі відповідей нейронної мережі.

Для гравця планується реалізація класичного управління за допомогою клавіш клавіатури. Очікується, що кнопки будуть відповідати за рух танка вперед і назад, а також за поворот ліворуч і праворуч. Поворот планується виконувати через обертання корпусу навколо центральної осі, що дасть змогу точно контролювати напрямок руху. Рух при цьому має бути реалізовано через фізичний компонент Rigidbody2D, що дозволить враховувати інерцію, силу тертя та плавність пересування.

Окремо передбачалася механіка стрільби, яка була реалізована з обмеженням у вигляді перезарядки. Тобто після кожного пострілу має запускатися таймер, що забороняє стріляти до завершення періоду перезарядки. Це додасть ігровому процесу додаткової динаміки і змусить гравця обдумувати свої дії наперед.

Система управління супротивником була побудована інакше. Замість безпосереднього контролю планується реалізувати цикл, у якому гра періодично формуватиме опис поточної ситуації на полі бою. У запит передбачається включати дані про здоров’я обох танків, кількість урону, відстань між ними, стан перезарядки та наявність доступного укриття. Уся ця інформація буде передаватися на сервер мовної моделі у вигляді англійського тексту.

Нейронна мережа, отримавши опис ситуації, має повернути коротку відповідь – одну з доступних команд, таких як «Attack», «Retreat», «Take cover» або подібну. У відповіді не планується жорстко задавати координати чи маршрути – лише загальний тип дії. Виходячи з отриманої відповіді, в ігровій логіці супротивника буде активовано відповідний сценарій поведінки [17].

Наприклад, якщо буде отримана команда «Attack», супротивник має розвернутись у бік гравця, наблизитися на задану дистанцію та підготуватися

до стрільби. У випадку відповіді «Take cover» планується активація алгоритму, що дозволяє знайти найближче укриття та пересуватись до нього з урахуванням перешкод і власної швидкості. Для таких сценаріїв передбачається реалізація базової навігації у межах 2D–простору.

У випадках, коли модель поверне відповідь, що не підпадає під жоден з відомих шаблонів, має бути реалізована перевірка її змісту. Якщо відповідь не відповідає очікуваному формату або порожня, система має надіслати повторний запит з уточненням формату. Проте, щоб не створювати додаткових затримок, повторна відповідь не очікується. Натомість планується активація однієї з простих запасних поведінок, обраної випадково: наприклад, оборона на місці, маневрування або відхід на невелику дистанцію.

Функціональність обох типів танків – гравця та NPC – загалом майже однакова. Обидва зможуть переміщатися у двох напрямках, обертатися, стріляти, перезаряджатися та шукати укриття. Принципова різниця полягатиме лише у способі прийняття рішень. Гравець буде робити це вручну в режимі реального часу, а супротивник – шляхом обробки текстового запиту за допомогою мовної моделі.

Планується також реалізація незалежного повороту башти, що дозволить вести вогонь не змінюючи напрямку руху. Для гравця ця функція активується окремим механізмом прицілювання, а для супротивника – автоматично, орієнтуючись на позицію гравця. Башта буде повертається незалежно від корпусу, забезпечуючи більш гнучку і точну стрільбу.

У системі стрільби був реалізований також механізм врахування перешкод, які можуть блокувати постріл. Наприклад, якщо гравець знаходиться за укриттям, супротивник має оцінити, чи є сенс стріляти, чи варто змінити позицію. Для цього потрібно було створити просту систему перевірки видимості мішені.

Усі описані дії викликаються у відповідь на команди нейронної мережі або натискання клавіш гравцем. Таким чином, логіка керування в грі поєднуватиме два різні підходи: ручне управління та зовнішній ШІ, який діє

на основі аналізу ситуації. Така система має забезпечити гнучкість, варіативність боїв та умови для тактичної взаємодії.

2.6 Розробка тактичних сценаріїв для поведінки супротивника

Щоб зробити супротивника у грі більш реалістичним і непередбачуваним, була впроваджена система тактичних сценаріїв, між якими нейронна мережа має обирати залежно від поточної ситуації. Замість жорстко прописаної логіки у вигляді «помітив гравця – стріляй», система повинна працювати гнучко, орієнтуючись на умови бою, параметри обох сторін і стан укриттів.

Модель повертає не точну команду до дії, а назву тактичного стану, який потім інтерпретується ігровою логікою. Такий підхід дозволяє структурувати поведінку супротивника та зменшити ризик неоднозначної реакції. Кожен тактичний стан містить кілька можливих дій, які можуть реалізовуватись залежно від контексту. Це дозволяє зберегти варіативність, але водночас тримати поведінку в межах очікуваного шаблону [18].

Було реалізовано чотири основні тактичні стани: «Retreat», «Defense», «Attack» і «Breakthrough». Кожен із них має власні підсценарії, які описують конкретну поведінку на полі бою.

Стан «Retreat» активується у випадках, коли супротивник має критично низький рівень здоров’я, програє за кількістю шкоди або не має змоги вести вогонь через перезарядку. У межах цього сценарію передбачено три варіанти дій:

- простий відступ – «Run away»;
- відхід назад без укриттів – «Retreat backward without cover»;
- відступ із використанням укриттів – «Retreat using cover».

Перший варіант передбачає максимально швидке покидання зони обстрілу, без урахування перешкод. Другий – повільніший, але з обережним

рухом заднім ходом. Третій варіант вимагає реалізації системи пошуку доступних укриттів у напрямку від гравця, щоб супротивник міг прикритися хоча б частково під час відступу.

Сценарій «Defense» буде актуальним у ситуаціях, коли супротивник не має переваги, але й не перебуває в критичному стані. Його завдання – стримувати гравця, відповідаючи вогнем та зберігаючи свою позицію. У межах цього стану передбачається три підходи:

- утримання позиції – «Hold position defensively»;
- вогонь з–за укриття – «Fire back using cover»;
- стояння на місці з прямим обстрілом – «Stand still and fire back».

Ці варіанти дають змогу зробити поведінку супротивника спокійною, але напруженою. Якщо в полі зору є укриття, він має використовувати його. Якщо немає – зберігати позицію і стріляти у відповідь. Така поведінка буде природною для ситуації, коли супротивник очікує на дії гравця.

Стан «Attack» буде застосовуватись тоді, коли супротивник має помітну перевагу – наприклад, більше здоров’я, коротший час до пострілу або кращу позицію. У цьому випадку модель може рекомендувати:

- повільне просування з використанням укриттів – «Advance slowly using cover»;
- дистанційний обстріл – «Attack while keeping distance»;
- ближню атаку – «Attack at close range».

Ці підсценарії мають бути взаємозамінні й активуватись залежно від розташування об’єктів на карті. Якщо є укриття на шляху – активується варіант з повільним просуванням. Якщо дистанція до гравця велика – супротивник може залишатися на місці й вести обстріл здалеку. Якщо гравець ослаблений і близько, тоді обирається агресивна ближня атака.

Останній сценарій – «Breakthrough» –використовується у рідкісних ситуаціях, коли супротивник має явну тактичну перевагу або доступ до флангів гравця. Це дозволяє зробити його дії несподіваними і змінити темп бою. У межах цього стану передбачувано дві ключові дії:

- обхід з боку – «Move in and attack from the side»;
- маневр для атаки з тилу – «Move in and attack from behind».

Ці тактики потребують реалізації системи навігації та оцінки позиції гравця. Якщо супротивник бачить, що може обійти гравця або підійти з іншого боку, він може різко змінити маршрут. Це зробить бій менш лінійним і змусить гравця постійно змінювати стратегію.

Також важливо було передбачити випадки, коли модель повертає щось незрозуміле або неочікуване. У таких ситуаціях може активуватись стандартна захисна поведінка або випадковий підсценарій із блоку «Defense», як найбезпечніший варіант (рис. 2.2).

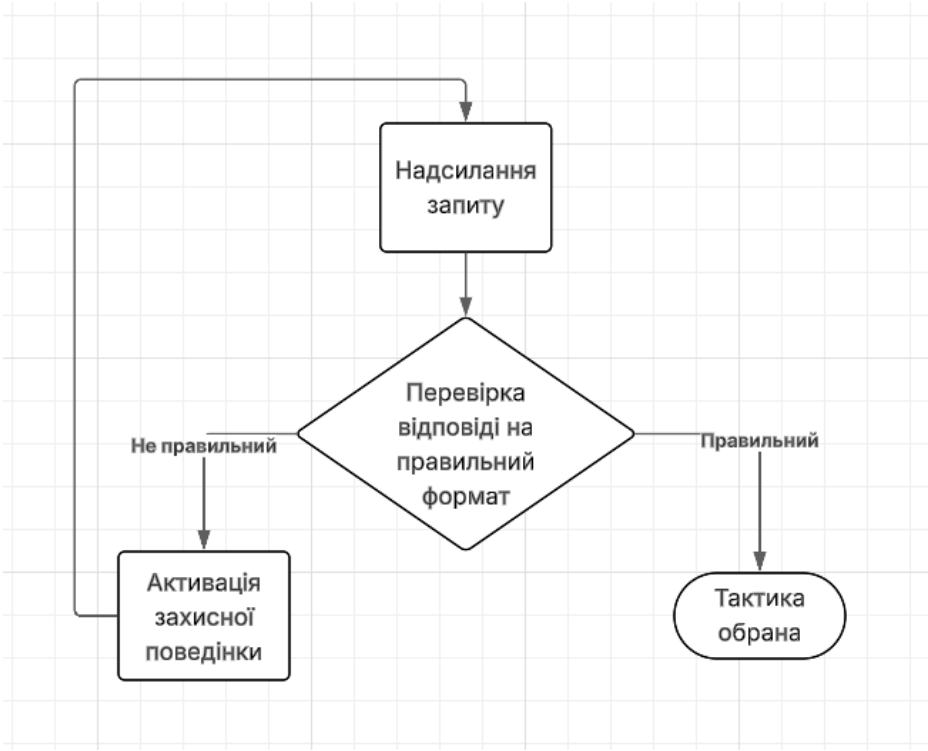

Рисунок 2.2 – Перевірка відповіді від нейромережі

Щоб така система працювала стабільно, потрібно, щоб був зроблений механізм зіставлення отриманої відповіді з певною тактикою. Якщо модель повертає назву одного зі сценаріїв, активується відповідна поведінка. Якщо у відповіді присутній лише підсценарій (наприклад, «Attack at close range»), система має визначити, до якого стану він належить, і діяти відповідно.

Такий підхід до побудови логіки поведінки супротивника дає змогу зробити ШІ більш схожим на живого противника. Він не просто стріляє, як тільки бачить гравця, а реально оцінює ситуацію, змінює позиції, атакує або відступає. Кожен бій при цьому виглядає по–різному, навіть якщо початкові умови були однаковими.

Така структура сценаріїв дозволяє масштабувати гру в майбутньому – можна додавати нові тактики, адаптувати поведінку до типу мапи чи навіть змінювати стиль ШІ залежно від рівня складності [19]. Але навіть із базовим набором у чотири стани і дванадцять підсценаріїв можливо досягти значної різноманітності та глибини геймплею (рис. 2.3).

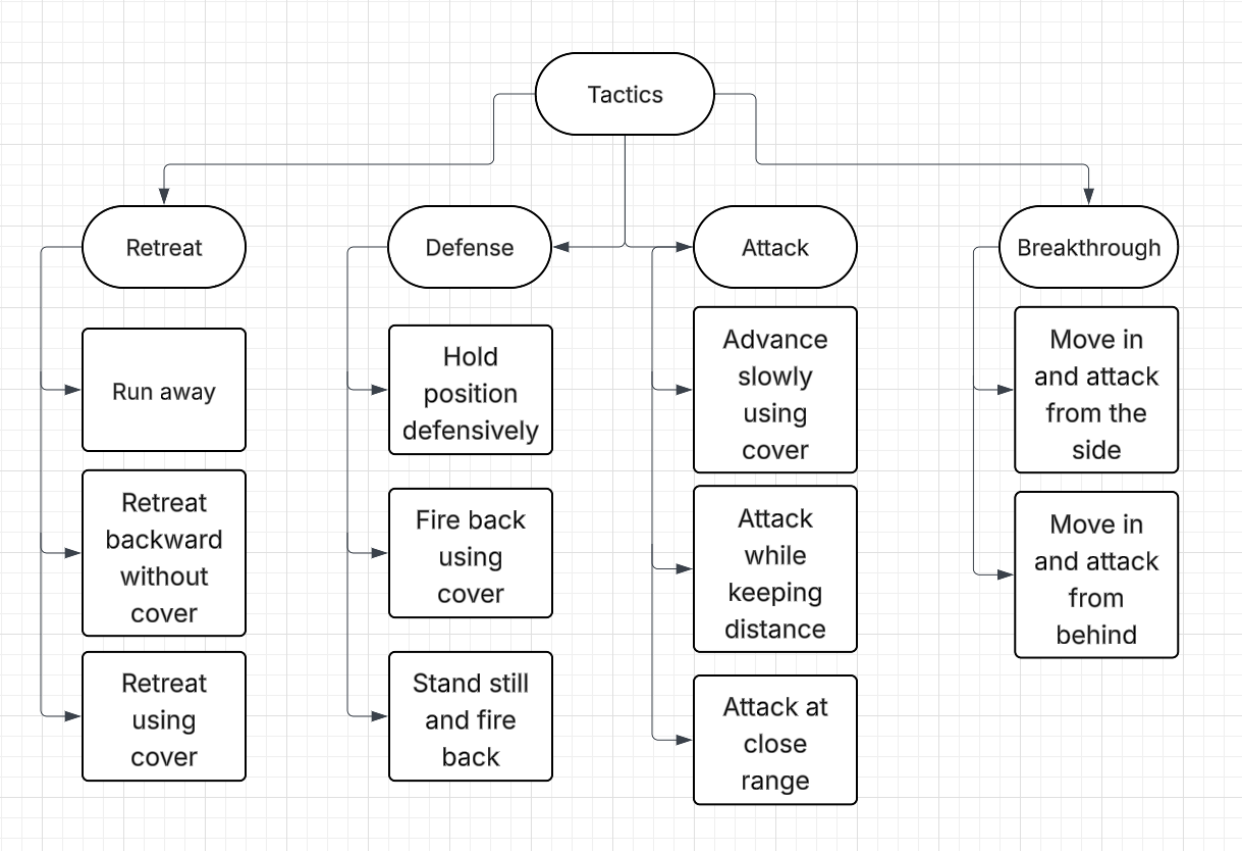

Рисунок 2.3 – Усі можливі тактики

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АДАПТИВНОЇ ПОВЕДІНКИ СУПРОТИВНИКА В ГРІ

3.1 Вибір середовища розробки

Для реалізації гри, де поведінку супротивника визначає нейронна мережа, потрібно середовище, яке дозволить зручно створити як ігрову частину, так і налаштувати обмін даними з зовнішнім сервісом. Після аналізу доступних варіантів було вирішено працювати з рушієм Unity, мовою програмування C# та редактором Visual Studio Code. Таке поєднання забезпечує потрібну функціональність, стабільність і достатню гнучкість на всіх етапах розробки.

Unity є сучасним рушієм, який підтримує як двовимірну, так і тривимірну графіку, фізику, систему колайдерів, а також дозволяє легко працювати з тегами, шарами та компонентами. Це спрощує створення сцени з гравцем, супротивником і укриттями. Важливо, що Unity також дозволяє використовувати бібліотеку NavMeshPlus (рис. 3.1),Це дозволить організувати рух ворожого танка з урахуванням перешкод і укриттів, що критично для виконання таких тактик, як «відступ з використанням укриття» або «флангова атака».

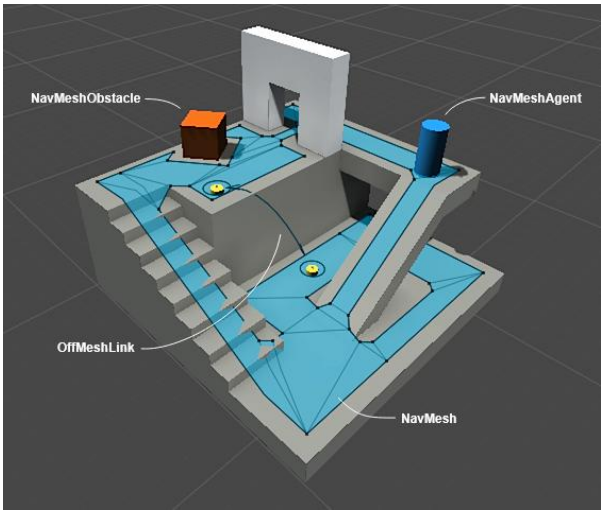

Рисунок 3.1 – система навігації рушія Unity

Вибір мови C# обумовлений тим, що вона є основною для Unity, має велику кількість документації та дозволяє створювати модульну структуру проєкту. Це дає змогу розділити логіку на окремі компоненти – наприклад, обробку здоров'я, стрільбу, пересування та взаємодію з нейронною мережею.

Редактор Visual Studio Code (VSC) з відповідними розширеннями забезпечує підсвічування синтаксису, автодоповнення, зручну навігацію по коду та роботу з системами контролю версій (рис. 3.2). Завдяки своїй легкості та швидкості він не обтяжує систему, що важливо при паралельному запуску рушія Unity та емуляторів.

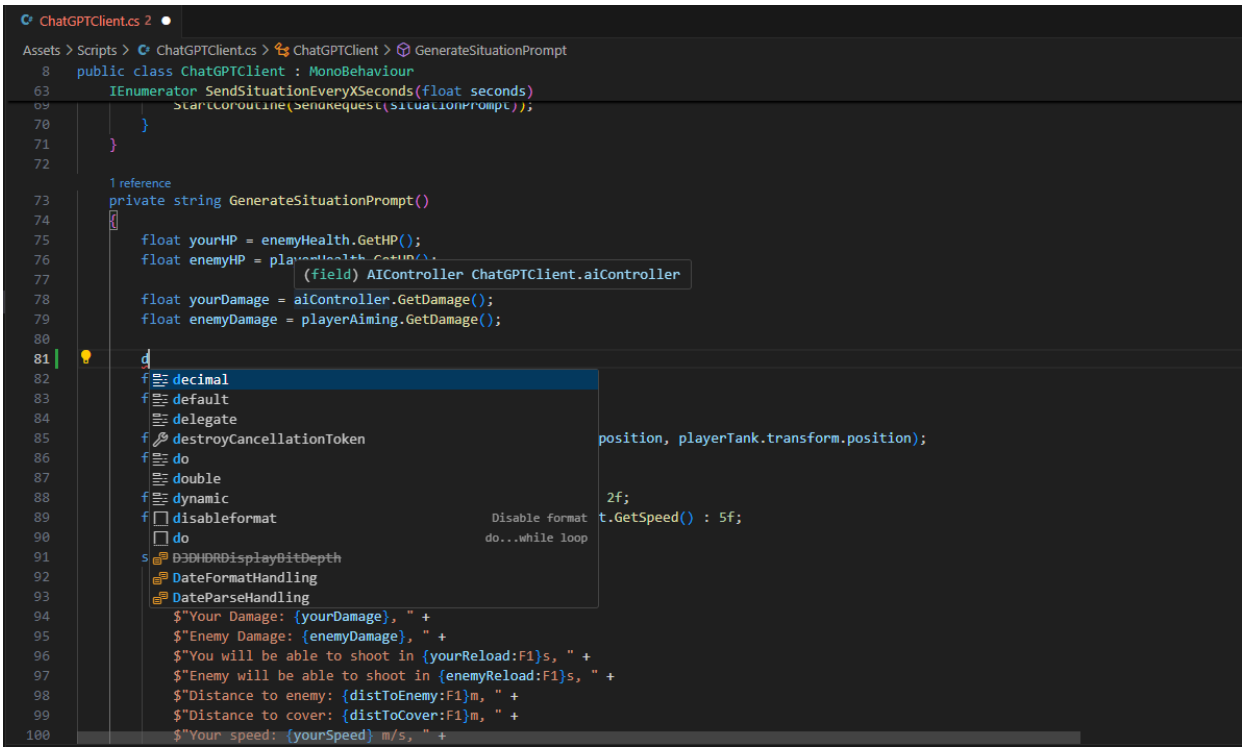

Рисунок 3.2 – Коректор синтаксису в VSC

Також Unity підтримує виконання HTTP–запитів, що дає змогу зв'язати гру з зовнішнім сервісом – наприклад, із платформою OpenAI [20]. Це дозволяє надсилати інформацію про поточну ситуацію в бою та отримувати у відповідь тактику, яку потрібно застосувати [21]. Для обробки відповіді у форматі JSON використовується бібліотека Newtonsoft.Json, яка проста у використанні та добре підходить для швидкої серіалізації й десеріалізації даних.

Таким чином, обране середовище дозволяє створити не просто звичайну 2D–гру, а систему з адаптивним супротивником, що змінює свою поведінку залежно від ситуації на полі бою.

3.2 Архітектура проекту та структура сцени

Для побудови гри було створено одну основну сцену, в якій розташовано всі ключові об’єкти: гравець, ворожий танк, укриття, камера та інтерфейс. Уся гра реалізована в двовимірному просторі, і переміщення відбувається в площині *X*–*Y*. Структура сцени була продумана таким чином, щоб забезпечити просту, але функціональну взаємодію між елементами [22].

Укриття були реалізовані як звичайні об’єкти з колайдерами, яким присвоєно тег «Cover». Це дозволило скрипту штучного інтелекту знаходити найближчі з них і використовувати для реалізації тактик оборони або відступу. Позиції укриттів задавались вручну у редакторі сцени, що дало змогу точно контролювати бойову зону.

Гравець і супротивник були реалізовані як окремі об’єкти, кожен з яких містить набір компонентів:

— танк гравця має компоненти Rigidbody2D, PlayerMovement.cs, PlayerAiming.cs, Health.cs, а також UI–елементи для відображення здоров’я;

— ворожий танк має Rigidbody2D, AIController.cs, NavMeshAgent, Health.cs, а також отримував команди від ChatGPTClient.cs.

Навігація супротивника реалізовувалась з використанням бібліотеки NavMeshPlus, яка була інтегрована до проєкту через пакетний менеджер Unity. З її допомогою було створено навігаційну сітку по 2D–сцені, що дозволило використовувати стандартний компонент NavMeshAgent у ворожому танку. Усі об’єкти, які мали слугувати перешкодами (в тому числі укриття), були позначені як «невпроникні» для навігації через NavMeshModifier.

Камера була прив’язана до гравця і автоматично слідкувала за його переміщенням. Це забезпечувалося компонентом Cinemachine Virtual Camera, що дозволило уникнути складного коду для ручного переміщення камери.

Вся логіка обміну даними з нейронною мережею була винесена в окремий компонент ChatGPTClient.cs, який прикріплювався до об’єкта ворога. Саме цей скрипт займався формуванням ситуаційних запитів, надсиланням їх через HTTP–запити на платформу OpenAI, а також обробкою відповідей і передачею вибраної тактики в AIController [23].

Проєкт було структуровано по папках для зручності: Scripts/, Prefabs/, Scenes/, Materials/, UI/ (рис. 3.3). Це дозволило підтримувати порядок у проєкті, полегшило навігацію між скриптами та ресурсами, а також зробило структуру максимально зрозумілою для подальших оновлень.

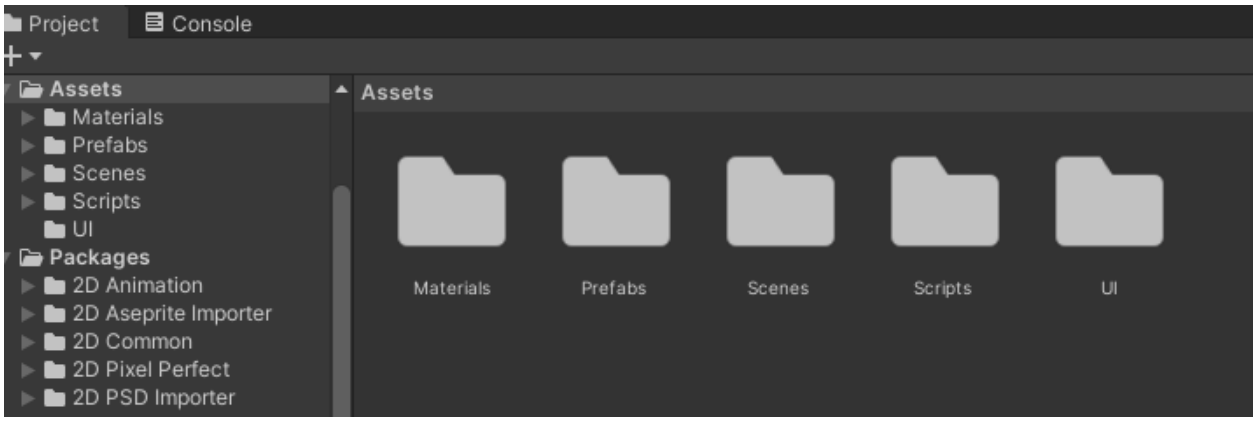

Рисунок 3.3 – Папки проекту

Загалом архітектура була реалізована таким чином, щоб усі елементи гри були максимально незалежними один від одного, згідно SOLID принципам. Це спростило налагодження, дозволило швидко змінювати окремі частини гри, не зачіпаючи інші, і забезпечило стабільність під час інтеграції з ChatGPT.

3.3 Реалізація управління гравцем

Керування танком гравця було реалізовано за допомогою скрипта PlayerMovement.cs, який відповідає за переміщення вперед і назад, а також за обертання корпусу вліво та вправо. Скрипт було прикріплено до об’єкта гравця, що містить компонент Rigidbody2D, необхідний для роботи фізики в 2D–просторі.

Основна ідея полягала в тому, щоб забезпечити гравцеві інтуїтивне керування, при якому натискання клавіш *W* та *S* відповідає за рух вперед і назад відповідно до поточного напрямку корпусу танка, а клавіші *A* та *D* – за обертання вліво і вправо. Для цього у Update(), який викликається кожен кадр, перевіряється натиснення клавіш і задається напрямок, а у FixedUpdate(), який викликається кожен кадр фізики, застосовується швидкість через Rigidbody2D.velocity.

Обертання корпусу реалізовано через метод Transform.Rotate, що дозволило плавно повертати об’єкт без використання фізичних моментів. Це забезпечує передбачувану поведінку танка незалежно від кількості кадрів. Варто окрема зазначити що мається на увазі під незалежністю від кількості кадрів. Через те, що перевірка чи натиснута кнопка відбувається у методі Update, вона буде викликатись стільки разів, скільки окремих моментів здатен порахувати комп’ютер. Щоб вирішити цю проблему, результат розрахунків помножують на *Time.deltaTime*, який зберігає в собі затрачений час на обробку кадру (рис. 3.4).

Рух реалізовано так, що танк завжди рухається по напрямку вектора transform.up, який у Unity відповідає за локальну «вісь вперед» об’єкта. Це дало змогу поєднати обертання та рух без складної математики.

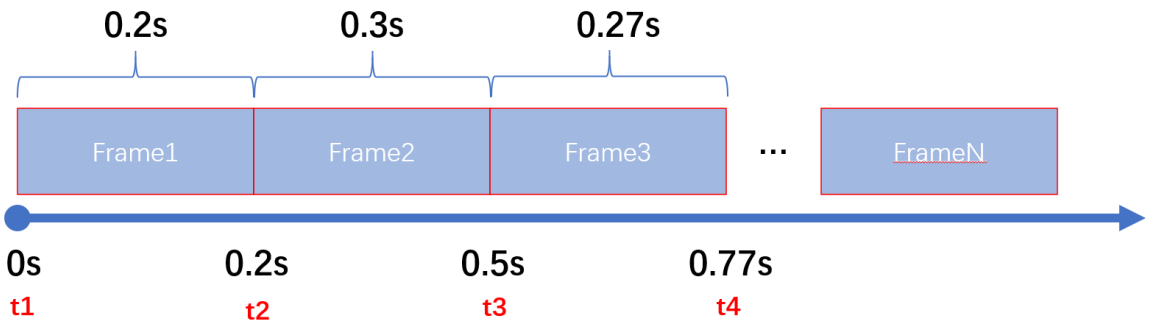

Рисунок 3.4 – Time.deltaTime

Поточне значення швидкості задається у вигляді публічного параметра movementSpeed, тому за потреби його можна легко змінювати в редакторі.

Також у скрипті передбачено систему перевірки компонентів. У методі Start() виконується спроба автоматично знайти Rigidbody2D, який відповідає за симуляцію фізики, і Transform, якщо вони не були вказані вручну. Це зроблено для підвищення гнучкості – навіть якщо скрипт буде перенесено на інший об’єкт, він зможе працювати без додаткових налаштувань.

Оскільки гра створюється в стилі вигляд зверху (top–down), було важливо забезпечити плавне, але чітке керування, без інерції чи коливань, які могли б ускладнити прицілювання. Саме тому всі рухи танка є прямолінійними, а обертання реалізовано на рівні трансформацій (компоненту transform), а не через фізичні сили, які б надавали танку інерцію, ускладнюючи його керування за допомогою вбудованої у рушій системи навігації.

В результаті гравець отримує чітке керування, що відповідає класичним канонам танкових аркад. Танком легко маневрувати між укриттями, обертатися в напрямку атаки чи відступу, і при цьому завжди мати контроль над ситуацією на полі бою.

3.4 Реалізація прицілювання та стрільби гравця

Для реалізації стрільби та прицілювання гравця був створений окремий скрипт PlayerAiming.cs, який відповідає за обертання башти танка в напрямку миші та за створення снарядів під час натискання кнопки вогню. Цей скрипт було прикріплено до об’єкта гравця разом із PlayerMovement.cs.

Обертання башти реалізовано через напрямок вектора, що йде від позиції башти до позиції миші у світових координатах. Для цього координати миші було перетворено з екранних у світові за допомогою методу Camera.main.ScreenToWorldPoint, якмй перетворює координати курсору миші на екрані в світові. Далі, в кожному кадрі в методі Update(), башта орієнтується в бік миші шляхом оновлення вектора вгору (up).

Ця реалізація дозволяє гравцеві завжди дивитися в напрямку курсора, незалежно від руху танка. Це дуже важливо в іграх, де потрібно стріляти в напрямку, відмінному від напрямку руху.

Стрільба реалізована через натискання лівої кнопки миші (Input.GetMouseButtonDown*(0)*) (рис. 3.5). При цьому створюється новий об’єкт кулі (*Bullet*), який ініціалізується в позиції дула (*muzzle.position*) і в напрямку дула (*muzzle.rotation*). Після створення на кулю накладається імпульс через Rigidbody2D.AddForce, що забезпечує її рух уперед із заданою швидкістю.

Крім того, було реалізовано систему перезарядки. У скрипті передбачено два параметри: *reloadTime* – загальний час перезарядки, та *remainReloadingTime* – час, що залишився до наступного пострілу. Якщо гравець натискає кнопку вогню раніше, ніж завершиться перезарядка, постріл не виконується. Це дозволяє запобігти безперервному спаму снарядами та дає змогу гнучко балансувати геймплей.

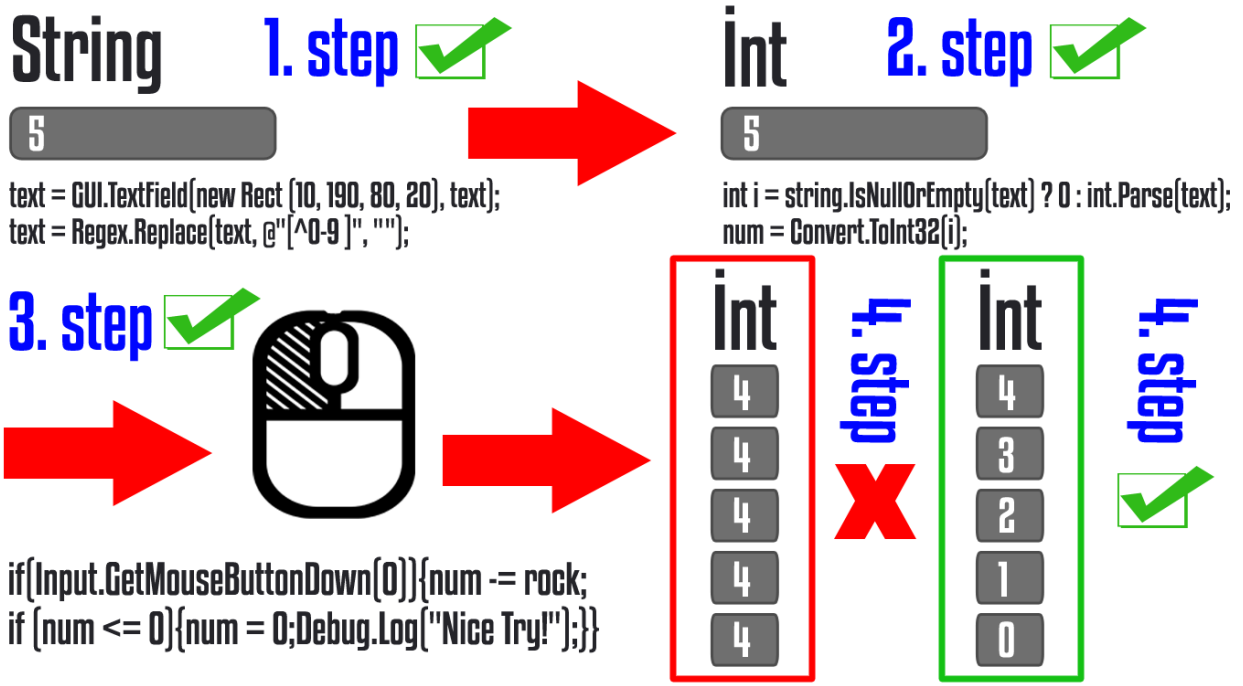

Рисунок 3.5 – Натискання кнопки миші в Unity

Усі параметри – швидкість кулі (BulletSpeed), шкода (BulletDamage), час перезарядки (reloadTime) – винесені у вигляді SerializeField, тому їх можна налаштовувати прямо у редакторі Unity без необхідності змінювати код (рис. 3.6).

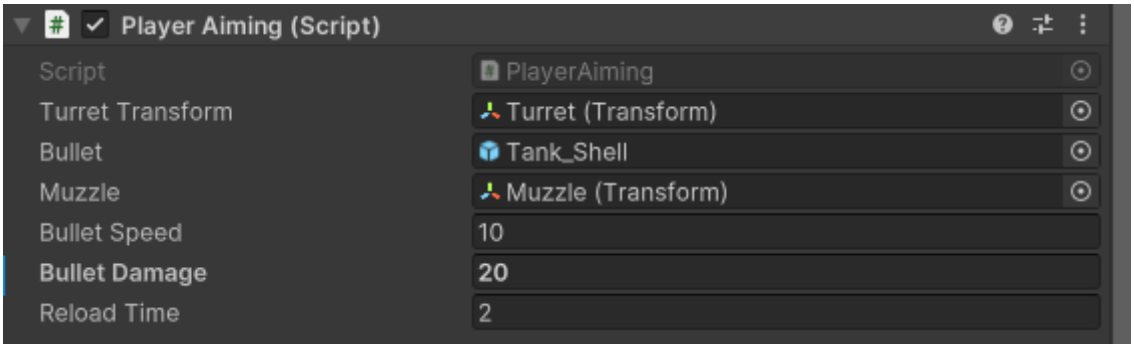

Рисунок 3.6 – Налаштування скрипту в редакторі

Для гнучкості реалізації було передбачено методи GetDamage(), GetReloadTime() та GetRemainingReload(), які дозволяють іншим скриптам (наприклад, ChatGPTClient.cs) отримувати актуальну бойову інформацію про

гравця. Це особливо важливо для формування коректного ситуаційного запиту до нейромережі.

Завдяки такій реалізації стрільба гравця є плавною, зручною і передбачуваною. Гравець може вести вогонь у будь–якому напрямку, не зупиняючи танк, а також чітко контролювати ритм стрільби завдяки перезарядці.

3.5 Створення логіки супротивника

Вся поведінка ворожого танка була реалізована в окремому скрипті AIController.cs, який відповідає за переміщення, обертання корпусу, прицілювання, стрільбу та виконання вибраної тактики [24]. Скрипт був прикріплений до об’єкта ворожого танка, який також має компонент Rigidbody2D і NavMeshAgent для навігації сценою.

Основною задачею цього скрипта було забезпечити динамічну реакцію ворожого танка на зміну ігрової ситуації. Для цього було реалізовано набір тактик, які описують різну бойову поведінку. Перехід між тактиками відбувається після обробки запиту до нейромережі, а конкретна реалізація обраної дії – через блок вибору [25].

У кожному кадрі скрипт перевіряє наявність гравця, обертає башту в його бік за допомогою методу RotateTurret(), та у випадку готовності до пострілу виконує метод Shoot(). Башта ворога орієнтується на гравця, аналогічно до гравця, але без використання миші – позиція гравця береться напряму через тег «Player».

Стрільба супротивника працює за тією ж логікою, що й у гравця: використовується параметр *reloadTime* та локальний таймер *remainReloadingTime*, який рахує час до наступного пострілу. Кожен постріл супроводжується створенням кулі на позиції дула (*muzzle*) та її прискоренням

у напрямку transform.up. Значення шкоди (*BulletDamage*) та швидкості (*BulletSpeed*) винесені в змінні для подальшого використання як у внутрішній логіці, так і для запитів у ChatGPTClient.

Ключовим елементом цього скрипта є метод ApplyTactic(string tactic), який отримує відповідь від нейромережі та визначає, яку тактику слід застосувати. Далі, в Update(), залежно від активної тактики, виконується відповідна дія: відступ, наступ, тримання позиції або флангова атака. Для кожної з тактик був написаний окремий приватний метод: RetreatToCover(), AdvanceWithCover(), FlankFromSide() та інші. Це зробило код читабельним, модульним і простим для подальшого розширення.

Рух ворожого танка реалізовано через компонент NavMeshAgent. Він працює разом із навігаційною сіткою, яка створюється за допомогою бібліотеки NavMeshPlus [26]. Ця інтеграція дозволила реалізувати логіку руху до укриття, атак з флангу, відступу з урахуванням перешкод тощо. Коли танк не рухається, NavMeshAgent зупиняється, а об’єкт стоїть на місці, що дозволяє, наприклад, реалізувати «стояти й стріляти» без додаткової логіки.

Якщо тактика, отримана з нейромережі, не збігається з жодною з передбачених, у скрипті передбачено блок *default*, який виконує резервну поведінку – зазвичай HoldPosition() – і надсилає повторне нагадування до нейромережі про правильний формат відповіді. Це дозволяє підтримувати стабільність гри навіть у випадку помилкової відповіді [27].

Крім того, у скрипті реалізовано методи GetDamage(), GetReloadTime() та GetRemainingReload(), які використовуються для формування ситуаційного запиту в ChatGPTClient. Це дозволяє нейромережі аналізувати бойову ситуацію з урахуванням справжніх параметрів супротивника [28].

Таким чином, AIController.cs поєднує класичну логіку керування NPC з можливістю динамічного керування поведінкою з боку нейронної мережі, що робить супротивника непередбачуваним і адаптивним у бою [29].

3.6 Взаємодія з ChatGPT

Для реалізації зв’язку гри з нейронною мережею був створений окремий скрипт ChatGPTClient.cs, який виконує роль посередника між грою та платформою OpenAI. Основне його завдання – формувати текстовий опис ситуації в грі, надсилати цей опис на сервер нейромережі через HTTP–запит і отримувати у відповідь одну з тактичних інструкцій, яка далі застосовується ворогом.

Спочатку скрипт підключає всі необхідні компоненти, зокрема AIController, Health, PlayerAiming, PlayerMovement та NavMeshAgent. Це потрібно для того, щоб мати доступ до всіх параметрів, які формують ситуацію на полі бою: здоров’я, шкода, дистанція, перезарядка, швидкість тощо. Усі ці значення використовуються при формуванні запиту.

У методі Start(), який викликається на початку гри, скрипт виконує пошук об’єктів гравця та супротивника за тегами «Player» і «Enemy», ініціалізує доступ до їхніх скриптів, а також відправляє перший вступний запит до нейромережі. Цей запит містить коротке пояснення, що відбувається у грі, список доступних тактик та інструкцію, як має виглядати відповідь. Це потрібно, щоб модель розуміла контекст і надсилала відповіді у правильному форматі.

Далі в скрипті запускається корутина SendSituationEveryXSeconds, яка кожні 7 секунд формує оновлений опис ситуації і надсилає його. Метод GenerateSituationPrompt() створює текст повідомлення у форматі: «Your HP: 150, Enemy HP: 200, Your Damage: 80, Enemy Damage: 100» і т.д. У нього входять всі критично важливі параметри, включно з відстанню до ворога, до укриття, часом до наступного пострілу, швидкістю пересування та іншими змінними (рис. 3.7).

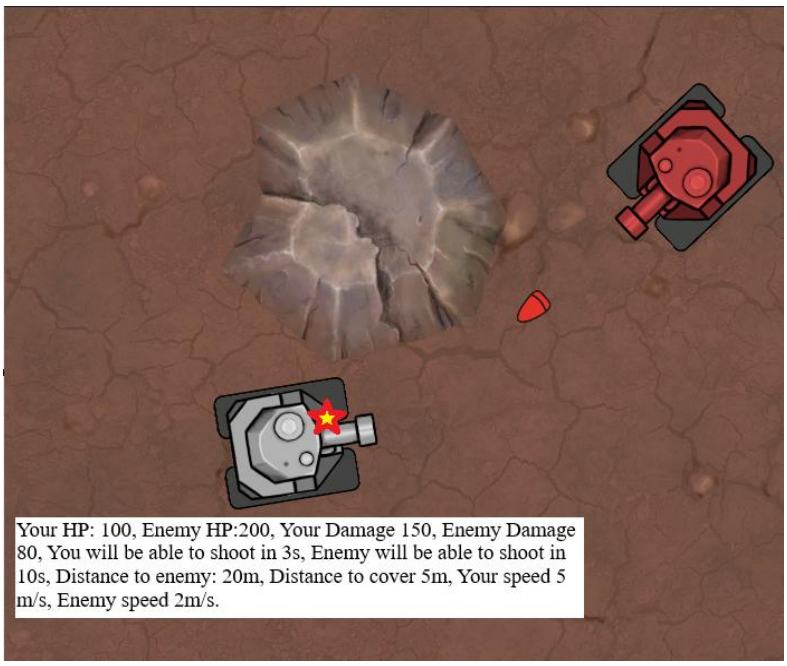

Рисунок 3.7 – Приклад надсилання запиту під час гри

Запит формується як JSON–структура, що містить модель нейромережі, список повідомлень і ключ API. Для надсилання використовується UnityWebRequest, де запит налаштовується вручну з усіма заголовками. Усі запити надсилаються методом POST, а відповіді обробляються у форматі JSON за допомогою бібліотеки Newtonsoft.Json.

Після отримання відповіді виконується метод ParseChatGPTResponse, у якому з тексту виділяється лише тактика в лапках – наприклад «Attack while keeping distance». Якщо виявлено, що відповідь некоректна (наприклад, не містить жодної з тактик або містить зайвий текст), скрипт автоматично надсилає нагадування, що відповідь має бути у форматі з одного рядка, взятого в лапки, і з використанням лише доступних варіантів.

Далі обрана тактика передається у AIController.cs, де вона застосовується через метод ApplyTactic (рис. 3.8).

```
public void ApplyTactic(string tactic)
{
    switch (tactic)
    {
        case "run away":
            RetreatWithoutStrategy();
            break;
        case "retreat backward without cover":
            RetreatStraightBack();
            break;
        case "retreat using cover":
            RetreatToCover();
            break;
        case "hold position defensively":
            HoldPosition();
            break;
        case "stand still and fire back":
            HoldPosition();
            break;
        case "advance slowly using cover":
            AdvanceWithCover();
            break;
        case "attack while keeping distance":
            AttackFromDistance();
            break;
        case "attack at close range":
            CloseRangeAttack();
            break;
        case "move in and attack from the side":
            FlankFromSide();
            break;
        case "move in and attack from behind":
            FlankFromBehind();
            break;
        default:
            Debug.LogWarning("Unknown tactic: " + tactic);
            HoldPosition();

            // Надсилання повідомлення з нагадуванням
            ChatGPTClient client = FindObjectOfType<ChatGPTClient>();
            if (client != null)
            {
                client.ResendReminder();
            }

            break;
    }
}
```

Рисунок 3.8 – Набір прописаних тактик в методі ApplyTactic

Крім періодичних оновлень, у скрипті реалізовано окремий метод SendImmediateSituation, який викликається вручну у випадках, коли потрібно миттєво оновити тактику – наприклад, після потрапляння по ворогу.

Також у скрипті передбачено систему логування через Debug.Log, яка дозволяє бачити всі надіслані запити, отримані відповіді та помилки, якщо вони виникають (наприклад, якщо відсутній інтернет або неправильний ключ).

В результаті ChatGPTClient.cs забезпечує надійний канал зв’язку між грою та штучним інтелектом, дозволяючи супротивнику адаптувати свою поведінку в реальному часі, враховуючи всі зміни на полі бою.

3.7 Система здоров’я та анімація індикатора

Щоб гравець і супротивник мали можливість отримувати шкоду, а також щоб ця шкода відображалася в інтерфейсі, була реалізована система здоров’я, яка включає в себе як логічну, так і візуальну частину. За це відповідає скрипт Health.cs, який прикріплюється до об’єкта гравця та до об’єкта супротивника окремо.

Основна логіка системи дуже проста: кожен об’єкт має певну максимальну кількість здоров’я (*maxHealth*), яка зменшується при отриманні урону через виклик методу Damage. Якщо здоров’я знижується до нуля або нижче, об’єкт знищується. Це зроблено через вбудований метод Destroy, що дозволяє не тільки прибрати візуальну частину, а й очистити сцену від компонентів, що більше не потрібні.

Крім зберігання числового значення здоров’я (*currentHealth*), скрипт також оновлює індикатори здоров’я в інтерфейсі. Для цього використовується два Image–елементи з типом Filled – один активний (який показує актуальний рівень HP), а інший – фонова анімаційна панель, яка плавно «доганяє» основну. Завдяки цьому гравець не просто бачить зменшення, а відчуває, коли і в якому розмірі був нанесений урон, навіть якщо не дивився прямо на танк (рис. 3.9).

Анімація реалізована через корутину AnimateHealthBar(), яка поступово зменшує значення *fillAmount* заднього шару до нового рівня. Вона працює за принципом Lerp – поступового наближення до цілі з урахуванням швидкості, що налаштовується окремим параметром *HpAnimSpeed*. Під час урону, якщо

анімація вже триває, попередня корутина переривається, щоб не виникало візуальних «конфліктів».

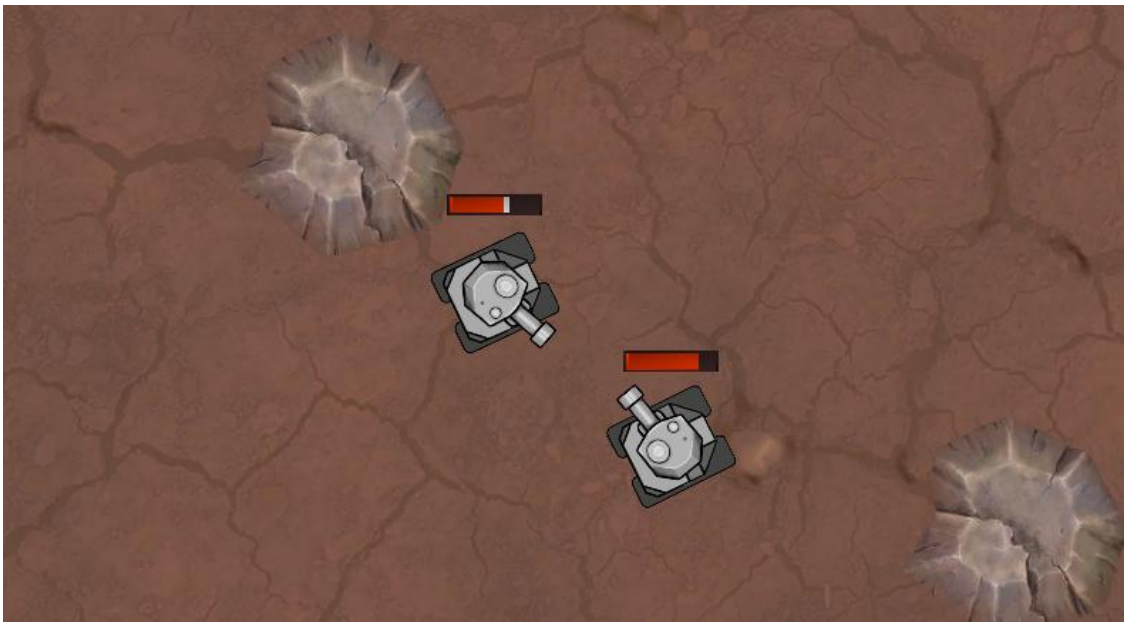

Рисунок 3.9 – Анімація полоси здоров’я

Окремо варто згадати інтеграцію з ChatGPTClient.cs. Якщо пошкодження отримує супротивник (об’єкт з тегом «Enemy»), скрипт викликає метод SendImmediateSituation() у клієнті штучного інтелекту. Це дозволяє оперативно інформувати нейромережу про зміну ситуації на полі бою. Наприклад, якщо у ворога залишилося лише 10% HP, нейромережа може вирішити відступити, а не атакувати.

Щоб надати доступ до значень здоров’я ззовні (наприклад, для ситуаційного запиту до нейромережі), скрипт реалізує метод GetHP(), який повертає поточне значення.

Усі параметри – максимальне здоров’я, швидкість анімації, посилання на індикатори – зроблені публічними або SerializeField, тому ними легко керувати в редакторі Unity без необхідності змінювати код [30].

Таким чином, Health.cs не лише відповідає за базову логіку бою (отримання шкоди, знищення об’єктів), а й забезпечує плавне та приємне

візуальне сприйняття змін HP, а також тісно взаємодіє з нейромережею в реальному часі.

3.8 Пошук укриттів та взаємодія з ними

Одним з важливих елементів геймплею є укриття, які можуть впливати на рішення, що приймає супротивник. Саме наявність укриттів відкриває можливості для тактик на кшталт «відступ з використанням укриття» або «повільне просування з прикриттям». Щоб реалізувати це у грі, була створена система динамічного пошуку укриттів, яка працює без ручного прив’язування координат або об’єктів.

Всі укриття на сцені мають тег «Cover». Це дозволяє швидко знаходити їх з будь–якого скрипта за допомогою методу GameObject.FindGameObjectsWithTag. Такий підхід зручний, оскільки дозволяє додавати або прибирати укриття без жодних змін у коді. Після розміщення укриття у сцені та встановлення відповідного тегу – воно автоматично потрапляє до списку доступних для аналізу об’єктів.

Пошук найближчого укриття реалізовано в методі FindNearestCover(), який перебирає всі об’єкти з тегом «Cover» і знаходить той, який має найменшу відстань до ворожого танка (рис. 3.10). Для розрахунку використовується Vector2.Distance(...), що дозволяє враховувати реальну дистанцію в просторі, а не просто координати по осі.

Якщо жодне укриття не знайдено, метод повертає null, і супротивник переходить до резервного варіанту тактики – наприклад, відступу без укриття. Це дозволяє уникати помилок у логіці, якщо з якоїсь причини на сцені немає об’єктів з тегом «Cover».

У методах тактичних дій (RetreatToCover(), AdvanceWithCover() тощо) координати укриття використовуються як ціль для NavMeshAgent. Агент пересувається до укриття автоматично, з урахуванням навігаційної сітки.

Якщо потрібно зайняти позицію не просто біля укриття, а за ним, то до координат укриття додається зсув у напрямку, протилежному до ворога. Таким чином супротивник ніби «ховається» з протилежного боку об’єкта.

Ця логіка дозволяє робити поведінку супротивника більш схожою на людську. Він не просто тупо біжить вперед, а шукає захист, підбирає позицію й намагається уникнути шкоди, якщо розуміє, що не витримає наступної атаки.

Окремо важливо зазначити, що дистанція до найближчого укриття також враховується під час формування ситуаційного запиту до нейромережі. Вона є частиною тексту, який надсилається в ChatGPT, і на основі неї мережа може прийняти рішення, наприклад, не атакувати, а зайняти укриття.

У результаті така реалізація укриттів – проста в технічному плані, але надзвичайно корисна для загального геймплею. Вона працює динамічно, не вимагає прив’язки до координат, і може масштабуватися при додаванні нових сцен або рівнів.

Рисунок 3.10 – Зайняття укриття

3.9 Формування запиту до нейромережі

Один із ключових елементів гри – це передача актуального стану бою нейронній мережі для отримання логічного рішення щодо подальших дій супротивника. Щоб це було можливо, у грі реалізовано систему динамічного формування ситуаційного текстового запиту, який описує все, що відбувається у грі з точки зору супротивника.

Формування запиту реалізовано у скрипті ChatGPTClient.cs у методі GenerateSituationPrompt(). Саме він збирає всі дані про поточний стан ворога, гравця, їхню зброю, здоров'я та положення, і формує з них зрозумілий для нейромережі опис. Результат – це один абзац англійською мовою, у якому всі числові значення представлені в зручному форматі.

У запиті слово «You» завжди означає супротивника (керованого нейромережею), а «Enemy» – гравця. Це зроблено для того, щоб мережа могла чітко розуміти, про кого йде мова, не плутаючи ролі. Це важливо, бо ChatGPT аналізує текст, як людина, і для нього важливо мати чіткі орієнтири.

Дані для запиту беруться безпосередньо з компонентів AIController.cs (для ворога) та PlayerAiming.cs, PlayerMovement.cs, Health.cs (для гравця). Зокрема:

- здоров'я – через метод GetHP() для обох сторін;
- шкода від снарядів – через GetDamage();
- час до наступного пострілу – через GetRemainingReload();
- швидкість руху – через NavMeshAgent.speed для ворога і PlayerMovement.GetSpeed() для гравця;
- відстань між танками – через Vector2.Distance(...);
- відстань до найближчого укриття – через окремий метод FindNearestCoverDistance().

Після складання тексту він надсилається до нейромережі через HTTP–запит і використовується як вхідне повідомлення моделі.

Окремо варто зазначити, що до запиту завжди додається фраза–нагадування: Write ONLY tactic from the list, як показано у лістингу 3.1.

Лістинг 3.1 Формування запиту для нейрожережі:

```
string prompt =
        $"Your HP: {yourHP}, " +
        $"Enemy HP: {enemyHP}, " +
        $"Your Damage: {yourDamage}, " +
        $"Enemy Damage: {enemyDamage}, " +
        $"You will be able to shoot in {yourReload:F1}s, " +
        $"Enemy will be able to shoot in {enemyReload:F1}s, " +
        $"Distance to enemy: {distToEnemy:F1}m, " +
        $"Distance to cover: {distToCover:F1}m, " +
        $"Your speed: {yourSpeed} m/s, " +
        $"Enemy speed: {enemySpeed} m/s."+
        "Write ONLY tactic from the list";
```

Це потрібно, щоб ChatGPT не відповідав зайвими поясненнями, а видавав чітко одну тактику, взяту в лапки – наприклад: «Retreat using cover». Якщо відповідь не відповідає формату, або не збігається з жодною з відомих тактик, то гра автоматично надсилає нагадування з точним списком дозволених відповідей і просить повторити.

Запит формується кожні 7 секунд, але також може надсилатися в екстрених ситуаціях – наприклад, коли ворог отримує урон. У таких випадках викликається метод SendImmediateSituation(), який форсує запит незалежно від таймера.

Цей підхід дозволяє нейромережі мати повну картину ситуації на полі бою та приймати рішення не наосліп, а з урахуванням реального стану танка, характеристик зброї, положення гравця, наявності укриття, часу до наступного пострілу і так далі.

Таким чином, система формування запиту виступає як міст між грою та логікою, яку створює нейромережа. Вона перетворює ігрову ситуацію на простий і зрозумілий текст, який ChatGPT може обробити та на основі якого видати логічну тактику.

3.10 Реалізація тактик поведінки супротивника

Щоб зробити поведінку супротивника в грі максимально адаптивною, у скрипті AIController.cs було реалізовано систему тактик. Усього передбачено одинадцять варіантів поведінки, кожен із яких відповідає певному бойовому стилю: відступ, оборона, наступ або прорив. Реалізація побудована так, що супротивник не має фіксованої моделі дій, а обирає поведінку динамічно, отримуючи її з нейронної мережі у вигляді текстового рядка. Цей рядок потім обробляється методом ApplyTactic(string tactic), у якому через конструкцію switch викликається відповідна функція поведінки.

Наприклад, коли супротивник отримує інструкцію «Run away», він визначає напрямок, протилежний до позиції гравця, і рухається у зворотному напрямку на певну відстань. Якщо ж отримано команду «Retreat backward without cover», то відступ відбувається строго назад по локальній осі танка, незалежно від положення гравця. Це дозволяє йому зберігати напрямок башти навіть під час відступу. У разі, якщо нейромережа вибирає тактику «Retreat using cover», супротивник спочатку знаходить найближче укриття, після чого розраховує точку за укриттям з урахуванням положення гравця, і прямує саме туди. Для цього використовується пошук об’єктів із тегом «Cover» та розрахунок відстані до них. Якщо жодного укриття на сцені немає, ворог виконує простий відступ без покриття.

Оборонна поведінка також має кілька варіантів. У найпростішому випадку супротивник просто зупиняється на місці та тримає приціл на гравця. Якщо ж тактика передбачає стрільбу, то у випадку готовності

(*remainReloadingTime* <= 0) виконується метод Shoot(), який створює кулю на дулі та надає їй імпульс у напрямку башти.

Для агресивної поведінки було реалізовано декілька варіантів. Якщо обрано «Advance slowly using cover», супротивник знаходить укриття між собою та гравцем і рухається до нього, просуваючись невеликими кроками. Якщо укриття немає, він переходить у режим ближньої атаки. У разі тактики «Attack while keeping distance», танк намагається залишатися на відстані не менше ніж 10 метрів від гравця, орієнтуючись у просторі так, щоб підтримувати дистанцію і водночас залишатися у зоні вогню. А при виборі «Attack at close range» ворог прямує прямо до гравця, наближається максимально близько та веде стрільбу з мінімальної дистанції.

У тактиках прориву логіка ще агресивніша. Якщо обрано «Move in and attack from the side», розраховується перпендикуляр до вектора напрямку від супротивника до гравця, і ворог намагається обійти з боку. При «Move in and attack from behind» розраховується напрямок позаду гравця, і супротивник рухається до точки, що дозволяє атакувати з тилу. Обидві ці тактики дозволяють зробити ворога більш розумним, адже він не просто стріляє, а намагається зайняти вигідну позицію.

Кожна з реалізованих тактик винесена в окремий метод, що дозволяє зберегти читабельність коду та спростити модифікацію (рис. 3.11).

Танк сам знаходить шлях, оминає перешкоди та коригує кут повороту згідно з напрямком руху. Обертання корпусу виконується через аналіз вектора швидкості, що дозволяє танку візуально правильно повертатися під час руху. Для цього береться положення танку у нинішньому та попередньому кадрах та знаходиться їх різниця.

```
1 reference
private void RetreatStraightBack()
{
    Debug.Log("Tactic: Retreat backward without cover");
    Vector2 retreatDir = -transform.up;
    agent.SetDestination((Vector2)transform.position + retreatDir * 5f);
}

1 reference
private void RetreatToCover()
{
    Debug.Log("Tactic: Retreat using cover");
    Transform cover = FindNearestCover();
    if (cover)
    {
        Vector2 dirAway = (transform.position - player.position).normalized;
        Vector2 safePos = (Vector2)cover.position + dirAway * 1.5f;
        agent.SetDestination(safePos);
    }
    else
    {
        RetreatWithoutStrategy();
    }
}
```

Рисунок 3.11 – Окремі методи для кожної з тактик

Усі переміщення реалізовано через NavMeshAgent, який оновлює точку призначення відповідно до обраної тактики, використовуючи сгенеровану навігаційну мапу (рис. 3.12).

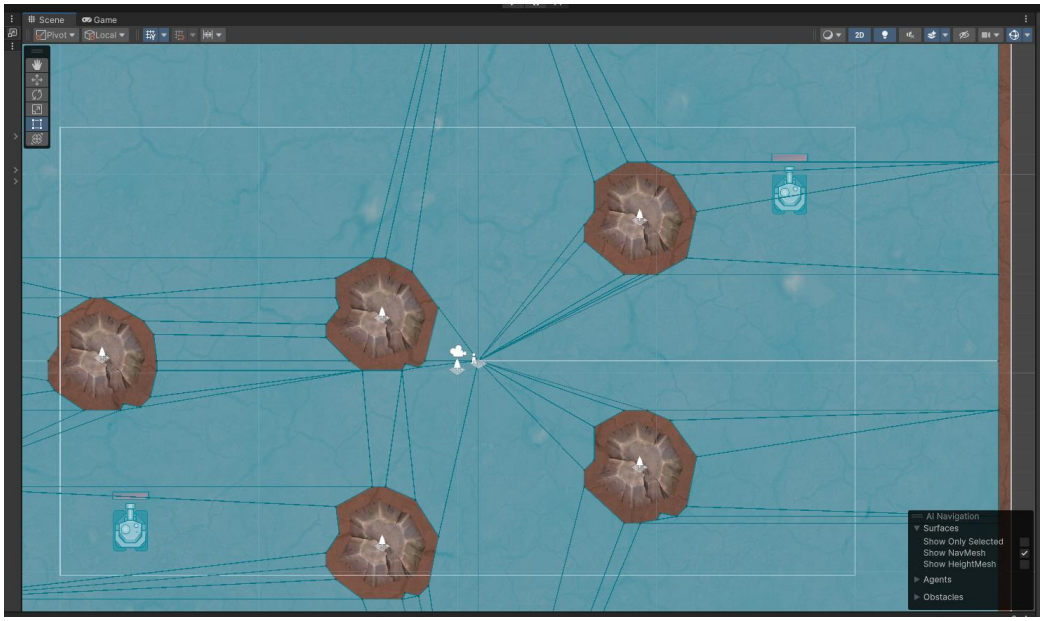

Рисунок 3.12 – Сгенерована навігаційна мапа на рівні

Завдяки реалізованій системі, ворог у грі не поводиться передбачувано і не діє за шаблоном. Кожна його дія – результат рішення, прийнятого на основі

повної ситуації на полі бою. А самі дії – це чітко реалізовані поведінкові патерни, які дозволяють поєднати штучний інтелект з реальними ігровими механіками.

3.11 Тестування

Після завершення реалізації всіх компонентів гри було проведено тестування з метою перевірки коректності роботи ключових механік, а також взаємодії між грою та нейромережею. Основна увага приділялась перевірці поведінки супротивника в різних бойових ситуаціях, правильності формування запитів, стабільності обміну з інтерфейсом прикладного програмування, а також загальній якості геймплею.

Тестування проводилось на локальній машині у середовищі редактора Unity. На сцені було розміщено базові об’єкти – гравець, супротивник, кілька укриттів (рис. 3.13) – і перевірялась робота кожної з тактик, що надсилаються з нейронної мережі (рис. 3.14). Кожна тактика активувалась через відповідь від ChatGPT, після чого вручну перевірялась відповідність дій ворога обраній поведінці.

Рисунок 3.13 – Розставлена сцена з укриттями

Було протестовано всі варіанти тактичних сценаріїв: відступ, оборона, агресивний наступ, прорив з флангу.

Рисунок 3.14 – Вивід обраних тактик у консоль

Супротивник правильно реагував на зміну ситуації: обирав укриття при низькому здоров’ї, тримав позицію в обороні, а також атакував з відстані або в ближньому бою, залежно від умов (рис. 3.15).

Рисунок 3.15 – Вібир тактики відступу при низькому здоров’ї

Як можна побачити на рисунку 3.16, спочатку нейромережа обрала тримати оборону, а коли здоров’є опустилось до низького рівня, вирішила відступити за укриття.

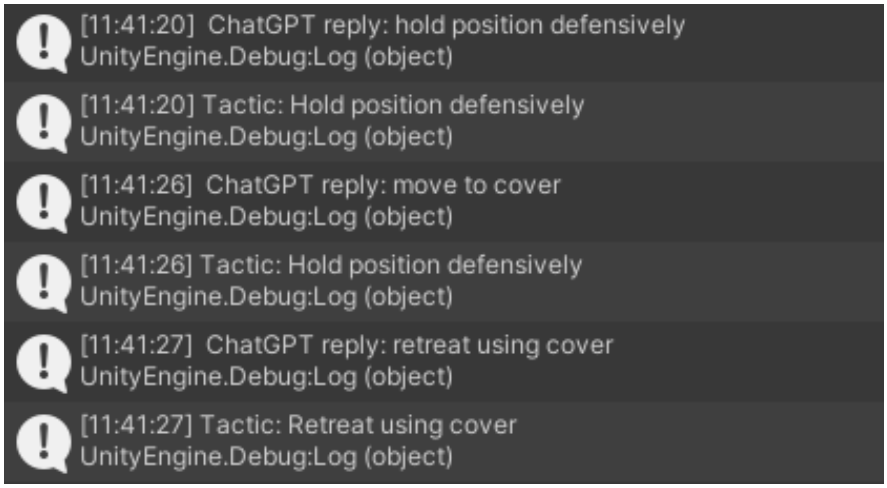

Рисунок 3.16 – Консоль виводу тактик

У випадках, коли нейромережа повертала некоректну або непередбачену відповідь, система відпрацьовувала резервний сценарій та відправляла повторний запит із нагадуванням про правильний формат відповіді (рис. 3.17).

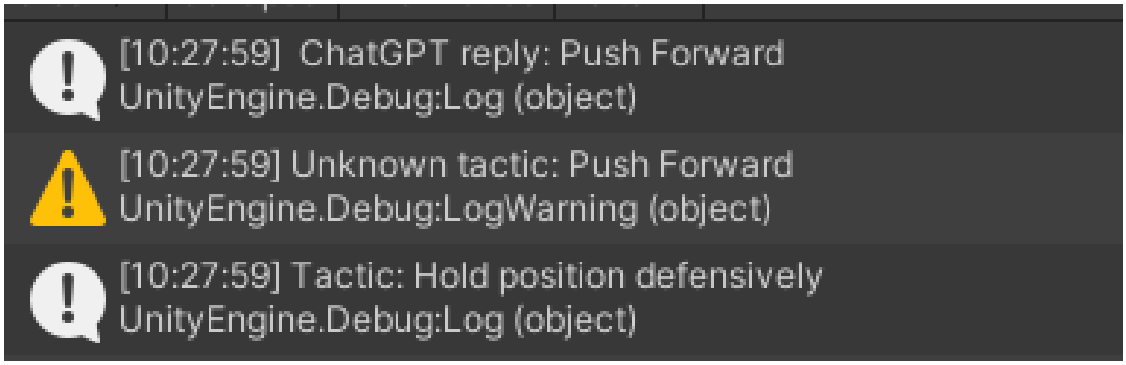

Рисунок 3.17 – Вибір резервної тактики при неправильній відповіді

Було порівняно близько 50 різних ситуацій з різними накращими тактиками та на основі цього був створений графік з відсотком правильних відповідей від нейромережі (LLM) та звичайним ШІ (AI), який працює як вибір випадкової тактики в залежності від пари параметрів (рис. 3.18).

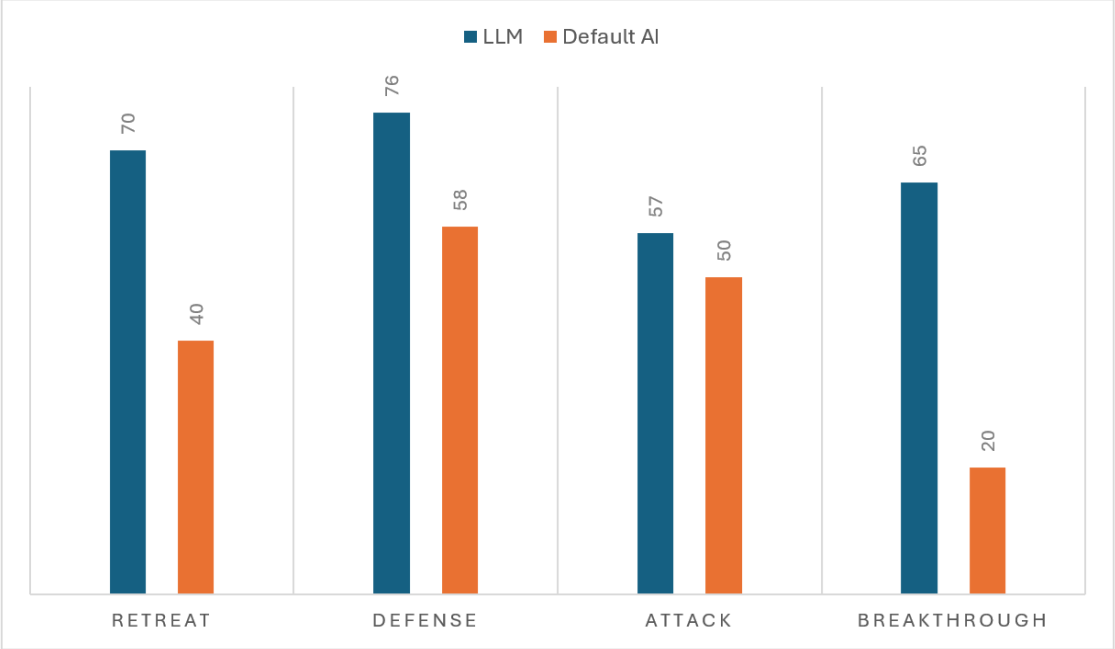

Рисунок 3.18 – Порівняння відсотку правильних відповідей

Результати порівняння, зображені на графіку, чітко показують, що нейромережа набагато краще справляється з вибором правильної тактики, ніж звичайний алгоритм. Наприклад, у ситуаціях, де потрібно відступити, модель правильно обрала тактику у 70% випадків, тоді як звичайний ШІ - тільки у 40%. Це можливо через те, що модель отримує повну інформацію про стан здоров’я, перезарядку та позиції, тому може зважити ризики.

У сценарії «Defense» різниця ще більша: нейромережа дала правильну відповідь у 76% випадків, тоді як звичайний алгоритм - лише в 58%.

У сценарії «Attack» розрив між результатами вже менший: 57% проти 50%. Але це не означає, що модель гірше працює - просто в цьому типі ситуацій навіть випадковий вибір іноді буває прийнятним, бо всі три варіанти передбачають якусь форму атаки.

Найцікавіше - це «Breakthrough». Тут модель показала 65% правильних відповідей, а випадковий ШІ - лише 20%. Тобто в складних ситуаціях, коли потрібно обійти гравця або зайти з флангу, модель поводиться набагато розумніше, тоді як звичайний алгоритм майже ніколи не вибирає правильний варіант.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований проект, який демонструє новий підхід до створення ігрового штучного інтелекту на основі нейронної мережі. Класичні методи побудови поведінки супротивників у відеоіграх, як показує практика, хоч і залишаються ефективними для базових сценаріїв, не забезпечують достатньої гнучкості, що обмежує динамічність ігрового процесу та знижує рівень занурення гравця у віртуальний світ.

Розроблена гра, яка була створена на рушії Unity з використанням мови програмування C#, дозволила на практиці реалізувати інтеграцію з великою мовною моделлю ChatGPT. Усі бойові рішення супротивника приймаються нейронною мережею, яка на основі даних про поточну ситуацію – таких як здоров’я, відстань до гравця, доступність укриттів і стан перезарядки – обирає найбільш логічну та доцільну тактику. Це зробило поведінку NPC значно менш передбачуваною, варіативною і ближчою до дій, які можна очікувати від живого супротивника.

У процесі реалізації було створено функціональну систему, яка дозволяє не лише реагувати на ігрові події, а й оновлювати бойову стратегію в реальному часі. Завдяки періодичним запитам, що формуються в текстовому вигляді та надсилаються через інтерфейс прикладного програмування до нейромережі, гра адаптується під ситуацію на полі бою. При цьому кожна відповідь обробляється, перевіряється на коректність і безпосередньо впливає на поведінку супротивника.

Таким чином, робота довела, що використання великих мовних моделей у геймдизайні відкриває нові перспективи. Впровадження ChatGPT у систему управління NPC дозволяє створювати дійсно адаптивну поведінку. Результати роботи апробовано у вигляді 3 тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [31].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Yannakakis, G. N., & Togelius, J. (2018). *Artificial intelligence and games* (Vol. 2, pp. 2475–1502). New York: Springer.

2. Champandard, A. J. (2003). *AI game development: Synthetic creatures with learning and reactive behaviors*. New Riders.

3. Zhu, X. (2019). Behavior tree design of intelligent behavior of non–player character (NPC) based on Unity3D. *Journal of Intelligent & Fuzzy Systems*, *37*(5), 6071–6079.

4. Buckland, M. (2005). *Programming game AI by example*. Jones & Bartlett Learning.

5. Miyake, Y. (2015). Current status of applying artificial intelligence in digital games. Handbook of Digital Games and Entertainment Technologies, 253–292.

6. Aversa, D. (2022). *Unity Artificial Intelligence Programming: Add powerful, believable, and fun AI entities in your game with the power of Unity*. Packt Publishing Ltd.

7. Tverdokhlib, A. O. (2024). The role of artificial intelligence in information retrieval systems optimization.

8. Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image classification methods. *Advanced Information Systems*, *9*(1), 5–12.

9. Felicia, P. (2017). *Unity from Proficiency to Mastery (Artificial Intelligence): Implement Challenging AI for FPS and RPG Games* (Vol. 1). Patrick Felicia.

10. Gupta, P. (2024). *AI in gaming: creating adaptive and intelligent game environments*.

11. Lyubchenko, V., Veretelnyk, K., Kots, P., & Lyashenko, V. (2024). Digital image segmentation procedure as an example of an NP–problem. *Multidisciplinary Journal of Science and Technology*, *4*(4), 170–177.

12. Kuzomin, O., & Lyashenko, V. (2022). Agent–Based Model as a Research Tool.

13. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016, September). Vizdoom: A doom–based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)* (pp. 1–8). IEEE.

14. Kuzomin, O., Dudka, O., Vasylenko, O., Radchenko, V., & Lyashenko, V. (2020). Using of ontologies for building databases and knowledge bases for consequences management of emergency.

15. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., & Al–Dhaifallah, M. (2023). Statistical data analysis models for determining the relevance of structural image descriptions. *IEEE Access*, *11*, 126938–126949.

16. Олексенко, В. В., & Білаш, Д. А. (2023). Ігровий програмний застосунок з навчання основам теорії ігор. РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ, 305.

17. Kyaw, A. S., Peters, C., & Swe, T. N. (2013). *Unity 4. x Game AI Programming*. Packt Publishing.

18. Upadhyay, U., Shah, N., Ravikanti, S., & Medhe, M. (2019). Transformer based reinforcement learning for games. *arXiv preprint arXiv:1912.03918*.

19. Tvoroshenko, Irina, et al. «Modification of models intensive development ontologies by fuzzy logic.» (2020).

20. Smed, J., & Hakonen, H. (2017). Algorithms and networking for computer games. John Wiley & Sons.

21. Tvoroshenko, I., & Gorokhovatskyi, V. (2022). The Application of Hybrid Intelligence Systems for Dynamic Data Analysis.

22. Lyubchenko, V., Matarneh, R., Kobylin, O., & Lyashenko, V. (2016). Digital image processing techniques for detection and diagnosis of fish diseases.

23. Igras–Cybulska, M., Kolber–Bugajska, B., Salamon, R., Babiuch, P., Jegierski, H., Jegierski, M., ... & Węgrzyn, P. (2024). Supporting Unity developers

with an AI–powered asset: Insights from an exploratory user study on multiplayer game development.

24. Dablander, M. (2024). Future Research Avenues for Artificial Intelligence in Digital Gaming: An Exploratory Report. *arXiv preprint arXiv:2412.14085*.

25. Palacios, J. (2018). *Unity 2018 Artificial Intelligence Cookbook: Over 90 recipes to build and customize AI entities for your games with Unity*. Packt Publishing Ltd.

26. Håkansson, C., & Fröberg, J. (2021). Application of machine learning to construct advanced NPC behaviors in Unity 3D.

27. Lyashenko, V., Kobylin, O., & Minenko, M. (2018, October). Tools for investigating the phishing attacks dynamics. In *2018 International Scientific–Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)* (pp. 43–46). IEEE.

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

29. Arslan Akram, R. T., Saqib, S., Nazir, F., & Awan, M. M. Advanced AI Mechanics in Unity 3D for Immersive Gameplay. A Study on Finite State Machines & Artificial Intelligence.

30. Sindhu, R. M., Annabel, L. S. P., & Monisha, G. (2022, April). Development of a 2d game using artificial intelligence in unity. In 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 1031–1037). IEEE.

31. Куртенок М.Д. (2025) Дослідження застосування нейромереж для аналізу ігрового середовища та управління персонажем у тактичній грі. *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. 73–74.